POLITECHNIKA WARSZAWSKA

DYSCYPLINA NAUKOWA – INFORMATYKA TECHNICZNA
I TELEKOMUNIKACJA/
DZIEDZINA NAUK – NAUKI INŻYNIERYJNO-TECHNICZNE

Rozprawa doktorska

mgr inż. Marcin Gregorczyk

Badanie przydatności technologii Software-Defined Networking do wykrywania i przeciwdziałania zagrożeniom w sieciach teleinformatycznych

Promotor
dr hab. inż. Wojciech Mazurczyk, prof. uczelni

WARSZAWA 2022

Składam serdeczne podziękowania mojemu promotorowi,
Panu dr hab. inż. Wojciechowi Mazurczykowi, prof. PW,
za poświęcony dla mnie czas, o każdej porze dnia i nocy,
za wszystkie rady, za cierpliwość
i pomoc w rozwiązywaniu wielu problemów (nie tylko naukowych).

Pracę tę dedykuję mojej rodzinie, żonie Magdalenie,
synowi Krzysztofowi oraz córkom Małgorzacie i Joannie.

Badanie przydatności technologii Software-Defined Networking do wykrywania i przeciwdziałania zagrożeniom w sieciach teleinformatycznych

Streszczenie.

Niniejsza rozprawa doktorska wpisuje się w gałąź informatyki związaną z badaniami nad cyberbezpieczeństwem. Poruszana w niej problematyka badawcza dotyczy wykrywania i przeciwdziałania zagrożeniom w sieciach teleinformatycznych za pomocą technologii SDN. Na pracę składa się cykl czterech artykułów opublikowanych w renomowanych czasopismach i materiałach konferencyjnych. W ramach załączonych publikacji przedstawiono nowatorskie systemy detekcji i mitygacji wybranych zagrożeń w sieciach teleinformatycznych pokrywających wszystkie warstwy stosu TCP/IP. Przedstawiono także nowatorski atak na samą architekturę technologii SDN oraz potencjalne sposoby ochrony przed nim.

Wszystkie przeprowadzone badania oraz wyciągnięte na ich podstawie wnioski są, w odróżnieniu od większości pozycji znanych z literatury, rezultatem eksperymentów przeprowadzonych z wykorzystaniem rzeczywistych i używanych obecnie w świecie IT technologii, co podnosi ich wiarygodność i przydatność w praktyce.

Słowa kluczowe: Software-Defined Networking, bezpieczeństwo, zagrożenia sieciowe, mechanizmy zabezpieczeń

Usability study of Software-Defined Networking technology for detecting and mitigating threats in communication networks

Abstract. This dissertation constitutes the branch of computer science research dedicated to cybersecurity. It deals with the detection and mitigation of threats in communication networks using SDN technology. This PhD thesis consists of series of four articles published in international journals or conference proceeedings. These publications introduce novel detection and mitigation systems of selected threats in communication networks covering all layers of TCP/IP stack. Additionally, they also describe a novel attack on the architecture of SDN technology and potential ways to protect against it.

All conducted research and conclusions drawn, in contrast to most of existing research, were obtained via experiments conducted using real-world technologies currently used in the IT world, which increases their credibility and usefulness in practice.

Keywords: Software-Defined Networking, security, network threats, security mechanisms

Spis treści

Dorobek autora rozprawy

Pełna lista publikacji oraz cytowania na dzień 24.03.2022 (pogrubiono publikacje będące częścią rozprawy):

1. **Gregorczyk Marcin, Mazurczyk Wojciech: Inferring Flow Table State through Active Fingerprinting in SDN Environments: A Practical Approach, W: Proceedings of the 18th International Conference on Security and Cryptography / De Capitani di Vimercati Sabrina, Samarati Pierangela (red.), 2021, SCITEPRESS – Science and Technology Publications, Lda, ISBN 978-989-758-524-1, DOI: 10.5220/0010573905760586, s. 576 - 586, Punktacja MEiN = 70.**
2. **Gregorczyk Marcin, Żórawski Piotr, Nowakowski Piotr, Cabaj Krzysztof, Mazurczyk Wojciech: Sniffing Detection Based on Network Traffic Probing and Machine Learning, IEEE Access, vol. 8, 2020, DOI:10.1109/ACCESS.2020.3016076, Punktacja MEiN = 100, Cytowania GS = 3, Cytowania Scopus = 2, Scopus SNIP: 2018 = 1,718, Impact Factor WoS: 2019 (2-letni) = 3,745 - 2019 (5-letni) =4,076.**
3. Cosmas John, Jawad Nawar, Ali Kareem, Meunier Ben, Zhang Yue, Li W., Gregorczyk Marcin, Mazurczyk Wojciech, Cabaj Krzysztof, Lacaud Mathias: Network and Application Layer Services for High Performance Communications in Buildings, W: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB) Proceedings , 2020, Institute of Electrical and Electronics Engineers, ISBN 978-1-7281-5784-9, s. 1-8, DOI:10.1109/BMSB49480.2020.9379769, łączna liczba autorów: 16, Punktacja MEiN = 20.
4. Nowakowski Piotr, Żórawski Piotr, Cabaj Krzysztof, Gregorczyk Marcin, Purski Maciej, Mazurczyk Wojciech: Distributed packet inspection for network security purposes in software-defined networking environments, W: ARES '20: Proceedings of the 15th International Conference on Availability, Reliability and Security, 2020, Association for Computing Machinery, ISBN 978-1-4503-8833-7, s. 1-7, DOI:10.1145/3407023 .3409210, Punktacja MEiN = 70.
5. **Cabaj Krzysztof, Gregorczyk Marcin, Mazurczyk Wojciech, Nowakowski Piotr, Żórawski Piotr: Network Threats Mitigation Using Software-Defined Networking for the 5G Internet of Radio Light System, Security and Communication Networks, Wiley, vol. 2019, 2019, s. 1-22, DOI:10.1155/2019/4930908, Punktacja MEiN = 40, Cytowania WoS = 10, Cytowania Scopus = 13, Cytowania GS = 14, Scopus SNIP: 2016 = 0,842, Impact Factor WoS: 2019 (2-letni) = 1,288 - 2019 (5-letni) =1,306.**
6. Cabaj Krzysztof, Gregorczyk Marcin, Mazurczyk Wojciech, Nowakowski Piotr, Żórawski Piotr: Sniffing Detection within the Network, W: Proceedings of the 14th International Conference on Availability, Reliability and Security - ARES 2019, ICPS, 2019, Association for Computing Machinery, ISBN 978-1-4503-7164-3, s. 1-8, DOI:10.1145/3339252. 3341494, Punktacja MEiN = 70, Cytowania GS = 2.

7. **Cabaj Krzysztof, Gregorczyk Marcin, Mazurczyk Wojciech: Software-defined networking -based crypto ransomware detection using HTTP traffic characteristics, Computers & Electrical Engineering, vol. 66, 2018, s. 353-368, DOI: 10.1016 / j.compeleceng .2017.10.012, Punktacja MEiN = 70, Cytowania WoS = 54, Cytowania Scopus = 78, Cytowania GS = 138, Scopus SNIP: 2018 = 1,395, Impact Factor WoS: 2018 (2-letni) = 2,189 - 2018 (5-letni) = 2,337.**
8. Cabaj Krzysztof, Gregorczyk Marcin, Mazurczyk Wojciech, Nowakowski Piotr, Żórawski Piotr: SDN-based Mitigation of Scanning Attacks for the 5G Internet of Radio Light System, W: ARES 2018 Proceedings of the 13th International Conference on Availability, Reliability and Security / Doerr Christian, Schrittwieser Sebastian, Weippl Edgar (red.), 2018, Association for Computing Machinery, ISBN 978-1-4503-6448-5, s. 1-10, DOI:10.1145/3230833.3233248, Punktacja MEiN = 70, Cytowania Scopus = 9, Cytowania WoS = 4, Cytowania GS = 13.
9. Gregorczyk Marcin: Security of Linux Operating System – best practices, W: Prace Seminarium Naukowego Instytutu Telekomunikacji Politechniki Warszawskiej. Tom 2 / Jakubiak Andrzej (red.), 2016, Oficyna Wydawnicza PW, ISBN 978-83-7814-581-3, s. 58-67, Punktacja MEiN = 5.
10. Gregorczyk Marcin, Mazurczyk Wojciech, Szczypiorski Krzysztof: Wpływ steganografii sieciowej na opóźnienia w telefonii IP, Przegląd Telekomunikacyjny- Wiadomości Telekomunikacyjne, SIGMA NOT, vol. LXXXV, nr 8-9/2012, 2012, s. 797-808, Punktacja MEiN = 4.

Udział w projektach naukowych:

- IoRL - Internet of Radio Light (2017-2020) - ufundowany przez UE w ramach programu Horizon 2020. Rola w projekcie: starszy specjalista naukowo-techniczny.

Bibliometria:

- Sumaryczny IF: 7,222
- Sumaryczny SNIP: 3,955
- Sumaryczna punktacja MEiN: 469
- Cytowania Google Scholar: 170
- Cytowania Scopus: 102
- Cytowania Web of Science: 68
- H-index: 3

1. Wprowadzenie

Po wielkim sukcesie sieci 4G, oczekuje się, że sieci 5G, poprzez szeroki wachlarz usług wysokiej jakości i dostępności, będą kontynuacją w rozwoju infrastruktury następnej generacji [1]. Sieci 5G są obecnie w centrum zainteresowania zarówno jednostek naukowo-badawczych, rządowych jak i przedsiębiorstw na całym świecie [2]. Wraz z ich rosnącą popularnością, należy wziąć pod szczególną uwagę szeroko rozumiane aspekty bezpieczeństwa. Sieci 5G będą miały pozytywny wpływ na zadowolenie klientów i innowacyjność świadczonych usług teleinformatycznych, jednak poprzez połączenie wielu technologii, wprowadzone zostaną nowe wektory ataku, które mogą zostać wykorzystane przez cyberprzestępców [3]. Oznacza to, że poza niebezpieczeństwami znanymi z tradycyjnych sieci komputerowych, będziemy mieli do czynienia z nowymi, bardziej wyrafinowanymi atakami, skupionymi właśnie na specyfice sieci 5G. Jest to szczególnie istotne dzisiaj, gdy liczba użytkowników i urządzeń podłączonych stale do sieci wciąż wzrasta, a działanie rządów, organizacji i przedsiębiorstw na całym świecie jest uzależnione od Internetu.

Jednym z fundamentów sieci 5G jest technologia Software-Defined Networking (SDN) [4], a ta z kolei jest podstawą zdobywających coraz większą popularność chmur obliczeniowych, zarówno publicznych, prywatnych, jak i hybrydowych [5]. Zdecydowana większość producentów urządzeń sieciowych (fizycznych oraz wirtualnych) wspiera SDN za pomocą protokołu OpenFlow [6]. SDN poprzez oddzielenie infrastruktury od aplikacji kontrolujących jej działanie, umożliwia zarządzanie siecią w ustandaryzowany, a co najważniejsze, scentralizowany sposób [6].

Centralizacja jest tu kluczowym elementem, biorąc pod uwagę rozmiar współczesnych i przyszłych sieci oraz zasięg, rodzaje czy częstotliwość przeprowadzanych cyberataków [7]. Właśnie ze względu na centralizację widoku infrastruktury sieciowej technologia SDN będzie miała przewagę nad tradycyjnymi narzędziami podnoszącymi bezpieczeństwo sieci jak Intrusion Detection System/Intrusion Prevention System (IDS/IPS) czy firewall lub też będzie je uzupełniała czy z nimi współistniała [8], [9]. Niezbędne jest jednak przeprowadzenie badań, aby stwierdzić, czy tego typu rozwiązania będą także skutecznie wykrywać i przeciwdziałać obecnie występującym zagrożeniom cyberbezpieczeństwa. W literaturze naukowej można znaleźć publikacje związane z bezpieczeństwem i technologią SDN [6], [10], [11], [12], jednak jest ich wciąż mało i nie wyczerpują one całego spektrum możliwych zagadnień.

Według amerykańskiej Agencji ds. Cyberbezpieczeństwa i Bezpieczeństwa Infrastruktury[1] cyberbezpieczeństwo to sztuka ochrony sieci, urządzeń i danych przed nieuprawnionym dostępem lub wykorzystaniem w celach przestępczych oraz praktyka zapewniania poufności, integralności i dostępności informacji. Jest to także ciągły wyścig zbrojeń, gdzie atakujący wymyślają coraz bardziej wyszukane sposoby przełamywania zabezpieczeń informatycznych.

[1] https://www.cisa.gov

Z drugiej strony broniący tych systemów także starają się wykorzystać nowe technologie, aby zapewnić odpowiedni poziom bezpieczeństwa. Obie strony szukają nowych rozwiązań oraz sposobów przeciwdziałania swoim oponentom [13]. Architektura SDN poprzez scentralizowany widok całej infrastruktury sieciowej może stać się odpowiednią bronią w celu przeciwdziałania cyberprzestępcom. SDN nie może być jednak traktowane jako remedium na każdy rodzaj zagrożeń, jak na przykład fizyczna ochrona przed niepowołanym dostępem do serwerowni czy phishing. Jednak, gdy atak jest związany choćby w najmniejszym stopniu z sieciami teleinformatycznymi, SDN może zostać wykorzystany do jego wykrycia oraz zmitygowania.

Należy jednak zwrócić uwagę, że jedną z przesłanek, dla której została stworzona technologia SDN, jest łatwe i programowalne zarządzanie infrastrukturą, wpisującą się w nurt podejścia Software-Defined (Networking, Storage, Data Center) czy też Everything as a Service (XaaS) [14], [15]. Niestety, jak to często bywa z nowymi technologiami, ich twórcy skupiają się w pierwszym rzędzie na docelowej funkcjonalności, pomijając często aspekty zapewniania bezpieczeństwa [16], [17]. Dlatego też w ostatnim czasie pojawiają się prace naukowe poruszające tematy związane z koniecznością zapewnienia bezpieczeństwa samej technologii SDN [18], [6], [19], [11].

W związku z powyższym w niniejszej rozprawie doktorskiej przeprowadzono systematyczną analizę istniejącej literatury naukowej pod kątem sieci SDN i ich wykorzystania w celu poprawy cyberbezpieczeństwa. Zbadane zostały także aspekty związane z atakami na samą architekturę sieci opartych na SDN, które z racji swojego nowatorskiego podejścia do tematu infrastruktury sieciowej mają specyficzne wymagania, założenia, jak również i podatności.

Artykuły naukowe wchodzące w skład niniejszej rozprawy ([20], [21], [22], [23]), zostały opublikowane w renomowanych czasopismach oraz materiałach konferencyjnych, umieszczonych w wykazie czasopism punktowanych przez Ministerstwo Edukacji i Nauki, co potwierdza zasadność oraz innowacyjność podejmowanych badań i przydatność uzyskanych wyników. Wszystkie wymienione powyżej prace naukowe były także wynikiem praktycznych eksperymentów przeprowadzonych z wykorzystaniem rzeczywistych i używanych obecnie w świecie IT technologii, co podnosi ich wiarygodność w przeciwieństwie do prac opartych jedynie na symulacjach. Ponadto, artykuły [21] i [22] zostały opublikowane jako rezultaty badań projektu Internet of Radio Light[2] (IoRL), ufundowanego w ramach programu Horizon 2020 przez Komisję Europejską i realizowanego na Politechnice Warszawskiej w latach 2017-2020.

[2] https://iorl.5g-ppp.eu

1.1. Teza rozprawy

Technologia SDN zmienia dotychczasowe, utarte podejście do zarządzania sieciami teleinformatycznymi. Jak każda inna, nowatorska technologia w dowolnej dziedzinie, budzi ona pewne kontrowersje, obawy, ale i nadzieje. Jednym z aspektów, branych pod uwagę przy implementacji technologii SDN, jest bezpieczeństwo zarówno jej samej, jak i możliwość wykorzystania jej do zwiększenia bezpieczeństwa obecnych i przyszłych sieci teleinformatycznych. W związku z tym motywacją do przeprowadzenia badań naukowych i powstania niniejszej rozprawy jest odpowiedź na pytanie, czy technologia SDN może być właściwym wyborem w celu zapewnienia odpowiedniego poziomu bezpieczeństwa w sieci. Dodatkowo badania miały na celu zwiększenie ochrony użytkowników, systemów i firm, adresując popularne zagrożenia sieciowe.

W związku z powyższym teza niniejszej rozprawy została sformułowana następująco: *Możliwe jest stworzenie efektywnych metod zabezpieczeń z wykorzystaniem technologii Software-Defined Networking w celu skutecznego wykrywania i zapobiegania atakom sieciowym.*

Udowodnienie powyższej tezy będzie możliwe poprzez osiągnięcie głównych celów badawczych wymienionych poniżej:

- **Cel 1:** Opracowanie, zaimplementowanie i ocena systemu detekcji złośliwego oprogramowania typu ransomware, opartego na technologii SDN wykorzystującego analizę ruchu HTTP oraz porównanie charakterystyk dwóch rodzin ransomware: Locky oraz CryptoWall [20].
- **Cel 2:** Analiza nowatorskiego, zoptymalizowanego algorytmu wykrywania skanowania portów [21].
- **Cel 3:** Ocena zaproponowanego systemu detekcji oraz mitygacji wykorzystującego technologię SDN dla zagrożeń typu Denial of Service (DoS) [21].
- **Cel 4:** Detekcja pasywnego podsłuchu w sieci za pomocą nowatorskiej metody opartej na uczeniu maszynowym oraz ruchu warstwy aplikacji stosu TCP/IP [22].
- **Cel 5:** Zbadanie efektywności ulepszonego ataku typu *fingerprinting* na architekturę technologii SDN oraz zaproponowanie sposobów obrony przed nimi [23].

Powyższe cele zostały zrealizowane w ramach opublikowanych artykułów naukowych, których współautorem jest autor niniejszej rozprawy, a które przedstawiają szczegółowo zarówno podjęty problem jak i wykorzystaną metodykę badawczą, a także rezultaty przeprowadzonych badań oraz wyciągnięte wnioski. Zaproponowane metody wykrywania oraz mitygacji wybranych zagrożeń za pomocą technologii SDN odznaczają się wysoką skutecznością, co potwierdza słuszność przedstawionej powyżej tezy.

Dalsza struktura rozprawy przedstawia się następująco: rozdział 2. wyjaśnia działanie technologii SDN oraz zaadresowane, w ramach przedstawionych publikacji, wybrane zagrożenia w sieciach teleinformatycznych. Z kolei w rozdziale 3. zaprezentowano aktualny stan wiedzy odnośnie technologii SDN oraz aspektów związanych z jej bezpieczeństwem. Publikacje wchodzące w skład niniejszej rozprawy przedstawiono w rozdziałach 4., 5., 6. i 7. W końcu rozdział 8. zawiera podsumowanie przeprowadzonych prac badawczych oraz wskazuje potencjalne kierunki dalszego ich rozwoju.

2. SDN i cyberbezpieczeństwo

W poniższym rozdziale opisano podstawy technologii SDN, jej główne komponenty oraz cechy. Opisano także cyberzagrożenia, które zostały zaadresowane za pomocą tego typu rozwiązania w ramach publikacji wchodzących w skład niniejszej rozprawy doktorskiej [20], [21], [22], [23].

2.1. Technologia SDN

Technologia SDN, według wspierającej jej rozwój Open Networking Foundation[3] (ONF), to: *fizyczne oddzielenie płaszczyzny sterowania siecią od płaszczyzny przesyłania danych, gdzie płaszczyzna sterowania kontroluje kilka urządzeń.*

Architekturę technologii SDN zilustrowano na Rys. 2.1. Składa się ona z:

- Warstwy infrastruktury (Infrastructure Layer), w ramach której szybkie i tanie przełączniki (switch), fizyczne i wirtualne, przekazują ruch sieciowy do odpowiednich interfejsów.
- Warstwy zarządzania (Control Layer), w której specjalne oprogramowanie zwane kontrolerem (Controller) realizuje zarządzanie i steruje pracą przełączników.
- Warstwy Aplikacji (Application Layer), zawierającej realizujące funkcje biznesowe oraz infrastrukturalne aplikacje (poprzez kontroler).

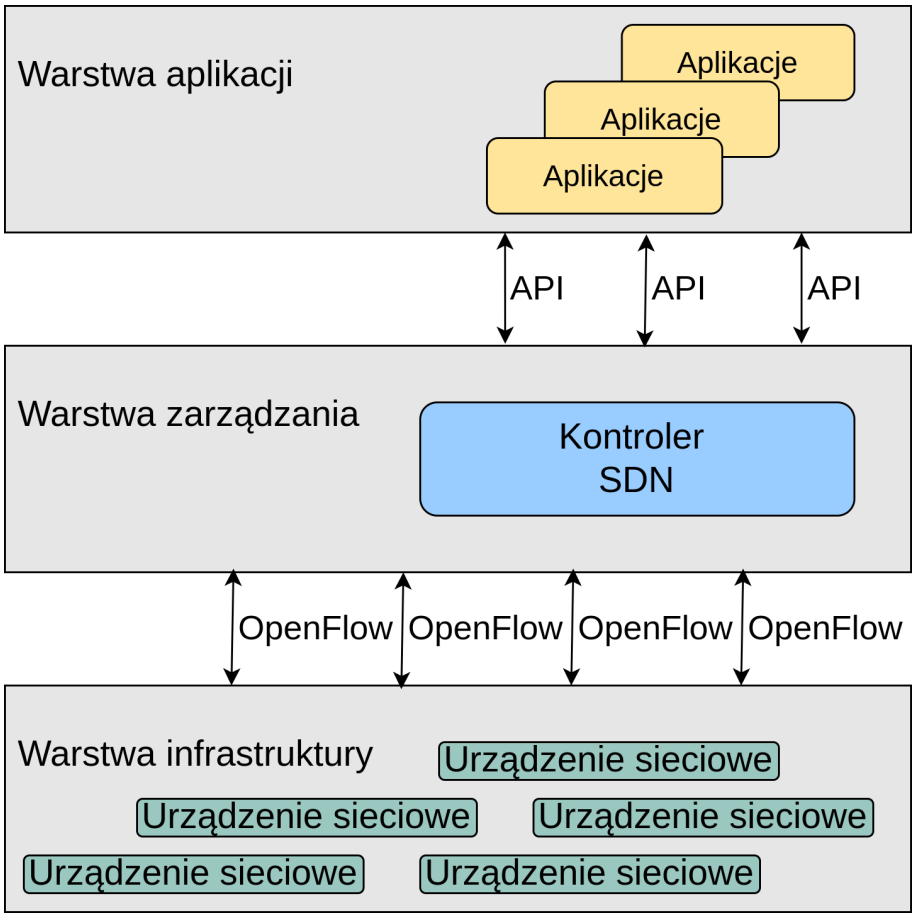
Jest to nowe podejście w realizacji i zarządzaniu siecią, które jest dynamiczne, łatwe w zarządzaniu, efektywne kosztowo i elastyczne, co czyni je idealną bazą dla dynamicznych aplikacji o wysokiej przepustowości. Technologia ta oddziela funkcje sterowania siecią i przesyłania danych, umożliwiając bezpośrednie programowanie sterowania siecią i abstrahowanie od infrastruktury bazowej dla aplikacji i usług sieciowych [24].

Protokół OpenFlow, również rozwijany przez ONF, standaryzuje komunikację pomiędzy warstwami zarządzania i infrastruktury obsługiwanych urządzeń sieciowych. OpenFlow jest pierwszym standardowym interfejsem zaprojektowanym specjalnie dla SDN, zapewniającym wysokowydajne, dokładne sterowanie ruchem w urządzeniach sieciowych wielu producentów [25]. Oryginalny projekt specyfikacji OpenFlow jest stale rozwijany i obecnie zawiera nowe funkcje, aby sprostać wyzwaniom narzucanym przez stale rozwijającą się technologię SDN [26].

Technologia SDN jest [24]:

- *Bezpośrednio programowalna*: funkcje sterowania są oddzielone od funkcji przesyłania, co umożliwia programowalne konfigurowanie urządzeń.
- *Zwinna (Agile)*: abstrahowanie sterowania od przekazywania informacji pozwala administratorom dynamicznie dostosowywać przepływ ruchu w całej sieci do zmieniających się potrzeb biznesowych i infrastrukturalnych.

[3] https://opennetworking.org



Rys. 2.1. Uproszczona architektura SDN.

- *Centralnie zarządzana*: scentralizowany kontroler SDN utrzymuje globalny widok sieci, który przedstawiany jest aplikacjom jako pojedynczy, logiczny przełącznik.
- *Konfigurowalna programowo*: SDN umożliwia administratorom sieci bardzo szybkie konfigurowanie, zarządzanie, zabezpieczanie i optymalizowanie zasobów sieciowych za pomocą dynamicznych, zautomatyzowanych programów SDN, które mogą oni pisać samodzielnie, ponieważ programy te nie zależą od oprogramowania firm trzecich.
- *Oparta na otwartych standardach i niezależna od producentów sprzętu i oprogramowania*: dzięki wykorzystaniu otwartych standardów, SDN upraszcza projektowanie i obsługę sieci, ponieważ instrukcje dostarczane są przez kontrolery SDN, a nie przez wiele urządzeń i protokołów specyficznych dla danego producenta.

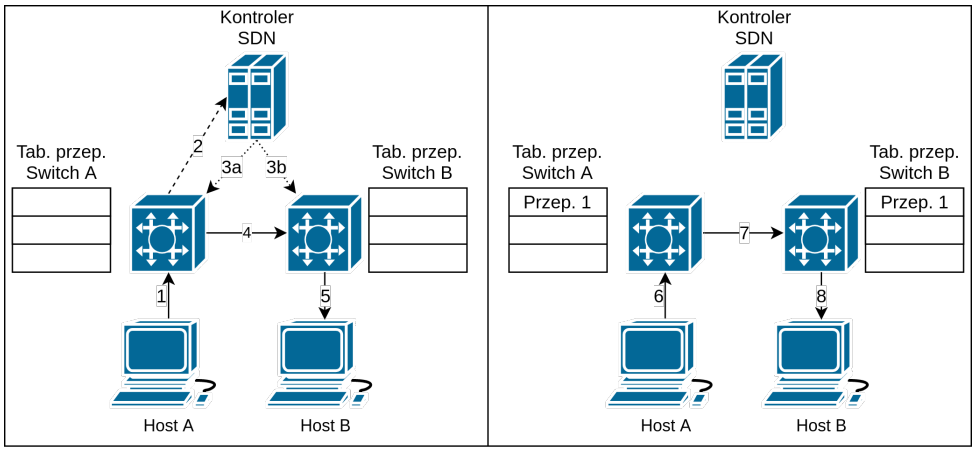
W architekturze rozwiązań SDN pakiety o podobnych charakterystykach są grupowane w przepływy (flows), które przełączniki przetwarzają w ten sam sposób. Jest to realizowane za pomocą specjalnych tablic przepływów (flow tables), które zawierają informacje o tym, co zrobić z danym pakietem [27]. W szczególności może zostać podjęta decyzja o wysłaniu ruchu do kontrolera w celu jego dalszej analizy; modyfikacji określonych pól pakietu; usunięciu pakietu z sieci; przesłaniu pakietu dalej zgodnie z jego przeznaczeniem.

Dzięki technologii SDN administrator sieci zyskuje kontrolę nad całą infrastrukturą z wykorzystaniem jednego panelu sterowania. Związane z tym zalety to [28]:

- Przejrzysty widok na topologię sieci pozwala na podejmowanie lepszych decyzji, np. bardziej efektywne równoważenie obciążenia i lepsze rozłożenie ruchu.
- Łatwe dodawanie nowych komponentów do sieci (takich jak urządzenia typu router, switch czy load balancer).
- Nie potrzeba konfigurować setek urządzeń osobno, ponieważ wszystko można zrobić z jednego centralnego panelu sterowania.
- Dużo łatwiej jest przeprojektować architekturę sieci bez konieczności fizycznej ingerencji w okablowanie.
- Konfiguracja sieci jest kompilowana automatycznie przez kontroler, co minimalizuje ryzyko popełnienia błędu przez człowieka.
- SDN zapewnia automatyzację technologii sieciowej, ułatwiając i przyspieszając pracę administratora.
- Administrator może w łatwiejszy sposób, niż to się odbywało w przypadku tradycyjnych sieci teleinformatycznych, kontrolować bezpieczeństwo systemu.

Aplikacja SDN to oprogramowanie przeznaczone do wykonywania określonych zadań w środowisku SDN. Kontaktuje się ona z kontrolerem za pomocą specjalnego interfejsu programistycznego (Application Programming Interface – API). Aplikacje SDN mogą zastępować i rozszerzać funkcje, które są implementowane w urządzeniach sprzętowych w sieciach tradycyjnych. Aplikacje uruchomione na kontrolerze, który zarządza infrastrukturą sieciową, mogą realizować nie tylko funkcje biznesowe, ale także chronić przed cyberatakami z zewnątrz jak i wewnątrz sieci [29]. W zależności od reguł zainstalowanych przez aplikację kontrolera, przełącznik zarządzany przez OpenFlow może, na polecenie kontrolera, zachowywać się jak router, switch, firewall, lub pełnić inne role (np. rozwiązania typu load balancer, czy traffic shaper). Dzięki temu, bez utraty czasu na fizyczną rekonfigurację czy zmianę okablowania elementów sieciowych, można realizować wymagane funkcje sieciowe potrzebne w danym momencie [6].

Przykład przepływu ruchu w sieciach SDN zilustrowano na Rys. 2.2. Na początku przedstawionej komunikacji, tablice przepływów obu przełączników (A i B) są puste. Host A wysyła pakiet do hosta B (1). Przełącznik A nie wie co należy zrobić z takim ruchem, więc wysyła go do kontrolera, aby to on podjął decyzję (2). W tym celu używana jest specjalna wiadomość OpenFlow – *Packet_In*, która zawiera (w zależności od wybranej konfiguracji) cały pakiet lub tylko jego nagłówki. Kontroler po analizie pakietu i zastosowaniu wytycznych aplikacji SDN, może zdecydować o zainstalowaniu reguł, które zezwolą na taki ruch (3a i 3b). Jest to także realizowane za pomocą wiadomości OpenFlow – *Flow_Mod*. Pakiet jest wysyłany z przełącznika A do B (4), a następnie dostarczany do hosta B (5). Gdy host A ponownie wyśle pakiet do hosta B (6), przełącznik A, dzięki zainstalowanemu wcześniej w tablicy przepływowi, nie musi



Rys. 2.2. Przykładowy przepływ ruchu w sieciach SDN.

się kontaktować z kontrolerem i automatycznie przekazuje ruch do przełącznika B (7), który dostarczy ruch do hosta B (8). W przedstawionym przykładzie, aplikacje SDN sterujące pracą kontrolera, mogą także zdecydować o usunięciu pakietu z sieci lub jego dowolnej modyfikacji, jeśli wynika to z implementowanych przez aplikacje funkcjonalności realizujących cele infrastrukturalne czy biznesowe, jak na przykład ochrona przez cyberatakami.

2.2. Wybrane cyberzagrożenia

W 2011 roku Departament Obrony Stanów Zjednoczonych[4] oficjalnie zaliczył cyberprzestrzeń jako element piątej domeny działań wojennych, tj. operacji informacyjnych [30]. Pozostałe cztery domeny to działania na lądzie, morzu, w powietrzu i przestrzeni kosmicznej. Uwidacznia to, jak ważnym aspektem w świecie IT jest bezpieczeństwo i ochrona przed cyberatakami.

W celu usystematyzowania istniejących zagrożeń warto posłużyć się MITRE ATT&CK Framework[5], który stał się jednym z najbardziej szanowanych i najczęściej cytowanych źródeł w dziedzinie cyberbezpieczeństwa. MITRE ATT&CK to baza wiedzy o technikach, które atakujący może wykorzystać, aby wyrządzić szkody w systemach i sieciach. Wiedza o nich może także zostać wykorzystana do obrony przed zagrożeniami cyberbezpieczeństwa. Jest to ciągle aktualizowana struktura typowych taktyk, technik i procedur używanych przez specjalistów od cyberbezpieczeństwa oraz cyberprzestępców. W chwili powstawania niniejszej rozprawy MITRE ATT&CK składa się z 14 taktyk oraz 218 technik ataków[6]. Tab. 2.1 przedstawia zestawienie wszystkich taktyk oraz liczbę poszczególnych technik w każdej z nich.

Ważne są także zależności pomiędzy taktykami i technikami w poszczególnych fazach ataku. Zazwyczaj najpierw atakujący wykona pewnego rodzaju rekonesans, zanim przejdzie

[4] https://www.defense.gov
[5] https://attack.mitre.org
[6] Stan na dzień: 12.01.2022

Tabela 2.1. Zestawienie taktyk MITRE ATT&CK

Taktyka	**Cel**	**Liczba Technik**
Reconnaissance	Zebranie informacji na potrzeby przyszłych ataków	10
Resource Development	Ustalenie zasobów do wspierania ataków	7
Initial Access	Uzyskanie dostępu do sieci	9
Execution	Uruchomienie złośliwego kodu	12
Persistence	Utrzymanie pozycji	19
Privilege Escalation	Uzyskanie zwiększonych uprawnień	13
Defense Evasion	Unikanie wykrycia	40
Credential Access	Wykradanie loginów i haseł	15
Discovery	Odkrywanie atakowanego środowiska	29
Lateral Movement	Poruszanie się po środowisku	9
Collection	Zebranie istotnych danych	17
Command and Control	Komunikacja do oraz kontrola nad przejętym środowiskiem	16
Exfiltration	Kradzież danych	9
Impact	Niszczenie systemów i danych	13

Tabela 2.2. Zestawienie zaadresowanych w niniejszej rozprawie zagrożeń i ich przynależność do taktyk i technik w ramach MITRE ATT&CK.

Publikacja	**Zagrożenie**	**Taktyka**	**Technika**
[20]	Ransomware	Impact	Data Encrypted for Impact
[21]	TCP SYN Scan	Reconnaissance	Active Scanning
[21]	DHCP Starvation	Impact	Denial of Service
[22]	Sniffing	Credential Access	Network Sniffing
[23]	Fingerprinting	Discovery	System Information Discovery

do ataku, który może uwidocznić kolejne możliwości, tj. wektory ataku.

Publikacje wchodzące w skład niniejszej rozprawy [20], [21], [22] oraz [23] adresują pięć różnych rodzajów zagrożeń w ramach czterech taktyk. Tab. 2.2 prezentuje to zestawienie. Poniżej scharakteryzowano każde z zagrożeń oraz jego przynależność do taktyk w ramach MITRE ATT&CK.

Ransomware

Ransomware jest rodzajem złośliwego oprogramowania tzw. *malware*. Jego nazwa pochodzi od angielskich słów *ransom*, które oznacza okup oraz *software* - oprogramowanie. Ransomware blokuje dostęp do systemu czy plików użytkownika. Szczególnie niebezpieczne jest oprogramowanie typu crypto ransomware, które szyfruje ważne dane czy aplikacje użytkowników, a ich odzyskanie jest możliwe dopiero po zapłaceniu okupu [31]. Należy zwrócić uwagę,

że jeśli proces szyfrowania został zaimplementowany prawidłowo, odszyfrowanie plików bez znajomości tajnego klucza jest w zasadzie niemożliwe.

Ransomware w ramach MITRE ATT&CK zaliczany jest do taktyki *Impact*, w której atakujący próbuje manipulować, zakłócać lub niszczyć systemy i dane użytkownika. Powoduje to zakłócenia dostępności lub narażenie integralności danych poprzez manipulowanie procesami biznesowymi i operacyjnymi w wyniku podmiany lub niszczenia danych. Atakujący w ramach techniki *Data Encrypted for Impact* mogą szyfrować dane w systemach docelowych lub w dużej liczbie systemów w sieci w celu przerwania dostępności zasobów systemowych i sieciowych. Mogą oni także próbować uniemożliwić dostęp do przechowywanych danych, szyfrując pliki lub dane na dyskach lokalnych i zdalnych oraz zachowując dla siebie klucz deszyfrujący. Może to mieć na celu uzyskanie od ofiary okupu w zamian za odszyfrowanie lub przekazanie klucza deszyfrującego. W przypadku, gdy okup nie zostanie zapłacony, klucz nie jest przekazywany i ofiara traci na zawsze dostęp do swoich danych.

Ransomware jest jednym z największych zagrożeń w świecie IT, zarówno dla użytkowników domowych jak i firm, od małych do wielkich korporacji [32]. Uwidaczniają to poniższe statystyki:

- Według raportu przedstawionego przez Datto[7] w 2019 roku ransomware jest najbardziej dochodowym rodzajem zagrożenia dla cyberprzestępców [33].
- 37% organizacji respondentów ankiety przeprowadzonej w styczniu i lutym 2021 roku przez Sophos[8] zostało dotkniętych przez ransomware [34].
- Według serwisu Business Insider[9] największy okup zapłacony przez firmę ubezpieczeniową to 40 milionów dolarów w marcu 2021 roku [35].
- National Security Institute[10] donosi, że przeciętny okup wzrósł z 5 tysięcy dolarów w 2018 do 200 tysięcy dolarów w 2021 [36].
- Według Coveware[11] średni czas przestoju firmy po ataku ransomware wynosi 21 dni w czwartym kwartale 2020 roku [37].

Istnieje kilka sposobów obrony przed ransomware, na przykład utworzenie kopii zapasowej danych, czy instalacja oprogramowania antywirusowego. W zaprezentowanej w rozdziale 4. publikacji przedstawiono nowatorskie podejście do wykrywania komunikacji oprogramowania ransomware z serwerem C&C oparte na technologii SDN, które bazuje na charakterystyce tej komunikacji. Analizując taki ruch, możliwe jest wykrycie sekwencji wysyłanych wiadomości, co może wskazywać na obecność tego typu złośliwego oprogramowania i w rezultacie skutecznie je zablokować.

[7] https://www.datto.com
[8] https://www.sophos.com
[9] https://www.businessinsider.com
[10] https://www.nsi.org
[11] https://www.coveware.com

TCP SYN Scan

Skanowanie portów w warstwie transportowej stosu TCP/IP jest zazwyczaj elementem rekonesansu, który atakujący wykorzystuje, aby dowiedzieć się, jakie usługi są uruchomione na serwerze lub całej podsieci adresów IP [38]. W przypadku skanowania 'TCP SYN', używana jest flaga SYN w protokole TCP do zainicjowania połączenia. Skanowanie typu 'TCP SYN' jest najczęściej stosowanym rodzajem skanowania portów ze względu na jego uniwersalność i szybkość [39]. W pracy [40] zdefiniowano wymagane zachowanie każdego urządzenia TCP/IP w ten sposób, że przychodzące żądanie połączenia rozpoczyna się segmentem SYN, po którym z kolei musi nastąpić pakiet SYN/ACK z usługi nasłuchującej. Przy użyciu tej metody możliwe jest skanowanie tysięcy portów na sekundę. Taki typ skanowania jest zwykle określany jako skanowanie półotwarte, ponieważ atakujący nie kończy procesu tzw. *three-way handshake*, podczas którego nawiązywane jest poprawne połączenie TCP. Szybkość skanowania jest bardzo duża, ponieważ nie jest tracony czas na dokończenie nawiązywania lub zrywania połączenia. Technika ta pozwala atakującemu na skanowanie przez firewalle stanowe ze względu na powszechną konfigurację, zgodnie z którą segmenty TCP SYN dla nowego połączenia są dozwolone dla każdego portu. Skanowanie TCP SYN może również natychmiast wykryć 3 z 4 ważnych typów statusu portu: otwarty, zamknięty i filtrowany [41].

Atakujący może uzyskać listę usług działających na zdalnych hostach i wykorzystać tę wiedzę w dalszych atakach. W ramach klasyfikacji MITRE ATT&CK atak ten jest kategoryzowany jako taktyka *Reconnaissance* i technika *Active Scanning*, czyli aktywne skanowanie zwiadowcze w celu zebrania informacji, które mogą być wykorzystane podczas namierzania potencjalnych celów ataku. Aktywne skanowanie to takie, w którym przeciwnik zbiera informacje o infrastrukturze ofiary dzięki specjalnie spreparowanemu ruchowi sieciowemu, w przeciwieństwie do innych form rozpoznania, które nie wymagają bezpośredniej interakcji [42]. Skanowanie portów może być także używane przez botnety (np. Mirai [43]), gdzie zwielokrotnienie liczby urządzeń skanujących pozwala na jeszcze szybsze przeprowadzenie tego typu ataku.

Wczesne wykrycie i zablokowanie skanowania może uniemożliwić lub utrudnić dalsze fazy ataków. Jeśli atakujący nie jest w stanie wykryć działających usług, nie będzie w stanie wyrządzić im żadnych szkód. Ograniczone jest także zużycie zasobów poprzez zmniejszenie liczby nawiązanych sesji TCP. Technologia SDN, jak udowodniono w publikacji [21] zaprezentowanej w rozdziale 5., może wykrywać i zapobiegać próbom skanowania portów z dużą skutecznością.

DHCP Starvation

Dynamic Host Configuration Protocol (DHCP) jest protokołem do automatycznego przydzielania adresów IP urządzeniom podłączonym do sieci, który w zamyśle miał ułatwić komunikację i zarządzanie dynamicznymi sieciami [44] . Technologia ta eliminuje konieczność indywidualnego, ręcznego konfigurowania urządzeń sieciowych i składa się z dwóch komponentów:

serwera DHCP oraz jego klientów (urządzeń sieciowych). Po podłączeniu do sieci, a następnie okresowo, klient żąda od serwera DHCP adresu IP, aby mógł się komunikować z innymi urządzeniami. Podczas tworzenia protokołu DHCP nie wzięto jednak pod uwagę aspektów związanych z bezpieczeństwem.

DHCP Starvation to atak, w którym cały zakres dostępnych adresów IP przydzielanych z puli przez serwer DHCP jest zarezerwowany, a co za tym idzie, zużyty i niedostępny dla innych hostów [45]. Atak jest relatywnie prosty do zaimplementowania, ponieważ klient, przy użyciu sfałszowanych adresów MAC, może wciąż prosić o kolejny adres IP, w zasadzie nigdy go nie używając. Możliwość przeprowadzenia takiego ataku została opisana w publikacji [44]. W przypadku wyczerpania całej puli dostępnych adresów IP, serwer przestaje świadczyć usługę, ignorując wszelkie żądania o przydzielenie adresu IP. Można zatem ten atak uznać za atak typu Denial of Service (DoS), w którym następuje przeciążenie lub wyczerpanie zasobów systemu świadczącego określoną usługę. Atak ten zatem paraliżuje komunikację IP urządzeń w warstwie internetowej stosu TCP/IP. Dotyczy to przede wszystkim dynamicznych i skalowalnych środowisk, jak chmury obliczeniowe czy publiczne sieci z otwartym dostępem. DoS oraz jego szczególny rodzaj Distributed DoS (DDoS) są obecnie traktowane jako znaczące zagrożenie w świecie IT [46].

Atak typu DoS w MITRE ATT&CK zaliczany jest również do taktyki *Impact*, jednak do techniki *Network Denial of Service*. Obrona przed tym atakiem nie jest trywialna, ponieważ zazwyczaj dużym wyzwaniem jest poprawne odróżnienie ruchu, który jest elementem ataku od zwyczajnego ruchu pochodzącego od klienta chcącego skorzystać z usługi. W publikacji [21] przedstawionej w rozdziale 5.. zaproponowano nowatorskie podejście, oparte na technologii SDN, które adresuje atak DHCP Starvation znacząco ograniczając jego skuteczność.

Sniffing

Podsłuch (sniffing) jest typowym elementem fazy rekonesansu wielu ataków sieciowych [38]. Może być wykonany przez atakującego, który jest w stanie uzyskać dostęp do docelowej infrastruktury sieciowej. Głównym celem takich działań jest identyfikacja systemów, protokołów i aplikacji, które są uruchomione w danej infrastrukturze. Następnie atakujący może określić potencjalne podatności, które mogą być wykorzystane do przejęcia kontroli nad siecią. Intruz może także poznać m.in. loginy i hasła przesyłane przez nieszyfrowane protokoły. W większości przypadków działania te poprzedzają właściwy atak, dlatego też należy zauważyć, że wczesne wykrycie sniffingu ma ogromne znaczenie, gdyż pozwala obrońcy przygotować się do dalszych faz ataku. Ponadto, jeżeli system bezpieczeństwa wprowadzony w sieci jest w stanie wykryć podsłuchujące urządzenia, może podjąć działania zaradcze i w ten sposób zmniejszyć prawdopodobieństwo ataku. Sniffing jest zazwyczaj realizowany za pomocą dedykowanego oprogramowania, zwanego snifferami [47], z których najbardziej

znane przykłady to TcpDump[12] czy Wireshark[13]. Działają one na podstawie pasywnej analizy ruchu sieciowego przesyłanego wewnątrz sieci. Ze względu na tę cechę, działania takie są zazwyczaj trudne do wykrycia.

MITRE ATT&CK klasyfikuje Sniffing jako taktykę *Credential Access* oraz technikę *Network Sniffing*. Sniffing może również ujawnić szczegóły konfiguracji, takie jak działające usługi, numery wersji i inne cechy sieci (np. adresy IP, nazwy hostów, identyfikatory VLAN) niezbędne do późniejszych działań związanych z *Lateral Movement*, czyli stopniowym poruszaniem się cyberprzestępców po sieci podczas wyszukiwania kluczowych danych i zasobów, które są ostatecznie celem ich kampanii ataków. Atak ten dotyczy zatem warstwy łącza danych w stosie TCP/IP oraz pośrednio pozostałych warstw zależnych od niej.

Obroną przed atakiem sniffingu może być szyfrowanie ruchu sieciowego, jednak dotyczy to warstwy aplikacji stosu TCP/IP. Nawet jeśli atakujący nie będzie mógł odszyfrować przesyłanej informacji, nadal będzie mógł odczytać metadane prowadzonej komunikacji, co może być równie niebezpieczne [48].

Sniffing ze względu na swoją pasywną naturę, jest trudny do wykrycia. Jednakże, jak udowodniono w artykule [22] w rozdziale 5., z pomocą sztucznej inteligencji, a dokładnie uczenia maszynowego, jest to możliwe z dużą skutecznością.

Fingerprinting

Fingerprinting to technika, w ramach której sprzęt i oprogramowanie jest dzielone na rozłączne klasy poprzez analizę komunikatów wysyłanych przez nie, zazwyczaj w odpowiedzi na jakąś formę aktywnego sondowania. Klasy te mogą odpowiadać na przykład systemowi operacyjnemu, marce, czy konfiguracji routera. Fingerprinting może pomóc w sporządzeniu listy węzłów sieci i zidentyfikowaniu hostów podatnych na zagrożenia pod względem bezpieczeństwa i odporności na awarie [49]. Takie działanie, jeśli zostało przeprowadzone przez administratora, nie budzi kontrowersji (podobnie jak skanowanie portów). Jednak atakujący może użyć technik fingerprinting'u, aby poznać podatności czy konfigurację urządzeń sieciowych i wykorzystać tę wiedzę w dalszych etapach swoich nieetycznych działań.

MITRE ATT&CK zalicza fingerprinting do taktyki *Discovery* i techniki *System Information Discovery*, w którym informacje uzyskane podczas sondowania mogą zostać wykorzystane do kształtowania dalszych zachowań, w tym do określenia, czy atakujący w pełni zainfekuje cel i/lub podejmie określone działania.

W przypadku publikacji [23] przedstawionej w rozdziale 7. udowodniono, że odpowiednio spreparowany ruch może ujawnić parametry użytego przełącznika SDN, które powinny pozostać poufne (m.in. rozmiar tablicy przepływów). Atak taki jest możliwy ze względu na specyfikę architektury technologii SDN, tj. oddzielenie płaszczyzny sterowania od transportu. W artykule [23] zaproponowano także dwie metody obrony przed zagrożeniem poznania

[12] https://www.tcpdump.org
[13] https://www.wireshark.org

poufnych parametrów przełącznika SDN.

Jak można zauważyć z Tab. 2.2, trzy z pięciu adresowanych zagrożeń (TCP SYN Scan, Sniffing, Fingerprinting) służą atakującemu do wykonania aktywności związanych z rekonesansem czy wykryciem, z jakimi usługami ma on do czynienia. Uzyskane dzięki temu informacje mogą posłużyć lub w ogóle umożliwić przeprowadzenie dalszych faz ataków, jak na przykład pozostałe dwa zaadresowane zagrożenia (Ransomware czy DHCP Starvation).

3. Stan wiedzy

Naukowcy oraz specjaliści z branży IT, widząc potencjał, jaki drzemie w technologii SDN z punktu widzenia jej przydatności w zarządzaniu infrastrukturą sieciową, zaczęli także badać aspekty związane z bezpieczeństwem tej architektury. Powstały liczne prace naukowe o bezpieczeństwie technologii SDN, a co za tym idzie - przeglądy literatury naukowej związanej z tą tematyką m.in. [18], [50], [6], [10], [51], [52], [11], [53], [12], czy [54].

W niniejszym rozdziale przedstawiono stan wiedzy związany z szeroko pojętymi aspektami bezpieczeństwa w technologii SDN. W literaturze światowej temat bezpieczeństwa SDN rozpatrywany jest przede wszystkim w dwóch aspektach [6]:

- podnoszenia bezpieczeństwa użytkowników i systemów w sieci z wykorzystaniem specjalistycznych aplikacji i mechanizmów korzystających z infrastruktury SDN,
- bezpieczeństwa architektury samej technologii SDN.

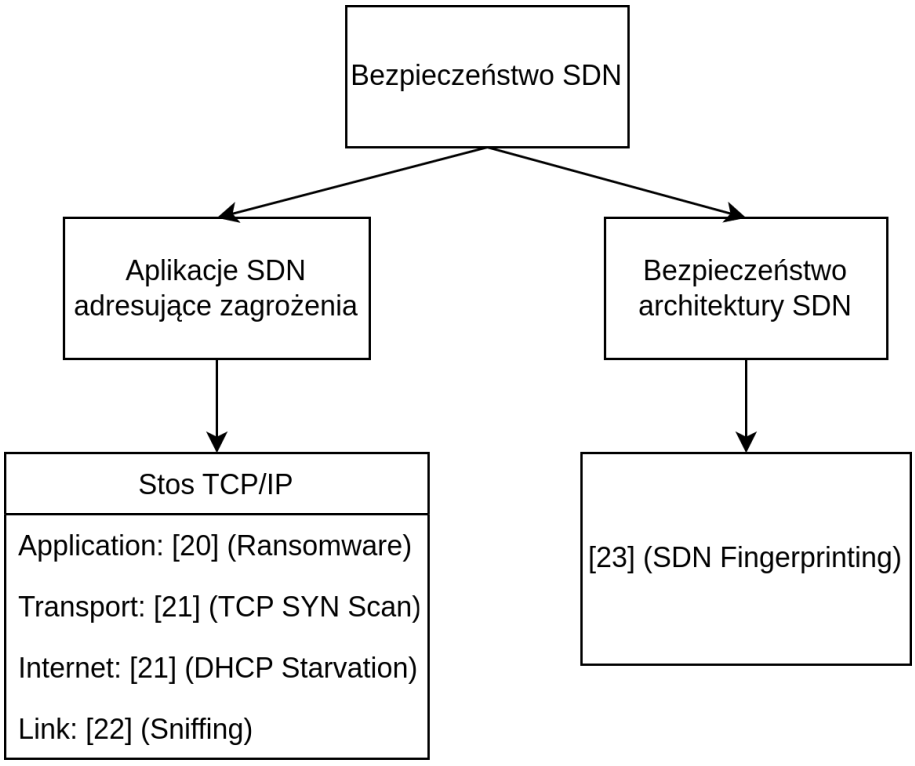
Podział ten jest w zasadzie oczywisty - SDN jako technologia nowa, ale jednocześnie posiadająca wielkie możliwości, wydaje się być dobrym wyborem do stworzenia efektywnych metod bezpieczeństwa w celu wykrywania i zapobiegania atakom sieciowym. Z drugiej jednak strony takie nowatorskie podejście wprowadza nowe elementy i czynniki, które sprawiają, że pojawiają się zagrożenia, o których do tej pory nie było mowy.

W niniejszej rozprawie wprowadzono jeszcze jeden podział w ramach aplikacji podnoszących bezpieczeństwo w sieciach SDN, a mianowicie podział ze względu na warstwę stosu TCP/IP, do której odnosi się zabezpieczenie. Został on po raz pierwszy zastosowany w ramach niniejszej rozprawy. Należy zwrócić także uwagę, że podział ten nie ma zastosowania w przypadku zabezpieczania samej technologii SDN, ponieważ tego typu ataki biorą na cel samą architekturę, jaką wprowadza i wykorzystuje ta technologia. Wspomnianą klasyfikację zilustrowano na Rys. 3.1, uwzględniając publikacje wchodzące w skład niniejszej rozprawy [20]–[23]. Należy zwrócić uwagę, że wspomniane publikacje pokrywają całe spektrum aspektów związanych z bezpieczeństwem technologii SDN.

Poniżej scharakteryzowano wybrane prace naukowe dotyczące tematyki niniejszej rozprawy zgodnie z przedstawioną powyżej klasyfikacją. W pracach [20]–[23] zawarto analizę stanu wiedzy pod kątem konkretnych, rozważanych problemów badawczych. Natomiast w tym rozdziale przedstawiono pozycje opisujące szerszy aspekt bezpieczeństwa powiązanego z technologią SDN, uwzględniając także najnowsze publikacje.

3.1. Aplikacje podnoszące bezpieczeństwo w sieciach SDN

Aplikacje SDN implementują logikę sterowania, która, przetłumaczona na polecenia zrozumiałe przez kontroler, rekonfiguruje urządzenia w płaszczyźnie infrastruktury. Dotychczas powstało wiele aplikacji SDN służących usprawnieniu usług wymaganych do zabezpieczenia systemów i sieci, takich jak egzekwowanie polityk bezpieczeństwa, kontrola dostępu, firewalling czy implementacja urządzeń pośrednich tzw. middlebox [55]–[59], losowa zmiana



Rys. 3.1. Bezpieczeństwo technologii SDN

hostów [57] (tj. losowa i częsta zmiana adresów IP hostów końcowych, wprowadzająca celowe zaburzenie widoku atakujących na infrastrukturę) [60], automatyczna analiza i kierowanie podejrzanego ruchu do dalszej inspekcji przez wyspecjalizowane urządzenia [61], wykrywanie anomalii w ruchu [62]–[64], precyzyjna kontrola dostępu do sieci oparta na przepływach [65], czy precyzyjne egzekwowanie polityk dla osobistych aplikacji mobilnych [66].

W literaturze można znaleźć także publikacje związane z wykrywaniem i przeciwdziałaniem atakom sieciowym z wykorzystaniem technologii SDN, jak np. podszycie (spoofing) [59], [67]–[74]. Dostępne są także prace adresujące ataki socjotechniczne, np. [75] czy [76].

Poniżej przedstawiono publikacje, mitygujące za pomocą technologii SDN, opisane w tej rozprawie rodzaje zagrożeń (w ramach [20], [21], [22]), tj: złośliwe oprogramowanie, skanowanie sieciowe, ataki DoS oraz podsłuch sieciowy. Należy zauważyć, że wybrano tylko takie prace, które przedstawiają metodykę, technikę lub procedurę obrony przed cyberzagrożeniami za pomocą technologii SDN.

Złośliwe oprogramowanie

Malware, w szczególności ransomware, jest jednym z najczęściej spotykanych zagrożeń w świecie IT. Nie dziwi więc fakt, że naukowcy szukają coraz bardziej wyrafinowanych metod przeciwdziałania złośliwemu oprogramowaniu przy użyciu nowych technologii, takich jak SDN.

Autorzy w [77] proponują dwie metody mitygacji ransomware w czasie rzeczywistym, bazując na wynikach analizy zachowania rodziny CryptoWall. Ponadto, prezentują aplikację współpracującą z kontrolerem SDN oraz przeprowadzają ocenę przedstawionego systemu. Zaproponowane metody opierają się na dynamicznym tworzeniu czarnych list (black list) serwerów proxy ransomware wykorzystywanych do przekazywania komunikacji pomiędzy ofiarą a serwerem zarządzającym. W pierwszej metodzie wszystkie żądania DNS przekazywane są przez aplikację do kontrolera, a następnie kontroler sprawdza nazwy domen z bazą czarnych list. W przypadku wykrycia złośliwej domeny, kontroler odrzuca pakiet, blokuje cały ruch z węzła źródłowego i alarmuje osoby odpowiedzialne za bezpieczeństwo sieci. Druga metoda została opracowana w celu zwiększenia wydajności tej pierwszej. Zrealizowana aplikacja wysyła kopię żądań DNS do kontrolera, a kontroler i aplikacja współbieżnie wykonują ten sam proces, który został zdefiniowany w pierwszej metodzie. Dzięki temu druga metoda nie wprowadza dodatkowego opóźnienia w komunikacji. Czas potrzebny na reakcję - czyli zablokowanie ruchu do złośliwych domen, nie przekraczał 100ms, co w zestawieniu ze średnim czasem pobierania klucza szyfrującego ransomware wynoszącym ponad 9s, daje wystarczającą ochronę przed CryptoWall.

Z kolei w pracy [78] autorzy wykorzystują tzw. programowalne silniki przekierowujące do monitorowania ruchu sieciowego pomiędzy zainfekowanym komputerem a serwerem zarządzającym ransomware. Następnie wyodrębniane są z tego ruchu wysokopoziomowe cechy przepływu, wykorzystywane do klasyfikacji oprogramowania typu ransomware. Procesor przepływów oraz binarny klasyfikator "random forest" pozwala na wykrycie złośliwej aktywności sieciowej bez wymogu głębokiej inspekcji pakietów. Wyniki badań opartych na symulacji pokazują, że zaproponowany model klasyfikacyjny osiąga współczynnik wykrywalności przekraczający 86%.

W artykule [79] przedstawiono architekturę o nazwie Malware Analysis Architecture (MARS) służącą obronie przed złośliwym oprogramowaniem. Składa się ona z piaskownicy (sandbox), kontrolera SDN oraz puli zasobów. W sandboxie uruchamiane jest złośliwe oprogramowanie i wykonywane są połączenia do sieci Internet lub do elementów sieci lokalnej. Zdarzenia związane z ruchem sieciowym przesyłane są do modułów na kontrolerze. Następnie kontroler przekazuje przepływy sieci intranetowej do powiązanych urządzeń w puli zasobów i zarządza charakterystykami przepływów (np. przepustowością) w celu egzekwowania wcześniej zdefiniowanych polityk. Kontroler może również wejść w interakcję z piaskownicą i ponownie uruchomić nową analizę. W pracy [80] autorzy proponują rozwiązanie WormHunter, czyli system obrony przed robakami oparty na SDN. WormHunter zarządza tablicami przepływów w przełącznikach SDN i dynamicznie rozmieszcza różne systemy będące pułapkami (honeypot) o różnych strukturach sieciowych. Jeśli WormHunter wykryje anomalie, przekierowuje podejrzany ruch do sieci rozwiązań typu honeypot, aby umożliwić jego bezpieczną analizę. Wszystkie zachowania związane z atakiem są także rejestrowane w systemie w celu dalszej analizy. Autorzy przedstawili także analizę porównawczą

rozwiązania WormHunter z podobnymi systemami detekcji nieopartymi na technologii SDN. Wynikło z niej, że zaproponowane podejście jest rozwiązaniem łatwiejszym w implementacji, bardziej elastycznym i skalowalnym oraz wymagającym mniejszych zasobów sprzętowych.

