Ransomware Detection and Mitigation using Software-Defined Networking: The Case of WannaCry

Maxat Akbanov[a], Vassilios G. Vassilakis[a,*], Michael D. Logothetis[b]

[a] *Dept. of Computer Science, University of York, York, United Kingdom*
[b] *Dept. of Electrical & Computer Engineering, University of Patras, Patras, Greece*

Abstract

Modern day ransomware families implement sophisticated encryption and propagation schemes, thus limiting chances to recover the data almost to zero. We investigate the use of software-defined networking (SDN) to detect and mitigate advanced ransomware threat. We present our ransomware analysis results and our developed SDN-based security framework. For the proof of concept, the infamous WannaCry ransomware was used. Based on the obtained results, we design an SDN detection and mitigation framework and develop a solution based on OpenFlow. The developed solution detects suspicious activities through network traffic monitoring and blocks infected hosts by adding flow table entries into OpenFlow switches in a real-time manner. Finally, our experiments with multiple samples of WannaCry show that the developed mechanism in all cases is able to promptly detect the infected machines and prevent WannaCry from spreading.

Keywords: WannaCry, ransomware, software-defined networking, OpenFlow, malware analysis

1. Introduction

Nowadays ransomware presents a huge and the fastest growing problem for all types of users from small households to large corporations and government bodies [1]. Starting from relatively simple fake antivirus applications in 2008, ransomware has evolved during the time and emerged into sophisticated forms such as crypto type ransomware. The apotheosis of this evolution is the occurrence of a new type of ransomware which combines the usage of exploits with worm-like spreading mechanisms to propagate itself in both internal and external networks. Moreover, the emergence of new ransomware families, such as WannaCry [2], showed that ransomware keeps evolving and cyber criminals are upgrading the ransomware code with more sophisticated features, such as worm

[*] Corresponding author
Email address: `vv573@york.ac.uk` (Vassilios G. Vassilakis)

propagation components and public-key encryption mechanisms. Therefore, from the research perspective, the design and development of new countermeasures is considered as an important task.

At the same time, a new emerging technology, known as software-defined networking (SDN) [3, 4], presents an important step towards completely programmable networks. This results in improved network resilience [5], performance [6, 7], and security [8]. SDN has also the potential to be used in different types of networks, including wireless [9], optical [10], smart grid [11], and Internet of things [12]. However, not many works have investigated the potential of SDN for ransomware threat detection and mitigation. Most of the existing studies focus on the security of the SDN itself, rather than considering working prototypes of security systems based on SDN properties. Only a few published papers investigate SDN-based malware detection and mitigation. Jin *et al.* [13] consider a mobile malware detection system based on SDN architecture. Several detection algorithms are examined, including IP blacklisting and a *connection success ratio* algorithm, and implemented using the Floodlight SDN controller. The developed system is able to detect malicious activities using real-time traffic analysis. Ceron *et al.* [14] design and develop an SDN-based malware analysis system which is capable to dynamically modify the network environment based on malicious activities. It has been demonstrated that the developed solution could trigger more malware events than traditional solutions.

With regard to current SDN-based solutions for ransomware threat, Cabaj *et al.* [15, 16] investigate several proposed methods. In particular, the SDN-based solution of [15] aims at improving the protection against the CryptoWall ransomware. Two approaches are introduced that try to block CryptoWall's connections with the command and control (C&C) server from infected hosts by using dynamic IP blacklisting. This solution utilizes an application written for the POX controller, which connects to a blacklist database and performs dynamic checks on IP addresses. The main drawback of these approaches is the requirement to pre-define the ransomware proxy servers used in the blacklisting. In [16], the network communication of the CryptoWall and Locky ransomware families is investigated. The proposed detection approach is based on an analysis of HTTP message sequences used during the communication with the C&C server. The feasibility of the proposed approaches has been confirmed by implementing and obtaining experimental results based on OpenvSwitch and POX. In particular, simulation results show a detection rate of 97-98% with only 4-5% false positives when relying on blacklisted domains. However, CryptoWall does not have a worm component, which makes its mitigation simpler. Our work extends the works of Cabaj *et al.* and presents a first attempt to investigate the feasibility of SDN techniques to detect and mitigate crypto ransomware with worm-spreading capabilities, such as WannaCry. Ransomware families which do not necessarily require communication with C&C servers in order to propagate are of particular interest.

In this work, we present our ransomware analysis results and our developed SDN-based security framework. For the proof of concept, the infamous WannaCry ransomware is used. However, the developed framework is also applicable in other ransomware families. In particular, we examine the behaviour of WannaCry during its execution in an isolated virtual lab environment. Based on the obtained results, we design an SDN detection and mitigation framework and develop a solution based on OpenFlow [17, 18], which is currently the most widely adopted SDN standard. The developed solution detects suspicious activities through network traffic monitoring and blocks infected hosts by adding flow table entries into OpenFlow switches in a real-time manner. The logic of the proposed framework has been implemented in the POX controller. For detection purposes, our implementation utilizes the WannaCry's features and its generated traffic. Finally, our experimental results with multiple samples of WannaCry show that the developed mechanism is able to promptly detect, in all cases, the infected machines and prevent WannaCry from spreading.

To the best of our knowledge, this is the first work that investigates and develops an SDN-based mitigation mechanism for ransomware with worm components, such as WannaCry. Furthermore, we have performed a comprehensive WannaCry analysis, both static and dynamic, and the identified WannaCry features have been used in our developed mechanism for real-time detection.

The rest of paper is organized as follows. Section 2 presents the background information on WannaCry and SDN. Sections 3 and 4 present the main findings from our conducted static and dynamic analysis of WannaCry, including its inherent network indicators. Section 5 presents our proposed design for an SDN-based detection and mitigation framework. Section 6 discusses our implementation, testbed, and experimental results. Finally, Section 7 draws the conclusions and discusses potential future directions.

