Article:
Alotaibi, Fahad and Vasilakis, Vasileios orcid.org/0000-0003-4902-8226 (2021) SDN-Based Detection of Self-Propagating Ransomware : The Case of BadRabbit. IEEE Access. pp. 28039-28058. ISSN 2169-3536

https://doi.org/10.1109/ACCESS.2021.3058897

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 8, 2021, accepted February 4, 2021, date of publication February 11, 2021, date of current version February 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3058897

SDN-Based Detection of Self-Propagating Ransomware: The Case of BadRabbit

FAHAD M. ALOTAIBI[1] **AND VASSILIOS G. VASSILAKIS**[2]
[1]College of Science and Arts Sharoura, Najran University, Najran 66446, Saudi Arabia
[2]Department of Computer Science, University of York, York YO10 5GH, U.K.

Corresponding author: Vassilios G. Vassilakis (vv573@york.ac.uk)

This work was supported by the Najran University.

ABSTRACT In the last decade, many ransomware attacks had the ability to spread within local networks or even outside them. At the same time, software defined networking (SDN) has provided a major boost to networks by transferring intelligence from network devices to a programmable logically centralised controller. The latter can be programmed to be compatible with the requirements of a wide range of networks and environments in a straightforward manner. This has motivated researchers to design SDN-based security solutions against threats targeting traditional networks and systems. This article investigates the use of SDN to detect and mitigate the risk of self-propagating ransomware. The infamous BadRabbit ransomware has been used for the proof of concept. To achieve this, an extensive analysis of BadRabbit was performed to identify its characteristics and understand its behaviour at both the infected device level and at the network level. As a result, several unique artifacts were extracted from BadRabbit, which could facilitate its detection. These artifacts were relied upon to design an SDN-based intrusion detection and prevention system. Our system comprises five modules, namely deep packet inspection, ARP scanning detection, packet header inspection, honeypot, and SMB checker. The first two modules have been inspired by other works and have been included for comparison with the existing solutions. Three other modules rely on novel SDN-based methods for ransomware detection. We have also evaluated the efficiency and the performance of our system in terms of detection time, CPU utilisation, as well as TCP and ping latency. Finally, the proposed approach has also been tested for other ransomware families, such as WannaCry and NotPetya. Our experimental results show that the system is effective in terms of detecting self-propagating ransomware and outperforms other proposed approaches.

INDEX TERMS Self-propagating ransomware, intrusion detection and prevention, SDN security, BadRabbit detection.

I. INTRODUCTION

As technology advances and different types of systems become connected to the Internet, cyberthreats are increasing in extent and impact. Ransomware is one such threat which has benefited from people's dependence on new technologies and applications. This threat is a new, unlawful business model emerging from previously known notions of "blackmail". This model exploits organizations' and individuals' increasing need for and dependence on their data by denying them access and requesting ransom in return for restoration of such [1]. Unfortunately, this business model has proved highly successful since 2013 and has accordingly resulted in significant financial losses [2]. The evolution of ransomware went through many stages of development as it appeared from a virus that infects vulnerable systems until achieving the ability to self-propagate within a network. Moreover, ransomware-as-a-service (RaaS) has increased the risk of this threat by giving non-professionals access to this type of tool, and thus increasing the number of potential sources of delivery [1].

Despite the existence of several defences that have prevented the payment of over $100M illegal income for criminals in 2019, ransomware was categorized as one of the highest threats in 2019 according to the Europol report to assess the threat of organized crime online [3]. Some of the largest ransomware attacks occurred in 2017, when three different types of ransomware were deployed, namely WannaCry, Petya/NotPetya and BadRabbit [4]. In the literature, WannaCry and Petya have been adequately studied and

The associate editor coordinating the review of this manuscript and approving it for publication was Giacomo Verticale.

the functions and methods they use to propagate and implement their goals are relatively well understood [5]–[9]. On the contrary, adequate studies are not available for BadRabbit. It has used more advanced techniques such as the targeted attack on Eastern European countries. This resulted in hitting important government sectors such as Odessa International Airport [10]. Therefore, understanding the mechanism by which this type of targeted ransomware operates is one of the major objectives of our work.

Software-defined networking (SDN) is a major development for networks, which has enhanced their flexibility through the transfer of intelligence and decision making from the data plane to a logically centralised controller [11]. This development has motivated well-known technology companies such as Google, Facebook, and Microsoft to support this technology through the funding of the Open Networking Foundation (ONF) to develop standards for SDN [11]. Moreover, Google has networked its data centres using SDN, which is expected to greatly enhance the applicability of SDN-based systems in the future [12]. Consequently, examining security threats on traditional networks and finding solutions to them through the use of SDN, may allow for the provision of a suitable security architecture that can be applied in the future. Despite that, not many SDN-based solutions have been proposed to mitigate the risk of various types of ransomware. The proposed solutions are mostly limited to restricting the risk of malware families such as CryptoWall, Locky, WannaCry, Cerber, and ExPetr [13]–[17]. Therefore, studying previous solutions and providing SDN-based solutions to mitigate the risk from BadRabbit is another incentive point.

The rest of the paper is organised as follows. In Section II we review related works on SDN-based malware and ransomware detection. In Section III we state the problem and discuss the limitations of existing solutions. We also justify our approach and discuss the novelty of our work compared with the existing works. In Section IV we perform a comprehensive static and dynamic analysis of BadRabbit. We present its unique features which can be used to improve the detection. In Sections V and VI we present the design and the implementation of our proposed system that comprises five modules. In Sections VII and VIII we present our experimental setup and our evaluation results. Besides BadRabbit, the proposed approach has also been tested against WannaCry and NotPetya, and has been compared with existing ransomware detection approaches. Finally, Section IX concludes the paper.

II. RELATED WORK

In this section, we review the most important and recent SDN-based works on malware detection in general and on ransomware detection in particular.

A. SDN-BASED MALWARE DETECTION

One of the first efforts to utilize SDN as a security solution against malware was by Jin and Wang [18]. The authors were motivated by the flexibility that SDN delivers and the associated controller programmability. Their proposals are based on detecting mobile malware by assuming that mobiles are connected to access point SDN switches. In their implementation, Jin and Wang proposed four detection algorithms:

1) Using internal or external IPs blacklists, to match with the connections requested by the phones.
2) Based on the assumption that the number of successful connections will be greater than failed connections in healthy phones, they suggested using a connection success ratio algorithm to compare successful communications with failures, as the probability that the device is infected increases with an increasing number of its failed connections. To achieve successful detection, the authors suggested using a predefined value for the difference between the two connections. For example, in the event that the number of unsuccessful connections was greater than successful connections by 20, the device would be blocked.
3) Based on the assumption that communications from uninfected devices will occur at lower rate and to recently accessed destinations, the authors suggested implementing a recently accessed hosts list to allow phones to access the destinations in the list without forwarding to the controller. Otherwise, required access destinations from a specific device are placed in a queue, and if this exceeds a certain number per second, the device would be blocked. Otherwise, access is allowed and the destination is added to the Recently Accessed Hosts.
4) Implementing an algorithm to analyse similar patterns on the network, assuming that malware might infect other phones in the network. These devices perform similar activities in terms of their communications such as identical destinations, similar connection time duration, and that they have the same operating systems.

Evaluation consists of measuring the delay and the maximum number of packets handled by the controller by using Cbench as a simulator to generate packets from 10,000 unique MAC addresses for 10 seconds. This process was repeated 10 times, as it was found that if using the four algorithms the delay increases by 3.2%, while the number of packets processed by the controller decreases by 27.9%.

Ceron *et al.* [19] discussed the difficulties inherent to analysing modern malware that only activates under certain conditions such as the type of network, network structure, and interactions within the network, as some malware may be difficult to analyse in traditional analysis environments due to its development and its identification of predetermined targets. Therefore, they suggest using SDN as a solution to overcome this issue, as SDN allows considerable flexibility in terms of controlling and modifying the network structure. Ceron *et al.* assessed their proposal by analysing 50 malware samples in three different environments: an open environment with no restrictions, which resulted in a mean of 38.94 events being detected; a partial environment with some open ports,

resulting in a mean of 30.04 events being detected; and a closed environment, which was a closed local network without any Internet connection, which resulted in a mean of 18.90 events being detected.

B. SDN-BASED RANSOMWARE DETECTION

When the scope is narrowed to mitigating the risk of ransomware using SDN, to the best of our knowledge, only five pieces of work have discussed the use of SDN as a security solution to detect and mitigate ransomware [13]–[17]. These works can be divided into two categories: ransomware with worm capabilities and non-spreading ransomware. For non-spreading ransomware, the aim is to prevent the victim's device being encrypted, while for ransomware with worm capabilities, the aim is to prevent the victim's device being encrypted in addition to preventing the spread of the ransomware within, and indeed, outside the local network. Table 1 summarises the features of the proposed approaches in the literature, whereas Table 2 gives a comparison between the proposed solutions to mitigate the risk of ransomware. These works are briefly reviewed below.

TABLE 1. SDN-based ransomware detection approaches proposed in the literature.

Feature/Paper	[13]	[14]	[15]	[16]	[17]
Worm capabilities				X	X
HTTP features		X			
DNS inspection	X			X	
Packets inspection					X
HTTPS features			X		
IPs/Domains blacklisting	X			X	
Port blocking				X	X
ML based		X	X		

Cabaj and Mazurczyk [13] studied CryptoWall 3.0, which is a non-spreading ransomware. They proposed a solution to prevent the victim's device being encrypted by matching the domains used in external communications with a dynamic blacklist. Thus, the communication between the infected device and the command and control (C&C) server is detected and the exchange of the encryption key is stopped, which leads to the encryption being suspended. The authors also proposed another solution to provide better time efficiency, which is to keep the communication in progress between the devices and external parties without the need to wait for the SDN controller's instructions, and to forward a copy to the SDN controller. The latter then performs the inspection and blocks any ongoing communication with a suspicious domain. Moreover, a comparison between the *iptables* firewall and the proposed approaches has been made, which showed that *iptables* might lead to delay of 300 ms on insertion of 1000 new rules, while both suggested approaches have a maximum delay of only 180 ms. In a follow up work, Cabaj *et al.* [14] utilized SDN to detect two families of non-spreading ransomware, namely CryptoWall and Locky. In their implementation of the proposed solution, they used the size of the first three HTTP Post messages sequentially as a detection method, which achieved a 97-98% true positive rate and a 2-3% false positive rate.

In another solution that is based on the characteristics of the messages exchanged between the controller and the ransomware, Cusack *et al.* [15] found that cyber-criminals began to change the protocol used to communicate between themselves and their ransomware. They used HTTPS to encrypt these messages, which prevents the inspection process. Hence, the authors proposed a solution based on the unencrypted part of HTTPS, specifically the header, to detect malware using machine learning (ML). To demonstrate the feasibility of their solution, they tested it against the Cerber ransomware, which resulted in an 87% true positive rate.

When it comes to ransomware with worm-spreading capabilities, researchers have studied two related families: WannaCry and ExPetr. For WannaCry [16], the suggestion is to use two applications. The first uses a dynamic IP blacklist to detect WannaCry's communication with its C&C, which prevents the victim's device from being encrypted. The second application monitors the ports used by WannaCry (specifically ports 139, 445, 443, 9001, and 9050) to detect and block WannaCry's self-propagating within and outside the network, in addition to preventing the encryption process. The authors tested their solution, which resulted in successful detection and blocking of WannaCry. It is worth noting that in their second application, three ports belonging to a legitimate service used routinely, specifically 443 for HTTPS, 445 for SMB, and 139 for NetBIOS, were blocked. For ExPetr [17], the suggested solutions are based on blocking suspicious ports as well as inspecting HTTP and SMB payloads. In the case of the organization in question not using SMB, the port blocker can be applied to block any connection that attempts to use SMB. Where SMB is a legitimate service, SMB packets are inspected to search for a Bitcoin address; if the address is found, the connection would be blocked for a pre-defined time duration (e.g., 90 minutes). Similarly, the HTTP packet inspection method is used to search for `PROPFIND` in order to identify attempts to connect to the `admin$` shared folder.

III. PROBLEM STATEMENT

In this section, we state the problem and identify the limitations of the existing solutions. We also explain and justify our approach. As shown in Table 2, only two of the existing solutions consider ransomware with worm (self-propagating) capabilities. That is, [13]–[15] propose approaches to detect ransomware communications with the C&C, in order, for example, to prevent a victim's device from being encrypted. On the other hand, approaches proposed in [16], [17] attempt to detect ransomwares' attempts to spread within a network. Our results of the BadRabbit ransomware analysis given later in Section IV indicate that a system infected with the BadRabbit ransomware does not make attempts to communicate with third parties, as it depends on generating an encryption key using Microsoft libraries and then encrypting this key with a public key in a form of behaviour similar to that of ExPetr [17]. The aim of our work is to detect attempts

TABLE 2. Comparison of ransomware detection and prevention approaches using SDN.