Z kolei w artykule [81] przedstawiono wyniki analizy ransomware (WannaCry) oraz framework do obrony przed nim. Rozwiązanie wykrywa podejrzane działania poprzez monitorowanie ruchu sieciowego i blokuje zainfekowane hosty poprzez dodawanie wpisów do tablicy przepływów w przełącznikach SDN w czasie rzeczywistym. Podobne rozwiązanie o nazwie ExPetr zaproponowano w [82] dla złośliwego oprogramowania typu ransomware. Składa się ono z trzech mechanizmów bezpieczeństwa zaimplementowanych jako moduły w kontrolerze SDN. Mechanizmy te obejmują: blokowanie portów, inspekcję pakietów Server Message Block (SMB) oraz analizę wiadomości HTTP. W przypadku wykrycia złośliwej aktywności kontroler komunikuje się z przełącznikami SDN za pomocą protokołu OpenFlow i instaluje odpowiednie wpisy w ich tablicach przepływów. W szczególności kontroler blokuje maszyny uznane za zainfekowane, monitorując i reagując w czasie rzeczywistym na generowany przez nie ruch sieciowy. Wyniki eksperymentalne wskazują na wysoką skuteczność przedstawionego rozwiązania, ponieważ we wszystkich przypadkach złośliwe działanie zostało wykryte i zablokowane wystarczająco szybko, aby zapobiec rozprzestrzenianiu się ExPetr na pozostałe maszyny. W innej pracy [83] o podobnym charakterze autorzy przeprowadzili analizę ransomware o nazwie BadRabbit i przedstawili sposób walki z nim w sieciach opartych na SDN. Rozwiązanie składa się z pięciu modułów: głębokiej inspekcji pakietów, wykrywania skanowania ARP, analizy nagłówków pakietów, honeypota oraz monitorowania pakietów SMB. Autorzy ocenili wpływ swojego rozwiązania na wprowadzane opóźnienia oraz jego skuteczność w wykrywaniu dwóch innych rodzin ransomware: WannaCry oraz NotPetya. Stwierdzono, że inspekcja pakietów powodowała większe opóźnienia w sieci oraz potrzebowała dwukrotnie więcej czasu procesora niż pozostałe moduły. Udowodniono też, że dwa z komponentów (analiza nagłówków pakietów i honeypot) są w stanie wykryć także inne rodzaje rodzin ransomware.

W literaturze można także znaleźć próby wykorzystania sztucznej inteligencji wraz z technologią SDN do wykrywania i przeciwdziałania cyberzagrożeniom. W [84] autorzy przedstawiają wysoce skalowalny, hybrydowy framework do wykrywania infekcji złośliwym oprogramowaniem w ekosystemie Internet of Things (IoT). Ocena systemu za pomocą dostępnych publicznie zbiorów danych wykazała, że odznacza się ono skutecznością detekcji wynoszącą 99.83%, a jednocześnie nie wymaga dużych zasobów obliczeniowych.

Skanowanie sieciowe

Technologia SDN jest bardzo często wykorzystywana do realizacji proaktywnych technik zabezpieczeń takich jak Moving Target Defense (MTD). Jest ona szczególnie przydatna w wykrywaniu i przeciwdziałaniu skanowania sieciowego, ze względu na możliwość wprowadzania szybkich (programowalnych) zmian w infrastrukturze sieciowej.

Autorzy w [85] proponują system Reconnaissance Deception System (RDS) do obrony przed atakami zwiadowczymi. RDS blokuje rekonesans sieci poprzez opóźnianie akcji skanowania przez atakujących oraz udostępnianie nieprawdziwych informacji. Zarządza on także wirtualnymi widokami sieci dla każdego hosta i podejmuje odpowiednie działania obronne. Według przeprowadzonej przez autorów oceny rozwiązania, wydłuża ono 115 razy czas potrzebny do zidentyfikowania podatnych na ataki punktów końcowych w sieci.

Z kolei w pracy [86] zaprezentowano adaptacyjny system o nazwie ACyDS. Dostarcza on wirtualne widoki sieciowe dla każdego hosta w sieci. Pole TTL w nagłówkach protokołu IPv4 wykorzystywane jest do zarządzania wirtualnymi widokami, a systemy honeypot są rozmieszczane losowo w każdym wirtualnym widoku. Autorzy twierdzą, że proponowane podejście zniechęca do rekonesansu, zapobiega kolizjom oraz zwiększa prawdopodobieństwo i pewność wykrycia atakujących, nie przedstawiono jednak wyników przeprowadzonych eksperymentów. Warto zauważyć, że w artykule [87] zaproponowano podobne rozwiązanie, nazwane Customized Information Networks for Deception and Attack Mitigation (CINDAM), które zapobiega atakom poprzez tworzenie widoków dla każdego hosta. Według autorów proponowany system może utrudnić atakowanie sieci bez wpływania na operacje sieciowe i modyfikowanie oprogramowania klienta lub serwera. Jednak autorzy, jak w poprzednim przypadku, nie przedstawiają wyników oceny zaproponowanego systemu.

W [60] przedstawiono system OpenFlow Random Host Mutation (OF-RHM) oparty na zmianie adresów IP z dużą nieprzewidywalnością i częstotliwością. Kontroler SDN odpowiada za częstą modyfikację i przypisanie losowego wirtualnego adresu IP do każdego hosta. Prawdziwy adres IP nie jest zmieniany, a operacja ta jest przezroczysta dla hostów końcowych. Hosty te są osiągalne poprzez wirtualne adresy IP pozyskane z serwera DNS. Tylko uprawnione urządzenia końcowe mogą korzystać z prawdziwych adresów IP. Według przedstawionych przez autorów wyników badań, OF-RHM może uniemożliwić zbieranie informacji przez atakującego wykorzystującego skanery w 99% przypadków oraz pozwala na ochronę hostów przed robakami sieciowymi w 90%. Z kolei w artykule [88] zaproponowano inne techniki, które umożliwiają powiązanie hosta z jego prawdziwym IP na podstawie tożsamości źródła i czasu. Randomizacja tożsamości źródła i czasu powoduje zaprezentowanie nieprawdziwego widoku sieci atakującym. Wprowadzone przez autorów metryki (odstraszanie, zwodzenie i wykrywalność) umożliwiły przeprowadzenie analizy teoretycznej i eksperymentalnej, według których proponowane rozwiązanie jest skuteczne w zwalczaniu zagrożeń związanych z rekonesansem sieciowym.

Następnie autorzy w [89] proponują rozwiązanie Self-adaptive End-point Hopping Technique (SEHT) z wykorzystaniem MTD. Proponowana technika składa się z kontrolera randomizacji, przełączników przeskoków oraz komponentów przeskoków końcowych zarządzających ramkami Ethernet za pomocą sterownika TAP w systemie Linuks. Kontroler randomizacji monitoruje nieudane żądania sieciowe, wykrywa typ strategii skanowania sieci i współpracuje z przełącznikami przeskoków i komponentami przeskoków końcowych przez predefiniowany

okres czasu w celu implementacji wybranej strategii. Według przeprowadzonych przez autorów analiz teoretycznych i symulacji, SEHT jest w stanie udaremnić 90% różnych rodzajów ataków skanujących.

W pracy [90] autorzy prezentują inne wykorzystanie rozwiązania MTD w celu uniemożliwienia wykonania skanowania sieciowego, odznaczające się wysoką skalowalnością. Ocenili oni także zaproponowaną metodę, używając wielu parametrów, takich jak: nieprzewidywalność, okresowość czy unikalność. Twierdzą oni, że proponowana technika zapewnia kluczowe właściwości bezpieczeństwa i pozwala odróżnić ruch inicjowany przez atakujących od poprawnego ruchu. Nie przedstawiono jednak wyników tych badań. Zaprezentowano natomiast wpływ zaproponowanego rozwiązania na wprowadzane opóźnienie, które wynosi średnio 20ms.

Autorzy w [91] proponują metodę obrony sieciowej, opartą na losowej mutacji nazw domen i adresów (RDAM). Nazwy domen wszystkich hostów są okresowo zmieniane przy użyciu dynamicznego generowania nazw domen, co zwiększa przestrzeń skanowania atakującego i zmniejsza prawdopodobieństwo dotarcia do celów przy użyciu listy zapytań DNS. Klienci muszą się uwierzytelnić w specjalnej usłudze, która kontroluje dostęp do nazw i adresów. Proponowana metoda może udaremnić ataki skanujące i propagację robaków komputerowych. Autorzy przeprowadzili ewaluację swojego systemu opartą na symulacji i, w zależności od zastosowanego sposobu skanowania, skuteczność rozwiązania waha się od 80% do 99,7%. Zaprezentowano także wyniki badań wprowadzanych opóźnień, które w najgorszym przypadku wynoszą 26,72ms.

System ochrony przed cyberrekonesansem w postaci fingerprinting'u został przedstawiony w artykule [92]. Celem tego ataku jest uzyskanie informacji o hostach docelowych w celu poczynienia przygotowań do przyszłych ataków. Autorzy proponują opartą na SDN metodę fingerprint hopping (FPH), która także wykorzystuje podejście MTD w celu zmylenia przeciwnika. Przeprowadzone badania (z wykorzystaniem emulatora Mininet) wykazały, że rozwiązanie jest w stanie zatrzymać wszystkie próby przeprowadzenia fingerprinting'u systemów operacyjnych i ich wersji.

Następnie w pracy [93] zaproponowano metodę obrony przed zagrożeniami skanowania hostów/portów oraz atakami typu fingerprint systemu operacyjnego lub usług. Każdy pakiet jest buforowany w celu zapewnienia spójnego zachowania, następnie zwracane są losowe odpowiedzi na żądania oraz dodawane są losowe opóźnienia w nawiązywaniu połączenia TCP. Autorzy w przeprowadzonych badaniach udowadniają, że proponowane metody skutkują dużym narzutem opóźnienia dla atakujących próbujących odkryć usługi oraz 100%-200% narzutem dla skanowania portów i ataków fingerprinting systemu operacyjnego.

Z kolei w artykule [94] zaproponowano rozwiązanie Intrusion Prevention System (IPS) adresujące atak skanowania portów z wykorzystaniem danych i statystyk z przełączników SDN. Opracowano i zaimplementowano nieinwazyjną i lekką metodę, która charakteryzuje się niskim narzutem sieciowym oraz niskim zużyciem pamięci i mocy obliczeniowej. Uzyskane

wyniki oparte na symulacji pokazały, że proponowana metoda jest skuteczna w wykrywaniu i zapobieganiu atakom skanowania portów (brak błędów pierwszego i drugiego rodzaju).

Mechanizm opisany w [95] także wykorzystuje MTD do ochrony sieci przed skanerami. Może on działać w połączeniu z systemem IPS, nie wpływając na jego normalne zachowanie. W tym celu kontroler SDN zmienia nagłówki pakietów przechodzących przez przełączniki, wykorzystując wirtualne adresy IP, podczas gdy IPS kontynuuje pracę, monitorując rzeczywiste adresy IP urządzeń. Przeprowadzone badania (z wykorzystaniem emulatora Mininet) pokazują, że rozwiązanie może efektywnie powstrzymywać skanowanie sieciowe, jednak może jednocześnie powodować zmniejszenie przepływności sieci.

Denial of Service

Technologia SDN udostępnia globalny widok sieci, co znacznie ułatwia wykrywanie zmasowanych i rozproszonych ataków typu odmowy usługi – (D)DoS. Specjalistyczne aplikacje są w stanie szybko zareagować na pierwsze oznaki przeprowadzanego ataku i wprowadzić odpowiednie zmiany chroniące infrastrukturę sieciową.

W celu obrony przed atakiem DoS, autorzy w [96] proponują metodę Random Route Mutation (RRM), która łączy teorię gier i optymalizację spełniania ograniczeń w celu określenia właściwej strategii przy jednoczesnym zapewnieniu wymogów bezpieczeństwa, wydajności oraz Quality of Service (QoS). RRM modeluje wybór trasy jako optymalizację zadowolenia z ograniczenia i formalizuje go przy użyciu Satisfiability Modulo Theory (SMT) w celu identyfikacji efektywnych tras. Autorzy podają algorytmy pozwalające na wdrożenie RRM zarówno w sieciach konwencjonalnych jak i definiowanych programowo. Przeprowadzone symulacje udowodniły, że RRM może chronić do 90% pakietów przed atakami napastników, w porównaniu do schematów routingu jednościeżkowego. Z kolei w pracy [97] autorzy proponują model o nazwie Agile Virtual Network Framework służący do obrony przed atakami DDoS poprzez proaktywną zmianę infrastruktury krytycznych zasobów. Wykorzystuje on sieci wirtualne do dynamicznej realokacji zasobów sieciowych i migruje infrastrukturę do nowych zasobów przy zachowaniu integralności sieci. Przeprowadzone symulacje wykazały skuteczność przywracania obniżonej przepustowości (80%) spowodowanej atakiem DDoS poprzez migrację do bezpiecznego miejsca w ciągu zaledwie kilku sekund.

W artykule [98] zaproponowano protokół i aplikację przeciwko atakom DoS typu SYN flood, nazwaną OPERETTA (OPEnflow based REmedy to TCP SYN flood Attacks). Jest to rozwiązanie zaimplementowane w kontrolerze SDN, które zarządza przychodzącymi segmentami TCP SYN oraz odrzuca fałszywe żądania połączenia. W wyniku przeprowadzonych symulacji stwierdzono, że zużycie procesora i pamięci przez OPERETTA jest niskie i jednocześnie pozwala na ochronę przed atakami TCP SYN flood.

Następnie w pracy [99] przedstawiono architekturę wykrywania i zapobiegania anomalii, która wykorzystuje możliwości OpenFlow. Anomalie są wykrywane przy użyciu wartości

entropii obliczanych na podstawie ruchu sieciowego, zbieranych w 30-sekundowych odstępach czasu. Podejście to polega na zarządzaniu białą listą i blokowaniu innego ruchu sieciowego. OpenFlow jest wykorzystywany do zapobiegania atakom poprzez modyfikacje tablicy przepływów. Autorzy opisują swoje eksperymenty dla zagrożeń typu DDoS, robaków i skanowania portów, które potwierdzają skuteczność rozwiązania. Z kolei w [100] zaproponowano podejście do projektowania zabezpieczeń SDN w celu zapobiegania atakom DDoS i śledzenia źródła ataku. Proponowane podejście wprowadza nową płaszczyznę SDN, płaszczyznę bezpieczeństwa, jako dodatek do płaszczyzny infrastruktury i równolegle do płaszczyzny zarządzania SDN. Autorzy poprzez przeprowadzone symulacje pokazują, że proponowany system wymusza różne poziomy bezpieczeństwa zdefiniowane przez użytkownika w czasie rzeczywistym przy niskim narzucie i minimalnej konfiguracji. Średni czas reakcji od rozpoczęcia ataku do całkowitego zablokowania wynosi 3s.

Transparent Intrusion Detection System (TIDS) to rozwiązanie zaproponowane w [101] do wykrywania i zapobiegania atakom DoS. Po odizolowaniu źródła i celu ataku, wysyłana jest wiadomość Flow Modification Request zawierająca listę adresów, które mają zostać zablokowane na określony czas. Na podstawie przeprowadzonych symulacji autorzy wskazują, że proponowany system jest przezroczysty dla urządzeń zewnętrznych, a nieprawidłowe numery sekwencji TCP, ataki SYN flood i inne podobne zagrożenia mogą być łatwo wykryte bez dodatkowego zużycia zasobów.

Z kolei w [102] wprowadzono aplikację SDN o nazwie Dossy w celu łagodzenia ataków DoS wraz z podszyciem adresów IP/MAC i innymi atakami sieciowymi. Aplikacja śledzi węzły w sieci. Początkowo, kontroler SDN zbiera informacje o adresach MAC i IP wszystkich hostów i łączy je razem w tablicy hash. Każdy komunikat Packet_In jest przetwarzany, a adresy IP/MAC są analizowane z uwzględnieniem zarządzanej tablicy hash. Dossy zmniejsza przepustowość kontrolera o 7,76% i zwiększa opóźnienie o 7,37% w porównaniu do domyślnego przełącznika uczącego.

FlowFence to zaproponowany w [103] lekki i szybki system wykrywania i łagodzenia ataków DoS. Kontroler SDN, aby zapobiec przeciążeniu węzłów, koordynuje przydział pasma dla kontrolowanych łączy i ogranicza szybkość transmisji przepływu wzdłuż ścieżki. Przepływy przekraczające przydzielone limity mają nałożoną karę administracyjną. FlowFence pozwala na uniknięcie przeciążenia zasobów sieciowych przez użytkowników przy jednoczesnym dodaniu niewielkiego narzutu. Czas potrzebny na wykrycie i zmitygowanie ataku DoS wynosi 7s. Inną propozycją, która wykorzystuje ograniczanie przepływów jest SDNManager zaproponowany w [104], gdzie informacje o przepływach są stale monitorowane w sieci, a zapotrzebowanie na pasmo jest szacowane na przyszłość. Podczas szacowania przepustowości przyjęto model autoregresji z heteroskedastycznością warunkową (ARCH), powszechnie stosowany w ekonometrii. Za pomocą tego modelu mierzone są zmiany w szeregach czasowych oraz szacowana jest zmienność całkowitej przepustowości sieci. Zbadano także wpływ ataku DoS o różnym nasileniu na czas odpowiedzi kontrolera SDN i porównano je z innymi dostępnymi

rozwiązaniami. Z badań wynika, że SDNManager wprowadza dodatkowe stałe opóźnienie rzędu 0,3ms, podczas gdy inne rozwiązania w czasie ataku powodują zwiększenie czasu odpowiedzi kontrolera do nawet 2ms.

Z kolei autorzy artykułu [105] proponują opartą na chmurze architekturę łagodzenia ataków DDoS o nazwie DaMask, która umożliwia wykrywanie ataków oraz szybką reakcję na nie. DaMask składa się z DaMask-D (system wykrywania ataków sieciowych) oraz DaMask-M (moduł reagowania na ataki). Jeśli DaMask-M otrzyma alert, moduł wyszukuje dla niego środek zaradczy. Domyślnym środkiem zaradczym jest odrzucenie pakietu. DaMask-M posiada zestaw wspólnych interfejsów API, dzięki czemu różne środki zaradcze mogą być dostosowane do różnych typów ataków DDoS. Trzy podstawowe środki zaradcze są zdefiniowane w następujący sposób: prześlij dalej, odrzuć i zmodyfikuj. Jeśli środek zaradczy zostanie wybrany, DaMask-M przesyła reguły przepływu do przełącznika poprzez kontroler SDN. W wyniku przeprowadzonych symulacji osiągnięto skuteczność wykrywania ataków DDoS na poziomie 89,3%.

Przełączniki SDN przechowują kilka różnych statystyk dotyczących liczby pakietów i przesłanych bajtów. Statystyki te mogą być stale zbierane przez kontroler i przetwarzane w celu wykrywania ataków DDoS. Przykładowo niektóre badania przeprowadzają detekcję DDoS poprzez analizę rozkładu liczby pakietów wejściowych [106] lub stosunku pakietów wejściowych do przesłanych [107]. Takie metody są szczególnie skuteczne w przypadku ataków DDoS pośrednio skierowanych na sam kontroler. Z kolei rozwiązanie SoftGuard [108] dedykowane dla ataków DoS o niskiej szybkości, działa poprzez dodanie reguł monitorowania do każdego przełącznika, aby okresowo mierzyć całkowitą ilość danych pasujących do tych reguł oraz stale monitorować dostępną przepustowość. Gdy nastąpi znaczący spadek przepustowości, rozpoczyna się faza identyfikacji atakującego. Podczas fazy wykrywania, interfejsy przełączników powyżej średniej przepustowości są oznaczane, jeśli są podobne do wcześniej ustalonego spadku przepustowości. Jako metodę walki z atakującymi autorzy zaproponowali zmniejszenie pasma lub odrzucanie ich ruchu. Przeprowadzone eksperymenty potwierdzają skuteczność zaproponowanej metody, która, wprowadzając dodatkowy 1% obciążenia CPU, wykrywa ponad 90% ataków.

Niektóre badania wprowadzają także systemy obronne do wykrywania i łagodzenia ataków typu link flooding (LFA). LFA wykorzystuje legalnie wyglądające przepływy o niskiej gęstości do zalewania grupy wybranych łączy i jest trudny do odróżnienia przez tradycyjne metody zabezpieczeń. W [109] zaproponowano metodę proaktywnego lokalizowania przeciążonych łączy nazwaną Woodpecker. System wykorzystuje globalny algorytm oceny, aby określić, czy w danej chwili przeprowadzany jest atak LFA. Woodpecker stosuje zrównoważy ruch i eliminuje wąskie gardła routingu, które mogą być wykorzystane przez przeciwnika. Wyniki przeprowadzonych symulacji pokazują, że wykorzystanie przepustowości łączy atakowanych przez LFA może być zmniejszone o około 50%, przy jednoczesnym zmniejszeniu utraty pakietów. Inna propozycja o nazwie LFADefender [110] analizuje sieć w celu zidentyfikowania

potencjalnych łączy docelowych o dużej gęstości przepływu, stale monitoruje przeciążenie łączy, przekierowuje ruch z dala od przeciążonych łączy w celu złagodzenia skutków trwającego ataku oraz blokuje złośliwy ruch. Każde z powyższych zadań jest obsługiwane przez oddzielny moduł na szczycie kontrolera. Architekturę o nazwie RADAR do wykrywania i mitygowania ataków DDoS z wykorzystaniem adaptacyjnej analizy korelacji przedstawiono z kolei w [111]. Rozwiązanie to posiada moduły detekcji dla ataków typu link flooding, SYN flooding oraz DNS amplification. Analizuje także ruch sieciowy przechwytywany w różnych miejscach, ogranicza go, jeśli zostanie zaklasyfikowany jako podejrzany, oraz odrzuca pakiety przy użyciu opartej na portach techniki max-min fairness. Autorzy przeprowadzili 12 różnych symulacji, które potwierdzają skuteczność wykrywania ataków w 80-100%.

Głębokie uczenie zostało zaproponowane do wykrywania ataków DDoS w środowiskach SDN w pracy [112]. Ponieważ kontroler ma pełny wgląd w topologię sieciową i może analizować każdy pakiet, który nie należy do wcześniej znanego przepływu, jest on zatem w stanie przetwarzać pakiety w celu wyodrębnienia cech z pól nagłówków i uruchomić moduł głębokiego uczenia w celu wykrycia ataków DDoS. Kontroler może wtedy szybko zareagować i zainstalować reguły przepływu w celu odrzucenia podejrzanych pakietów na przełącznikach. Według przeprowadzonej symulacji skuteczność wykrywania ataków DDoS osiąga 98%. Natomiast w pracy [113] przeanalizowano rozkład czasowy trafień w tablicy przepływów i scharakteryzowano zachowanie atakującego przy użyciu sieci neuronowej. Podejście takie jest możliwe, jednak autorzy stwierdzają, że może to mieć negatywny wpływ na dostępność świadczonych usług. W wyniku symulacji, w zależności od użytych parametrów systemu, system wykrywa poprawnie 11-100% ataków.

W ostatnich latach także technologia blockchain została wykorzystana do obrony przed atakami DDoS. Autorzy pracy [114] proponują CoChain-Sc – rozwiązanie służące wykrywaniu ataków DDoS w ramach ruchu międzydomenowego i wewnątrz domeny systemu autonomicznego (AS). Wykorzystane są w tym celu uczenie maszynowe oparte na entropii oraz technologia blockchain Ethereum (tzw. inteligentne kontrakty). Według przeprowadzonych symulacji, system w 100% wykrywał przeprowadzone ataki DDoS, przy jednoczesnym 23% błędzie pierwszego rodzaju. Ponadto inna publikacja [115] przedstawia koncepcję mitygowania ataków DDoS, w której adresy IP na czarnej liście są zarządzane za pomocą inteligentnych kontraktów blockchain. Nie przedstawiono jednak badań potwierdzających słuszność zaproponowanej metody.

Rozwiązania przeciwdziałające atakom DDoS są także implementowane w warstwie infrastruktury architektury SDN w celu osiągnięcia lepszej wydajności w porównaniu z rozwiązaniami opartymi na kontrolerach. Przykładowo, FastFlex [116] opiera się na koncepcji multimodalnej warstwy infrastruktury w celu złagodzenia ataków na przełączniki SDN, eliminując w ten sposób narzut pasma i opóźnienie związane z komunikacją do kontrolera podczas ataku. Przeprowadzone symulacje pokazują, że FastFlex pozwala zachować 80% przepustowości łącza w przypadku ataku DDoS. Z kolei język Poseidon zaprezentowany w [117]

oraz system wykonawczy i komponent orkiestracji oferują razem adaptowalne rozwiązanie, które może wyrażać szeroką gamę polityk obrony przed DDoS i reagować na ataki w sposób efektywny. Autorzy przeprowadzili badania i symulacje, według których Poseidon odznacza się tylko 1% błędu pierwszego rodzaju i wprowadza dwukrotnie mniejsze opóźnienie niż inne systemy tego typu.

Podsłuch sieciowy

Szybkie wprowadzanie zmian w infrastrukturze SDN może zostać wykorzystane, aby utrudnić atakującym możliwość skutecznego podsłuchiwania komunikacji sieciowej.

W [118] autorzy proponują oparte na SDN podejście Double Hopping Communication (DHC) do walki ze snifferami. Przedstawione podejście dynamicznie zmienia adresy IP, numery portów w pakietach, oraz ścieżki routingu. Ruch jest rozdzielany na wiele przepływów i przesyłany różnymi ścieżkami. Dodatkowo dane z wielu węzłów są łączone tak, aby atakujący nie mogli odtworzyć danych komunikacyjnych. Autorzy twierdzą, że DHC utrudnia używanie snifferów i powoduje trudności w pozyskiwaniu pełnych danych komunikacyjnych. W przeprowadzonych symulacjach, prawdopodobieństwo przechwycenia całej komunikacji przez podsłuchujący host jest praktycznie równe zeru, natomiast system zwiększa opóźnienie w sieci o 7%.

Z kolei Random Route Mutation (RRM) to proaktywna technika zaproponowana w [119], wykorzystująca SMT w sieci nakładkowej. RRM, w celu obrony przed podsłuchem, umożliwia losową zmianę tras wielu przepływów w sieci jednocześnie. Według przeprowadzonych symulacji, proponowana technika pozwala zmniejszyć odsetek pakietów podsłuchiwanych do mniej niż 10%. Autorzy w [120] również używają RRM do opracowania architektury opartej na SDN w celu przeciwdziałania rekonesansowi w sieci. Wykorzystują oni charakterystykę ruchu sieciowego do wykrywania ataków przy użyciu macierzy entropii. RRM jest wyzwalany w zależności od wyniku wykrywania anomalii sieciowych lub po upływie predefiniowanego okresu. Generowanie losowej ścieżki routingu odbywa się przy użyciu ulepszonego algorytmu kolonii mrówek jako problemu plecakowego. Proponowana metoda zwiększa trudność rozpoznania i podsłuchu oraz zmniejsza wpływ ataków DoS. Autorzy na podstawie symulacji twierdzą, że ich podejście jest w stanie obronić 90% pakietów przed atakami.

W pracy [121] autorzy proponują mechanizm, który może zapobiegać atakom Man in the Middle (MitM) i podsłuchiwaniu ruchu sieciowego poprzez zastosowanie nowej metody translacji adresów sieciowych. Rozwiązanie to opiera się na wielu działaniach wykonywanych wewnątrz przełącznika SDN w celu wzmocnienia poziomu bezpieczeństwa zgodnie z żądaniami użytkownika. Autorzy nie przedstawili jednak wyników badań zapobiegania podsłuchom, a jedynie jaki wpływ ma zaproponowany mechanizm na opóźnienia w sieci. Natomiast w artykule [122] opisano problem ochrony prywatnych danych w komunikacji sieciowej przed podsłuchiwaniem oraz zaproponowano metodę MTD do udaremnienia tego typu ataków przez wykorzystanie mechanizmu Protocol-Oblivious Forwarding (POF). Rozwią-

zanie to, w celu ukrycia poufnych informacji, bazuje na pełnej randomizacji stosu protokołów, pakowania wiadomości oraz ścieżki routingu. Autorzy twierdzą, że proponowane strategie zwiększają trudność reorganizacji treści wiadomości dla podsłuchiwacza i zmniejszają prawdopodobieństwo przechwycenia pakietów wiadomości sesyjnych.

Z kolei moduł warstwy aplikacji SDN zaprezentowany w [123] uniemożliwia podsłuchującemu pełne przechwycenie przepływów komunikacyjnych pomiędzy komponentami systemu Supervisory Control And Data Acquisition (SCADA). Mechanizm ten wykorzystuje dynamiczny i statyczny routing wielościeżkowy do częstej modyfikacji tras komunikacyjnych pomiędzy urządzeniami SCADA, zwiększa prywatność informacji w tego typu sieciach, a także utrudnia atakującemu przechwytywanie przepływów pomiędzy tymi urządzeniami. Autorzy na podstawie symulacji przeprowadzili ocenę i stwierdzili, że dynamiczne reguły o krótszym czasie życia utrudniają atakującemu przechwytywanie przepływów, ale zwiększają narzut zarządzania w kontrolerze. Atakujący w najgorszym przypadku będzie w stanie przechwycić 75% ruchu sieciowego.

Tab. 3.1 podsumowuje opisane w tej części rozprawy publikacje oraz porównuje je z przedstawionymi w niniejszej rozprawie pracami [20]–[22] (pogrubiono prace wchodzące w skład rozprawy).

Tabela 3.1. Publikacje o aplikacjach podnoszących bezpieczeństwo w sieciach SDN.

Praca	Proponowana metodyka, technika lub procedura	Metoda ewaluacji	Sposób ochrony
Złośliwe oprogramowanie			
[20]	**Analiza sekwencji komunikatów ruchu HTTP i ich rozmiarów (dla CryptoWall oraz Locky). Blokada ruchu do wykrytego serwera proxy.**	**Eksperyment**	**Detekcja, mitygacja**
[77]	Dynamiczne tworzenie czarnej listy serwerów proxy na podstawie żądań DNS. Blokada ruchu z zainfekowanego hosta.	Eksperyment	Detekcja, mitygacja
[78]	Monitorowanie ruchu sieciowego z wykorzystaniem uczenia maszynowego.	Symulacja	Detekcja
[79]	Dynamiczne rekonfigurowanie środowiska sieciowego i uruchamianie malware w sandbox.	Eksperyment	Mitygacja
[80]	Dopasowanie sygnatur i wykrywanie anomalii z wykorzystaniem statystyk. Podejrzany ruch jest kierowany i obserwowany w dynamicznej i wirtualnej sieci honeypot.	Symulacja	Detekcja, mitygacja
[81]	Monitorowanie ruchu SMB. Blokowanie zainfekowanych hostów.	Eksperyment	Detekcja, mitygacja
[82]	Monitorowanie ruchu HTTP oraz SMB. Blokowanie portów zainfekowanych maszyn (dla EXPetr).	Eksperyment	Detekcja, mitygacja
[83]	Głęboka inspekcja pakietów, wykrywanie skanowania ARP, analiza nagłówków pakietów, honeypot, analiza pakietów SMB. Blokowanie ruchu z zainfekowanego hosta.	Eksperyment	Detekcja, mitygacja
[84]	Algorytm detekcji malware oparty na głębokim uczeniu.	Eksperyment	Detekcja
Skanowanie sieciowe			
[21]	**Użycie przesuwnego okna do zliczania rozpoczętych połączeń TCP. Przekroczenie ustawionego progu oznacza skanowanie.**	**Eksperyment**	**Detekcja, mitygacja**

[85]	Inny widok sieci dla każdego hosta opóźnia proces skanowania.	Symulacja	Detekcja, zapobieganie
[86]	Pole TTL w nagłówku IPv4 użyte do zarządzania wirtualnymi widokami sieciowymi dostarczanymi do każdego hosta.	Eksperyment Brak wyników	Detekcja, zapobieganie
[87]	Widok sieci jest zmieniany dla poszczególnych hostów. Hosty, które wysyłają pakiety na adresy IP honeypot'ów są uznawane za atakujących.	Symulacja	Detekcja, mitygacja, zapobieganie
[60]	Częste przydzielanie losowego, wirtualnego adresu IP. Hosty są osiągalne poprzez wirtualne adresy IP uzyskane z serwera DNS.	Symulacja	Zapobieganie
[88]	Prezentowanie nieprawdziwego widoku sieci atakującym na podstawie randomizacji tożsamości źródła i czasu.	Symulacja	Zapobieganie
[89]	Periodyczna zamiana adresów IP i portów, monitorowanie nieudanych żądań sieciowych.	Symulacja	Detekcja, mitygacja, zapobieganie
[90]	Klienci muszą nawiązać kontakt z serwerem DNS, aby dotrzeć do wirtualnego adresu IP właściwego serwera.	Eksperyment Brak wyników	Zapobieganie
[91]	Zastosowanie losowej mutacji nazw domen i adresów. Klienci muszą się uwierzytelnić w specjalnej usłudze, która zarządza dostępem do wybranych nazw domenowych i adresów.	Symulacja	Zapobieganie
[92]	IDS wykrywa sygnatury fingerprinting'u. Fingerpringing i obrona przed nim są modelowane jako gra.	Symulacja	Detekcja, mitygacja
[93]	Wysyłanie fałszywych informacji na temat otwartych portów i systemów operacyjnych do atakującego.	Eksperyment	Zapobieganie
[94]	Wykorzystanie danych i statystyk z przełączników SDN do wykrywania skanowania.	Symulacja	Zapobieganie
[95]	Zmiana adresów IP w nagłówkach pakietów przechodzących przez przełączniki oraz wykorzystanie IPS monitorującego rzeczywiste IP.	Symulacja	Zapobieganie

Denial of Service			
[21]	**Porównywanie czasu pomiędzy kolejnymi wiadomościami DHCP z wyznaczonymi doświadczalnie modelami popularnych systemów operacyjnych. Blokowanie wiadomości DHCP, które są wysyłane w nieodpowiednich odstępach czasu.**	**Eksperyment**	**Detekcja, mitygacja**
[96]	Losowa trasa jest określana przy użyciu teorii gier w celu określenia optymalnej strategii dla ataku. Przepływy są przekierowywane przy użyciu wybranych ścieżek routingu.	Symulacja	Detekcja, mitygacja
[97]	Dynamiczna realokacja zasobów sieciowych.	Symulacja	Mitygacja
[98]	Pakiety SYN są wysyłane do kontrolera, który odpowiada pakietem SYN-ACK do klienta bez wysyłania do hosta docelowego.	Symulacja	Zapobieganie
[99]	Wykrywanie anomalii za pomocą algorytmów używających entropii. Dodawanie przepływów z wysokim priorytetem w celu zatrzymania złośliwego ruchu.	Eksperyment	Detekcja, mitygacja
[100]	Przełączniki wysyłają pakiety do silnika detekcji, który instaluje reguły na kontrolerze blokujące atak.	Symulacja	Detekcja, mitygacja, zapobieganie
[101]	Analiza przepływów w celu wykrycia anomalii. Blokowanie listy adresów biorących udział w ataku na określony czas.	Symulacja	Detekcja, mitygacja
[102]	Analiza przepływów i wiadomości Packet_In w celu wykrycia ataku DoS. Blokowanie adresów atakujących.	Symulacja	Detekcja, mitygacja
[103]	Wykrywanie przeciążenia ruchu przez przełączniki SDN. Sterowanie przez kontroler pasmem dla przepływów.	Eksperyment	Detekcja, mitygacja
[104]	Monitorowanie ruchu i szacowanie zapotrzebowania na przepływność. Karanie przepływów przekraczających szacunki.	Eksperyment	Zapobieganie

[105]	Wykrywanie anomalii na podstawie teorii grafów. Złośliwy ruch może zostać odrzucony, zmodyfikowany lub przesłany dalej.	Symulacja	Detekcja, mitygacja
[106]	Odrzucanie ruchu na podstawie ustawionych progów wiadomości Packet_In.	Eksperyment	Detekcja, mitygacja
[107]	Monitorowanie stosunku wiadomości Packet_In do wysłanych bajtów. Blokowanie ruchu, gdy odbiega on od normy.	Symulacja	Detekcja, mitygacja
[108]	Monitorowanie przełączników przez specjalnie zainstalowane reguły. Zmniejszanie pasma lub odrzucanie podejrzanego ruchu.	Eksperyment	Detekcja, mitygacja
[109]	Przełączniki informują o przeciążeniu. Kontroler przekierowuje ruch związany z atakiem do serwerów honeypot, wyrównuje przepływy na przeciążonych łączach za pomocą alternatywnych ścieżek lub usuwa pakiety przy użyciu czarnej listy.	Symulacja	Detekcja, mitygacja
[110]	Monitorowanie przeciążonych łączy, przekierowywanie ruchu oraz blokowanie atakujących.	Symulacja	Detekcja, mitygacja
[111]	Ograniczanie podejrzanego ruchu i odrzucanie ruchu pochodzącego z ataków przy pomocy opartej na portach techniki max-min fairness.	Symulacja	Detekcja, mitygacja
[112]	Klasyfikacja ruchu wykorzystująca głębokie uczenie do wykrywania ataków DdoS oraz instalacja reguł blokujących atak.	Symulacja	Detekcja, mitygacja
[113]	Analiza rozkładu czasowego trafień w tablicy przepływów i wykrywanie napastnika przy użyciu sieci neuronowej.	Symulacja	Detekcja, mitygacja
[114]	Analiza anomalii w ruchu między domenami i wewnątrz domeny za pomocą uczenia maszynowego i technologii blockchain.	Symulacja	Detekcja, mitygacja
[115]	Zarządzanie czarna listą IP za pomocą inteligentnych kontraktów blockchain.	Brak	Mitygacja
[116]	Wyzwalanie alarmu o ataku w płaszczyźnie infrastruktury SDN powoduje przekierowywanie ruchu na inne łącza.	Symulacja	Detekcja, mitygacja

[117]	Wprowadzanie modularnych abstrakcji pozwalające operatorom na implementacje odpowiednich polityk.	Symulacja	Detekcja, mitygacja
	Podsłuch sieciowy		
[22]	**Pomiar czasów odpowiedzi hosta i wykorzystanie uczenia maszynowego do wykrycia podsłuchu.**	**Eksperyment**	**Detekcja, Mitygacja**
[118]	Dynamiczna zmiana tras pakietów, adresów i portów uniemożliwia podsłuchanie kompletnej komunikacji sieciowej.	Symulacja	Zapobieganie
[119]	Przekazywanie pakietów przez losowe i zmieniane okresowo ścieżki routing'u.	Symulacja	Zapobieganie
[120]	Przekazywanie pakietów przez losowe trasy wybierane na podstawie algorytmu kolonii mrówek.	Symulacja	Zapobieganie
[121]	Szyfrowanie, fragmentacja, powielanie i przestawianie pakietów w przełączniku.	Eksperyment	Zapobieganie
[122]	Losowa zamiana protokołów, adresów i portów.	Brak	Zapobieganie
[123]	Wielościeżkowy routing, każdy przełącznik przekazuje tylko część ruchu.	Symulacja	Zapobieganie

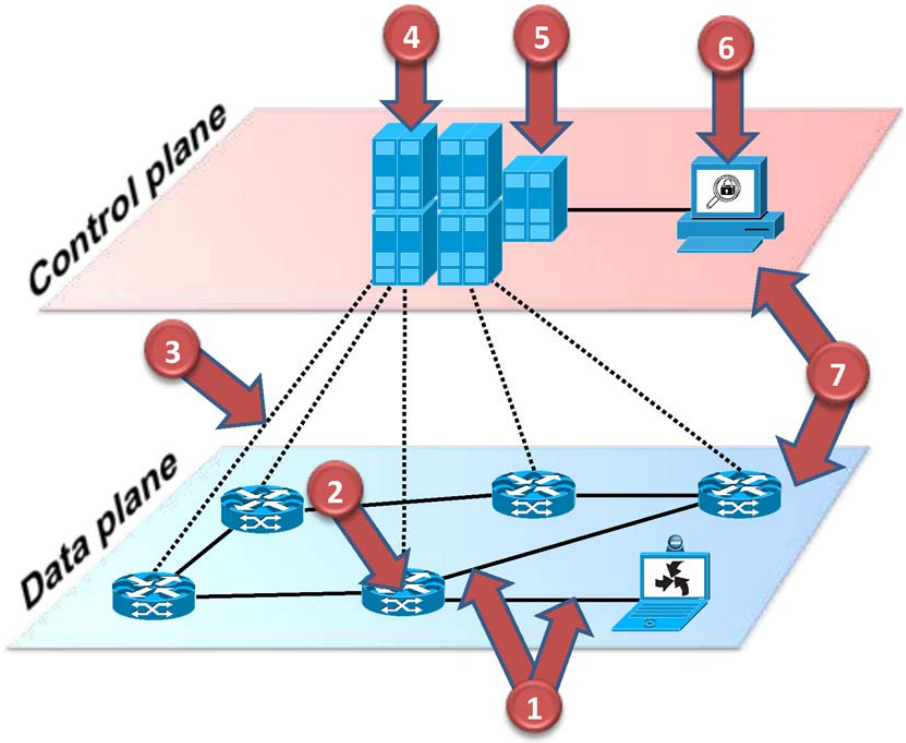
3.2. Bezpieczeństwo technologii SDN

Technologia SDN poprzez rozdzielenie płaszczyzny sterowania od płaszczyzny transportu, zmienia sposób zarządzania sieciami w centrach danych. Jednak implementując znaczne zmiany w sposobie funkcjonowania infrastruktury sieciowej, poza oczywistymi korzyściami, pojawiły się także sposoby wykorzystania przez cyberprzestępców wprowadzonych elementów w infrastrukturze jak i interakcji pomiędzy nimi w stosunku do sieci tradycyjnych. Uważa się, że bezpieczeństwo i niezawodność sieci SDN są ich najważniejszym aspektem, który musi zostać wzięty pod uwagę w najbliższej przyszłości ([124], [125], [126]).

Kilka wektorów ataków zostało zidentyfikowane zarówno w architekturze SDN [124] jak i w sieciach opartych na protokole OpenFlow ([127], [128], [129], [130], [131], [26], [18], [132]). Podczas gdy niektóre wektory ataków są typowe dla obecnie istniejących sieci, inne są bardziej specyficzne dla SDN, takie jak np. ataki na komunikację w płaszczyźnie sterowania i scentralizowane kontrolery. Warto wspomnieć, że większość wektorów ataku jest niezależna od technologii lub protokołu (np. OpenFlow), ponieważ reprezentują one zagrożenia na warstwach konceptualnych i architektonicznych samej sieci SDN.

Rys. 3.2 oraz Tab. 3.2 przedstawia siedem głównych zidentyfikowanych wektorów ataków w architekturze technologi SDN. Pierwszy wektor to sfałszowany lub zmanipulowany ruch w warstwie danych, który może być wykorzystany do ataku na urządzenia sieciowe i kontrolery. Drugi pozwala atakującemu wykorzystać luki w urządzeniach i w konsekwencji dokonywać zmian w sieci lub powodować jej wadliwe działanie. Pojedynczy przełącznik, przejęty przez atakującego, może zostać wykorzystany do odrzucania lub spowalniania ruchu sieciowego, kopiowania ruchu na portach (np. w celu kradzieży danych), czy wstrzykiwania fałszywego ruchu aby przeciążyć kontroler lub sąsiednie przełączniki. Wektory zagrożeń trzeci, czwarty i piąty wskazane w Tab. 3.2 są uważane za najbardziej niebezpieczne, ponieważ mogą zagrozić poprawnemu działaniu sieci. Ataki na ruch w warstwie zarządzania, kontrolery i aplikacje mogą w łatwy sposób zapewnić atakującemu kontrolę nad siecią. Przykładowo, wadliwy lub złośliwy kontroler czy aplikacja może zostać użyta, w celu kradzieży danych czy do przeprogramowania całej sieci. Szósty wektor zagrożeń związany jest z atakami na stacje administracyjne i z wykorzystaniem znajdujących się na nich potencjalnych podatności. Skompromitowana jednostka administracyjna podłączona bezpośrednio do sieci sterującej, zapewni atakującemu zasoby umożliwiające łatwiejsze przeprowadzenie ataku na przykład na kontroler. Ostatni, siódmy wektor zagrożeń to brak mechanizmów do analizy śledczej i usuwania skutków awarii, co może uniemożliwić szybkie i bezpieczne przywrócenie sieci do stanu normalnego działania.

Jak przedstawiono w Tab. 3.2, wskazane tam wektory ataku o numerach 3-5 są specyficzne dla SDN, jako że bazują na oddzieleniu płaszczyzny sterowania od płaszczyzny transportu oraz wynikają z wprowadzenia nowego elementu w infrastrukturze, jakim jest centralny kontroler. Pozostałe wektory ataku są obecnie spotykane w tradycyjnych sieciach, należy jednak inaczej je adresować, biorąc pod uwagę specyfikę tego typu architektury.



Rys. 3.2. Wektory ataku w SDN [6]

Sposoby przeciwdziałania różnym zagrożeniom i problemom związanym z SDN obejmują zwiększanie bezpieczeństwa i niezawodności kontrolerów, ochronę i izolację aplikacji [127], [128], [124], [125], zarządzanie zaufaniem między kontrolerami a urządzeniami przekierowującymi [124], weryfikację integralności kontrolerów i aplikacji [124], informatykę śledczą i analizę powłamaniową [124], [125], mechanizmy weryfikacji [128], [133], [134] oraz niezawodne płaszczyzny sterowania [124], [125], [135], [134]. Mechanizmy ochrony i izolacji powinny być częścią każdego kontrolera. Aplikacje powinny być odizolowane od siebie nawzajem oraz od kontrolera. Różne techniki, takie jak domeny bezpieczeństwa oraz mechanizmy ochrony dostępu do danych, powinny być wprowadzone w celu uniknięcia zagrożeń ze strony aplikacji sieciowych.

Warto wspomnieć, że istnieją również inne podejścia do mitygowania zagrożeń bezpieczeństwa w SDN, takie jak języki deklaratywne do eliminowania podatności protokołów sieciowych [136]. Języki opisowe tego typu mogą określać ograniczenia semantyczne i strukturalne oraz właściwości bezpiecznego dostępu do komunikatów OpenFlow. Następnie kompilator może wykorzystać te dane wejściowe do znalezienia błędów programisty w implementacji operacji na komunikatach.

Wprowadzono także propozycje zapewniające podstawowe właściwości bezpieczeństwa, takie jak uwierzytelnianie [137] i kontrola dostępu [138]. C-BAS [137] jest opartą na certyfikatach architekturą uwierzytelniania, autoryzacji i rozliczalności (AAA), służącą do poprawy kontroli bezpieczeństwa w środowiskach opartych na technologii SDN. Rozwiązania typu C-BAS można uczynić wysoce bezpiecznymi i niezawodnymi poprzez hybrydowe architek-

Tabela 3.2. Zagrożenia ogólne i specyficzne dla SDN

Wektor ataku	Specyficzny dla SDN?	Potencjalne skutki dla SDN
1	NIE	Cel ataków DDoS, wpływający na działanie sieci
2	NIE	Wadliwie działający przełącznik może spowolnić całą sieć
3	TAK	Wykorzystanie scentralizowanego kontrolera
4	TAK	Kompromitacja kontrolera zagraża całej sieci
5	TAK	Uruchomienie złośliwych aplikacji na kontrolerze
6	NIE	Łatwe przeprogramowanie całej sieci
7	NIE	Utrudnione odzyskiwanie danych po ataku i diagnostyka błędów

tury systemowe, które łączą różne technologie i techniki z zakresu systemów rozproszonych, bezpieczeństwa oraz odporności na błędy i włamania [139], [140], [141].

Poniżej przedstawiono wybrane publikacje, w których opisano i zaadresowano zagrożenia wynikające ze specyfiki technologii SDN tak jak przedstawiona w rozprawie publikacja [23].

FloodDefender [142] został zaproponowany jako framework do ochrony kontrolera SDN przed atakami DDoS. Znajduje się on pomiędzy kontrolerem a aplikacjami w celu wykrywania ataków zalewających kontroler wiadomościami Packet_In (generowanymi, gdy w przełączniku brakuje odpowiedniego wpisu w tablicy przepływów), filtrowania pakietów atakujących i zarządzania regułami przepływu. FloodDefender chroni także przełącznik-ofiarę przed wyczerpaniem pasma kanału sterowania, wysyłając część wiadomości Packet_In do sąsiednich przełączników, które następnie przekazują je do kontrolera. Według przeprowadzonych symulacji wykorzystanie CPU w kontrolerze, jak i wykorzystanie tablicy przepływów w przełączniku utrzymuje się poniżej 15% podczas ataku DDoS.

PacketChecker zaproponowany w [143] chroni przed złośliwym wstrzykiwaniem pakietów w sieciach opartych na technologii SDN. Wykorzystując struktury danych typu hash i śledząc poprawność pakietów sieciowych, rozwiązanie to tworzy instancje mapujące w celu powiązania unikalnych szczegółów charakteryzujących urządzenia płaszczyzny danych, na przykład adresów MAC, identyfikatorów Datapath ID (DPID) OpenFlow i portów przełączników. PacketChecker pobiera informacje o polach nagłówka z pierwszej wiadomości Packet_In otrzymanej od każdego przełącznika podłączonego do sieci, co jest wykonywane w celu zarejestrowania przełącznika i zapisania odpowiednich struktur w pamięci. Kolejne wiadomości Packet_In muszą być weryfikowane z zapisanymi informacjami, tak aby pakiety wykazujące jakiekolwiek niezgodności były natychmiast odrzucane w celu uniemożliwienia ich przetwarzania przez inne aplikacje kontrolera. Kontroler dodaje odpowiednie reguły przepływów w celu odrzucenia pakietów pochodzących ze zidentyfikowanych źródeł wstrzykujących pakiety.

Przeprowadzone symulacje pokazały, że PacketChecker pozawala utrzymać stabilne działanie sieci podczas przeprowadzanych ataków.

W pracy [144] zaproponowano nowy mechanizm fingerprinting'u urządzeń sieciowych SDN. Autorzy twierdzą, że choć przełączniki sieciowe pochodzą od tego samego producenta, często mają różne cechy i wykazują zauważalne rozbieżności związane z czasem przetwarzania przy wykonywaniu tych samych funkcji. Wszystkie urządzenia, które zostały niedawno podłączone do sieci, muszą zostać zarejestrowane w module fingerprinting'u, zanim zostaną uruchomione. Następnie operatorzy sieci mogą stworzyć wstępny obraz sieci i za każdym razem, gdy urządzenie łączy się z kontrolerem, moduł obronny blokuje każdą aplikację sieciową próbującą nawiązać połączenie z nowym urządzeniem. Ten stan blokady trwa do momentu zakończenia procesu autoryzacji i jeśli proces uwierzytelniania zakończy się powodzeniem, blokada jest usuwana, w przeciwnym razie urządzenie pozostaje odizolowane od pracy w sieci. W przypadku niepowodzenia uwierzytelniania moduł fingerprinting'u ostrzega operatora sieci, że dołączane jest potencjalnie niebezpieczne urządzenie. Należy jednak zauważyć, że autorzy w artykule przedstawili jedynie koncepcję, natomiast brak jest informacji o szczegółach implementacji oraz wynikach badań.

Framework KISS, zaproponowany w [145], ma na celu podniesienie bezpieczeństwa poprzez mechanizm rejestracji i kojarzenia urządzeń, który wymusza użycie bezpiecznego, szyfrowanego kanału oraz lekkiego i zdecentralizowanego generatora/weryfikatora kluczy losowych. Kanał szyfrowania wykorzystuje bibliotekę kryptograficzną NaCl[14] wraz z Poly 1305, SHA512 MAC i kilkoma innymi znanymi schematami haszowania. Losowe generowanie kluczy szyfrujących jest realizowane przez mechanizm Integrated Device Verification Value (iDVV), który według autorów może zapewnić systemowi odpowiednią losowość w generowaniu kluczy. Centrum dystrybucji przystosowane jest do realizacji procesów rejestracji i asocjacji. Na etapie rejestracji, przed ustanowieniem kanału komunikacyjnego, urządzenia mogą się zarejestrować i uzyskać bezpieczne klucze, które będą wykorzystywane w procesie asocjacji. Na etapie kojarzenia oba końce komunikacji, nadawca i odbiorca, żądają pary klucz inicjalny – klucz do synchronicznego wygenerowania bezpiecznych kluczy kryptograficznych kanału przy użyciu mechanizmu iDVV. Wygenerowane klucze kryptograficzne mogą być użyte do szyfrowania/rozszyfrowania kanału komunikacyjnego. Przeprowadzone eksperymenty wykazały, że zastosowanie lekkiego mechanizmu kryptograficznego skutkuje niskim opóźnieniem w procesie generowania kluczy w porównaniu z innymi znanymi schematami kryptograficznymi. Autorzy zwrócili także uwagę, że przy nieodpowiednim doborze mechanizmów szyfrujących może być wymagane trzykrotne zwiększenie mocy obliczeniowej, aby obsłużyć taką samą liczbę przepływów.

SM-ONOS [146] jest administracyjnym systemem uprawnień, który zarządza poziomami dostępu do kontrolera, a nawet do innych zasobów sieciowych, o które proszą obce aplikacje. System autoryzacji SM-ONOS opiera się na egzekwowaniu dwóch rodzajów polityk kontroli

[14] https://nacl.cr.yp.to

dostępu dla aplikacji sieciowych: pierwszy rodzaj składa się z zestawu polityk poziomu dostępu zdefiniowanych przez twórców aplikacji, w których różne role mogą decydować o przyznaniu określonych przywilejów dostępu do zasobów sieciowych i usług rezydujących w interfejsie kontroli aplikacji. Drugi rodzaj polityk dostępu odnosi się do ograniczeń ustalanych ręcznie przez administratorów sieci. Polityki te opisują rodzaje przepływów, które aplikacje mogą obsługiwać, oraz segmenty sieci, w których mogą działać. Aplikacja sieciowa nie może zostać aktywowana, dopóki oba rodzaje polityk nie zostaną całkowicie zatwierdzone przez administratorów sieci. Przeprowadzone eksperymenty potwierdziły skuteczność proponowanego systemu, jednocześnie zbadano, że system wprowadza dodatkowy koszt w postaci spadku wydajności sieci o 5-20%, co według autorów jest rozsądną wartością.

Path Class Approach (PCA) [147] to podejście do bezpieczeństwa skoncentrowane na danych, w którym pakiety przepływów, określane jako obiekty danych, są mapowane na określone ścieżki przesyłania przy użyciu etykiet. Wybrana ścieżka ma gwarantować określone wymagania ochronne oraz zestaw polityk bezpieczeństwa, które mają zapewnić poufność, integralność i dostępność informacji zawartych w tych przepływach. W przedstawionych symulacjach dynamicznie szacowano dostępność łącza na podstawie jego aktualnego wykorzystania. Dzięki temu możliwe jest programowanie tras pakietów, które spełniają wymagania bezpieczeństwa skoncentrowane na danych dla konkretnych obiektów. Przeprowadzone symulacje pokazały, że mechanizm PCA wykazał się skuteczną reakcją na przeciążenia ścieżek generowane przez ataki DoS, efektywnie usuwał przeciążone łącza ze zbioru istniejących ścieżek przepływu i zastępował je nowymi ścieżkami, które egzekwowały wymagania ochrony i polityki bezpieczeństwa skoncentrowane na danych. W zależności od symulowanego scenariusza czas potrzebny na detekcję nieprawidłowej ścieżki wynosiła od 954 do 2837ms.

Z kolei autorzy w [148] zaproponowali rozwiązanie Global Flow Table (GFT), które dostarcza innym aplikacjom związanym z bezpieczeństwem istotnych szczegółów o przepływach sieciowych w taki sposób, aby inne aplikacje SDN mogły wykorzystywać te informacje jako gotowe dane wejściowe. GFT buduje globalną bazę danych tablic przepływów, żądając informacji o tablicach przepływów do ograniczonego zestawu przełączników sieciowych, a także oblicza każdą ścieżkę przepływu w sieci przy użyciu efektywnych algorytmów. Funkcje te zapewniają globalny widok stanu wszystkich ścieżek przepływu w sieci, który jest przydatny w innych mechanizmach obronnych. Przeprowadzone symulacje potwierdziły skuteczność zaproponowanego rozwiązania.

Temat analizy powłamaniowej w środowiskach SDN został zaprezentowany w artykule [149] i obejmuje wykrywanie anomalii z wykorzystaniem sztucznej inteligencji, wyzwalanie alarmów oraz formatowania danych dla dowodów zebranych w miejscu ataku. Moduł pozyskiwania i ekstrakcji danych jest w stanie skonstruować zbiory danych dowodowych, wykorzystując informacje zebrane z dzienników kontrolnych i dzienników uruchamiania, obrazów pamięci i nośników danych. Następnie moduł fuzji danych integruje w postaci klastrów wszystkie informacje pozyskane ze źródeł fizycznych i programowych. Klastry stają

się źródłem danych wejściowych, które są podawane do analizy danych z wykorzystaniem technik uczenia maszynowego, specjalizujących się w wykrywaniu anomalii, wyzwalaniu alarmów i dokumentacji zgłoszonego ataku. W opisywanej pracy, autorzy nie przedstawili informacji na temat implementacji i przeprowadzonych badań.

Natomiast NOSArmor [150] to propozycja, która buduje kompletny szkielet bezpieczeństwa SDN, wykorzystując integrację kilku indywidualnych mechanizmów ochrony. Różne podejścia do zabezpieczeń są zintegrowane jako bloki bezpieczeństwa, które oddzielnie zajmują się konkretnymi wektorami ataków zagrażających kontrolerowi SDN. Osiem bloków, tj. autoryzacja oparta na rolach, separacja procesów, mediator konfliktów reguł, menedżer zasobów i kontroler wywołań systemowych, stanowi szkielet zabezpieczeń, z czego pięć zapewnia środki bezpieczeństwa przeciwko wektorom ataków powstającym w warstwie aplikacji. Pozostałe trzy bloki, tj. weryfikator protokołu OpenFlow, weryfikator łącza oraz śledzący lokalizację, mają za zadanie łagodzić zagrożenia pochodzące z płaszczyzny danych. Wyboru bloków dokonano w celu rozwiązania następujących problemów związanych z bezpieczeństwem: nieautoryzowane aplikacje uzyskujące dostęp do wewnętrznej pamięci masowej lub wymuszające polecenia przełącznika; weryfikacja zaufanych hostów; spreparowane komunikaty protokołów pochodzące od złośliwych hostów; aplikacje instalujące w kontrolerze sprzeczne reguły polityk; złośliwe aplikacje nadużywające funkcjonalności kontrolera; nieuczciwe urządzenia wysyłające nieprawdziwe wiadomości OpenFlow oraz aplikacje nadużywające zasobów sieciowych. Autorzy za pomocą symulowanego środowiska przedstawili skuteczność zaproponowanych mechanizmów adresujących różne problemy spotykane w sieciach SDN.