2. Background

2.1. The Case of WannaCry

On 9 February 2017 researchers from Fortinet discovered the first sample of WannaCry, which they named as beta-version of the ransomware [19]. This version encrypted files by using the AES-128 algorithm and did not have any worm component implemented. On 28 March 2017, the same researchers found another improved version named as WannaCry 1.0, which used a hardcoded dictionary to access server message block (SMB) shared folders and dropped a Tor browser download link in the *cfg* file.

An enhanced WannaCry 2.0 version included critical improvement in propagation process, by implementing the worm module with leaked exploits from Shadow Brokers. In fact, this was the version that was observed during a massive attack on 12 May 2017 in more than 150 countries worldwide [2]. As stated in security reports, over 300,000 machines had been infected in a wide range of sectors, including healthcare, government, telecommunications, and gas/oil production. A unique feature which hinders the defense measures against WannaCry is its ability to spread using a worm component. This necessitates the development of protection mechanisms which can react quickly and in real time.

During the infection phase, WannaCry uses:

- The *EternalBlue* exploit for the SMB vulnerability that was patched by Microsoft on 14 March 2017 and has been described in the security bulletin MS17-010 [20]. This vulnerability allows the attackers to execute remote code by sending specially crafted messages to an SMBv1 server, connecting to TCP ports 139 and 445 of unpatched Windows systems.

- The *DoublePulsar* backdoor for gaining access and executing code on compromised machines. This essentially enables the installation of additional malware components on the machine. During the distribution process, WannaCry relies on the EternalBlue to enable an initial infection via the SMB vulnerability and if successful, attempts to implant the DoublePulsar backdoor on the compromised machines.

On 13 May 2017 a researcher from the MalwareTech company accidentally stopped the spreading of WannaCry by registering the following kill-switch domain, which was embedded in WannaCry's code [21]:
iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
On 14 May, a security researcher from Comae Technologies [22] reported the findings of two other versions of WannaCry that used different kill-switch domains. They identified that new versions of WannaCry utilized domain names that differed only in two letters. On 15 May, researchers from Rendition Security [23] identified another version of WannaCry, which did not implement a kill-switch domain check, as in previous versions. The researchers reported that previous successful mitigation approach based on sink-holing of the kill-switch domain did not work, and as an alternative method they suggested limiting the SMB traffic and implementing a host-based firewall. In fact, our proposed protection mechanism of Sections 5 and 6 follows this approach by utilizing the SDN functions.

2.2. The Concept of SDN

SDN is an emerging paradigm of programmable networks, that decouples the control and data planes [3]. This changes the way networks are designed and managed, and also enables new security solutions.

The data plane is responsible for forwarding and modification of packets, whereas the control plane determines the rules of how packets must be handled. The separation of the two planes enables the devices of the data plane (i.e., routers and switches) to function as simple forwarding elements, while the network control logic is implemented in a logically centralized controller. The control plane determines how individual packets should be handled and sends this information down to the data plane. This approach greatly simplifies the management of network devices, since they no longer need to understand a wide range of different protocols, but only need to understand the instructions received from the controllers. For the communication between SDN controllers and SDN devices the dominant protocol today is OpenFlow [18].

Controllers maintain a view of the entire network and implement policy decisions. Each SDN device (e.g, OpenvSwitch) has a *flow table* where the packet handling rules are stored in *flow entries*. The latter can be created, modified, or deleted by the controllers. During the network operation, when a device receives a packet, packet's fields are compared against flow entries. Then, the packet is processed and forwarded according to the rules in the flow table or is forwarded to the controller if no matching rule exists. This approach enables real-time network traffic management, including promising applications in the cybersecurity domain.

SDN controllers can be implemented either in software or in hardware. Software controllers are more popular nowadays and support a wide range of programming languages, such as Python (e.g., POX and Ruy controllers), Java (e.g., Floodlight, OpenDayLight, and ONOS controllers), or C++ (e.g., NOX controller) [24, 25].

3. WannaCry Analysis

In this section, we present our results from static and dynamic analysis of WannaCry. To perform static analysis, two virtual machines (VMs) were used. The characteristics of the host machine are: Intel Core i7-4700MQ 2.40 GHz and 16 GB RAM. The 1st VM was running Windows 7 SP1 and was infected with WannaCry 2.0. The 2nd VM was running REMnux and was used for malware analysis.

To perform dynamic analysis, a virtual testbed of Fig. 1 was built. This scheme allows observing domain name system (DNS) queries made by WannaCry during the infection and replication process, as performed by the worm component across the internal and external networks via the port 445 of the SMBv1 protocol. The REMnux machine acts as DNS and HTTP/HTTPS server, and is able to intercept all network communications using Wireshark. DNS and HTTP services in REMnux were enabled using the FakeDNS and HTTP Daemon utilities, respectively.

3.1. Static Analysis

We analyzed two WannaCry executables: the worm component and the encryption component. Their corresponding hashes and basic characteristics are shown in Table 1. Below we present our main findings from the static analysis.

Analysis with the Pestudio tool has revealed that the worm and the encryption components contain dynamic-link libraries (DLLs), as shown in Tables 2 and 3. During its execution, the worm invokes the *iphlpapi.dll* in order to retrieve network configuration settings for the infected host. The *kernel32.dll* and *msvcrt.dll* are two most invoked libraries by the encrypter. It was found that WannaCry uses Microsoft's crypto, file management, and C runtime file application programming interfaces (APIs). The Crypto API library is used to generate and manage random symmetric and asymmetric cryptographic keys.