Ransomware	Worm	Methodology	Advantages	Disadvantages
CryptoWall [13]	No	Matching the domains used in the DNS with a dynamic domain blacklist.	Prevent victim encryption.	Pre-defined list.
CryptoWall and Locky [14]	No	Pre-defined suspicious HTTP Post features.	A possible prevention of victim encryption.	Possibility of high False Positives rate. High pressure on the controller as a result of forwarding three packets of each connection.
Cerber [15]	No	Combination of observed suspicious network connections from Cerber to its C&C analysed by ML.	No need to use pre-defined lists. Possible victim encryption prevention.	High False Positives rate. Possibility of victim encryption 12.5%. Detection accuracy 87%.
WannaCry [16]	Yes	Matching connections to pre-defined suspicious ports and IPs blacklist.	Prevent victim encryption and ransomware spreading.	Pre-defined list. High False Positives rate due to block legitimate ports.
ExPetr [17]	Yes	Port blocker. HTTP and SMB packets inspection.	Prevent victim encryption and ransomware spreading.	One packet inspection instead of chain of packets.

to spread ransomware at the network level rather than to prevent device encryption, which has already been addressed in previous works.

Solutions presented in [16], [17] may result in high numbers of false positives by blocking an unnecessary high number of legitimate users. Moreover, they are essentially dependent on the network not needing the SMB service. This would allow port blocking as a solution to prevent ransomware from spreading. These solutions are effective in this regard, but do not take into account the possibility of the need for this service on the network, thus preventing users from accessing a legitimate service. In [17], as an alternative solution the authors suggested inspecting HTTP and SMB packets to search for the specific values that ExPetr uses, such as the Bitcoin address and the `admin$` folder name. This solution may be useful in preventing ExPetr from spreading but may not prevent other types of ransomware because of the need for prior knowledge, such as Bitcoin addresses, file names, or other unique strings.

In fact, all the solutions discussed above assume the controller's ability to access and analyse the packets very rapidly. The controller's capabilities in this regard, the network size, and typical amounts of traffic have not been studied sufficiently. Indeed, it is difficult for the controller to inspect all the traffic destined for a specific service or analyse the domains or IP addresses used, especially if the network contains a large number of running devices. The difficulty here is based on the network performance, as additional inspection means increased delays and/or degradation in performance. Therefore, a general solution must be found that detects different types of ransomware without affecting significantly the efficiency of the network and the requirements of its users. As an alternative solution, a honeypot could provide these features and is a proportionate and efficient solution investigated in our work.

The use of honeypot as a ransomware solution has been suggested in a number of studies. For example, Gomez-Hernandez *et al.* in [20] use a honeypot to detect ransomware, suggesting that files on the network devices can be deployed to act as a means of threat detection. These files are connected to a monitoring process, and therefore, when opening or reading any of these files, the process responsible will be detected and then blocked. The authors discussed the advantages of their proposal, stating that this approach does not require the application of initial training or prior knowledge of threats, thus allowing for efficient detection of unknown threats. To demonstrate the effectiveness of the approach, files were distributed across a Linux network, where Bash-Ransomware and Linux.Encoder were used as samples, and WannaCry was also used by using WineHQ to run it. This method achieved a 100% detection rate. Recently, several types of honeypots have been proposed for the specific detection of ransomware. For instance, shared folders with other devices, a web server, a fake memory partition, a file and e-mail or pictures as tokens. These traps were trialled against six types of ransomware, including CryptoWall, which was run 50 times on the device and for which a 100% detection rate was achieved [21]. In other words, these solutions are based on the creation of a sample that is not used by the network or its clients, and thus any attempt to reach it can reasonably be considered suspicious.

In fact, the honeypot approach may be an effective and straightforward solution to implement, whether at the network level using systems to act as traps, as suggested in [21], or at the level of devices, as suggested in [20], by using files to act as traps. The effectiveness and privilege of this approach lies in its ease of implementation and its ability to detect unknown threats. On the other hand, each approach has negative aspects, which for this approach lies in the demand for a considerable awareness on the part of the network's users not to access these files or systems, whereby any access to these traps will be considered suspicious and thus the device or process responsible for access is blocked, resulting in a false positive detection.

On the other hand, regardless of the drawbacks of the traffic inspection previously discussed, payload inspection has also been used in this work as a supplementary measure to mitigate the spread of BadRabbit within a given network. In particular,

the following modules are suggested by [19] for designing an SDN-based malware analysis architecture. These modules are working on top of the POX controller:

1) *Inspection:* This inspects the traffic within the network as well as the traffic directed outside the network to match it with predefined values such as specific IP addresses, specific values used by malware, and specific ports. In the event of a match, either the containment, the configuration manager, or the analysis control module is assigned to deal with connection based on the possible associated risk. For example, not allowing malware to attack predefined external destinations using the containment module.
2) *Containment:* This is used to mitigate the risk from malware using OpenFlow to install rules in SDN switches, as the risk can be one of either spreading to or attacking other internal/external devices.
3) *Configuration Manager:* This is used to create changes to the network to analyse malware activities in several environments, where changes can be implemented to the network topology by adding or deleting services (such as adding a web server or removing an SMTP server), or configuring the network to be similar to a university network.
4) *Analysis Control:* This is used for the safety control of the analysis process. The module enables the controller to re-execute the malware several times as well as revert the device to a clean state.

Note, that the aformetioned modules have not been adopted as part of the existing ransomware detection mechanisms (i.e., [16], [17]). We believe that they deserve attention and have been considered in our proposed design.

IV. BadRabbit ANALYSIS

In this section, the main findings from our extensive analysis of BadRabbit ransomware are presented. The main machine specifications that were used in the analysis are: Intel Core i7 8550U, 1.80 GHz, and 16GB RAM. To perform the analysis, VirtualBox was used to host the virtual machines (VMs). For the static analysis, two VMs were used: Windows 10 and REMnux. For dynamic analysis, four VMs were hosted, with the following roles: REMnux as a gateway where the other systems were linked to it and also as a fake HTTP service, two Windows 10 systems, one infected with BadRabbit and the other healthy, and one Windows 7 system.

A. THE WORM COMPONENT

By analysing the BadRabbit main file, it was found that the file size is large but contains little data in the source code. In addition, there is a presence of two suspicious entropy values (Figure 1). This suggests that the file is compressed or encrypted. After the file was executed in a safe environment, it was found that the file works as a dropper for another file, specifically the worm file named `infpub.dat`. The worm file itself has been coded to work for propagation within the local network, in addition to a drop-in for three other files.

```
PE check for 'flashplayer.bin':
Entropy: 7.891914 (Min=0.0, Max=8.0)
MD5     hash: fbbdc39af1139aebba4da004475e8839
SHA-1   hash: de5c8d858e6e41da715dca1c019df0bfb92d32c0
SHA-256 hash: 630325cac09ac3fab908f903e3b00d0dadd5fdaa
SHA-512 hash:
74eca8c01de215b33d5ceea1fda3f3bef96b513f58a750dba04b0d
04d4721a9a96fa2e18c6888fd67fa77686af87
.text entropy: 6.584104 (Min=0.0, Max=8.0)
.rdata entropy: 7.177259 (Min=0.0, Max=8.0)
.data entropy: 0.183339 (Min=0.0, Max=8.0)
.rsrc entropy: 4.204086 (Min=0.0, Max=8.0)
.reloc entropy: 3.293139 (Min=0.0, Max=8.0)
```

FIGURE 1. The dropper entropy.

Two of these files are dropped into the Windows directory, while the third is dropped into the Windows Temp folder. These files are:

- `Dispci.exe`: Performs the encryption and decryption with the help of the DiskCryptor driver, `cscc.dat`, which encrypts files individually based on their extensions.
- `Cscc.dat`: A legitimate file recognised as a DiskCryptor driver, and which is used to encrypt/decrypt system partitions. In this instance, `cscc.dat` encrypts all the disk partitions.
- `Random Value.tmp`: A Mimikatz tool used to extract account authentication data from the system.

The worm component runs the above files as follows:

- In order to run the DiskCryptor driver, the malicious file (`infpub.dat`) creates a service called Windows Client Side Caching DDriver.
- Schedules a task to execute the `dispci.exe` file.
- Uses ConnectNamePipe as a communication point with the Mimikatz tool.

Also, the worm component schedules three system tasks, which are:

- Rhaegal: runs the `dispci.exe` file each time the system is turned on.
- Drogon: reboots the system once to enable the encryption process.
- Viserion: shuts down the system after the completion of the encryption process.

B. THE ENCRYPTION COMPONENT

A high-level analysis of the encryption component (`dispci.exe`) reveals that the file contains false information to deceive the user and to hide its true nature. This information was extracted using the Pescanner tool, where the file was described as a Microsoft Display Class Installer and the product name given as GrayWorm. However, signature scans indicate that it is a diskcoder typically used by Windows-based ransomware (Figure 2). For a more accurate analysis the strings were extracted; it was found that the file contains an RSA-2048 key and 113 file extensions. To understand the encryption mechanism, the libraries and functions used were extracted. It was found that the `ADVAPI32.dll` library functions are used extensively to generate keys and

```
=========================================================
LegalCopyright  : http://diskcryptor.net/
FileVersion     : 1.1.846.118
ProductName     : GrayWorm
ProductVersion  : 1.1
FileDescription : Microsoft Display Class Installer
OriginalFilename: dispci.exe
Translation     : 0x0009 0x04b0

Signature scans
=========================================================

Clamav: dispice.bin: Win.Ransomware.Diskcoder-6355410-0
```

FIGURE 2. Dispci.exe PE header.

```
call    ds:CreateFileMappingW
mov     ebx, eax
test    ebx, ebx
jz      short loc_401B1E
mov     edx, [ebp+dwNumberOfBytesToMap]
push    edx             ; dwNumberOfBytesToM
push    0               ; dwFileOffsetLow
push    0               ; dwFileOffsetHigh
push    6               ; dwDesiredAccess
push    ebx             ; hFileMappingObject
call    ds:MapViewOfFile
```

FIGURE 3. File mapping.

to perform the encryption and decryption processes; these functions are presented in Table 3. It is worth noting that CryptDestroyKey is used to destroy the encryption key after using it. Moreover, the functions CryptImportPublicKeyInfo, CryptStringToBinaryW, CryptDecodeObjectEx, and CryptBinaryToStringW from the `CRYPT32.dll` library are used to load the integrated public key to encrypt the file encryption key.

TABLE 3. Functions used by the encryption component.

CryptDestroyHash	CryptAcquireContextW	CryptDeriveKey
CryptDuplicateKey	CryptDuplicateHash	CryptHashData
CryptGetHashParam	CryptDecrypt	CryptDestroyKey
CryptCreateHash	CryptEncrypt	CryptGenRandom
CryptGetKeyParam	CryptReleaseContext	CryptSetKeyParam

The extensions of the files that are encrypted by BadRabbit are given in Appendix A. The malware author's public key (RSA-2048) is given in Appendix B.

C. ENCRYPTION PROCESS

The encryption component file (`dispci.exe`) operates as follows. It first invokes the malware author's public key using CryptImportPublicKeyInfo and then calls the file extensions. After that, the disk is accessed through the use of the command `ArcName\multi(0)disk(0)rdisk(0)partition(1)\`, and an attempt is made to access different file system formats, such as NTFS, FAT12, FAT16, and FAT32. Next, the AES-CBC 128-bit key is generated using the CryptGenRandom function, and then file mapping begins to encrypt the files (Figure 3). It is worth noting that there are exceptions to some folders, these folders are AppData, ProgramData, Program Files, and Windows. After the encryption is complete, the AES key is itself encrypted using the author's public key, and a `Readme.txt` file containing information about the payment mechanism and the encrypted key is dropped in the *C* path. Noticeably, the file also drops a shortcut of itself called DECRYPT onto the desktop for the decryption.

D. NETWORK ENUMERATION

BadRabbit depends on various methods to enumerate the devices on the network. Some of these methods depend on the active connections in the network, while others depend on the extraction of network device addresses (whether active or inactive) through ARP requests. These enumeration methods work as follows:

- By extracting recent TCP connections from GetExtendedTcpTable, which contains a group consisting of the IP addresses and ports communicated within a network.
- By using the GetIpNetTable function, which enumerates the network's ARP entries in a table specifying the physical addresses, and returns this information in a table structure.
- By using NetServerEnum to extract the addresses of all servers in the network, whether all servers generally or servers that contain a specific service.
- Utilizing the Windows Server DHCP Services API via the DhcpEnumSubnetClients, DhcpEnumSubnets, and DhcpGetSubnetInfo functions to extract subnets and clients in subnets, in addition to extracting information about a specific client.

E. PROPAGATION METHODS

BadRabbit makes several attempts to spread within the network using four methods detailed below:

1. The worm component runs the Mimikatz tool to extract the authentication information from the infected system and the Active Directory in Windows networks. The worm file first drops a temporary file there with a random name and then creates a named pipe to act as a point of communication between the worm and the tool as shown in Figure 4. After that, it runs the tool using the command line, where the command used to run the tool contains the name of the temporary file in addition to the name of the named pipe created (Figure 5). Furthermore, the worm attempts to use the Windows Management Interface Command (WMIC) in order to access the Windows Management Instrumentation (WMI) through the use of any extracted passwords and usernames.

