2IC80 - Group 29 - Final Project
Reverse engineering NotPetya

Vincent Hoogendam 1440551 Mischa ter Haar 1444816
Valentijn Koppert 1575120 Lars Giesberts 1263684

June 2023

1 Introduction

For our subject to study we have chosen to reverse engineer the NotPetya malware attack. In our attempt, we hope to gain knowledge about the attack specifically and malware attacks in general. Furthermore, we aim to develop a skill set to deal with these kinds of problems as it is also a worthwhile venture.

2 History

Ransomware and cyber-attacks have been around for quite a while, but the way in which they operate, and their respective objectives, have changed with time. What started as a malicious link in an email that, once accidentally installed, asked for 300 dollars in Bitcoin in return for a decryption key, has now evolved into software that can paralyze an entire country. NotPetya was predated by the similarly named Petya attack that occurred in 2016[1]. Petya spread via a malicious email attachment and aimed to encrypt the infected PC's hard drive, not by encrypting specific files, but by installing its own bootloader and encrypting the file table. This effectively makes the PC "forget" where its files are. If you pay the required bitcoin, the software will revert the changes. In 2017, a new version of the malicious software started spreading. It still encrypted the file table but now it spread way quicker, and not via email attachments. Entire networks in Ukraine were affected via exploits in Windows called EternalBlue and EternalRomance. This new version, dubbed NotPetya by Kapersky, encrypted not only the file table, but all the files. NotPetya is not ransomware, however, as the ID of the infected PC, meant as a receipt to send along with the payment so that the decryption key could be sent to the right customer, was randomly generated and thus of no use. Furthermore, in the encryption process, the files were also damaged beyond repair. The real purpose of Notpetya might have been the paralyzing of Ukraine, a country Russia is in conflict with, with the premise that the virus came from Russia.

3 Attack Description

NotPetya first spread in June of 2017. The root infestation has been traced back to a backdoor in popular accounting software in Ukraine called M.E DOC. The difference with Petya is that Petya could not spread on its own, it instead relied on users to click on malicious email attachments. NotPetya did not have this problem as it employed several exploits to spread itself in an entire network. One of the exploits, called EternalBlue, was developed by the NSA but got leaked by the "Shadow Broker" hacking group. EternalBlue was used before the NotPetya attack when WannaCry took place. A patch was released in between the two attacks but not enough systems updated in time.
EternalBlue exploits the SMB protocol by using a vulnerability in version 1 of SMB that allows remote code execution from attackers due to the mishandling of specially crafted packets. Another exploit that is used is called Mimikatz and it is a tool that allows the attacker to get the credentials, such as passwords, that are still stored in RAM. These two exploits together form a powerful way to quickly and easily spread the malware over entire systems.
Once NotPetya has infested itself in a system, it starts by harvesting credentials on the network to increase its spreading capabilities. After it has completed this task it is time to activate the payload. What exactly happens at this stage is determined by what antivirus software NotPetya can find on the system [2]. If none are detected, all files are encrypted, the Master Boot Record is wiped and anti-forensic techniques, in which journals of changes to the file system and event logs are deleted, are deployed. If Kapersky is present at the system, The anti-forensics are not executed and the function that overrides the initial sectors of the physical drives is not called. Finally, the system is rebooted.

4 Technical Setup

For our reverse engineering, we used Ghidra, an open-source reverse engineering tool developed by the NSA. Next to a decompiler, Ghidra provides several additional functionalities to aid in reverse engineering. The SHA256 hash of the executable we reverse-engineered is:
`027cc450ef5f8c5f653329641ec1fed91f694e0d229928963b30f6b0d7d3a745`
The Ghidra project we created can be found on GitHub: `https://github.com/Giebelsl/2IC80`

5 Attack Analysis

5.1 Root Function and Initial Malware Setup

Looking at the code in Ghidra, there are two functions that are exported from the .dll file; entry and Ordinal_1. We concluded that the entry function does not do much in the actual attack and identified Ordinal_1 as the root function, from which we were able to reverse-engineer its nested functions to obtain the

execution process of the NotPetya attack.