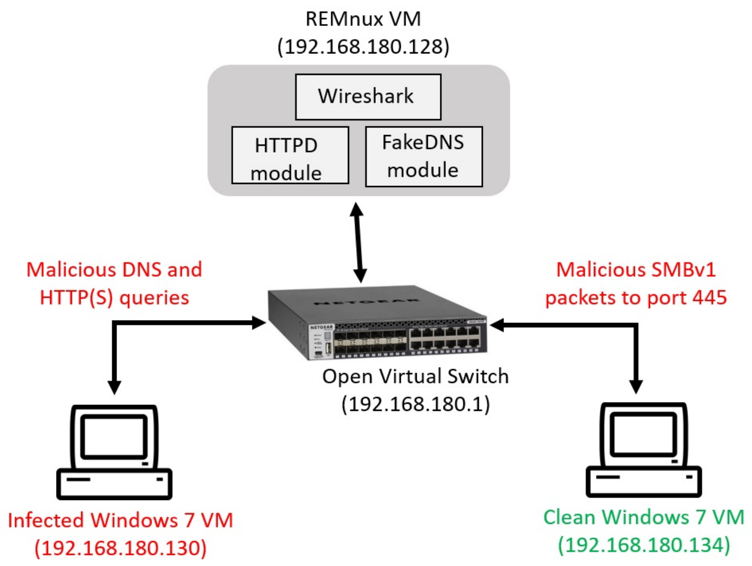


Figure 1: Virtual testbed for dynamic WannaCry analysis: Two Windows 7 VMs (infected and clean), one OpenvSwitch, and one REMnux VM with HTTP Daemon, FakeDNS utility, and Wireshark.

Table 1: WannaCry worm and encryption components: Hashes and file types.

	Worm Component
MD5	db349b97c37d22f5ea1d1841e3c89eb4
SHA1	e889544aff85ffaf8b0d0da705105dee7c97fe26
SHA256	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa 614ea04703480b1022c
File Type	PE32 executable (GUI) Intel 80386, for MSWindows
	Encryption Component
MD5	84c82835a5d21bbcf75a61706d8ab549
SHA1	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
SHA256	ed01ebfbc9eb5bbea545af4d01bf5f107166184048043 9c6e5babe8e080e41aa
File Type	PE32 executable (GUI) Intel 80386, for MSWindows

Table 2: Dynamic Link Libraries (DLLs) invoked by WannaCry's worm component.

Library	Imports	Description
ws2_32.dll	3	Windows Socket 2.0 32-bit
iphlpapi.dll	2	IP Helper API
wininet.dll	3	Internet Extensions for Win32
kernel32.dll	32	Windows NT BASE API Client
advapi32.dll	11	Advanced Windows 32 Base API
msvcp60.dll	2	Windows NT C++ Runtime Library
msvcrt.dll	28	Windows NT CRT

Table 3: Dynamic Link Libraries (DLLs) invoked by WannaCry's encryption component.

Library	Imports	Description
kernel32.dll	54	Windows NT BASE API Client
advapi32.dll	10	Advanced Windows 32 Base API
user32.dll	1	Multi-UserWindows USER API Client
msvcrt.dll	49	Windows NT CRT

3.2. Dynamic Analysis

Our dynamic analysis has revealed that, when started, the worm component invokes the *InernetOpenUrl* function and attempts to establish a connection with the following domain: *www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com*

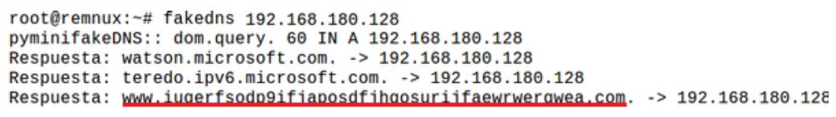
This, in fact, is a kill-switch domain and if is active, the worm stops running. In the case that the worm is not able to establish a connection with this domain, it continues to run and registers itself as a "Microsoft Security Center (2.0) Service" *mssecsvs 2.0* process on the infected machine.

The FakeDNS utility has captured the malicious DNS request on port 80, as shown in Fig. 2. At the same time, Wireshark reveals the DNS packet query field from the infected machine to the DNS server, as shown in Fig. 3.

After installing itself as a service, the worm component extracts the hard-coded *R resource* and then copies it to *C:\Windows\taskche.exe*. The R resource represents the encryption component of WannaCry. When invoked, the encrypter checks if at least one of the three mutual exclusion objects (mutexes) exists:

$$GlobalnMsWinZonesCacheCounterMutexA$$
$$GlobalnMsWinZonesCacheCounterMutexW$$
$$MsWinZonesCacheCounterMutexA$$

If the mutex is present on the system, then the encrypter immediately terminates. Otherwise, the encryption process begins. To encrypt each file, a different



```
root@remnux:~# fakedns 192.168.180.128
pyminifakeDNS:: dom.query. 60 IN A 192.168.180.128
Respuesta: watson.microsoft.com. -> 192.168.180.128
Respuesta: teredo.ipv6.microsoft.com. -> 192.168.180.128
Respuesta: www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com. -> 192.168.180.128
```

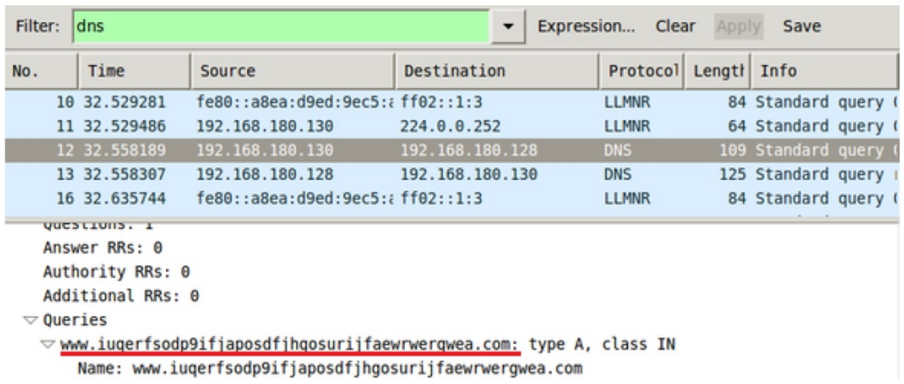
Figure 2: A malicious DNS request as captured by the FakeDNS utility of the REMnux VM.