2. Using the authentication information extracted by the Mimikatz tool, BadRabbit attempts to access SMB shared services. If unsuccessful, the worm contains a list of usernames and passwords that it can use to launch a dictionary attack against network devices that use SMB. Appendix C reports the built-in usernames used by the worm, while the integrated passwords are given in Appendix D.

3. Attempts to copy the files `infpub.dat` and `cscc.dat` to the `admin$` folder, by establishing a remote connection to `admin$`, and then copying BadRabbit files

```
call    ds:CreateNamedPipeW
mov     [ebp+hNamedPipe], eax
cmp     eax, 0FFFFFFFFh
jz      short loc_1000705F
push    esi             ; lpOverlapped
push    eax             ; hNamedPipe
call    ds:ConnectNamedPipe
test    eax, eax
jz      loc_1000712F
push    1Eh
pop     edi

                        ; CODE XREF: sub_10006FFE
push    esi             ; lpBytesLeftThisMessage
lea     eax, [ebp+TotalBytesAvail]
push    eax             ; lpTotalBytesAvail
push    esi             ; lpBytesRead
push    esi             ; nBufferSize
push    esi             ; lpBuffer
push    [ebp+hNamedPipe] ; hNamedPipe
dec     edi
mov     [ebp+TotalBytesAvail], esi
call    ds:PeekNamedPipe
```

FIGURE 4. Creation of a named pipe to act as a line of communication between Mimikatz and the worm file.

```
lea     eax, [ebp+CommandLine]
push    offset aWsWs    ; "\"%ws\" %ws"
push    eax             ; LPWSTR
mov     [ebp+Dst], edi
```

FIGURE 5. The command used to run Mimikatz and the named pipe.

to the same folder. After that, BadRabbit can be executed remotely.

4. The worm component exploits one of MS17-010's vulnerabilities called EternalRomance. It is an SMB vulnerability previously leaked by The Shadow Brokers group to spread across the network, where this is one of several vulnerabilities that resulted from poor handling of transactions. According to the Cisco Talos Intelligence Group [22], exploiting the vulnerability is similar to one of the exploits that was published on GitHub [23]. Appendix E shows two snippets from the worm source code, showing the similarity between the exploit published in [23] and the worm file. These snippets describe the use of the strings "Frag" and "Free" to match them in the response analysis stage. In case of a match, the information leakage is successful and there is an opportunity for the continuation of the exploit [24].

F. DEFENSIVE STRATEGIES

From BadRabbit worm file source code analysis we found that certain defensive techniques have been deployed to avoid detection and hinder the code analysis. The first task performed by the worm component (`infpub.dat`) is to generate hashes for all processes running on the infected device. Afterwards, the presence of specific hashes is checked, where these values represent the hashes for antivirus software; more precisely, the hashes of Dr. Web and McAfee products. Dealing with these processes is dependent on the antivirus used. For example, in the case of matching any of these hashes, certain malicious tasks such as accessing SMB shared folders will not be performed. Furthermore, the worm file applies anti-debugging techniques; the IsDebuggerPresent function is used to determine any debuggers running and then disavow the debugger by placing traps to prevent it from properly monitoring any processes being performed (Figures 6 and 7).

```
call    ds:IsDebuggerPresent
mov     dword_419880, eax
push    1
call    sub_40B122
pop     ecx
push    0               ; lpTopLevelExceptionFilter
call    ds:SetUnhandledExceptionFilter
push    offset ExceptionInfo ; ExceptionInfo
call    ds:UnhandledExceptionFilter
cmp     dword_419880, 0
jnz     short loc_406851
```

FIGURE 6. Using the function IsDebuggerPresent to identify debugger existence.

```
= dword ptr  8

mov     edi, edi
push    ebp
mov     ebp, esp
call    sub_4081A3
push    [ebp+arg_0]
call    sub_407FF4
pop     ecx
push    0FFh            ; uExitCode
call    sub_407F7A
int     3               ; Trap to Debugger
endp
```

FIGURE 7. Traps created for debuggers.

After completion of its activities, the worm deletes the records to prevent an analyst from tracking the operations that occurred on the infected machine. This is done by deleting the logs in System, Security, Application, and Setup, as shown in Figure 8. Also, the `fsutil.exe usn deletejournal/D %c` command is executed to delete the existing Update Sequence Number, as it is considered a record containing all changes that occurred on files.

G. NETWORK ACTIVITY

1) TEST SCENARIOS AND CONFIGURATIONS

BadRabbit's network activity was captured by running the file under four different scenarios to more accurately understand its behaviour. These scenarios share the structure described in Figure 9 using the network configuration shown in Table 4, but differ in terms of the victims' devices. Specifically, a Windows 10 system with IP address 10.0.0.4 and a Windows 7 system with IP address 10.0.0.6, where some additional modifications were made to these systems. Ransomware execution is performed in the Windows 10 system with IP address 10.0.0.5 The REMnux system acts as a gateway with

```
mov     esi, offset aWswevtutilClWs ; "%wswevt
push    esi             ; LPCWSTR
push    eax             ; LPWSTR
call    ebx ; wsprintfW
push    offset aSystem  ; "System"
lea     eax, [ebp+var_208]
push    eax
push    esi             ; LPCWSTR
push    eax             ; LPWSTR
call    ebx ; wsprintfW
push    offset aSecurity ; "Security"
lea     eax, [ebp+var_208]
push    eax
push    esi             ; LPCWSTR
push    eax             ; LPWSTR
call    ebx ; wsprintfW
push    offset aApplication ; "Application"
lea     eax, [ebp+var_208]
```

FIGURE 8. Deleting the log records.

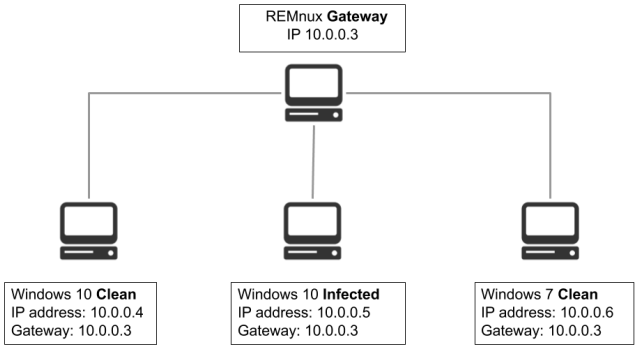

FIGURE 9. The lab scheme used in static and dynamic analysis.

TABLE 4. Systems used in network-level analysis and their assigned IP addresses.

System	IP address
REMnux (Gateway)	10.0.0.3
Windows 10 (Clean)	10.0.0.4
Windows 10 (Infected)	10.0.0.5
Windows 7 (Clean)	10.0.0.6

IP address 10.0.0.3. The modifications that have been made and the scenarios that have been considered are as follows:

- *Scenario 1:* Two Windows systems are connected to the network. One system is running Windows 7 that has a password that does not exist in the ransomware's dictionary attack list. The second system is Windows 10 and has the same authentication information as the infected system. The reason for using such a structure is to identify the order in which BadRabbit propagates and the techniques it uses to do so.
- *Scenario 2:* One Windows 7 system is connected to the network which is unpatched with regard to MS17-010 vulnerabilities. Furthermore, SMBv1 authentication has been activated in both the infected system and the Windows 7 system. As a result, it can be determined whether the ransomware has exploited the EternalRomance vulnerability or not.
- *Scenario 3:* One Windows 10 system is connected to the network but which has different authentication information that is also not in the ransomware's dictionary list. This has been done to monitor ransomware activity in an otherwise well-secured network.
- *Scenario 4:* Two Windows systems are connected to the network, one of which is a Windows 10 system with a username and a password that exists in the ransomware's dictionary list, whilst the other one is a Windows 7 with a password that does not exist in the list or that is used by other devices on the network.

Although there is no clear evidence about the use of web services using web transmission protocols such as HTTP and HTTPS from the static analysis, there is a possibility that BadRabbit copies itself through the `admin$` folder, as explained previously in Section IV-E. This could be achieved through an extension of HTTP called WebDAV [25]. To verify this hypothesis, INetSim was used to act as a simulator for web services, and indeed other services as well. Our findings are presented in the following subsection.

2) BadRabbit ACTIVITIES IN THE CONSIDERED SCENARIOS

The traffic that was generated by the infected system was analysed using Wireshark. This has revealed the steps BadRabbit follows to self-propagate in each of the tested scenarios. In general, the infected system first checks the availability of two services in other clients, specifically the HTTP and SMB services. After identifying clients that use the SMB protocol, an attempt is made to establish a connection between the infected system and systems that use SMB by using authentication information stolen from the session (Figure 10). The stolen authentication information belongs only to the infected system, thanks for not using the Active Directory in the network. However, in the case of using the Active Directory, all the extracted passwords will be tried against each client.

If the connection attempt is unsuccessful, another method of propagating within the network is used by exploiting the EternalRomance vulnerability. To do this, the infected system tries to find vulnerable systems through sending SMB requests that contain the dialect `NT LM 0.12`, which represents SMB NTLMv1 challenge-response authentication protocol. If the receiver is using SMBv1, the response will include the same dialect which is an indicator of EternalRomance vulnerability existence possibility (Figure 11). Also, by monitoring the SMB connections it was observed that fixed sizes are used for the first three packets. In particular, first the infected system sends a request with a size of 127 bytes, then receives an answer with a size of 228, and then sends a request with a size of 232. Another attempt to spread, but against specific victims, is by contacting clients/servers that use web services (specifically, the HTTP protocol on port 80) and then trying to access the `admin$` shared folder through HTTP PROPFIND (Figure 12).

Furthermore, the infected system launches a dictionary attack on NTLMSSP authentication as a target to

10.0.0.3	TCP	49744 → 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK
10.0.0.5	TCP	445 → 49744 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10.0.0.6	TCP	49745 → 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK
10.0.0.5	TCP	445 → 49745 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
10.0.0.6	TCP	49745 → 445 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
10.0.0.6	SMB	Negotiate Protocol Request
10.0.0.5	SMB2	Negotiate Protocol Response
10.0.0.6	SMB2	Negotiate Protocol Request
10.0.0.5	SMB2	Negotiate Protocol Response
10.0.0.6	SMB2	Session Setup Request, NTLMSSP_NEGOTIATE
10.0.0.5	SMB2	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIR
10.0.0.6	SMB2	Session Setup Request, NTLMSSP_AUTH, User: MSEDGEWIN10\IEUse
10.0.0.5	SMB2	Session Setup Response, Error: STATUS_LOGON_FAILURE
10.0.0.6	TCP	49745 → 445 [RST, ACK] Seq=1021 Ack=743 Win=0 Len=0
10.0.0.6	TCP	49746 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_

FIGURE 10. An attempt to identify available HTTP and SMB services on clients and an unsuccessful attempt to log in via SMB using stolen information.

```
16:13:34.555171  10.0.0.5   10.0.0.6   SMB   Negotiate
16:13:34.555492  10.0.0.5   10.0.0.4   SMB   Negotiate
 331: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits)
net II, Src: PcsCompu_e6:e5:59 (08:00:27:e6:e5:59), Dst: PcsCompu_68:
net Protocol Version 4, Src: 10.0.0.5, Dst: 10.0.0.6
mission Control Protocol, Src Port: 49759, Dst Port: 445, Seq: 1, Ack
OS Session Service
Server Message Block Protocol)
B Header
gotiate Protocol Request (0x72)
 Word Count (WCT): 0
 Byte Count (BCC): 34
 Requested Dialects
 ▼ Dialect: NT LM 0.12
     Buffer Format: Dialect (2)
     Name: NT LM 0.12
```

FIGURE 11. An attempt to use SMBv1.

10.0.0.5	10.0.0.3	HTTP	OPTIONS / HTTP/1.1
10.0.0.3	10.0.0.5	HTTP	HTTP/1.1 200 OK
10.0.0.5	10.0.0.3	HTTP	OPTIONS /admin%24 HTTP/1.1
10.0.0.3	10.0.0.5	HTTP	HTTP/1.1 200 OK
10.0.0.5	10.0.0.3	HTTP	PROPFIND /admin%24 HTTP/1.1

FIGURE 12. An attempt to access the admin$ share folder using HTTP PROPFIND.

access SMB. In some connections, the infected system also tries to access the SMB shared folders, as shown in Figure 13. After completing these actions, the infected system performs a full network scan using ARP to extract the clients connected to the network; the infected system sends ARP queries three times for each address in the network. It is worth noting that the process of scanning the network using ARP is widely used by different families of ransomware, such as WannaCry.

