UNDERSTANDING MALWARE

An introduction to the wonderful world of malware

Max 'Libra' Kersten
Security Through Explanation

Abstract

This paper is an introduction into the world of malware. The lay-out of the paper is suited for both readers with and without knowledge in this field. In each chapter, each section gets a bit more technical, allowing the user to fully understand the concept at first, before moving on to the more technical explanation. Each chapter explorers a different part of the whole malware ecosystem. In later chapters, multiple parts are combined to give a deeper insight to malware, and its development, as a whole.

If you have any questions, feel free to e-mail me at info@maxkersten.nl.

Table of Contents

Preface

Malware is forming a threat to all computers, regardless of their updates or the network they are in. Security measures simply cannot stop all malicious programs, especially when the biggest threat to a system's safety is the user himself. Upon opening an infected attachment or clicking on a link to a fraudulent website, a system can already be compromised. If a company gets infected, the consequences can be enormous; if it is a user's home PC getting infected, this user might lose all of his pictures, documents and other memorable files.

Obviously, prevention is better than the need to cure. In the case that you have been targeted or you happen to get infected with malware, knowing how to minimise the damage and clean your systems is valuable knowledge; hence the purpose of this paper. By educating how malware works, how it acts around the anti-virus products and how to analyse what unknown malware does, the created environment will be safer for all users.
To understand why improved security is important for a system to be protected against malware, some recent examples are given below to display the impact of systems that get infected.

Examples

WannaCry

The first piece of ransomware to spread itself using a worm feature. The exploitable service that was used was the Windows SMB service. The exploit used exploit is dubbed EternalBlue. The vulnerability was discovered by the National Security Association (NSA) and leaked by a group of hackers named the ShadowBrokers. Even computers that did not have an internet connection got infected, because the worm moved lateral in the network. One infected computer could infect the rest of the network, regardless if the devices were attached to the internet.

The media reported a lot about this outbreak, even though the earnings of the campaign were marginal. The eventual cashing out was prevented by blocking the bitcoin transaction. The amount of money obtained by the creators remains unknown. The amount of infected devices is hard to measure, if possible at all. The outbreak was all across the globe, so the total amount of infections ranges in the couple of hundred thousands, if not more. The malware contained an, accidental, kill switch. After this switch was triggered, the global outbreak stopped. This kill switch was no cure, but it helped tremendously to reduce the amount of new infections. (Sherr, 2017) (Urbelis, 2017) (US Government, 2017) (Gibbs A. H., 2017)

NotPetya
Starting in Ukraine, the NotPetya ransomware spread itself from the MeDoc accounting software using the EternalBlue exploit, the same as the one used by WannaCry. All the users of MeDoc had the malware on the system as it was pushed via an update of the accounting software. Moving laterally, the infection spread rather fast throughout dozens of companies. Similar to WannaCry, the impact was noticed all around the world. Companies that got infected lost tens of thousands of dollars, if not more, due to corrupt systems. The malware had some sort of kill switch to avoid the infection. This switch was published soon after the outbreak started. Alas, the kill switch was local on a machine. If a certain file existed on the machine, the malware would stop executing. Because the kill switch was checked for each machine, there was no global prevention. (Bisson, 2017) (Arghire, 2017) (Cimpanu, Vaccine, not Killswitch, Found for Petya (NotPetya) Ransomware Outbreak, 2017)

Ransomware in the San Francisco Metro systems
During the Thanksgiving weekend in 2016, customers could travel freely with the metro for two days due to a ransomware infection on the servers. The ransom was 100 Bitcoins, which equalled to $73,086 at that time. In total, 2112 computers were infected and 30 GB of data was allegedly compromised. The ransomware creators threatened to leak the data if the ransom would not be paid. (Williams, 2016) (Gibbs S. , 2016)