16-byte symmetric AES key is generated with the help of the *CryptGenRandom* function. Then, every generated AES key is encrypted with the public RSA key (which is part of the encrypter) and stored inside the file header starting with *WANACRY!*. The encrypted files are renamed and get the *.WNCRY* extension. The encrypter has a password-protected ZIP file. By disassembling the encrypter, as shown in Fig. 4, it was possible to reveal the password of the ZIP file: “WNcry@2ol7”. The most important contents of the ZIP file are briefly described below:

- *msg* is a folder containing rich text format (RTF) files which are the instructions in different languages for displaying the extortion messages to victims.
- *b.wnry* contains instructions for decrypting user files.
- *c.wnry* is a list of the Tor addresses with *.onion* extension and a link to the Tor browser compressed installer.
- *s.wnry* is a compressed file with the Tor browser executable.
- *taskdl.exe* is an executable for deleting files with the extension *.WNCRY*.
- *taskse.exe* is an executable for running WannaCry remote desktop protocol sessions.
- *u.wnry* is the decryption component of WannaCry.

Our analysis has also shown that WannaCry attempts to establish persistence on the infected device by:

- Creating an entry in the Windows registry, so that it is invoked every time the infected computer reboots.
- Adding itself to the AutoRun feature of Windows.
- Utilizing the *icacls* command to enable full access to all files on the infected device.
- Deleting the backup copies and preventing the device rebooting in the *safe mode*.
- Attempting to eliminate SQL and MS Exchange database processes by executing certain shell commands.



```
Filter: dns                                   Expression...  Clear  Apply  Save

No.  Time       Source                  Destination      Protocol  Lengtl  Info
  10 32.529281  fe80::a8ea:d9ed:9ec5:ε  ff02::1:3        LLMNR         84  Standard query (
  11 32.529486  192.168.180.130         224.0.0.252      LLMNR         64  Standard query (
  12 32.558189  192.168.180.130         192.168.180.128  DNS          109  Standard query (
  13 32.558307  192.168.180.128         192.168.180.130  DNS          125  Standard query
  16 32.635744  fe80::a8ea:d9ed:9ec5:ε  ff02::1:3        LLMNR         84  Standard query (

    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▽ Queries
    ▽ www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com: type A, class IN
        Name: www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
```

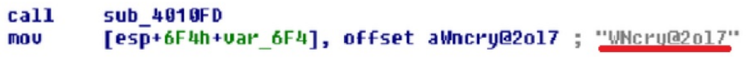
Figure 3: A malicious DNS request as captured by the Wireshark on the REMnux VM.

4. WannaCry Communications

After performing initial interactions and checking the connectivity with the kill-switch domain, the worm functionality is established by initiating the *mssecsvs 2.0* service. This service tries to spread WannaCry's payload through the SMB vulnerability on any vulnerable system. In order to perform this, WannaCry creates two separate threads that simultaneously replicate the payload in internal (local) and external networks. In the local network, before starting the propagation process, WannaCry obtains the IP addresses of local network interfaces using the *GetAdaptersInfo* function and identifies the available subnets.

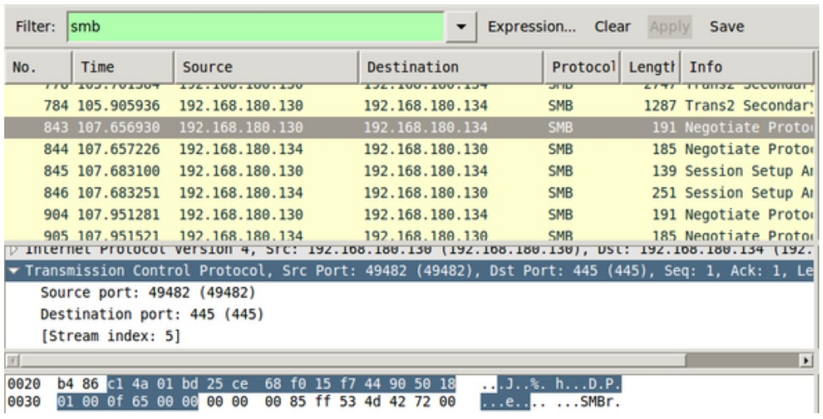
After that, WannaCry attempts to connect to all possible IP addresses in the existing local networks on the TCP port 445 (the default port for SMB over IP). If the connections is established, WannaCry attempts to exploit the EternalBlue vulnerability of the SMB service, as explained in [20]. During our experiments, we observed connection attempts where the infected machine (IP 192.168.180.130) sent SMB packets to a Windows host (IP 192.168.180.134), as shown in Fig. 5. At the same time, we also observed that WannaCry tried to spread to external networks by generating IP addresses and attempting to connect to TCP port 445. This has been detected using Wireshark on REMnux, as shown in Fig. 6.