In scenario 1, after identifying clients who use SMB, an attempt is made to communicate with these clients through the use of authentication information stolen from the session. In case the connection succeeds (as is the case in this scenario) between the infected system (10.0.0.5) and the clean Windows 10 system (10.0.0.4), the infected system requests a connection to `admin$`, and then checks the presence of the `cscc.dat` file. If the file is found, it is opened. In the absence of the file, a error is raised. After that the infected system inquires about the existence of the `infpub.dat` file. If the file exists, it is overwritten. In the absence of such a file, the file is created and data is transferred to it (Figure 14). When the file transmission is complete, the DCE/RPC protocol is used to request a query of SVCCTL by using DCE/RPC Endpoint Mapper (EPM) and then calling it to control Windows services, especially StartServiceW, to execute the ransomware (Figure 15). After the process is over, the infected

SMB	Tree Connect AndX Request, Path: \\10.0.0.6\IPC$
SMB	Tree Connect AndX Response
SMB	NT Create AndX Request, Path: atsvc
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: browser
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: eventlog
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: lsarpc
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: netlogon
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: ntsvcs
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: spoolss
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: samr
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: srvsvc
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: scerpc
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: svcctl
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	NT Create AndX Request, Path: wkssvc
SMB	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
SMB	Logoff AndX Request

FIGURE 13. An attempt to access SMB share folders.

10.0.0.4	SMB2	Tree Connect Request Tree: \\10.0.0.4\admin$
10.0.0.5	SMB2	Tree Connect Response
10.0.0.4	SMB2	Ioctl Request FSCTL_QUERY_NETWORK_INTERFACE_INFO
10.0.0.5	SMB2	Ioctl Response FSCTL_QUERY_NETWORK_INTERFACE_INFO
10.0.0.4	SMB2	Create Request File: cscc.dat
10.0.0.5	SMB2	Create Response, Error: STATUS_OBJECT_NAME_NOT_FOUND
10.0.0.4	SMB2	Create Request File:
10.0.0.5	SMB2	Create Response File:
10.0.0.4	SMB2	Close Request File:
10.0.0.5	SMB2	Close Response
10.0.0.4	SMB2	Create Request File: infpub.dat
10.0.0.5	SMB2	Create Response File: infpub.dat

FIGURE 14. Successful authentication using credentials stolen from the session.

10.0.0.4	DCERPC	Bind: call_id: 2, Fragment: Single, 2 context items: EPM
10.0.0.5	DCERPC	Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 ma
10.0.0.4	EPM	Map request, SVCCTL, 32bit NDR
10.0.0.5	EPM	Map response, SVCCTL, 32bit NDR
10.0.0.4	TCP	49763 → 49669 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
10.0.0.5	TCP	49669 → 49763 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
10.0.0.4	TCP	49763 → 49669 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
10.0.0.4	DCERPC	Bind: call_id: 2, Fragment: Single, 2 context items: SVC
10.0.0.5	DCERPC	Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 ma
10.0.0.4	DCERPC	AUTH3: call_id: 2, Fragment: Single, NTLMSSP_AUTH, User:
10.0.0.4	SVCCTL	OpenSCManagerW request
10.0.0.5	TCP	49669 → 49763 [ACK] Seq=291 Ack=803 Win=2101760 Len=0
10.0.0.5	SVCCTL	OpenSCManagerW response
10.0.0.4	SVCCTL	Unknown operation 60 request
10.0.0.5	SVCCTL	Unknown operation 60 response
10.0.0.4	SVCCTL	StartServiceW request
10.0.0.5	TCP	49669 → 49763 [ACK] Seq=451 Ack=1203 Win=2101248 Len=0
10.0.0.4	TCP	49760 → 445 [ACK] Seq=413811 Ack=3531 Win=262656 Len=0
10.0.0.4	TCP	49762 → 135 [ACK] Seq=273 Ack=237 Win=2102016 Len=0
10.0.0.5	SVCCTL	StartServiceW response

FIGURE 15. Using the DCE/RPC and EPM protocols to request SVCCTL.

system also lunches a dictionary attack on the other systems to authenticate through NTLMSSP as explained earlier.

In scenario 2, the infected system performs the same steps described previously and in the same order, starting from checking the services used and then trying to log into the SMB shared folders using the stolen authentication data. In this scenario, however, if the connection is not successful, the infected system determines the clients using HTTP and then attempts to access `admin$` through PROPFIND requests supported by an extension to the HTTP protocol, namely WebDAV. This protocol provides file sharing and remote control service. As this protocol is not configured on the REMnux system, this method was not successful, as shown in Figure 16. After that, the infected device tries to exploit the SMB vulnerability on the Windows 7

10.0.0.5	10.0.0.3	HTTP	PROPFIND /admin%24/ HTTP/1.1
10.0.0.3	10.0.0.5	HTTP	HTTP/1.1 501 Method Not Implemented
10.0.0.5	10.0.0.3	HTTP	PROPFIND /admin%24/infpub.dat HTTP/1.1
10.0.0.3	10.0.0.5	HTTP	HTTP/1.1 501 Method Not Implemented

FIGURE 16. An attempt to move suspicious files to the admin$ folder.

device (10.0.0.6), but this was not successful either. Two experiments were performed: one using the SMBv1 protocol and the other using SMBv2. In both, the infected system did not succeed in exploiting the vulnerability correctly, despite the evidence in the static analysis of coding to attempt to use the EternalRomance vulnerability. In practice, however, no evidence was found to verify this.

In scenario 3, the goal is to analyse the activities carried out by the infected system after the malware propagation methods within the network have completed. Our experiments show that the infected system performs only the usual propagation processes as in the previous scenarios.

In scenario 4, the infected system first tried the stolen authentication data on Windows 10 system (10.0.0.4), and then on Windows 7 (10.0.0.6); the two attempts did not succeed. After that, an effort was made to launch a dictionary attack on Windows 7, and that did not succeed either. However, afterwards BadRabbit did not move to the other system (10.0.0.4) to start a dictionary attack. It is worth noting that this behavior was observed through many attempts to analyse BadRabbit dynamically, as the infected system launches a dictionary attack on only the last system, attempted to log into it.

After the self-propagation process within the network is completed, the infected system is restarted to complete the file encryption process. After that, when trying to reboot the system, a message appears asking to pay the ransom amount. This is because BadRabbit rewrites the Master Boot Record (MBR), which makes the device unable to access the operating system.

V. RANSOMWARE DETECTION SYSTEM: DESIGN

To design effective solutions against BadRabbit, both our analysis (in Section IV) as well as the solutions proposed in the literature (in Section II) have been taken into account. We have reviewed and evaluated the proposed solutions against ransomware activities and have taken into consideration their efficiency and the extent of their impact on the network performance. As a result, we have designed and implemented an SDN-based intrusion detection and prevention system (IDPS) comprising five modules. This sections describes the system design, whereas Section VI provides the implementation details using POX SDN controller. The IDPS detects and blocks self-propagating ransomware, such as BadRabbit. Two of the modules, namely Deep Packet Inspection (DPI)-based and ARP-scanning based detection, have been inspired by the work of [17]. At the same time, three novel modules/methods have been developed, namely Packet Header Inspection (PHI), the use of a honeypot, and a detection based on SMB packet size.

A. MODULE 1: DEEP PACKET INSPECTION (DPI)

As identified in Section IV, BadRabbit attempts to transfer a file when SMB authentication succeeds or via HTTP PROPFIND requests. This file has a fixed and unique name, `infpub.dat`, and there are no services known by this name. For the purposes of detection, HTTP and SMB traffic inspection are used to search for the name of this file. Figure 17 gives a high-level representation of the concept.

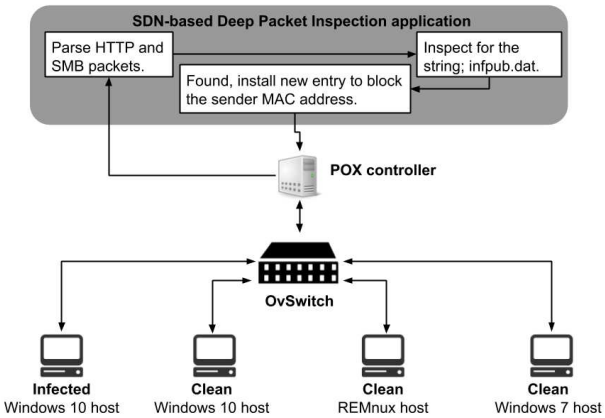

FIGURE 17. Conceptual design of DPI detection framework.

Algorithm 1 shows the mechanism for applying the DPI-based method. The following steps are performed sequentially when a new packet arrives:

1) The switch forwards packets to the controller.
2) The SMB and HTTP inspection applications at controller match the destination port with the ports used for the HTTP and SMB services, specifically ports 80, 445 and 139.
3) In the case of non-matching, the controller forwards the packet to the *forwarding.l2_learning* component, which means that the controller will not inspect the following incoming packets.
4) In the case of matching, all the subsequent incoming traffic is inspected. This is done by forwarding the packet, and then halting the event before it reaches the *forwarding.l2_learning* component.
5) The controller checks each incoming packet to search for the `infpub.dat` string.
6) In the event of matching, the sender of the packets is blocked through the installation of a new rule to the switch to prevent it from sending any packets from the source port to the clients.

B. MODULE 2: PACKET HEADER INSPECTION (PHI)

BadRabbit attempts to reach systems that have an active SMBv1 protocol. To do this, BadRabbit uses the `NT LM 0.12` dialect in SMB requests in the packets sent to other systems. SMBv1 has numerous known security vulnerabilities and therefore any attempt to use it should be considered suspicious and require attention [26], [27]. Inspecting the first packet of each SMB connection to search for the `NT LM 0.12` dialect represents an effective solution to preventing

Algorithm 1: DPI Algorithm

```
Require PacketIn event
Function DPI
packet = PacketIn.tcp
if packet is None then
    return
else if packet.dstport == 80 or 445 or 139 then
    if packet.find("infpub.dat") = True then
      Install new entry to block the sender MAC
      address from sending from the source port
    else
      Forward the packet
      PacketIn.halt
    end if
else
    return
end if
```

BadRabbit from propagating across a network and, indeed, preventing the use of this unsafe protocol within the network in general. Figure 18 gives a high-level representation of the concept.

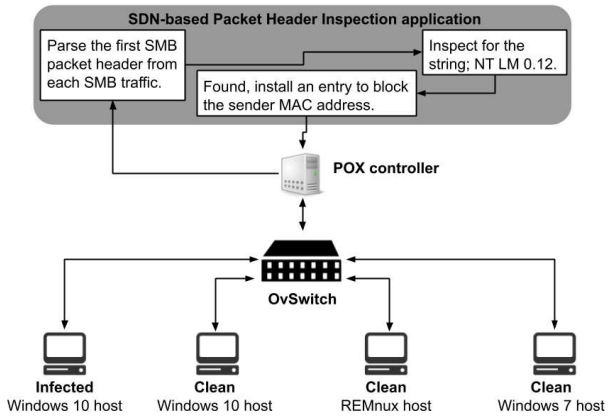

FIGURE 18. Conceptual design of the PHI detection framework.

Algorithm 2 explains the steps used to implement the PHI-based method. This approach successively performs the following steps:

- Parsing packets to determine if the packet is TCP.
- If the packet is TCP, then the ports to which the packets are sent are matched to the SMB ports, specifically ports 445 and 139.
- If the packet is directed to one of these ports, the *pack()* function is used to convert the packet into strings. Then, the *find()* function is used to search for the `NT LM 0.12` dialect.
- In case the dialect is present in the packet, *of.ofp_flow_mod* is used to direct the switch to add a new flow entry that blocks the MAC address of the packet sender from sending any packet that has the same destination port (139 or 445).
- If the dialect is not found, the packet is returned to the *forwarding.l2_learning* component. As a result, the incoming traffic will not be inspected.

Algorithm 2: PHI Algorithm

```
Require PacketIn event
Function PHI
packet = PacketIn.tcp
if packet is None then
    return
else if packet.dstport == 445 or 139 then
    if packet.find("NT LM 0.12") = True then
      Install new entry to block the sender MAC
      address from sending from the source port
    else
      return
    end if
else
    return
end if
```

C. MODULE 3: HONEYPOT-BASED

Based on the results of BadRabbit dynamic analysis, presented in Section IV, we have identified that BadRabbit first attempts to transmit packets on ports 445 and 139 to all network devices without consideration for whether they are active or inactive systems. Therefore, installing a trap system (honeypot) on the network and ensuring that it cannot, or is not, accessed by legitimate network devices is an appropriate solution for detecting suspicious activities, especially BadRabbit. For this, an application that works on the top of the controller is implemented to monitor the SMB and HTTP connections directed to the honeypot system, and that blocks systems that try to access it. Figure 19 presents the steps for the proposed approach to detect suspicious activities at the SDN network level using a honeypot.

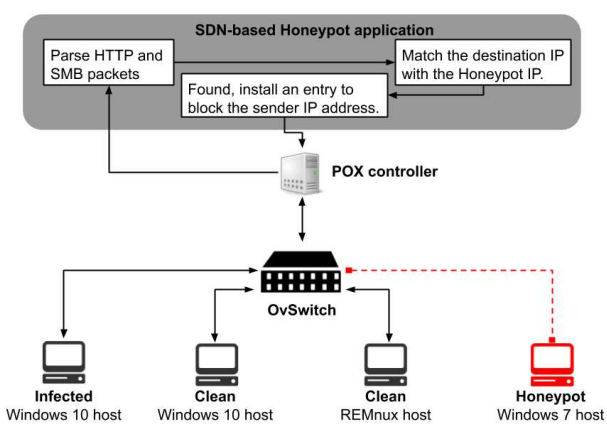

FIGURE 19. Conceptual design of Honeypot detection framework.