The first function called by Ordinal_1 sets up the rest of the attack by attempting to obtain three privileges, which can be found in table 1.. Thereafter, it checks for the presence of three antivirus executables. It uses an encryption routine to hide the names of the processes it is looking for. Through a brute force algorithm by Carbon Black[3], the process names and the corresponding antivirus products were found. A description of these can be found in table 2. It sets flags for which privileges it has acquired and which antivirus programs are present. For the antivirus programs, Kaspersky receives a unique flag while Norton and Symantec share the same flag. Later in its execution process, it checks these flags to see if it has the appropriate privileges to perform certain tasks and to check if antivirus programs are present. It also assigns memory for itself to be stored on the system's hard drive. After this initial setup, the Ordinal_1 routine begins the actually harmful steps of the attack.

Privilege	SeShutdownPrivilege	SeDebugPrivilege	SeTcbPrivilege
Description	The right to shut down the system	The right to debug and adjust the memory of a process owned by another account	The right to act as a part of the operating system

Table 1: Privileges aquired by NotPetya

Encrypted value	**Antivirus program**
0x2E214B44	Kaspersky
0x651B3005	Norton
0x6403527E	Symantec

Table 2: Antivirus scans done by NotPetya

5.2 Command line handling

The function that we have renamed `several_command_line_argument` seems to handle command line arguments given to NotPetya. It takes as a parameter `ordinal_cmdline_string` argument, which is a wide string containing the entire command line input. The code first checks if the argument is not null, and if it is not it determines the length of the string. The string is then passed to the `CommandLineToArgvW` function. The purpose of this function is to split the command line string up into an array of individual arguments. Consequently, the first argument is extracted and converted to an integer by the `StrToIntW` function. If the integer is larger than zero, it is assigned to the `int_of_pointer_to_first_command_line_arg` variable. The code loops and processes any further arguments. Whilst handling the individual arguments, the code checks for substring '-h'. If this is found, the function `FUN_100069a2`

is called. A check for a colon is also performed. If encountered, the argument is split into 2 parts, one before the colon and one after. These are then processed by the function `FUN_10006de0`.
After everything is processed, the memory used by the argument array is freed by the `LocalFree` function.

```
undefined4 several_command_line_arguments(LPCWSTR ordinal_cmdline_string)

{
  LPCWSTR function_parameter;
  LPWSTR *commandlineArg;
  int commandlineArg_int;
  LPWSTR substring_location;
  uint local_value1;
  uint local_value0;
  WCHAR pointer_function_parameter;

  if (ordinal_cmdline_string != (LPCWSTR)0x0) {
    function_parameter = ordinal_cmdline_string;
    do {
      pointer_function_parameter = *function_parameter;
      function_parameter = function_parameter + 1;
    } while (pointer_function_parameter != L'\0');
    if ((int)function_parameter - (int)(ordinal_cmdline_string + 1) >> 1 != 0) {
      local_value0 = 0;
      commandlineArg = CommandLineToArgvW(ordinal_cmdline_string,(int *)&local_value0);
      if (commandlineArg != (LPWSTR *)0x0) {
        if (0 < (int)local_value0) {
          commandlineArg_int = StrToIntW(*commandlineArg);
          local_value1 = 1;
          if (0 < commandlineArg_int) {
            int_of_pointer_to_first_command_line_arg = commandlineArg_int;
          }
          if (1 < local_value0) {
            do {
              function_parameter = commandlineArg[local_value1];
              substring_location = StrStrW(function_parameter,L"-h");
              if (function_parameter == substring_location) {
                FUN_100069a2();
                break;
              }
              substring_location = StrChrW(function_parameter,L':');
              if (substring_location != (LPWSTR)0x0) {
                *substring_location = L'\0';
                FUN_10006de0(function_parameter,substring_location + 1,1);
```

```
            }
            local_value1 = local_value1 + 1;
          } while (local_value1 < local_value0);
        }
      }
      LocalFree(commandlineArg);
    }
  }
  if (int_of_pointer_to_first_command_line_arg == 0) {
    int_of_pointer_to_first_command_line_arg = 0x3c;
  }
  return 0;
}
```