Poniżej przedstawiono także publikacje poruszające zagadnienia najbardziej zbliżone tematycznie do publikacji [23] autora niniejszej rozprawy, a w których opisano sposoby na uzyskanie przez atakującego pewnych informacji z infrastruktury SDN, które powinny być traktowane jako poufne. Niestety publikacje te zawierają badania często jedynie w środowisku symulacyjnym, co obniża ich wiarygodność w odniesieniu do rzeczywistych sieci.

Autorzy w [151] przedstawiają sposób obrony przed atakiem Know Your Enemy (KYE), wykorzystującym instalowanie reguł przepływów na żądanie w reaktywnych sieciach jak SDN. Atak KYE dzieli się na dwie fazy, z których w czasie pierwszej (rekonesansu) atakujący wysyła pakiety sondujące do docelowego przełącznika i obserwuje reguły przepływów instalowane w reakcji na przesłane dane. W drugiej fazie (wnioskowania), atakujący jest w stanie odkryć informacje o konfiguracji sieci. Autorzy zaproponowali zaciemnienie przepływów jako sposób obrony przed KYE, gdzie wykorzystuje się zdolność przełączników SDN do modyfikowania przesyłanych pakietów. Gdy atakujący wysyła przepływ sondujący do przełącznika, kontroler dodaje pojedynczą regułę przepływu, która nakazuje przełącznikowi zmodyfikować jego niektóre pola nagłówka i przekazać pakiety do innego przełącznika, a proces ten jest powtarzany dla wcześniej ustalonej liczby przełączników. Działanie takie ma na celu zmylenie atakującego. Przeprowadzone symulacje wykazały, że narzut w postaci opóźnienia przekazywania pakietów

zależy od liczby pośredniczących przełączników i wynosi od 43ms dla jednego przełącznika do 80ms dla sześciu.

Natomiast w pracy [152] przedstawiono techniki pozwalające na fingerprinting sieci SDN, w tym parametry polityki, takie jak hard i soft timeouts, pola dopasowania OpenFlow używane przez kontroler SDN, reakcję kontrolera na zdarzenie związane z zapełnieniem tablicy oraz informacje o topologii sieci docelowej. Wykorzystując tę wiedzę, można przeprowadzić atak, zwłaszcza na płaszczyznę infrastruktury SDN. Wykonane symulacje wykazały, że możliwe jest wyznaczenie wartości parametrów hard i soft timeouts z błędem wynoszącym od 0,62 do 3,4%.

W końcu artykuł [153] zwraca uwagę na wąskie gardło komunikacyjne między kontrolerem a przełącznikami. Tablica przepływów, funkcjonująca jako pamięć podręczna między kontrolerem a przełącznikami, łagodzi to wąskie gardło poprzez buforowanie reguł przepływu otrzymywanych z kontrolera na każdym przełączniku, ale jej rozmiar jest bardzo ograniczony ze względu na wysoki koszt. Autorzy zaproponowali modele ataków, które wykorzystują specyficzne, podobne do pamięci podręcznej zachowania tablicy przepływów, aby poznać jej wewnętrzną konfigurację i stan, a następnie odpowiednio zaprojektować parametry ataku. Na postawie przeprowadzonych symulacji oceniono skuteczność zaproponowanej metody na 90%.

Należy jednak zauważyć, że w pracach [152], [153] sformułowano wnioski na podstawie pomiarów czasu odpowiedzi systemu w środowisku symulacyjnym, co poddaje w wątpliwość skuteczność rozwiązania w rzeczywistych warunkach sieciowych. Ponadto autorzy nie zaproponowali metody obrony przed przedstawionymi atakami.

Tab. 3.3 podsumowuje opisane w tej części publikacje oraz porównuje je z przedstawioną w ramach niniejszej rozprawy pracą [23] (pogrubiono pracę wchodzącą w skład rozprawy).

Tabela 3.3. Publikacje adresujące aspekty bezpieczeństwa technologii SDN.

Praca	Proponowana metodyka, technika lub procedura	Metoda ewaluacji	Sposób ochrony
Bezpieczeństwo SDN			
[23]	Szacowanie poufnych parametrów tablicy SDN poprzez wykorzystanie techniki aktywnego fingerprinting'u z użyciem adaptacyjnych algorytmów.	Eksperyment	Opis nowego ataku, propozycja ochrony
[142]	Filtrowanie pakietów oraz zarządzanie tabelami przepływów.	Symulacja	Detekcja, mitygacja
[143]	Porównywanie metadanych przychodzących pakietów z informacjami z przełączników sieciowych.	Symulacja	Detekcja, mitygacja
[144]	Uwierzytelnianie nowych urządzeń przed interakcją z nimi.	Brak	Zapobieganie
[145]	Rejestracja nowych urządzeń, szyfrowanie komunikacji.	Eksperyment	Zapobieganie
[146]	Ograniczanie aplikacji politykami dostępowymi.	Eksperyment	Zapobieganie
[147]	Rozdzielenie przepływów zgodnie z wymaganiami bezpieczeństwa.	Symulacja	Zapobieganie
[148]	Globalna tablica przepływów pozwala na scentralizowany monitoring sieci.	Symulacja	Detekcja, zapobieganie
[149]	Spostrzeżenia dotyczące rozwoju wielowarstwowego systemu analizy powłamaniowej w środowisku SDN.	Brak	Analiza powłamaniowa
[150]	Integracja kilku indywidualnych mechanizmów ochrony w jednym narzędziu.	Symulacja	Detekcja, zapobieganie
[151]	Zaciemnienie przepływów poprzez modyfikację pakietów.	Symulacja	Detekcja, mitygacja
[152]	Wyznaczanie parametrów polityk SDN (timeout) na podstawie pojedynczej odpowiedzi systemu.	Symulacja	Opis nowego ataku
[153]	Wyznaczenie parametrów polityk SDN (rozmiar tablicy) na podstawie pojedynczej odpowiedzi systemu. Przeprowadzenie dopasowanego ataku.	Symulacja	Opis nowego ataku

4. Detekcja zagrożeń typu ransomware na podstawie analizy charakterystyki ruchu HTTP

Ransomware, jako złośliwe oprogramowanie (opisane w 2.2), to jedno z największych zagrożeń dotyczących zarówno użytkowników domowych jak i firm, od małych do wielkich korporacji [32].

W artykule "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics", opublikowanym w czasopiśmie "Computers and Electrical Engineering", przedstawiono nowatorskie podejście do wykrywania komunikacji oprogramowania ransomware z serwerem C&C oparte na technologii SDN, które wykorzystuje charakterystykę tej komunikacji. Na podstawie obserwacji ruchu sieciowego dwóch rodzin oprogramowania crypto ransomware (najbardziej rozpowszechnionych w momencie publikacji, tj. CryptoWall oraz Locky), udowodniono, że analiza sekwencji komunikatów HTTP i ich rozmiarów jest wystarczająca do wykrycia tego typu zagrożeń. Ponadto, przeprowadzona analiza ujawniła podobieństwa w komunikacji obu rodzin złośliwego oprogramowania. Z kolei podobieństwa te można wykorzystać do stworzenia wydajnego i skutecznego systemu detekcji.

W związku z tym, głównymi nowatorskimi cechami poniższego rozwiązania są:

- Wykorzystanie architektury opartej na SDN do łagodzenia skutków działania crypto ransomware, co pozwala na stworzenie elastycznego i skutecznego systemu detekcji i prewencji.
- Przedstawienie wyników analizy behawioralnej opartej na pomiarach sieciowych dwóch rodzin ransomware, a mianowicie CryptoWall i Locky.
- Zaprojektowanie systemu detekcji i łagodzenia skutków ransomware, który opiera się na wnioskach z przeprowadzonych analiz behawioralnych, o których mowa powyżej.
- Opracowanie i eksperymentalna ocena prototypu zaproponowanego systemu detekcji.

Wykorzystanie technologii SDN jest w tym przypadku kluczowe w szczególności ze względu na centralizację widoku sieci i możliwości analizy komunikacji sieciowej w dużych i hybrydowych środowiskach. Większość producentów sprzętu sieciowego wspiera technologię SDN zarówno w fizycznych jak i wirtualnych urządzeniach, co sprawia, że zaproponowane rozwiązanie jest uniwersalne i elastyczne.

Sama technologia SDN została także wykorzystana do blokowania ruchu oznaczonego jako podejrzany, aby uchronić pozostałe systemy przed zagrożeniami typu crypto ransomware.

Uzyskane rezultaty badań eksperymentalnych potwierdzają, że zaproponowane rozwiązanie jest skuteczne i wydajne (97% pozytywnych detekcji ransomware). Należy jednak zwrócić uwagę, że badania te zostały przeprowadzone w 2017 roku dla popularnego w tym czasie złośliwego oprogramowania CryptoWall i Locky. Metoda opisana w tej publikacji może się okazać nieskuteczna w detekcji nowych generacji rodzin ransomware, które stają się coraz bardziej wyrafinowane.

Warto podkreślić, że przeprowadzone badania nie są oparte jedynie na symulacjach, co podnosi wiarygodność otrzymanych wyników. Artykuł został także dostrzeżony w środowisku naukowym – liczba cytowań w bazie Google Scholar na dzień 24.03.2022 wynosi 138, w Scopus 78, natomiast w WoS 54.

Computers and Electrical Engineering 66 (2018) 353–368

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics☆

Krzysztof Cabaj[a], Marcin Gregorczyk[b], Wojciech Mazurczyk[b,*]

[a] Warsaw University of Technology, Institute of Computer Science, Warsaw, Poland
[b] Warsaw University of Technology, Institute of Telecommunications, Warsaw, Poland

ARTICLE INFO

Article history:
Received 23 November 2016
Revised 16 October 2017
Accepted 17 October 2017
Available online 27 October 2017

Keywords:
Ransomware
Malware
Software-defined networking
Network security

ABSTRACT

Ransomware is currently one of the key threats facing individuals and corporate Internet users. Especially dangerous is crypto ransomware that encrypts important user data, and it is only possible to recover it once a ransom has been paid. Therefore, devising efficient and effective countermeasures is a pressing necessity. In this paper we present a novel Software-Defined Networking (SDN) based detection approach that utilizes the characteristics of the ransomware communication. Based on an observation of network communication between two crypto ransomware families, namely CryptoWall and Locky, we conclude that an analysis of the HTTP message sequences and their respective content sizes is enough to detect such threats. We show the feasibility of our approach by designing and evaluating a proof-of-concept SDN-based detection system. The experimental results confirm that the proposed approach is feasible and efficient.

1. Introduction

The year 2016 was named "the year of the ransomware" by the mass media, and this type of threat is currently considered by the security community and law enforcement agencies (e.g., Europol's recent "2016 Internet Organized Crime Threat Assessment" report [1]) as a key threat to Internet users. Ransomware is a type of malicious software that is designed for direct revenue generation and which after infection holds the victim's machine or user's critical data "hostage" until a payment is made. Ransomware developers are constantly improving their "products" making it harder to design and develop effective and long-lasting countermeasures. Considering the fact that more and more devices are foreseen to be connected to the Internet due to the Internet of Things (IoT) paradigm, it makes it a perfect environment for ransomware to spread in the foreseeable future [2]. The ransomware plague is so widely spread that there are even crime-as-a-service tools available in the dark web (like TOX ransomware-construction kit [3]) that allow even inexperienced cybercriminals to create their own customized malware, to manage infections, and profits.

There are two main types of modern ransomware, i.e., locker and crypto. The infection for both kinds of malicious software happens in a similar way, i.e., a user machine is infected by means of various attack vectors, e.g., by drive-by-download, malvertisement, phising, spam, or different forms of social engineering, etc. However, what comes after the infection is different for both types. *Locker ransomware* denies the user access to an infected machine but typically the underlying system and files are left untouched. On the other hand, *crypto ransomware* is a kind of data locker that prevents the user from

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. Zhihan Lu.

* Corresponding author.
E-mail addresses: kcabaj@ii.pw.edu.pl (K. Cabaj), m.gregorczyk@tele.pw.edu.pl (M. Gregorczyk), w.mazurczyk@tele.pw.edu.pl (W. Mazurczyk).

https://doi.org/10.1016/j.compeleceng.2017.10.012
0045-7906/

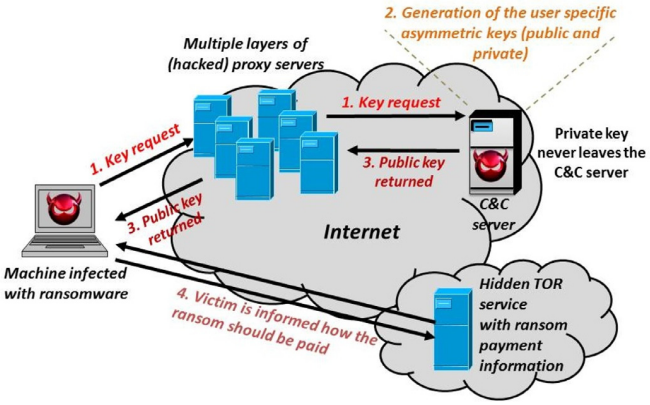


Fig. 1. Typical asymmetric key cryptography-based ransomware scheme.

accessing her/his vital files or data (e.g., documents, pictures, videos, etc.) by using some form of encryption. Therefore, attacked files are useless until a ransom is paid and the decryption key is obtained. Then after the user's machine is locked or data is encrypted the victim is presented with an extortion message. In many cases, paying the ransom to the cybercriminal is the only way to get back access to the machine/data. The value of the requested ransom differs and is typically in the range US$300–$700, and the favored payment currency is bitcoins [2]. It must be emphasized that not only individual users are currently targeted, but also companies and institutions like hospitals and law enforcement agencies, etc. Clearly, effective and efficient solutions to counter ransomware infections are desired.

Although the first cases of crypto ransomware have been known for more than 10 years (e.g., Trojan.Gpcoder), it must be emphasized that the recent plague (Symantec reported an astounding 4000% rise in crypto ransomware incidents in 2014 [4]) of this type of malware is related to the improved design of the cybercriminals' "products". The main difference now is that crypto ransomware has moved from custom or symmetric key to asymmetric key cryptography (Fig. 1). In this case, when the machine is infected, it contacts the C&C (Command & Control) server through multiple proxy servers (which are typically legitimate but hacked machines) to request a public encryption key. At the C&C a pair of matching public-private keys is generated for each infection and the public key is returned to the compromised host (the private key never leaves the C&C server). Then the public key is used to securely transfer the session key to encrypt the chosen files, which are deemed the most important for the user. It is worth noting that if correctly implemented, asymmetric crypto ransomware is (practically) impossible to break. The most prominent ransomware, and one of the first to introduce asymmetric key cryptography, is CryptoWall 3.0, which was discovered at the beginning of 2015, and later followed by others, such as CryptoWall 4.0 and Locky, etc.

Software-defined networking (SDN) is one of the emerging networking paradigms [5]. Its main benefit is that it allows the decoupling of the control and data planes, i.e., the underlying network infrastructure is abstracted from the applications. Therefore, the network can be managed in a logically centralized way. Apart from many potential applications [5], recently SDN has become a promising opportunity to provide security for current networks in a more flexible and effective manner [6]. Moreover, currently, SDN serves as the base of cloud computing environments in public, private, and hybrid clouds, and it is envisioned to become a core of future network services. The majority of network device manufactures are supporting SDN with their physical and virtual equipment using the OpenFlow protocol. SDN is standardizing the management of heterogeneous networks. Applications written for the SDN controller will work without additional adjustments to different devices supporting SDN in both physical and virtual environments. These are the main reasons why SDN was chosen as the key technology for our solution.

Taking the above into account, in this paper we present a dedicated SDN-based system for ransomware detection and mitigation. It must be noted that when it is possible to successfully discover ransomware communication, then, obviously, sometimes, it may be too late to prevent encryption for that particular victim. However, it is still possible to utilize this incident to provide feedback for the detection system to "save" other users. This phenomenon is well-known in nature where a single organism often has to make a self-sacrifice for the sake of the group [7]. As more and more analogies between cybersecurity and nature are continuously drawn [8], this may be another opportunity to reuse natural experiences in this regard to improve communication networks' defenses.

The proposed detection system, in this paper, has a focus on the crypto ransomware that utilizes asymmetric cryptography. While designing and developing the system we took into consideration results from the traffic analysis of two modern ransomware families, i.e., CryptoWall and Locky. Based on the performed analyses we concluded that these ransomware families share some similarities, which can be utilized to create an efficient and effective detection system. Thus, the main contributions of the paper are:

- Use of the SDN-based architecture to mitigate crypto ransomware, which allows the creation of a flexible and effective detection and prevention system.
- Presentation of results of the network measurements-based behavioral analysis of two ransomware families, namely CryptoWall and Locky.
- Design of the SDN-based detection and mitigation system that relies on our findings from the performed behavioral analyses mentioned above.
- Development and evaluation of the proof-of-concept implementation of the proposed detection system.

The rest of this paper is organized as follows: first, we present existing work related to malware detection using the characteristics of the HTTP traffic, SDN-based approaches to cyber threat detection and prevention including ransomware, and, in particular, existing ransomware detection methods. In Section 3 we present the results from the behavioral analyses of two ransomware families. In the next section, a SDN-based ransomware detection system that relies on ransomware HTTP traffic characteristics is described and evaluated. Finally, Section 5 concludes our work.

2. Related work

In this section we review existing work, first related to ransomware detection and then SDN-based threat detection with special emphasis for ransomware countermeasures. Moreover, we analyse methods for discovering threats based on HTTP protocol traffic analysis.

Malware detection has been extensively studied in recent years. In general, such techniques can be broadly divided depending on where the malware activities are observed, either at the network-level [9,10], system-level [11,12], or both [13]. In this paper we propose a network-level SDN-based solution.

From the functional perspective, still the most common approach to malware detection is payload-based, i.e., these approaches are only effective if the malware communication is invariant [14]. Thus, if the malicious software is using plaintext communication protocol, then such invariants may exist (e.g., protocol keywords can serve as a part of the payload signatures). The same situation can be experienced for several encrypted malware communication protocols, where certain invariant plaintext fragments result in certain invariant encrypted data (if encryption keys are not wisely used), which can be easily applied as a signature as well [15]. However, often various malware families (including ransomware) do not exhibit the characteristics indicated above, which means they can circumvent payload signature-based detection systems [16]. Recently, a novel approach was proposed that uses tamper resistant features of the transport layer protocol to distinguish malware heartbeat messages (that sustain the connection with the C&C) from the legitimate traffic. However, the authors noted that they observed a substantial decrease in the detection of malicious software in which traffic is disguised in HTTP messages [17].

When it comes specifically to countering ransomware several works exist. In [18] the author proposed a Heldroid system that employs static taint analysis together with lightweight symbolic execution to find code paths that indicate device-locking activity or attempts to encrypt files on external media. The authors of [19] describe the results from a long-term study of ransomware between 2006 and 2014. Based on the gathered data, they concluded that the number of ransomware families with sophisticated destructive capabilities remains quite small. They also proposed a detection system that is based on the monitoring of abnormal file system activity. Scaife et al. [20] introduced an early-warning detection system for ransomware that monitors all file activities and alerts the user in case something suspicious is identified using a union of three features, i.e., file type changes, similarity measurement, and entropy. Another recent work [21] introduced EldeRan, which is a machine learning approach for dynamically analysing and classifying ransomware. The proposed solution monitors a set of actions performed by applications in their first phases of installation and tries to detect the characteristics signs of ransomware. The obtained experimental results proved that this approach is effective and efficient, which also shows that dynamic analysis can support ransomware detection by utilizing the set of characteristic features at run-time that are common across families, and that helps the early detection of new variants.

When it comes to SDN-based solutions tailored to security purposes, the first work that proposed a general SDN-based anomaly detection system was put forward by Mehdi et al. in 2011 [6]. The authors showed how four traffic anomaly detection algorithms could be implemented in an SDN context using OpenFlow compliant switches and NOX controller. The obtained experimental results proved that these algorithms are significantly more accurate in identifying malicious activities in home networks when compared to the ISP. Further, other researchers utilized SDN to detect network attacks [22] or to monitor dynamic cloud networks [23].

Several recently published papers deal with SDN-based malware detection. Jin and Wang [24] analyzed malicious software behaviors on mobile devices. Based on the acquired knowledge they proposed several mobile malware detection algorithms, and implemented them using SDN. Their system was able to perform real-time traffic analysis and to detect malicious activities based only on the connection establishment packets. In [25] the authors designed and developed an SDN-based architecture specialized in malware analysis aimed at dynamically modifying the network environment based on malicious software actions. They demonstrated that this approach is able to trigger more malware events than traditional solutions. To the best of our knowledge, no works, so far, have proposed ransomware detection using a SDN-based system.

HTTP traffic characteristics have been already used as a recognition property for detecting malware. For instance, a characteristic combination of HTTP URI request parameters in the malware communication may be utilized to discover malicious

Table 1
Network measurement statistics for CryptoWall 4.0 and Locky ransomware.

Ransomware type	No. of samples	No. of sample executions	Size of traffic traces
CryptoWall 3.0/4.0	359 (331 for CW3.0 and 28 for CW4.0)	3700	≈5 GB
Locky	428	2200	≈330 MB

communication. For example, Perdisci et al. showed that clustering HTTP traffic can be used to extract behavioral features that can be used to recognize HTTP-based malware [26]. In [27] Zegers investigated whether the order of HTTP request headers can be used to recognize malware communication. However, the author states that different websites have their own header order. This is similar to applications that communicate over HTTP (Windows updates and anti-viruses). Therefore, the conclusion is that it is unfeasible to use the HTTP headers' order to reliably identify malicious software. Kheir [28] also analysed User Agent (UA) anomalies within malware HTTP traffic to extract the signatures for its detection. His observation was that almost one malware out of eight uses a suspicious UA header in at least one HTTP request (this includes typos, information leakage, outdated versions, and attack vectors such as XSS and SQL injection). Based on the above, the authors developed a new classification technique to discover malware user agent anomalies. The obtained experimental results showed that this approach considerably increases the malware detection rate. A similar approach was proposed in [29], where to detect malware the authors took advantage of statistical information about the usage of the UA of each user together with the usage of a particular UA across the whole analysed network and typically visited domains. However, it must be noted that, so far, no approach exists that takes into account the sequences of HTTP messages and their respective sizes as the main feature for the detection system, as we propose in this paper.

The first dedicated SDN-based ransomware security solutions designed to improve the protection against ransomware was recently proposed in [30]. The authors introduced two approaches that took advantage of the fact that without successful communication to the C&C server the infected host is not able to retrieve the public key and, as a result, it cannot start the encryption process. Both methods rely on dynamic blacklisting of the proxy servers used to relay communication between an infected machine and the C&C server. However, the main drawback of the proposed approaches is that they are efficient only when the ransomware proxy servers have been previously identified by means of behavioral analysis of known malware samples. Therefore, it is not possible to discover new campaigns when they first appear.

Due to this fact, in the paper, we take a different angle, i.e., we utilize the characteristics of the network communication between the infected host and a proxy server to detect the ransomware data exchange. As we observe, based on two "popular" ransomware families, the communication protocol utilized in the analyzed malware samples are similar; therefore, the utilization of this knowledge can lead to the development of an efficient and effective ransomware countermeasure. Such an approach can also be treated as a complementary solution to the detection methods proposed in [30], as it can form a source of feedback for the dynamic blacklisting of the proxy servers.

3. Crypto ransomware traffic characteristics based on CryptoWall and Locky families

Based on our experiences with crypto ransomware, we decided to choose two families, namely CryptoWall and Locky, to present their communication characteristics and to illustrate different levels of malware sophistication. These two ransomware examples utilize asymmetric cryptography; however, they differ when it comes to the network part of the communication, although they both use HTTP protocol. That is why, in the following subsections, we describe both ransomware families' communication characteristics in detail. Samples for further analysis were gathered from a freely available source, mainly the malwr.com service. Gathered samples were later evaluated using the Maltester dynamic analysis environment [31] developed at the Warsaw University of Technology. Table 1 summarizes our measurement efforts for the two malware families indicated above.

3.1. CryptoWall communication characteristics

CryptoWall 4.0 has been active since October 2015, and it superseded the previous 3.0 version. From the network traffic perspective, to communicate with the C&C server CryptoWall uses domain names instead of direct IP addresses. An analysis of the traffic from infected machines revealed that CryptoWall communication is based on HTTP POST messages. The communication is directed to the scripts uploaded onto the hacked web servers (proxy servers) and it is encrypted using the RC4 algorithm. However, the encryption key is very easy to retrieve as it is incorporated within the HTTP request.

The CryptoWall communication is depicted in Fig. 2. During the first data exchange, the ransomware reports its unique identifier and the victim's IP address to the C&C, which acknowledges the received information. In the second exchange, the response contains an image containing instructions for the victim and TOR address of the ransom webpage, the victim's personal code and an RSA 2048-bit public key that is used for encrypting the data. Then the encryption process starts. Finally, during the last data exchange an acknowledgement for the public key reception is provided. It is worth noting that CryptoWall 4.0 does not report the end of the encryption process to the C&C and does not report the number of encrypted files, which was the case for its predecessor – CryptoWall 3.0.

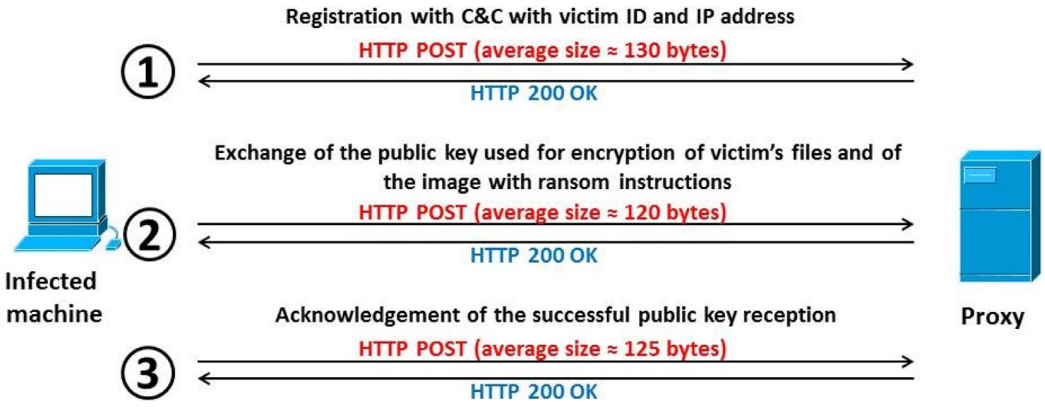


Fig. 2. CryptoWall 4.0 communication.

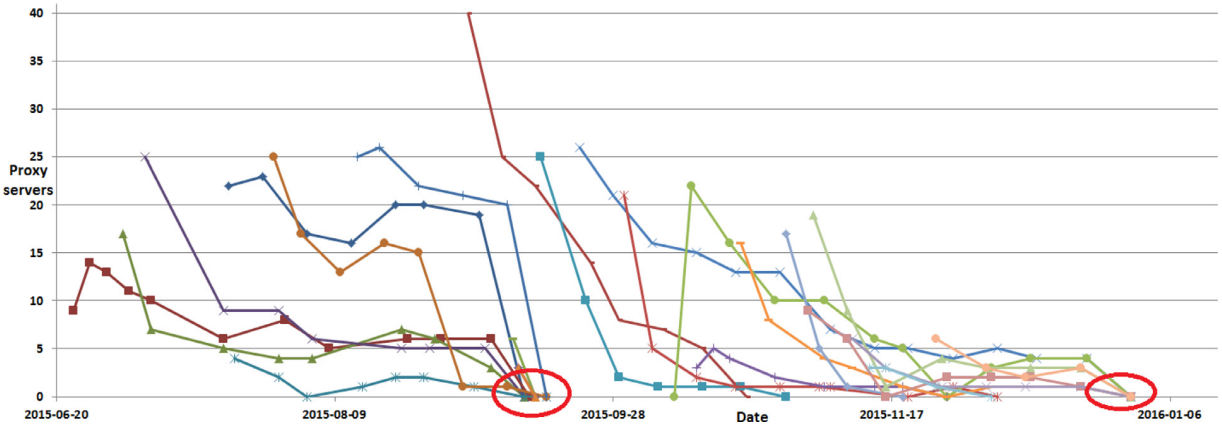


Fig. 3. CryptoWall 3.0 proxy servers' activity.

During our research we investigated a total of 359 CryptoWall samples, from which 28 were CryptoWall 4.0 and the rest were CryptoWall 3.0. Each CryptoWall sample contained a hardcoded list of proxy servers that is used during the transfer of the public key from the attacker C&C server. We also discovered that typically these servers are victims too. To execute the proxy script, the cybercriminals behind CryptoWall utilize compromised legitimate servers. When a new campaign of CryptoWall appears there are many samples with the same proxy list. Our analyses for CryptoWall 4.0 revealed 19 proxy lists. The average proxy list contained 47 server addresses (the shortest list had 27 and the longest 70 addresses). Using information about the servers in the proxy list we investigated how long these servers had been responsive. Figs. 3 and 4 illustrate the number of responsive servers in the detected proxy lists for CryptoWall 3.0 and 4.0, respectively. What should be emphasized is that the longest responding proxy server was active for as long as 11 weeks.

Two of the time instants observed in Fig. 3 are particularly interesting and are highlighted with red ovals. The first one is related to the massive shutdown of proxy servers and the immediate appearance of a few new samples with new proxy lists. The second represents the complete shutdown of the CryptoWall 3.0 infrastructure.

It is worth noting that at the time of the CryptoWall 3.0 shutdown there was a noticeable lack of CryptoWall 4.0 server activity (Fig. 4). It might be assumed that during this time the cybercriminals were performing (most probably) some kind of infrastructure update or maintenance. Another, more general conclusion is that typically the number of active servers from the proxy server list decreased over time.

3.2. Locky communication characteristics

Locky ransomware samples have been observed since mid-February 2016. Its communication patterns are similar to those of the CryptoWall family. The first resemblance is related to the utilization of HTTP POST messages. However, in this case the encryption scheme used to secure the data exchanged was better chosen. Thus, when this article was written, it was not possible to decrypt the Locky communication. Therefore, the information presented in this subsection is mostly deduced based on our previous experience and knowledge of ransomware. The communication in the case of Locky consists of four HTTP POST exchanges, which (most probably) serve the same aim as in the case of CryptoWall 3.0/4.0 (Fig. 2). For example,

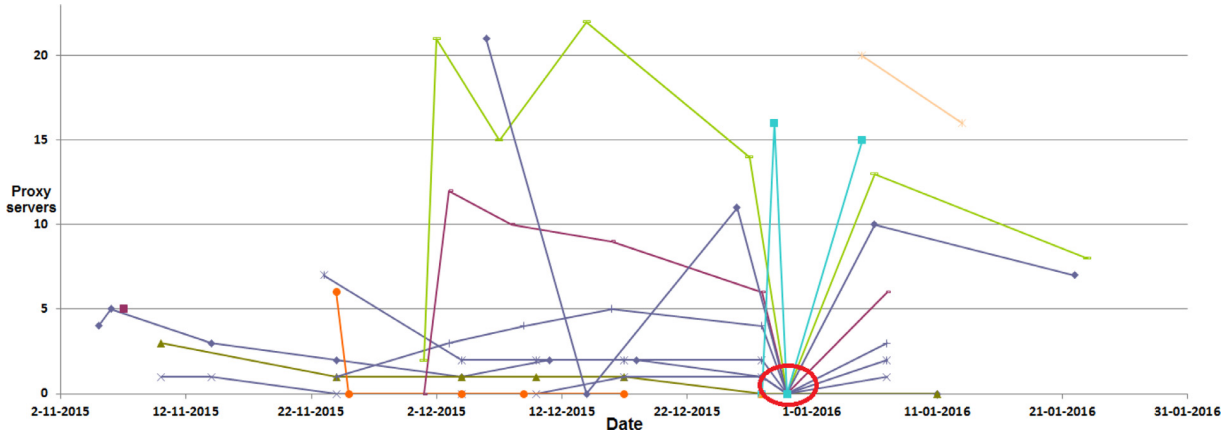


Fig. 4. CryptoWall 4.0 proxy servers' activity.

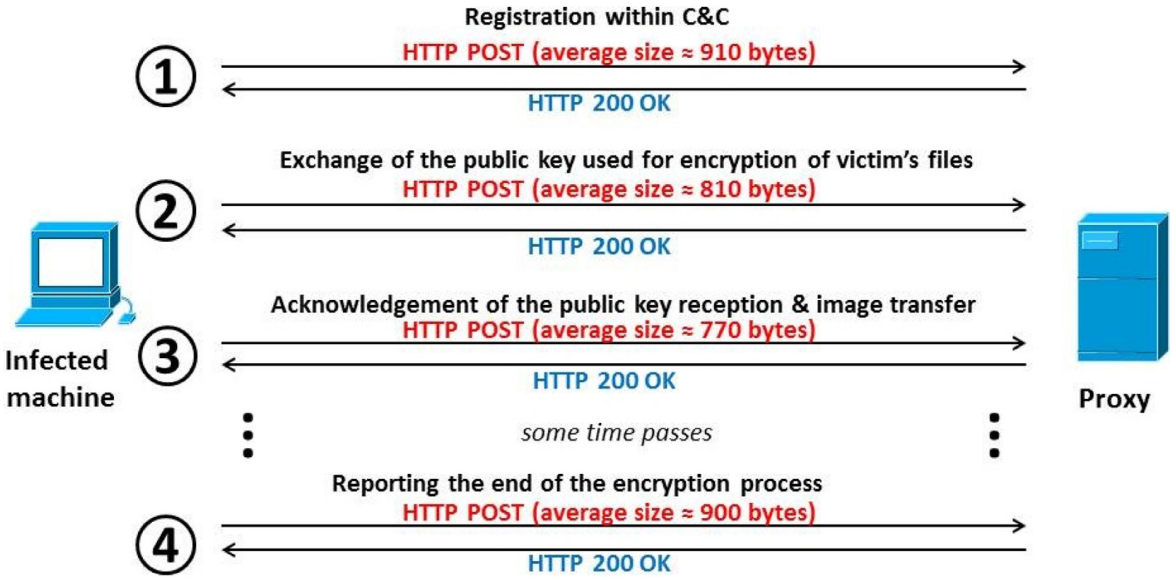


Fig. 5. Locky (most probable) communication.

it is possible that the image transfer as well as the public encryption key transmission is performed during the third HTTP data exchange as the size of the resulting instructions presented to the user when extorting the ransom is of a similar size as the HTTP 200 OK response.

The other difference between Locky and CryptoWall is that the disruption of the communication when transmitting the public key in the case when CryptoWall was able to block the encryption process. However, Locky, in such cases, did not stop the encryption and used a hardcoded key (probably one dedicated key per sample). Additionally, in Locky the communication capabilities to contact the C&C server have been significantly extended. Each Locky sample has its own list of C&C IP addresses hardcoded. In the case that these addresses are already blocked it uses a DGA (Domain Generation Algorithm) to generate new C&C domains and tries to contact them. It is also worth noting that the first Locky samples generated exactly the same HTTP POST message sizes. However, Locky has evolved over time and currently the message sizes vary, as presented in Fig. 5.

We have been gathering Locky samples since the end of March 2016. Identified samples were executed in the Maltester dynamic analysis environment. Obtained results allowed characterizations of the Locky network behavior. During these more than eight months we observed various changes in Locky activities. The most interesting findings are further elaborated in this section. Until now, we observed three distinct versions of the Locky communication protocol. The first version used the same URL, i.e., main.php and in all executions the sizes of the messages sent to the C&C server were the same: 101, 55, and 94 bytes of (probably) encrypted binary data. At the beginning of April the protocol was changed, and then each execution of the ransomware sample resulted in randomly generated message sizes. This behavior is similar to the change

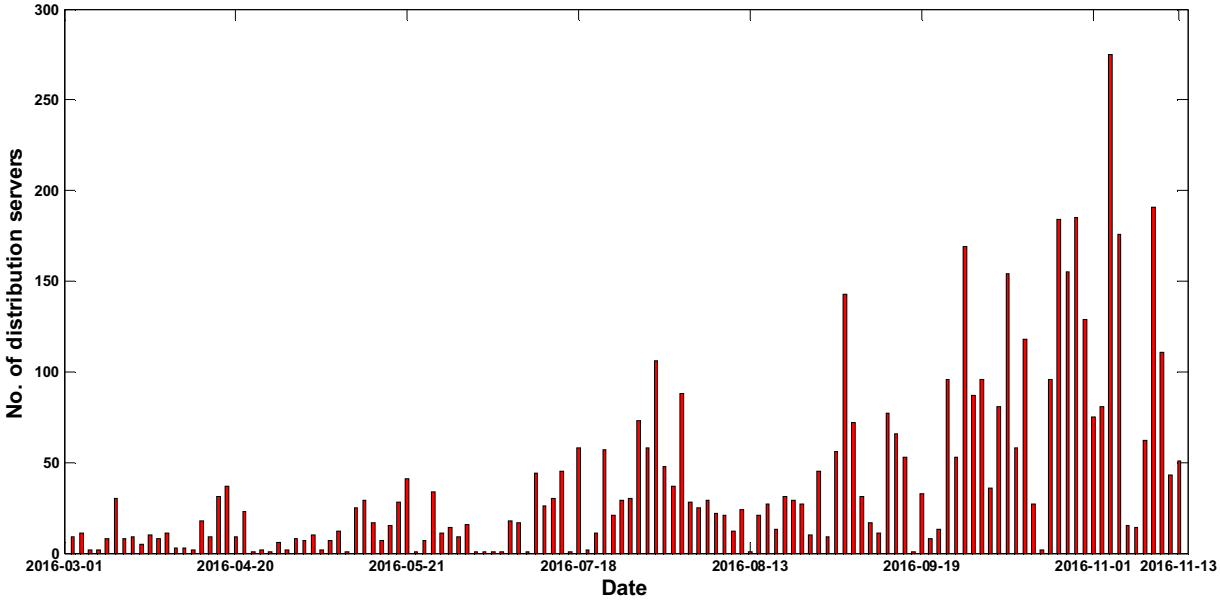


Fig. 6. Locky distribution servers' statistics (based on data from http://ransomwaretracker.abuse.ch).

Table 2
Statistics of the analyzed Locky samples.

C&C Server URL	Date of first sample analysis	No. of samples	No. of hardcoded C&C	No. of different DGA algorithms	DLL entry point name
main.php	2016.03.21	41	18	7	–
submit.php	2016.03.28	24	15	3	–
userinfo.php	2016.05.03	226	42	9	–
access.cgi	2016.05.30	2	2	1	–
/upload/_dispatch.php	2016.05.31	18	14	9	–
/php/upload.php	2016.08.01	11	16	3	–
/data/info.php	2016.08.29	19	14	8	1
apache_handler.php	2016.09.27	20	22	8	1
linuxsucks.php	2016.10.24	9	11	5	2
message.php	2016.11.03	57	24	20	11
information.cgi	2016.11.21	1	3	1	1

observed in the CryptoWall family during the transition from version 3.0 to 4.0. The last observed modification to the communication protocol happened around the end of July, when the encoding of data using only ASCII printable characters was first observed. This Locky version has greater message sizes and was used during our experiments.

From March 2016 we also observed changes in how the malware was distributed. Initially, during the first stage, typically, a Microsoft Word or Excel document with embedded hostile macro was sent as SPAM, and later the main stage of the ransomware was downloaded from the distribution server and executed. Later, around the end of May, we started observing encrypted executables. The first stage macro in the document downloaded an encrypted file, decrypted it on the infected machine and executed it. Obviously, the file is hosted on the distribution server without the decryption key and cannot be executed. The number of distribution servers between March and November 2016 is illustrated in Fig. 6. It can be observed that till June 2016 the number of distribution servers discovered daily was typically not higher than 50; however, in the last months it doubled with a few spikes reaching more than 150. This means that Locky was still active and the cybercriminals behind it were still trying to reach as many victims as possible. At the time of finishing this article, after a week of silence, the first sample of a new campaign was discovered and analyzed.

The most eye-catching aspect of Locky are the URLs used for C&C communication. During our research we investigated 11 distinct URLs. We decided to use their names as indicators of the new Locky campaigns. Our analyses revealed that each campaign lasted no longer than one month. From version 3.0 of the communication protocol we observed various features of the gathered samples in more detail. We discovered, among others, hardcoded C&C servers, used DGAs (Domain Generation Algorithms), and entry function DLL names. All the analyzed details are summarized in Table 2. The provided information allows an illustration of the extent of the resources used by the cybercriminals' infrastructure.

The second column of Table 2 contains the date when we analyzed the first sample from a given campaign. We did our best to analyze samples as they appeared; however, sometimes when the attackers introduced more modifications to

their code it took more time to analyze how the samples can be gathered. Therefore, the provided dates can be used only as an approximate time of the start of a new campaign. The fourth and fifth columns are associated with the various aspects of communication with the C&C server. As mentioned at the beginning of this section, Locky used more reliable C&C communication when compared to CryptoWall. Firstly, Locky tried to contact the hardcoded IP addresses of the C&C servers. Our analysis shows that each sample had from two to five such addresses hardcoded within its binary. During our research we observed that if the hardcoded C&C servers were shut down then the next samples were equipped with previously unseen IP addresses for malicious servers. Due to this fact we observed an average of almost 18 C&Cs per analyzed campaign (maximum of 42). In cases when all hardcoded C&C servers were not active, Locky switched to the second method in which it utilized DGA. What is interesting, is that samples from the same campaign used various DGA algorithms, which in effect led to various domains being generated during the same time frame. The number of used DGA algorithms is presented in the fifth column in Table 2. The sixth column contains the number of used Entry Functions. The first six campaigns used normal executables during the main stage of the ransomware infection. However, at the end of August, we noticed a change in the malware distribution technique. We discovered that the downloaded file was a DLL library. To execute malware in such a form additional information, i.e., the name of the entry function, is needed. The first two campaigns utilizing such a solution used the same, simple entry function with the name “qwerty”. However, the next campaigns switched to more complicated and often modified entry function names. Obviously, all such changes were introduced to make ransomware analysis more difficult and to increase the time needed for security professionals to understand its behavior.

4. SDN-based ransomware detection based on HTTP traffic characteristics

As mentioned in the previous section, all analyzed ransomware families utilize a custom protocol that is used for downloading the encryption key and image, etc. from the attacker’s C&C server. Due to the fact that the communication process is similar for both families, this characteristic feature can be used for ransomware infection detection. That is why we use this knowledge to introduce a novel network traffic classification method that is based solely on the size of the data inserted by the victim into the outgoing sequence of the HTTP POST messages.

4.1. Proposed detection method

The proposed scheme can be divided into three main phases (Fig 7). The first is a learning phase in which real ransomware samples and the network traffic generated by them were used to extract the characteristic features from the outgoing HTTP messages (particularly their size). During the second, fine-tuning phase we focus on adjusting the parameters of the detection method. Finally in the last, detection phase we utilize data gathered during the two previous phases and detect infections using the proposed SDN based solution. Later, in Section 4.4 we present the results of the evaluation of the introduced detection method using both malicious and benign network traffic. It must also be emphasized that for each ransomware family the described three-phase procedure is conducted separately. All the phases are described in detail in the following sections.

4.1.1. Learning phase

In this phase, the data used later for the detection purposes were prepared and preprocessed. During this process, ransomware samples for each ransomware family (f), which were accumulated (details on how this dataset was formed can be found in Section 4.3), were executed in the controlled environment and all traffic generated by the infected machine was captured for further preprocessing. For this purpose we utilized the Maltester environment as described in [31]. From the obtained traffic, initially, three HTTP POST messages sent by the malware were located and the respective content lengths were extracted. In the result, the kth infection ($k \in (1, n)$, where n is the total number of all ransomware samples for a given family) is characterized by a vector S_{fk} consisting of three numbers $[s_{1k}, s_{2k}, s_{3k}]$, with each representing the content size sent in the consecutive HTTP POST messages (POST triples vector). The set of all S_{fk} was treated as a base to establish the main distinguishing feature for detection purposes.

4.1.2. Fine-tuning phase

When the learning set of vectors S_{fk} was (separately) prepared for each ransomware family (f), then the centroid vector of the corresponding feature vectors (C_f), as well as the minimal and maximal Euclidean distances for all vectors from S_{fk} to C_f, were calculated. Then using C_f and datasets that consist of the HTTP traffic from new infections (not utilized during the learning phase) and the benign HTTP traffic (without infections), the limit distance (d_{Lf}) from the centroid value was fine-tuned (the exact fine-tuning procedure is explained later in the experimental section). During the fine-tuning phase, information related to the minimal and maximal distance to the centroid was utilized. As a result, at the end of this process all the necessary parameters used later during the detection phase were calculated. In particular, these were the three values to form the centroid vector C_f and the limit distance d_{Lf} that was used to assess whether the currently evaluated feature vector extracted from the monitored HTTP traffic was malicious or not (Fig. 8, left). It must also be emphasized that for each ransomware family there will be a separate centroid vector C_f as well as the limit distance d_{Lf} established. All details concerning the obtained experimental results are presented later in Section 4.4.

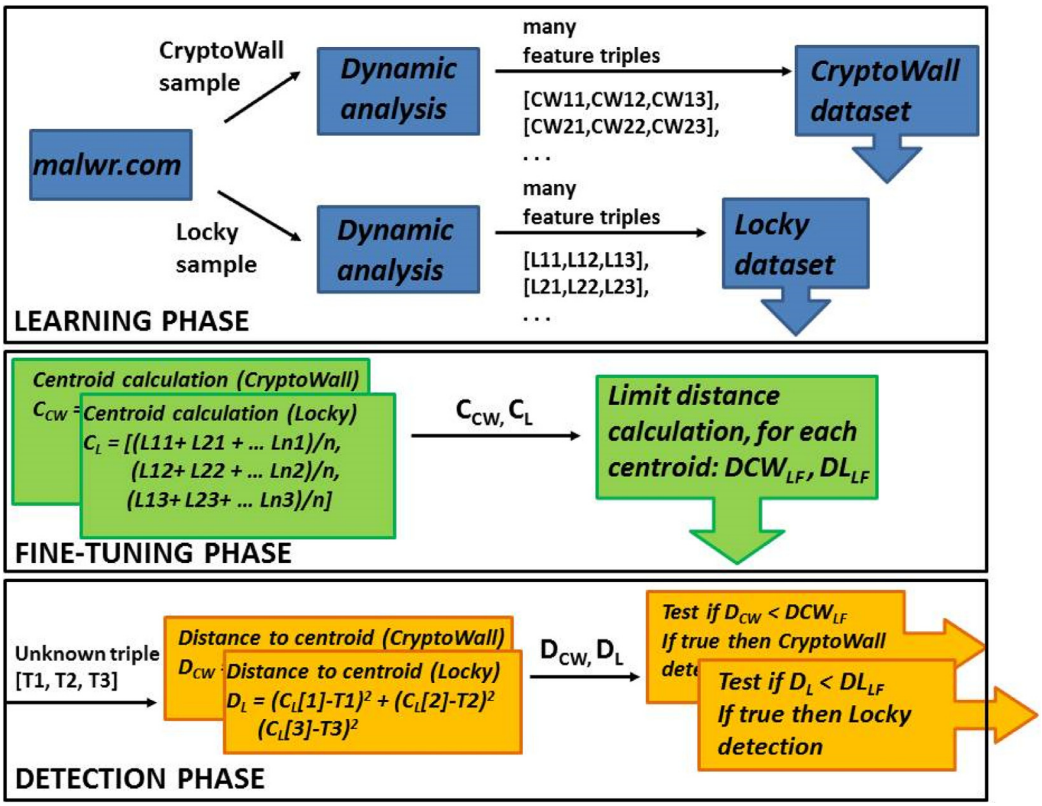


Fig. 7. Main phases of proposed detection system.

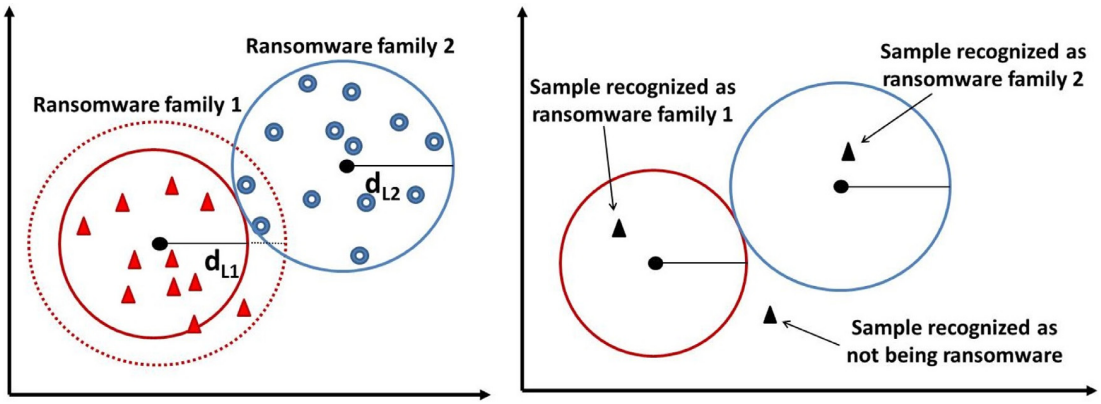


Fig. 8. Proposed detection method fine-tuning phase (left) and detection phase (right).

4.1.3. Detection phase

The HTTP connections that are the subject of the monitoring and potential ransomware detection must be initially preprocessed. During our experiments we developed two variants of the preprocessing software. The first one, mainly for experimental use, extracted data from the traffic traces (files in the .pcap format). The second one was directly integrated within the SDN controller and can be used for real-time detection of the existing ransomware infections. Both preprocessing solutions analyzed incoming TCP segments that contain HTTP traffic and reassembled the outgoing messages. When the whole request was reassembled, then the size of the data sent to the server and host IP or domain address were extracted from the HTTP header. The extracted host IP or domain name was then used for defining the destination HTTP server, which possibly could be a ransomware C&C or a proxy server. If confirmed as being malicious, information about such servers can be used, for example, to feed the ransomware countermeasures based on IP/domain blacklisting like the one proposed in [30] or directly block such traffic using the SDN-based system proposed in this paper.

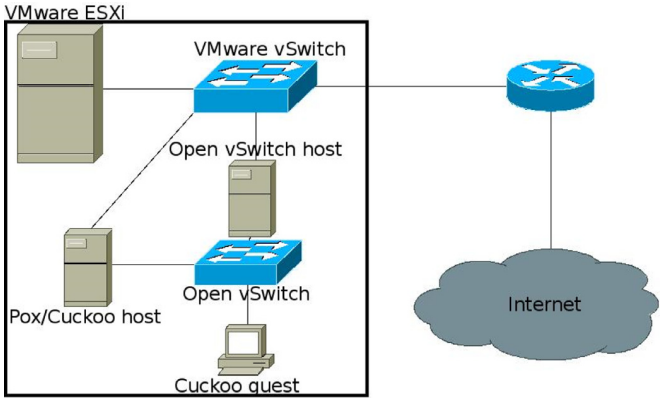


Fig. 9. Experimental testbed.

In the next step, for each HTTP server identified, as indicated above (malicious or not), a list was created that consisted of the generated HTTP messages sizes. Any new HTTP message was thus added at the end of the list associated with a given server's IP address or a domain name. If after insertion of a new value to the list, its size was equal to or greater than three (as for the proposed detection method we analyze three consecutive HTTP POST messages' sizes), then the detection of possible ransomware was performed. For this aim, the test vector was constructed using the last three values from the list. Then the distance between this vector and the centroid for a given ransomware family was measured. If the calculated value was smaller than the limit distance (d_{Lf}) established during the learning phase, the system decided that this HTTP message sequence was a sign of ransomware infection (Fig. 8, right). For example, consider the HTTP POST messages sequence presented in Fig. 5 for Locky. In this case, the extracted test vector was [910, 810, 770]. If during the learning phase the obtained centroid vector for this ransomware family had a value of [900, 800, 775] and the limit distance was fine-tuned to the value 20, then this traffic will be detected as ransomware, because the calculated Euclidean distance for this sample was 15.

4.2. Experimental test-bed

In the experimental evaluation we decided to utilize an OpenFlow protocol that enables typical networking devices to be supervised by an external controller, which is currently the standard for implementing the SDN paradigm. The experimental environment used during the experiments is depicted in Fig. 9. All operating systems were installed on virtual machines, which were managed by VMware ESXi hypervisor installed at a server with an Intel Xeon E5-2450 1.9 GHz (24 logical CPUs, 24 GB RAM). To exclude any interfering factors during the experiments, no other virtual machines were active on the hypervisor.

To securely execute the malware samples during the experiments with SDN detection, an application Cuckoo Sandbox [32] was used. In more detail, we utilized:

- A Cuckoo guest with Microsoft Windows 7 SP1 $\times$ 86_64 installed. We decided to use this OS as currently it is dominant with a market share of almost 50% of desktops [33]. Therefore, it is likely that it would be targeted by cybercriminals. It must also be noted that the update and firewall services were intentionally disabled on this host so the malware samples could be executed with no interruption.
- A Pox/Cuckoo and Open vSwitch (OVS) host using Debian 8.5 $\times$ 86_64 with kernel 3.16.36.

For separation purposes, two additional subnetworks were used: first for the Pox/Cuckoo traffic (using vNICs connected to Open vSwitch) and second for the management traffic (vNICs connected to VMware vSwitch).

The functioning of the proposed test-bed was as follows: after the Cuckoo agent (listener) was started on the guest machine, a snapshot with memory was taken. The OVS was acting as a default gateway for the Cuckoo guest, so the traffic directed towards the Internet was always passing through it. The OVS was controlled using the OpenFlow protocol by a Python-based SDN controller – Pox. The SDN application (based on the Layer 2 learning switch) forced the OVS to forward every HTTP packet to the controller for inspection. Next, at the controller the decision on the packet was made. During the experiments, the Cuckoo host first contacted the ESXi API to restore the Windows host from the snapshot. Then ransomware samples were executed. As a consequence, the traffic directed towards the C&C server was generated based on which SDN detection and prevention system made the decision to block it or not.

As stated before, the Pox application is based on a Layer 2 learning switch. The first requisite was to configure the OVS to send the whole packet to the SDN controller instead of the default 120 bytes. It was needed for the decision about rejecting or allowing traffic that was made based on the HTTP POST message size. Next, for every event (incoming packet) the handler function was invoked to check if:

Table 3
Characteristics of datasets used for evaluation of proposed detection method.

Dataset name	No. of HTTP POST	No. of POST triples	No. of domains
Ransomware traffic for CW 4.0	750	250	250
Ransomware traffic for Locky	750	250	250
Alexa traffic	22 579	11 950	8187
MACCDC traffic	17 249	15 862	761

- the packet is valid (can be parsed),
- the TCP is used in the transport layer,
- the TCP destination port is 80 (HTTP traffic),
- the packet payload length is greater than 0 (to exclude TCP segments with no data, for example, segments only with an ACK flag).

If any of the checks failed, no blocking rule was set as such a packet was not considered to be malware traffic. If all checks were positive, the payload was passed to the custom "oracle" function where the introduced detection algorithm was implemented (see Section 4.1). If a ransomware infection was detected, then the appropriate blocking flows (bidirectional using only the hostile C&C server's IP address) were inserted into the OVS switch (otherwise an infection on any other machine cannot be not prevented).

4.3. Utilized evaluation datasets

As described in the previous section, as well in Sections 3.1 and 3.2, our own datasets were used during the learning phase. This artificially generated data were prepared during the dynamic analysis. The evaluation phase required independent data that contained traffic with and without hostile activity. This section describes how the relevant malicious and benign network traffic was gathered. We collected malware samples from two main locations: the *malwr.com* website, from which it is possible to directly download ransomware samples, and *ransomtracker.abuse.ch,* where one can obtain the addresses of malicious distribution servers (the ones from which the ransomware binary is downloaded to the attacked host). Gathered samples were then executed in the controlled environment: the one described in Section 4.2 and Maltester [31] where the generated test traffic was captured. This formed the first dataset with malicious HTTP traffic.

To confirm that the proposed detection method is valid and does not generate too many false-positives, we utilized normal, benign HTTP traffic. As our proposed detection approach relies on HTTP POST message characteristics, we needed representative network dumps. However, it must be noted that it is difficult to obtain unencrypted HTTP traffic captures as they might contain sensitive and private data (logins, passwords, or cookies), thus such network dumps are usually not publically shared. Typically these repositories contain only metadata without upper layer payloads. Therefore, we generated benign HTTP traffic using 200,000 of the most popular websites from the Alexa ranking (extracted from http://s3.amazonaws.com/alexa-static/top-1 m.csv.zip on 19.10.2016). We wrote a bash script that launched each of the top-ranked Alexa websites in the Google Chrome web browser. It is worth noting that many pages are coded to generate a lot of POST messages without any users' interaction. This is to gather knowledge about the users' environment (operating system, web browser, setup, etc.) and not for usual POST purposes (like submitting forms or cookies). Such behavior is normal and can be observed world-wide, therefore the obtained traffic should not be considered as artificial or synthetic. As every website needs a certain amount of time to be fully loaded, to speed up the process, 25 pages were launched in parallel for 25 seconds. To exclude the possibility of influencing the interaction between different pages, the Google Chrome web browser was executed in Incognito Mode. Additionally, after every iteration, the web browser's cache was cleared. Therefore, the whole process of benign HTTP traffic generation took almost 56 hours, but as a result we obtained real-life HTTP traffic as it is typically experienced by Internet users.

The last part of the HTTP traffic was obtained from the 2012 MACCDC (maccdc.org) cyber defense competition data dumps. It is a capture the flag type event, which means that the HTTP data are not fully realistic, but they provided examples of various types of network attacks and contained many POST messages. Table 3 summarizes the statistics of the gathered HTTP POST and domains values.

4.4. Experimental results

This section describes in detail the process for the proposed detection method and its fine-tuning, and it presents the obtained experimental results. The experiments were performed independently for CryptoWall 4.0 and Locky ransomware families, and the results are described in the following subsections.