ATM malware updated for Windows 10
Using an external keyboard or an SMS message, the ATM malware enabled the attacker to obtain thousands of dollars in mere minutes. It targeted Diebold ATM machines, an ATM vendor which runs on the Kalignite Platform. Changing a small part of the code, the malware's attack surface could increase to 40 different ATM vendors located in 80 different countries. (Regalado, 2017) (Beltov, 2017)

Lesson learnt
As shown in the examples above, malware is an imminent threat to all sorts of systems. Older malware can be detected by their signature, but signatures of new malware are unknown. To detect new or modified malware, their behaviour is analysed with a heuristic analysis. Even updated systems are vulnerable, even though a patched system is harder to infiltrate. To aid in the detection, suspicious files can be analysed by manual tooling. The next chapters will discuss the creation of the malware, the attack vectors, basic analysis methods, the hardening of the malware and advanced analysis methods used by anti-virus suites.

Types of Malware

Even though every piece of malware is unique, the programs fall into different categories. In the table below, multiple sorts of malware will be discussed. (Lord, 2012) (Barraco, 2013) (Kevin Savage, 2015) (SentinelOne, 2016)

CATEGORY	DEFINITION
ADWARE	Adware shows unwanted advertisements to the user on the infected system. Adware also has the potential to track a user's system. Some adware is close to spyware; sometimes the two are intertwined. Adware is often bundled with free or illegal software and installed unnoticed.
BOTS	Bots are pre-programmed to execute certain tasks. Some bots are used to transform the infected system into part of a botnet. A botnet consists of slaves (or zombies) which are controlled by a Command & Control server (C&C). When a system is infected, the bot awaits orders from the C&C server to execute the pre-programmed actions.
RANSOMWARE	Ransomware locks the system until a ransom is paid. All the file shares of the computer and those accessible by the computer will be encrypted, excluding the essential files for the operating system. There are two different types of ransomware: one that locks the system completely and one that locks the user's files. The currency of the ransom is usually Bitcoin, to make the transaction as anonymous as possible. To pressure payment, the file system is often deleted within a given timeframe if the ransom is not paid.
BACKDOORS	A backdoor is a hidden way to access a system for the attacker whilst staying under the radar. Using this backdoor, the attacker can navigate through the system and perform actions depending on the gained privileges.
ROOTKITS	A rootkit is often referred to as a backdoor, albeit a special kind. Whereas backdoors are only used to access a system, a rootkit has the option to execute tasks with the root privilege.

SPYWARE	Spyware spies upon the user. The functions of spyware include – but are not limited to – logging keystrokes, collection information on the connected drives, or monitoring browser activity. Spyware is often bundled with free software and installed unnoticed.
TROJAN HORSE	A Trojan horse obtained its name from the city in the Greek Epos "The Trojan War", in which the horse unloaded soldiers in the city of Troy unnoticed. (University, 2008) Pretending to be a normal program, the Trojan horse has a hidden agenda. The payload of the malware could be any other type of malware to let the attacker gain access to the system or exploit a service.
VIRUSES AND WORMS	Viruses and worms are self-spreading programs with the ability to collect information or execute tasks. The main difference between worms and viruses is worm's the ability to replicate without interaction of a user.
FILE-LESS MALWARE	An existing process is injected with a malicious code during runtime; the code is then executed from the memory. The file of the hijacked process is not altered nor is the malicious program persisted on anywhere but in the memory. A reboot is enough to remove the file-less malware, unless the malware infected other persisted files.