Algorithm 3 explains the honeypot-based detection method. The steps to be taken when a new packet is received are as follows:

1) The switch forwards the first packet of each new connection to the controller.
2) The controller inspects the packet header to identify HTTP and SMB packets.
3) If the packet is HTTP or SMB, the controller inspects the header to identify the destination IP address.
4) Then, the controller matches the packet destination IP address with the honeypot IP address.
5) If the packet is directed to the honeypot, a new rule will be created and enforced to prohibit the sender IP address from communicating within the network using the destination port or, if not, the controller will direct the packet to the switch.

Algorithm 3: Honeypot Algorithm

```
Require PacketIn event
HoneypotIP = "10.0.0.6"
Function Honeypot
packet = PacketIn.tcp
if packet is None then
    return
else if packet.dstport == 445 or 80 then
    IP = PacketIn.ipv4
    ipaddress = IP.dstip
    if ipaddress == HoneypotIP then
      Install new entry to block the sender IP
      address from communicating within the network
    else
      return
    end if
else
    return
end if
```

D. MODULE 4: ARP SCANNING-BASED DETECTION

According to Rouka *et al.* [17], detecting network scans may represent an effective solution to the detection of malware activity, and thus the detection of BadRabbit is possible using this approach. Based on the results of our analysis, BadRabbit performs a network scan using ARP. Consequently, the method presented in [17] can be used to detect BadRabbit. Figure 20 gives a high-level explanation of how this module operates.

Algorithm 4 describes the ARP scanning based detection method. The module relies on the *dict* function in Python to store two values: the originating address for ARP requests and the number of unanswered requests. The ARP library in POX can be used to determine whether the packet is ARP or otherwise, and also to specify requests and responses. This application implements the following steps:

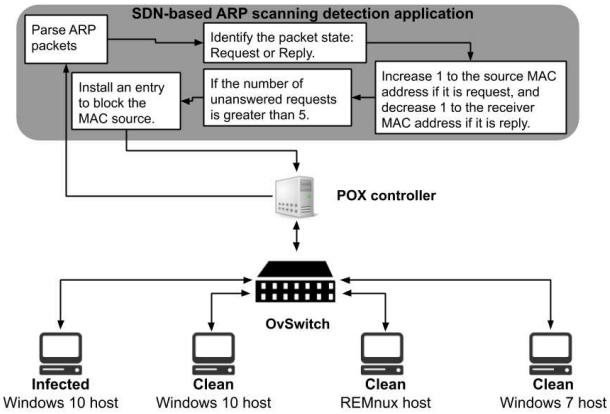

FIGURE 20. Conceptual design of ARP scanning detection framework.

1) Parses incoming packets and identifies ARP packets using the *ARP_TYPE* function.
2) If the packet is ARP, the source IP address is checked to compare it with 0.0.0.0. This step helps to decrease the possibility of false positive, as this IP address is used by new clients on the network that have yet to be assigned an IP address [28]. As a result, if the source IP address is 0.0.0.0, the packet will be forwarded to the *forwarding.l2_learning* component.
3) If the packet does not originate from 0.0.0.0, the ARP state is determined as either request or reply.
4) Increase 1 to the source MAC address if it is a request, and decrease 1 to the receiver MAC address if it is a reply.
5) If the number of unanswered requests is greater than a predefined threshold (say 5), a new rule is installed to block the source MAC address from communicating within the network for a predefined duration (e.g., 20 minutes).

E. MODULE 5: SMB PACKET SIZE CHECKER

According to the results of our BadRabbit analysis, the system infected with BadRabbit will exchange with the benign system three consecutive SMB packets of a fixed and unique size. Thus, a BadRabbit-detecting application can be designed based on these characteristics. It is worth noting that traffic characteristics have also been used by other works to detect ransomware in SDN networks [14], [15]. These solutions, however, analyse HTTP and HTTPS traffic rather than the SMB traffic. Figure 21 gives a high-level explanation of how this approach works, while Figure 22 shows the process of exchanging the first three packets between the infected system and the benign system.

Algorithm 5 illustrates the SMB packet size checker application.. This application extracts the size of each incoming SMB packet to port 445 to compare it with the three aforementioned values and then stores the sender MAC address and the number of suspicious SMB packets in a dictionary. It then blocks the sender MAC address in the event that the number of suspicious sent packets exceed a certain threshold

Algorithm 4: ARP Scanning-Based Detection Algorithm

```
Require PacketIn event
Values = dict(MAC,No)
Function ARP-Scanning-Detection
packet = PacketIn.ARP_TYPE
if packet is None then
    return
if packet.payload.protosrc == IPAddr("0.0.0.0") then
    return
if packet.payload.opcode == arp.REQUEST then
    Add the MAC source to the dictionary
      Values and increase one to the No field
    if No > 5 then
      Install new entry to block the sender MAC
      address from communicating within the network
    else
      return
if packet.payload.opcode == arp.REPLY then
    Get the MAC address and decrease one to the No
  field
else
    return
end if
```

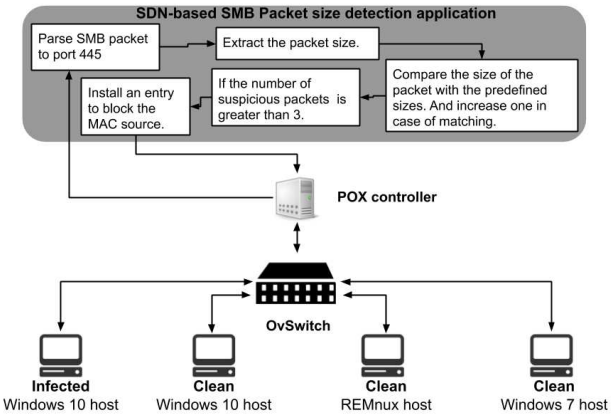

FIGURE 21. Conceptual design of the SMB packet size detection framework.

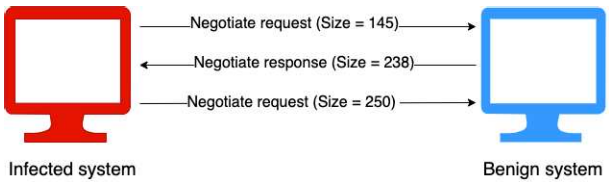

FIGURE 22. BadRabbit SMB negotiation.

(e.g. three packets). This approach applies the following steps to address suspicious SMB traffic:

1) Parse incoming SMB packets to port 445.
2) Extract the size of each SMB packet using the *len* function.
3) Compare the size of the packet with predefined sizes (145, 238, 250).
4) In case of a match, the MAC address of the source and a value of 1 are stored in a directory, so that the value is increased by one for each matching case.
5) If the number of suspicious packets exceeds the threshold (e.g., three packets), the MAC address of the source is blocked through installing a rule that instructs the switches to block all incoming connections from the blocked source for a predefined duration (e.g., 20 minutes).

Algorithm 5: SMB Packet Size Checker Algorithm

```
Require PacketIn event
Suspicious_size = dict(MAC,No)
Function SizeChecker
packet = PacketIn.tcp
if packet is None then
    return
else if packet.dstport == 445 then
    PacketSize = len(packet)
    if PacketSize == 145 or 238 or 250 then
      Add the MAC source to the dictionary
  Suspicious_size
        and increase one to the No field
      if No > 3 then
        Install new entry to block the associated MAC
  address from
        sending packets from the detected source port
      else
        return
    else
      Forward the packet
      PacketIn.halt
    end if
else
    return
end if
```

VI. RANSOMWARE DETECTION SYSTEM: IMPLEMENTATION

The IDPS has been implemented on top of the POX controller. The implementation of the five modules relies on a PacketIn event. It is created when a new packet arrives at the switch and does not match any of the entries in the forwarding tables, or there is an entry in the table that includes a procedure that specifies the packet transmission to the controller [29]. Consequently, when a packet reaches the switch and has no match, it will be directed to the controller. Moreover, the *launch function* was used to initialize the application, as this is required for the applications to function properly [29]. Our developed code can be found on GitHub [30].

To implement the IDPS modules, an SDN testbed has been configured and used. This testbed includes Ubuntu 16.04.6 LTS as a physical machine that hosts five other systems through the use of the VirtualBox virtualization

environment. The Ubuntu machine also acts as an Open vSwitch, while the VMs work as follows: REMnux, as a fake DNS and HTTP server using the INetSim tool, three Windows systems, two of which run Windows 10 and one runs Windows 7, and where one of the Windows 10 acts as an infected system, while the other two systems are benign. Furthermore, the Ubuntu machine has been utilized to work as a POX controller. Figure 23 details the testbed topology and configuration.

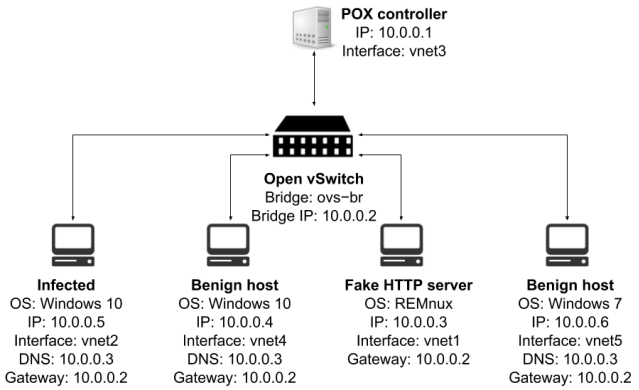

FIGURE 23. SDN testbed topology and configuration.

A. MODULE 1: DPI

To implement this module (Algorithm 1), the *libopenflow_01.py* file in the POX/Openflow folder must be modified (the modification will be made to the POX utility in the controller system). This file has a variable that restricts the number of bytes transferred to the controller. Hence, the attribute *miss_send_len in libopenflow_01.py* file has been modified from 128 to 1600. Whilst the command *./pox.py misc.full_payload forwarding.l2_learning DPI* is sufficient to run the application, it is possible to add new components to enable appropriate notifications. The full command for this is as follows: *$./pox.py misc.full_payload forwarding.l2_learning DPI samples.pretty_log log.level —DEBUG info.packet_dump*

The *misc.full_payload* component is also invoked to send all the traffic to the controller without any restriction to the size of the transmission [29].

B. MODULE 2: PHI

This module (Algorithm 2) does not require access to all the network traffic or to the data section in the packet; it suffices to check the header of the first packet of each new traffic only. There is no need to modify the POX utility or use the *misc.full_payload* component. Thus, the following command can be used to run the application: *$./pox.py forwarding.l2_learning PHI samples.pretty_log log.level —DEBUG info.packet_dump*.

C. MODULE 3: HONEYPOT

The honeypot-based approach (Algorithm 3) does not require full access to the network traffic or to the full packet information as is the case of the DPI based method. It only needs to inspect the header, which is included in the first 128 bytes of each packet. Therefore, there is no need to invoke special components such as *misc.full_payload* or modify the POX utility. Therefore, the following command is sufficient to run the application: *$./pox.py forwarding.l2_learning Honeypot samples.pretty_log log.level —DEBUG info.packet_dump*.

D. MODULE 4: ARP SCANNING

The ARP scanning detection application (Algorithm 4) does not need full access to all traffic. It only needs a header from each ARP packet. As a result, there is no need to implement modifications to the switch or controller. Therefore, the following command is sufficient to run the application correctly: *$./pox.py forwarding.l2_learning ARP-Detection samples.pretty_log log.level —DEBUG info.packet_dump*.

E. MODULE 5: SMB CHECKER

The SMB packer size checker application needs access to all traffic and therefore there is a need to amend the POX application as previously explained in the case of the DPI based method. There is also a need to use the *misc.full_payload* component to forward all traffic to the controller, and thus the following command has to be used to run the application correctly: *$./pox.py misc.full_payload forwarding.l2_learning SizeChecker samples.pretty_log log.level —DEBUG info.packet_dump*.

VII. SYSTEM VALIDATION

In this section, the experimental results of the five proposed methods/modules are presented. Our aim is to validate the effectiveness of each method in detecting self-propagating ransomware, focusing on BadRabbit. Each module was enabled separately on the controller and the ransomware was executed on the infected system. To obtain accurate results, reliance was placed on traffic inspection using Wireshark, monitoring the controller's graphical interface to track alerts, and checking the Windows folder on the benign systems to determine whether BadRabbit had managed to transfer the suspicious files to these systems or not.

A. DPI RESULTS

To evaluate the efficacy of the solution, the same authentication information were used on the infected system and on a clean Windows 10 system with IP address 10.0.0.4. In addition, an HTTP emulator (InetSim) is run on the REMnux system to ensure that the HTTP traffic is inspected. BadRabbit was executed on the infected system and the *infpub* string was detected in the SMB traffic after 7 minutes 14 seconds (Figure 24), whereas the string was detected in the HTTP traffic after 7 minutes 27 seconds (Figure 25). Figures 26 and 27 show the packets responsible for triggering this detection, while Figure 28 shows the attempts by the infected system to connect to port 445 after the ban. Although the suspicious file name was detected, the file was created on

```
<-> BadRabbit attempt to self-propgate !!!. At the time: ', '2020-06-30

flow has been installed for 10.0.0.5 with destination port 445
Blocked SMB suspicious packet from port 50608 to port 445
```

FIGURE 24. Detection of the *infpub* string in the SMB traffic.

```
flow has been installed for 10.0.0.5 with destination port 80
Blocked Suspicious packet from port 50846 to port 80
```

FIGURE 25. Detection of the *infpub* string in the HTTP traffic.