During the SMB probing by WannaCry, one of the important characteristics of the generated traffic is that it contains two hardcoded IP addresses: 192.168.56.20 and 172.16.99.5. They can be obtained by extracting the strings from the WannaCry executable. In particular, WannaCry sends three NetBIOS session setup packets, where two of them contain the aforementioned hardcoded IP addresses. As part of its activity, WannaCry also attempts to reach the C&C servers by using the *c.wnry* file, which contains the configuration data, a list of possible *.onion* addresses to be connected, and the compressed Tor installation file. During its communication with Tor addresses, WannaCry initiates a secure HTTPS connection to port 443, and uses common Tor ports 9001 and 9050 for



```
call    sub_4010FD
mov     [esp+6F4h+var_6F4], offset aWncry@2ol7 ; "WNcry@2ol7"
```

Figure 4: The password of the ZIP file in the encryption component when disassembling with IDA Pro.



```
No.  Time        Source           Destination      Protocol Length Info
784  105.905936  192.168.180.130  192.168.180.134  SMB      1287   Trans2 Secondary
843  107.656930  192.168.180.130  192.168.180.134  SMB       191   Negotiate Proto
844  107.657226  192.168.180.134  192.168.180.130  SMB       185   Negotiate Proto
845  107.683100  192.168.180.130  192.168.180.134  SMB       139   Session Setup An
846  107.683251  192.168.180.134  192.168.180.130  SMB       251   Session Setup An
904  107.951281  192.168.180.130  192.168.180.134  SMB       191   Negotiate Proto
905  107.951521  192.168.180.134  192.168.180.130  SMB       185   Negotiate Proto
▷ Internet Protocol Version 4, Src: 192.168.180.130 (192.168.180.130), Dst: 192.168.180.134 (192.
▾ Transmission Control Protocol, Src Port: 49482 (49482), Dst Port: 445 (445), Seq: 1, Ack: 1, Le
    Source port: 49482 (49482)
    Destination port: 445 (445)
    [Stream index: 5]

0020  b4 86 c1 4a 01 bd 25 ce  68 f0 15 f7 44 90 50 18   ...J..%. h...D.P.
0030  01 00 0f 65 00 00 00 00  00 85 ff 53 4d 42 72 00   ...e.... ...SMBr.
```

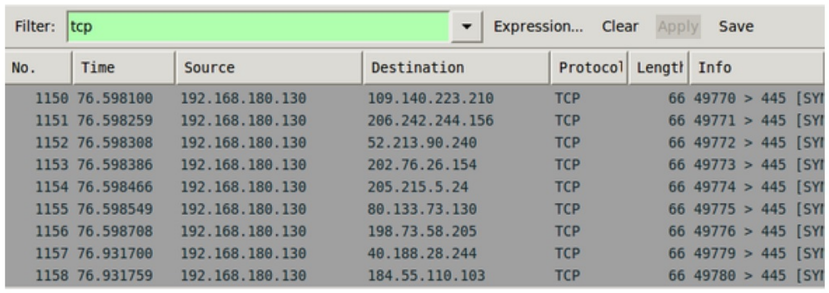
Figure 5: SMB packets sent and received by the infected machine in the local network: Attempting the SMB exploit.

network traffic and directory information. The aforementioned identified behaviour of WannaCry has been used as a basis for designing and implementing our detection and mitigation mechanism of Section 5.

5. The Proposed SDN-Based Mechanism

Our proposed SDN-based detection and mitigation mechanism relies on inspection of the DNS traffic with dynamic blacklisting, which particularly observes the network traffic for the presence of malicious domain names or IP addresses used during WannaCry's communication with the C&C server (as identified in Sections 3 and 4). As soon as such an attempt is detected, it is blocked. The list of malicious domain names is commonly specified in a local blacklisting file or by using online databases. One of the main benefits of this method is that it provides simplicity in implementation and effectiveness in detection and mitigation of malware activity [15]. Therefore, this approach was used as a basis for our proposed mechanism.

The conceptual design of the proposed mechanism is depicted in Fig. 7. The main detection and mitigation functionality is carried out by the developed SDN application. This application has been implemented on the SDN controller, which allows inspecting the entire network traffic and issuing instructions to the OpenFlow switch to update its flow table by installing appropriate rules.



```
No.   Time       Source           Destination       Protocol Length Info
 1150 76.598100  192.168.180.130  109.140.223.210   TCP         66 49770 > 445 [SY
 1151 76.598259  192.168.180.130  206.242.244.156   TCP         66 49771 > 445 [SY
 1152 76.598308  192.168.180.130  52.213.90.240     TCP         66 49772 > 445 [SY
 1153 76.598386  192.168.180.130  202.76.26.154     TCP         66 49773 > 445 [SY
 1154 76.598466  192.168.180.130  205.215.5.24      TCP         66 49774 > 445 [SY
 1155 76.598549  192.168.180.130  80.133.73.130     TCP         66 49775 > 445 [SY
 1156 76.598708  192.168.180.130  198.73.58.205     TCP         66 49776 > 445 [SY
 1157 76.931700  192.168.180.130  40.188.28.244     TCP         66 49779 > 445 [SY
 1158 76.931759  192.168.180.130  184.55.110.103    TCP         66 49780 > 445 [SY
```

Figure 6: SMB packets sent by the infected machine to external networks: Attempting the SMB exploit (TCP port 445).

In Fig. 7, numbers 1 to 5 represent different steps of the detection and mitigation process and are explained below:

- *Step 1*: Malicious TCP traffic from an infected host arrives to the OpenFlow switch. This traffic includes SMB probing and DNS query packets generated by WannaCry.
- *Step 2*: Assuming that initially there is no flow entry for the given infected host, the switch redirects all TCP traffic to the controller (this is the default action), in order to obtain further instructions on how to handle these packets.
- *Step 3*: All packets that are received by the controller, are passed to the application. The application performs the following functions: parses packets, checks against the blacklist database file, and creates new flow entries in the switch. After receiving the packets, the application parses them in order to find any matches with WannaCry's inherent network indicators (as discussed in Section 4).
- *Step 4*: The packets are checked against the database file containing the list of IP addresses and simultaneously with any matches to the TCP port numbers used by WannaCry.
- *Step 5*: If malicious communication is detected, the application creates a new flow entry in the switch instructing it to block the malicious traffic originating from the infected host.