Attack Vectors

Malware needs permissions to perform actions, similar to the permissions a user needs to perform the same actions. An attacker has multiple options, which are often referred to as vectors, to attack. The goal of an attack differs per vector. Each vector yields a different result for the attacker. Not every possible exploit technique is discussed for each vector. The unique component is explained, together with a short explanation on what each vector is. (nhinkle♦, 2010) (Atwood, 2008)

ATTACK VECTOR	DEFINITION
PROCESSES	A process is an instance of an executed binary. Examples of processes are Google Chrome, Microsoft Word or IceWeasel. A process is initiated by a user. To exploit a process, the stack can be smashed, the code flow can be altered or the binary can be edited. The result of the alteration is the execution of shellcode to execute a given payload.
SERVICES	Similar to a process, a service is an executed binary. The difference between a service and a process is the fact that a service is not initiated by a user. Therefore, a service can also run without any user being logged in on the system. Services are generally not directly graphically visible in the desktop environment. Services can be used to execute shellcode, but also to spy upon all users on a machine, instead of one user. Important to note is the fact that the operating system also has services which are running in the background. These services are attack vectors as well, but are categorised in the category 'Operating System' below. The services, as described in here are services from installed programs.

OPERATING SYSTEM	The CPU has two modes: kernel and user. In the kernel mode, there are no restrictions. The user mode cannot directly access the hardware or create references in the memory. To access the hardware or create a memory reference, a kernel API is used, to delegate the tasks to kernel level. Upon gaining access to the kernel, objects can be made which cannot be directly removed by a user initiated process; this evades most anti-virus suites.

Attack Techniques

As explained above, there are multiple vectors which can be exploited. Every vector has an attack surface which can be used, meaning that there are multiple ways to breach the system's security. The most common techniques will be explained in this chapter. (Piscitello, 2016) (Doshi, 2006) (QNX, n.d.)

ATTACK TECHNIQUE	DEFINITION
PRIVILEGE ESCALATION	To fully control the system the malware is on, the highest privilege is a necessity. With the permissions of this privilege, files and settings can be altered without any notice to the users on the system. Escalation can happen in two ways: horizontally and vertically. The first way is to compromise accounts with the same permissions; the latter is to compromise an account with more privileges. Getting *root* or *administrator* privilege is the ultimate goal, since it grants complete access to the whole system.
WEB APPLICATION EXPLOITS	Websites can contain vulnerabilities which give access to the database, administrator panels or even complete control of the web server. Remote code execution, SQL injections, Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) are commonly found examples.
LOCAL EXPLOITS	A local exploit is an exploit that is executed on a computer. Examples of local exploits are malicious e-mail attachments, rogue USB flash drives or other files on network shares.
REMOTE EXPLOITS	Remote exploits can be executed without any previous interaction with the system, unlike local exploits. These include Metasploit, Immunity CANVAS or single exploits such as the ones posted on Exploit-DB.
DENIAL OF SERVICE	A service, which can be a program or an operating system, can be stopped or crashed. A Denial of Service (DoS) attack forces the service to stop. A SYN-flood – another form of denial of service – is used to flood a target with requests. The target is not able to cope with so many requests and comes to a halt.

DISTRIBUTED DENIAL OF SERVICE	A distributed denial of service (DDoS) is a coordinated DoS attack to one target. Every machine has an IP address: upon flooding a target, the abnormal behaviour is flagged and potentially blocked. This would stop the attack. By distributing the attack over thousands of machines, the server cannot stop the flood by blocking one connection. Blocking every machine that sends a request to the target would also deny legitimate users access to the target. To prevent damage from the attack, services such as CloudFlare act as a server in between and delay the requests for some seconds. This provides breathing room for the target and stops the denial of the service, even though the servers might experience a heavy load.

Payloads and the infection process

The payload is the reason why the malware was created in the first place. The delivery of the payload often compromises the system's integrity. There are no set 'concepts' for payload modules, unlike the attack techniques. Often, the malware will create a 'hole' in the system that is used to re-enter the system after the payload is executed.