4.4.1. Locky ransomware

Initial data for the Locky family consisted of more than 250 traces of successful infections with recorded data exchanges between the infected victim and the C&C server. The complete dataset was divided into two distinct parts: the first, which

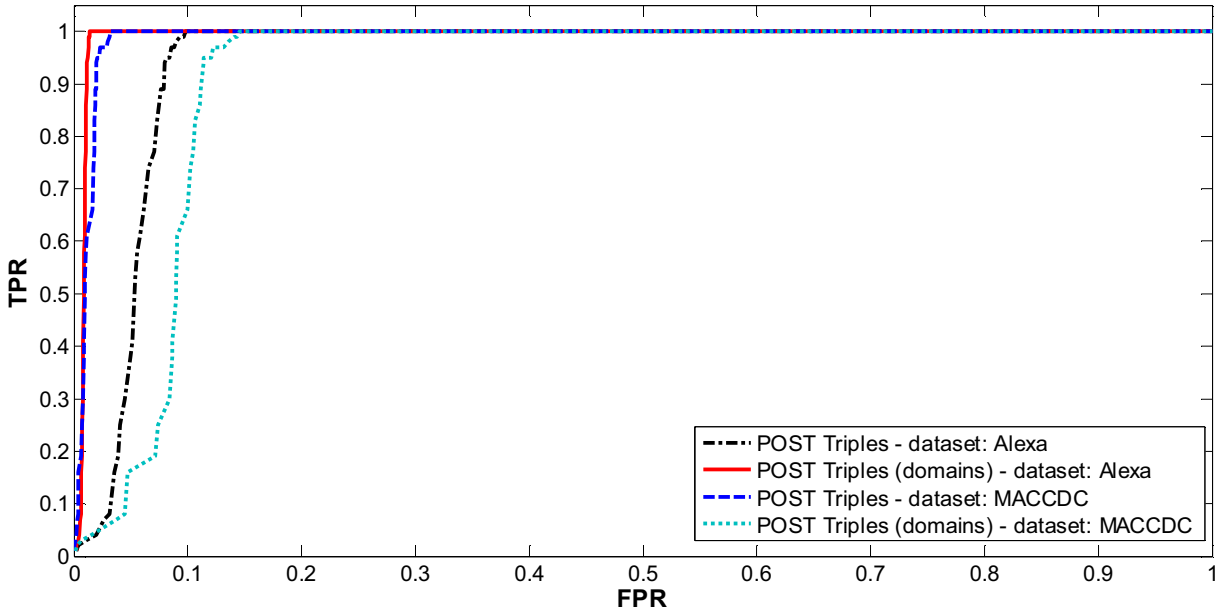


Fig. 10. ROC curves for Locky detection for Alexa and MACCDC datasets.

contained 150 samples, was used during the learning phase for the centroid parameters and the limit distance calculations. For each point in this dataset, the distance to the centroid was calculated. The results of the average, minimal, and maximum distance values were then used to determine the range of acceptable values for the limit distance parameters. For performance reasons, all calculations of the distance use the distance square, which avoids CPU intensive calculations of the square root. The results for the learning phase include the minimal distance (actually distance square) of 6914 and maximal of 274,370.

The second part of the original dataset consisted of 100 samples and it was used for the calculation of the TPR (True Positive Rate), which was needed to prepare the ROC curve for the proposed detection method. This was a reasonable decision as this data were generated from real infections (performed using the controlled and protected environment), and we are confident that they represent the real characteristics of the Locky communication traffic. During this phase the number of traces flagged as containing Locky infections was analyzed for various distance values, i.e., average, minimal, and maximum. Due to the fact that all the data contains infections, under an ideal situation we should flag all of them as malicious. The TPR was calculated as a ratio of flagged traces and the total number of samples in the dataset. The process of the FPR (False Positive Rate) calculation was similar. We obtained clean traces using two distinct datasets, described in Section 4.3, and we referred to them as Alexa and MACCDC. In these datasets, contrary to the previous dataset, which contained ransomware traffic, we should not detect any Locky infections. Therefore, all the connections recognized as Locky were false positives. For the ROC curve preparation purpose we checked these datasets using the same distance range as used for the TPR calculations. During our research we investigated two types of ROC curves. The first one was associated with all tested consequent triples of POST messages, regardless how many triples were associated with this domain and how many were flagged as malicious. This was mostly true for benign domains, because in successful ransomware traffic only three or four messages were observed. These plots were denoted as POST Triples. The second one associated all calculations with domains, for which at least one triple was flagged as a Locky transmission. In this case we could count at least one for a given domain. These plots were denoted as POST Triples (domain). Fig. 10 presents the ROC curves independently for Alexa and MACCDC datasets and both features, i.e., POST Triples and POST Triples (domain).

The ROC plots for both datasets were similar, so we decided to merge them for the final fine-tuning of the Locky detector. The final ROC curve is presented in Fig. 11. The results show that the most suitable limit distance (d_L) for our method is 260,000, because it offers the best trade-off between true positives (97%) and false positives (4.95%) and (2.2%) respectively for analyzed POST triples and domains.

4.4.2. CryptoWall 4.0 ransomware

During our research we performed similar analyses as presented in Section 4.4.1 for CryptoWall 4.0. From our previous studies [30,31] we had obtained almost 270 traces containing communication traffic between the infected host and the CryptoWall C&C servers. As this value was very similar to the number of traces used for the Locky experiments we decided to split this dataset in a similar way, i.e., used 150 traces for the learning phase and 100 for the testing phase. It turned out that the HTTP message sizes sent by CryptoWall 4.0 were much smaller than those of the Locky family. After the centroid

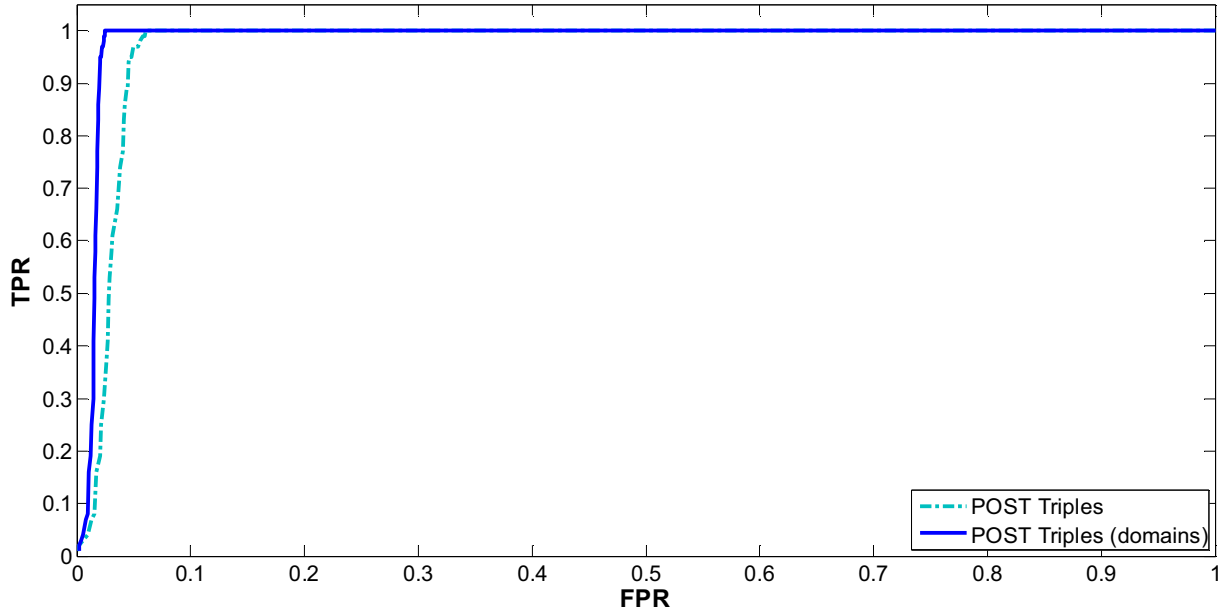


Fig. 11. ROC curves for Locky detection for merged Alexa and MACCDC datasets after final fine-tuning of detector.

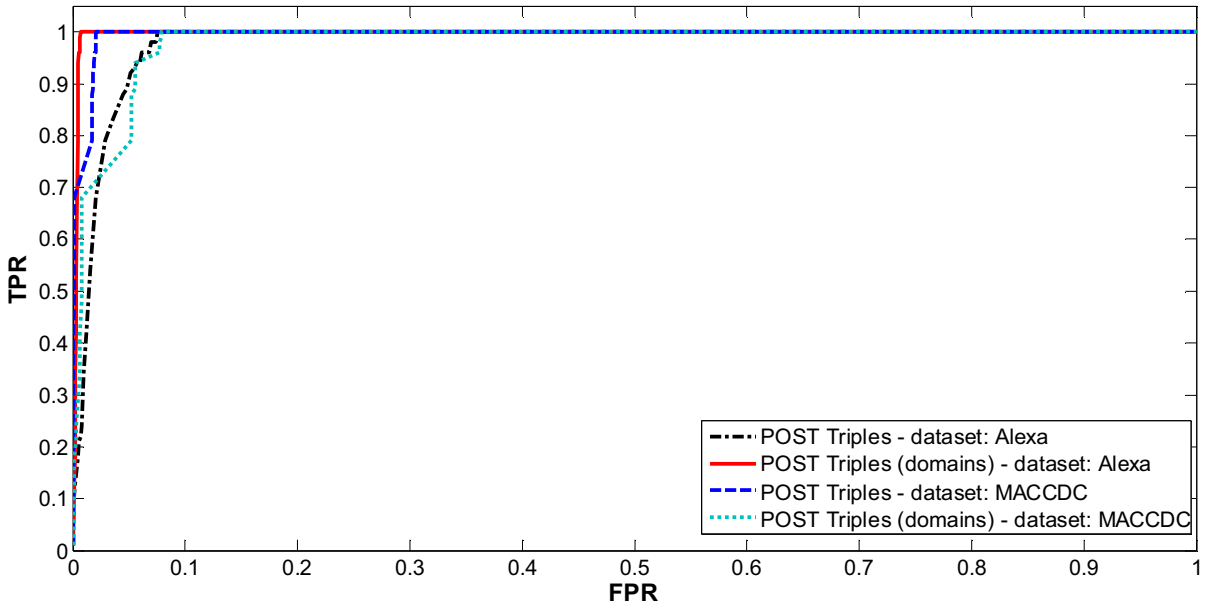


Fig. 12. ROC curves for CryptoWall 4.0 detection for Alexa (left) and MACCDC (right) datasets.

calculation we discovered that the minimal distance (for CPU performance reasons it was actually the distance square) was 25 and the maximal distance was 893. These values were then used during the TPR and FPR calculations. For TPR, to obtain 100% detection accuracy we used a limit distance of 1050. Using the second part of the traces containing the CryptoWall C&C communications, the Alexa and MACCDC datasets we gathered data for the ROC curves are illustrated in Fig. 12.

The ROC curves for both datasets were similar, so we decided to merge them for the final fine-tuning of the CryptoWall detector. The final ROC plot is illustrated in Fig. 13. The results show that the best limit distance (d_L) is 900, because it offers the best trade-off between true positives – 98% and false positives – 4.2% and 1.2% respectively for analyzed POST triples and domains.

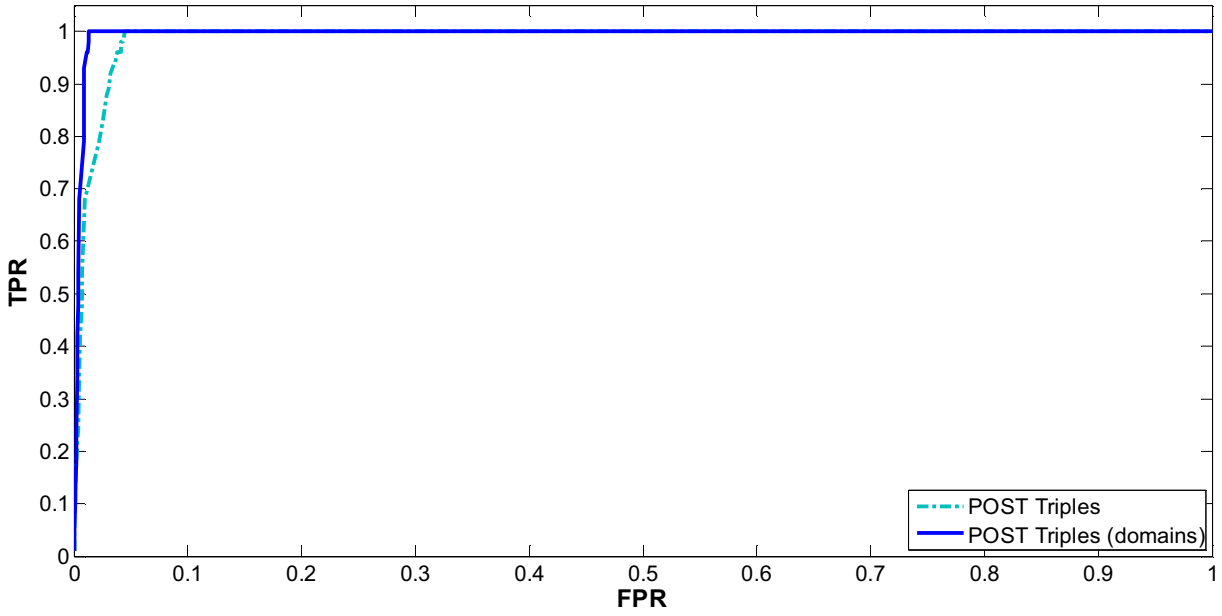


Fig. 13. Final ROC plots for CryptoWall 4.0 detection for merged Alexa and MACCDC datasets and after final fine-tuning of detector.

5. Conclusion

In this paper we proposed a novel SDN-based ransomware detection system that relies on the characteristics of the malware's communication. By performing network measurements for two very popular ransomware families, i.e., CryptoWall and Locky, we observed that a promising approach would be to detect the malicious communication between an infected host and the attacker C&C server using the HTTP traffic characteristics (HTTP message sequences and their corresponding sizes). Therefore, we designed and developed a detection system that uses the SDN solution to provide a rapid reaction to the discovered threat. The experimental results obtained using real ransomware samples proved that even such a simple approach is feasible and offers good efficacy. We were able to achieve detection rates of 97–98% with 1–2% or 4–5% false positives when relaying on domains or POST triples, respectively.

To counter ransomware, in general, we believe that it is vital to try to target and break the business model of the malware developers/exploiters by, for example, determining the most profitable malware families and disrupting their infrastructure. A complementary approach is to educate users to pay more attention to what types of files they open or what types of websites they visit. It is clear that less infections result in a lower profit and higher operation costs for the cybercriminals. Future work on ransomware detection should involve taking into account the sizes of the HTTP responses from the C&C server. Moreover, it is possible to combine the proposed approach with others, e.g., those based on blacklisting of malicious IP addresses and domains. Furthermore, it is vital to monitor ransomware development trends to provide effective countermeasures as fast as possible to limit the number of infected machines.

References

[1] Europol. Internet organised crime threat assessment 2016 (IOCTA); September 2016. URL: https://www.europol.europa.eu/content/internet-organised-crime-threat-assessment-iocta-2016.
[2] Savage K, Coogan P, Lau H. The evolution of ransomware. Symantec security response; August 2015. URL: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf.
[3] McAfee Labs. Meet 'Tox': ransomware for the rest of Us; May 2015. URL https://blogs.mcafee.com/mcafee-labs/meet-tox-ransomware-for-the-rest-of-us/.
[4] Symantec. Internet security threat report; April 2015. URL https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf.
[5] Kreutz D, Ramos FV, Verissimo P, Rothenberg C, Azodolmolky S, Uhlig S. Software-defined networking: a comprehensive survey. Proc IEEE January 2015;103(1):14–76.
[6] Mehdi SA, Khalid J, Khayam SA. Revisiting traffic anomaly detection using software defined networking. In: Proc. of the 14th international conference on recent advances in intrusion detection (RAID 2011); 2011. p. 161–80.
[7] Tofilski A, Couvillon M, Evison S, Helantera H, Robinson E, Ratnieks F. Preemptive defensive self-sacrifice by ant workers. Am Nat 11.2008,;172(5):E239–43.
[8] Mazurczyk W, Rzeszutko E. Security - a perpetual war: lessons from nature. IEEE IT Prof 2015;17(January/February 1):16–22.
[9] Gu G, Perdisci R, Zhang J, Lee W. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the 17th USENIX security symposium; 2008.
[10] Wurzinger P, Bilge L, Holz T, Goebel J, Kruegel C, Kirda E. Automatically generating models for botnet detection. In: Backes M, Ning P, editors. ESORICS 2009. LNCS, 5789. Heidelberg: Springer; 2009. p. 232–49.

[11] Bailey M, Oberheide J, Andersen J, Mao ZM, Jahanian F, Nazario J. Automated classification and analysis of internet malware. In: Kruegel C, Lippmann R, Clark A, editors. RAID 2007. LNCS, 4637; 2007. p. 178–97.
[12] Bayer U, Comparetti PM, Hlauschek C, Kruegel C, Kirda E. Scalable, behavior-based malware clustering. Network and distributed system system security symposium; 2009.
[13] Jacob G, Hund R, Kruegel C, Holz T. Jackstraws: picking command and control connections from bot traffic. 20th USENIX security symposium; 2011.
[14] Idika N, Mathur AP. A survey of malware detection techniques. Purdue University; 2007. Technical Report.
[15] Rieck K, Schwenk G, Limmer T, Holz T, Laskov P, Botzilla. Detecting the phoning home of malicious software. In: Proceedings of the 25th ACM symposium on applied computing (SAC), March; 2010.
[16] Rossow C, Dietrich CJ. ProVeX: detecting botnets with encrypted command and control channels. In: Proc. of 10th international conference, DIMVA 2013, July 18–19; 2013. p. 21–40.
[17] Celik ZB, Walls R, McDaniel P, Swami A. Malware traffic detection using tamper resistant features. Military communications conference (MILCOM), October Tampa, FL, USA; 2015.
[18] Andronio N. Heldroid: fast and efficient linguistic-based ransomware detection M.Sc. thesis. University of Illinois at Chicago; 2015.
[19] Kharraz A, Robertson W, Balzarotti D, Bilge L, Kirda E. Cutting the Gordian knot: a look under the hood of ransomware attacks. 12th conference on detection of intrusions and malware & vulnerability assessment (DIMVA 2015), July 9–10 Milan, Italy; 2015.
[20] Scaife PTN, Carter H, Butler KR. Cryptolock (and drop it): stopping ransomware attacks on user data. In: 2016 IEEE 36th international conference on distributed computing systems; 2016. p. 303–12.
[21] Sgandurra D, Muñoz-González L, Mohsen R, Lupu EC. Automated dynamic analysis of ransomware: benefits, limitations and use for detection. Computing research repository (CoRR), Ithaca, NY (USA): Cornell University; September 2016. abs/ 1609.03020, arXiv.org E-print Archive.
[22] Zaalouk A, Khondoker R, Marx R, Bayarou K. OrchSec: an orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions. In: Proc. of network operations and management symposium (NOMS); 2014. p. 1–9.
[23] Shin S, Gu G. CloudWatcher: network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In: Proc. of 20th IEEE international conference on network protocols (ICNP); 2012. p. 1–6.
[24] Jin R, Wang B. Malware detection for mobile devices using software-defined networking. In: Proc. of GENI research and educational experiment workshop (GREE '13); 2013. p. 81–8.
[25] Ceron JM, Margi CB, Granville LZ. MARS: an SDN-based malware analysis solution. In: IEEE symposium on computers and communication (ISCC); 2016. p. 525–30.
[26] Perdisci R, Lee W, Feamster N. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In: Proc. of the USENIX symposium on networked systems designs and implementation (NSDI); April 2010.
[27] Zegers R. HTTP header analysis M.Sc. thesis. University of Amsterdam; 2015.
[28] Kheir N. Analyzing HTTP user agent anomalies for malware detection. In: Proc. of 7th international workshop, DPM 2012, and 5th international workshop, SETOP 2012, September 13–14; 2012. p. 187–200.
[29] Grill M, Rehak M. Malware detection using HTTP user-agent discrepancy identification. In: IEEE international workshop on information forensics and security (WIFS); 2014. p. 221–6.
[30] Cabaj K, Mazurczyk W. Using software-defined networking for ransomware mitigation: the case of cryptowall. IEEE Netw. 2016(November/December). doi:10.1109/MNET.2016.1600110NM.
[31] Cabaj K, Gawkowski P, Grochowski K, Osojca D. Network activity analysis of CryptoWall ransomware. Przeglad Elektrotechniczny 2015;91(11):201–4 URL http://pe.org.pl/articles/2015/11/48.pdf.
[32] Cuckoo sandbox: automated malware analysis, URL: https://cuckoosandbox.org.
[33] Net market share, URL: https://www.netmarketshare.com/operating-system-market-share.aspx.

Krzysztof Cabaj holds M.Sc. (2004) and Ph.D. (2009) in computer science from Warsaw University of Technology (WUT). Assistant Professor at WUT. Instructor of Cisco Academy courses at International Telecommunication Union Internet Training Centre (ITU-ITC). His research interests include: network security, honeypots and data-mining techniques. He is the author or co-author of over 30 publications in the field of information security.

Marcin Gregorczyk is Ph.D. student and holds M.Sc. (2012) degree in telecommunications from the Warsaw University of Technology (WUT), Warsaw, Poland. RedHat Certified Architect and instructor of RedHat Enterprise Linux courses. His research interests include: Linux operating system security and network security especially cloud related.

Wojciech Mazurczyk received the M.Sc., Ph.D. (Hons.), and D.Sc. (habilitation) degrees in telecommunications from the Warsaw University of Technology (WUT), Warsaw, Poland, in 2004, 2009, and 2014, respectively. He is currently an Associate Professor with the Institute of Telecommunications, WUT. His research interests include network security, bio-inspired cybersecurity and networking and information hiding.

5. Detekcja i mitygacja ataków sieciowych TCP SYN Scan oraz DHCP Starvation

Projekt IoRL[15] (Internet of Radio Light), w którym autor niniejszej rozprawy brał aktywny udział, był realizowany w latach 2017-2020 i został ufundowany przez Komisję Europejską w ramach programu Horizon 2020. Politechnika Warszawska została poproszona o zapewnienie odpowiedniego poziomu cyberbezpieczeństwa opracowywanego systemu, szczególnie w ramach trzech aspektów - skanowania portów (port scanning), zabezpieczenia przed atakiem Denial of Service (DoS) oraz podsłuchem (sniffing).

Architektura systemu opracowanego w ramach projektu IoRL składa się z wielu komponentów, które koncentrują się wokół sieci 5G. Trzonem sieci 5G jest technologia SDN i to właśnie na jej podstawie zbudowano narzędzie nazwane Integrated Security Framework (ISF) dedykowane dla projektu IoRL. W poniżej przedstawionej publikacji zawarto przede wszystkim nowatorskie sposoby przeciwdziałania atakom skanowania portów (TCP SYN Scan) oraz ataku DHCP Starvation, który jest przykładem ataku DoS (opisane w 2.2).

Artykuł "Network Threats Mitigation Using Software-Defined Networking for the 5G Internet of Radio Light System", opublikowany w czasopiśmie "Security and Communication Networks" jest znacząco rozszerzoną wersją publikacji konferencyjnej [41], która przedstawiała jedynie koncepcję systemu oraz symulacyjnie otrzymane wyniki. Poniższa, rozszerzona wersja artykułu, przedstawia wyniki uzyskane drogą praktycznych eksperymentów z wykorzystaniem technologii SDN. Wyniki te pokazują, że zaproponowane rozwiązania są wydajne oraz skuteczne.

Głównymi zaletami przedstawionego mechanizmu przeciwdziałania skanowaniu typu 'TCP SYN' w przeciwieństwie do innych publikacji są: wprowadzanie mniejszych opóźnień w operacjach na pakietach, użycie przesuwnego okna do zliczania podejrzanych połączeń oraz koncentracja na adresach IP zamiast MAC.

Z kolei w przypadku ataku DHCP Starvation głównymi atutami rozwiązania jest podejście elastyczne i skalowalne, idealnie pasujące do dynamicznych środowisk wirtualnych czy chmurowych. Przeprowadzono także badania na ponad 97% ówcześnie wykorzystywanych systemów operacyjnych. Nieszablonowe podejście, polegające na celowym odrzucaniu specyficznego ruchu (złośliwego lub nie), dało zaskakująco dobre rezultaty.

Oba zaproponowane rozwiązania wykorzystują technologię SDN nie tylko do detekcji, ale także do zatrzymania złośliwych działań, znacznie zmniejszając ich skuteczność. Zaproponowana metoda wykrywania ataku TCP SYN Scan odznaczała się 99% skutecznością. Z kolei w przypadku ataku DHCP Starvation, wykryto wszystkie próby wysycenia puli adresów, wprowadzając jednocześnie niewielkie opóźnienie w pobieraniu adresu IP przez postronne urządzenia (średnio 2,5s).

Warto także wspomnieć, że narzędzie ISF, stworzone specjalnie na potrzeby podnoszenia

[15] https://iorl.5g-ppp.eu

bezpieczeństwa w ramach projektu IoRL, ma budowę modułową i możliwa jest jego łatwa rozbudowa w przyszłości (jak miało to miejsce w przypadku mechanizmów zaproponowanych w [154], [47] czy [22]). Dodanie kolejnych modułów adresujących inne zagrożenia nie powinno stanowić problemu.

Zaproponowane rozwiązania mają także swoje niedoskonałości. Przedstawiony mechanizm obrony przed skanowaniem chroni jedynie przed jednym z wielu typów skanowań sieciowych. Z kolei rozwiązanie adresujące atak DHCP Starvation zostało przetestowane względem jednego z dostępnych narzędzi. Kolejne badania pozwolą na dalsze testowanie i rozwój przedstawionych mechanizmów.

Liczba cytowań publikacji na dzień 24.03.2022 w bazie Google Scholar wynosi 14, w Scopus 13, natomiast w WoS 10.

Hindawi
Security and Communication Networks
Volume 2019, Article ID 4930908, 22 pages
https://doi.org/10.1155/2019/4930908

WILEY | Hindawi

Research Article

Network Threats Mitigation Using Software-Defined Networking for the 5G Internet of Radio Light System

Krzysztof Cabaj, Marcin Gregorczyk, Wojciech Mazurczyk, Piotr Nowakowski, and Piotr Żórawski

Warsaw University of Technology, Warsaw, Poland

Correspondence should be addressed to Wojciech Mazurczyk; wmazurczyk@tele.pw.edu.pl

Received 12 October 2018; Accepted 2 January 2019; Published 21 February 2019

Guest Editor: Bogdan Księżopolski

Currently 5G communication networks are envisioned to offer in a near future a wide range of high-quality services and unfaltering user experiences. In order to achieve this, several issues including security, privacy, and trust aspects need to be solved so that the 5G networks can be widely welcomed and accepted. Considering above, in this paper, we take a step towards these requirements by proposing a dedicated SDN-based integrated security framework for the Internet of Radio Light (IoRL) system that is following 5G architecture design. In particular, we present how TCP SYN-based scanning activities and DHCP-related network threats like Denial of Service (DoS), traffic eavesdropping, etc. can be detected and mitigated using such an approach. Enclosed experimental results prove that the proposed security framework is effective and efficient and thus can be considered as a promising defensive solution.

1. Introduction

With the great success and development of 4G mobile networks, it is expected that the 5th generation wireless systems (in short 5G) will be a continued effort toward rich ubiquitous communication infrastructure, promising wide range of high-quality services. It is envisioned that 5G communication will offer significantly greater data bandwidth and huge capability of networking resulting in unfaltering user experiences for (among others) virtual/augmented reality, massive content streaming, telepresence, user-centric computing, crowded area services, smart personal networks, Internet of Things (IoT), smart buildings, smart cities, to name just a few.

The 5G communication is currently in the center of attention of industry, academia, and governments worldwide. 5G drives many new requirements for different network capabilities. As 5G aims at utilizing many promising network technologies, such as Software Defined Networking (SDN), Network Functions Virtualization (NFV), Information Centric Network (ICN), Network Slicing, Cloud Computing, Multi-Access Edge Computing (MEC), etc. and supporting a huge number of connected devices integrating above-mentioned advanced technologies and innovating new techniques will surely bring tremendous challenges for security, privacy, and trust. Therefore, secure network architectures, mechanisms, and protocols are required as the basis for 5G to address this problem and follow security-by-design but also security by operations rules. Finally, as in 5G networks even more user data and network traffic will be transferred, the big data security solutions assisted by AI techniques should be sought in order to address the magnitude of the data volume and to ensure security concerns at stake (*e.g.,* data security, privacy, etc.).

Internet of Radio Light is a Horizon 2020 project which aims at developing an architecture for smart buildings [1], supermarkets, museums, or even train stations [2], using a 5G Radio Light multicomponent carrier, Frequency Division Duplex (FDD) broadband system consisting of a VLC (Visible Light Communication) downlink channel in the unlicensed THz spectrum and mmWave up/downlink channels in unlicensed 30-300 GHz spectrum. It allows wireless communication networks to be deployed in buildings that can provide bitrates greater than 10 Gbit/s, latencies of

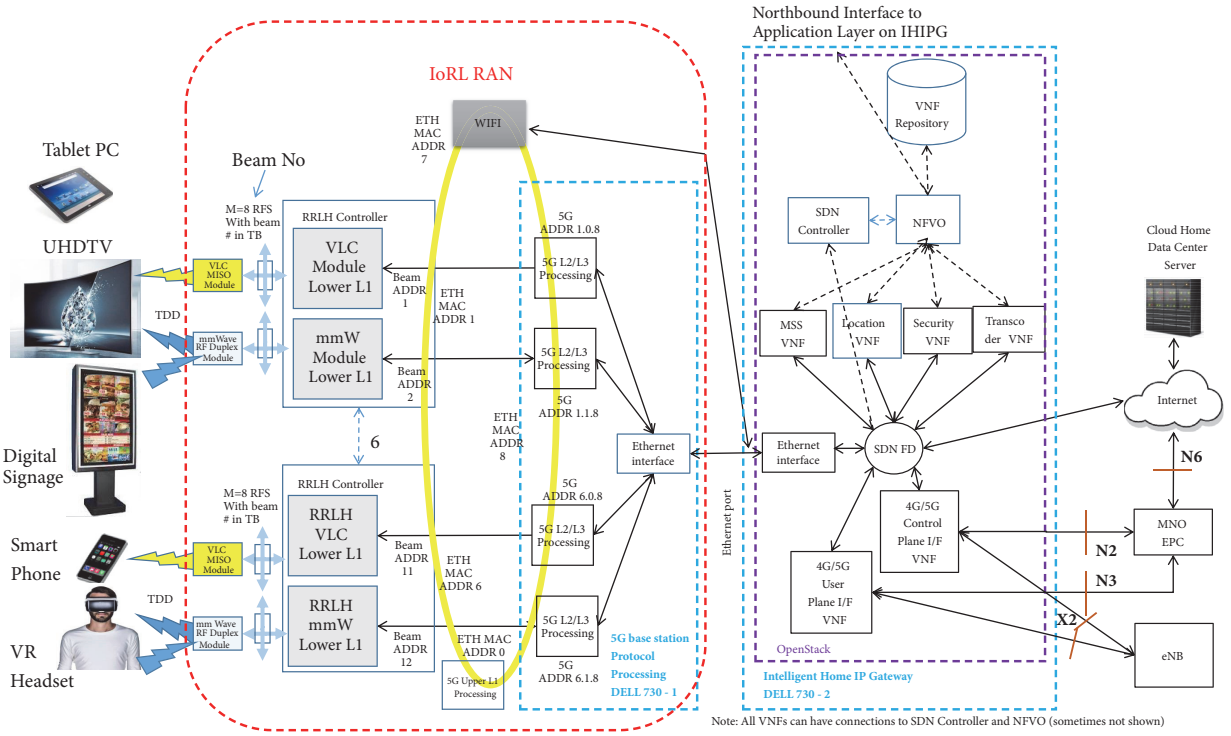


Figure 1: IoRL architecture.

less than 1 ms, and location accuracy of less than 10 cm, whilst reducing EMF levels and interference, lowering energy consumption at transmitter/receiver, and increasing User Equipment (UE) energy battery lifetime.

The IoRL system part responsible for all network layers as well as upper layers processing utilizes SDN in Intelligent Home IP Gateway (IHIPG), which is shown in Figure 1. This element allows network service providers to develop functionality like security monitoring, energy saving, location sensing, network slicing, lights configuration, video and network transport configuration, and network security applications. Moreover, it provides the means to locate network operations and management functions between the Intelligent Home IP Gateway (HIPG) and the Cloud Home Data Centre (CHDC) server in a configurable way to meet the different OPEX and CAPEX needs of different Mobile Network Operators (MNOs). Furthermore, it does not require MNO approval for deployment. This step change in performance and flexibility is a very attractive solution for retailers since it will increase their ability to promote their brand and products thereby improving their profitability, which will incentivize them to raise capital to finance the upgrade of their building network infrastructure.

However, apart from obvious benefits that such a system can offer to the users some challenges and issues must be addressed first. As IoRL integrates various networking technologies, i.e., VLC, mmWave, SDN, WLAN, and eNB/HeNB, and each of them is characterized with a specific set of characteristic features, potential security threats and vulnerabilities still are often not completely resolved and still need addressing.

In this paper, we want to take a step towards dedicated SDN-based integrated security solution for the IoRL system which will be able to monitor network traffic and is capable of detecting and mitigating various types of network threats by means of reconfiguration of the SDN switches flow tables. Thus, the hostile traffic will be simply removed as soon as possible from the network and in an ideal case it never reaches the victims. We initially demonstrated the potential of the proposed concept in [3] where the simulation results were presented for thwarting TCP SYN scanning attacks. However, it must be noted that in this paper we present experimental results using proof-of-concept implementation for two types of threats. Apart from detection and mitigation of the scanning activities, we evaluate also DHCP-related threats like DHCP pool exhaustion attack which is a type of the Denial of Service (DoS) threat, traffic eavesdropping, or rogue DHCP server placement.

Thus, the contribution of this paper can be summarized as follows: we propose an integrated security framework for the IoRL system which is based on SDN technology as well as based on the examples of the chosen attacks; we show that such a solution has potential and it is effective and efficient.

The rest of the paper is organized as follows. Section 2 presents the general overview of the IoRL system architecture and its main threats are characterized. Next, in Section 3 the related work is presented. In Section 4, the network threats including scanning, DoS, and traffic eavesdropping

are described and their impact on the communication networks security is emphasized. Then in Section 5, we define the architecture of the SDN-based integrated security framework and describe the inner workings of the designed and implemented detection and mitigation modules. Next in Section 6, we detail how the experiments have been conducted and in what test-bed, while in Section 7 we evaluate the proposed security modules and present their optimal configuration parameters. Finally, Section 8 concludes our paper and outlines potential directions for future work.

2. IoRL System Description

2.1. Overall System Architecture. The IoRL system consists of the three main parts: user equipment (UE), radio access network (RAN), and NFV/SDN part. The developed system will be connected to the outside world using Internet or directly to the mobile operator using native 4G/5G protocols. As mentioned the overall architecture is presented in Figure 1. During the test scenarios as UE we will use currently available electronic devices, for example, SMART TV sets, tablets, or mobile phones, with custom device connected by USB used for receiving VLC and receiving and transmitting mmWave signals. The RAN part is responsible for providing wireless communication for both VLC and mmWave technologies. These functions are provided by the so-called RRLH (Remote Radio Light Head) which are built in lights roses. The last element of the IoRL subsystem is responsible for all layer 3 functions. Due to usage of NFV paradigm all new functions of the IoRL system can be added as additional Virtualized Network Functions (VNFs). In the remainder of this paper we focus on presenting details and results on the proposed security monitoring functions implemented as SDN application.

2.2. IoRL Threats. As already mentioned, IoRL system integrates various concepts and networking technologies, i.e., VLC, mmWave, SDN, WLAN, and eNB/HeNB, and each of them has a specific set of characteristic features and potential security threats and vulnerabilities that still are often not completely resolved and still need addressing. For such a heterogeneous system, a careful threat analysis is needed in order to identify the most important security hazards. In more detail, the potential IoRL system threats can be assigned to one of the following groups:

(i) *IoRL end user devices-related threats (user-specific)*: these threats include, *e.g.,* Man-in-the-Middle (MitM) attacks performed by the malicious users in order to influence legitimate IoRL users or their devices in order to capture their sensitive data (like credentials), to tamper with the legitimate users' communication (in order to impersonate them), or to disrupt it via Distributed Denial of Service (DDoS) attacks. Finally, some privacy-related attacks can be envisioned where the attacker tries to gain, *e.g.,* targeted users location information, their movements, and/or habits patterns.

(ii) *IoRL infrastructure-related threats (component-specific)*: these include, *e.g.,* attacks on crucial points of the IoRL architecture in order to overload or impersonate them, *e.g.,* by performing DDoS attacks on the SDN or RRLH controllers or eavesdropping and then spoofing their communication. It is also worth noting that these types of attacks can be launched by malicious IoRL users (insider threat) or by remote attackers (remote threat) residing somewhere in the Internet or in the vicinity of the IoRL system components, for example, mmWave or VLC receivers and/or transmitters.

(iii) *IoRL-related threats (architecture-specific)*: such attacks can be possible or could be amplified because of the heterogeneous nature of the proposed system involving coexistence and integration of various networking technologies. Due to this fact, some unexpected interactions or vulnerabilities can be discovered. For example, due to integration of diverse types of technologies, some simplifications may be needed to ensure their cooperation which may lead to the situation that no security is provided or there are security mechanisms; however they are not compatible.

To address above-mentioned threats, we design and implement the security monitoring and management system, i.e., Integrated Security Framework (ISF) that is tailored specifically to the requirements of the IoRL system. Based on the SDN controller, a centralized system for security monitoring and manageability providing near real-time awareness of network incidents status and effective enforcement of security policy is developed. The details of the initial prototype of the SDN-based security solution are presented in Section 5. It must be also noted that in this paper we focus on the detection and mitigation of the scanning and DHCP-related threats which we treat as examples to prove that the proposed solution is effective and efficient.

3. Related Work

Software-defined networking (SDN) is recently one of the most important and promising networking paradigms [4]. The main advantage of SDN is the opportunity to decouple the control and data planes, which means that the underlying network infrastructure is abstracted from the applications. In result, the network can be managed in a logically centralized way. Apart from many potential applications of SDN [4], lately it has become an interesting option to provide security in a more flexible and effective manner in the current communication networks [5].

Moreover, the majority of network device manufactures are nowadays already supporting SDN with their physical and virtual equipment using the OpenFlow protocol. SDN is standardizing the management of heterogeneous networks. Applications written for the SDN controller will work without additional adjustments on various devices supporting SDN in both physical and virtual environments. Based on the above-mentioned facts, we decided to use SDN as a base for our detection solution presented in this paper.

When it comes to the SDN-based solutions tailored to security purposes, the first work that proposed a general SDN-based anomaly detection system was put forward by Mehdi et al. in 2011 [5]. The authors showed how four traffic anomaly detection algorithms could be implemented in an SDN context using OpenFlow compliant switches and NOX controller. The obtained experimental results proved that these algorithms are significantly more accurate in identifying malicious activities in home networks when compared to the ISP. Further, other researchers utilized SDN to detect network attacks [6] or to monitor dynamic cloud networks [7]. Recently Zhang et al. published an interesting survey on the security-aware measurement in SDN [8].

Several papers deal with SDN-based malware detection. Jin and Wang [9] analyzed malware behavior on mobile devices. Based on the acquired knowledge, they proposed several detection algorithms for mobile malicious software and implemented them using SDN. Their system was able to perform real-time traffic analysis and to detect malicious activities based only on the packets used for connection establishment. In [10], the authors designed and developed an SDN-based architecture specialized in malware analysis aimed at dynamically modifying the network environment based on malicious software actions. They demonstrated that this approach is able to trigger more malware events than traditional solutions.

Few papers focused on utilizing SDN in order to create a dedicated security solution against ransomware [11, 12]. The authors of [11] introduced two approaches that took advantage of the fact that without successful communication to the Command & Control (C&C) server the infected host is not able to retrieve the public key and, as a result, it cannot start the encryption process. Both methods rely on dynamic blacklisting of the proxy servers used to relay communication between an infected machine and the C&C server. On the other hand in [12], it is shown that SDN-based detection approach which uses the characteristics of the ransomware communication can be effective against two crypto ransomware families, namely, CryptoWall and Locky. Based on the results presented in this paper, it has been concluded that an analysis of the HTTP message sequences and their respective content sizes is enough to identify such threats with high accuracy.

Recently, research is focused more on mitigating DoS attacks directed against both infrastructure of SDN itself and devices behind it [13–18]. However, below we review the papers that are most related to our contribution, i.e., [19–22].

In [19], Shin et al. proposed AVANT-GUARD solution which helps to avoid saturation of the control plane. This is achieved by adding intelligence to the data plane and installing actuating triggers for responsiveness. This results in the reduced data and control planes' interactions thus introducing minimal overhead. In [20], authors, in the protocol independent solution named FLOODGUARD, utilize proactive flow rule analyzer which is based on the dynamic application tracking and packets' migration. In case of attack detection, it will temporary cache the table miss packets. Proof of concept and its evaluation showed the effectiveness and only minor overhead. Scalable and protocol independent framework FloodDefender proposed in [21] implements three novel modules: the table-miss engineering to prevent the communication bandwidth from being exhausted by offloading traffic to the neighbor switches; packet filter to identify attack by using B+ tree traffic; and flow rule management to search and purge unneeded entries in the flow table. A prototype was implemented in both software and hardware. More recently in [22] a solution named StateSec is proposed which offloads some of the controller's functions to the switch, resulting in the so-called stateful SDN. By utilizing entropy-based algorithms, different types of DDoS attacks can be detected. Authors also claim that such a technique can detect port scanning activities; however no experimental results have been provided.

3.1. SDN-Based Detection and Mitigation of the Scanning Attacks. Finally, there are several papers related to the SDN-based detection and mitigation of scanning attacks. In [23], Yuwen et al. proposed a probability-based delay scheme to mitigate possibility of the SDN network scan using packets response time difference. IP Hopping (IPH) in [24] utilizes transparent changing of the IP addresses in the active communication of two hosts in SDN. In addition of adopting application layer gateway, this can protect against network scanning with support for multichannel protocol. Shirali-Shahreza and Ganjali in order to identify horizontal scanning in [25] decouple protecting and decision-making element of the firewall and offload the intelligence to the cloud for inspections. FleXight was used for selecting parts of the traffic needed for decision-making. This solution is scalable but requires managing third party, i.e., cloud infrastructure. In [26], the authors used combined algorithm based on fuzzy logic for the detection of port scanning, which gives better results than security algorithms used alone. This also results in the reduced number of lines of code.

Last but not least in [27], the authors proposed solution named SLICOTS, which counts TCP SYN connections per host. If their number exceeds fixed threshold further traffic is denied. As this solution is the closest to the one proposed in this paper, we present the main differences between SLICOTS and our scanning protection module below.

In contrast to the existing research, our module uses IPv4 addresses instead of MAC addresses (used by SLICOTS), utilizes a sliding time window, and does not introduce delays in packet forwarding. Moreover, it introduces a possibility to detect and block malicious traffic originating from both inside and outside the monitored network. This is not possible with SLICOTS, as MAC addresses are utilized in point to point Layer 2 connections and packets passing, e.g., through the router network interface do not preserve the original MAC addressing. As such, all packets originating from the Internet would be incorrectly marked as originating from the router gateway and a single port scan would erroneously trigger SLICOTS which will result in shutting down the Internet connectivity for the entire LAN. Furthermore, our module improves upon SLICOTS by introducing a sliding time window for the observed "pending connections" as our experimental results show that over time failed connections tend to accumulate even for the benign traffic and absolute

counter of the failed connections would erroneously trigger detection event. Instead of keeping an absolute amount of failed connections, we keep a rate of failed connections over a given time. Finally, our module has the possibility of having superior performance of both packet forwarding times and CPU usage, as we operate on a copy of packets without interfering with normal packet forwarding and the most CPU intensive operations are optimized by utilizing an index and are performed periodically instead of on per packet basis.

3.2. SDN-Based Detection and Mitigation of the DHCP Starvation Attacks. To the best of our knowledge, there are only few papers related to DHCP security and SDN. In [28], it has been proposed that specialized, internal DHCP server is no longer needed. All DHCP requests and offers are handled by the controller and any other DHCP traffic is ignored. Unfortunately, this solution does not scale well as the controller has to maintain IP leases and perform DHCP transactions, which require additional memory and CPU resources. This could affect the whole network performance and security. Network Flow Guard (NFG) was proposed by the authors in [29] which aims to protect against rogue DHCP servers as well. This solution is more scalable as it uses existing DHCP server in the infrastructure instead of implementing its functionality in the controller. Whitelisting of the valid DHCP offers is implemented in the NFG. Every time, any server (rogue or legitimate) sends the DHCP offer, it will be examined and only whitelisted offers will be allowed and forwarded to the client. Unfortunately, examining every offer causes scalability issues as well. Moreover changing DHCP server configuration will require additional steps to be taken to reconfigure NFG, which can lead to the unexpected behavior. Last but not least in [30], authors propose solution against rogue DHCP server, starvation, and lease attack (which is another form of starvation attack). It is based on performing inspection of every DHCP packet and comparing it with hardcoded whitelist. Unfortunately no proper research methodology or experimental results were provided. All proposed solutions are not suitable for scalable and dynamic virtualized or cloud environments. If any change to the network infrastructure is made (like additional hosts, DHCP reconfiguration change, etc.), without updating applications above, network will behave unpredictably. This can lead to further issues including unstable work, unusability, or exposure to security threats.

When compared with existing works, our DHCP starvation protection solution is based on characteristics of the DHCP traffic for more than 97% of the currently most popular operating systems. DHCP requests that are not coherent with known patterns are dropped. What is important in terms of previous applications, any change to infrastructure, will not need further reconfiguration, which will improve scalability of virtualized or cloud environments. Moreover previous solutions do not cover scenario where the access point can be connected to the SDN-controlled switch and many DHCP requests can come from the same source. In our case, we focus on DHCP transaction ID rather than infrastructure layout or configuration. For the rogue DHCP server protection, our solution is very simple and does not require examining every DHCP offer message. It is based on the fact that DHCP infrastructure (not the configuration) is rather static, so all DHCP offers will be allowed only from the fixed server/port. All other offers (coming from other server or port) will be denied. This does not protect DHCP server from being compromised, but still scales well, as no additional overhead is needed on the SDN controller.

4. Network Threats and Their Influence on Security

In the literature, there exist many potential ways to classify network attacks. One of the most well-known classification is related to how the attacker can affect the passing network traffic or system resources. If the attack attempts to learn or make use of information from the passing network traffic or system but does not affect the transferred data or system resources, then it is called a *passive* attack. Alternatively, an *active* attack relies on altering the traffic flows/system resources or affect their operation.

In this paper, we address both types of threats which are exemplified by the scanning activities, Denial of Service (DoS) attacks, and traffic eavesdropping. In the following subsections, we describe in detail the network threats for which we will then propose detection and mitigation solutions. Obviously the proposed system can be easily extended by adding another modules responsible for thwarting other types of network threats which is out of the scope of this paper.

4.1. TCP SYN Scanning. The scanning activity, sometimes called recon, is one of the oldest and well-known network threats. The main aim of this activity is to find potentially vulnerable machine and enumerate all its services. In most cases, this process precedes actual attacks. Due to this fact, early detection of scanning activity is beneficial, and it gives valuable time to prepare network for further attacks. Moreover, if the security system introduced in the network can early detect and prevent network from full scanning activity, further attacks can be limited. If the attacker cannot find vulnerable machine, it is unable to compromise it.

Network services use listening TCP or UDP sockets on the specified ports in order to accept incoming network connections from the clients. For example, an HTTP server utilizes TCP listening socket on port 80 to accept connections and serve web pages to the clients. This functionality can be exploited by a rogue client who can scan the entire network in search for an open listening sockets in order to enumerate available services. The results of this scan can be used for further exploitation (for example, to find an outdated insecure web server or misconfigured FTP server with a default password).

It must be noted that currently there are many different types of scanning; however, the most popular is TCP SYN scanning which is described in detail below.

In summary, an attacker is attempting to determine the state of every TCP port of the target IP address (65536 ports in total) without establishing a full connection. This is achieved by sending a SYN segment addressed to every port on the

Table 1: Main details of various *nmap* modes.

nmap Mode	mode name	Delay b/w scan pkts.
T0	paranoid	300 s
T1	sneaky	15 s
T2	polite	up to 0.4 s
T3	normal	up to 0.1 s
T4	aggressive	up to 10 ms
T5	insane	up to 5 ms

server. If the server responds with SYN/ACK, it means the port is open. If the server responds with an RST segment, then this indicates that the port is closed but reachable. If there is no response after a specified amount of time, then typically it means that the packet has been filtered out by the firewall and the target port is unreachable.

One of the currently most popular scanning tools is *nmap* (https://nmap.org). It is a network reconnaissance tool commonly used in performing network service enumeration and vulnerability scans. The tool offers many parameters which allow controlling its scanning behavior, *e.g.,* delay between sent packets, response timeouts, parallelism of the scan, *etc*. Although the user can tune settings individually, the author of the tool provides six built-in templates, i.e., T0-T5 modes. These templates allow the user to specify how aggressive the scanning should be, while leaving *nmap* to pick the exact timing values. The main information about each *nmap* mode is included in Table 1.

In this paper, we will utilize *nmap* to generate the real-life scanning activity using its various modes (T1-T5). It must be noted that we have omitted T0 mode in our research. The detection of such scanning activity for which the delay between two consecutive scan packets is 5 minutes is practically impossible, as scanning process of 65536 ports, for only one IP address, would take almost 228 days.

4.2. DHCP-Related Threats. DHCP (Dynamic Host Configuration Protocol), which dynamically allocates IP addresses to machines, is described in RFC 2131 [31]. Successful obtaining of IP address from the DHCP server from the pool of available IP addresses is based on exchange of four DHCP messages (Discover, Offer, Request, and ACK) during so-called transaction (Figure 2).

The first DHCP message (Figure 2(1)) is sent to the broadcast address of the Link layer (i.e., MAC address FF:FF:FF:FF:FF:FF) and the broadcast address of the Internet layer (i.e., IP address 255.255.255.255). This way, if the DHCP server is available on the network, it will always receive such a packet. The DHCP client and server typically use ports 68 and 67 to communicate. Most of the bytes of the DHCP Discover message are set to zero, but among others there is a 4-byte long transaction ID which identifies the flow of packets for this particular transaction. If the DHCP server has an available IP address in the pool, it replies with the DHCP Offer message (Figure 2(2)) with the same transaction ID. It is sent directly to the MAC address and IP address of the client and contains the proposed IP address with a netmask and a lease time as minimum, but can also provide networking details such as gateway address, DNS server addresses, domain name, TFTP server, and a filename used during boot, etc. The client, based on the offer from the DHCP server, requests such data using DHCP Request message (Figure 2(3)), but again using broadcast addresses and the same transaction ID. Finally the server sends acknowledgement message ACK (Figure 2(4)). At this point, the client can use offered IP address and the server removes it from the pool of available IP addresses.

Because the DHCP protocol has been designed only to ease maintenance of hosts in the network, no effort was done to secure it properly. There are several issues related to the mechanism described above:

(i) There is practically no possibility of implementing authentication, authorization, encryption, or use of the trusted third party.

(ii) The server has no possibility of finding out if the DHCP Discover and Request messages are sent from the legitimate user or an attacker.

(iii) It is quite easy to fabricate DHCP messages.

(iv) The pool of available addresses has limited size; thus when exhausted, no new clients are able to obtain new IP addresses. In such situation, new DHCP Discover messages will be ignored by the server resulting in a successful DoS attack.

RFC 2131 also states that DHCP clients are responsible for all messages retransmission (for example, in the case of packet losses occurrence). The client must adopt a retransmission strategy that incorporates a randomized exponential backoff algorithm to determine the delay between retransmissions. During one DHCP transaction, the first retransmission (for example, DHCP Discover message) should take place after 4s (±1s) timeout since sending the first DHCP packet. The next retransmissions will occur after 8s (±1s) and 16s (±1s) since the previous packet (the second and the third packet). The last timeout of 32s (±1s) will finish overall transaction timeout (containing 4 DHCP Discover packets). If the transaction was unsuccessful during this time (no IP address has been successfully obtained), new transaction will be started after some administrative timeout. Unfortunately, different operating system families implement this mechanism nonidentically. This will be explained in Section 6.3.

4.2.1. DHCP Pool Exhaustion Attack. The time required for exchanging four DHCP messages (i.e., Discover, Offer, Request, and Ack) in order to successfully obtain IP address depends on the speed, load, and delays of the network but can be considered as short (ca. 1-3 ms). An attacker can use this mechanism to perform a DoS attack which is called *DHCP pool exhaustion or DHCP starvation attack*. It must be noted that, for example, the DHCP pool containing subnetwork with /24 prefix (with 254 usable IP addresses) can be exhausted by the attacker in less than 1 second which results in a rapid and efficient DoS attack. As a result, the communication of the entities that depends on DHCP IP

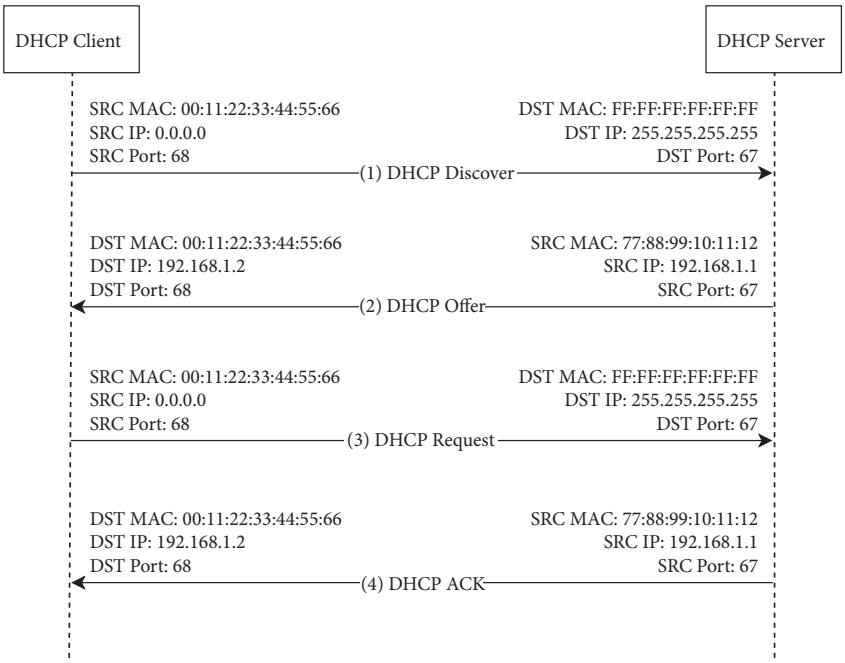


Figure 2: Typical DHCP messages exchange between the client and the server.

addresses allocation will be completely paralyzed. This is especially the case for the dynamic and scalable environments (like clouds or public networks with open access).

4.2.2. Rogue DHCP Server. Rogue DHCP server is an entity on a Local Area Network that is providing IP addresses for other hosts but is not under administrative control of the network or infrastructure legitimate owner. Starting or connecting additional DHCP server often can be just a mistake leading to misbehaviour of network operation and network connectivity problems (like connecting to other hosts on the same LAN). Alternatively, if the rogue DHCP server is under attacker's control, this can lead to very serious security issues. Rogue DHCP offer can contain default gateway and DNS server settings pointing to the rogue server itself. This will lead to "man in the middle attack" or eavesdropping as all traffic will be routed through malicious host. Other solutions for protecting network like IDS/IPS can be useless as affected DHCP client traffic will not be reaching these security devices at all.

4.2.3. MAC Address Spoofing and Traffic Eavesdropping. Currently one of the fairly frequent cases of network attack involves spoofing of packets/frames source address, where attacker conceals his identity within the network in order to attack either other hosts or the network infrastructure itself. In the LAN, a learning network switch in order to perform its function must keep track of which device (identified by its MAC address) is connected to a physical interface port. This information is usually kept in memory in a forwarding/switching table of a fixed size and updated whenever a frame with a new MAC address is encountered. Because MAC address can be spoofed, it is possible to fill the forwarding table completely by intentionally sending bogus frames to the switch. This in turn forces the switch to broadcast all frames to all ports (except the one on which they were received) which in result effectively transforms switch into a simple network hub. In such a case, the attacker is able to sniff all network traffic that is reaching this intermediate device.

5. SDN-Based Integrated Security Framework

The concept of the SDN-based integrated security framework utilizes a programmable network switch supporting OpenFlow protocol and a controller software running on a dedicated machine. The programmable switch handles packets switching using a rule table which is managed by the controller. The rules stored in the table define the behavior of the switch, i.e., whether to drop or to forward the packet to the specified physical port. If an incoming packet does not match any of the already installed rules, it is temporarily saved in the switch buffer and the controller is notified to conduct its inspection. The notification consists of a unique packet identification number, the switch network interface (from which the packet originated), and the initial bytes of the packet containing both network and transport protocols' details. On demand, the packet summary can be extended to contain the whole packet payload. Then the controller software can decide whether to drop, let the packet through, change header fields, or add/remove VLAN tags, *etc.*, optionally installing a new flow rule on the switch so the action is repeated automatically on similar packets without the controller's involvement.

Unlike standard networking solutions, this approach allows for separation of the data and control planes; all management data (known routes, destinations) is collected, kept, and processed by only one device (the controller). This design allows the controller to pose as a central network monitoring device, capable of, e.g., correlating various events and providing additional security features.

Usually the very first couple of packets from the particular communication reveal its intent. Traditional firewall software or hardware uses this feature to block simple attacks, but often lacks the ability to use a contextual detection. Having a benefit of the centralized packets inspection we are able to detect also more complex attacks performed across the entire network which usually would be left intact by a traditional firewall. These patterns are frequently present in the reconnaissance phase of the attack.

For example, an SDN controller can monitor the initiated TCP connections per network client. This is performed in order to detect possible TCP port scanning activity using *nmap* tool; benign TCP connections are usually initiated with a high rate of success and a relatively stable number of connections per hour. However, a TCP port scan can be potentially distinguished from the regular traffic as it typically manifests itself in large bursts of initiated TCP connections.

In our study, we use POX SDN controller which supports a modular architecture. Our security service is implemented as a set of POX modules running alongside a built-in switch/router module, which handles normal packet forwarding. Each module implements a set of event handling routines, in particular an incoming packet event, which is triggered when a packet is sent to the controller. The event handler receives a copy of the packet, as illustrated in Figure 3. This way our modules act as a set of packet filters that are executed when a new network connection is initiated. Each specialized module keeps track of every network connection, packet contents, and host activity, *i.e.*, in line with its functionality.

In the following subsection, we will describe in detail the security modules that were designed and developed especially for the purpose of this paper and they are focused on detection and mitigation of scanning activities, DoS attacks, and traffic eavesdropping.

5.1. Scanning Detection and Protection Module. Scan protection module works by tracking the state of each TCP connection in the traffic handled by the switch.

When a client attempts to initiate a TCP connection (a SYN packet is sent), the controller creates a description object (IPv4 addresses, ports) and sets its state to "pending". This state will not change until a full TCP handshake is completed, in which case the status of this connection changes to "connected". When the connection is closed (either via RST or FIN flag) the description object is removed.