For example, the new flow table of the OpenFlow switch may contain the entries shown in Table 4. In this case, the IP address 192.168.180.130 was taken from the blacklist database file (Step 3) and the relevant TCP port numbers (445, 139, etc.) are known to be used by the SMB protocol.

6. Experimental Testbed and Results

In order to evaluate the feasibility of our proposed mechanism, we have built an experimental testbed, depicted in Fig. 8. All experiments were conducted on

Table 4: Entries in the flow table of the OpenFlow switch.

Rule	Src addr	Src port	Dst addr	Dst port	Protocol	Action
1	192.168.180.130	any	any	445	TCP	deny
2	192.168.180.130	any	any	139	TCP	deny

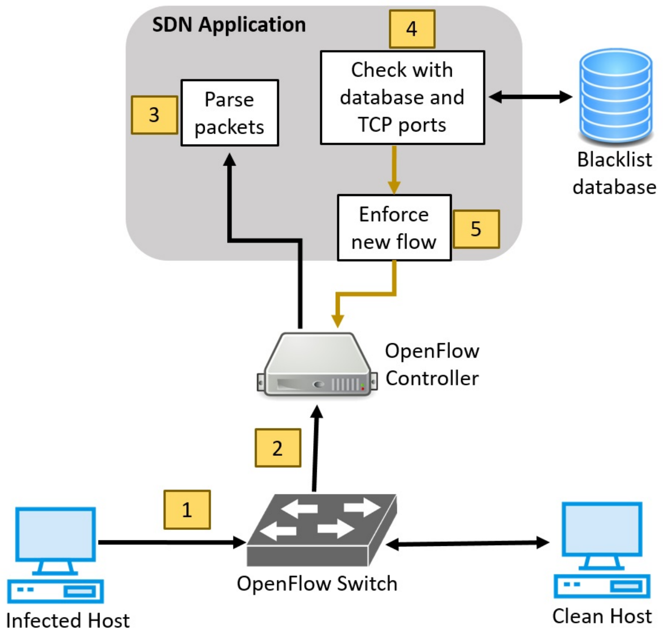


Figure 7: Conceptual design of the proposed SDN-based mechanism.

VMs with the help of the VMWare hypervisor. The testbed consists of six VMs. Three VMs are running Windows 7 SP1, two VMs are running Ubuntu 12.04 and one VM is running REMnux. All VMs are located in the same subnet: 192.168.180.0/24. One of the Windows VMs has been infected, whereas two other Windows VMs have the vulnerability required by WannaCry (as explained in Subsection 2.1). The two Ubuntu VMs are running the SDN controller and the SDN switch. The REMnux VM is running DNS and HTTP services to attract the communication of WannaCry in the infected VM.

As primary software for our SDN switch, the popular OpenvSwitch was chosen. It supports the OpenFlow protocol versions starting from the initial 1.0 to the latest 1.5 version. For our SDN controller, the Python-based open source POX software was chosen. The main advantage of POX is its flexibility, as it provides fast and easy programmability and supports a wide range of different applications. The core logic of network traffic handling has been implemented by two Python-based plugins written for the POX controller. These plugins check the network traffic for blacklisted IP addresses (IP address blocker) and TCP port numbers (TCP port blocker). A simplified Python code for the TCP port blocker is presented below:

```python
##################################
#       TCP port blocker
##################################
from pox.core import core #Imports the POX core object
#Specifies the ports to block in port_list variable
port_list = set()
#Function handles packet events and kills the ones with a specified port
def port_inspect (packet_event):
   tcp_packet = event.parsed.find("tcp")
   if not tcp_packet: return #Not TCP packet
   if tcp_packet.srcport in port_list or tcp_packet.dstport in port_list:
   #Halts the event and installs a flow table entry
   core.getLogger("blocker").debug("Blocked TCP port %s <-> %s",
       tcp_packet.srcport, tcp_packet.dstport)
   event.halt = True
#Function block ports on the switch
def block (*ports=" "):
   #Adds specified ports through command line
   port_list.update(int(i) for i in ports.replace(","," ").split())
   #Listens to packet events
   core.openflow.addListenerByName("PacketIn", port_inspect)
```

If the check triggers an alert, then a new flow entry is installed in the OpenvSwitch to block the corresponding traffic flow. Both plugins are placed into *ext* directory of the POX controller and are invoked simultaneously from the command line during the network operation. These two plugins implement the functionality of the SDN application shown in Fig. 7. In particular, the TCP port blocker plugin identifies packets with TCP ports 443, 139, 445, 9001 and 9050, and if detected, creates a corresponding flow entry in the OpenvSwitch to block the traffic from the infected host. The IP address blocker plugin has a similar logic, and handles traffic for the specified blacklisted IP addresses. Due to space limitations, we do not present the code for the IP address blocker.

The blacklist IP addresses are placed into an *csv* file and are loaded from the *ext* directory of the POX controller. The contents of the csv file are presented in Table 5. In this file, the 2nd column specifies the IP addresses observed during WannaCry infection, whereas the 3rd column specifies the IP address of the infected host. Tests have been performed in order to ensure that the IP address blocker plugin performs inspections for the given host traffic for any match with the IP addresses of the 2nd column. If a match is found, then the traffic from the host is blocked.