10.0.0.5	SMB2	Close Response
10.0.0.4	SMB2	Create Request File: infpub.dat
10.0.0.4	TCP	[TCP Retransmission] 50608 → 445 [PSH, ACK] S
10.0.0.5	SMB2	Create Response File: infpub.dat
10.0.0.4	TCP	50608 → 445 [ACK] Seq=2039 Ack=2683 Win=26188
10.0.0.4	TCP	[TCP Retransmission] 50608 → 445 [ACK] Seq=20
10.0.0.5	TCP	445 → 50608 [ACK] Seq=2683 Ack=3499 Win=26265
10.0.0.4	TCP	[TCP Previous segment not captured] 50608 → 4
10.0.0.4	TCP	[TCP Retransmission] 50608 → 445 [ACK] Seq=18
10.0.0.3	TCP	[TCP Retransmission] 50606 → 445 [SYN] Seq=0
10.0.0.3	TCP	[TCP Retransmission] 50606 → 445 [SYN] Seq=0
10.0.0.2	TCP	[TCP Retransmission] 50607 → 445 [SYN] Seq=0
10.0.0.2	TCP	[TCP Retransmission] 50607 → 445 [SYN] Seq=0
10.0.0.4	TCP	[TCP Retransmission] 50608 → 445 [ACK] Seq=34
10.0.0.4	TCP	[TCP Retransmission] 50608 → 445 [ACK] Seq=34
10.0.0.5	TCP	[TCP Dup ACK 381#1] 445 → 50608 [ACK] Seq=268
10.0.0.5	TCP	445 → 50607 [RST, ACK] Seq=1 Ack=1 Win=0 Len=

FIGURE 26. A capture of traffic showing the SMB packet being detected.

10.0.0.3	HTTP	PROPFIND /admin%24/infpub.dat HTTP/1.1
10.0.0.3	TCP	[TCP Retransmission] 50539 → 80 [PSH, ACK
10.0.0.5	TCP	80 → 50539 [ACK] Seq=1 Ack=177 Win=30336
10.0.0.5	TCP	80 → 50539 [PSH, ACK] Seq=1 Ack=177 Win=3
10.0.0.3	TCP	50539 → 80 [FIN, ACK] Seq=177 Ack=183 Win
10.0.0.3	TCP	[TCP Out-Of-Order] 50539 → 80 [FIN, ACK]

FIGURE 27. A capture of traffic showing the HTTP packet being detected and the blocking of this traffic.

10.0.0.4	TCP	50608 → 445 [ACK] Seq=22479 Ack=2683 Win=26188
10.0.0.4	TCP	[TCP Retransmission] 50608 → 445 [ACK] Seq=224
10.0.0.5	TCP	[TCP Dup ACK 395#1] 445 → 50608 [ACK] Seq=2683
10.0.0.4	TCP	[TCP Out-Of-Order] 50608 → 445 [ACK] Seq=4959
10.0.0.4	TCP	[TCP Out-Of-Order] 50608 → 445 [ACK] Seq=4959
10.0.0.4	TCP	[TCP Out-Of-Order] 50608 → 445 [ACK] Seq=6419
10.0.0.4	TCP	[TCP Out-Of-Order] 50608 → 445 [ACK] Seq=6419
10.0.0.4	TCP	[TCP Out-Of-Order] 50608 → 445 [ACK] Seq=7879
10.0.0.4	TCP	[TCP Out-Of-Order] 50608 → 445 [ACK] Seq=7879

FIGURE 28. A capture of the traffic showing the infected system being blocked.

the clean system, but the data was not transferred to it as a result of the ban (Figure 29).

B. PHI RESULTS

BadRabbit was executed on the infected system, while the PHI application was running on the top of the controller. After five seconds, an alert has raised showing an attempt to use SMBv1 on port 445 by the infected system (Figure 30). Consequently, the infected system was blocked from using port 445 as Figure 31 shows. Thirty seconds later, another alarm was issued showing that the infected system had been banned from using port 139 as it sent a packet that contained the suspicious dialect (Figure 32). The packet responsible for this alert is shown in Figure 33.

C. HONEYPOT RESULTS

BadRabbit was executed on the infected system, and the honeypot application was invoked on the top of the controller. Eight seconds later, the honeypot application detected an attempt to communicate with the honeypot itself on port 445, where this attempt was issued from the infected system (Figure 34). One minute and six seconds from the time at which it was run, the application also detected an attempt to reach port 80 on the honeypot. This attempt was also issued from the infected system, as shown in Figure 35. Packets responsible for triggering this detection are shown in Figures 36 and 37. These attempts led to the prohibition of the infected system from communicating with network devices on ports 445 and 80, as shown in Figure 38. The infected system tried to communicate with the HTTP simulator (REMnux system) post-detection, but the connection was not allowed due to its prohibition already being in place.

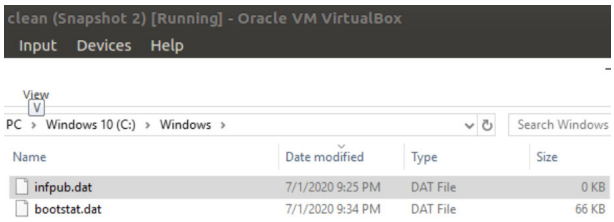

FIGURE 29. A screenshot of the clean system showing the *infpub* file as empty.

```
found ! <-> attmept to use SMB version 1 !!!. At the time: ', '2020-06-30

  ] flow has been installed for 10.0.0.5 with destination port 445
  ] Blocked SMBv1 packet from port 50576 to port 445
```

FIGURE 30. Detection of an attempt to use SMBv1 on port 445.

10.0.0.5	10.0.0.6	SMB	Negotiate Protocol Request
10.0.0.5	10.0.0.6	TCP	[TCP Retransmission] 50576 → 445
10.0.0.6	10.0.0.5	SMB2	Negotiate Protocol Response
10.0.0.5	10.0.0.6	SMB2	Negotiate Protocol Request
10.0.0.5	10.0.0.6	TCP	[TCP Retransmission] 50576 → 445
10.0.0.6	10.0.0.5	SMB2	Negotiate Protocol Response
10.0.0.5	10.0.0.6	SMB2	Session Setup Request, NTLMSSP_NE
10.0.0.5	10.0.0.6	TCP	[TCP Retransmission] 50576 → 445
10.0.0.6	10.0.0.5	SMB2	Session Setup Response, Error: ST
10.0.0.5	10.0.0.6	SMB2	Session Setup Request, NTLMSSP_AU
10.0.0.5	10.0.0.6	TCP	[TCP Retransmission] 50576 → 445
10.0.0.6	10.0.0.5	SMB2	Session Setup Response, Error: ST
10.0.0.5	10.0.0.6	TCP	50576 → 445 [RST, ACK] Seq=993 Ac
10.0.0.5	10.0.0.6	TCP	50576 → 445 [RST, ACK] Seq=993 Ac
10.0.0.5	10.0.0.6	TCP	50578 → 445 [SYN] Seq=0 Win=64240
10.0.0.5	10.0.0.6	TCP	[TCP Out-Of-Order] 50578 → 445 [S
10.0.0.5	10.0.0.6	TCP	[TCP Retransmission] 50578 → 445

```
    Byte Count (BCC): 34
  ▼ Requested Dialects
    ▼ Dialect: NT LM 0.12
```

FIGURE 31. The SMBv1 packet that triggered the detection on port 445.

```
flow has been installed for 10.0.0.5 with destination port 139
Blocked SMBv1 packet from port 50571 to port 139
```

FIGURE 32. Detection of an attempt to use SMBv1 on port 139.

10.0.0.5	10.0.0.6	SMB	Negotiate Protocol Request
10.0.0.5	10.0.0.6	TCP	[TCP Retransmission] 50548 → 139
10.0.0.6	10.0.0.5	SMB2	Negotiate Protocol Response
10.0.0.5	10.0.0.6	SMB2	Negotiate Protocol Request
10.0.0.5	10.0.0.6	TCP	[TCP Retransmission] 50548 → 139
10.0.0.6	10.0.0.5	SMB2	Negotiate Protocol Response
10.0.0.5	10.0.0.6	TCP	50548 → 139 [FIN, ACK] Seq=324 A
10.0.0.5	10.0.0.6	TCP	[TCP Out-Of-Order] 50548 → 139 [

FIGURE 33. The SMBv1 packet that triggered the detection on port 139.

```
[blocker                ] installing flow for 10.0.0.5 w
[blocker                ] Blocked suspicious HTTP or SMB
 BadRabbit self-propogation attempt
('Detection time is: ', '2020-07-02 01:07:17.761512')
A SMB request to the HoneyPot
```

FIGURE 34. Detection of SMB packet directed to the honeypot.

```
[blocker                ] installing flow for 10.0.0.5 w
[blocker                ] Blocked suspicious HTTP or SMB
BadRabbit self-propogation attempt
('Detection time is: ', '2020-07-02 01:08:05.176376')
A HTTP request to the HoneyPot
```

FIGURE 35. Detection of HTTP packet directed to the honeypot.

```
10.0.0.6        TCP     50790 → 445 [SYN] Seq=0 Win=64240 Le
10.0.0.6        TCP     [TCP Out-Of-Order] 50790 → 445 [SYN]
```

FIGURE 36. The SMB packet that triggered the detection.

```
10.0.0.6        TCP     50807 → 80 [SYN] Seq=0 Win=64240 Le
10.0.0.6        TCP     [TCP Out-Of-Order] 50807 → 80 [SYN]
```

FIGURE 37. The HTTP packet that triggered the detection.

```
10.0.0.3        TCP     50858 → 80 [SYN] Seq=0 Win=64240 Le
10.0.0.3        TCP     [TCP Out-Of-Order] 50858 → 80 [SYN]
Broadcast       ARP     Who has 10.0.0.34? Tell 10.0.0.5
```

FIGURE 38. Blocking the infected system's connections to the port 80.

D. ARP SCANNING-BASED DETECTION RESULTS

BadRabbit was run and 5 minutes 1 second later, two systems were detected and banned as they attempted to make more than five unanswered requests (Figures 39 and 40). After analysing the traffic and looking at the clean system that has the IP address 10.0.0.4, it was found that the application did not detect BadRabbit sufficiently quickly, allowing time for the ransomware to transfer its files to the clean system (Figures 41 and 42). This is be because BadRabbit does not perform a large ARP scan except after exhausting its attempts to spread to systems that it manages to reach without a scanning. Therefore, this application cannot be used against BadRabbit due to the detection speed being insufficient, whereby the risk of BadRabbit being able to spread has not been greatly reduced.

```
[ARP Requests monitor   ] Blocked host with IP 10.0.0.5 on port 3
('Detection time is : ', '2020-07-17 00:35:22.582020')
[dump:82-98-1a-00-cf-49 ] [ethernet][arp]
10.0.0.4 has performed 6 unanswered ARP reuqests.
[ARP Requests monitor   ] Blocked host with IP 10.0.0.4 on port 4
```

FIGURE 39. Detection of two systems that performed more than five unanswered ARP requests.

```
 duration=41.503s, table=0, n_packets=55, n_bytes=2478, idle_timeout
timeout=1800, idle_age=0, dl_src=08:00:27:e6:e5:59 actions=drop
 duration=41.418s, table=0, n_packets=31, n_bytes=1302, idle_timeout
timeout=1800, idle_age=1, dl_src=08:00:27:af:af:88 actions=drop
```

FIGURE 40. Table entries showing the two systems being blocked.

E. SMB PACKET SIZE CHECKER RESULTS

BadRabbit was executed on the system and 21 seconds later the infected system was detected, as shown in Figure 43. The detection was sufficiently fast that the infected system was blocked before it proceeded to its subsequent self-propagation steps. As a result, the risk of BadRabbit was contained and no successful self-propagation occurred.

```
10.0.0.5        SMB2    Close Response
10.0.0.4        SMB2    Create Request File: infpub.dat
10.0.0.4        TCP     [TCP Retransmission] 51122 → 445
10.0.0.5        SMB2    Create Response File: infpub.dat
10.0.0.4        TCP     51122 → 445 [ACK] Seq=2039 Ack=26
```

FIGURE 41. A part of the suspicious file transmission to the benign system.