By counting the number of "pending" connections per client IP address, we should be able to distinguish a scanning machine from a benign one. Because even legitimate TCP connections sometimes fail, we focus not on a raw number of "pending" connections, but on the *rate* of their appearance—we count the connection attempts occurring

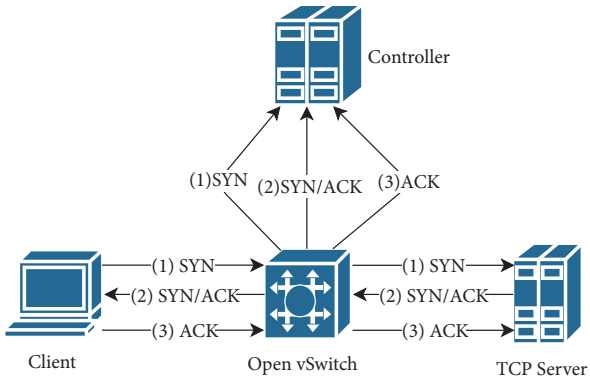


Figure 3: Packet flow of a TCP handshake in our experimental environment.

within a predefined time window (characterized with the parameter "window size" denoted with W_s). If this number exceeds a certain value (i.e., "threshold", T), that particular host (client IP address) is probably trying to scan the network—the controller instructs the switch to drop packets from this IP address for a configurable amount of time (i.e., "ban time", B_t).

The module works as follows (Figure 4):

(1) Install an OpenFlow rule on the switch that sends a copy of all TCP traffic to the controller (this is done due to performance reasons and the original traffic is forwarded in an uninterrupted manner).

(2) Create two hash maps in memory:

(i) *all_tcp_connections<FlowDescription, TCPState>* contains a summary of all tracked TCP connections (IPv4 addresses, ports, and connection states).

(ii) *per_ip_tcp_connections<IPv4Address, <FlowDescription, TCPState≫* organizes *all_tcp_connections* map per IPv4 address in order to optimize lookup performance. FlowDescription is an order-invariant hash structure containing source and destination IPv4 addresses and TCP ports. TCPState is a structure containing a description of a TCP connection with the connection state, the connection creation timestamp, and the last matching packet timestamp.

(3) When a TCP segment is sent to the controller:

(a) Determine IPv4 addresses (source and destination) and TCP ports (source and destination).

(b) Look up the TCP connection in *all_tcp_connections*. If it does not exist, create a new entry and store it in *all_tcp_connections* and *per_ip_tcp_connections* (marking the source IPv4 address as the connection origin). Mark the connection state as `Unknown`. Set a connection creation timestamp. Determine which IPv4 address initiated the connection to deduce whether a packet is incoming or outgoing.

(c) Analyze TCP flags in the packet and the connection state. If the TCP state is:

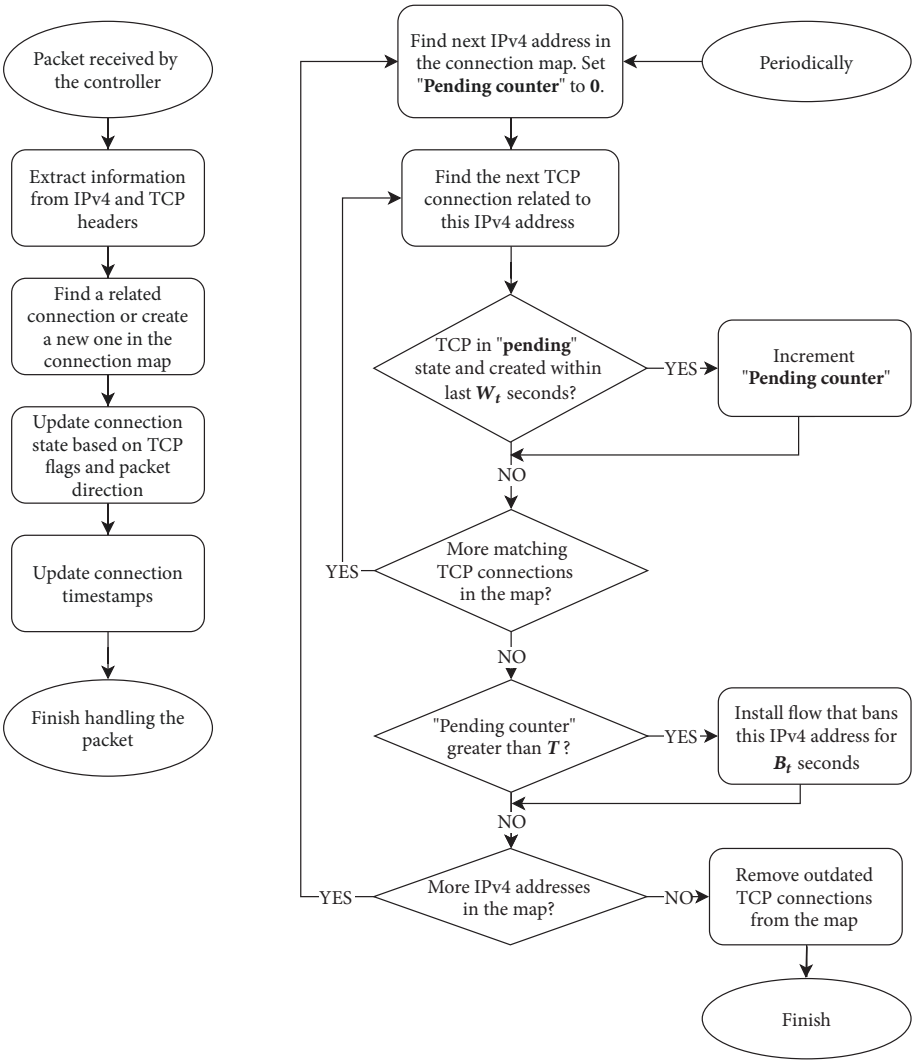


Figure 4: The TCP SYN scanning detection algorithm.

(i) `Unknown`:

(A) On outgoing `SYN` flag: mark connection as `SynSent`

(ii) `SynSent`:

(A) On incoming `SYN/ACK` flag: mark connection as `SynAckReceived`

(B) On incoming `RST` flag: mark connection as `RstReceived`

(iii) `SynAckReceived`:

(A) On outgoing `ACK` flag: mark connection as `Connected`

(iv) `Connected`:

(A) On `RST` or `FIN` flag: mark connection as `Closed`

(v) `Closed`:

(A) No action

(vi) `RstReceived`:

(A) No action

(d) Update the timestamp of the last received packet in the TCP connection.

(4) Periodically iterate over *per_ip_tcp_connections*. For each IPv4 address origin,

(a) Count the number of "pending connections". The connection is considered as "pending" when it is in `SynSent`, `SynAckReceived`, `RstReceived` states and the connection has been created within the last W_s seconds.

If the "pending connections" number exceeds T, mark the IPv4 address as banned and install a rule on the

Open vSwitch that drops all packets containing that IPv4 address for B_t seconds but still send a copy of the packet to the controller.

As already mentioned in Section 3, the main advantages of our approach when compared with the other state-of-the-art SDN-based TCP SYN attack detection methods (especially SLICOTS [27]) include

(i) utilization of IPv4 addressing instead of MAC addressing which gives more flexibility and is not prone to simple DoS attacks which can result in losing connectivity of the whole LAN,

(ii) usage of the sliding time window to keep a rate of failed connections over a given time instead of keeping an absolute amount of failed connections,

(iii) superior performance in terms of limited delays in packet forwarding.

5.2. Detection Module for the DHCP Pool Exhaustion Attacks. The second module limits the number of IP addresses that can be leased from the DHCP server. Exhausting the pool of available IP addresses will lead to successful DoS attack on the DHCP server; thus no new DHCP clients will be able to obtain IP address and thus they will be unable to perform any type of IP-based communication.

The most trivial solution to this problem would be to block this type of attack by counting the number of active leases per interface and provide an active response in a form of blocking of further DHCP traffic on that interface when a certain threshold is reached. Different interfaces can have a varied number of expected clients, so the threshold can be also set per interface. It must be noted that such a solution significantly increases the time needed to perform a complete DHCP exhaustion attack thus making it unfeasible (especially when the attacker has access only to a single interface). Usage of the DHCP protocol by legitimate devices with IP address already assigned is very rare (only for lease renewal or freeing the address), so typically they will be not affected. However, it should be emphasized that while blocking DHCP traffic no new device connected to the affected port will be able to obtain an IP address. Paradoxically, if the wireless access point (AP) is connected to a port which has such a security solution active, the result can be exactly opposite. In this case, the attacker does not need to make an attempt to completely exhaust DHCP pool as he can only request few IP addresses, and after reaching the assumed threshold, all DHCP-related traffic will be denied even for the legitimate clients. Thus this will appear as a DoS attack for other legitimate users within the AP range, but the DHCP server will be up, fully operational, and not affected by the attacker. Therefore, a more sophisticated solution is required and has been proposed in this paper, which is based on three observations:

(i) DHCP server is not able to find out if the DHCP Discover and Request messages are sent from the legitimate user or an attacker.

(ii) In case when the DHCP message is lost/blocked, the DHCP clients are responsible for all messages retransmissions.

(iii) Retransmission strategy of the DHCP clients can be measured and used as a security benchmark.

Considering above the designed module requires the following steps to complete:

(1) Create a structure in the memory with information related to the DHCP transaction ID, number of packets sent during particular DHCP transaction, delays between packets retransmission, and MAC address of the device requesting IP.

(2) Install an OpenFlow rule that sends all DHCP Discovery traffic to the controller for the inspection. DHCP traffic volume is only a small fraction of overall traffic; therefore no performance issues nor noticeable resource utilization are expected.

(3) Drop the very first DHCP Discovery packet for every new DHCP transaction, forcing the client to retransmit it after certain timeout.

(4) When the retransmitted DHCP Discover message with the same transaction ID is identified, update the structure in memory, measure the delay between the original DHCP message and the retransmitted one, and check whether this value is in the range of accepted values and make a decision whether DHCP traffic should be

(i) accepted (legitimate user identified),

(ii) dropped again in order to wait for the next retransmission (cannot make a decision yet),

(iii) banned for administrative amount of time (attacker discovered).

It is worth noting that dropping DHCP Discovery messages will introduce additional delay in the process of obtaining IP address for the legitimate clients. This can be considered as a "cost" of implementing such a solution. But on the other hand, without providing effective security solution to alleviate such threats, the attacker can easily and quickly exhaust pool of available IP addresses making the network unusable for legitimate users.

5.3. Rogue DHCP Server Protection Module. The third module is based on the fact that the infrastructure of DHCP server is rather static, i.e., the MAC or IP address and port on the switch, that it is connected to, will not change often. In fact such a change can never happen as it might not be needed. This module, when enabled, will allow only DHCP offers being sent from the legitimate server MAC/IP/switch port and all other offers (possibly coming from the rogue DHCP server but also possibly from the accidentally started, misconfigured DHCP servers) will be dropped and will not reach DHCP clients. On the other hand, reconfiguration of the DHCP service itself can be made quite frequently (e.g., modification to the pool size, adding static reservation,

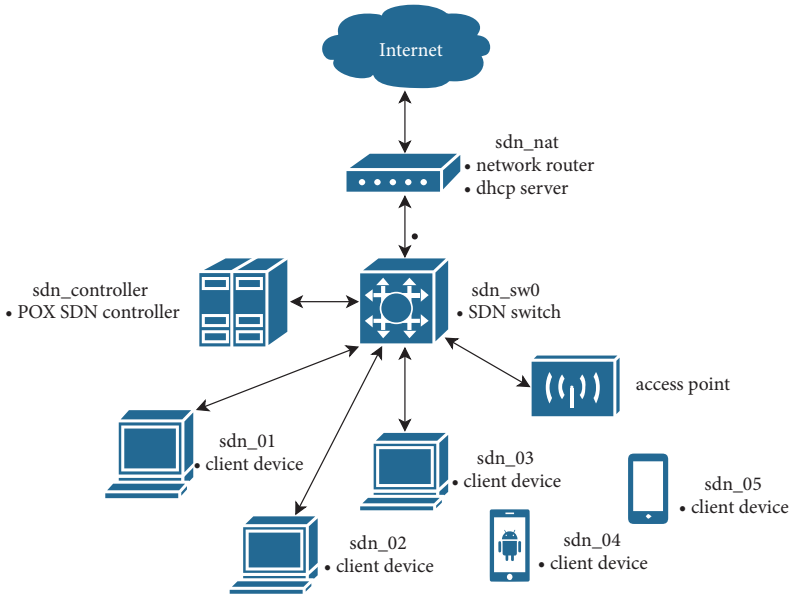


Figure 5: The experimental test-bed.

DHCP parameters changes, which in result would change the DHCP offers). Our module does not examine the content of every DHCP offer; thus it does not introduce any additional overhead for the SDN controller. Additionally, it does not require reconfiguration on the DHCP service configuration update which means it is scalable.

Because the algorithm used for this module is quite simple, we decided not to perform its experimental evaluation. However, we have verified that this module works well and is effective.

5.4. MAC Address Spoofing Detection Module. The fourth module allows only a fixed number of devices (i.e., MAC addresses) that can communicate through a particular interface on the switch. This is done by keeping a limited (configurable capacity) list of "known" MACs, built in first come, first served fashion as the network traffic is routed by the switch. If a packet from an unknown host is encountered and the list is full, the packet is dropped and the filtering rule is installed to prevent further communication.

The total number of addresses across all interfaces should be lower than the maximum capacity of the switch forwarding table. This way the switch will never enter the broadcast mode, making it impossible to sniff passing traffic by the attacker.

In the future, this module can be further extended to support removing hosts from the "known" list if no communication occurs for a given amount of time.

Due to the fact that the algorithm used in this case is quite straightforward, it is pointless to perform experimental evaluation for this module. That is why we have only verified that the module functions correctly and deemed further experiments unnecessary.

6. Experimental Test-Bed and Methodology

As mentioned in the previous sections, for the purpose of this paper, we have developed the dedicated security-related detection and mitigation modules which are a part of the integrated SDN-based security framework. In the remaining of this section, we first introduce the details of our experimental test-bed and outline the methodology of the conducted experiments. Then we focus on presenting obtained results for all designed and developed protection modules.

6.1. Experimental Test-Bed. In our experimental evaluation, the following test-bed has been utilized (see Figure 5). All virtual machines run Ubuntu 16.04 LTS on a machine with Intel Xeon CPU E5-2630 v2 @ 2.60GHz on which Xen v. 4.4 VM Hypervisor is operating. Physical PC *sdn_03* is running Windows 10 operating system. Mobile devices *sdn_04* and *sdn_05* are Samsung S8 with Android 7.0 Nougat and Apple iPhone with iOS 11.4.1. On the *sdn_sw0* an Open vSwitch v. 2.5.2 SDN software switch is installed. Machine *sdn_controller* runs POX, a Python implementation of a controller (github devel, "carp" branch). Although the switch is implemented in software, its host virtual machine is unreachable from inside the created network. This design is applied in order to separate the client and the administrative networks for increased security, *i.e.*, *sdn_sw0* and *sdn_controller* are connected with each other through a dedicated line.

Sdn_01 and *sdn_02* hosts act as client devices emulating the behavior of benign and rogue users depending on the test scenario. *Sdn_0n* and *sdn_nat* form a subnet and *sdn_nat* serves as both a gateway, providing Internet access to the "client" machines and a DHCP server which assigns IP

addresses in this network. *Sdn_01* has a static, fixed IP address reserved in DHCP network range, while *sdn_02-5* gets a dynamic IP address assigned by the DHCP server.

Windows, Android, and iOS (or more generally Mac OS X) cover more than 97% of the mobile operating systems market [32] and that is why they have been chosen for the DHCP exhaustion attack analysis. Moreover Android 7.x, Windows 10, and iOS 11.4 are the most popular versions of these operating systems families [33–35]. It is worth nothing that it is possible to emulate Android on x86 architecture (therefore on virtual machine) but during experiments the behavior of the DHCP implementation was different than when tested on a real device. That is why physical devices were used instead. Additionally, the proposed testbed falls into convergent, hybrid (physical and virtual) assumptions of IoRL architecture.

In our experimental test-bed, the following toolset has been used:

(i) *nmap v.7.01*: as already mentioned, one of the most popular network reconnaissance tools typically utilized in performing network service enumeration and vulnerability scans,

(ii) *Open vSwitch v.2.5.2*: software network switch conforming to OpenFlow protocol acting as the main switch in our test-bed,

(iii) *POX (Carp)*: OpenFlow SDN controller written in Python 2.7 expanded with our security-related modules,

(iv) *PhantomJS v2.1.1*: a headless browser used in the experiment to simulate benign network traffic,

(v) *Dhcpstarv* (http://dhcpstarv.sourceforge.net): an open source tool that implements DHCP starvation attack.

6.2. Experimental Methodology for the Scanning Attacks Detection. The novelty of our TCP SYN scan detection solution is the excellent routing performance in comparison to other solutions [27]. We achieve this by passing network traffic into two parallel streams: the first one is routed directly through the network switch without any interference caused by the switch (except of packet-drop rules), while a secondary stream of data is being copied to the controller for analysis. By utilizing such an approach, our detection solution should not introduce any additional delays in packet forwarding; however it has the disadvantage of being reliant on the reaction time—the controller receives packet summary in parallel to it being routed and usually the decision is made after the original packet already left the switch interfaces.

In order to validate the simulation results obtained in our previous paper [3], we have enabled our detection module on the controller machine and performed extensive experiments that measured both attack prevention effectiveness and performance impact on the network infrastructure and routing capabilities. Similarly to the previous experiments, we have conducted two types of experiments—the first one measured the True Positive detection rate by tracking an attack performed using *nmap* tool on a client machine, while the second one measured the False Positive detection by tracking a benign user browsing the Internet on a client machine.

Our experiments measured the following factors:

(i) True Positive rate: the ratio between the number of blocked *nmap* packets to all packets sent by the tool.

(ii) False Positive rate: the ratio between the number of erroneously blocked packets to all packets sent by PhantomJS.

(iii) Reaction time: the time needed by the infrastructure to detect and block malicious traffic.

(iv) Missed packet rate: the ratio between the number of packets that are a part of the scanning activity and should be blocked by the controller's algorithm, but failed to do so due to implementation factors (i.e., due to slow reaction time).

(v) Resource usage: the consumption of the CPU and RAM resources.

(vi) Forwarding performance: the impact on the forwarding performance introduced by our solution.

Our detection algorithm (see Figure 4) consists of two independent parts: the packet logging, executed every time the controller receives a packet, and the actual detection logic, executed periodically.

Simulations performed for the previous paper [3] were done under the assumption that the detection logic, parametrized by the *threshold*, *windowsize*, and *ban time* (see Section 5.1), can be executed for every packet. However, due to the performance constraints, the detection routine cannot be performed each time we receive a new packet; thus we need to introduce a separate parameter, *check interval* which denotes how often the failed connections limit is enforced. This parameter bears the compromise between controller's CPU usage and reaction time: the more frequent the connection check is, the more the CPU time is required and the faster the reaction time is. However, extreme values of the *check interval* parameter can cause the controller to overload and disconnect from the switch, disabling the scan protection completely.

In order to optimize the performance of the scanning detection, the old TCP connections are removed from the memory. If the TCP connection is marked as `Closed` (a proper sequence of RST/FIN flags was exchanged) or the connection has timed out after 180 seconds, then it is removed. However, we make sure that the memory cleanup actions do not interfere with the detection algorithm—for example, if the *windowsize* parameter exceeds 180 seconds, we extend the timeout time to match the *windowsize* parameter. This allows us to use a smaller *check interval* parameter in most cases.

In our experiments, we evaluated the optimal values for *threshold*, *windowsize,* and *bantime* parameters obtained in our previous research, with a set of different *check interval* settings. As before, we assessed our solution across *nmap's* T1-T5 scans and PhantomJS web browsing. The given *threshold*, *windowsize,* and *bantime* parameters are presented in Table 2

Table 2: *Nmap* detection parameters evaluated during experiments.

Window size [s]	Threshold	Ban time [s]
50	4	16
100	7	16
200	13	16
300	19	16
400	26	16
450	29	16
500	32	16
550	34	16
600	38	16
650	41	16

and the *check interval* parameter equals 0.01, 0.02, 0.05, 0.10, 0.20, 0.50, 1.00, or 2.00 seconds.

In order to create a reliable and repeatable experimental environment, we have utilized Fabric (http://www.fabfile.org/), a high level Python library designed to execute shell commands remotely over SSH. This library provides easy means to orchestrate shell commands across multiple hosts within the control of a single Python script running on a single machine.

The data recording phase of our experiment scenario has the following steps:

(1) Start POX software with our TCP SYN scan detection module on the SDN_Controller.

(2) Start *vmstat* on the SDN_Controller to monitor CPU nad RAM usage on the machine.

(3) Start *tcpdump* on the Input and Output network interfaces of the SDN_Switch machine to monitor the incoming and outgoing traffic on the machine.

(4) Wait 30 seconds for POX to initialize and record machine's idle CPU and RAM usage (i.e., with no traffic).

(5) Generate the network traffic via either (A): TCP scan attack or (B): PhantomJS web browsing by running

  (A) *nmap* scan on the SDN_01 machine that scans all ports of the SDN_Nat01 machine,

  (B) PhantomJS headless web browser on the SDN_01 machine that visits the popular websites on the Internet every 5 seconds.

(6) Wait for 660 seconds (amount chosen to be slightly longer than the widest considered *windowsize* parameter—see Table 2).

(7) Terminate the traffic generating tool.

(8) Wait 300 seconds for the POX controller to perform necessary memory cleanups and return to the idle state.

(9) End *tcpdump* on the SDN_Switch.

(10) Generate simple summary files and synchronize *tcpdump*, *vmstat,* and *POX log* data to the SDN_Nat machine.

The data obtained from the *tcpdump* tool contains on one hand the input packets (generated by the client machine performing the attack) and on the other hand the output packets (filtered by the rules installed on the switch).

We can measure the algorithm's efficiency by comparing these two files, looking for packets that were not forwarded through (i.e., filtered out) by the switch and their overall percentage contribution. On the packets that were not filtered out a delta between arrival and departure timestamp (which are provided by the network adapter) can be calculated which, compared to the delta acquired in the control run (i.e., with the security module disabled), can be used as an estimation of the overhead that our system introduces to the network.

We were also interested in the *reaction time* (between algorithm's decision and it actually making an effect on the switch); however it happened to be difficult to measure using POX and Open vSwitch event logs, as they are subject to many delay factors and could contain major discrepancies between timing reported and the actual packet blocking on the switch. Instead we used a delta between arrival timestamp of a packet that should trigger the detection and arrival timestamp of a first packet that was, in fact, filtered out. This method has an unfortunate effect of providing only an upper bound to the reaction time, as it can only be measured with accuracy up to the incoming packet interval. The most accurate results are acquired with measurements done on *nmap* T5 scan, which sends packets in sub-millisecond intervals. On the other hand, T1 scan has the worst accuracy of up to 15 seconds. The results are acquired by a separate Python script in the postprocessing phase of the experiment.

We assumed that the experimental results should not vary highly in comparison to the simulation results; however we identified the following factors that could skew the results:

(i) *nmap's* timing algorithm: the tool's behaviour (i.e., scan speed) changes if some of its packets are filtered out. During data gathering for simulation, we did not interfere and allowed the tool to run at full speed. In the actual experiment, some of the traffic will be blocked. If the tool reduces scan speed significantly, it might fall below the detection threshold after some time which does not happen in the simulation data.

(ii) CPython's Global Interpreter Lock: due to the limitations of CPython, the POX controller runs as a one-core process with two interleaved threads (one for handling communication with Open vSwitch and one for calculating connection statistics). In some cases, the threads might not meet the time constraints, resulting in detection misses or disconnection, which renders the system defenseless.

(iii) Open vSwitch rule installation delay: every time the controller decides to block specified network stream, there is a significant delay between the moment the decision is made and the moment the rule becomes active.

(iv) Implementation differences between the simulation software and actual implementation on POX controller: although our final implementation closely

resembles the initial code used in the simulation, some differences were introduced, especially in regards to the way we install the ban rule on the switch. In our final code, we keep track of rules installed on the switch and do not install a ban rule if there is already a similar rule in place.

6.3. Experimental Methodology for the DHCP Starvation Attacks Detection. In order to measure characteristics of the DHCP message retransmissions experimental evaluation has been conducted for Windows 10, Android OS, and iOS as well as for Dhcpstarv tool. The traffic related to the DHCP was captured on the sdn_nat host, with intentionally disabled DHCP service as in this case DHCP clients were unable to successfully obtain IP address and they infinitely retransmitted DHCP discovery messages forming transactions. The set of 100 DHCP transactions for Windows, 100 for Android, 100 for iOS, and 300 for Dhcpstarv were captured and they were used in the learning phase of the proposed security system. During this phase, DHCP discovery messages retransmissions timeouts were examined and the resulting characteristics (the minimal, the maximal, the average, and the standard deviation values) are presented in Section 7.2.

During the testing phase, both our security system and the DHCP server were enabled. Unfortunately legitimate DHCP client, after successfully obtaining an IP address, stops retransmitting DHCP discovery messages (which is how DHCP protocol was designed and should operate). Forcing such OS to reobtain IP address would require flushing current IP address by, for example, restarting network interface. This is not an easy task to automate for Windows and especially for Android and iOS operating systems. On the other hand, Dhcpstarv tool still tries to obtain as many IP addresses as possible; therefore it never stops this activity. Due to the facts above, our security system was tested only for Dhcpstarv tool (with 100% accuracy). On the other hand for legitimate devices (with Windows, Android, and iOS), another set of 100 DHCP transactions was captured for each OS and used in simulation to find out how many DHCP Discovery messages would be blocked by the proposed solution. The obtained results are presented in Section 7.2 as well.

7. Experimental Results

In this section, first we present experimental results obtained for the evaluation of the scanning detection module and then the results for the DHCP-related threats detection are introduced and analyzed.

7.1. Scanning Attacks Detection. Figure 3 presents the results of an exemplary *nmap* T5 scan mode. It is visible that the implementation of our solution behaves generally in line with our expectations. As *nmap* starts its scanning activity, it sends a large number of packets in a short burst. In this case, the controller is unable to precisely react and it initially passes through a number of packets (potentially large as it depends on the chosen *nmap* scanning mode). This is indicated in Figure 6 by a large red line near the beginning of the experiment. However, shortly after the initial burst,

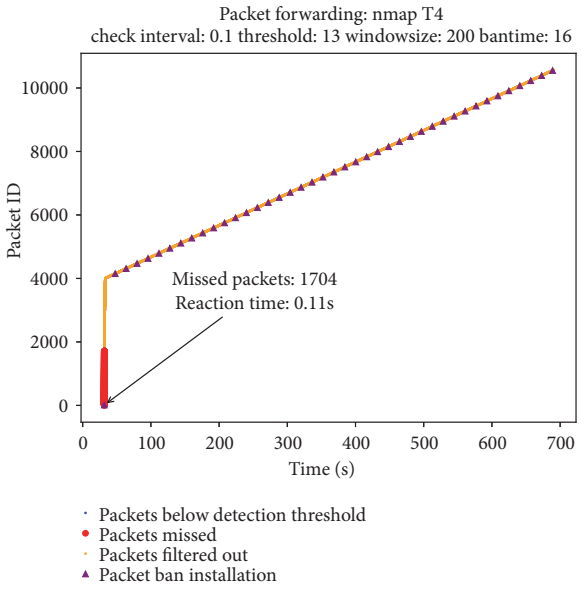


Figure 6: Packet forwarding trace during an exemplary experiment.

the controller installs a rule on the switch which successfully drops packets incoming from the rogue client. The *nmap* tool monitors the packet retransmission rate and once a significant packet loss occurs, it scales down the packet sending rate as it assumes that the scan exceeded network throughput. Therefore, after the initial burst of packets has been suppressed, the controller has the ability to reliably detect and block further attacks from this machine, as long as they occur within the *ban time* parameter time window.

Figure 7 presents exemplary CPU consumption (in [%]) and RAM utilization (in kilobytes (KiB)) during the experiment. It can be seen that after 30 seconds from the beginning of the experiment *nmap* the scanning activity is clearly visible as a spike in RAM and CPU utilization. As the initial burst of packets is handled by the controller, the algorithm quickly completes its database of connections which requires more RAM and CPU utilization with each incoming packet. The usage of both resources rises until ca. 210 seconds of the experiment when old TCP connections are removed from the database (as exactly at 180 seconds the TCP connection timeout has passed) and then the CPU usage decreases significantly.

The freed RAM memory is not returned to the system immediately, due to the optimization used by the Python interpreter to reduce the number of memory allocations. This phenomenon makes it difficult to estimate the real RAM consumption by our algorithm. However, because the algorithm periodically iterates over its connection database in a linear fashion, the memory occupancy can be correlated with the CPU usage.

At this point in time (i.e., after 210 seconds) the *nmap* scan rate is constant. On the other hand, because the experiment has been running for more than 180 seconds, with each cleanup cycle some of the connections are removed due to exceeded timeout. These two mechanisms keep the database

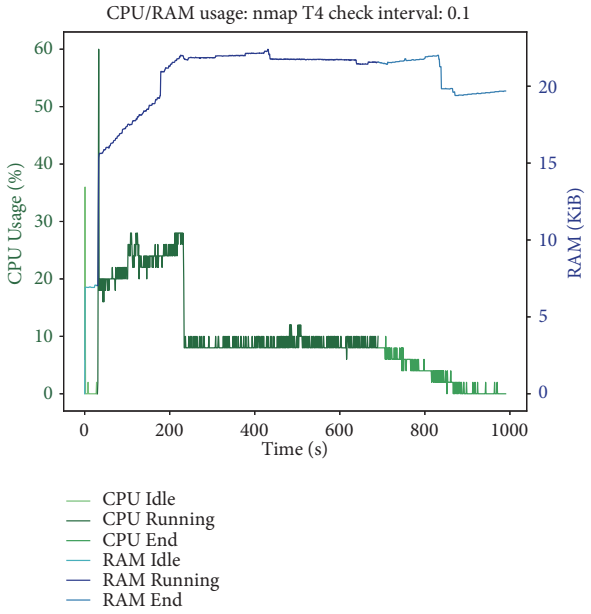


Figure 7: Resource consumption during an exemplary experiment.

size roughly constant, as the same amount of new connections is added to the database as the number of connections is removed. This is reflected in a constant CPU utilization during the *nmap* scan phase of the experiment.

In the case of nmap T1 scan mode, there is no initial burst of packets and the packet rate is significantly lower than in the case of T5 mode. However, our evaluation still marks some packets as missed—this is caused by two separate issues.

The first issue is the same as in case of nmap T5 scan mode during the initial burst—the controller's algorithm works in parallel to packet forwarding on the switch and due to that it has a small delay between the packet forwarding and packet inspection or rule installation. Due to this factor, in the real-world implementations at least one packet will always be missed when compared to the previously obtained simulation results.

The second issue is caused by the differences between our simulation software and the actual implementation of the algorithm on the controller. The simulation software did not account for the way we prevent installation of multiple duplicate rules on the switch.

The optimal results proposed by our simulation software were slightly too optimistic and balanced on the edge of detection when used on real hardware. For example, the experiment that utilized parameters found as *Threshold=7, Window Size=100* missed some packets, as illustrated in Figure 8. This is because nmap T1 mode sends a packet every 15 seconds with some random minor timing delays between the packets. When we divide the proposed 100 seconds time window by 15 seconds between each packet, we see that nmap will be able to send around 6.66 packets in the proposed time window, which is slightly less than proposed threshold value equal 7. There is a slight chance that at some point the time window will contain 6 incomplete connections instead of expected 7, allowing one scan packet to pass through. In order to solve this issue a slight increase in the time window size is necessary, e.g., to 105 seconds in to accommodate 7 packets sent by nmap (see Figure 9).

The results presented in Table 3 are oriented towards determining the optimal value for the *check interval* parameter in order to minimize the reaction delay of attack mitigation. The reaction delay depends on the frequency of inspection of TCP connection statistics gathered in memory and the time needed by the Open vSwitch to install a banning rule. Since the latter factor is outside of our control, we try to optimize the former factor, i.e., *check interval* parameter to achieve the best compromise between the CPU usage and the reaction time. The results—the average CPU utilization by the POX controller and the reaction time—are influenced by the following:

(i) Nmap scan mode determines the frequency of scanning packets generation and time constraints.

(ii) Window size parameter influences the memory cleanup feature and CPU utilization if the value exceeds 180 seconds—we delete timed out TCP connections that are older than 180 seconds and *window-size*.

(iii) Check interval parameter is the frequency of how often we scan the memory map for the IP addresses that exceed given threshold value.

In case the POX controller gets overloaded and fails to respond to Open vSwitch on time, the Open vSwitch disconnects itself from the controller and enters fallback mode, where it passes all network traffic without filtering, effectively disabling all security features. Such cases were marked in Table 3 with bold font. As observed, the *check interval* parameters shorter than 0.1 seconds tend to overload the controller during the most intensive nmap scans. We selected the 0.1 seconds value as optimal because it offers the shortest reaction time while still being able to withstand most nmap scans. It should be noted however that *check interval* parameter equal to *0.2 seconds* provides only slightly worse accuracy while the CPU utilization is nearly halved, which might be more useful in a nonexperimental environment.

Additionally, we have noticed that the *Threshold* values 4 and 7 tend to distort the results, as they trigger detection much earlier than in the rest of experiments. In case of T5, T4, and T3 scans, there is an initial, small burst of packets being sent by the tool, followed by a 1 s pause, as visible in Figure 10. In this case, the initial packets sent by *nmap* trigger detection; however no packets are filtered out, because there is no traffic for the next 1 s. As previously described, reaction time is determined by the interval between the ban installation timestamp and the first packet that has been filtered out. In this particular case this includes 1 s when *nmap* remains silent. Thus, for these experiments, the reaction times can not be reliably compared with other results presented in Table 3.

Based on the gathered experimental data, we estimated that the rule installation time in our experimental testbed should be within 0.025–0.075 second range. Therefore

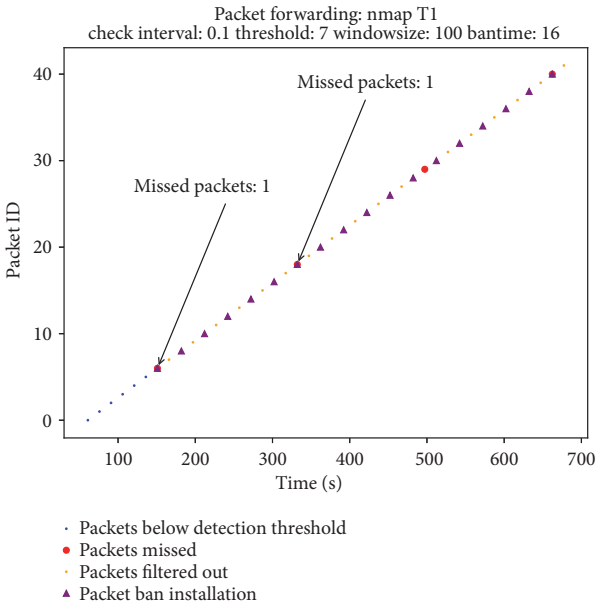


Figure 8: Packet forwarding trace where detection parameters are too optimistic.

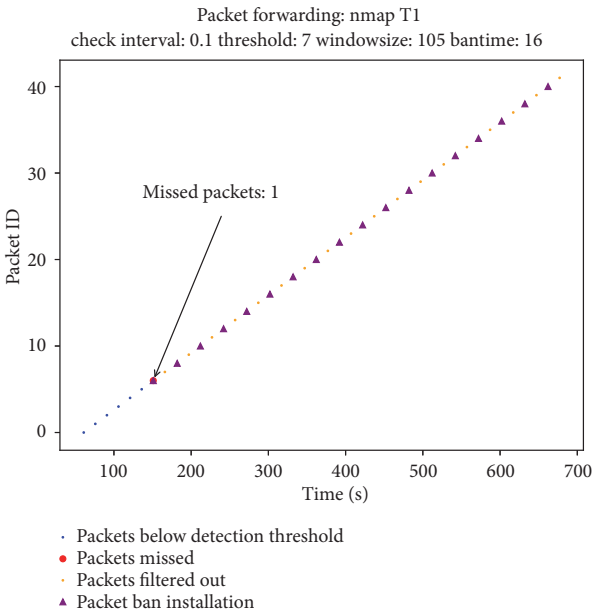


Figure 9: Packet forwarding trace with adjusted window size parameter.

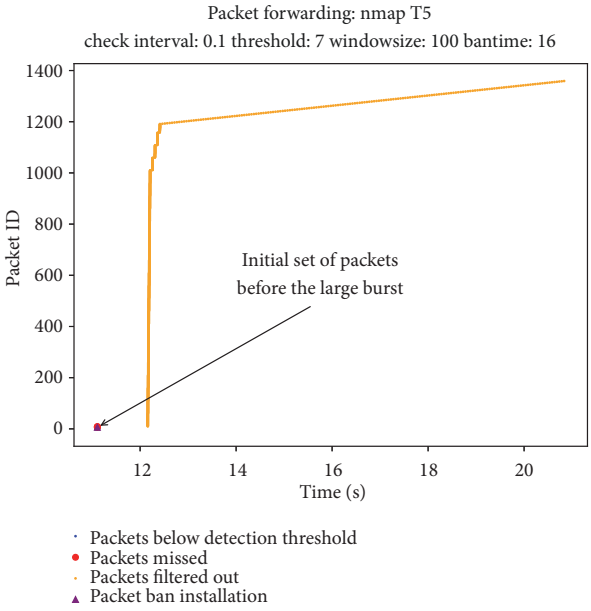


Figure 10: Initial set of packets sent by nmap during T5 scan.

decreasing *check interval* parameter beyond this range brings little to none benefit in terms of the reaction time. Future works could include exploring the aspect of utilizing a faster runtime environment (such as using C++ or C language instead of Python on the controller machine) and further code optimization to achieve higher CPU performance.

After determining the optimal *check interval* parameter value, i.e., 0.1 seconds, we compared the simulation results of *nmap* mitigation and false positive rate with the measured experimental results. This is presented in Table 4. It must be noted that the obtained experimental results are similar to those acquired through simulations and this proves that our system (when the proper parameters are configured) is effective in detecting and mitigating *nmap* scans, while being nonintrusive to the benign traffic.

The measured packet forwarding time was 1.71×10^{-6} seconds for the unprotected and 3.35×10^{-6} seconds for

Table 3: Experimental results for various *nmap* scan modes and check interval values. Results contain the average CPU utilization during the experiment and the reaction delay. Results in bold indicate controller's overload and failure.

			Check Interval															
			0.01 s		0.02 s		0.05 s		0.10 s		0.20 s		0.50 s		1.00 s		2.00 s	
nmap mode	W_s	T	CPU avg%	R. t. [s]	CPU avg%	R. t. [s]	CPU avg%	R. t. [s]	CPU avg%	R. t. [s]	CPU avg%	R. t. [s]	CPU avg%	R. t. [s]	CPU avg%	R. t. [s]	CPU avg%	R. t. [s]
T5	50	4	**94.0**	—	**78.9**	—	**54.8**	—	**42.4**	—	30.0	1.050	3.9	1.050	12.0	1.050	2.3	1.687
	100	7	**99.6**	—	**91.1**	—	**56.3**	—	15.9	1.050	7.4	1.049	3.9	1.050	2.7	1.050	2.3	1.685
	200	13	**91.3**	—	**78.9**	—	39.1	0.037	23.0	0.081	10.4	0.153	4.9	1.139	3.3	0.840	2.5	1.394
	300	19	**99.0**	—	**88.3**	—	49.1	0.050	30.9	0.075	12.6	0.176	6.5	0.707	4.0	0.865	3.0	1.402
	400	26	**91.9**	—	**88.5**	—	**71.8**	—	28.3	0.107	15.2	0.437	9.0	0.602	5.5	1.139	4.0	1.378
	450	29	**92.9**	—	**88.6**	—	60.9	0.081	33.9	0.115	18.1	0.189	9.3	1.001	4.9	1.125	3.9	1.396
	500	32	**95.3**	—	**90.6**	—	**75.5**	—	36.3	0.133	18.9	0.136	9.4	0.996	5.2	1.132	3.6	1.480
	550	34	**96.6**	—	**91.3**	—	**73.6**	—	38.0	0.119	25.2	0.193	9.8	1.133	5.7	1.271	3.7	1.458
	600	38	**100.0**	—	**95.0**	—	**72.1**	—	39.2	0.134	21.5	0.202	8.5	0.871	5.7	0.834	4.0	1.402
	650	41	**99.9**	—	**90.0**	—	**65.8**	—	36.9	0.159	21.0	0.170	10.4	0.619	5.8	1.114	3.7	1.407
T4	50	4	**74.0**	—	**53.6**	—	26.6	1.100	20.5	1.100	4.6	1.100	2.4	1.100	2.1	1.100	1.6	1.100
	100	7	**81.4**	—	**58.5**	—	**42.1**	—	10.1	1.100	4.6	1.100	2.4	1.100	1.9	1.100	1.5	1.549
	200	13	**81.2**	—	**54.2**	—	22.4	0.053	13.1	0.082	7.8	0.150	3.3	0.519	2.5	0.721	1.7	1.210
	300	19	**91.7**	—	**66.3**	—	37.5	0.052	18.3	0.060	10.3	0.165	4.9	0.302	2.8	0.850	2.0	1.172
	400	26	**89.7**	—	**72.5**	—	38.4	0.049	24.2	0.056	11.4	0.154	6.2	0.532	3.5	0.481	2.5	1.148
	450	29	**93.0**	—	**100.0**	—	52.6	0.047	31.0	0.070	14.1	0.212	6.1	0.194	4.1	0.764	2.5	1.236
	500	32	**91.0**	—	**81.6**	—	65.1	0.050	30.7	0.089	15.5	0.151	6.7	0.411	4.7	0.833	2.6	1.272
	550	34	**93.8**	—	**86.3**	—	59.0	0.053	32.0	0.071	17.0	0.193	7.7	0.533	4.4	0.547	3.0	1.509
	600	38	**95.3**	—	**100.0**	—	73.4	0.058	35.5	0.101	18.6	0.176	7.1	0.397	4.3	0.852	2.7	1.421
	650	41	**99.3**	—	**95.7**	—	57.1	0.049	34.6	0.122	22.3	0.151	7.9	0.427	4.4	0.630	2.9	1.411
T3	50	4	35.2	1.100	**56.2**	—	**40.0**	—	5.0	1.100	2.7	1.100	1.5	1.100	1.0	1.100	0.8	1.374
	100	7	**50.6**	—	**40.8**	—	7.8	1.100	4.7	1.100	2.3	1.100	1.1	1.100	0.8	1.100	0.6	1.212
	200	13	**81.0**	—	**52.2**	—	16.9	0.044	13.8	0.074	6.9	0.150	3.5	0.364	2.2	0.652	1.7	1.260
	300	19	**79.5**	—	**64.3**	—	34.5	0.040	18.8	0.060	7.2	0.197	4.7	0.496	2.8	0.683	1.9	1.190
	400	26	**86.0**	—	**72.1**	—	32.5	0.049	24.0	0.063	11.3	0.167	6.1	0.472	3.2	0.471	2.5	1.223
	450	29	**92.9**	—	**98.6**	—	47.1	0.037	30.5	0.082	13.9	0.178	6.4	0.510	3.8	0.779	2.3	1.362
	500	32	**88.8**	—	**80.8**	—	60.9	0.046	28.4	0.109	16.3	0.191	6.7	0.313	4.2	0.706	2.5	1.347
	550	34	**90.2**	—	**86.5**	—	61.4	0.056	34.1	0.095	16.1	0.202	7.2	0.360	4.4	0.654	2.8	1.272
	600	38	**95.3**	—	**100.0**	—	70.5	0.057	36.7	0.113	14.5	0.169	7.2	0.351	4.2	0.854	2.5	1.397
	650	41	**99.3**	—	**94.7**	—	63.3	0.054	33.1	0.132	17.0	0.119	7.9	0.308	4.5	0.552	2.9	1.282
T2	50	4	15.6	0.399	8.1	0.400	4.7	0.400	1.5	0.400	1.4	0.400	0.9	0.572	0.6	1.029	0.3	0.900
	100	7	11.1	0.400	11.0	0.400	2.0	0.400	1.3	0.400	0.7	0.400	0.3	0.400	0.3	0.700	0.2	1.301
	200	13	9.0	0.400	4.6	0.400	2.1	0.400	1.5	0.400	0.8	0.400	0.3	0.500	0.3	0.650	0.2	0.750
	300	19	12.4	0.400	6.5	0.400	3.1	0.400	1.6	0.400	1.3	0.400	0.4	0.450	0.3	0.800	0.2	0.560
	400	26	14.9	0.400	7.6	0.400	3.4	0.401	1.9	0.400	1.2	0.400	0.6	0.400	0.3	0.400	0.3	1.601
	450	29	16.6	0.400	8.6	0.400	3.9	0.401	2.5	0.400	1.4	0.400	0.7	0.400	1.5	0.401	0.3	1.201
	500	32	17.6	0.400	9.3	0.400	4.4	0.400	2.4	0.400	1.5	0.400	0.8	0.400	0.5	0.801	0.4	1.601
	550	34	18.6	0.400	9.7	0.400	4.8	0.400	3.3	0.400	1.6	0.400	0.9	0.392	0.6	0.801	0.4	2.002
	600	38	19.8	0.400	10.7	0.400	5.3	0.400	2.9	0.400	1.5	0.400	0.7	0.400	0.4	0.801	0.3	1.201
	650	41	20.3	0.400	10.5	0.400	4.5	0.400	2.5	0.400	2.6	0.400	1.0	0.801	0.5	0.400	0.3	1.601

Table 3: Continued.

nmap mode	W_s	T	Check Interval 0.01 s CPU avg%	0.01 s R. t. [s]	0.02 s CPU avg%	0.02 s R. t. [s]	0.05 s CPU avg%	0.05 s R. t. [s]	0.10 s CPU avg%	0.10 s R. t. [s]	0.20 s CPU avg%	0.20 s R. t. [s]	0.50 s CPU avg%	0.50 s R. t. [s]	1.00 s CPU avg%	1.00 s R. t. [s]	2.00 s CPU avg%	2.00 s R. t. [s]
T1	50	4	3.5	15.010	2.2	15.002	1.5	15.013	1.2	15.005	0.3	15.012	0.2	15.004	0.3	15.004	0.4	15.005
	100	7	1.8	15.012	1.2	15.015	0.6	15.006	0.3	15.000	0.2	15.013	0.2	15.004	0.1	15.008	0.2	15.015
	200	13	1.9	15.015	1.3	15.004	0.5	15.004	0.4	15.004	0.2	15.012	0.1	15.001	0.1	15.015	0.2	15.004
	300	19	6.4	15.001	1.4	15.005	0.6	15.008	0.3	15.005	0.6	15.004	0.1	15.001	0.1	15.015	0.1	15.005
	400	26	2.2	15.007	1.6	15.009	0.6	15.013	0.3	15.004	0.2	15.008	0.2	15.012	0.1	15.010	0.1	15.012
	450	29	2.4	15.012	1.8	15.012	0.8	15.014	0.5	15.002	0.3	15.009	0.3	15.013	0.2	15.015	0.2	15.008
	500	32	2.4	15.014	1.5	15.015	0.9	15.004	0.4	15.015	0.3	15.001	0.3	15.012	0.2	15.012	0.2	15.015
	550	34	2.4	15.004	1.5	15.012	0.9	15.015	0.5	15.000	0.3	15.008	0.3	15.015	0.2	15.013	0.2	15.012
	600	38	2.5	15.015	1.7	15.015	1.0	15.001	0.6	15.013	0.2	15.015	0.2	15.001	0.1	15.015	0.2	15.012
	650	41	2.5	15.010	1.6	15.008	0.7	15.001	0.4	15.001	1.1	15.004	0.2	15.004	0.2	15.000	0.1	15.001

Table 4: Comparison of the expected versus actual detection performance. *T* stands for *threshold*.

Window size [s]	T	Simulated			Measured		
		T1	T2	FPR	T1	T2	FPR
50	4	98.5	99.6	86.9	85.4	99.9	79.6
100	7	97.4	99.4	68.2	89.0	99.8	80.8
200	13	95.2	98.9	46.4	95.3	99.6	57.0
300	19	93.0	98.4	21.9	93.1	99.5	18.6
400	26	72.3	97.9	12.0	90.5	99.3	7.1
450	29	89.4	97.5	10.9	89.4	99.3	7.4
500	32	88.0	97.3	10.2	88.3	99.2	7.2
550	34	87.2	96.8	1.9	87.5	99.1	1.9
600	38	85.8	96.8	1.9	86.0	99.0	2.8
650	41	84.7	96.6	1.9	84.6	98.9	2.9

Table 5: Retransmission timeouts for the consecutive DHCP messages for Windows 10.

Message	Min. [s]	Max. [s]	Average [s]	Std. dev. [s]
2.	2.094	4.999	3.946	0.550
3.	6.251	9.255	8.102	0.572
4.	14.500	17.266	16.045	0.604

the protected system, which is a negligible difference in comparison to the other state-of-the-art solutions.

To summarize it can be concluded that the best parameters for our scanning detection module are *windowsize*: 550 seconds, *threshold*: 34 connections, *bantime*: 16 seconds, and *check interval*: 0.1 seconds.

7.2. DHCP Starvation Attacks Detection. First we have performed measurements of the DHCP transaction retransmissions for different operating systems and the obtained results for the legitimate DHCP traffic have been used to determine the normal behavior. Then we preformed the same operation for the DHCPstarv tool to determine the characteristics of the malicious activities. In the end all the results have been used to feed the detection system. The details of the measurements are presented below.

Table 5 contains results of the DHCP transaction retransmissions for Windows 10. It should be noted that in this case during a single DHCP transaction four packets are broadcasted and the second, the third, and the fourth packets are sent after ca. 4, 12, and 28 seconds since the initial DHCP message. This gives 4, 8, and 16 seconds timeouts, respectively. After sending the fourth packet, administrative timeout of ca. 32 seconds is counted and if no reply is received, the transaction is considered as failed (ca. 60s). What is more analyzing collected dataset of DHCP transactions for Windows OS showed that the timeout between consecutive transactions (thus the time between sending the first DHCP message for two consecutive transactions) can be described with the series ca. 60s, 360s, 60s, 360s, etc. This means that every new transaction is started immediately after previous transaction (60s) or after five minutes (360s).

Table 6: Retransmission timeouts for the consecutive DHCP messages for Android OS.

Message	Min. [s]	Max. [s]	Average [s]	Std. dev. [s]
2.	1.806	2.202	2.017	0.119
3.	3.603	4.400	4.004	0.236
4.	7.240	10.461	8.050	0.610
5.	14.638	19.374	16.800	1.093

Table 6 contains similar results, but for the DHCP transactions collected for Android OS. In this case, during a single transaction five packets are broadcasted and the second, the third, the fourth, and the fifth packets are typically sent after ca. 2, 6, 14, and 30 seconds. This is caused by the timeouts used which are 2, 4, 8, and 16 seconds, respectively. After sending the fifth packet, timeout of ca. 8 s is counted and if no reply is received, transaction is considered as failed (ca. 38 s.) Analyzing collected dataset also showed that the timeout between consecutive transactions (thus the time between sending the first DHCP message for two consecutive transactions) can be described with the series 38, 300-1200, 38, 300-1200 seconds, etc. This means that a new transaction is not started immediately after the previous one but after the random time of about 300-1200s of the administrative timeout. The timeouts between retransmitted DHCP messages within the transaction are smaller than in the Windows dataset, most probably because mobile users typically want to get their online content quickly. Decreasing the timeout between DHCP messages helps to achieve this aim.

DHCP retransmissions for iOS are somewhat more complex than two previous cases. There are two types of transactions which occur in turn. The first transaction type contains nine packets, and the second to the ninth packet are sent after ca. 1.6, 4.3, 9.1, 17.6, 26.1, 34.6, 43.1, and 51.6 seconds. This corresponds to 1.6, 2.7, 4.8, 8.5, 8.5, 8.5, and 8.5 seconds timeout, respectively. Table 7 contains characteristics of those retransmissions. The next transaction will start after ca. 10s timeout. Second transaction type contains only two packets, and the second one is also sent after ca. 1,6 second timeout. The initial transaction started by iOS is the one with 9 packets, and the second is with 2 packets. We are excluding shorter transaction as

(i) our solution is based on transactions with at least three packets,

(ii) the first initiated transaction by iOS is always the one with nine packets.

Finally, Table 8 contains DHCP transaction characteristics for the Dhcpstarv attack tool. In this case, during one transaction three packets are broadcasted and the second and the third packets are sent after ca. 2 and 4 seconds. After sending the third packet timeout of ca. 2 s is set and if no reply is received during this period, the transaction is considered as failed after 6s timeout. The analysis of this dataset also showed that the timeout between consecutive transactions can be described with the series 6 s, 6 s etc. This means that another

Table 7: Retransmission timeouts for the consecutive DHCP messages for iOS.

Message	Min. [s]	Max. [s]	Average [s]	Std. dev. [s]
2.	1.022	2.979	1.627	0.418
3.	1.998	5.053	2.722	0.748
4.	4.015	12.937	5.028	1.537
5.	7.972	9.004	8.505	0.262
6.	7.982	8.975	8.494	0.253
7.	7.938	9.070	8.506	0.298
8.	7.905	9.000	8.471	0.269
9.	8.009	9.006	8.483	0.274

Table 8: Retransmission timeouts for the consecutive DHCP messages for Dhcpstarv tool.

Message	Min. [s]	Max. [s]	Average [s]	Std. dev. [s]
2.	2.000274	2.002858	2.002490	0.000161
3.	2.000428	2.002922	2.002517	0.000157

Table 9: Cost introduced by the proposed solution for the legitemate devices.

Operating System	Windows	Android	iOS	Overall
Cost [s]	3.946	2.017	1.657	2.540
Retransmissions	100	100	101	301

transaction is started immediately after the previous one. It should be noted that results from Table 8 contain values with high precision in order to show differences between minimal, maximal, and average values. With typical precision as used in the previous tables, these differences would not be visible.

During the learning phase of the experiment, the minimum and the maximum retransmission timeout values for Dhcpstarv tool were examined. The result was used to create the pattern to describe malicious activity. Therefore if the first retransmission of any DHCP discovery transaction would occur between 2.000274 and 2.002858 seconds, such traffic should be blocked as a possible threat. Likewise the second retransmission will be blocked if the retransmission timeout would take place between 2.000428 and 2.002922 seconds.

Using the values above, experiments for the test dataset were performed. As a result, only once legitimate device (iOS) was identified as malicious; thus the second DHCP packet (therefore the first retransmission) was also dropped. This caused additional delay (2.950614) in obtaining IP address for these devices, which as already mentioned can be considered as a kind of "cost" for the proposed system. The average delays introduced for Android, Windows, iOS, and overall for all OSes are shown in Table 9. It should be noted that, in the proposed solution, for Windows and Android OS devices only one retransmission has to take place (as this is how the proposed mitigation method works). For the iOS, it can be one or two retransmissions, but in this case retransmissions are faster than for Windows and Android OSes (see Tables 5 and 6). The time needed to successfully obtain IP address (fixed value of 1-3ms) is negligible.

Table 10: Delay in obtaining IP address from the DHCP server for legitimate user for diffrent scenarios.

Scenario	1	2	3	4
Delay	1-3ms	∞	2.54s	2.54s
Protection	NO	NO	YES	YES

It can be assumed that if a legitimate device is booting, about 2.5 seconds for obtaining IP address can be negligible as well (in terms of overall booting time). But on the other hand, if the device is up and running, and only the access to network is enabled, then this can result in a bad user experience. Nevertheless, without providing such a mitigation system that will block DHCP exhaustion, the access to the network can be completely blocked by an attacker.

Another type of the trivial DHCP starvation attack can be based on flooding network with the DHCP messages (in order to speed up the process of the DHCP address pool exhausting):

(i) not utilizing retransmissions (i.e., only one message in one transaction) sent with the high rate,

(ii) utilizing retransmissions but still sending them with the high rate (vary timeout delay between retransmissions within a single transaction).

Such an attack would be easily blocked by the proposed system as timeouts between DHCP messages would be too small, and not comparable to known patterns of most popular operating systems.

Implementing the proposed solution will result in increased delay in obtaining IP address from the DHCP server for the legitimate users but also will make DHCP pool starvation impossible using trivial tools or Dhcpstarv tool.

For the scenarios described in Table 10, implementation of proposed solution is summed up:

(1) only legitimate devices on the network with the mitigation system disabled,

(2) legitimate and attacker devices on the network with the mitigation system disabled,

(3) legitimate devices with the enabled mitigation system,

(4) legitimate and attacker devices with an active mitigation system.

8. Conclusions and Future Work

In this paper, we introduced a novel, dedicated, SDN-based integrated security framework for the IoRL system which is created using the 5G networks architecture guidelines. We also presented how the proposed defensive solution can be used to detect and mitigate various network threats. To demonstrate its usefulness, we tested it with exemplary network threats which included TCP SYN scanning and DHCP-related attacks (DHCP pool exhaustion, traffic eavesdropping, and DHCP rogue server placement).

Obtained experimental results performed using proof-of-concept implementation of the proposed system proved that

it is efficient and effective. Another important advantage of the developed framework is that it is easily extendable; i.e., countering a new threat is possible by simply adding a new specialized security module. It must be also noted that the developed modules are quite straightforward yet they defend against serious and important threats.

Future work will be devoted to the extension of the capabilities of the integrated security framework so it is able to mitigate a wider number of threats. We will also consider applying more sophisticated detection algorithms for the existing security modules.

Data Availability

The network traffic data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

The parts of the research presented in this paper were accepted for presentation and presented at 13th International Conference on Availability, Reliability and Security (ARES 2018).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors gratefully acknowledge the financial support of the EU Horizon 2020 program towards the Internet of Radio Light project H2020-ICT 761992.

References

[1] J. Cosmas, B. Meunier, K. Ali et al., "5G Internet of radio light services for supermarkets," in *Proceedings of the 2017 14th China International Forum on Solid State Lighting: International Forum on Wide Bandgap Semiconductors China (SSLChina: IFWS)*, pp. 69–73, Beijing, China, November 2017.

[2] J. Cosmas, B. Meunier, K. Ali et al., "A Scalable and License Free 5G Internet of Radio Light Architecture for Services in Train Stations," in *Proceedings of the 24th European Wireless Conference*, European Wireless, 2018.

[3] K. Cabaj, M. Gregorczyk, W. Mazurczyk, P. Nowakowski, and P. Żórawski, "SDN-based Mitigation of Scanning Attacks for the 5G Internet of Radio Light System," in *Proceedings of the the 13th International Conference on Availability, Reliability and Security (ARES 2018)*, pp. 1–10, Hamburg, Germany, August 2018.

[4] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[5] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Proceedings of the 14th International conference on Recent Advances in Intrusion Detection (RAID 2011)*, pp. 161–180, 2011.

[6] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "OrchSec: an orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions," in *Proceedings of Network Operations and Management Symposium (NOMS '14)*, pp. 1–9, IEEE, Krakow, Poland, May 2014.