At first, the exploit will achieve administrator (or root, depending on your operating system) privileges, minimising the privilege boundary set by the operating system and increasing the chances of correctly executing the payload. Secondly, the malware will make sure it is persisted on the computer and executed during or after the boot of the operating system. Lastly, the malware will create an opening for the attacker to communicate with. This can be anything, from sending a message to the command and control server, to creating a fully functioning backdoor with FTP or shell access.
Different malware families and samples behave differently, especially ransomware. Ransomware simply encrypts the files in the user folders in order to blackmail the owner to pay for the decryption of his/her files. Zeus, a banking Trojan, injected itself in the web browser to obtain credentials of bank accounts. This way, the bank account of the victim could be used to transfer money to the attacker's bank account. These two examples are two commonly used approaches, yet there are an unlimited number of approaches possible. (Malwarebytes, 2017) (Cimpanu, Ransomware Was the Most Prevalent Malware Payload Delivered via Email in Q2 2017, 2017) (TechoPedia, 2017) (Tend Micro, 2015) (Kaspersky, 2017)

Anti-virus suite architecture

On the desktop platform, the read and write permissions for folders are managed by Access Control Lists. These lists contain read, write and execute permissions per file and folder or user group, depending on the operating system. Limited users can read, write and execute their own files in their own folders, whereas the administrator/root user has these permissions for all files in all folders. (RedHat, 2017) (Microsoft, 2017)

On mobile platforms, applications are separated from each other. This provides more security by default, because malicious applications are unable to access the content or memory of other files. Alas, a new problem emerges due to this set up: the anti-virus suite(s) can't access the memory of a malicious application, because both are running with the same privilege. The root user for normal applications on Android is turned off by default. Turning it on (rooting the device) would mitigate this negative impact on the anti-virus, but it would also allow the malware to obtain similar permissions and access any part of the memory of any application to infest itself in, including the anti-virus suite.

Updating the signature database

If the anti-virus suite is evaded, the malware can hide itself in, depending on the privileges of the malware, nearly any process or file. This makes the search for malware on the computer after the infection rather difficult. Because programs are not executed in a sandbox, exploits in one program can lead to a complete takeover of the computer. A drive by download in a browser can lead to a backdoor. This backdoor can be used to launch an exploit that gives administrator privilege to the attacker. Nowadays more and more programs use a sandbox, but not all of them are efficient or hard to fool.

Due to the limitations created by the operating system, the anti-virus scanners are set back in time. The result of this setback is traceable to the desktop environment in the earlier days of malware development: signature detections. To detect signatures, the signature of the analysed malware by the analyst needs to be added to database. This approach creates two problems.

The first problem is the fact that the malware analyst is always too late with the analysis. People have already been infected. Even when the malware analyst is fast and minimises the time it takes to analyse the sample and add the signature to the database, there will be delay.

The second wave of delay is on the user's end. When the signature database gets updated, the clients need to download the additional signatures before they are safe against the latest analysed threads. In the early days of the Internet on desktops, when people paid per megabyte and usually had to decide to either internet or call, the delay was long.
After the Internet matured, computers are nearly permanently online. Mobile phones however, are not as advanced. Whereas data has become cheaper of the years, mobile phones are not always connected. Another problem with mobile phones regarding the signature updates, is data usage. Even though a lot of phones have 3G or 4G, not everybody has a data limit without a limit. Therefore, updates from programs are often postponed unless the user is connected to the Wi-Fi. In the meantime, the mobile phone is connected to the internet and applications can be downloaded on the telephone. This gap in time between the release of the update and the installation of the update is hard to prevent for anti-virus suites.

Self-modifying malware

Using a language with direct access to the memory that is used, the program can modify the contents of its own memory. An example of such a language would be C. If one would dump the memory used by a piece of self-modifying malware, shut the malware down and repeat the process, the result would be two different dump files. This means that scanning the memory of the malware is not useful anymore unless the program uses a specific algorithm or has other detectable components. Recompiling the program as another program with slightly different content, the hash of the malware is also changed. Starting the newly compiled malware and deleting the old malware would remove some of the evidence that the previous version was executed on the system. (Dalasta, 2011)

A lot of detection mechanisms of anti-virus suites are negated using self-modifying malware. The biggest reason why it isn't used on a broad scale, is the complexity to create such a piece of malicious software. Even if a malicious engineer has the knowledge, the return of investment of the malware is not worth it. The simple techniques continue to work because users are still the weakest link.