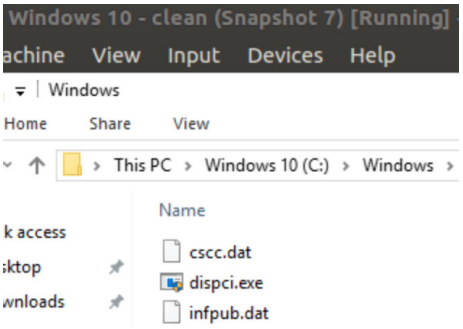

FIGURE 42. BadRabbit files on the benign system.

```
SMB Monitor             ] Blocked host with MAC address
port 3: Sending three or more suspicious SMB packets
'Detection time is : ', '2020-07-23 10:07:09.826123')
```

FIGURE 43. Blocking a system responsible for sending three suspicious SMB packets.

VIII. PERFORMANCE EVALUATION AND COMPARISON

In this section, we present our performance evaluation results for the five proposed detection methods. Two modules rely on methods proposed in [16], [17]. Specifically, DPI to look for specific values inside the packet, and monitoring ARP scanning. Three other modules rely on novel methods in SDN-based ransomware detection. The performance is evaluated based on five criteria: the time taken to detect BadRabbit, CPU utilization rate, Ping latency, TCP latency, and the capability of the solutions to detect other types of ransomware. Measuring CPU usage and delay for both Ping and TCP was achieved by running BadRabbit, the proposed solution, and the evaluation mechanism for ten minutes. To measure the CPU usage, the script reported in [31] was used. To measure the Ping delay, the *ping* command available on most systems like Windows and Linux was used. Finally, to measure the TCP delay the *hping* utility was used. Furthermore, in this section, the capability of the suggested solutions to detect other types of ransomware has been examined. To this end, two modules (PHI and honeypot) were implemented and tested against the NotPetya ransomware.

A. DETECTION TIME AND CPU UTILISATION

The BadRabbit detection times and the CPU utilization rates were measured. Table 5 shows the detection time for each module in addition to the average CPU utilization rate. The latter is also depicted in Figure 44 to better illustrate the

TABLE 5. Detection time and CPU utilisation of the proposed methods.

Detection approach	Detection duration time	CPU usage
DPI	7 minutes 14 seconds for SMB 7 minutes 27 seconds for HTTP	16.971%
PHI	5 seconds	7.448%
Honeypot	8 seconds for SMB, 1 minute 6 seconds for HTTP	6.934%
ARP-scanning based	5 minutes 1 second	11.579%
SMB Packets Size	21 seconds	7.712%

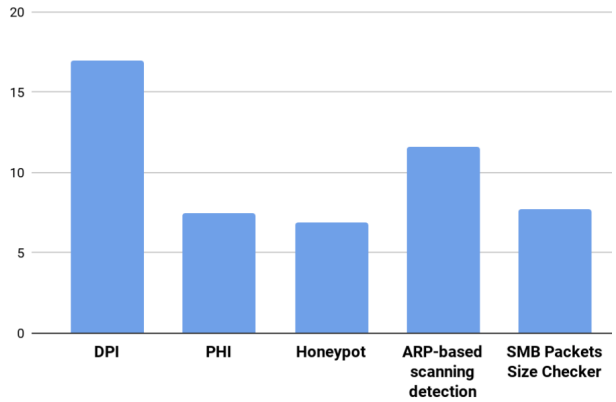

FIGURE 44. Average CPU utilisation (%).

differences. The interpretation of these results is as follows. The PHI solution achieved the fastest detection time, as it detected the attempt to access via SMBv1 in only five seconds, while the honeypot detected an attempt to access SMB in eight seconds. However, the attempt to access via HTTP was detected after 1 minute 6 seconds, while the SMB Packet Size Checker achieved BadRabbit detection after only 21 seconds. It is worth noting that the PHI solution depends only on inspection of each packet header destined to port 445 or 135, while the honeypot depends only on detecting attempts to access a service on a specific device on the network, while the SMB Packet Size solution takes more time to process SMB traffic to compare it with three predefined sizes. Moreover, the CPU usage for the three solutions was the lowest of the available solutions as CPU usage was 7.712% for SMB Packets Size, 7.448% for the PHI, whilst the honeypot had the lowest CPU usage at 6.934%. The other two solutions achieved detecting times between 5 and 7 minutes, where the ARP scanning achieved detection in 5 minutes 1 second using 11.579% of the CPU. DPI achieved detection times of 7 minutes 14 seconds for SMB, and 7 minutes 27 seconds to detect the string in the HTTP traffic with a relatively high CPU usage of 16.971%, which may be due to the processing of all traffic directed at SMB and HTTP, which would quite naturally require quite high CPU usage.

B. TCP AND PING DELAY

Our measurements of the TCP and Ping traffic delays gave distinct results for each of the proposed methods. Tables 6 and 7 show the highest delay, minimum delay and average delay in addition to the number of packets that

TABLE 6. Ping delay for the proposed methods.

	Ping latency (ms)			Packets
Detection method	Min	Max	Avg	
DPI	0.144	55.59	1.63	598
PHI	0.153	60.318	1.66	601
Honeypot	0.108	92.496	1.84	603
ARP-scanning based	0.164	55.374	2.39	601
SMB Packets Size	0.147	51.336	1.33	601

TABLE 7. Ping and TCP delay for the proposed methods.

	TCP latency (ms)			Packets
Detection method	Min	Max	Avg	
DPI	0.7	1093.8	77.3	598
PHI	6.8	97.8	34.9	601
Honeypot	5.8	76.8	37.1	603
ARP-scanning based	5.0	85.5	35.0	601
SMB Packets Size	6.5	103.4	35.2	601

were processed in a ten-minute period. Figure 45 illustrates the differences in the TCP and Ping delay of the proposed methods. In regards to TCP delays, three solutions achieved similar results of the average delay and the same number of processing packets; PHI, ARP scanning, and SMB Packets Size Checker processed 601 packets in ten minutes. The average delay was 34.9 ms for PHI, 35.0 ms for ARP scanning, and 35.2 ms for SMB Packets Size Checker. The difference was not significant between these three solutions and the honeypot; the latter achieved an average delay of 37.1 ms and handled more packets, at 603. These four methods depend on fewer operations compared to the DPI method, which handles all HTTP and SMB traffic to search for a specific string. This leads to a high delay rate of 77.3 ms and processing of fewer packets, at 598. In ping delay measurement, ARP scanning had the highest delay, at an average of 2.39 ms, due to the fact that this solution relies on the analysis of all ARP requests and replies. DPI and PHI achieved similar average delays, at 1.63 and 1.66 ms, respectively. By contrast, the honeypot achieved an average delay of 1.84 ms, while the SMB Packets Size Checker achieved the lowest delay among other solutions, with an average of 1.33 ms.

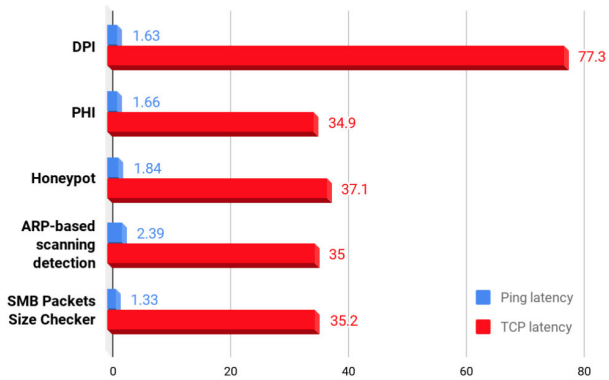

FIGURE 45. Ping and TCP latency (ms).

Despite the competence that DPI provided in monitoring, its impact was significant both in terms of CPU usage and

network delays. This is due to the amount of data processed by the controller. Other solutions were found to have less impact on the network compared to DPI.

C. DETECTING OTHER RANSOMWARE FAMILIES

Two of the proposed methods (PHI and honeypot) may provide solutions to other threats on the network, particularly ransomware. This is because some types of ransomware rely on two particular methods in an attempt to propagate across the network. The first method is to try to access the SMBv1 protocol, because it contains security flaws which could facilitate the spread of malware. The second method is an attempt to access specific ports on all systems on the network, mostly ports 80 and 445. Thus, PHI can be applied to detect the first method, and the honeypot can be used to detect the second method. To demonstrate this, PHI and honeypot have been implemented against NotPetya. The analysis of NotPetya in [32], [33] showed that it uses SMBv1 as well as trying to access all the systems in the network on ports 80 and 445. As a result, PHI and honeypot were found to be successful in both detecting and blocking the system infected with NotPetya before it was able to successfully spread itself to other systems on the network. Figure 46 shows the detection of the infected device using PHI, while Figures 47 and 48 show the detection of the infected device's attempts to access the honeypot on ports 445 and 80.

FIGURE 46. Detection of NotPetya's attempt to access SMBv1.

FIGURE 47. Detection of NotPetya attempting to access port 445 on the honeypot.

D. DETECTING POLICY CHANGES

Another important issue is related to detecting policy changes in an SDN environment. That is, when installing new Flow Table or firewall rules, or when defining new Access Control List (ACL) policies, conflicts may arise due to already installed rules [34].

Generally speaking, there is a possibility that ransomware may be granted access, even despite being blocked by the controller. The cause could be that there were old rules that allowed this access, before the new rule was installed trying to block a connection. As an example, consider the following scenario. A device with an IP address 10.0.0.1 is allowed to access a webserver through port 80. After that, assume that

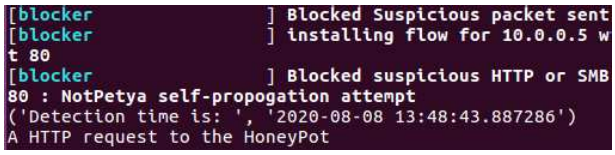

FIGURE 48. Detection of NotPetya attempting to access port 80 on the honeypot.

the device has been infected with ransomware. The detection mechanism then installs a new rule, that prevents the infected device accessing other devices or services in the network through the port 80. As a result, there are two rules, one of which allows access and the other that denies it, which constitutes a conflict. This may cause the connection to be allowed or denied, depending on the rule priorities. Similar violations may occur due to multiple conflicting ACL policies.

Referring specifically to our proposed approach, some of the modules, such as DPI, PHI, and SMB based, halt the suspicious packet, thus stopping the ransomware from infecting other devices. In other words, even if there is a conflict, the infected device will not be able to infect any other device, even if it can access it (assuming that there is a conflict). Two other modules, namely Honeypot and ARP-Scanning-based, could indeed cause a conflict which leads to violating the installed rule and spreading ransomware. This is because these two approaches depend entirely on monitoring a certain number of legitimate packets (the number of ARP requests, and the attempt to reach the webserver) to impose restrictions on the infected device. Resolving such conflicts is beyond the scope of this article and the interested reader may refer relevant works [34]–[36].

In summary, the proposed methods achieved different results when evaluated, with DPI showing the weakest performance in terms of delay and CPU load. It resulted in a delay in TCP traffic that was twice as long as the other solutions and also used approximately 17% of the CPU. By contrast, other methods resulted in less pressure on the CPU capacity and less delay in TCP traffic. Moreover, two methods achieved successful detection of another type of ransomware family as PHI and honeypot detected NotPetya's attempts to self-propagate within the network.

IX. CONCLUSION

After performing a thorough analysis of BadRabbit, it was found that this ransomware family does not need to communicate with external entities (e.g., C&C servers) to exchange an encryption key. Instead, it uses a public key integrated into its files. Therefore, methods used in [13], [14] would not be suitable for this type of ransomware. As a result, the focus of this project was on blocking BadRabbit's attempts at self-propagation. We have implemented an SDN-based IDPS which consists of five modules to detect and block self-propagating ransomware, such as BadRabbit. Two modules rely on methods proposed in [17]. Specifically, deep packet inspection to look for specific values, and monitoring ARP scanning. Three other modules rely on

novel methods in SDN-based ransomware detection. They include inspecting the packet header to block SMBv1 access attempts, an SMB packet size checker, and finally using a honeypot in the network to detect any attempts to access port 80 or 445 of the honeypot system.

These methods have been evaluated based on TCP and ping delays, CPU utilization, detection duration, and the capability to detect other types of ransomware, such as NotPetya. It was found that traffic inspection resulted in a greater delay than any of the other methods considering TCP traffic, while the ping approach was causing the least delay. The traffic inspection module utilized nearly 17% of the CPU, which is double that of any other method. Furthermore, two of the modules (packet header inspection and honeypot) are able to detect other types of ransomware. To demonstrate this, they have been implemented against NotPetya and were successful at both detection and timely blocking.

In our future work, we plan to test the performance and the efficiency of the IDPS in a live network. For validation purposes, the presence of different applications and realistic background traffic will be considered, as well as the operation of other security appliances and functions. We also plan to investigate any conflicts that might be caused by conflicting security policies, when defending against different types of threats.