[7] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Proceedings of the 2012 20th IEEE International Conference on Network Protocols, ICNP 2012*, 6, 1 pages, IEEE, Austin, TX, USA, November 2012.

[8] H. Zhang, Z. Cai, Q. Liu, Q. Xiao, Y. Li, and C. F. Cheang, "A Survey on Security-Aware Measurement in SDN," *Security and Communication Networks*, vol. 2018, Article ID 2459154, 14 pages, 2018.

[9] R. Jin and B. Wang, "Malware detection for mobile devices using software-defined networking," in *Proceedings of the 2013 2nd GENI Research and Educational Experiment Workshop, GREE 2013*, pp. 81–88, March 2013.

[10] J. M. Ceron, C. B. Margi, and L. Z. Granville, "MARS: An SDN-based malware analysis solution," in *Proceedings of the 2016 IEEE Symposium on Computers and Communication, ISCC 2016*, pp. 525–530, Messina, Italy, July 2016.

[11] K. Cabaj and W. Mazurczyk, "Using Software-Defined Networking for Ransomware Mitigation: The Case of CryptoWall," *IEEE Network*, vol. 30, no. 6, pp. 14–20, 2016.

[12] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics," *Computers and Electrical Engineering*, vol. 66, pp. 353–368, 2018.

[13] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A DDoS Attack Detection Method Based on SVM in Software Defined Network," *Security and Communication Networks*, vol. 2018, Article ID 9804061, 8 pages, 2018.

[14] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS attack detection and mitigation using SDN: methods, practices, and solutions," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 425–441, 2017.

[15] L. Dridi and M. F. Zhani, "A holistic approach to mitigating DoS attacks in SDN networks," *International Journal of Network Management*, vol. 28, no. 1, 2018.

[16] D. Yin, L. Zhang, and K. Yang, "A DDoS Attack Detection and Mitigation With Software-Defined Internet of Things Framework," *IEEE Access*, vol. 6, pp. 24694–24705, 2018.

[17] Q. Niyaz, W. Sun, and A. Y. Javaid, "A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)," *EAI Endorsed Transactions on Security and Safety*, vol. 4, no. 12, Article ID 153515, 2017.

[18] W. Wang, X. Ke, and L. Wang, "A HMM-R Approach to Detect L-DDoS Attack Adaptively on SDN Controller," *Future Internet*, vol. 10, no. 9, p. 83, 2018.

[19] S. Shin, V. Yegneswaran, and P. Porras, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 413–424, Berlin, Germany, 2013.

[20] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS Attack Prevention Extension in Software-Defned Networks," in *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 239–250, 2015.

[21] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "FloodDefender: Protecting data and control plane resources under SDN-aimed

DoS attacks," in *Proceedings of the 2017 IEEE Conference on Computer Communications, INFOCOM 2017*, May 2017.

[22] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for DDoS protection in software defined networks," in *Proceedings of the 2017 IEEE Conference on Network Softwarization, NetSoft 2017*, pp. 1–9, July 2017.

[23] H. Yuwen, Z. Liancheng, W. Zhenxing, and K. Yazhou, "Probability-based delay scheme for resisting SDN scanning," in *Proceedings of the 2nd IEEE International Conference on Computer and Communications, ICCC 2016*, pp. 1096–1101, China, October 2016.

[24] Z. Zhao, F. Liu, D. Gong, L. Chen, F. Xiang, and Y. Li, "An SDN-based IP hopping communication scheme against scanning attack," in *Proceedings of the 9th IEEE International Conference on Communication Software and Networks, ICCSN 2017*, pp. 559–564, May 2017.

[25] S. Shirali-Shahreza and Y. Ganjali, "Protecting Home User Devices with an SDN-Based Firewall," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 1, pp. 92–100, 2018.

[26] S. Dotcenko, A. Vladyko, and I. Letenko, "A fuzzy logic-based information security management for software-defined networks," in *Proceedings of the 16th International Conference on Advanced Communication Technology: Content Centric Network Innovation!, ICACT 2014*, pp. 167–171, February 2014.

[27] R. Mohammadi, R. Javidan, and M. Conti, "SLICOTS: An SDN-based lightweight countermeasure for TCP SYN flooding attacks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 487–497, 2017.

[28] Rietz, A. Brinner, and R. Cwalinsky, *Improving Network Security in Virtualized Environments with Openflow*, 2015.

[29] J. H. Cox, R. J. Clark, and H. L. Owen, "Leveraging SDN to improve the security of DHCP," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization (SDNNFV-SEC 2016)*, pp. 35–38, 2016.

[30] J. L. Wang and Y. C. Chen, "An SDN-based defensive solution against DHCP attacks in the virtualization environment," in *Proceedings of the 2017 IEEE Conference on Dependable and Secure Computing*, pp. 529-530, Taipei, Taiwan, August 2017.

[31] R. Droms, "RFC 2131 Dynamic Host Configuration Protocol," 1997, https://www.ietf.org/rfc/rfc2131.txt.

[32] "Operating System Market Share Worldwide," 2018, http://gs.statcounter.com/os-market-share.

[33] "Android Version Market Share Worldwide," 2018, http://gs.statcounter.com/os-version-market-share/android.

[34] "Desktop Windows Version Market Share Worldwide," 2018, http://gs.statcounter.com/os-version-market-share/windows/desktop.

[35] "Mobile & Tablet iOS Version Market Share Worldwide," 2018, http://gs.statcounter.com/os-version-market-share/ios/mobile-tablet/worldwide.

6. Wykrywanie podsłuchu sieciowego na podstawie analizy metryk ruchu

W zamieszczonej w tym rozdziale pracy “Sniffing Detection Based on Network Traffic Probing and Machine Learning” opublikowanej w czasopiśmie “IEEE Access”, usystematyzowano sposoby i narzędzia do wykrywania snifferów oraz zaproponowano nowatorską metodę opartą na sztucznej inteligencji. Jak się okazuje, istniejące obecnie metody detekcji tego typu ataku są w większości przestarzałe i nieefektywne. Wszelkie publicznie dostępne narzędzia również nie spełniają swojej funkcji.

W przedstawionym artykule podzielono rozwiązania do wykrywania sniffingu na metody polegające na sprowokowaniu systemu, na którym działa sniffer do określonej reakcji (challenge-based) oraz sposoby oparte na pomiarach odpowiedzi (measurement-based). W pierwszym przypadku tylko jedna, znana z literatury metoda nadal była możliwa do wykorzystania. Polega ona na sprowokowaniu sniffera do rozwiązania nazwy domenowej podsłuchanego adresu IP. Rozwiązanie to zostało zaimplementowane w Integrated Security Framework za pomocą technologii SDN. Technologia ta w łatwy sposób daje możliwość tworzenia odpowiednio spreparowanego ruchu i wysłania go na właściwy port, oczekując określonej odpowiedzi, w tym przypadku zapytania Domain Name System (DNS).

Poniższa publikacja jest rozszerzoną wersją artykułu konferencyjnego [47], jednak w istotny sposób zmienia poprzednią metodę, opartą na pomiarach odpowiedzi. To, co wyróżnia tę pracę, to zastosowanie technik uczenia maszynowego w celu wykrycia hosta posiadającego aktywny sniffer. Oprócz używanego głównie w tym celu protokołu ICMP użyto po raz pierwszy protokołu warstwy wyższej (HTTP), sprawdzono wydajność metody w wykrywaniu sniffera działającego na systemie Linux. Zastosowano także dodatkowy czynnik - obciążenie CPU - co powoduje, że otrzymane wyniki są bardziej wiarygodnie i bliższe warunkom rzeczywistym. Wykrywanie sniffera było wykonane zgodnie z zasadą black-box, to znaczy detekcja podsłuchu polegała wyłącznie na analizie znanych faktów zewnętrznych, bez znajomości stanu sniffera działającego na podejrzanym hoście.

Warto zauważyć, że na potrzeby tej publikacji, zostały wypożyczone z IBM Polska sp. z o.o. dwa serwery najwyższej klasy Enterprise, tj. IBM POWER AC822 oraz AC922 wyposażone w wydajne karty GPU, znacznie przyspieszające obliczenia w uczeniu maszynowym.

Otrzymane wyniki pokazują, że przedstawiona metoda jest bardzo obiecująca i osiąga skuteczność wykrywania sniffera na poziomie 99%. Pewnym ograniczeniem jest to, że zaproponowany mechanizm zakłada zalewanie sieci dużą liczbą pakietów, co może być nieakceptowalne w niektórych środowiskach sieciowych. Nie zmienia to jednak faktu, że opisany sposób detekcji jest bardzo skuteczny w wykrywaniu snifferów.

Ponownie badania skuteczności metod detekcji przeprowadzono za pomocą rzeczywistych eksperymentów, a nie symulacyjnie, co dodatkowo potwierdza słuszność założeń i skuteczność rozwiązania.

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 24, 2020, accepted August 5, 2020, date of publication August 12, 2020, date of current version August 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3016076

Sniffing Detection Based on Network Traffic Probing and Machine Learning

MARCIN GREGORCZYK[1], PIOTR ŻÓRAWSKI[1], PIOTR NOWAKOWSKI[1], KRZYSZTOF CABAJ[2], AND WOJCIECH MAZURCZYK[2], (Senior Member, IEEE)
[1]Institute of Telecommunications, Warsaw University of Technology, 00-665 Warsaw, Poland
[2]Institute of Computer Science, Warsaw University of Technology, 00-665 Warsaw, Poland

Corresponding author: Marcin Gregorczyk (m.gregorczyk@tele.pw.edu.pl)

This work was supported by the EU Horizon 2020 Program towards the Internet of Radio-Light Project under Grant H2020-ICT 761992.

ABSTRACT Cyber attacks are on the rise and each day cyber criminals are developing more and more sophisticated methods to compromise the security of their targets. Sniffing is one of the most important techniques that enables the attacker to collect information on the vulnerabilities of the devices, protocols and applications that can be exploited within the targeted network. It relies mainly on passively analyzing the traffic exchanged within the network, and due to its nature, such an activity is difficult to discover. That is why, in this article, we first revisit existing techniques and tools that can be used to perform sniffing as well as the corresponding mitigation methods. Based on this background, we propose a novel measurement-based detection method that infers whether the sniffing software is active on the suspected machine by network traffic probing and machine learning techniques. The presented experimental results prove that the proposed solution is effective.

INDEX TERMS AI, artificial intelligence, ML, machine learning, network security, sniffing, threat detection.

I. INTRODUCTION

Currently, whole societies are becoming even more dependent on open networks. The Internet changes various aspects of everyday life, like commercial activities, business transactions and government services being offered online. As a result, this has led to the fast development of new cyber threats and numerous information security issues, which are exploited by cyber criminals. Moreover, currently the attackers are devising increasingly sophisticated methods to compromise the security of the devices that the users utilize.

Sniffing is a typical part of the reconnaissance phase of the network attack. It can be executed by the attacker who is able to access the targeted network infrastructure. Moreover, the attacker can analyze the passing network traffic using, e.g., an insecure WiFi network, or as a result of successful insider threat actions. The main aim of such activities is to identify the machines, protocols and applications that are running within the targeted network. After that the attacker can determine the potential vulnerabilities that can be used to compromise the network. In most cases these actions precede the actual attack. Therefore, it must be noted that an early detection of sniffing is of great importance as it allows the defender to prepare for further attack phases. Moreover, if the security system introduced within the network is able to detect sniffing devices, it can take countermeasure actions and thus reduce the attack probability. Sniffing is typically executed with the help of dedicated software, called sniffers, of which most notable examples include *TcpDump*[1] or *Wireshark*.[2] They rely on passive analysis of the network traffic transmitted within the network. Due to this characteristic, such activities are typically difficult to detect.

Currently, in the existing literature there are several approaches for sniffing detection that have been proposed [10]–[13]. They mostly focus on trying to determine whether a wired or wireless Network Interface Controller (NIC) is set to the promiscuous mode, which typically is used to diagnose network connectivity issues. In this case, NIC forwards all frames allowing the machine to analyze them even if they are intended for other devices in the network. It must be noted that in a non-promiscuous mode, unless the

The associate editor coordinating the review of this manuscript and approving it for publication was Ana Lucila Sandoval Orozco.

[1]https://www.tcpdump.org
[2]https://www.wireshark.org

frame is addressed to that specific NIC's MAC address (or is a broadcast or a multicast frame), it is dropped.

However, as we reveal in this article, existing sniffing detection approaches are mostly outdated and the majority are not effective any more. That is why further research is needed to establish which defensive measures can be utilized to identify such activities. Additionally, novel approaches are needed to effectively deal with such threats.

We decided to address a sniffing attack within the security framework of the Internet of Radio-Light (IoRL) system which is being created within the Horizon 2020 project. It aims at developing an architecture for smart buildings [1], supermarkets, museums or even train stations [2] using a 5G Radio-Light multi-component carrier, Frequency Division Duplex (FDD) broadband system consisting of a Visible Light Communication (VLC) downlink channel in the unlicensed THz spectrum and mmWave up/downlink channels in the unlicensed 30-300 GHz spectrum. IoRL allows wireless communication networks to be deployed in buildings that can provide data rates greater than 10 Gbit/s, latencies of less than 1 ms and location accuracy of less than 10 cm, whilst reducing the EMF levels and interference, lowering the energy consumption at the transmitter/receiver and increasing the User Equipment (UE) energy battery lifetime.

However, apart from the obvious benefits that such a system can offer to the users, some challenges and issues must be addressed first. As IoRL integrates various networking technologies, i.e. VLC, mmWave, SDN, WLAN and eNB/HeNB, and each of them is characterized with a specific set of features, there are potential security threats and vulnerabilities that are still often not completely resolved and still need addressing.

That is why, for the IoRL system a dedicated Integrated Security Framework (ISF) has been designed and is continuously developed within the project to mitigate various threats and provide an adequate level of security [3], [4]. While designing the ISF we have utilized experiences and analyzed solutions from existing finished or ongoing 5G security projects and standardization activities. A key, fundamental technology used within the ISF is Software-Defined Networking (SDN) that can easily realize important security features like security monitoring and management. Based on the SDN controller, a centralized system for security monitoring and manageability, providing near real-time awareness of network incidents status and effective enforcement of security policy can be designed and developed. For this purpose, security-related Virtual Network Functions (VNFs) can be created that will perform various security functions. Such a solution will also work effectively by enabling correlation, aggregation and analysis of the security-related data originating from different sources to provide a complete network-wide view of the security posture (security analytics).

In our previous work [5] we presented initial results for the developed detection method that relies on inflicting artificial load on the investigated machine and measuring its Round-Trip Time (RTT) with and without the load. The proposed solution turned out to be effective, especially for Windows-based machines. This article can be treated as an extended version of [5], however, here we make the following novel contributions:

- we develop a novel measurement-based sniffing detection method that relies on network traffic probing and additionally uses machine learning techniques,
- we perform a systematic and detailed experimental evaluation of the proposed solution,
- we use not only ICMP, but also an application layer protocol (HTTP) for traffic probing,
- we also propose how to detect sniffing on Linux hosts, which was not addressed in the initial work [5] and
- in the experimental part we also present a performance evaluation using features like CPU load and variable 'flooding'/'idle' period lengths.

The rest of this article is structured as follows. Section II provides a review of the existing approaches for the sniffing detection. Next, in Section III, we verify whether the previously proposed methods and tools can be still used to discover such threats. In Section IV we present the design and implementation of our own approach for sniffing detection. Section V contains the obtained experimental results and a description of the fine-tuning of the proposed solution to improve its efficiency. Finally, Section VI concludes our work and provides insights for potential future work in this direction.

II. RELATED WORK

In this section we first review the existing research work related to sniffing detection, and then we present the tools that are publicly available and can be used to defend networks against such a threat.

To the best of the authors' knowledge, currently there are no papers or work related solely to network sniffing detection that utilizes Machine Learning (ML)/Artificial Intelligence (AI). However, currently both ML and AI are commonly used for many cybersecurity related tasks. For example, a survey [6] was published by Ghaffarian and Shahriari about software vulnerability analysis and discovery using Machine-Learning and Data-Mining Techniques. Likewise, another survey [7] was published recently by Xin *et al.* about using Machine Learning for an efficient cybersecurity intrusion detection system. The security of the AI itself is also under researchers' interest. In [8] the authors present how to prepare malicious input for AI processes, which will greatly decrease its effectiveness. Finally, in [9] the authors presented the SecureML system for scalable privacy-preserving Machine Learning, which helps with massive data collection that raises privacy concerns. This proves that the utilization of ML/AI is a promising research direction in cybersecurity. In the reminder of this section we will focus on reviewing the existing research work and tools related to sniffing detection.

A. EXISTING RESEARCH WORK ON SNIFFING DETECTION

The existing literature related to sniffing detection is typically divided into [10] *(i)* detection at a local host or *(ii)* at Local Area Network (LAN) segment levels. An example of a local host detection is presented in [14]. The authors detect a sniffer by running a special, centrally managed agent on every device connected to the network. This agent compares the destination address of every captured Ethernet frame and matches it with the local address. If they are not equal, a sniffer might be running on such a device. Unfortunately, this solution assumes that every device is under administrator control, and that such an agent can be deployed. In this scenario, a more sophisticated solution like Host-based Intrusion Detection System (HIDS) should be used. There is also the possibility of compromising such a device by the attacker, disabling or reconfiguring agent so it always reports ''no sniffer''. Our solution is more general, we assume that each device is a black-box that cannot be controlled from the inside (like in every public, dynamic environment). In the rest of this article we focus only on network-based solutions, thus we omit a description of host-based approaches.

Existing network-based sniffer detection methods can be divided into two main subgroups:

- *Challenge-based* where the defender tries to provoke the sniffing machine into responding by sending specifically crafted network traffic. Typically, packets with a forged MAC addresses or Domain Name System (DNS) messages are utilized for this purpose or
- *Measurement-based* where the suspicious machine is subjected to temporary, intentional traffic load and then based on its response it is determined whether the sniffer is active or not.

1) CHALLENGE-BASED APPROACHES

As previously mentioned, the aim of such methods is to stimulate the sniffing machine into responding to the intentionally crafted network traffic. It exploits the fact that often sniffers are not fully passive and tend to respond to certain types of traffic.

Typically, packets with forged MAC addresses are used for this purpose. In a normal situation, such packets would be rejected by the NIC and therefore never reach the operating system (OS) for processing. However, when the NIC is in the promiscuous mode, some OSs treat such traffic as legitimate and respond accordingly, betraying that it is sniffing. Previously proposed methods of this kind include the utilization of specially crafted packets. For this purpose, ARP requests with intentionally incorrect group MAC addresses set can be used and sent to the suspected machine [11]. The other approach uses an ICMP-based method in which the messages of this protocol are sent to an investigated target with the correct destination IP address, but with a forged destination MAC address [10]. Another flavor of the same approach is related to the utilization of DNS messages to provoke the sniffing machine to respond [12]. It must be noted, that by default most sniffers perform a reverse DNS lookup on the traffic that is sniffed. Often for this purpose specially crafted packets with unused IP addresses are utilized, and when encountered by a sniffer DNS lookup request can be generated and discovered by the defender. A similar approach presented in [15] is based on a decoy and honeypot. Specially crafted traffic, containing for instance, a FTP logging session with unique and fake credentials (login and password) is sent to the host, suspected of running a sniffer. If the attacker is actively monitoring the captured traffic, or looking for specific signatures like login session, it might try to use the bait. On the FTP server side, a logging attempt with such unique credentials will prove that the sniffer captured the crafted traffic. Unfortunately, this solution assumes that the attacker will try to use the captured information, but this might never happen as the attacker could be looking for a different type of data or simply not use it at all.

Obviously, a sophisticated attacker running a sniffer is able to thwart replies to any network traffic while sniffing is in progress, thus, making the above mentioned techniques of limited usefulness.

2) MEASUREMENT-BASED APPROACHES

These methods rely on inflicting intentional, artificial load on the suspicious machine by sending specially crafted types of network traffic and observing its performance degradation in response to this traffic, thus measuring the RTT. The rationale for such solutions is that when the NIC is activated in a promiscuous mode, then instead of dropping traffic not directed to this particular device, the OS is overloaded and this causes noticeable performance decreases. Such approaches have been proposed, for example, in [12] and [11].

There are several issues related to the existing solutions of this kind. First of all, the proposed methods can be influenced if there is heavy traffic within the monitored LAN. Second, if not handled with care, as this technique assumes flooding the target machine with potentially heavy traffic, it can lead to unintentional Denial of Service (DoS) attacks.

B. EXISTING ANTISNIFFING TOOLS

While conducting this research we analyzed the tools that aim to allow sniffer detection. Below we present their descriptions, and then in subsection III-A the results verifying whether they are still effective when used in the current communication networks. The analyzed tools include *Promqry*, *Sniffdet*, *Anti-sniffer* and *Nmap sniffer-detect NSE*.

Promqry[3] is a Microsoft tool from 2012 that can be used to detect network interfaces that are running in promiscuous mode. *Promqry* can determine if a Windows 2000 or later based device has network interfaces in a promiscuous mode, which can be a sign that a network sniffer is running on the system. This tool functions by listing the interfaces in a promiscuous mode on the local machine by requesting information on the interface status through WinAPI. To analyze

[3]https://www.microsoft.com/en-us/download/details.aspx?id=1851

external machines, an IP address range can be specified, then these addresses are checked using a ping query, and if the host is running/online, the tool attempts to connect to the host with a Remote Procedure Call (RPC) and check the interface status to discover if the promiscuous mode is used.

Sniffdet[4] is an open source tool under the GPL license that aims at combining a set of tests for remote sniffer detection in TCP/IP network environments. These tests utilize DNS, ICMP, ARP traffic or latency measurements to determine whether the machine's NIC is running in the promiscuous mode. The latest version is from 2006. The tool itself is not well documented, therefore, it is hard to describe how exactly it works. By executing it and viewing its configuration files our assumptions on specific tests are:

- *DNS-based*: sends spoofed packets with fake MAC and IP addresses to the examined machine. If the tested host has an interface in the promiscuous mode it may perform a revDNS query for the fake IP address,
- *ICMP-based*: sends an ICMP echo request to the examined machine with a fake MAC and correct IP address. Again, if the tested host has an interface in the promiscuous mode it may reveal itself by replying to the ping,
- *ARP-based*: is similar to the ICMP-based test, but instead of a ping the ARP protocol is utilized and
- *Latency*-based: using a ping tool it measures the RTT when flooding the target with custom packets.

Anti-sniffer[5] is a similar tool to *Sniffdet* and it was destined for operating systems like Windows 95, 98, NT, NetBSD and Linux. It performs various forms of tests like OS specific tests, DNS tests and network latency ones.

Nmap sniffer-detect NSE[6] is part of the well-known *Nmap* scanner and works by exploiting incomplete packet examination by various operating systems.[7] It sends ARP packets to the sniffing host with the destination MAC address crafted in such a way that it resembles a broadcast or a multicast address. If the host is not sniffing and its network card is in the normal mode of operation, the packet will be dropped by the NIC. However, if the host is sniffing, the packet will reach the network stack of the operating system. If the network stack of the target system checks the destination MAC only partially, which is the case for a couple of the OSes that the tool has built-in support, such a crafted ARP packet will receive a response.

Compared to the existing mechanisms and tools that are presented above, it must be emphasized that the novel approach, which we propose in this article, is also measurement-based and relies on inflicting an artificial load on the investigated machine and monitoring of the changes in responses. However, based on the experimental evaluation of the existing approaches it turned out that they are not successful any more (see Section III). Therefore, to detect sniffing we propose a different technique that is based on the *macof* tool,[8] which temporary floods the targeted device with TCP packets containing randomized MAC addresses and then uses the *ping* to measure the RTT and the *curl* tool download data rate in the corresponding time periods. Then based on the received responses during the 'flooding' period as well as during the 'idle' ones our solution can determine with high probability whether the sniffer is active or not.

III. EVALUATION OF FEASIBILITY OF EXISTING TOOLS & METHODS FOR SNIFFER DETECTION

In this section we verify the existing tools and methods for sniffing detection described in subsection II-B. We want to evaluate if they are still applicable and effective in today's networks.

A. EVALUATION OF EXISTING ANTISNIFFING TOOLS

Promqry is able to detect a promiscuous mode in Windows (and only for this OS), however, to be successful it requires an active participation from the sniffing host (due to the RPC usage), and thus we consider such a behavior as out of scope for this article as we do not assume that the defender has any control over the inspected host.

Despite the fact that the last version of the *Sniffdet* tool is from 2006, we were able to compile it on the machine Supermicro X10DRi with 2 x Xeon E5-2637 v4 @ 3.50GHz, 64GB RAM and Fedora 29 Linux installed. However only three (ICMP-based, ARP-based, and latency-based) out of the four detection methods could be tested as while executing the DNS-based test we experienced an application crash. During our experiments with this tool two physical servers were used, one with a Fedora 29 Linux acting as a testing host and the other with a Windows 8.1 Enterprise N x64 acting as a sniffing host. They were both directly connected using an Ethernet cable. *Sniffdet* was executed on the Linux host with all three working tests against the Windows server with and without Wireshark sniffer running. As stated before, DNS-based test causes the application to crash without any useful output. Both the ICMP and latency test were not able to properly detect the usage of Wireshark on the tested machine – *Sniffdet* reported negative results every time it was executed. Finally, the ARP-based test successfully reported the sniffer existence on the examined host when indeed the Wireshark was running. On the other hand, when sniffing was disabled, *Sniffdet* froze for many hours without any useful output. Due to this fact, the tool could not be examined properly.

The *LOpht AntiSniff v1.02.1* could only be installed on Windows XP due to its installer being a 16-bit application. Unfortunately, the application itself, after successful installation on a Windows XP failed to recognize the network adapter being present in the system, which prevented us from testing its effectiveness.

[4]Sniffdet URL: http://sniffdet.sourceforge.net
[5]http://www.firewall.cx/downloads/network-sniffers-packet-capturing/4-anti-sniffer-v102.html
[6]https://svn.nmap.org/nmap/scripts/sniffer-detect.nse
[7]http://www.securityfriday.com/promiscuous_detection_01.pdf
[8]https://linux.die.net/man/8/macof

Nmap sniffer-detect NSE was tested in three scenarios:

1) A physical machine probing another physical machine (our server probing Raspberry Pi gen.1, Raspberry Pi gen.3 and the Tower PC) — directly connected via an Ethernet cable,
2) A physical machine probing a virtual machine (our server probing its own guest VMs) and
3) A virtual machine probing a virtual machine (guest VMs on our server probing each other).

The operating systems installed on the machines were:

- Server — Centos 7.6.1810,
- Raspberry Pi gen.1 — Raspbian 9.6,
- Raspberry Pi gen.3 — Fedora 29,
- Two VMs — Centos 7.5.1804,
- Tower PC — Windows 8.

In the first scenario, *nmap* reported a negative result for all three probed machines, regardless if the sniffer was active or not. In the second and third scenario we determined that the result depends on the type of the virtual network card attached to the virtual machine. If the *virtio* card was installed, the tool detected sniffing regardless of if it was actually ongoing. On the other hand, if the *e1000* card was installed the sniffing was correctly detected, but the application crashed when the machine was not eavesdropping.

To summarize, our experiments have shown that publicly available tools and described methods are no longer useful. It is not possible to use them to clearly determine whether a sniffer is running on a given host or not.

B. EXISTING DETECTION METHODS

In the following subsections we describe different possibilities for sniffing detection.

1) DNS-BASED APPROACH

As previously mentioned, sniffers, in their default configuration, try to reverse-lookup the encountered IP addresses in the DNS records for the convenience of the user. It is possible to exploit this behaviour by providing the host suspected of sniffing with packets destined to a fake IP address. If the machine is running a sniffer, it will try to resolve the fake IP via reverse-DNS requests, which can be monitored by a network administrator.

2) FORGED MAC ADDRESSES APPROACH

During normal operation, the firmware of a network interface drops Ethernet frames that are irrelevant to the host. This means that frames with the destination MAC address not matching exactly the adapter's own or not being the special broadcast address, are removed to avoid their unnecessary processing in the network stack of the operating system. If this behaviour is disabled, which is done by switching the network card to the promiscuous mode, then all transmitted packets reach the operating system and are available and processed by the sniffing program.

The forged MAC address method [10] relies on the OS's network stack characteristic feature, i.e., lack of additional MAC address verification, which is a sensible optimization, as in a normal scenario this would be a double-check and a waste of the kernel's processing time.

Sending an IP packet, e.g., opening a TCP connection, UDP datagram or ICMP echo request with an invalid destination MAC address, but correct destination IP to a benign host should result in no action, because the packet will be dropped by the adapter. On the other hand, if encountered by a sniffing host then a response will be generated, because in that case the MAC address is no longer checked by the NIC (running in the promiscuous mode) and the higher layer is correct (IP address matches the host's).

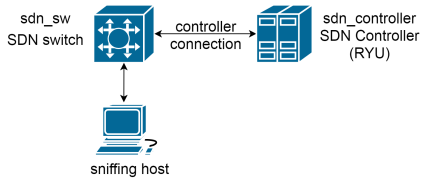


FIGURE 1. Experimental testbed to evaluate feasibility of DNS-based and forged MAC address approaches.

3) FEASIBILITY OF DNS-BASED AND FORGED MAC ADDRESS-BASED SOLUTIONS

The first two detection scenarios were realized in our virtualized SDN environment as illustrated in Figure 1. We utilized *Open vSwitch* version 2.10.1 and *Ryu* controller version 4.30. We developed a custom *Ryu* module that instructs the *Open vSwitch* to forge packets and send them to the sniffing host.

During the DNS-based experiments we sent TCP segments to the sniffing host with:

- randomized source and destination MAC addresses,
- randomized source and destination IP addresses,
- randomized source and destination TCP ports,
- randomized SEQ and ACK numbers and
- two null-byte payload.

It must be noted that a MAC address cannot be fully random, as for instance, multicast frames can be ignored by the network equipment (switches) or NICs. Such a frame would never reach the sniffing host's TCP/IP stack, thus a false-negative situation can occur.

Using this approach we were able to detect reverse DNS requests on the SDN switch coming from the examined machine when the *TcpDump* (with the default configuration) was running, which matched the randomized IP addresses we provided in the forged packets.

During the MAC-based experiment we sent a correct ICMP echo request to the sniffing machine, but with a randomized destination MAC address.

However, it must be noted that we were unable to replicate the behaviour previously reported in [10], i.e., a response when the machine is sniffing on the following operating systems:

- Fedora 29 Linux (with kernel version 5.0.5),
- Windows XP Professional Service Pack 3,
- Windows 7 Professional Service Pack 1,
- Windows 8.1 Pro and
- Windows 10 Education.

This suggests that this sniffing detection method is no longer effective for relatively modern OSes.

It is also worth noting that due to the nature of the two detection methods presented above, i.e., that each mechanism is either successful in sniffing detection or not and practically no parameter tuning is possible, we verified only that they are still feasible (so the technique still can be successfully used). Note, that the nature of this method makes it difficult to provide any numerical results.

To summarize, the DNS-based approach can still be useful in sniffing detection, however, the MAC-based approach is not applicable any more. That is why we incorporated the DNS-based detection method within the developed IoRL security framework.

IV. PROPOSED ARTIFICIAL LOAD-BASED SNIFFING DETECTION METHOD

As previously discussed in Section I, the main motivation for developing countermeasures against sniffing attacks is to enrich the Integrated Security Framework of the IoRL system. That is the main reason why we developed and evaluated the proposed defensive mechanism within the SDN-based environment.

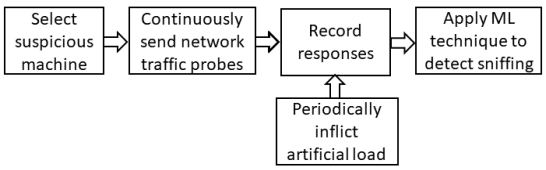


FIGURE 2. Overview of steps of proposed artificial load-based detection method.

The proposed detection method targets the sniffing application directly. The general concept of this approach is illustrated in Figure 2. The suspected machine is first selected according to the adopted security policy, e.g, every host is periodically under investigation for a certain amount of time. Then, the investigated machine is continuously probed using, for instance, a *ping* tool or other tool that allows the measurement and collection of the resulting response times. Periodically, we execute the *macof* utility to flood the suspected machine with a large number of packets with randomized MAC addresses to inflict an artificial load.

If the machine is not sniffing it is expected that there would be no differences between the response times with and without *macof* as the flooding packets will be simply discarded at the hardware level, and thus not interfere with the operating system.

However, if the machine is sniffing then each of the flooded packets will be recorded, analyzed or displayed to the user.

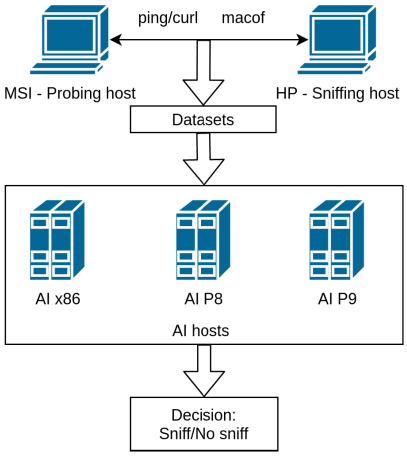


FIGURE 3. Experimental test-bed for proposed artificial load-based detection method.

If the volume of the traffic is high enough, then the packet processing in the sniffing program should engage a significant fraction of the processing power, thus, there could be a notable difference in the response times between the periods with and without the flooding with *macof*.

To discover these fluctuations in the response times we employ ML algorithms. Thus, based on the collected results, machine learning techniques are utilized to decide whether the host is running a sniffer or not.

In the remainder of this section we present the developed experimental test-bed and methodology for the proposed sniffer detection method.

A. EXPERIMENTAL TEST-BED

In the rest of this subsection we describe the details of the test-bed. For this approach we temporarily resigned from the SDN and controller-generated packets due to a low maximum throughput of the solution (in our tests, around 1,000 packets per second). However, it must be noted that SDN can still be utilized as an efficient means to block all the incoming and outgoing traffic from the host suspected of running the sniffer. Moreover, the SDN is a core component of our security framework as well as in the IoRL network itself. In such a case, using the SDN-based solution for blocking sniffing attempts and other nefarious actions seems to be a natural choice.

The utilized test-bed includes two setups:

- two hosts for running the measurement-based experiment,
- three hosts for training AI software and testing obtained results.

As our solution is based on a previously described measurement-based approach, two hosts were connected directly via an Ethernet cable without an Ethernet switch between them. The reason for such a setup is that a switch would introduce additional delays, which can change during long-running experiments. The measured values of, for example, the ping RTT are very small (less than one millisecond),

thus for the measurement-based approach even a small deviation can be of significance. The resulting test-bed is presented in Figure 3.

Moreover, the list of utilized hardware within the test-bed is as follows:

- Probing host MSI GL72 with Intel i7-6700HQ 4-core, 8-threads, 32GB RAM and RedHat Enterprise Linux 7.7,
- Sniffing host HP EliteBook 820 G3 with Intel i5-6300U, 2-core, 4-threads, 16GB RAM and RedHat Enterprise Linux 7.7,
- AI host x86 Dell PowerEdge R640 with Intel Xeon Gold 6140, 18-core, 36-threads, 64GB RAM and Ubuntu 16.04 LTS,
- AI host P8 AC822 with IBM POWER8 20-core, 160-threads, 256GB RAM, 4x Tesla P100-SXM2-16GB GPU and RedHat Enterprise Linux 7.7 and
- AI host P9 AC922 with IBM POWER9 32-core, 128-threads, 512GB RAM, 4x Tesla V100-SXM2-16GB GPU and RedHat Enterprise Linux 7.7.

It should be noted that AC922 is a building block of the very first of the TOP500 supercomputers, called Summit, for the last 1.5 years.[9] It is also considered as one of the best AI platforms on the market.[10] Additionally, AC922 is a building block of the second TOP500 supercomputer, called Sierra, for the last 1.5 years as well.[11]

During the experiments we also utilized the following software:

- *tcpdump-4.9.2* – a command-line sniffer,
- *curl-7.29* – a command-line tool for transferring data using various protocols, HTTP among others,[12]
- *apache-2.4.6* – an efficient web server,[13]
- modified version of *macof* tool (which is a part of the Dsniff suit toolset)[14] and
- *Driveless AI dai*-1.8.0 – Automatic Machine Learning Platform[15] with *CUDA 10.2* – parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs).[16]

We modified the source code of *macof* to control the number of generated packets per second.

B. EXPERIMENTAL METHODOLOGY

Compared to the experimental methodology that we used in our previous work [5], we not only utilize *ping*-based RTT for measurements but also the data rate of the file transfer via the HTTP protocol. Moreover, we use a Linux operating system on the sniffing host instead of a Windows OS, as in [5] the used method turned out to be unsuccessful for Linux-based hosts. Additionally, we are using variable 'idle' and 'flooding' periods of 5, 10, 15 and 30s. Another novel aspect is that we are not only testing the behavior of the sniffing host running on battery and AC power, but also under heavy CPU consumption and idle states, which better demonstrate real-life sniffing host functions (as the sniffing could be not the only activity running on the sniffing host).

Two connected hosts: probing (MSI) and sniffing (HP) were used during the first phase of the experiment, i.e., measurement-based. The probing machine was responsible for both generating the flood of packets using *macof* and establishing the baseline for the measured responses.

To prove that not only ping RTT can be used as a measurement metric (based on conclusions presented from the related work section), but also application layer protocols, HTTP was used as an example.

Therefore, we assume that the sniffing host is serving any service over the network, HTTP for example, and there is access to such a service via a firewall. This might be an unusual situation, but such an approach serves as an example of using application layer protocols instead of ICMP (ping). Obviously, the utilization of other protocols is feasible as well, however, further studies in this aspect are needed. In [5], *ping*-based RTT was utilized to establish the baseline and this solution was tested here as well, but for the Linux OS as a sniffing host.

As in [5], we determined that in our experimental environment the probing machine using *macof* could handle a maximum packet flooding of around 10,000 packets per second. This is why we used this setting for conducting all our experiments.

Taking the above into account, the following experiments were performed:

- For the whole duration of the experiment, we either executed the *ping* with the lowest possible interval (0.01s) targeted at the suspicious host and recorded the corresponding RTTs or utilized the *curl* tool to download a 1MB file from the suspicious host and recorded the download data rate (in B/s),
- Then for the next 10, 20, 30 or 60s we activated the modified version of the *macof* tool, however, for the first half of this period (i.e., 5, 10, 15 or 30s) it is idle (i.e., no packets are sent - in the rest of the paper we will call it an 'idle' period) and for the rest of the time (i.e., 5, 10, 15 or 30s) it generates 10,000 packets per second ('flooding' period). In both cases we noted the corresponding *ping* RTTs or *curl* download data rate,
- In half of the cases for both of these periods the sniffer was activated and in the remaining time it stayed idle (sniffing/no sniffing). This allowed us to compare the delays with the artificially inflicted load for the case of a normal user machine and for a device with the NIC set to the promiscuous mode and
- In terms of the sniffing host, another two factors were taken into consideration: working on battery/AC power

[9] https://www.top500.org/system/179397
[10] https://www.ibm.com/uk-en/marketplace/power-systems-ac922
[11] https://www.top500.org/system/179398
[12] http://curl.haxx.se
[13] http://httpd.apache.org
[14] https://monkey.org/∼ dugsong/dsniff
[15] https://www.h2o.ai/products/h2o-driverless-ai
[16] https://developer.nvidia.com/cuda-zone

and CPU intensive load/idle. The load has been inflicted based on computing the MD5 checksum from the */dev/zero* pseudo-device. As the sniffing host (HP) had four threads, four such jobs were executed, which caused 100% CPU utilization.

Experiments were executed for all considered configurations, therefore, in total 64 experiments were executed:

- 2x for curl/ping,
- 2x for battery/AC power,
- 2x for CPU load/idle states,
- 2x for sniffing/no sniffing activities and
- 4x 5, 10, 15, 30s 'idle'/'flooding' periods.

In every experiment 100 'idle' and 100 'flooding' periods (5, 10, 15, 30s) were recorded. In total, more than 53 hours of network traffic was recorded across all experiments. The longest experimental group using the 30s period was recorded for more than 26 hours (one experiment took 1h 40min times 16 for four binary parameters).

When the *ping*-based RTT and *curl* download data rate were finally recorded, they were used in the last phase of the experiment – sniffing detection using ML techniques. For this purpose we utilized Driveless AI (dai) software, which is an AI framework, and it provides many useful features and supports GPU processing. One of the most important features is the auto-tuning of the ML algorithms, which allows a focus on the main problem (which is sniffing detection) and not tuning ML parameters.

It should be noted that our solution assumes that we do not have access to the operating system of the host being suspected of running a sniffer. In the IoRL project, the network is open, dynamic and available to all users. Thus, in such a scenario, some of them may perform malicious activity (like sniffing). In the closed environments, dedicated software, like HIDS or antivirus software, can be pre-installed and prevent or detect the usage of such forbidden software. In fact, many simple tools (like *ps* in Linux) can be easily utilized to check whether a sniffer is running or not. However, note that in our case we have to treat every device as a black-box. Therefore, we assume that we are not able to examine whether the sniffing software is running on the machine from the host-level perspective.

Obviously, differences in the response times of the probed host (i.e., expressed as ping RTT or curl download data rate) can be a result of the OS load caused by running an application not necessarily related to the sniffing activity. We addressed that issue by adding two factors that we mentioned before, i.e., running on battery/AC power and by introducing an additional CPU load. This will simulate different behaviours in the suspected host.

The next section presents the obtained results for the proposed method.

V. EXPERIMENTAL RESULTS

In this section we present and describe the results of our experiments. Firstly, we will explain whether running a sniffer on the host will result in differences in its response to the measurements, which will justify using this method. Next, we will describe the obtained experimental results of the proposed detection solution, empowered by ML, using raw data and statistical representations of gathered datasets.

A. RATIONALE BEHIND PROPOSED METHOD

First we want to establish whether there is indeed no significant difference between the ping response times and in *curl* download data rates with and without flooding with the *macof* when the sniffer is not active. The obtained experimental results confirm this observation and they are presented in Figures 4a, 5a, 6a and 7a.

Surprisingly, when the sniffer is active, the *ping* response time could be increased or decreased during the *macof* flooding, depending on the type of the experiment – see Figures 4b and 5b.

Finally, if we apply an artificial CPU load, the ping response time is noticeably higher, when comparing 'idle' and 'flooding' periods, which is an unexpected result (Figure 5b). This can be explained by intelligent overclocking technology implemented on modern Intel CPUs. On the other hand, in a different experiment (running on battery, CPU idle), the *ping* response time is decreased, which confirms our initial assumptions (Figure 4b).

For the case when the sniffer is active for the experiment where the inspected host is running on a battery and its CPU is idle, the resulting *curl* download data rate surprisingly increases (see Figure 6b). In the other performed experiment, where the host was on AC power and under CPU load the result is very ambiguous. Thus, there is no noticeable difference between the 'idle' and the 'flooding' periods with and without an active sniffer (see Figure 7a and 7b).

Different results from the *ping*-based and *curl*-based experiments can be explained by the fact that the *ping* is handled in the Linux kernel, which has higher priority than user-space applications, like an HTTP server.

Note also that in our previous work [5], for each 'idle' and 'flooding' period, the mean, the median and the standard deviation of the obtained *ping*-based RTT were calculated. Then based on the obtained values three markers were determined using equation:

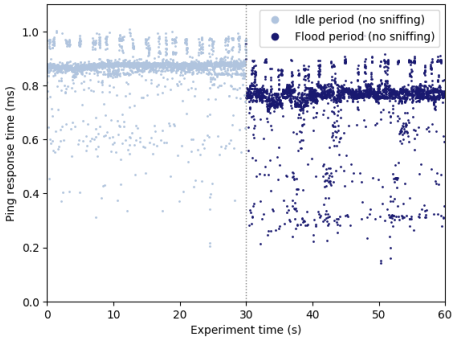
$$marker(x) = \left| \frac{x_{flooding}}{x_{idle}} - 1 \right|, \tag{1}$$

where x is the mean, the median or the standard deviation.

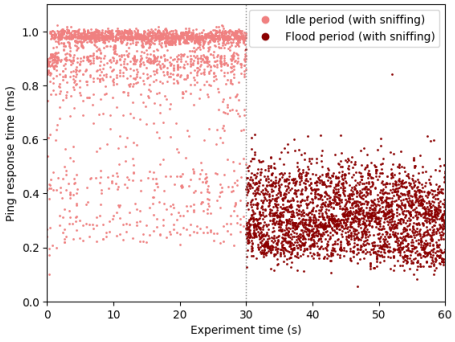
It turned out, that using such metrics worked well for detecting the sniffer on the Windows OS. However, in the case of the Linux OS the method did not yield satisfactory results. That is why, to improve this situation in this article we decided to use ML techniques to solve this problem.

B. ML-BASED APPROACH USING RAW DATA FROM MEASUREMENTS

First, we want to evaluate how the proposed ML-based detection method would work on the raw data coming directly from

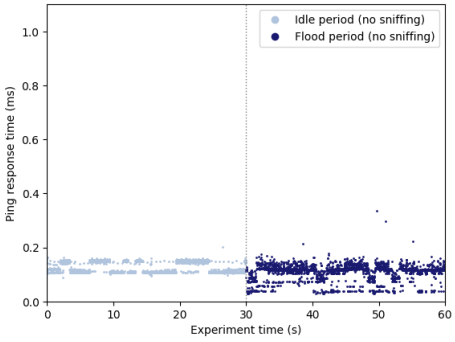


(a) Sniffing inactive: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).

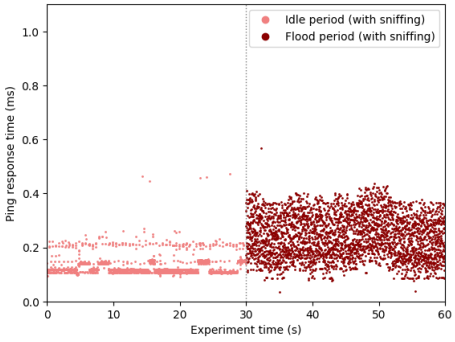


(b) Sniffing active: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).

FIGURE 4. Exemplary experiment for ping response time (running on battery, CPU idle) – sniffing inactive (a) and active (b).



(a) Sniffing inactive: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).



(b) Sniffing active: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).

FIGURE 5. Exemplary experiment for ping response time (AC power, CPU load) – sniffing inactive (a) and active (b).

the measurements. To verify this the results of the performed experiments described in subsection IV-B were saved in a CSV file. These files were then used as an input for the *dai* software. During the first run of the experiments, we decided to use the recorded raw data (instead of mean, median, and standard deviation of 'idle' and 'flooding' period). The CSV file contained the data described below:

- *FlagCPU* (Boolean): '0' means that the sniffing host was idle with only the sniffer running, while '1' denotes that additional CPU greedy processes were running,
- *FlagAC* (Boolean): '0' means that the sniffing host was running on the battery and '1' indicates that the sniffing host was connected to the AC power supply,
- *FlagFlood* (Boolean): '0' means that the *macof* tool was operating in the 'idle' mode while '1' denotes that it was operating in the 'flooding' mode,
- *Value* (Real number): measured the *ping* RTT [ms] or *curl* download data rate [B/s] and
- *FlagSniff* (Boolean): '1' means that the sniffer was running on the sniffing host, while '0' indicates that the sniffer was not activated. This was the parameter we wanted to determine using the ML-based approach.

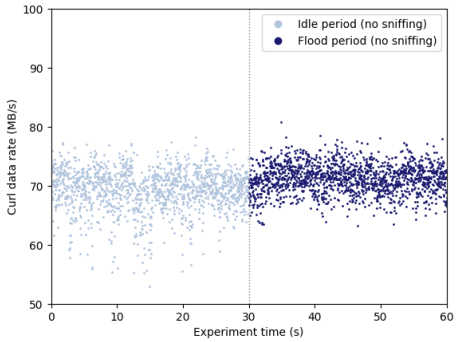
Table 1 presents exemplary rows of such a file for the *ping*-based RTT experiment.

TABLE 1. Results of first phase of sniffing detection – exemplary rows of CSV file for *ping*-based RTT.

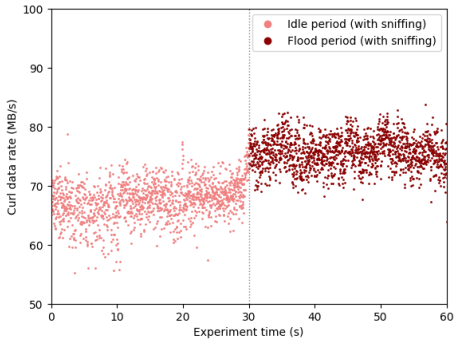
FlagCPU	FlagAC	FlagFlood	Value [ms]	FlagSniff
1	0	0	0.22	0
0	1	1	0.34	1
1	1	0	0.41	0
0	0	1	0.12	1

It should be noted that during real-world sniffing detection, the probing host is unaware of whether the sniffing host is running on battery or AC power and, moreover, whether it is under a CPU heavy load or not. That is why all the data from these cases were merged into one file. The only fact known by the probing host was whether the *macof* tool was operating in 'idle' or 'flooding' periods, thus the final CSV as an input for the *dai* software contained following the rows: *FlagFlood*, *Value*, *FlagSniff*.

Experiments performed using *ping* and *curl* tools were executed and processed separately. There are two main reasons behind such a procedure. First of all, the units for *ping* and *curl* are not the same (RTT is in [ms] while download data rate in [B/s]). The second reason is their order of magnitude. An average value for all experiments for *ping*-based RTT was 0.41 ms and for the *curl* download data rate it was 78818100 B/s. Additionally, the experiments for different period lengths (5, 10, 15, 30s) were executed and processed

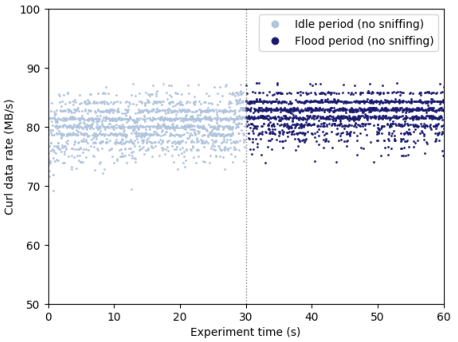


(a) Sniffing inactive: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).

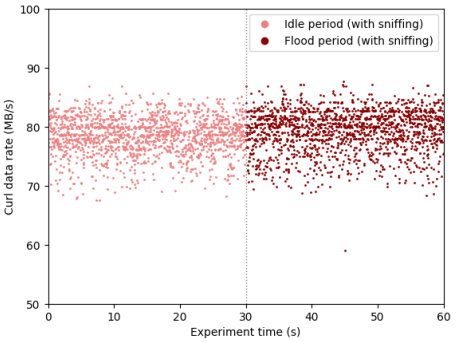


(b) Sniffing active: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).

FIGURE 6. Exemplary experiment for curl download data rate (running on battery, CPU idle) – sniffing inactive (a) and active (b).



(a) Sniffing inactive: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).



(b) Sniffing active: first 30 seconds – 'idle' period (left) and next 30 seconds – 'flooding' period (right).

FIGURE 7. Exemplary experiment for curl download data rate (AC power, CPU load) – sniffing inactive (a) and active (b).

separately as finding the most reasonable value was one of the aims of this research. Last but not least, the ML experiments were executed on three different hosts, i.e., x86, P8, and P9.

As the traffic generated by the *ping* tool directed to the suspicious host was executed with a high rate (100 ICMP packets/s) as well as downloading 1 MB file on a 1 Gbit/s network gives a theoretical data rate of 125 MB/s, the CSV file with recorded values contained many entries (one row per one ping and curl download). Obviously, regarding the *curl* tool, there were some delays in downloading the file in the infinite loop (for every download, current time had to be recorded to match the 'idle' or 'flooding' period), thus the theoretical data rate was also not achievable. As a result, the file was downloaded with the rate of about 65 downloads/s (thus 65MB/s as a 1MB file was downloaded). For the 30s period, the file with all the factors combined (CPU, battery, flooding, sniffer) for the *ping*-based experiments contained more than 4 million entries, and for the *curl*-based more than 2.4 million entries.

Moreover, the input files were divided into training and test parts using the 75:25 ratio. As the *dai* is performing internal validation, dividing the test dataset into test and validation datasets was pointless. On the other hand, external cross validation was performed by additionally shuffling the datasets four times and dividing them again using the 75:25 ratio.

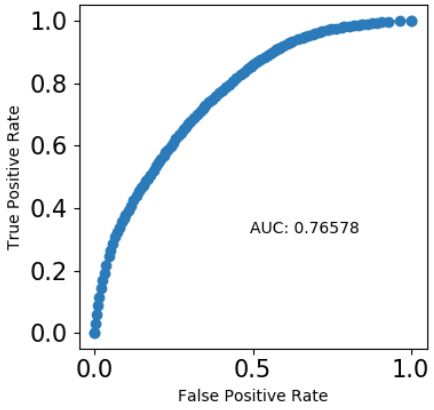


FIGURE 8. ROC curve for *curl*-based experiment with raw data.

Therefore, a total of 40 experiments were executed for each of the three hardware platforms (120 in total):

- 2x for *ping*/*curl*,
- 4x for 5, 10, 15, 30s 'idle'/'flooding' periods and
- 5x for cross validation.

Again the results are unsatisfactory. The best Area Under Curve (AUC) for the Receiver Operating Characteristic (ROC) curve for such an experiment was 0.765, as presented in Figure 8. Moreover, Table 2 presents the test confusion

TABLE 2. Test confusion matrix for *curl*-based experiment with raw data.

	Predicted: 0	Predicted: 1	Error
Actual: 0	105258	143573	58.00%
Actual: 1	22600	226199	9.00%

matrix for this experiment, and it should be noted that the obtained error rate is too high.

C. ML-BASED APPROACH USING MEAN, MEDIAN AND STANDARD DEVIATION REPRESENTATION OF MEASUREMENTS

Next, to improve the results presented in the previous subsection we decided to use statistical values as proposed in our previous work [5]. For each 'idle' and 'flooding' period, the mean, the median and the standard deviation of the obtained *ping*-based RTT value and the *curl* download data rate were calculated. As a result, the input file for the ML-based experiments was orders of magnitude smaller, instead of millions of entries there were only 1600 entries (200 'idle'/'flooding' periods $\times$ 2 CPU load/idle $\times$ 2 battery/AC $\times$ 2 sniffer/no sniffer). This file was split into training and testing datasets (again using the 75:25 ratio) five times for cross validation.

Each CSV file used for training and testing contained the following data:

- *FlagFlood* (Boolean): '0' indicates *macof* was in the 'idle' period, not generating packets; '1' means *macof* was flooding the suspected host with artificial packets,
- *Mean* (real number): *ping* RTT value or a *curl* download data rate for each period,
- *Median* (real number): *ping* RTT or *curl* download data rate median value for each period,
- *Standard Deviation* (real number): *ping*-based RTT result or *curl* download data rate standard deviation value for each period and
- *FlagSniff* (Boolean): '1' means that the sniffer was running on the suspected host; '0' means the sniffer was not running on it. This was the result we were expecting to obtain to determine whether the host is malicious or not.

Providing these statistical measures for each period helped the *dai* to train a model that can reliably predict the FlagSniff, i.e., to discover whether the sniffer is active or not.

1) TIME-RELATED FACTORS

During the sniffing detection, the amount of time that could be spent on this activity should be considered. The sniffer can be running for a long time, but also for the short periods (as the attacker can be aware of the specific process and the sniffer is only activated for a short time). It must be noted that our solution contains three time-related factors:

- T1 – creating input dataset: in our solution it depends on the length of the 'idle'/'flooding' periods,
- T2 – ML training: depends on the processing power of the utilized hardware and
- T3 – sniffing detection using the trained model (negligible comparing to previous components).

In our experiment we use 100 'idle' and 'flooding' periods. This means, that during the first phase (measurement-based), one experiment duration (T1) was:

- 1h 40min for the 30s period,
- 50min for the 15s period,
- $\sim$ 33.33min for the 10s period and
- $\sim$ 16.67min for the 5s period.

Obviously, ML model training is an activity that has to be periodically re-executed, thus the model has to be re-trained or at least re-tested – providing that artificial input data should give the expected results, and if not the model needs to be re-trained. This takes time and, as mentioned, it depends on the performance of the hardware. It must be noted that it is not required to have enterprise-class hardware to perform the training phase. Commercial cloud providers, like IBM or Google, provide special platforms and products for AI applications. On the other hand, a trained model that gives good results, can be used without any concerns. This means that the training time (T2) cannot be considered as a fixed component for every sniffing detection using the proposed solution.

Finally, sniffing detection using the trained ML model does not require high processing power, and what is more important is that it does not require using a GPU. It can be executed on a low-end computer (e.g., a Raspberry PI), and still the results will be obtained very quickly – within a few seconds (T3), which is negligible when compared to, for instance, the 1h 40min for the 30s period (T1).

2) DETECTION RESULTS

Below we describe the exemplary results for the representative dataset with the overall highest ROC curve AUC (thus P9 as AI training host and 30s period) for both *ping* RTT and *curl* download data rates. In both cases the *dai* software found the optimal parameters for *XGBoost* and *LightGBM* ML models by training these models with different parameters. However, in the end, the *LightGBM* was not selected due to the low performance during the model tuning stage. On the other hand, the *XGBoost* seems like a good choice in this case as many winners in the Kaggle's competitions utilize it due to its parallelization, distributed computing, Out-of-Core computing, and cache optimization of the data structures and algorithms [16]. *XGBoost* is an optimized distributed gradient boosting library, designed to be highly efficient, flexible and portable. It implements ML algorithms under the Gradient Boosting framework. It also provides a parallel tree boosting (also known as GBDT and GBM) that solves many data science problems in a fast and accurate way [18].

Figures 9 and 10 present ROC curves for the validation and testing of the ping-based exemplary experiment (P9, 30s). Additionally, Tables 3 and 4 provide confusion matrices for the validation and test phases of the ML technique. It should be noted that the resulting AUC is very high, i.e., 0.999, and thus the error rate is very small.

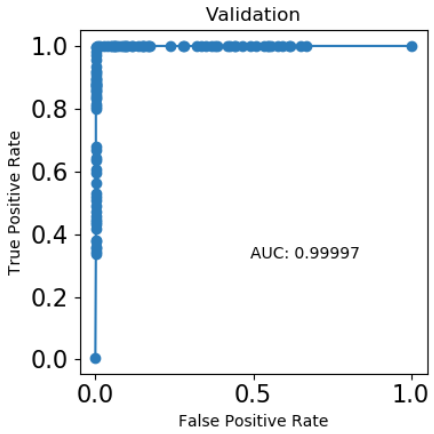


FIGURE 9. ROC curve for exemplary ping-based experiment validation (P9, 30s).