5.3 Device and Network Reconnaissance

In the next step of its execution, the routine checks if it owns SeDebugPrivilege and if it does, it first checks if it is already installed on the device. If it is not, it creates a file for itself on the system's hard drive. Next, it gets the geometry of the drive and overwrites a certain part of the drive. Then, it checks for the presence of Kaspersky and tries to execute a function. If Kaspersky is running or the nested function fails, the function immediately returns. Otherwise, it dismounts the drive volume and then returns.
After this is done, it does a certain time calculation to perform a shutdown. There are two separate paths in which it does this depending on the operating system version on the device.

Next, the malware tries to gain information on the network the infected device is a part of. It first gets information on adapters and possible DHCP clients. It also repeatedly looks for TCP tables and IP tables and obtains all visible servers in a given domain.

5.4 Spreading of the malware

At this point, we established that NotPetya checks for the necessary privileges it needs to do its malicious behaviour. The next step in NotPetya's operation is self-propagation. NotPetya used several ways to propagate itself, some famous techniques NotPetya uses are known as EternalBlue and Mimikatz[4].

5.4.1 EternalBlue

We already touched upon a brief history of EternalBlue in section 3. Due to time constraints and the complexity of the EternalBlue exploit we have not been able to reverse engineer the precise workings of EternalBlue. However since EternalBlue is a significant part of NotPetya we still wanted to provide a

brief overview of how EternalBlue works. We then argue which part of the code executes EternalBlue.

EternalBlue used a vulnerability in Windows Server Message Block (SMB). The exploit for this vulnerability consists of 3 stages [5]:

1. Craft a list that will create a buffer overflow on the victim machine.
2. In order for the exploit to work the list should overflow into the `Srvnet.sys` buffer. Intuitively this means that the victim machine needs to store the received data in a memory address right before the `Srvnet.sys` buffer. Messages are sent to the victim machine in such a way that the overflow is into the `Srvnet.sys` buffer.
3. The crafted list is sent. On the victim machine, the list overflows into a part of `Srvnet.sys`. This part gets executed when the `Srvnet` connection is terminated, eventually resulting in malicious remote code execution.

We will now look into what part of the code performs the EternalBlue exploit. Note that since we did not fully reverse engineer this part of the code, functions, and variables are often not renamed. The only hint of EternalBlue we found, is linked to the Kaspersky antivirus. We know that EternalBlue only runs when the Kaspersky software is detected [2]. We found that `function_100096c7` performs this Kasperksy check.

```
if (((AntivirusFlags & 4) != 0) && ...)
{
    ...
}
```

This function gets called as part of a thread that is created in the `ordinal_1` function. Exploring the main function of this thread reveals a rabbit hole of function calls. We had hoped that in this rabbit hole, we would be able to find some evidence that SMB was used. The use of SMB would be a good indicator to verify that this part of the code does indeed use the EternalBlue exploit. Sadly, we were unable to find any indication that SMB was used in the given timeframe.

5.4.2 Mimikatz

The other significant way in which NotPetya spreads is by gathering credentials. This is done using a slightly rewritten version of the open-source program Mimikatz [6]. We found the function that is responsible for harvesting credentials in `ordinal_1`. We renamed the function to:
`HarvestCredentialsUsingHarvestingBinary`. Interestingly the function only gets executed when NotPetya already has sufficient privileges. The function scans through memory and extracts login credentials that are stored in plain text.