APPENDIX A

The extensions of the files that are encrypted by BadRabbit are:

```
.3ds,.7z,.accdb,.ai,.asm,.asp,
.avhd,.back,.bak,.bmp,.brw,.c,
.cc,.cer,.cfg,.conf,.cpp,.crt,
.ctl,.cxx,.dbf,.der,.dib,.disk,
.doc,.docx,.dwg,.eml,.fdb,.gz,
.hdd,.hpp,.hxx,.iso,.java,.jfif,
.jpe,.jpeg,.jpg,.js,.kdbx,.key,
.mdb,.msg,.nrg,.odc,.odf,.odg,
.odm,.odp,.ods,.odt,.ora,.ost,
.ovf,.p12,.p7b,.p7c,.pdf,.pem,
.php,.pmf,.png,.ppt,.pptx,.ps1,
.pvi,.py,.pyw,.qcow,.qcow2,.rar,
.rtf,.scm,.sln,.sql,.tar,.tib,
.tiff,.vb,.vbox,.vbs,.vcb,.vdi,
.vhd,.vhdx,.vmc,.vmdk,.vmsd,.vmtm,
.vmx,.vsdx,.vsv,.work,.xls,.xlsx,
.xvd,.zip,.pyc..aspx,.cab,.cs,
.djvu,.h,.mail,.odi,.xml,.vfd,
.tif,.pst,.pfx,.ova,.rb
```

APPENDIX B

BadRabbit's author's public key (RSA-2048) is:


```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEA5clDuVFr5sQxZ+feQlVvZcEK0k4u
CSF5SkOkF9A3tR6O/xAt89/PVhowvu2TfBTR
snBs83hcFH8hjG2V5F5DxXFoSxpTqVsR4lOm
5KB2S8ap4TinG/GN/SVNBFwllpRhV/vRWNmK
gKIdROvkHxyALuJyUuCZlIoaJ5tB0YkATEHE
yRsLcntZYsdwH1P+NmXiNg2MH5lZ9bEOk7YT
MfwVKNqtHaX0LJOyAkx4NR0DPOFLDQONW9OO
hZSkRx3V7PC3Q29HHhyiKVCPJsOW1l1mNtwL
7KX+7kfNe0CefByEWfSBt1tbkvjdeP2xBnPj
b3GE1GA/oGcGjrXc6wV8WKsfYQIDAQAB
```


APPENDIX C

Table 8 reports the built-in usernames used by the worm.

TABLE 8. BadRabbit built-in usernames to launch a dictionary attack.

Administrator	Admin	User	netguest	buh	boss
rdpadmin	Guest	root	manager	ftp	backup
otheruser	User1	Test	support	rdp	user-1
ftpadmin	rdpuser	work	operator	nas	ftpuser
nasadmin	nasuser	alex	superuser	asus	

APPENDIX D

Integrated passwords: Administrator, administrator, Guest, guest, User, user, Admin, admin, Test, test, root, 123, 1234, 12345, 123456, 1234567, 12345678, 123456789, 1234567890, Administrator123, administrator123, Guest123, guest123, User123, user123, Admin123, admin123, Test123, test123, password, 111111, 55555, 77777, 777, qwe, qwe123, qwe321, qwer, qwert, qwerty, qwerty123, zxc, zxc123, zxc321, zxcv, uiop, 123321, 321, love, secret, sex, god.

APPENDIX E

Figures 49 and 50 demonstrate the use of Frag and Free strings, respectively.

```
text:10002353                 mov     edx, [edx+edi]
text:10002353 ; ---------------------------------------
text:10002356                 db 81h
text:10002357                 db 0F2h ; ò
text:10002358                 db  46h ; F
text:10002359                 db  72h ; r
text:1000235A                 db  61h ; a
text:1000235B                 db  67h ; g
text:1000235C ; ---------------------------------------
text:1000235C                 jz      short loc_1000239E
text:1000235E                 inc     ecx
text:1000235F                 movzx   edx, cx
text:10002362                 cmp     edx, eax
text:10002364                 jle     short loc_10002353
text:10002366
```

FIGURE 49. The use of the Frag string.

```
loc_1000239E:                                   ; CODE XREF:
                movzx   ecx, cx
                add     ecx, edi
                mov     eax, [ecx+8]
; ---------------------------------------------------------
                db 0BFh
                db  46h ; F
                db  72h ; r
                db  65h ; e
                db  65h ; e
```

FIGURE 50. The use of the Free string.

REFERENCES

[1] *Industrial Control Systems Emergency Response Team (ICS-CERT)*, Malware Trends, Department of Homeland Security, Washington, DC, USA, Oct. 2016.

[2] M. Conti, A. Gangwal, and S. Ruj, "On the economic significance of ransomware campaigns: A Bitcoin transactions perspective," *Comput. Secur.*, vol. 79, pp. 162–189, Nov. 2018, doi: 10.1016/j.cose.2018.08.008.

[3] *Internet Organised Crimethreat Assessment (IOCTA)*, European Cybercrime Centre, The Hague, Netherlands, 2019.

[4] Kaspersky. (Mar. 2017). *The Biggest Ransomware Threats of 2017*. [Online]. Available: https://www.kaspersky.com/resource-center/threats/biggest-ransomware-threats-2017

[5] Q. Chen and R. A. Bridges, "Automated behavioral analysis of malware: A case study of wannacry ransomware," in *16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Cancun, Mexico, Jan. 2017, pp. 454–460, doi: 10.1109/ICMLA.2017.0-119.

[6] D.-Y. Kao and S.-C. Hsiao, "The dynamic analysis of WannaCry ransomware," in *Proc. 20th Int. Conf. Adv. Commun. Technol. (ICACT)*, Seoul, South Korea, Feb. 2018, p. 159, doi: 10.23919/ICACT.2018.8323682.

[7] M. Akbanov, V. G. Vassilakis, and M. D. Logothetis, "WannaCry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms," *J. Telecommun. Inf. Technol.*, vol. 1, pp. 113–124, Apr. 2019, doi: 10.26636/jtit.2019.130218.

[8] M. Akbanov, V. G. Vassilakis, I. D. Moscholios, and M. D. Logothetis, "Static and dynamic analysis of WannaCry ransomware," in *Proc. IEICE Inf. Commun. Technol. Forum*, Graz, Austria, Jul. 2018, p. 2, doi: 10.34385/proc.32.SESSION02_2.

[9] J. S. Aidan, H. K. Verma, and L. K. Awasthi, "Comprehensive survey on petya ransomware attack," in *Proc. Int. Conf. Next Gener. Comput. Inf. Syst. (ICNGCIS)*, Jammu, India, Dec. 2017, p. 122, doi: 10.1109/ICNGCIS.2017.30.

[10] *MS Office DDE Exploit/BadRabbit Ransomware*, Univ. Hawaii-West Oahu, Kapolei, HI, USA, Oct. 2017.

[11] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[12] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014, doi: 10.1109/SURV.2014.012214.00180.

[13] K. Cabaj and W. Mazurczyk, "Using software-defined networking for ransomware mitigation: The case of CryptoWall," *IEEE Netw.*, vol. 30, no. 6, pp. 14–20, Dec. 2016, doi: 10.1109/MNET.2016.1600110NM.

[14] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics," *Comput. Electr. Eng.*, vol. 66, pp. 353–368, Feb. 2018, doi: 10.1016/j.compeleceng.2017.10.012.

[15] G. Cusack, O. Michel, and E. Keller, "Machine learning-based detection of ransomware using SDN," in *Proc. ACM Int. Workshop Secur. Softw. Defined Netw. Netw. Function Virtualization (SDN-NFV)*, Tempe, AZ, USA, 2018, pp. 1–6, doi: 10.1145/3180465.3180467.

[16] M. Akbanov, V. G. Vassilakis, and M. D. Logothetis, "Ransomware detection and mitigation using software-defined networking: The case of WannaCry," *Comput. Electr. Eng.*, vol. 76, pp. 111–121, Jun. 2019, doi: 10.1016/j.compeleceng.2019.03.012.

[17] E. Rouka, C. Birkinshaw, and V. G. Vassilakis, "SDN-based malware detection and mitigation: The case of ExPetr ransomware," in *Proc. IEEE Int. Conf. Informat., IoT, Enabling Technol. (ICIoT)*, Doha, Qatar, Feb. 2020, pp. 150–155, doi: 10.1109/ICIoT48696.2020.9089514.

[18] R. Jin and B. Wang, "Malware detection for mobile devices using software-defined networking," in *Proc. 2nd GENI Res. Educ. Exp. Workshop*, Salt Lake City, UT, USA, Sep. 2013, pp. 81–88, doi: 10.1109/GREE.2013.24.

[19] J. M. Ceron, C. B. Margi, and L. Z. Granville, "MARS: An SDN-based malware analysis solution," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Messina, Italy, Aug. 2016, pp. 525–530, doi: 10.1109/ISCC.2016.7543792.

[20] J. A. Gomez-Hernandez, L. Alvarez-Gonzalez, and P. Garcia-Teodoro, "R-locker: Thwarting ransomware action through a honeyfile-based approach," *Comput. Secur.*, vol. 73, pp. 389–398, Mar. 2018, doi: 10.1016/j.cose.2017.11.019.

[21] A. El-Kosairy and M. A. Azer, "Intrusion and ransomware detection system," in *Proc. Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS)*, Riyadh, Saudi Arabia, Apr. 2018, pp. 1–7, doi: 10.1109/CAIS.2018.8471688.

[22] N. Biasini. (Oct. 2017). *Threat Spotlight: Follow the Bad Rabbit*. [Online]. Available: https://blog.talosintelligence.com/2017/10/bad- rabbit.html

[23] W. Wang. (Mar. 2018). *MS17-010/zzzexploit.py*. [Online]. Available: https://github.com/worawit/MS17-010/blob/master/zzz_exploit.py

[24] S. Hurley and S. Frankoff. (Nov. 2017). *BadRabbit MS17-010 Exploitation Part One: Leak and Control*. [Online]. Available: https://www.crowdstrike.com/blog/badrabbit-ms17-010-exploitation- part-one-leak-and-control/

[25] L. Dusseault. (Jun. 2007). *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. [Online]. Available: http://www.webdav.org/specs/rfc4918.html

[26] N. Pyle. (Sep. 2016). *Stop using SMB1—Microsoft Tech Community-425858*. [Online]. Available: https://techcommunity.microsoft.com/t5/storage-at-microsoft/stop-using-smb1/ba-p/425858

[27] (Jul. 2020). *Indiana University Knowledge Base, About the SMBv1 Retirement*. [Online]. Available: https://kb.iu.edu/d/aumn

[28] R. Woundy and K. Marez. (Dec. 2006). *Cable Device Management Information Base for Data-Over-Cable Service Interface Specification (DOCSIS)*, [Online]. Available: https://tools.ietf.org/html/rfc4639

[29] J. McCauley. (Jun. 2015). *Installing POX—GitHub Pages*. [Online]. Available: https://noxrepo.github.io/pox-doc/html/

[30] F. M. Alotaibi. (Sep. 2020). *SDN Ransomware Detection*. [Online]. Available: https://github.com/Falkarshmi/SDN-Ransomware-Detection

[31] F. M. Alotaibi. (Sep. 2020). *CPU-Measure/CPU.py*. [Online]. Available: https://github.com/Falkarshmi/CPU-Measure/blob/master/CPU.py

[32] I. Gofman. (Oct. 2017). *Advanced Threat Analytics Security Research Network Technical Analysis: NotPetya*. [Online]. Available: https://www.microsoft.com/security/blog/2017/10/03/advanced-threat-analytics-security-research-network-technical-analysis-notpetya/

[33] PwC. (2017). *Petya Ransomware-Strategic Report*. [Online]. Available: https://www.pwc.com/vn/en/assurance/assets/pwc-petya- strategic-report.pdf

[34] M. Hussain, N. Shah, and A. Tahir, "Graph-based policy change detection and implementation in SDN," *Electronics*, vol. 8, no. 10, pp. 1–21, Oct. 2019, doi: 10.3390/electronics8101136.

[35] M. Hussain and N. Shah, "Automatic rule installation in case of policy change in software defined networks," *Telecommun. Syst.*, vol. 68, no. 3, pp. 461–477, Nov. 2017, doi: 10.1007/s11235-017-0404-2.

[36] H. Hu, W. Han, S. Kyung, J. Wang, G.-J. Ahn, Z. Zhao, and H. Li, "Towards a reliable firewall for software-defined networks," *Comput. Secur.*, vol. 87, pp. 1–17, Nov. 2019, doi: 10.1016/j.cose.2019.101597.

FAHAD M. ALOTAIBI received the bachelor's degree in computer science and networking from Shaqra University, Saudi Arabia, and the master's degree in cybersecurity from the University of York, U.K. During his B.A. studies, he has worked as a Freelancer in two fields such as penetration testing and e-commerce. He currently works as a Teaching Assistant with Najran University. During his work at Najran University, he was assigned to provide several courses for students to develop them in penetration testing in addition to e-commerce. He obtained the Scholarship from Najran University for his master's degree.

VASSILIOS G. VASSILAKIS received the Ph.D. degree in electrical and computer engineering from the University of Patras, Greece, in 2011. He is currently a Lecturer (an Assistant Professor) in cyber security with the University of York, U.K. He has been involved in EU, U.K., and industry funded research and development projects related to the design and analysis of future mobile networks and Internet technologies. His main research interests include network security, the Internet of Things, next-generation wireless and mobile networks, and software-defined networks. He has served as an Associate Editor for *IEICE Transactions on Communications*, *IET Networks*, and *Optical Switching and Networking* (Elsevier).