In order to verify the effectiveness of our mechanism, each experiment is divided in two phases. During the 1st phase, the victim's VM was infected, but the plugins have not been initialized in the POX controller. Also, the malicious

Table 5: The blacklist.csv file of the POX controller.

id	ip_0	ip_1
1	192.168.56.20	192.168.180.30
2	172.16.99.5	192.168.180.30
3	72.52.179.175	192.168.180.30
4	109.140.223.210	192.168.180.30
5	206.242.244.156	192.168.180.30
6	52.213.90.240	192.168.180.30
7	202.76.26.154	192.168.180.30
8	205.215.5.24	192.168.180.30
9	80.133.73.130	192.168.180.30
10	198.73.58.205	192.168.180.30
11	40.188.28.244	192.168.180.30
12	184.55.110.103	192.168.180.30

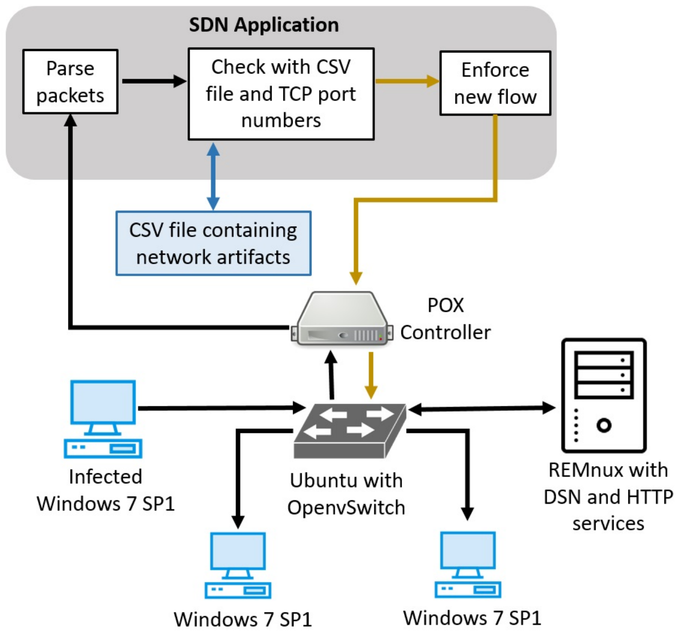


Figure 8: Experimental testbed for WannaCry detection and mitigation.

DNS queries, originated from the infected machine, were registered on REMnux machine. At the same time, the SMB probing with external IP address requests was observed on uninfected Windows hosts and REMnux machine. During the 2nd phase, the two Python plugins were started in the POX controller. After that, no malicious DNS queries or SMB probing packets were sent to uninfected machines. This shows that the POX controller has successfully identified the malicious traffic from the infected machine (IP address: 192.168.180.30) and blocked it by creating in real-time a corresponding flow entry in the OpenvSwitch.

7. Conclusion and Future Work

We have designed and implemented a feasible approach based on software-defined networking to detect and mitigate ransomware threat. Our experiments have been conducted with real samples of WannaCry ransomware. The developed solution involves an application built for a centralized controller which communicates with the switches using the OpenFlow protocol. In particular, our solution includes two plugins for blocking malicious network addresses and port numbers based on a blacklist database and involving WannaCry characteristics derived via static and dynamic analysis.

To the best of our knowledge, this is the first work to demonstrate that the security mechanisms based on software-defined networking are capable to successfully stop the infections from ransomware with worm-spreading capabilities. In particular, our experimental results show that the proposed mechanism is able to detect and block the traffic from infected host, and therefore secure the remaining untouched part of the network. Moreover, this work shows that the proposed approach is feasible in practice and capable to block worm components in real time. Furthermore, due to flexibility and programmability of software-defined networks, the presented mechanism can potentially be extended further to protect against other ransomware families.

As a future work, we plan to enhance the developed mechanism with hardware acceleration and to investigate the possibility of implementing anomaly-based detection algorithms using machine learning techniques. Another interesting research direction is to investigate collaborative intrusion detection which would combine information from multiple OpenFlow switches. Finally, for an improved performance and to address any scalability issues due to massive attacks, different types of controllers should be tested and optimized.

References

[1] O'Brien, D., 2017. Ransomware, Internet Security Threat Report, Symantec.

[2] Symantec, 2017. What you need to know about the WannaCry ransomware. Threat Intelligence, Oct. 2017.

[3] Nunes, B.A., Mendonca, M., Nguyen, X.-N., Obraczka, K., and Turletti, T., 2014. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials 16(3)*, pp. 1617-1634. DOI: 10.1109/SURV.2014.012214.00180

[4] Yang, H., Zhang, J., Zhao, Y., Han, J., Lin, Y. and Lee, Y., 2016. SUDOI: Software defined networking for ubiquitous data center optical interconnection. *IEEE Communications Magazine, 54(2)*, pp. 86-95. DOI: 10.1109/MCOM.2016.7402266

[5] Yang, H., Zhang, J., Zhao, Y., Ji, Y., Wu, J., Lin, Y., Han, J. and Lee, Y., 2015. Performance evaluation of multi-stratum resources integrated resilience for software defined inter-data center interconnect. *Optics Express, 23(10)*, pp. 13384-13398. DOI: 10.1364/OE.23.013384

[6] Wu, J., Dong, M., Ota, K., Li, J. and Guan, Z., 2018. Big data analysis-based secure cluster management for optimized control plane in software-defined networks. *IEEE Transactions on Network and Service Management, 15(1)*, pp.27-38. DOI: 10.1109/TNSM.2018.2799000

[7] Vassilakis, V.G., Moscholios, I.D. and Logothetis, M.D., 2017. Efficient radio resource allocation in SDN/NFV based mobile cellular networks under the complete sharing policy. *IET Networks, 7(3)*, pp. 103-108. DOI: 10.1049/iet-net.2017.0053