5.4.3 Combining Forces

Neither EternalBlue nor Mimikatz would be very successful on their own. EternalBlue was already patched by Microsoft by the time NotPetya came around. This meant that a lot of machines would not be vulnerable anymore. Mimikatz on the other hand needs relatively high privileges to harvest credentials. However, the combination of EternalBlue and Mimikatz proved especially powerful. NotPetya was able to spread to non-updated, vulnerable machines using EternalBlue. NotPetya then uses its newly acquired high privileges to gather credentials using Mimikatz. With these credentials, NotPetya could then try to infect machines that are not vulnerable to EternalBlue.

5.5 Encryption Function

NotPetya seems like ransomware on the surface, asking for payment in exchange for the decryption of the files on the computer, however, there have been no reports of anyone paying this ransom and getting their files decrypted. The NotPetya developers did offer to sell the decryption key for 100 Bitcoin [7]. NotPetya uses the `CryptAcquireContextW` of the Windows API to be able to use the cryptographic services on the system. It does this by acquiring the "Microsoft Enhanced RSA and AES Cryptographic Provider". If this fails, it will retry using different parameters for `CryptAcquireContextW`. When this has succeeded, the `generateKey` function is used. It then calls the `encryptFilesUsingKey` function to encrypt the files on the disc. This uses the Salsa20 algorithm to encrypt the files. It sets the boot screen to a file here as well.

6 Discussion

Even though we were able to figure out the global workings of NotPetya, in the given time we were unable to figure out the precise workings of certain functions. One reason for this is our limited knowledge of the C programming language and memory management. At least some of us were unfamiliar with the concept of manual memory management, which made it hard to distinguish between code that managed memory and code that had functionality related to the NotPetya virus. The most interesting functions that would require more work to fully capture their full behaviour are the functions concerning EternalBlue and Mimikatz.

7 Conclusion

For this project, we looked into the main function of the notPetya malware. By reverse engineering this function we were able to establish what NotPetya actually does when it is executed. NotPetya first performs some discovery on the infected machine. At this point, NotPetya gathers information about antivirus software that is present on the machine, and NotPetya attempts to acquire

the privileges it needs to perform its malicious behaviour. After that, we found evidence of how the malware attempts to spread, through the use of Mimikatz and EternalBlue. Last we found the encryption function that actually performs the encryption that renders the file system of the machine of a victim completely useless.

References

[1] Hypr, "Notpetya five facts to know about history's most destructive cyberattack," Jun. 2017, [Accessed 11. Jun. 2023]. [Online]. Available: `https://www.hypr.com/security-encyclopedia/notpetya`.

[2] LogRhythm, "NotPetya Technical Analysis | LogRhythm," *LogRhythm*, Mar. 2022. [Online]. Available: `https://logrhythm.com/blog/notpetya-technical-analysis`.

[3] CarbonBlack, "Carbon black threat research technical analysis: Petya / notpetya ransomware," Jun. 2017, [Accessed 10. Jun. 2023]. [Online]. Available: `https://blogs.vmware.com/security/2017/06/carbon-black-threat-research-technical-analysis-petya-notpetya-ransomware.html`.

[4] A. Greenberg, "The untold story of notpetya, the most devastating cyberattack in history," *Wired, August*, vol. 22, 2018.

[5] Z. Liu, C. Chen, L. Y. Zhang, and S. Gao, "Working mechanism of eternalblue and its application in ransomworm," in *Cyberspace Safety and Security: 14th International Symposium, CSS 2022, Xi'an, China, October 16–18, 2022, Proceedings*, Springer, 2022, pp. 178–191.

[6] ParrotSec, *mimikatz*, [Online; accessed 9. Jun. 2023], Jun. 2023. [Online]. Available: `https://github.com/ParrotSec/mimikatz`.

[7] Thomas Brewster, "'NotPetya' Hackers Demand $256,000 In Bitcoin To Cure Ransomware Victims," *Forbes*, Jul. 2017. [Online]. Available: `https://www.forbes.com/sites/thomasbrewster/2017/07/05/notpetya-hackers-demand-256000-in-bitcoin-to-cure-ransomware-victims/`.