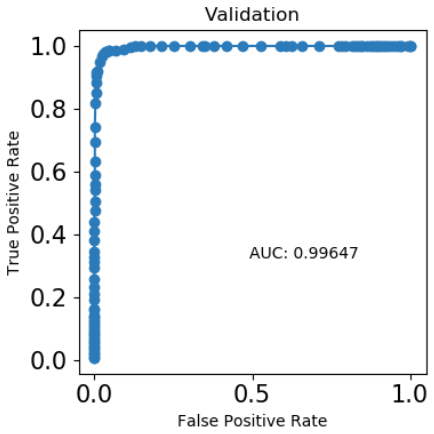


FIGURE 11. ROC curve for exemplary curl-based experiment validation (P9, 30s).

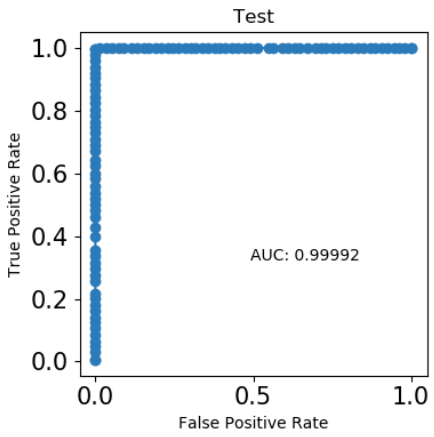


FIGURE 10. ROC curve for an exemplary ping-based experiment testing (P9, 30s).

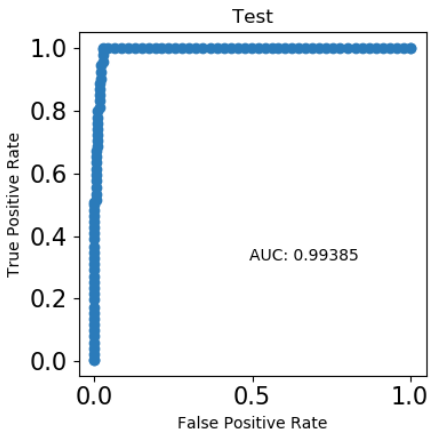


FIGURE 12. ROC curve for exemplary curl-based experiment validation (P9, 30s).

TABLE 3. Validation confusion matrix for *ping*-based experiment (P9, 30s).

	Predicted: 0	Predicted: 1	error
Actual: 0	594	5	1%
Actual: 1	0	601	0%

TABLE 4. Testing confusion matrix for ping-based experiment (P9, 30s).

	Predicted: 0	Predicted: 1	error
Actual: 0	200	1	0%
Actual: 1	1	198	1%

TABLE 5. Validation confusion matrix for *curl*-based experiment (P9, 30s).

	Predicted: 0	Predicted: 1	error
Actual: 0	595	18	3%
Actual: 1	17	570	3%

TABLE 6. Testing confusion matrix for *curl*-based experiment (P9, 30s).

	Predicted: 0	Predicted: 1	error
Actual: 0	182	5	3%
Actual: 1	0	213	0%

The obtained results for the *curl*-based experiment, although still very good, are slightly worse than for the *ping*-based one. Figures 11 and 12 present ROC curves for the validation and testing of the *curl*-based experiment (P9, 30s). Additionally, Tables 5 and 6 provide the confusion matrices for the validation and test phases of the ML algorithms. Again the resulting AUC is very high and it is in the range 0.994-0.996.

The detailed performance of the final models for the exemplary *ping*-based RTT and *curl*-based experiments are presented in Tables 7 and 8. It should be noted that the ROC curve and the AUC were used to evaluate the proposed

TABLE 7. Performance of final model for exemplary *ping*-based experiment (P9, 30s).

Scorer	Final ensemble scores on validation	Final ensemble standard deviation on validation	Final test scores	Final test standard deviation
AUC	0.99997	4.25E-05	0.99992	7.98E-05
ACCURACY	0.99897	0.0011896	0.9975	0.001988
AUCPR	0.99997	3.88E-05	0.99993	8.46E-05
F05	0.99917	0.00090347	0.99899	0.00091542
F1	0.99901	0.0011376	0.99748	0.0020895
F2	0.99958	0.0004852	0.99699	0.0017749
GINI	0.99993	8.49E-05	0.99985	0.00015964
LOGLOSS	0.49563	0.07203	0.4946	0.07203
MACROAUC	0.99997	4.25E-05	0.99992	7.98E-05
MCC	0.99793	0.0023804	0.99501	0.0039787

TABLE 8. Performance of final model for exemplary *curl*-based experiment (P9, 30s).

Scorer	Final ensemble scores on validation	Final ensemble standard deviation on validation	Final test scores	Final test standard deviation
AUC	0.99647	0.0020554	0.99385	0.0032594
ACCURACY	0.98127	0.0075326	0.9875	0.0075326
AUCPR	0.99561	0.0029037	0.99342	0.0035885
F05	0.98489	0.0060528	0.98157	0.0071124
F1	0.98074	0.0074557	0.9884	0.0074557
F2	0.98679	0.0049137	0.99533	0.0049137
GINI	0.99295	0.0041109	0.9877	0.0065187
LOGLOSS	0.08047	0.017053	0.083005	0.01817
MACROAUC	0.99647	0.0020554	0.99385	0.0032594
MCC	0.96264	0.014918	0.97516	0.014918

detection method. Note, that in this case we also utilized other performance indicators that also yielded very good results. These parameters include [17]:

- *AUC* (Area Under the Receiver Operating Characteristic Curve): is used to evaluate how well a binary classification model is able to distinguish between true positives and false positives.
- *Accuracy*: is the number of correct predictions calculated as a ratio of all predictions made.
- *AUCPR* (Area Under the Precision-Recall Curve): is used to evaluate how well a binary classification model is able to distinguish between precision recall pairs or points.
- *F1 score*: provides a measure of how well a binary classifier can classify positive cases (given a threshold value).
- *F0.5 score*: is the weighted harmonic mean of the precision and recall (given a threshold value). Unlike the F1 score, which gives equal weight to precision and recall, the F0.5 score gives more weight to precision than to recall.
- *F2 score*: is the weighted harmonic mean of the precision and recall (given a threshold value). Unlike the F1 score, which gives equal weight to precision and recall, the F2 score gives more weight to the recall than to precision.
- *GINI* (Gini Coefficient): is a well-established method to quantify the inequality among values of a frequency distribution and can be used to measure the quality of a binary classifier.
- *LOGLOSS*: is a logarithmic loss metric that can be used to evaluate the performance of a binomial or multinomial classifier. Only in this case a lower value is better.
- *MACROAUC* (Macro Average of Areas Under the Receiver Operating Characteristic Curves): is for multi-class classification problems, this score is computed by macro-averaging the ROC curves for each class (one per class).
- *MCC* (Matthews Correlation Coefficient): the goal of the MCC metric is to represent the confusion matrix of a model as a single number. The MCC metric combines the true positives, false positives, true negatives, and false negatives.

Additionally, in Tables 9 and 10 we present the final model parameters for the exemplary *ping*-based and *curl*-based experiments. Note, that these parameters were auto-tuned by the *dai* software. The meaning of the parameters is as follows [18]:

- *type*: the specific *XGBoost* model type used,
- *subsample*: the ratio of the training instances,
- *colsample bytree*: the subsample ratio of columns when constructing each tree,
- *max depth*: the maximum depth of a tree,
- *grow policy*: controls the way new nodes are added to the tree,
- *max leaves*: the maximum number of nodes to be added and
- *eta*: the learning rate, step size shrinkage used in update to prevent overfitting,
- *tree method*: the tree construction algorithm.

TABLE 9. Final model parameters for exemplary *ping*-based experiment (P9, 30s).

Type	Colsample bytree	Subsample	Max depth	Grow policy	Max leaves	Eta	Tree method
XGBoostGBMModel	0.6	1	0	lossguide	1024	0.03	gpu_hist

TABLE 10. Final model parameters for exemplary curl-based experiment (P9, 30s).

Type	Colsample bytree	Subsample	Max depth	Grow policy	Max leaves	Eta	Tree method
XGBoostGBMModel	0.8	0.7	0	lossguide	1024	0.03	gpu_hist

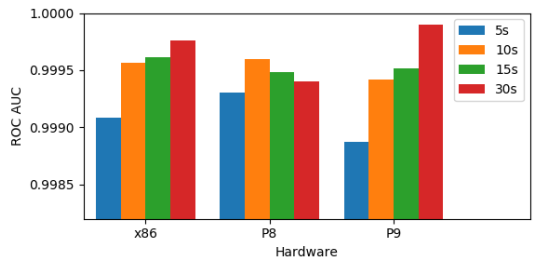


FIGURE 13. Sniffing detection results for *ping*-based experiment on different AI hardware for variable 'idle'/'flooding' periods.

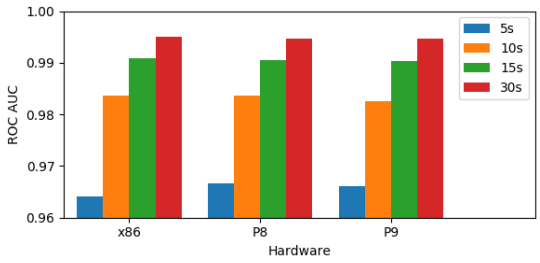


FIGURE 14. Sniffing detection results for *curl*-based experiment on different AI hardware for variable 'idle'/'flooding' periods.

As described previously, for every experiment, the AUC was used as an indicator for the best results. Figure 13 presents the results for all *ping*-based RTT experiments and Figure 14 illustrates the results for experiments based on the *curl* download data rate.

Based on the obtained results it may be concluded that apart from the 5s period, which turned out to be too short for the proposed detection method all other periods (i.e., 10s, 15s and 30s) achieved very good AUC results higher than 0.98 and the impact of the hardware used was negligible.

D. SUMMARY

As with any ML-based solution, it is not possible to provide a threshold for the amount of input data needed to perform the experiment correctly (thus successful detection of the sniffing activity). The general rule is, the more data provided the better and more accurate results can be obtained. Note however, that our main aim in this article was to show that the presented method can be successfully utilized for sniffing detection and not how many 'idle'/'flooding' periods are expected.

Therefore, the obtained results can be summarized as follows:

- It is not important which hardware was used for the ML training. There were some differences between the x86, P8, P9 platforms, but they are negligible. The reason for that is the fact that the ML algorithms are non-deterministic, and for the same input, the output can vary,
- The best results can be obtained for the higher 'idle'/'flooding' periods (i.e., 30s). This is not surprising as we are using statistic measures, therefore, a larger dataset results in more representative values and
- Experiments using the *ping*-based RTT provides better results than experiments based on the *curl* download data rate. It should be noted, that still the worst obtained result, i.e., 0.9641 (x86, curl, 5s), can still be considered as very good, which is comparable with the best results (around 0.97) obtained in [5]. This also proves that the application layer protocols can be utilized for the sniffing detection for the proposed solution as well.

VI. CONCLUSION AND FUTURE WORK

In this article we first revisited the existing sniffing detection solutions and showed that many of them are outdated and thus no longer effective. Motivated by these findings we proposed a novel approach that allows the identification of sniffing hosts, i.e., those that have NICs set to the promiscuous mode. Our approach is measurement-based and uses an artificial traffic load as well as the *ping* and *curl* tools for network traffic probing, together with the ML techniques. The obtained experimental results prove that the proposed detection method is very promising as it achieves an accuracy of 99%.

In our future work we would like to evaluate the proposed detection method with respect to virtualization environments and incorporate it further into the solution developed within the IoRL project. This would allow full utilization of the SDN potential. Moreover, we would like to transform the solution presented in this article into a usable, practical mechanism that would meet the requirements of current, modern communication networks.

ACKNOWLEDGMENT

The authors gratefully acknowledge the financial support of the EU Horizon 2020 program towards the Internet of Radio-Light project H2020-ICT 761992.

REFERENCES

[1] J. Cosmas *et al.*, "5G Internet of radio light services for supermarkets," in *Proc. 14th China Int. Forum Solid State Lighting, Int. Forum Wide Bandgap Semiconductors China (SSLChina: IFWS)*, Nov. 2017, pp. 69–73, doi: 10.1109/IFWS.2017.8245977.

[2] J. Cosmas, B. Meunier, and K. Ali, "A scalable and license free 5G Internet of radio light architecture for services in train stations," in *Proc. 24th Eur. Wireless Conf.*, May 2018.

[3] K. Cabaj, M. Gregorczyk, W. Mazurczyk, P. Nowakowski, and P. Żórawski, "SDN-based mitigation of scanning attacks for the 5G Internet of radio light system," in *Proc. 13th Int. Conf. Availability, Rel. Secur. ARES*, 2018, pp. 1–10.

[4] K. Cabaj, M. Gregorczyk, W. Mazurczyk, P. Nowakowski, and P. Żórawski, "Network threats mitigation using software-defined networking for the 5G Internet of radio light system," *Secur. Commun. Netw.*, vol. 2019, pp. 1–22, Feb. 2019, doi: 10.1155/2019/4930908.

[5] K. Cabaj, M. Gregorczyk, W. Mazurczyk, P. Nowakowski, and P. Żórawski, "Sniffing detection within the network," in *Proc. 14th Int. Conf. Availability, Rel. Secur. ARES*, 2019, p. 108.

[6] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Comput. Surv.*, vol. 50, no. 4, pp. 1–36, Nov. 2017, doi: 10.1145/3092566.

[7] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018, doi: 10.1109/ACCESS.2018.2836950.

[8] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 506–519, doi: 10.1145/3052973.3053009.

[9] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.

[10] D. Susid, "An evaluation of network based sniffer detection; Sentinel," M.S. thesis, Goeteborg Univ., Gothenburg, Sweden, 2004.

[11] Z. Trabelsi, H. Rahmani, K. Kaouech, and M. Frikha, "Malicious sniffing systems detection platform," in *Proc. Can. Conf. Electr. Comput. Eng. Toward Caring Humane Technol. CCECE*, Jan. 2004, pp. 201–207.

[12] D. Wu and F. Wong, "Remote Sniffer Detection," in *Computer Science Division*, vol. 14. Berkeley, CA, USA: Univ. California, Oct. 1998.

[13] Z. Trabelsi and H. Rahmani, "Promiscuous mode detection platform," in *Proc. 2nd Int. Workshop Secur. Inf. Syst.*, 2004, pp. 279–292.

[14] A. Mishra, R. S. Chowhan, and A. Mathur, "Sniffer detection and load balancing using aglets in a cluster of heterogeneous distributed system environment," in *Proc. IEEE 7th Power India Int. Conf. (PIICON)*, Nov. 2016, pp. 1–6.

[15] H. AbdelallahElhad, H. M. Khelalfa, and H. M. Korteb. (2002). *An Experimental Sniffer Detector: SnifferWall*. SEcurite des Communications sur Internet–SECI. Accessed: May 2020. [Online]. Available: http://www.lsv.fr/~goubault/SECI-02/Final/actes-seci02/pdf/008-Abdelallahelhadj.pdf

[16] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.

[17] *H2O.ai Driverless AI User Guide*. Accessed: Feb. 2020. [Online]. Available: http://docs.h2o.ai/driverless-ai/1-8-lts/docs/userguide/index.html

[18] *XGBoost Documentation*. Accessed: Feb. 2020. [Online]. Available: https://xgboost.readthedocs.io/en/latest/index.html

MARCIN GREGORCZYK was born in Radom, Poland, in 1986. He received the B.S. and M.S. degrees in telecommunication from the Warsaw University of Technology, Warsaw, Poland, in 2010 and 2012, respectively, where he is currently pursuing the Ph.D. degree with the Cybersecurity Division, Institute of Telecommunications. He is a RedHat Certified Architect and Instructor of RedHat Enterprise Linux courses. His research interests include Linux operating system security and network security especially cloud related. He took part in projects for EU.

PIOTR ŻÓRAWSKI received the B.Sc. and M.Sc. degrees in computer science from the Faculty of Electronics and Information Technology, Warsaw University of Technology (WUT), in 2016 and 2019, respectively, where he is currently pursuing the Ph.D. degree. His research interests include computer and network security, dynamic malware analysis, and reverse engineering. He took part in projects for EU and U.S. Air Force.

PIOTR NOWAKOWSKI received the B.Sc. degree in electronic engineering and the M.Sc. degree in computer science from the Faculty of Electronics and Information Technology, Warsaw University of Technology (WUT), in 2016 and 2018, respectively, where he is currently pursuing the Ph.D. degree. His research interests include network security, reverse engineering, parallel processing, and computer graphics.

KRZYSZTOF CABAJ received the M.Sc., Ph.D., and D.Sc. (habilitation) degrees in computer science from the Faculty of Electronics and Information Technology, Warsaw University of Technology (WUT), in 2004, 2009, and 2019, respectively. He is currently an Associate Professor with WUT. He was a Former Instructor of Cisco certificated Academy courses: CCNA Routing and Switching, CCNA Security, and CCNP with the International Telecommunication Union Internet Training Centre (ITU-ITC). His research interests include network security, honeypots, dynamic malware analysis, data-mining techniques, the IoT, and industrial control systems security. He is the author or coauthor of over 60 publications and supervisor of more than twenty five B.Sc. and M.Sc. degrees theses in the field of information security. He took part in over a dozen research projects, among others for EU, ESA, Samsung, U.S. Army, and U.S. Air Force. He is a Co-Leader of the Computer Systems Security Group, Institute of Computer Science.

WOJCIECH MAZURCZYK (Senior Member, IEEE) received the B.Sc., M.Sc., Ph.D. (Hons.), and D.Sc. (habilitation) degrees in telecommunications from the Warsaw University of Technology (WUT), Warsaw, Poland, in 2003, 2004, 2009, and 2014, respectively. He is currently a Professor with the Institute of Computer Science, WUT, and the Head of the Computer Systems Security Group. He also works as a Researcher with the Parallelism and VLSI Group, Faculty of Mathematics and Computer Science, Fern Universitaet, Germany. His research interests include bio-inspired cybersecurity and networking, information hiding, and network security. He is involved in the technical program committee of many international conferences and also serves as a Reviewer for major international magazines and journals. Since 2016, he has been the Editor-in-Chief of the *Open Access Journal of Cyber Security and Mobility*. Since 2018, he has been serving as an Associate Editor for the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY and the Mobile Communications and Networks Series Editor for *IEEE Communications Magazine*.

7. Szacowanie poufnych parametrów tablicy przepływów w przełączniku SDN

Publikacja zawarta w tym rozdziale pt. “Inferring Flow Table State through Active Fingerprinting in SDN Environments: A Practical Approach” została opublikowana w materiałach konferencji “SECRYPT 2021 18th International Conference on Security and Cryptography” oraz zaprezentowana przez autora niniejszej rozprawy w trakcie tego wydarzenia w dniu 08.07.2021 i bardzo dobrze przyjęta przez słuchaczy. Różni się ona od publikacji prezentowanych wcześniej w rozumieniu wykorzystania technologii SDN w cyberbezpieczeństwie. Poprzednie artykuły miały na celu pokazanie, jak SDN może przysłużyć się podnoszeniu bezpieczeństwa w sieciach teleinformatycznych dla różnych typów zagrożeń. Okazuje się jednak, jak wspomniano już wcześniej, że w technologii SDN, poprzez oddzielenie płaszczyzny sterowania od danych (w czym tkwi siła SDN), pojawiły się nowe podatności niespotykane do tej pory w tradycyjnych sieciach teleinformatycznych.

Jedną z takich podatności jest atak przepełnienia tablicy przepływów w przełączniku, co może prowadzić do jego niestabilnej pracy, co z kolei może skutkować wprowadzeniem dodatkowych opóźnień, szczególnie uciążliwych w przypadku aplikacji czasu rzeczywistego typu telefonia IP czy wideokonferencje. Aby przeprowadzić taki atak w sposób skuteczny, atakujący musi wywnioskować rozmiar i zajętość tablicy. W tym celu może zostać użyta metoda tzw. fingerprinting’u. Celem tego rozwiązania jest określenie specyficznych własności i parametrów usługi, systemu, czy urządzenia, które pozwolą na jego jednoznaczną identyfikację. Szczególnym rodzajem tej techniki jest tzw. active fingerprinting, w którym następuje aktywna interakcja z badaną usługą (poprzez wysłanie odpowiednio spreparowanego ruchu). Otrzymane odpowiedzi są analizowane i porównywane z listą znanych wartości w celu znalezienia dopasowania.

Rozmiar i użycie tablicy w przełączniku powinny być uznawane jako poufne informacje. Wiedza o tych parametrach może posłużyć atakującemu do odgadnięcia typu czy modelu danego urządzenia, co być może ujawnić kolejne wektory ataków.

W prezentowanym w tym rozdziale artykule:

- Przedstawiono w dokładny sposób scenariusz ataku, zaproponowanego wcześniej w [155] oraz jakie warunki muszą być spełnione, aby zakończył się on sukcesem.
- Eksperymentalnie wykazano, że zaproponowane w [155] rozwiązania do wnioskowania o stanie tablicy przepływów wykazują znaczące ograniczenia i nie mogą być stosowane w realnych środowiskach SDN.
- Zaproponowano nowatorską technikę ataku, która opiera się na aktywnym fingerprinting’u oraz algorytmach wykrywania skoków i zmian poziomów, co czyni ją bardziej odporną i adaptacyjną od metody przedstawionej w [155].
- Eksperymentalnie zbadano proponowane podejście w praktycznym stanowisku testo-

wym z wykorzystaniem oprogramowania popularnego obecnie w stosowanych konfiguracjach SDN.

- Porównano otrzymane wyniki symulacji [155] oraz ulepszonej metody w rzeczywistych warunkach sieciowych, co pokazało, że zaprezentowane nowatorskie podejście daje dużo wyższą skuteczność wnioskowania rozmiaru i użycia tablicy SDN.
- Oceniono poprzednio zaproponowane metody ochrony przed atakiem oraz zaproponowano inne, które w ocenie autorów są skuteczniejsze oraz bardziej adekwatne dla nowatorskiego podejścia technologii SDN.

Uzyskane rezultaty badań eksperymentalnych pokazują, że zaprezentowana ulepszona wersja ataku może osiągać skuteczność ponad 99% w środowiskach zbliżonych do obecnie występujących w rzeczywistych sieciach teleinformatycznych. Zaproponowano także metody obrony adekwatne do technologii SDN, które powinny skutecznie chronić przed tym rodzajem ataku. Przedstawiona metoda nie uwzględnia jednak dodatkowego ruchu podkładowego, który mógłby symulować bardziej realne warunki panujące w obecnych sieciach. Jest to jeden z kierunków obecnie prowadzonych badań.

Podobnie jak w przypadku poprzednich publikacji przeprowadzono praktyczne eksperymenty, które dodatkowo pokazały, że wyniki symulacji nie uwzględniają rzeczywistych warunków panujących w dynamicznych i niedoskonałych sieciach.

Inferring Flow Table State through Active Fingerprinting in SDN Environments: A Practical Approach

Marcin Gregorczyk[a] and Wojciech Mazurczyk[b]
Warsaw University of Technology, Institute of Computer Science, Warsaw, Poland
{*marcin.gregorczyk.dokt, wojciech.mazurczyk*}*@pw.edu.pl*

Keywords: SDN, Software-defined Networking, Security, Flow Table, Overflow Attack, Active Fingerprinting.

Abstract: Software-Defined Networking (SDN) is currently a popular and heavily investigated concept, e.g., in cloud computing. Despite its obvious benefits, the decoupling of the control and data planes brings new security risks. One of the major threats is overflow attack, which can lead to network instability. To perform it in an efficient manner, an attacker needs to infer the flow table state, and for this purpose, typically fingerprinting techniques are utilized. In this paper, first, we prove that the previously proposed fingerprinting method exhibits major limitations. Then, building upon the existing solution, we propose an improved attack technique which is able to predict the flow table state with more than 99% prediction accuracy. Moreover, our solution has additional advantages over state-of-the-art solutions, i.e., it is adaptive and robust, thus it is suitable for real-world applications. Finally, we also discuss potential countermeasures that can be used to thwart such threats.

1 INTRODUCTION

Software-defined Networking (SDN) paradigm changes the view on networking, e.g., in the current data centers or other environments where cloud computing is heavily utilized (Kreutz et al., 2015). The main characteristic feature of SDN, i.e., the decoupling of the control and data planes, offers an opportunity of using network equipment in a more programmable way. Note that, in traditional environments, switches and routers are standalone devices, which typically make all decisions based on the static configuration. However, in SDN, a central entity, i.e., the controller decides what happens with each traffic type and steers the switch how it should handle it (Kreutz et al., 2015). This is achieved by controlling the switch flow table so it forwards the network traffic to the proper destination. If a received packet does not match any rule defined on the switch, additional communication between the switch and the controller must occur. The controller may decide, e.g., to install a new flow in the switch flow table for such traffic. However, if the flow table is full, extra messages must be exchanged between the switch and the controller to remove one of the existing flows and install the new one. Note that an attacker may exploit such a flow table management process. If he uses active fingerprinting methods, it is possible, based on the response to the artificial traffic, to deduce the *size* of the switch flow table and its current *utilization rate*. Such a technique is feasible because depending on whether a specific flow exists in the flow table or if the flow table is full or not, the system performance differs. Flow table size and its current utilization rate are internal SDN characteristics and should be considered as confidential information and thus not revealed to the public. However, if an attacker is able to infer the flow table state, it can use this information to perform a carefully crafted flow table overflow attack (Zhou et al., 2018). This may cause the instability of the overloaded systems and their unpredictability. Finally, the decreased performance can inflict additional delays, which may negatively influence real-time applications such as Voice over IP (VoIP) or videoconferencing. Moreover, overflow attacks are especially dangerous for devices with limited resources, i.e., those that can afford only limited memory resources to store the flow table. This includes, for example, Internet of Things (IoT) equipment as it is predicted that SDN will enrich such application scenarios in the near future (Li et al., 2020; Flauzac et al., 2015). In effect, for devices with a small flow table size, such an attack will be

[a] https://orcid.org/0000-0002-1108-2780
[b] https://orcid.org/0000-0002-8509-4127

Gregorczyk, M. and Mazurczyk, W.
Inferring Flow Table State through Active Fingerprinting in SDN Environments: A Practical Approach.
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 576-586
ISBN: 978-989-758-524-1 ISSN: 2184-7711

especially dangerous as it will be much easier to overwhelm the flow table with the precisely crafted traffic. In (Ahmed et al., 2020) and (Yu et al., 2020), the authors propose methods to infer the SDN internal parameters, while (Xie et al., 2021) introduces the table overflow Low Rate Denial of Service Attack (LDoS) attack countermeasure. There are also other solutions which focus on detecting and mitigating attacks on SDN and its internal parameters (Baidya and Hewett, 2019; Wu and Chen, 2020; Nallusamy et al., 2020; Nurwarsito et al., 2020). However, their major drawback is that they were created and evaluated in a simulated environment (typically Mininet[1]), and thus they may not be applicable to real-world scenarios.

Considering the above, in this paper, our main novel contributions are to:

- experimentally demonstrate that the currently existing solutions to infer the flow table state exhibit limitations and cannot be applied in practical SDN environments;
- propose a novel attack technique which relies on active fingerprinting and algorithms for peak and level-shift detection, which makes it more robust and adaptive;
- experimentally evaluate the proposed approach in the practical testbed using real-world software products currently popularly used in SDN setups.

The rest of the paper is structured as follows. Section 2 outlines the assumed attack scenario. In Section 3, the experimental testbed and methodology are depicted. Results from the experimental evaluation are presented in Section 4, while in Section 5 we discuss potential countermeasures. Finally, Section 6 concludes our work.

2 ATTACK SCENARIO

In this paper, we consider an attacker trying to infer the flow table state to perform a carefully crafted overflow attack to disrupt the SDN-based network. To determine the size and utilization rate of the flow table, an attacker uses an active fingerprinting technique. To be effective, the malicious party needs to understand how the OpenFlow-based communication is performed. This is explained in detail below.

In the SDN environment, the controller installs a set of rules on a switch. If the incoming traffic matches one of the installed flows, it will be autonomously forwarded to the proper destination. Such a process is fast, as it does not involve any additional steps. We will denote the time needed for such an operation as *T1*. However, if the received network traffic does not match any rule, it must be sent to the controller for inspection. Then, the controller may decide to install an appropriate rule for such traffic. Unfortunately, additional processing will consume more time (*T2*) than in the former case. Likewise, if the flow table is already full, the controller will have to decide which flow should be removed, send such information to the switch, and finally install a new flow. This process will consume even more time (*T3*). Note that using carefully crafted traffic and by measuring the processing time needed to handle the traffic, an attacker may be able to accurately deduce the flow table state. Fig. 1 presents an assumed attack scenario.

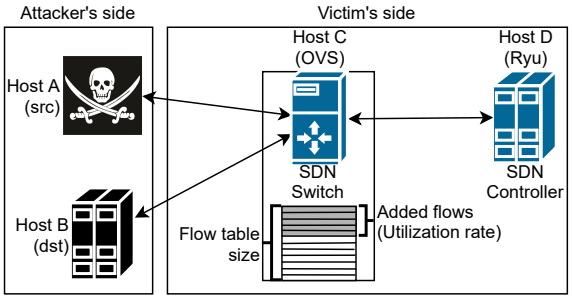


Figure 1: Attack scenario and the testbed used for the experimental evaluations.

Host A is the attacker, while Host B may be, in principle, unaware of the malicious activities. If the traffic sent from Host A to Host B is sent back to Host A, the time needed for such an operation will be recorded by the attacker as a Round Trip Time (RTT). It will be then considered as a measure of system performance. If, for any reason, the returning traffic does not reach Host A, Host B can be an active member of the attack as well. In such a case, instead of measuring RTT, the difference in time between sending a packet by Host A and receiving it by Host B can be utilized in the same manner. In this paper, we assume the former case. Sequence diagrams for exchanging messages between hosts in each system state, i.e., representing RTT as T1, T2, and T3, are illustrated in Fig. 2, 3, and 4.

The flow table size and its utilization rate should be considered as nonpublic information. By counting the transmitted packets and analyzing the obtained RTT (represented as T1, T2, and T3), the attacker can infer how many packets are needed to fill the flow table and its overall size. Since the flow table size depends on the hardware used, such activity is a form of active fingerprinting. Moreover, as already mentioned, such knowledge can be used to successfully perform an overflowing of the flow table. Such activ-

[1]http://mininet.org

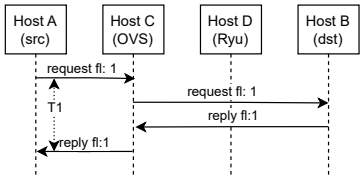


Figure 2: Sequence diagram for RTT measuring for T1 scenario (flow rule already exists in the table).

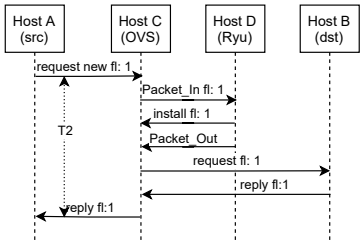


Figure 3: Sequence diagram for RTT measuring for T2 scenario (flow rule does not exist in the table; table not full).

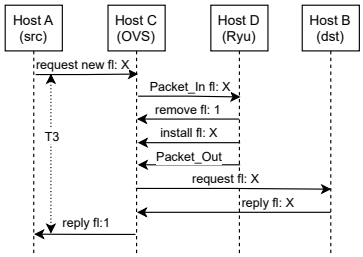


Figure 4: Sequence diagram for RTT measuring for T3 scenario (flow rule does not exist in the table; table full).

ity can significantly harm the overall system performance. Clearly, the attacker can simply flood an SDN switch indefinitely, which at some point will lead to the denial-of-service, as already studied in the literature (Kreutz et al., 2015), (Correa Chica et al., 2020). However, in the assumed threat model, the attacker aims to establish the table size and its utilization rate. This can be achieved by means of fingerprinting and provide more information on the setup, topology, and utilized software or hardware, which could be used as attack vectors on the infrastructure.

2.1 Previously Proposed Solution

In (Zhou et al., 2018) the authors implemented and evaluated the inference attack framework according to the attack scenario depicted above. To evaluate their technique, they used *Mininet*, i.e., simulated environment, as a network prototyping system to emulate hosts and the switch. They also utilized *libnet* (https://github.com/libnet/libnet) for generating artificial traffic, which raises concerns whether the RTT or only one-way delay was measured, as the fully spoofed traffic which was utilized might not be sent back to the source host. It must be also noted that the authors sent each spoofed packet only once. By generating such traffic and measuring the time needed to complete the rule installation process, and comparing it to the fixed thresholds (T1, T2, and T3), the authors tried to deduce the flow table state. Unfortunately, by using a simulation instead of real-world setup, they did not take into account several issues that need to be addressed.

First of all, typically, a switch sends packets to the controller using the Packet_In OpenFlow message when instructed to do so or does not know what to do with the packet. Then, the controller can decide to install a new rule allowing, blocking, or altering such traffic. However, it may also choose to send a packet back to the switch (using Packet_Out OpenFlow message) to be forwarded to the proper destination. Without this, the packet would be dropped, and thus every first packet for every new flow would be dropped as well. Removing and installing a new rule can be done parallel to sending a packet back to the switch. Therefore, only one packet could not be enough to distinguish between T2 and T3 RTTs (which allow to decide if the flow table is full or not). In other words, the system's response to a full and empty flow table for the first packet of a new flow would be very similar. It must be noted that in (Zhou et al., 2018), the authors used only one packet per flow, which, as we present in this paper, in practical evaluation, gives the worst results.

The second issue of the existing method is related to the fact that the same probe packet (which does not have a corresponding rule on the switch) is sent very frequently (so-called flooding). In such a case, after some time, the switch may become overloaded, which makes it hard to interpret the obtained RTTs. Thus, it must be noted that apart from the number of probes, also the interval between them should be considered as well. As mentioned in (Zhou et al., 2018), the authors used *libnet*, which as they claim, can generate tens of thousand packets per second. However, in our approach, we send packets at a fixed interval.

Finally, the last issue is related to anomalies that may typically occur in the network. Packets can be lost or processed longer than usual. Unfortunately, re-transmission is not always an option. For example, if the switch flow table is not full and a new packet is sent, in (Zhou et al., 2018) the authors assumed that RTT would be near T2. However, it can be much longer (even longer than the expected T3), due to anomalies). This means that if the same packet is sent again, the corresponding flow may be already installed, and the obtained RTT will be near T1. As a result, such system responses would be hard to un-

ambiguously interpret. Anomalies typically happen randomly and thus cannot be foreseen in advance and eliminated.

All above-mentioned issues lead to the conclusion that the values (T1, T2, and T3) measured using simulation may be of limited use when faced with real-world and diversified environments.

2.2 Proposed Active Fingerprinting Technique

To address the above-mentioned issues of the existing method, we propose a novel inferring technique that can be utilized in real-world networking scenarios and builds upon the method described in the previous subsection.

First of all, for measuring RTT, we propose to utilize the *ping* tool. In more detail, we define two parameters useful for the fingerprinting technique, i.e., the number of ping probes (in the remainder of the paper, we call it *probes* in short) and ping interval (in short *interval*). The former describes the number of ICMP Echo Request messages used for fingerprinting purposes, while the latter defines the gap between consecutive ICMP messages. The resulting calculated RTT is the arithmetic average of the values obtained for all probes during one execution of the ping tool. We decided to utilize the ping tool, therefore ICMP protocol, as an example of the most common way of measuring RTT to prove the effectiveness of the presented method (other types of traffic may be used as well). Note that in enterprise environments, ICMP may be often blocked, however, as described in 2, two hosts can take part in the attack. In such a scenario, the difference in time between sending a packet by Host A and receiving it by Host B can be utilized in the same manner as ping tool-based measurements.

Note that *probes* and *interval* constitute the fingerprinting observation window – the time the attacker requires to determine the flow table state. When this time is too short or too long, the attacker may not be able to correctly establish the characteristics mentioned above, which renders the attack less effective and a more tedious task.

To reduce the influence of anomalies caused by the networking environment, we deduce the system state based on its responses and not using fixed thresholds (as it was done in (Zhou et al., 2018)). In result, our approach is much more robust and able to automatically adapt to different types of networking equipment and environments. In more detail, we perform three steps to identify the expected changes in the system's response instead of comparing the absolute values of the measured RTT: *(1)* Measuring RTT for two types of traffic, i.e., control and noise (described in subsection 3.1); *(2)* Identifying peaks in the control traffic and analyzing their periodicity to estimate the flow table size (presented in subsection 3.2); *(3)* Finding level-shift in the noise traffic to estimate the flow table utilization rate (outlined in subsection 3.2).

3 EXPERIMENTAL TESTBED AND METHODOLOGY

The testbed that we utilized during the experimental evaluation is presented in Fig. 1. It includes four hosts: *Host A* is running the Linux ping6 command, generating ICMPv6 traffic (src), used to measure RTT between Host A and Host B; *Host B* is the destination of the ping6 command (dst); *Host C* is running Open vSwitch (v. 2.12.0), controlled by the SDN controller on Host D; *Host D* has Ryu (v. 4.34) SDN controller installed.

The above mentioned hosts are KVM virtual machines created on CentOS 8.3 server (kernel 4.18.0-240). To avoid the interference of external factors, only these virtual machines were running at the time of the experiments and they were connected to an isolated virtual network. Additionally, the CPU governor on the physical host was set to disable overclocking, which could lead to misleading results depending on CPU frequency. Each host has one vCPU and 4GB RAM, whereas the physical host is equipped with 12 physical cores (24 threads) and 144GB RAM. Due to these facts, we can assume that the impact of the virtualization overhead was minimized. Virtual machines were running CentOS 8.3 operating system as well.

To measure RTT, we decided to utilize the most common tool, i.e., the ping command. We want each new sent packet (probe) to cause a new flow to be installed in the flow table. However, as we introduce probes as a sequence of the same ping packets, we need to distinguish between the two sequences, so they can fall into the proper flow table rule. Therefore, to solve this issue, we decided to utilize the IPv6 flow label header field (Deering et al., 2017) to mark the consecutive ping messages. Specific flow labels can be assigned using the ping6 command (option -F) and thus easily identified. The same mechanism can also be used in the OpenFlow protocol (version 1.5.1), so it is possible to identify the traffic on the SDN switch, too.

Open vSwitch (OVS) running on Host C is set to connect to the Ryu controller running on Host D. Ryu controller during startup, firstly delete all existing rules on the OVS, then installs two initial rules:

a) redirect all IPv6 ICMP Echo Requests (type: 128 (Conta et al., 2006)) with a unique flow label to the controller; b) handle the remaining traffic with NORMAL action. With such configuration, the switch will send every IPv6 ICMP Echo Request packet with a unique flow label field for further inspection. Every other traffic, not related to the experiments like, for instance, ARP or even IPv6 ICMP Echo Reply (type: 129 (Conta et al., 2006)), will be forwarded to the proper destination and will not reach the controller. This allows to ensure that the controller is not additionally overloaded and the obtained results are accurate.

3.1 RTT Measurements

Figs. 2, 3, and 4 illustrate a sequence of packets/messages exchanged between hosts in the experimental testbed and the related measured RTTs (T1-T3). It should be noted that, in general, we can assume that T1<T2<T3; however, as described in Section 2.1, this may not always be the case.

The first part of our experiments is related to T1, T2, and T3 RTTs approximation. All experiments were measured separately to exclude their interference. For each RTT measurement scenario, we transmitted 1000 ICMP messages to provide statistical relevance. We assume that if the switch is not overloaded, T1, T2, and T3 will be almost constant (excluding anomalies). On the other hand, if the switch is unstable, it might be impossible to establish a correct range of each RTT.

To find the optimal values of T1, T2, and T3 for our needs and exclude the issues described in Section 2.1, we use parameters: *number of ping probes* (provided by ping command argument $-c$) and *ping interval* ($-i$). To establish the optimal ping probes value, we sent n ICMP messages with the same IPv6 flow label, $n \subset \{1,2,3,4,5\}$. We decided to use five as the maximum value of consecutive ICMP messages as we experimentally established that a new flow in the flow table is usually installed after sending 2-4 pings. We also decided to use three interval values: 0.001, 0.005, and 0.01s. In our setup, we determined that the intervals below 0.001s cause the switch and the controller to be flooded, which causes the measurements to be unpredictable in terms of delays as small as less than 1ms. As for the maximum value (0.01s), we empirically measured that the average T3 is about 5ms, therefore we doubled it.

To calculate T1, we sent one IPv6 ICMP message with a fixed flow label. Next, we confirmed that the flow is installed on the switch, and then we started generating 1000 IPv6 ICMP messages with the same flow label. As the rule for such traffic was initiated before, and this rule is installed on the switch, each ping is not sent to the SDN controller, but forwarded directly to the destination host (see Fig. 2). Each experiment was conducted for every combination of probes (1-5) and intervals (0.001, 0.005, and 0.01s).

To measure T2, a similar experiment was performed. The only difference is that a new flow label was used for every 1000 IPv6 ICMP Echo requests. Additionally, the flow table size was not limited, and all of the 1000 new flows were installed without any issues. Some extra messages were exchanged between OVS and SDN controller to install each flow (see Fig. 3). Again, each combination of probes and intervals was experimentally evaluated.

Finally, T3 was measured very similar to T2, but the flow table size was limited and filled before the experiment started. In such a case, an additional effort is required from the SDN controller to remove one of the existing rules to install the new one (see Fig. 4).

For statistical relevance purposes, each experiment was repeated ten times. We also calculated metrics such as minimum, maximum, and average values, and standard deviation. Therefore, 450 experiments were run in total (3 RTT x 5 probes x 3 intervals x 10 experiments).

3.2 Flow Table Size and Utilization Rate Measurements

As described in Section 2, the fingerprinting attack is successful if an attacker is able to infer the flow table size and its utilization rate. To perform the experimental evaluation of the flow table state inferring, we made the following assumptions:

- we manually limit the flow table size to ten different values (100, 200, ..., 1000);
- we manually filled half of the table with unique IPv6 flow labels not used further during the experiments (corresponding values: 50, 100, ..., 500).

To measure the flow table size and its utilization rate, we need to run two ping6 programs in one loop. The purpose of the first one was to generate a new flow label in each loop. We call it the *noise ping*. We use it to completely fill the flow table, which, in the result, will cause the change of the measured RTT. The second ping with the fixed flow label, i.e., *control ping*, is used to measure when a specific flow label is removed because of the introduced noise. Because the flow table in our environment uses the most popular algorithm, i.e., FIFO (First In First Out), the generated noise will fill the queue causing the control ping flow to be removed. However, as the control

ping is constantly being executed, after being pushed off from the FIFO queue, it will cause a short peak in the RTT measure (T2). After the rule installation and before noise pushes it again from the queue, the measured RTT should be around T1.

Moreover, during this part of our research, each experiment was executed ten times to obtain average values, and we used all combinations of the number of probes (1-5), intervals (0.001, 0.005, and 0.01s), and flow table size (100, 200, ..., 1000). We also assumed to generate ten times more noise flows than the actual flow table size. In total, this resulted in 1500 experiments (10 flow table sizes/utilization rate x 5 probes x 3 intervals x 10 experiments).

The obtained RTT measurements for the control and noise ping traffic are then utilized during the inferring procedure's remaining steps. To infer the flow table state based on RTT, we use three algorithms. Firstly, we need to find RTT peaks in the control traffic, which indicate the system state change (purging control ping rule) and for this purpose we used "Robust peak detection algorithm (using z-scores)"[2], claimed to be the best choice by the community. Z-score is a measure of how many standard deviations below or above the population mean a raw score is. The second algorithm is "Signal find peaks" from the well-known software collection SciPy[3]. We noticed that both algorithms are able to identify peaks with enough accuracy for our needs. After short tuning of the parameters, we were satisfied with the results and decided to use the first algorithm.

The next step was to find the periodicity in the control pings (which we use to infer the flow table size). We calculate the distance between each pair of peaks in the control pings and determine the most frequent one. Assuming that anomalies occur randomly, finding periodicity among them should not be possible. On the other hand, after sending a number of noise pings equal to the flow table size, all other flows will be removed. In such a case, the control ping peak should be visible, as a new flow will have to be installed (T1->T3). Note that we do not use any background traffic in our experiments, which would change the overall system behavior.

After successfully determining the control peak periodicity, an assumption can be made that the discovered value is the flow table size. As the flow table utilization rate cannot be larger than the flow table size, finding a level-shift in the noise traffic (i.e., the flow table utilization rate) can use a narrowed value range. For this purpose, we limit the analyzed data from ping number 0 to the ping number denoting the inferred flow table size. Next, we use the NumPy Convolve[4] algorithm, which returns the discrete linear convolution of two one-dimensional sequences.

To establish the best parameters for the proposed active fingerprinting technique, i.e., the number of probes and the interval value, we use the mean absolute percentage error (MAPE):

$MAPE = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$,

where A_t is the actual value, and F_t is the predicted value. Consequently, the prediction accuracy (PA) is defined as

$PA = max(1 - MAPE, 0)$.

We calculate MAPE of the inferring of table flow size/utilization rate for each combination of the number of ping probes and intervals. Additionally, we use the average values from 10 repetitions of the experiments. However, using mean values can be misleading due to error compensation. We noticed that the overall effect of the table size inferring was satisfying in many cases, but then the standard deviation of the obtained results proved that it should not be completely trusted. Thus, it must be emphasized that for the best configuration of our method, the standard deviation was on a very low level.

4 OBTAINED RESULTS

This section presents the obtained results for RTT measurements and the flow table size and utilization rate inferring process. First, we show that the measured RTTs cannot be compared to static thresholds (as it is proposed in (Zhou et al., 2018)) as this is not applicable to real-world setups. Then, we present the experimental results for our approach.

4.1 RTT Measurements

As mentioned, if we infer the flow table state just based on the measured RTT values, this can lead to incorrect predictions. In (Zhou et al., 2018), the authors claimed that RTT for the traffic for which flow entry exists in the flow table (T1) is in the range of 0.2-0.3ms. When the flow entry for a packet does not exist and the flow table is not full, RTT (T2) is between 3-5ms. Finally, the traffic for which the flow entry does not exist and the flow table is full, RTT (T3) is in the range 6-8ms. It should be noted that

[2]https://stackoverflow.com/questions/22583391/peak-recognition-in-realtime-timeseries-data/22640362

[3]https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html

[4]https://numpy.org/doc/stable/reference/generated/numpy.convolve.html

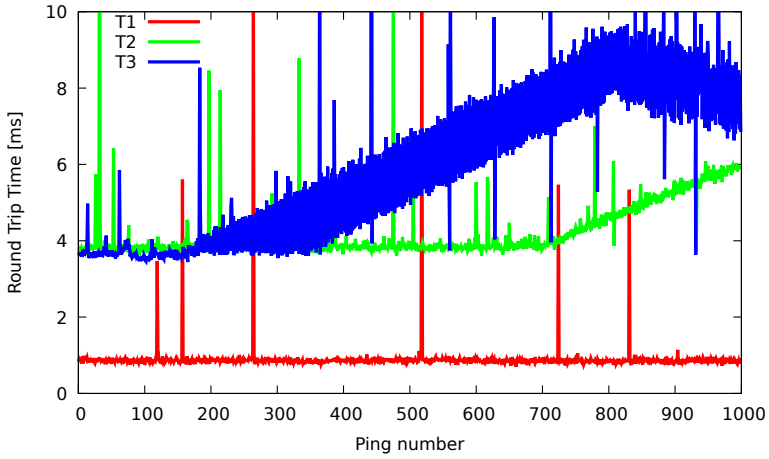


Figure 5: Anomalies and switch saturation (probes=1, interval=0.001s).

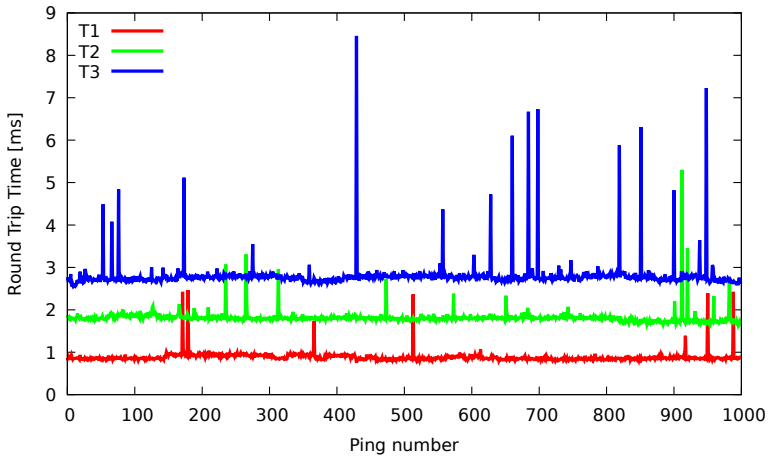


Figure 6: Stable switch operation (probes=3, interval=0.01s).

in real-world setups, such fixed thresholds are not realistic as the T1-T3 depend greatly on the used software and hardware. In the result, applying the provided ranges as general thresholds may not work correctly for every practical setup. Note that the authors also claim that T1, T2, and T3 values are contained in small ranges that do not overlap with each other.

However, through practical experiments, we determined that this is not always the case. All types of issues described in Section 2.1 are illustrated in Fig. 5 – note: the figure has been scaled down to 10ms for better visibility and the actual peak values can reach up to 70ms. Moreover, as in (Zhou et al., 2018), we used the number of probes equal to one; however, it must be emphasized that the authors did not state how often they sent their traffic but only that they use *libnet* to generate tens of thousand packets per second (thus we decided to use $interval = 0.001s$). Based on this figure, the following conclusions can be reached. First of all, the general rule assumed by the authors of (Zhou et al., 2018), i.e., that T1<T2<T3 is not always correct. In Fig. 5, it is visible that there are T1 peaks higher than T2 and T3, and T2 peaks higher than T3. Moreover, for pings 0-150, T2 is almost equal to T3, making it impossible to decide what is the current flow table state. Finally, after ping number 200 for T3 and 700 for T2, the switch has problems handling traffic in a timely manner and works unpredictably.

Note that if we consider also other factors while observing the traffic, i.e., various number of probes and intervals, the obtained results are less noisy, thus more suitable for our purpose. Fig. 6 presents the comparison between T1, T2, and T3 for $ping\ probes = 3$ and $interval = 0.01s$. It is visible that the peaks in each signal are still noticeable, but are not that so frequent as for the single probe case. Additionally, in this case, T1, T2, and T3 ranges are generally not overlapping each other. Thus, we decided to investigate further T1-T3 RTT results for the various number of probes and intervals.

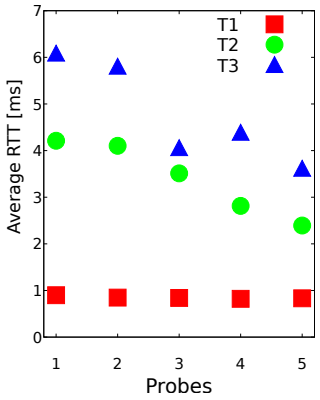


Figure 7: Comparison of the average RTT - interval=0.001s.

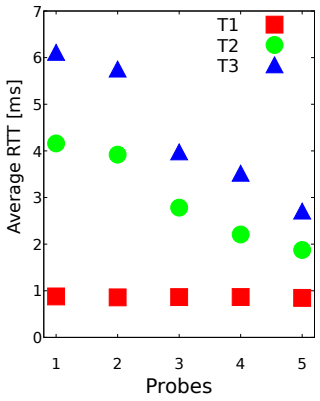


Figure 8: Comparison of the average RTT - interval=0.005s.

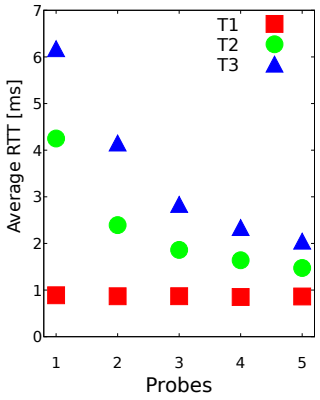


Figure 9: Comparison of the average RTT - interval=0.01s.

The obtained results are illustrated twofold in Figs. 7, 8, 9 and 10, 11, 12. Figs. 7, 8, and 9 compare the *average* RTTs for T1, T2, and T3 depending on the number of ping probes and intervals. On average, they all seem similar. Moreover, the general rule T1<T2<T3 is still valid. However, if T1 max value and T3 min value are taken into account (as presented in Figs. 10, 11, and 12), the opposite effect occurs (T1>T2>T3). As described, the reason for such a situation can be ping anomalies, lost packets, or switch saturation. This proves that making decisions about the flow table state solely based on the RTT measurement is incorrect. That is why, in the next subsection, we propose a more robust and adaptive approach.

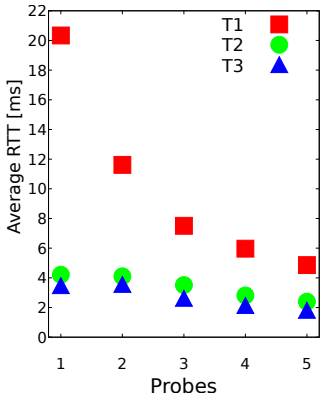


Figure 10: Comparison of the RTT: maximal T1, average T2, and minimal T3 - interval=0.001s.

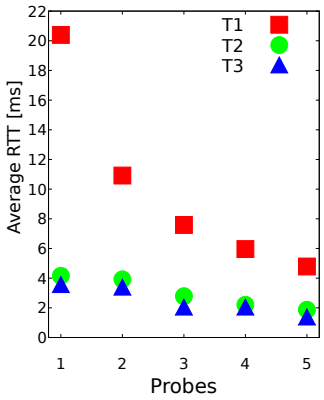


Figure 11: Comparison of the RTT: maximal T1, average T2, and minimal T3 - interval=0.005s.

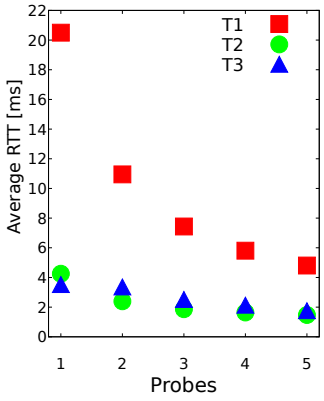


Figure 12: Comparison of the RTT: maximal T1, average T2, and minimal T3 - interval=0.01s.

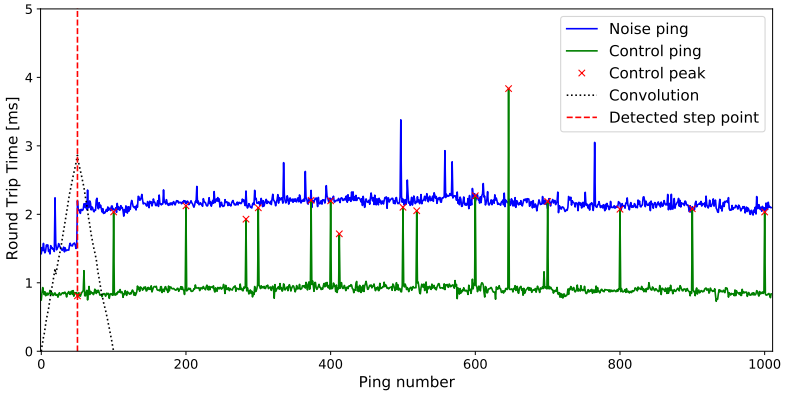


Figure 13: Finding peaks, their periodicity, and level-shift detection (*flow table size=100*, *flow table utilization rate=50*, *probes=3*, *interval=0.01s*).

4.2 Flow Table Size and Utilization Rate: Proposed Approach

As described in Section 3.1, we initially filled the flow table to 50% of its capacity for each experiment. Next, we used noise and control pings to decide the state of the flow table. The obtained results are presented below.

First, Fig. 13 presents an exemplary measurement with parameters: *flow table size=100*, *flow table utilization rate=50*, *ping probes=3*, and *ping interval=0.01s*. It should be noted that after 50 noise pings, there is a visible level-shift between 1.5ms and 2.2ms. At this point, the flow table was completely filled. Moreover, every ca. 100 control pings, there is a visible control ping peak (100, 200, ..., 1000). From these results, we can infer that the flow table size is 100 and the flow table is filled with 50 flows. Unfortunately, note that anomalies for control and noise pings are visible in Fig. 13 as well. Thus, the main issue is to determine the periodicity of the control ping based on which the flow table size can be inferred. If we narrow down the values from the ping number 0 to the predicted flow table size, we can determine a level-shift in the control noise, which, in turn, provides the flow table utilization rate (by subtracting the discovered table size and the detected level-shift). For this purpose, we use the algorithms described in Section 3.2 (finding peaks, their periodicity, and level-shift detection). Fig. 13 presents the effect of all algorithms with parameters: *probes=3*, *interval=0.01s*. The actual values of the flow table size and utilization rate for this experiment were 100 and 50, whereas the proposed approach inferred these values as 99 and 50, respectively.

Table 1 presents the overall results for the inferring errors of the table size and its utilization rate and the mean absolute percentage error for the proposed fingerprinting technique. We decided to exclude standard deviation results for the better visibility and focus on the average values.

Based on the presented outcome, it is visible that the worst results are obtained for *probes=1* – this again proves that using only one probe packet (as done in (Zhou et al., 2018)) is disadvantageous. Surprisingly, the highest value used in the experiments, i.e., 5 probes, gave worse results than for 2-4. Moreover, the results for *interval=0.001s* are worse than for *0.005* or *0.01s*. The latter intervals behave similarly. However, as we aim to infer the flow table size and utilization rate as fast as possible, the lower number of probes and intervals are more favorable. Note that by multiplying the number of ping probes and the interval value, we are able to roughly estimate the time needed to perform a single ping operation. Thus, it is visible that a trade-off between the time needed to infer the table flow state and the more accurate results must be made.

After determining the optimal values for both parameters for our active fingerprinting method, we present the final evaluation in Fig. 14. It presents ten flow table sizes (100, 200, ..., 1000). The red bar represents the actual flow table size, green – the inferred values using the proposed method, and blue – values as predicted by (Zhou et al., 2018). Note that the latter method is not using additional parameters, i.e., number of probes or interval which are essential for the approach proposed in this paper. Thus, it was not exactly possible to setup the same configuration for both methods. That is why, thresholds used in (Zhou et al., 2018) were evaluated against the configuration of the our method, which is the closest to the original approach, i.e., probes=1 and interval=0.001s. As it is visible, the proposed technique significantly outperforms the previously proposed approach and gives estimates very close to the actual values, whereas the

Table 1: Inferring error for all experiments (bold denotes the best result).