[8] Fichera, S., Galluccio, L., Grancagnolo, S.C., Morabito, G. and Palazzo, S., 2015. OPERETTA: An OPEnflow-based REmedy to mitigate TCP SYNFLOOD Attacks against web servers. *Computer Networks, 92*, pp.89-100. DOI: 10.1016/j.comnet.2015.08.038

[9] Vassilakis, V.G., Moscholios, I.D., Alzahrani, B.A. and Logothetis, M.D., 2016. A software-defined architecture for next-generation cellular networks. *Proc. IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, May 2016, pp. 1-6. DOI: 10.1109/ICC.2016.7511018

[10] Yang, H., Bai, W., Yu, A., Yao, Q., Zhang, J., Lin, Y. and Lee, Y., 2018. Bandwidth compression protection against collapse in fog-based wireless and optical networks. *IEEE Access, 6*, pp. 54760-54768. DOI: 10.1109/ACCESS.2018.2872467

[11] Li, G., Wu, J., Li, J., Ye, T. and Morello, R., 2017. Battery status sensing software-defined multicast for V2G regulation in smart grid. *IEEE Sensors Journal, 17(23)*, pp.7838-7848. DOI: 10.1109/JSEN.2017.2731971

[12] Fichera, S., Gharbaoui, M., Castoldi, P., Martini, B. and Manzalini, A., 2017. On experimenting 5G: Testbed set-up for SDN orchestration across network cloud and IoT domains. *Proc. IEEE Conference on Network Softwarization (NetSoft)*, Bologna, Italy, July 2017, pp. 1-6. DOI: 10.1109/NETSOFT.2017.8004245

[13] Jin, R. and Wang, B., 2013. Malware detection for mobile devices using software-defined networking. *Proc. 2nd GENI Research and Educational Experiment Workshop*, Washington, USA, March 2013. DOI: 10.1109/GREE.2013.24

[14] Ceron, J.M., Margi, C.B., and Granville, L.Z, 2016. MARS: An SDN-based malware analysis solution. *Proc. IEEE Symposium on Computers and Communication (ISCC)*, Messina, Italy, June 2016. DOI: 10.1109/ISCC.2016.7543792

[15] Cabaj, K. and Mazurczyk, W., 2016. Using software-defined networking for ransomware mitigation: The case of CryptoWall. *IEEE Network, 30(6)*, pp. 14-20. DOI: 10.1109/MNET.2016.1600110NM

[16] Cabaj, K., Gregorczyk, M. and Mazurczyk, W., 2018. Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics. *Computers & Electrical Engineering, 66*, pp. 353-386. DOI: 10.1016/j.compeleceng.2017.10.012

[17] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J., 2008. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Computer Commun. Review, 38(2)*, pp. 69-74. DOI: 10.1145/1355734.1355746

[18] Suzuki, K., Sonoda, K., Tomizawa, N., Yakuwa, Y., Uchida, T., Higuchi, Y., Tonouchi, T. and Shimonishi, H., 2014. A survey on OpenFlow technologies. *IEICE Trans. Commun., E97-B(2)*, pp. 375-386. DOI: 10.1587/transcom.E97.B.375

[19] S. Biddle, WannaCry: Evolving History from Beta to 2.0, May 2017, https://www.fortinet.com/blog/threat-research/wannacry-evolving-history-from-beta-to-2-0.html [March 14, 2019].

[20] Microsoft Security Bulletin MS17-010 - Critical.

[21] MalwareTech, How to accidentally stop a global cyber attacks, https://www.malwaretech.com/2017/05/how-to-accidentally-stop-a-global-cyber-attacks.html [March 14, 2019].

[22] M. Suiche, WannaCry - new variants detected!, May 2017, https://blog.comae.io/wannacry-new-variants-detected-b8908fefea7e [March 14, 2019].

[23] Rendition Infosec, New “no kill switch” WanaCry worm found!, https://blog.renditioninfosec.com/2017/05/wanacrypt0r-worm-with-kill-switch-patched-out/ [March 14, 2019].

[24] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., and Smeliansky, R., 2013. Advanced study of SDN/OpenFlow controllers. *Proc. 9th Central & Eastern European Software Engineering Conference in Russia*, Moscow, Russia, Oct. 2013. DOI: 10.1145/2556610.2556621

[25] Salman, O., Elhajj, I.H., Kayssi, A. and Chehab, A., 2016. SDN controllers: A comparative study. *Proc. 18th Mediterranean Electrotechnical Conference (MELECON)*, Limassol, Cyprus, April 2016, pp. 1-6. DOI: 10.1109/MELCON.2016.7495430

Maxat Akbanov received the M.Sc. degree in Cyber Security from the University of York, UK, in 2018. He is currently working at private sector in Kazakhstan and involved in developing several startup projects for governmental sponsored strategy "Digital Kazakhstan" and "Cyber Shield". His main research interests include network and malware forensics, software-defined networking, covert channels, cryptography, Internet of things, machine learning and artificial intelligence.

Vassilios G. Vassilakis is a Lecturer in Cyber Security at the University of York, UK. He's been involved in EU, UK, and industry funded R&D projects related to the design and analysis of future mobile networks and Internet technologies. His main research interests are in the areas of network security, next-generation wireless and mobile networks, Internet of things, and software-defined networks.

Michael D. Logothetis received his Dipl.Eng. degree and Doctorate in Electrical Engineering, both from the University of Patras, Greece, in 1981 and 1990 respectively. From 1991 to 1992 he was Research Associate in NTT's Telecommunication Networks Laboratories, Tokyo. In 2009 elected Professor in the ECE Department of the University of Patras. His research interests include teletraffic theory and optimization of telecommunications networks.