No. of probes	Interval [s]	Table size inferring relative error [%]	Table utilization rate inferring relative error [%]	MAPE [%]	PA [%]
1	0.001	67.35	67.59	67.47	32.53
2	0.001	16.30	36.23	26.27	73.73
3	0.001	6.30	7.26	6.78	93.22
4	0.001	8.00	2.33	5.16	94.84
5	0.001	10.05	3.90	6.97	93.03
1	0.005	52.88	60.27	56.57	43.43
2	0.005	3.40	0.02	1.71	98.29
3	0.005	15.07	3.81	9.44	90.56
4	0.005	13.38	3.13	8.26	91.74
5	0.005	26.32	5.04	15.68	84.32
1	0.01	59.68	61.27	60.48	39.52
2	0.01	3.00	0.00	1.50	98.50
3	**0.01**	**1.00**	**0.00**	**0.50**	**99.50**
4	0.01	17.59	0.43	9.01	90.99
5	0.01	32.94	3.06	18.00	82.00

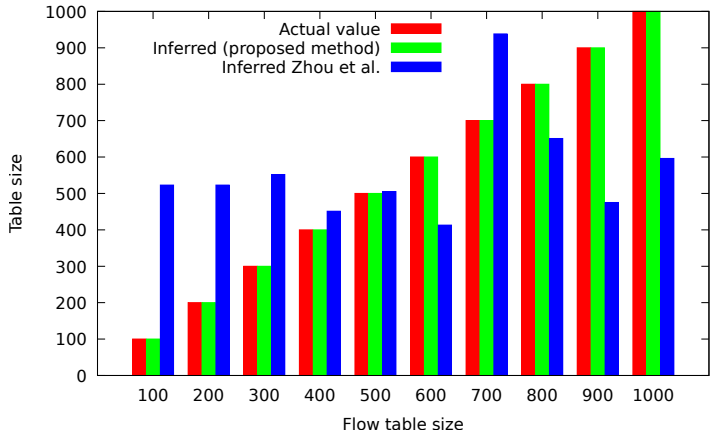


Figure 14: Result of inferring the flow table size for the best parameters (*probes=3*, *interval=0.01s*) compared with (Zhou et al., 2018) (*probes=1*, *interval=0.001s*).

method proposed in (Zhou et al., 2018) is not applicable to real-world scenarios and even then results in more serious underestimates and even overestimates. However, our approach was always able to infer it correctly.

5 COUNTERMEASURES

In the previous sections, we demonstrated that active fingerprinting could be effectively used to infer the flow table state in real-world environments is feasible. Therefore, it is vital to discuss also potential defensive solutions.

In (Zhou et al., 2018), the authors proposed two defensive solutions:

- *Routing Aggregation*, which relies on using fewer flow table entries – similar entries should be aggregated in groups;
- *Multilevel Flow Table Architecture* to implement additional memory which will extend the flow table possible size.

However, in our opinion, both solutions are inadequate to the demands of real-world applications. Routing aggregation is not implemented in OpenFlow switches, and it is unlikely to be added in the near future. It is also a completely different approach than the one used in modern switches. On the other hand, the Multilevel Flow Table Architecture solution is not always applicable in IoT devices where memory extension is not an option.

Recently (Nallusamy et al., 2020) published a paper about preventing flow table entries overflow using decision tree-based algorithm. It operates by classifying the entries, and by replacing the usual eviction process by pushing the low important entries into counting bloom filter which acts as a cache to prevent flow entry miss. However, again experiments were conducted using a simulated environment (Mininet),

thus it may not be applicable to real-world scenarios.

Considering the above, more robust mechanisms are desired, especially of a proactive nature. Below, we discuss potential suitable solutions, however, due to space limitation, we leave the evaluation of these countermeasures as our future work.

The first possibility is to utilize approaches that rely on the Moving Target Defense paradigm (Cho et al., 2020). Such techniques aim at continuous modification of the configuration of the defended system, shifting the attack surface and making the attacker's cyber reconnaissance methods ineffective. For countering the flow table state inferring solution presented in this paper, one of the techniques that rely on applying periodical dynamic changes to the network topology can be used (Sengupta et al., 2020). Such an approach can disorient the attacker by providing incorrect input data and increase uncertainty in the fingerprinting activities he performs.

An alternative solution is to use Cyber Deception (Wang and Lu, 2018), which aims to confuse the attacker by intentionally feeding him with incorrect information. In our case, it would be a deliberate modification of the system response, i.e., increasing or decreasing of the RTT.

Both above-mentioned mechanisms are promising defense methods against the attack described in this paper and their deployment does not need as significant modification to the underlying protocols/systems as the previously proposed countermeasures.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a robust and adaptive fingerprinting method which can be used to infer SDN switch flow table size and utilization rate. Obtaining such information allows a malicious party to perform an effective overflow attack causing disruption in the network. By running the experiments in the setup using real-world software products, we proved that the previously proposed technique, which was evaluated only via simulations, cannot be utilized in practice. On the other hand, the approach we propose scales well and it is compatible with different types of SDN software and hardware. Results obtained via experimental evaluation revealed that the resulting prediction accuracy to determine the flow table state is more than 99%. Finally, we also proposed some realistic defense mechanisms, however, we left the investigation of their effectiveness as our future work. In the future, we would like also to introduce an additional background traffic factor and reevaluate our proposed method, which will transform the solution presented in this paper into a usable and more practical mechanism, which can be successfully implemented in the industry.

REFERENCES

Ahmed, B. et al. (2020). Fingerprinting sdn policy parameters: An empirical study. *IEEE Access*, 8:142379–142392.

Baidya, S. and Hewett, R. (2019). Sdn-based edge computing security: Detecting and mitigating flow rule attacks. In *Symp. on Edge Computing*, page 364–370.

Cho, J. et al. (2020). Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys Tutorials*, 22(1):709–745.

Conta, A. et al. (2006). Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443, RFC Editor.

Correa Chica, J., Imbachi, J., and Botero Vega, J. (2020). Security in sdn: A comprehensive survey. *Journal of Network and Computer Applications*, 159:102595.

Deering, S. et al. (2017). Internet Protocol, Version 6 (IPv6) Specification. RFC 8200, RFC Editor.

Flauzac, O. et al. (2015). Sdn based architecture for iot and improvement of the security. In *IEEE AINA Wkshp.*, pages 688–693.

Kreutz, D. et al. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Li, Y. et al. (2020). Enhancing the internet of things with knowledge-driven software-defined networking technology: Future perspectives. *Sensors*, 20(12).

Nallusamy, P. et al. (2020). Decision tree-based entries reduction scheme using multi-match attributes to prevent flow table overflow in sdn environment. *IJNM*.

Nurwarsito, H. et al. (2020). Implementation of wlru algorithm to improve scalability in software defined network. In *SIET*, page 165–170.

Sengupta, S. et al. (2020). A survey of moving target defenses for network security. *IEEE Communications Surveys Tutorials*, 22(3):1909–1941.

Wang, C. and Lu, Z. (2018). Cyber deception: Overview and the road ahead. *IEEE Security Privacy*, 16:80–85.

Wu, Q. and Chen, H. (2020). Achieving a heterogeneous software-defined networks with camovisor. In *Int. Conf. on Electronics Technology*, pages 804–808.

Xie, S. et al. (2021). A table overflow ldos attack defending mechanism in software-defined networks. *SCN*.

Yu, M. et al. (2020). Flow table security in sdn: Adversarial reconnaissance and intelligent attacks. In *IEEE INFOCOM*, pages 1519–1528.

Zhou, Y. et al. (2018). Exploiting the vulnerability of flow table overflow in software-defined network. *SCN*, 2018:1–15.

8. Podsumowanie

Rozwój i wszechobecność Internetu zmieniły świat pod wieloma względami, a jednym z nich jest konieczność zachowania szczególnej czujności w zakresie bezpieczeństwa informacji i cyberzagrożeń. Ponieważ działania agresorów stają się coraz bardziej wyrafinowane i stale pojawiają się nowe zagrożenia, realizacja celów związanych z bezpieczeństwem informacji wymaga wykorzystania najnowszych technologii i narzędzi. Potrzeba zapewniania bezpieczeństwa w sieciach teleinformatycznych jest bezdyskusyjna. Dodatkowo zarządzanie i konfiguracja sieci to bardzo złożone, wymagające i czasochłonne zadania. Dlatego też, w niniejszej rozprawie przedstawiono technologię SDN, która może okazać się odpowiednim orężem w walce z cyberzagrożeniami spotykanymi w sieciach teleinformatycznych.

Jak wykazano, technologia SDN upraszcza i poprawia zarządzanie siecią poprzez zapewnienie wysoce elastycznych sieci opartych na zasadzie rozdzielenia płaszczyzn sterowania i infrastruktury. Dzięki programowalnemu podejściu technologia SDN umożliwia implementację nowych algorytmów i aplikacji bez konieczności zmiany samych urządzeń sieciowych. Z tych powodów stanowi ona idealną platformę do rozwoju i testowania nowych rozwiązań bezpieczeństwa.

W rozprawie wyjaśniono działanie technologii SDN, z jakich elementów się składa oraz jakie cechy posiada. Opisano także zaadresowane zagrożenia i ich przynależność do MITRE ATT&CK Framework. Przestawiono również stan wiedzy odnośnie technologii SDN i związanych z nią aspektów bezpieczeństwa.

Wszystkie cele badawcze założone w niniejszej rozprawie doktorskiej, które określono w podrozdziale 1.1 zostały osiągnięte. Przedstawiono ich realizację poprzez badania eksperymentalne i szczegółowe omówienie uzyskanych rezultatów w artykułach naukowych składających się na niniejszą rozprawę. Badania te pozwoliły potwierdzić sformułowaną na początku pracy tezę, iż “Możliwe jest stworzenie efektywnych metod zabezpieczeń z wykorzystaniem technologii Software-Defined Networking w celu skutecznego wykrywania i zapobiegania atakom sieciowym”.

W ramach przedstawionych publikacji oraz przeprowadzonych badań, zaprojektowano, zaimplementowano oraz przeprowadzono analizę systemów wykrywania oraz mitygacji popularnych zagrożeń w sieciach teleinformatycznych pokrywających swoim zasięgiem cały stos TCP/IP, co czyni te badania kompleksowymi w zakresie sieci teleinformatycznych. Przedstawiono także możliwy atak na architekturę SDN, jako przykład innego podejścia do tematu bezpieczeństwa tej technologii oraz potencjalne zabezpieczenia przed tym zagrożeniem. Znany z literatury podział tematu bezpieczeństwa technologii SDN na zwiększanie bezpieczeństwa sieci dzięki specjalistycznym aplikacjom oraz bezpieczeństwo architektury SDN także znalazł odzwierciedlenie w przedstawionych publikacjach.

Przeprowadzone badania są nowatorskie oraz wykorzystują realne technologie używane w dzisiejszych centrach danych. Były one też efektem praktycznych implementacji, co od-

różnia je od badań opartych na symulacjach dominujących obecnie w literaturze. Jak udowodniono w ostatniej z publikacji w ramach przedstawionego cyklu (rozdział 7.) symulacje nie uwzględniają wszystkich czynników i anomalii, które mogą występować w rzeczywistych sieciach, co może prowadzić do niewłaściwych wniosków co do skuteczności i wydajności proponowanych mechanizmów zabezpieczeń. Realizacja tych eksperymentów w fizycznym środowisku badawczym zwiększa wiarygodność przeprowadzonych badań oraz wyciągniętych wniosków. Znalazło to także odzwierciedlenie w pozytywnych recenzjach publikacji, które ukazały się w renomowanych czasopismach naukowych oraz na konferencji dotyczących bezpieczeństwa IT, co dodatkowo świadczy o istotności przedstawionych badań i uzyskanych wyników. Uzyskane rezultaty dla każdej z zaproponowanych metod to:

- 97% skuteczności wykrywania ransomware CryptoWall oraz Locky.
- 99% poprawnych detekcji skanowania TCP SYN.
- Wykryto wszystkie próby przeprowadzenia ataku DHCP Starvation, wprowadzając pomijalny koszt w postaci dodatkowych opóźnień.
- 99% poprawnych detekcji podsłuchu sieciowego z wykorzystaniem sniffera.
- 99% skuteczności szacowania dwóch poufnych parametrów tablicy przepływów w przełączniku SDN.

Zaprezentowane rozwiązania, mają oczywiście swoje ograniczenia (wymienione w rozdziałach 4., 5., 6. oraz 7.), jednak uzyskane wyniki oraz praktyczna implementacja eksperymentów, pozwalają stwierdzić, że wybrane kierunki badań są właściwe.

Dwa z czterech przedstawionych artykułów prezentują wyniki badań przeprowadzonych w ramach zakończonego z sukcesem europejskiego projektu IoRL finansowanego przez Komisję Europejską i prowadzonego na Politechnice Warszawskiej w ramach programu Horizon 2020.

Każda z przedstawionych w niniejszej rozprawie prac, zawiera dalsze, potencjalne kierunki badań, które mogą rozszerzać zaproponowane funkcjonalności, adresować ciągle zmieniające się zagrożenia oraz uwzględniać dodatkowe czynniki, odzwierciedlające warunki panujące w obecnie działających sieciach teleinformatycznych.

Kolejnym kierunkiem badań może być także zaadresowanie innych zagrożeń za pomocą technologii SDN, które pokryją w pełni wszystkie taktyki i techniki wchodzące w skład MITRE ATT&CK Framework. Efektem takich analiz byłoby stwierdzenie, czy jest możliwym zaadresowanie wszystkich grup znanych obecnie cyberzagrożeń za pomocą technologii SDN. W przyszłości można by także zbadać przydatność technologii SDN do wykrywania i przeciwdziałania nieznanym jeszcze zagrożeniom.

Ważnym aspektem byłoby również porównanie technologii SDN do tradycyjnych systemów zabezpieczeń (jak IDS/IPS czy firewall) i uzyskanie odpowiedzi na pytanie, do przeciwdziałania jakim zagrożeniom technologia SDN, ze względu na swoją specyfikę, nadaje się najlepiej, a do których jednak nie jest to trafiony wybór. Możliwym kierunkiem rozwoju jest

także stworzenie hybrydowego systemu obronnego, który łączyłby i wykorzystywał zalety nowatorskiej technologii SDN oraz sprawdzonych i dobrze znanych rozwiązań IDS/IPS.

Należy także przeprowadzić dogłębne badania bezpieczeństwa architektury samej technologii SDN, ze względu na nowe elementy infrastruktury sieciowej i zależności pomiędzy nimi, niespotykane w obecnych, tradycyjnych sieciach teleinformatycznych. Badania takie pozwolą wskazać kierunek rozwoju technologii SDN w celu zapewniania bezpiecznego i niezawodnego działania sieci w centrach danych. Brak odpowiednich fundamentów związanych z bezpieczeństwem, jak w przypadku każdej technologii, będzie powodował nawarstwianie się problemów, które mogą w rezultacie spowodować, że sieci SDN mogą zostać uznane za rozwiązanie nie do końca bezpieczne.

Ponieważ obecne systemy i sieci teleinformatyczne są znacznie większe i bardziej skomplikowane, niż te spotykane jeszcze kilkanaście lat temu, należy zatem wziąć pod uwagę aspekty związane ze skalowalnością, wydajnością oraz wszelkiego rodzaju kosztami wprowadzanymi przez proponowane rozwiązania. Skalowalność jako jedna z podstawowych cech środowisk chmurowych, ma bardzo duże znaczenie przy wyborze systemów bezpieczeństwa, które nie mogą ograniczać funkcji biznesowych realizowanych przez infrastrukturę poprzez wprowadzanie wąskiego gardła. Należy zatem przeprowadzić badania dotyczące tego czy SDN charakteryzuje się odpowiednią wydajnością, która jest w stanie sprostać coraz bardziej wyrafinowanym czy rozproszonym atakom.

Technologia SDN jest już nierozerwalną częścią środowisk chmurowych, które zmieniają nasz świat. Wykorzystanie jej potencjału w celu zapewniania bezpieczeństwa wydaje się być naturalnym krokiem. Przedstawione wyniki oraz wnioski, zaprezentowane w publikacjach wchodzących w skład niniejszej rozprawy, oraz przyszłe kierunki badań mogą pozwolić na stworzenie kompleksowego i praktycznego rozwiązania bezpieczeństwa, przeznaczonego dla obecnych i przyszłych sieci teleinformatycznych. Każde zagrożenie (znane obecnie oraz przyszłe) zaadresowane za pomocą technologii SDN będzie potwierdzać słuszność założonej w niniejszej rozprawie tezy.

Bibliografia

[1] R. Guerzoni, R. Trivisonno i D. Soldani, „SDN-based architecture and procedures for 5G networks”, w *1st International Conference on 5G for Ubiquitous Connectivity*, 2014, s. 209–214. DOI: 10.4108/icst.5gu.2014.258052.

[2] S. Li, L. D. Xu i S. Zhao, „5G Internet of Things: A survey”, *Journal of Industrial Information Integration*, t. 10, s. 1–9, 2018, ISSN: 2452-414X. DOI: https://doi.org/10.1016/j.jii.2018.01.005. adr.: https://www.sciencedirect.com/science/article/pii/S2452414X18300037.

[3] 5G-PPP, *5G PPP Phase 1 Security Landscape*, Strona odwiedzona: 09.12.2021. adr.: https://5g-ppp.eu/white-papers.

[4] F. Z. Yousaf, M. Bredel, S. Schaller i F. Schneider, „NFV and SDN—Key Technology Enablers for 5G Networks”, *IEEE Journal on Selected Areas in Communications*, t. 35, nr. 11, s. 2468–2478, 2017. DOI: 10.1109/JSAC.2017.2760418.

[5] S. Azodolmolky, P. Wieder i R. Yahyapour, „SDN-based cloud computing networking”, w *2013 15th International Conference on Transparent Optical Networks (ICTON)*, 2013, s. 1–4. DOI: 10.1109/ICTON.2013.6602678.

[6] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky i S. Uhlig, „Software-Defined Networking: A Comprehensive Survey”, *Proceedings of the IEEE*, t. 103, nr. 1, s. 14–76, 2015. DOI: 10.1109/JPROC.2014.2371999.

[7] Embroker Team, *2021 Must-Know Cyber Attack Statistics and Trends*, Strona odwiedzona: 09.12.2021. adr.: https://www.embroker.com/blog/cyber-attack-statistics.

[8] M. Monshizadeh, V. Khatri i R. Kantola, „Detection as a service: An SDN application”, w *2017 19th International Conference on Advanced Communication Technology (ICACT)*, 2017, s. 285–290. DOI: 10.23919/ICACT.2017.7890099.

[9] K. Nam i K. Kim, „A Study on SDN security enhancement using open source IDS/IPS Suricata”, w *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, s. 1124–1126. DOI: 10.1109/ICTC.2018.8539455.

[10] I. Ahmad, S. Namal, M. Ylianttila i A. Gurtov, „Security in Software Defined Networks: A Survey”, *IEEE Communications Surveys Tutorials*, t. 17, nr. 4, s. 2317–2346, 2015. DOI: 10.1109/COMST.2015.2474118.

[11] J. Chica, J. Cuatindioy i J. Botero, „Security in SDN: A comprehensive survey”, *Journal of Network and Computer Applications*, t. 159, s. 102 595, mar. 2020. DOI: 10.1016/j.jnca.2020.102595.

[12] O. Yurekten i M. Demirci, „SDN-based cyber defense: A survey”, *Future Generation Computer Systems*, t. 115, s. 126–149, 2021, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2020.09.006. adr.: https://www.sciencedirect.com/science/article/pii/S0167739X20303277.

[13] Ellucian, *2021 Must-Know Cyber Attack Statistics and Trends*, Strona odwiedzona: 09.12.2021. adr.: ellucian.com/blog/how-do-you-win-cybersecurity-arms-race.

[14] J. Qadir, N. Ahmed, F. Z. Yousaf i A. Taqweem, „Network as a Service: The New Vista of Opportunities”, *CoRR*, t. abs/1606.03060, 2016. arXiv: 1606.03060. adr.: http://arxiv.org/abs/1606.03060.

[15] Y. Cui, L. Yan, S. Li, H. Xing, W. Pan, J. Zhu i X. Zheng, „SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks”, *Journal of Network and Computer Applications*, t. 68, s. 65–79, 2016, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.04.005. adr.: https://www.sciencedirect.com/science/article/pii/S1084804516300480.

[16] J. English, *A quick guide to important SDN security issues*, Strona odwiedzona: 09.12.2021. adr.: https://www.techtarget.com/searchnetworking/feature/A-quick-guide-to-important-SDN-security-issues.

[17] B. Sokappadu, A. Hardin, A. Mungur i S. Armoogum, „Software Defined Networks: Issues and Challenges”, w *2019 Conference on Next Generation Computing Applications (NextComp)*, 2019, s. 1–5. DOI: 10.1109/NEXTCOMP.2019.8883558.

[18] S. Scott-Hayward, G. O'Callaghan i S. Sezer, „Sdn Security: A Survey”, w *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, s. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553.

[19] N. A. Aziz, T. Mantoro, M. A. Khairudin i A. F. b. A. Murshid, „Software Defined Networking (SDN) and its Security Issues”, w *2018 International Conference on Computing, Engineering, and Design (ICCED)*, 2018, s. 40–45. DOI: 10.1109/ICCED.2018.00018.

[20] K. Cabaj, M. Gregorczyk i W. Mazurczyk, „Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics”, *Computers & Electrical Engineering*, t. 66, s. 353–368, 2018, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2017.10.012. adr.: https://www.sciencedirect.com/science/article/pii/S0045790617333542.

[21] K. Cabaj, M. Gregorczyk, W. Mazurczyk, P. Nowakowski i P. Żórawski, „Network Threats Mitigation Using Software-Defined Networking for the 5G Internet of Radio Light System”, *Security and Communication Networks*, t. 2019, s. 22, 2019. DOI: 10.1155/2019/4930908.

[22] M. Gregorczyk, P. Żórawski, P. Nowakowski, K. Cabaj i W. Mazurczyk, „Sniffing Detection Based on Network Traffic Probing and Machine Learning”, *IEEE Access*, t. 8, s. 149 255–149 269, 2020. DOI: 10.1109/ACCESS.2020.3016076.

[23] M. Gregorczyk. i W. Mazurczyk., „Inferring Flow Table State through Active Fingerprinting in SDN Environments: A Practical Approach”, w *Proceedings of the 18th International Conference on Security and Cryptography - SECRYPT*, INSTICC, SciTePress, 2021, s. 576–586, ISBN: 978-989-758-524-1. DOI: 10.5220/0010573905760586.

[24] Open Networking Foundation, *Software-Defined Networking (SDN) Definition*, Strona odwiedzona: 25.07.2021. adr.: https://opennetworking.org/sdn-definition.

[25] Open Networking Foundation, *Software-Defined Networking: The New Norm for Networks*, Strona odwiedzona: 14.12.2021. adr.: https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks.

[26] K. Benton, L. J. Camp i C. Small, „OpenFlow Vulnerability Assessment”, ser. HotSDN '13, New York, NY, USA: Association for Computing Machinery, 2013, 151–152, ISBN: 9781450321785. DOI: 10.1145/2491185.2491222. adr.: https://doi.org/10.1145/2491185.2491222.

[27] M. Kuźniar, P. Perešíni i D. Kostić, „What You Need to Know About SDN Flow Tables”, w *Passive and Active Measurement*, J. Mirkovic i Y. Liu, red., Cham: Springer International Publishing, 2015, s. 347–359, ISBN: 978-3-319-15509-8.

[28] J. Ganczarenko, *The difference between SDN and NFV-a simple guide*, Strona odwiedzona: 25.07.2021. adr.: https://codilime.com/blog/the-difference-between-sdn-and-nfv-a-simple-guide.

[29] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin i Z. Zhang, „Enabling security functions with SDN: A feasibility study”, *Computer Networks*, t. 85, s. 19–35, 2015, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2015.05.005. adr.: www.sciencedirect.com/science/article/pii/S1389128615001619.

[30] Department of Defense, *Department of Defence: Strategy for Operating in Cyberspace*, Strona odwiedzona: 20.12.2021. adr.: https://csrc.nist.gov/CSRC/media/Projects/ISPAB/documents/DOD-Strategy-for-Operating-in-Cyberspace.pdf.

[31] B. A. S. Al-rimy, M. A. Maarof i S. Z. M. Shaid, „Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions”, *Computers & Security*, t. 74, s. 144–166, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.01.001. adr.: https://www.sciencedirect.com/science/article/pii/S016740481830004X.

[32] C. Everett, „Ransomware: to pay or not to pay?”, *Computer Fraud & Security*, t. 2016, nr. 4, s. 8–12, 2016, ISSN: 1361-3723. DOI: https://doi.org/10.1016/S1361-3723(16)30036-7. adr.: https://www.sciencedirect.com/science/article/pii/S1361372316300367.

[33] Datto, *Datto's Global State of the Channel Ransomware Report*, Strona odwiedzona: 22.12.2021. adr.: https://www.datto.com/resource-downloads/Datto2019_StateOfTheChannel_RansomwareReport.pdf.

[34] Sophos, *The State of Ransomware 2021*, Strona odwiedzona: 22.12.2021. adr.: https://www.sophos.com/en-us/medialibrary/pdfs/whitepaper/sophos-state-of-ransomware-2021-wp.pdf.

[35] B. Chang, *One of the biggest US insurance companies reportedly paid hackers $40 million ransom after a cyberattack*, Strona odwiedzona: 22.12.2021. adr.: https://www.businessinsider.com/cna-financial-hackers-40-million-ransom-cyberattack-2021-5.

[36] N. S. Institute, *The Growing Ransomware Wave*, Strona odwiedzona: 22.12.2021. adr.: https://www.nsi.org/2021/02/15/employee-cyber-security-awareness-ransomware-wave.

[37] Coveware, *Ransomware Payments Fall as Fewer Companies Pay Data Exfiltration Extortion Demands*, Strona odwiedzona: 22.12.2021. adr.: https://www.coveware.com/blog/ransomware-marketplace-report-q4-2020.

[38] W. Mazurczyk i L. Caviglione, „Cyber Reconnaissance Techniques", *Commun. ACM*, t. 64, nr. 3, 86–95, 2021, ISSN: 0001-0782. DOI: 10.1145/3418293. adr.: https://doi.org/10.1145/3418293.

[39] Mitre, *Common Attack Pattern Enumerations and Classifications*, Strona odwiedzona: 27.12.2021. adr.: https://capec.mitre.org/data/definitions/287.html.

[40] J. Postel, „Transmission Control Protocol", RFC Editor, STD 7, 1981. adr.: http://www.rfc-editor.org/rfc/rfc793.txt.

[41] K. Cabaj, M. Gregorczyk, W. Mazurczyk, P. Nowakowski i P. Żórawski, „SDN-Based Mitigation of Scanning Attacks for the 5G Internet of Radio Light System", w *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018, New York, NY, USA: Association for Computing Machinery, 2018, ISBN: 9781450364485. DOI: 10.1145/3230833.3233248. adr.: https://doi.org/10.1145/3230833.3233248.

[42] E. Bou-Harb, M. Debbabi i C. Assi, „Cyber Scanning: A Comprehensive Survey", *IEEE Communications Surveys Tutorials*, t. 16, nr. 3, s. 1496–1519, 2014. DOI: 10.1109/SURV.2013.102913.00020.

[43] A. Blaise, M. Bouet, S. Secci i V. Conan, „Split-and-Merge: Detecting Unknown Botnets", w *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, s. 153–161.

[44] R. Droms, „Dynamic Host Configuration Protocol", RFC Editor, RFC 2131, 1997. adr.: http://www.rfc-editor.org/rfc/rfc2131.txt.

[45] H. Mukhtar, K. Salah i Y. Iraqi, „Mitigation of DHCP starvation attack", *Computers & Electrical Engineering*, t. 38, nr. 5, s. 1115–1128, 2012, Special issue on Recent Advances in Security and Privacy in Distributed Communications and Image processing, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2012.06.005. adr.: https://www.sciencedirect.com/science/article/pii/S0045790612001140.

[46] D. Warburton, *DDoS Attack Trends for 2020*, Strona odwiedzona: 22.12.2021. adr.: https://www.f5.com/labs/articles/threat-intelligence/ddos-attack-trends-for-2020.

[47] K. Cabaj, M. Gregorczyk, W. Mazurczyk, P. Nowakowski i P. Żórawski, „Sniffing Detection within the Network: Revisiting Existing and Proposing Novel Approaches", w *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ser. ARES '19, New York, NY, USA: Association for Computing Machinery, 2019, ISBN: 9781450371643. DOI: 10.1145/3339252.3341494. adr.: https://doi.org/10.1145/3339252.3341494.

[48] K. Opsahl, *Why Metadata Matters*, Strona odwiedzona: 27.12.2021. adr.: https://www.eff.org/pl/deeplinks/2013/06/why-metadata-matters.

[49] Y. Vanaubel, J.-J. Pansiot, P. Mérindol i B. Donnet, „Network Fingerprinting: TTL -Based Router Signatures", ser. IMC '13, Barcelona, Spain: Association for Computing Ma-

chinery, 2013, 369–376, ISBN: 9781450319539. DOI: 10.1145/2504730.2504761. adr.: https://doi.org/10.1145/2504730.2504761.

[50] M. Coughlin, „A Survey of SDN Security Research", Colorado, CO, USA: University of Colorado Boulder, 2014.

[51] S. Bian, P. Zhang i Z. Yan, „A Survey on Software-Defined Networking Security", w *2nd International Workshop on 5G Security IW5GS*, ACM, grud. 2016. DOI: 10.4108/eai.18-6-2016.2264176.

[52] K. Benzekki, A. El Fergougui i A. El Belrhiti El Alaoui, „Software-defined networking (SDN): A survey", *Security and Communication Networks*, t. 9, lut. 2017. DOI: 10.1002/sec.1737.

[53] Q. Waseem, S. S. Alshamrani, K. Nisar, W. I. S. Wan Din i A. S. Alghamdi, „Future Technology: Software-Defined Network (SDN) Forensic", *Symmetry*, t. 13, nr. 5, 2021, ISSN: 2073-8994. DOI: 10.3390/sym13050767. adr.: https://www.mdpi.com/2073-8994/13/5/767.

[54] R. Deb i S. Roy, „A comprehensive survey of vulnerability and information security in SDN", *Computer Networks*, t. 206, s. 108 802, 2022, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2022.108802. adr.: https://www.sciencedirect.com/science/article/pii/S1389128622000299.

[55] H. Jamjoom, D. Williams i U. Sharma, „Don't Call Them Middleboxes, Call Them Middlepipes", w *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14, New York, NY, USA: Association for Computing Machinery, 2014, 19–24, ISBN: 9781450329897. DOI: 10.1145/2620728.2620760. adr.: https://doi.org/10.1145/2620728.2620760.

[56] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, N. McKeown i S. Shenker, „SANE: a protection architecture for enterprise networks", w *15th USENIX Security Symposium*, lip. 2006.

[57] G. Stabler, A. Rosen, S. Goasguen i K.-C. Wang, „Elastic IP and Security Groups Implementation Using OpenFlow", ser. VTDC '12, New York, NY, USA: Association for Computing Machinery, 2012, 53–60, ISBN: 9781450313445. DOI: 10.1145/2287056.2287069. adr.: https://doi.org/10.1145/2287056.2287069.

[58] K. Wang, Y. Qi, B. Yang, Y. Xue i J. Li, „LiveSec: Towards Effective Security Management in Large-Scale Production Networks", w *2012 32nd International Conference on Distributed Computing Systems Workshops*, 2012, s. 451–460. DOI: 10.1109/ICDCSW.2012.87.

[59] G. Yao, J. Bi i P. Xiao, „Source address validation solution with OpenFlow/NOX architecture", w *2011 19th IEEE International Conference on Network Protocols*, 2011, s. 7–12. DOI: 10.1109/ICNP.2011.6089085.

[60] J. H. Jafarian, E. Al-Shaer i Q. Duan, „Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking", ser. HotSDN '12, New York, NY, USA: Association for Computing Machinery, 2012, 127–132, ISBN: 9781450314770. DOI: 10.1145/2342441.2342467. adr.: https://doi.org/10.1145/2342441.2342467.

[61] S. Shin i G. Gu, „CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)”, w *2012 20th IEEE International Conference on Network Protocols (ICNP)*, 2012, s. 1–6. DOI: 10.1109/ICNP.2012.6459946.

[62] R. Braga, E. Mota i A. Passito, „Lightweight DDoS flooding attack detection using NOX/OpenFlow”, w *IEEE Local Computer Network Conference*, 2010, s. 408–415. DOI: 10.1109/LCN.2010.5735752.

[63] K. Giotis, G. Androulidakis i V. Maglaris, „Leveraging SDN for Efficient Anomaly Detection and Mitigation on Legacy Networks”, w *2014 Third European Workshop on Software Defined Networks*, 2014, s. 85–90. DOI: 10.1109/EWSDN.2014.24.

[64] S. A. Mehdi, J. Khalid i S. A. Khayam, „Revisiting Traffic Anomaly Detection Using Software Defined Networking”, w *Recent Advances in Intrusion Detection*, R. Sommer, D. Balzarotti i G. Maier, red., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 161–180, ISBN: 978-3-642-23644-0.

[65] J. Matias, J. Garay, A. Mendiola, N. Toledo i E. Jacob, „FlowNAC: Flow-based Network Access Control”, w *2014 Third European Workshop on Software Defined Networks*, 2014, s. 79–84. DOI: 10.1109/EWSDN.2014.39.

[66] A. Sapio, M. Baldi, Y. Liao, G. Ranjan, F. Risso, A. Tongaonkar, R. Torres i A. Nucci, „MAPPER: A Mobile Application Personal Policy Enforcement Router for Enterprise Networks”, w *2014 Third European Workshop on Software Defined Networks*, 2014, s. 131–132. DOI: 10.1109/EWSDN.2014.9.

[67] N. Sahri i K. Okamura, „Collaborative Spoofing Detection and Mitigation – SDN Based Looping Authentication for DNS Services”, w *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, t. 2, 2016, s. 565–570. DOI: 10.1109/COMPSAC.2016.6.

[68] M. Z. Masoud, Y. Jaradat i I. Jannoud, „On preventing ARP poisoning attack utilizing Software Defined Network (SDN) paradigm”, w *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, 2015, s. 1–5. DOI: 10.1109/AEECT.2015.7360549.

[69] J. H. Cox, R. J. Clark i H. L. Owen, „Leveraging SDN for ARP security”, w *SoutheastCon 2016*, 2016, s. 1–8. DOI: 10.1109/SECON.2016.7506644.

[70] T. Alharbi, D. Durando, F. Pakzad i M. Portmann, „Securing ARP in Software Defined Networks”, w *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, 2016, s. 523–526. DOI: 10.1109/LCN.2016.83.

[71] B. Liu, J. Bi i Y. Zhou, „Source Address Validation in Software Defined Networks”, ser. SIGCOMM ’16, New York, NY, USA: Association for Computing Machinery, 2016, 595–596, ISBN: 9781450341936. DOI: 10.1145/2934872.2960425. adr.: https://doi.org/10.1145/2934872.2960425.

[72] J. Kwon, D. Seo, M. Kwon, H. Lee, A. Perrig i H. Kim, „An incrementally deployable anti-spoofing mechanism for software-defined networks”, *Computer Communications*, t. 64, s. 1–20, 2015, ISSN: 0140-3664. DOI: https://doi.org/10.1016/

j.comcom.2015.03.003. adr.: https://www.sciencedirect.com/science/article/pii/S0140366415001267.

[73] F. Kuliesius i V. Dangovas, „SDN enhanced campus network authentication and access control system", w *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2016, s. 894–899. DOI: 10.1109/ICUFN.2016.7536925.

[74] D. M. Ferrazani Mattos i O. C. M. B. Duarte, „AuthFlow: authentication and access control mechanism for software defined networking", *Annals of Telecommunications*, t. 71, nr. 11, s. 607–615, grud. 2016, ISSN: 1958-9395. DOI: 10.1007/s12243-016-0505-z. adr.: https://doi.org/10.1007/s12243-016-0505-z.

[75] M. Masoud, Y. Jaradat i A. Q. Ahmad, „On tackling social engineering web phishing attacks utilizing software defined networks (SDN) approach", w *2016 2nd International Conference on Open Source Software Computing (OSSCOM)*, 2016, s. 1–6. DOI: 10.1109/OSSCOM.2016.7863679.

[76] T. Chin, K. Xiong i C. Hu, „Phishlimiter: A Phishing Detection and Mitigation Approach Using Software-Defined Networking", *IEEE Access*, t. 6, s. 42 516–42 531, 2018. DOI: 10.1109/ACCESS.2018.2837889.

[77] K. Cabaj i W. Mazurczyk, „Using Software-Defined Networking for Ransomware Mitigation: The Case of CryptoWall", *IEEE Network*, t. 30, nr. 6, s. 14–20, 2016. DOI: 10.1109/MNET.2016.1600110NM.

[78] G. Cusack, O. Michel i E. Keller, „Machine Learning-Based Detection of Ransomware Using SDN", ser. SDN-NFV Sec'18, New York, NY, USA: Association for Computing Machinery, 2018, 1–6, ISBN: 9781450356350. DOI: 10.1145/3180465.3180467. adr.: https://doi.org/10.1145/3180465.3180467.

[79] J. M. Ceron, C. B. Margi i L. Z. Granville, „MARS: An SDN-based malware analysis solution", w *2016 IEEE Symposium on Computers and Communication (ISCC)*, 2016, s. 525–530. DOI: 10.1109/ISCC.2016.7543792.

[80] Y. Hu, K. Zheng, X. Wang i Y. Y. and, „WORM-HUNTER: A Worm Guard System using Software-defined Networking", *KSII Transactions on Internet and Information Systems*, t. 11, nr. 1, s. 484–510, sty. 2017. DOI: 10.3837/tiis.2017.01.026.

[81] M. Akbanov, V. G. Vassilakis i M. D. Logothetis, „Ransomware detection and mitigation using software-defined networking: The case of WannaCry", *Computers & Electrical Engineering*, t. 76, s. 111–121, 2019, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2019.03.012. adr.: https://www.sciencedirect.com/science/article/pii/S0045790618323164.

[82] E. Rouka, C. Birkinshaw i V. G. Vassilakis, „SDN-based Malware Detection and Mitigation: The Case of ExPetr Ransomware", w *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, 2020, s. 150–155. DOI: 10.1109/ICIoT48696.2020.9089514.

[83] F. M. Alotaibi i V. G. Vassilakis, „SDN-Based Detection of Self-Propagating Ransomware: The Case of BadRabbit", *IEEE Access*, t. 9, s. 28 039–28 058, 2021. DOI: 10.1109/ACCESS.2021.3058897.

[84] S. Khan i A. Akhunzada, „A hybrid DL-driven intelligent SDN-enabled malware detection framework for Internet of Medical Things (IoMT)", *Computer Communications*, t. 170, s. 209–216, 2021, ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2021.01.013. adr.: https://www.sciencedirect.com/science/article/pii/S0140366421000347.

[85] S. Achleitner, T. F. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy i R. Chadha, „Deceiving Network Reconnaissance Using SDN-Based Virtual Topologies", *IEEE Transactions on Network and Service Management*, t. 14, nr. 4, s. 1098–1112, 2017. DOI: 10.1109/TNSM.2017.2724239.

[86] C.-Y. J. Chiang, Y. M. Gottlieb, S. J. Sugrim, R. Chadha, C. Serban, A. Poylisher, L. M. Marvel i J. Santos, „ACyDS: An adaptive cyber deception system", w *MILCOM 2016 - 2016 IEEE Military Communications Conference*, 2016, s. 800–805. DOI: 10.1109/MILCOM.2016.7795427.

[87] S. Robertson, S. Alexander, J. Micallef, J. Pucci, J. Tanis i A. Macera, „CINDAM: Customized Information Networks for Deception and Attack Mitigation", w *2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, 2015, s. 114–119. DOI: 10.1109/SASOW.2015.23.

[88] J. H. H. Jafarian, E. Al-Shaer i Q. Duan, „Spatio-Temporal Address Mutation for Proactive Cyber Agility against Sophisticated Attackers", w *Proceedings of the First ACM Workshop on Moving Target Defense*, ser. MTD '14, New York, NY, USA: Association for Computing Machinery, 2014, 69–78, ISBN: 9781450331500. DOI: 10.1145/2663474.2663483. adr.: https://doi.org/10.1145/2663474.2663483.

[89] D. Ma, C. Lei, L. Wang, H. Zhang, Z. Xu i M. Li, „A Self-adaptive Hopping Approach of Moving Target Defense to thwart Scanning Attacks", w *Information and Communications Security*, K.-Y. Lam, C.-H. Chi i S. Qing, red., Cham: Springer International Publishing, 2016, s. 39–53.

[90] D. C. MacFarland i C. A. Shue, „The SDN Shuffle: Creating a Moving-Target Defense Using Host-Based Software-Defined Networking", w *Proceedings of the Second ACM Workshop on Moving Target Defense*, ser. MTD '15, New York, NY, USA: Association for Computing Machinery, 2015, 37–41, ISBN: 9781450338233. DOI: 10.1145/2808475.2808485. adr.: https://doi.org/10.1145/2808475.2808485.

[91] K. Wang, X. Chen i Y. Zhu, „Random domain name and address mutation (RDAM) for thwarting reconnaissance attacks", *PLOS ONE*, t. 12, nr. 5, Y. Shang, red., e0177111, maj 2017. DOI: 10.1371/journal.pone.0177111. adr.: https://doi.org/10.1371/journal.pone.0177111.

[92] Z. Zhao, F. Liu i D. Gong, „An SDN-Based Fingerprint Hopping Method to Prevent Fingerprinting Attacks", *Security and Communication Networks*, t. 2017, s. 1–12, 2017. DOI: 10.1155/2017/1560594. adr.: https://doi.org/10.1155/2017/1560594.

[93] P. Kampanakis, H. Perros i T. Beyene, „SDN-based solutions for Moving Target Defense network protection", w *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014, s. 1–6. DOI: 10.1109/WoWMoM.2014.6918979.

[94] C. V. Neu, C. G. Tatsch, R. C. Lunardi, R. A. Michelin, A. M. S. Orozco i A. F. Zorzo, „Lightweight IPS for port scan in OpenFlow SDN networks", w *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, s. 1–6. DOI: 10.1109/NOMS.2018.8406313.

[95] S. Chiba, L. Guillen, S. Izumi, T. Abe i T. Suganuma, „Design of a Network Scan Defense Method by Combining an SDN-based MTD and IPS", w *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2021, s. 273–278. DOI: 10.23919/APNOMS52696.2021.9562686.

[96] J. H. Jafarian, E. Al-Shaer i Q. Duan, „Formal Approach for Route Agility against Persistent Attackers", w *Computer Security – ESORICS 2013*, J. Crampton, S. Jajodia i K. Mayes, red., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 237–254, ISBN: 978-3-642-40203-6.

[97] F. Gillani, E. Al-Shaer, S. Lo, Q. Duan, M. Ammar i E. Zegura, „Agile virtualized infrastructure to proactively defend against cyber attacks", w *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, s. 729–737. DOI: 10.1109/INFOCOM.2015.7218442.

[98] S. Fichera, L. Galluccio, S. C. Grancagnolo, G. Morabito i S. Palazzo, „OPERETTA: An OPEnflow-based REmedy to mitigate TCP SYNFLOOD Attacks against web servers", *Computer Networks*, t. 92, s. 89–100, 2015, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2015.08.038. adr.: https://www.sciencedirect.com/science/article/pii/S1389128615002984.

[99] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras i V. Maglaris, „Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments", *Computer Networks*, t. 62, s. 122–136, 2014, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.bjp.2013.10.014. adr.: https://www.sciencedirect.com/science/article/pii/S1389128613004003.

[100] A. Hussein, I. H. Elhajj, A. Chehab i A. Kayssi, „SDN Security Plane: An Architecture for Resilient Security Services", w *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, 2016, s. 54–59. DOI: 10.1109/IC2EW.2016.15.

[101] O. Joldzic, Z. Djuric i P. Vuletic, „A transparent and scalable anomaly-based DoS detection method", *Computer Networks*, t. 104, s. 27–42, 2016, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2016.05.004. adr.: https://www.sciencedirect.com/science/article/pii/S1389128616301347.

[102] Y. E. Oktian, S. Lee i H. Lee, „Mitigating Denial of Service (DoS) attacks in OpenFlow networks", w *2014 International Conference on Information and Communication Technology Convergence (ICTC)*, 2014, s. 325–330. DOI: 10.1109/ICTC.2014.6983147.

[103] A. F. M. Piedrahita, S. Rueda, D. M. F. Mattos i O. C. M. B. Duarte, „Flowfence: a denial of service defense system for software defined networking", w *2015 Global Information Infrastructure and Networking Symposium (GIIS)*, 2015, s. 1–6. DOI: 10.1109/GIIS.2015.7347185.

[104] T. Wang, H. Chen, G. Cheng i Y. Lu, „SDNManager: A Safeguard Architecture for SDN DoS Attacks Based on Bandwidth Prediction", *Security and Communication Networks*,

t. 2018, s. 7 545 079, sty. 2018, ISSN: 1939-0114. DOI: 10.1155/2018/7545079. adr.: https://doi.org/10.1155/2018/7545079.

[105] B. Wang, Y. Zheng, W. Lou i Y. T. Hou, „DDoS attack protection in the era of cloud computing and Software-Defined Networking", *Computer Networks*, t. 81, s. 308–319, 2015, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2015.02.026. adr.: https://www.sciencedirect.com/science/article/pii/S1389128615000742.

[106] S. Wang, S. Chandrasekharan, K. Gomez, S. Kandeepan, A. Al-Hourani, M. R. Asghar, G. Russello i P. Zanna, „SECOD: SDN sEcure control and data plane algorithm for detecting and defending against DoS attacks", w *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, s. 1–5. DOI: 10.1109/NOMS.2018.8406196.

[107] N. Goksel i M. Demirci, „DoS Attack Detection using Packet Statistics in SDN", w *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, 2019, s. 1–6. DOI: 10.1109/ISNCC.2019.8909114.

[108] R. Xie, M. Xu, J. Cao i Q. Li, „SoftGuard: Defend Against the Low-Rate TCP Attack in SDN", w *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, s. 1–6. DOI: 10.1109/ICC.2019.8761806.

[109] L. Wang, Q. Li, Y. Jiang, X. Jia i J. Wu, „Woodpecker: Detecting and mitigating link flooding attacks via SDN", *Computer Networks*, t. 147, s. 1–13, 2018, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.09.021. adr.: www.sciencedirect.com/science/article/pii/S1389128618309721.

[110] J. Wang, R. Wen, J. Li, F. Yan, B. Zhao i F. Yu, „Detecting and Mitigating Target Link Flooding Attacks Using SDN", *IEEE Transactions on Dependable and Secure Computing*, t. 16, nr. 6, s. 944–956, 2019. DOI: 10.1109/TDSC.2018.2822275.

[111] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau i J. Wu, „Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis", *IEEE Transactions on Information Forensics and Security*, t. 13, nr. 7, s. 1838–1853, 2018. DOI: 10.1109/TIFS.2018.2805600.

[112] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li i L. Gong, „Detection and defense of DDoS attack-based on deep learning in OpenFlow-based SDN", *International Journal of Communication Systems*, t. 31, nr. 5, e3497, sty. 2018. DOI: 10.1002/dac.3497. adr.: https://doi.org/10.1002/dac.3497.

[113] J. Cui, J. He, Y. Xu i H. Zhong, „TDDAD: Time-Based Detection and Defense Scheme Against DDoS Attack on SDN Controller", w *Information Security and Privacy*, W. Susilo i G. Yang, red., Cham: Springer International Publishing, 2018, s. 649–665.

[114] Z. Abou El Houda, A. S. Hafid i L. Khoukhi, „Cochain-SC: An Intra- and Inter-Domain DDos Mitigation Scheme Based on Blockchain Using SDN and Smart Contract", *IEEE Access*, t. 7, s. 98 893–98 907, 2019. DOI: 10.1109/ACCESS.2019.2930715.

[115] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati i B. Stiller, „A Blockchain - Based Architecture for Collaborative DDoS Mitigation with Smart Contracts", w *Security of Networks and Services in an All-Connected World*, D. Tuncer, R. Koch, R.

Badonnel i B. Stiller, red., Cham: Springer International Publishing, 2017, s. 16–29, ISBN: 978-3-319-60774-0.

[116] J. Xing, W. Wu i A. Chen, „Architecting Programmable Data Plane Defenses into the Network with FastFlex", ser. HotNets '19, New York, NY, USA: Association for Computing Machinery, 2019, 161–169, ISBN: 9781450370202. DOI: 10.1145/3365609.3365860. adr.: https://doi.org/10.1145/3365609.3365860.

[117] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, q. li, M. Xu i J. Wu, „Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches", sty. 2020. DOI: 10.14722/ndss.2020.24007.

[118] Z. Zhao, D. Gong, B. Lu, F. Liu i C. Zhang, „SDN-Based Double Hopping Communication against Sniffer Attack", *Mathematical Problems in Engineering*, t. 2016, s. 8 927 169, 2016, ISSN: 1024-123X. DOI: 10.1155/2016/8927169. adr.: https://doi.org/10.1155/2016/8927169.

[119] Q. Duan, E. Al-Shaer i H. Jafarian, „Efficient Random Route Mutation considering flow and network constraints", w *2013 IEEE Conference on Communications and Network Security (CNS)*, 2013, s. 260–268. DOI: 10.1109/CNS.2013.6682715.

[120] J. Liu, H. Zhang i Z. Guo, „A Defense Mechanism of Random Routing Mutation in SDN", *IEICE Transactions on Information and Systems*, t. E100.D, nr. 5, s. 1046–1054, 2017. DOI: 10.1587/transinf.2016EDP7377.

[121] M. Furukawa, K. Kuroda, T. Ogawa i N. Miyaho, „Highly secure communication service architecture using SDN switch", w *2015 10th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT)*, 2015, s. 1–3. DOI: 10.1109/APSITT.2015.7217098.

[122] D. Ma, L. Wang, C. Lei, Z. Xu, H. Zhang i M. Li, „Thwart eavesdropping attacks on network communication based on moving target defense", w *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, 2016, s. 1–2. DOI: 10.1109/PCCC.2016.7820610.

[123] E. Germano da Silva, L. A. Dias Knob, J. A. Wickboldt, L. P. Gaspary, L. Z. Granville i A. Schaeffer-Filho, „Capitalizing on SDN-based SCADA systems: An anti eavesdropping case-study", w *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, s. 165–173. DOI: 10.1109/INM.2015.7140289.

[124] D. Kreutz, F. M. Ramos i P. Verissimo, „Towards Secure and Dependable Software Defined Networks", ser. HotSDN '13, New York, NY, USA: Association for Computing Machinery, 2013, ISBN: 9781450321785. DOI: 10.1145/2491185.2491199. adr.: https://doi.org/10.1145/2491185.2491199.

[125] S. Sorensen, *Security implications of software-defined networks*, Strona odwiedzona: 25.07.2021. adr.: http://fiercetelecom.com/telecom/security-implications-software-defined-networks.

[126] S. M. Kerner, *Is SDN Secure?*, Strona odwiedzona: 25.07.2021. adr.: https://www.enterprisenetworkingplanet.com/security/is-sdn-secure.

[127] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh i B. B. Kang, „Rosemary: A Robust, Secure, and High-Performance Network Operating System", w

Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '14, New York, NY, USA: Association for Computing Machinery, 2014, 78–89, ISBN: 9781450329576. DOI: 10.1145/2660267.2660353. adr.: https://doi.org/10.1145/2660267.2660353.

[128] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson i G. Gu, „A Security Enforcement Kernel for OpenFlow Networks", ser. HotSDN '12, New York, NY, USA: Association for Computing Machinery, 2012, 121–126, ISBN: 9781450314770. DOI: 10.1145/2342441.2342466. adr.: https://doi.org/10.1145/2342441.2342466.

[129] R. Kloti, "OpenFlow: A security analysis", M.S. thesis, Dept. Inf. Tech. Elec. Eng., Swiss Fed. Inst. Technol. Zurich (ETH), Zurich, Switzerland, 2013.

[130] M. Wasserman and S. Hartman, "Security analysis of the Open Networking Foundation (ONF) OpenFlow switch specification", Internet Engineering Task Force, Apr. 2013. datatracker.ietf.org/doc/draft-mrw-sdnsec-openflow-analysis.

[131] S. Shin i G. Gu, „Attacking Software-Defined Networks: A First Feasibility Study", w *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, New York, NY, USA: Association for Computing Machinery, 2013, 165–166, ISBN: 9781450321785. DOI: 10.1145/2491185.2491220. adr.: https://doi.org/10.1145/2491185.2491220.

[132] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller i N. Rao, „Are we ready for SDN? Implementation challenges for software-defined networks", *IEEE Communications Magazine*, t. 51, nr. 7, s. 36–43, 2013. DOI: 10.1109/MCOM.2013.6553676.

[133] R. Chua, *SDN Security - An Oxymoron? New Interview with Phil Porras of SRI International*, Strona odwiedzona: 25.07.2021. adr.: https://www.sdxcentral.com/articles/news/sdn-security-oxymoron-phil-porras-sri/2013/02.

[134] J. Korniak, „The GMPLS Controlled Optical Networks as Industry Communication Platform", *IEEE Transactions on Industrial Informatics*, t. 7, s. 671–678, 2011.

[135] P. Fonseca, R. Bennesby, E. Mota i A. Passito, „A replication component for resilient OpenFlow-based networking", w *2012 IEEE Network Operations and Management Symposium*, 2012, s. 933–939. DOI: 10.1109/NOMS.2012.6212011.

[136] C. Jasson Casey, A. Sutton, G. Dos Reis i A. Sprintson, „Eliminating network protocol vulnerabilities through abstraction and systems language design", w *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013, s. 1–6. DOI: 10.1109/ICNP.2013.6733667.

[137] U. Toseef, A. Zaalouk, T. Rothe, M. Broadbent i K. Pentikousis, „C-BAS: Certificate -Based AAA for SDN Experimental Facilities", w *2014 Third European Workshop on Software Defined Networks*, 2014, s. 91–96. DOI: 10.1109/EWSDN.2014.41.

[138] F. Klaedtke, G. O. Karame, R. Bifulco i H. Cui, „Access Control for SDN Controllers", ser. HotSDN '14, New York, NY, USA: Association for Computing Machinery, 2014, 219–220, ISBN: 9781450329897. DOI: 10.1145/2620728.2620773. adr.: https://doi.org/10.1145/2620728.2620773.

[139] D. Kreutz, A. Bessani, E. Feitosa i H. Cunha, „Towards Secure and Dependable Authentication and Authorization Infrastructures", w *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*, 2014, s. 43–52. DOI: 10.1109/PRDC.2014.14.

[140] D. Kreutz i E. Feitosa, „Identity Providers-as-a-Service built as Cloud-of-Clouds: challenges and opportunities", w *Annals of Computer Science and Information Systems*, PTI, wrz. 2014. DOI: 10.15439/2014f465. adr.: https://doi.org/10.15439/2014f465.

[141] P. E. Veríssimo, N. F. Neves i M. P. Correia, „Intrusion-Tolerant Architectures: Concepts and Design", w *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2003, s. 3–36. DOI: 10.1007/3-540-45177-3_1. adr.: https://doi.org/10.1007/3-540-45177-3_1.

[142] G. Shang, P. Zhe, X. Bin, H. Aiqun i R. Kui, „FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks", w *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, s. 1–9. DOI: 10.1109/INFOCOM.2017.8057009.

[143] S. Deng, X. Gao, Z. Lu i X. Gao, „Packet Injection Attack and Its Defense in Software-Defined Networks", *IEEE Transactions on Information Forensics and Security*, t. 13, nr. 3, s. 695–705, 2018. DOI: 10.1109/TIFS.2017.2765506.

[144] N. Gray, T. Zinner i P. Tran-Gia, „Enhancing SDN security by device fingerprinting", w *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, s. 879–880. DOI: 10.23919/INM.2017.7987393.

[145] D. Kreutz, J. Yu, P. Esteves-Veríssimo, C. Magalhães i F. M. Ramos, „The KISS Principle in Software-Defined Networking: A Framework for Secure Communications", *IEEE Security Privacy*, t. 16, nr. 5, s. 60–70, 2018. DOI: 10.1109/MSP.2018.3761717.

[146] C. Yoon, S. Shin, P. Porras, V. Yegneswaran, H. Kang, M. Fong, B. O'Connor i T. Vachuska, „A Security-Mode for Carrier-Grade SDN Controllers", w *Proceedings of the 33rd Annual Computer Security Applications Conference*, ser. ACSAC 2017, Orlando, FL, USA: Association for Computing Machinery, 2017, 461–473, ISBN: 9781450353458. DOI: 10.1145/3134600.3134603. adr.: https://doi.org/10.1145/3134600.3134603.

[147] K. Wrona, M. Amanowicz, S. Szwaczyk i K. Gierłowski, „SDN testbed for validation of cross-layer data-centric security policies", w *2017 International Conference on Military Communications and Information Systems (ICMCIS)*, 2017, s. 1–6. DOI: 10.1109/ICMCIS.2017.7956483.

[148] X. Qiu, K. Zhang i Q. Ren, „Global Flow Table: A convincing mechanism for security operations in SDN", *Computer Networks*, t. 120, s. 56–70, 2017, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2017.04.002. adr.: https://www.sciencedirect.com/science/article/pii/S1389128617301251.

[149] S. hui Zhang, X. xu Meng i L. hai Wang, „SDNForensics: A Comprehensive Forensics Framework for Software Defined Network", w *Proceedings of the International Conference on Computer Networks and Communication Technology (CNCT 2016)*, Atlantis Press, 2017. DOI: 10.2991/cnct-16.2017.13. adr.: https://doi.org/10.2991/cnct-16.2017.13.

[150] H. Jo, J. Nam i S. Shin, „NOSArmor: Building a Secure Network Operating System”, *Security and Communication Networks*, t. 2018, s. 1–14, 2018. DOI: 10.1155/2018/9178425. adr.: https://doi.org/10.1155/2018/9178425.

[151] M. Conti, F. De Gaspari i L. V. Mancini, „A Novel Stealthy Attack to Gather SDN Configuration-Information”, *IEEE Transactions on Emerging Topics in Computing*, t. 8, nr. 2, s. 328–340, 2020. DOI: 10.1109/TETC.2018.2806977.

[152] B. Ahmed, N. Ahmed, A. W. Malik, M. Jafri i T. Hafeez, „Fingerprinting SDN Policy Parameters: An Empirical Study”, *IEEE Access*, t. 8, s. 142 379–142 392, 2020. DOI: 10.1109/ACCESS.2020.3012176.

[153] M. Yu, T. Xie, T. He, P. McDaniel i Q. K. Burke, „Flow Table Security in SDN: Adversarial Reconnaissance and Intelligent Attacks”, *IEEE/ACM Transactions on Networking*, t. 29, nr. 6, s. 2793–2806, 2021. DOI: 10.1109/TNET.2021.3099717.

[154] P. Nowakowski, P. Zórawski, K. Cabaj, M. Gregorczyk, M. Purski i W. Mazurczyk, „Distributed Packet Inspection for Network Security Purposes in Software-Defined Networking Environments”, w *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES ’20, New York, NY, USA: Association for Computing Machinery, 2020, ISBN: 9781450388337. DOI: 10.1145/3407023.3409210. adr.: https://doi.org/10.1145/3407023.3409210.

[155] Y. Zhou, K. Chen, J. Zhang, J. Leng i Y. Tang, „Exploiting the Vulnerability of Flow Table Overflow in Software-Defined Network: Attack Model, Evaluation, and Defense”, *Security and Communication Networks*, t. 2018, s. 1–15, 2018. DOI: 10.1155/2018/4760632. adr.: https://doi.org/10.1155/2018/4760632.