Malware Architecture

Knowing how malware operates and what vectors are targeted, a piece of malware can be developed to exploit a system. When a program is compiled, the code gets converted into assembly language. Using a decompiler, the original source can (partially) be retrieved unless the software has been obfuscated, packed or encrypted. Otherwise, the decompiled output might be in the language of the original code, but the code can be messy and not fully decompiled. In this chapter, the architecture of malware and common disguise techniques will be discussed.

Architecture

In the early days of malware, the program was spread and anti-virus suites would perform analysis based on file signatures. The signature of a file is unique, so the file system can scanned using this technique. To scan, the anti-virus suite starts by generating signatures for files on the computer and comparing them to the signature database. If there is a match, the file is malicious, if not, the file is not known. This detection method, also known as blacklisting, is too slow. Only already known malware can be detected. To prevent and evade detection, malware creators thought of different ways to stay undetected. (Schiffman, A Brief History of Malware Obfuscation: Part 1 of 2, 2010) (Schiffman, A Brief History of Malware Obfuscation: Part 2 of 2, 2010) (Yim, 2010)

Encrypted malware

Using encryption to hide the inner functionalities of the malware avoids detection by anti-virus suites. Using a different generated key in the algorithm, the signature differs for each version. This evades the anti-virus' signature detection. The encrypted asset is decrypted during runtime and the malware is then executed. The problem with this technique, however, is the decryption method. Upon decompiling the malware, the decryption function can be retrieved. The anti-virus suite can use this technique to decrypt files during the scan, revealing the inner workings of the malware. This leads to the detection of the malware.

Oligomorphic malware

To mitigate the negative effects of the encryption method, oligomorphic malware was introduced. The difference with the encryption method is small: the decryption function is changed. The amount of possibilities for the decryption function was, however, limited: only a couple of hundred options were available.

Polymorphic malware

In return, polymorphic malware was an improvement because of the countless possibilities of the decryption function. This removed the possibility to defeat the encryption, since every version had another decryption method. The anti-virus companies reacted on this by using a technique called ‘sandboxing’. The malware is executed in a virtual machine. The signature of the decrypted malware is located in the memory during runtime. The signature of this malware in the memory is compared with the signature database. Other factors started to come into play: suspicious behaviour would also be flagged. Based on these two, the scanned files would be classified as either legitimate or malicious.

Metamorphic malware

The difference between polymorphic- and metamorphic malware is small, yet important. To avoid the sandbox signature detection, not only the decryption method is changed, but also the inner workings of the malware. This creates a new signature for every version, letting the malware spread undetected.

Obfuscation

To prevent reverse engineers from analysing the malware, malicious coders often use a technique called obfuscation. Simply put, the code is reformed and scrambled to make it unreadable for humans, but readable for a computer. In the rest of this chapter, specific techniques will be described and analysed. (Yim, 2010) (Schiffman, A Brief History of Malware Obfuscation: Part 1 of 2, 2010) (Schiffman, A Brief History of Malware Obfuscation: Part 2 of 2, 2010)

Dead code insertion

To prevent analysis, the code can be hidden in plain sight by simply adding thousands of lines of garbage code. This code distracts the attention of the reverse engineer, who needs to find out what the inner working of the malware is. This technique is based on the known technique ‘hiding in plain sight’.

Reordering subroutines

Changing the order of the program also changes the flow of the program. Assuming that a program has *n* subroutines, the total amount of possibilities equals *n!* (*n* factorial). If each version has a different function flow, the reverse engineer has to find the structure again in each sample.

Control flow obfuscation

Obfuscating the control flow of a program can cause a failure upon decompilation. Simply put, the amount of calls between functions and options are too much to decompile, crashing the decompiler. This results in the failure to read the code of a function, or even a whole program. Instead of the control flow graphs and pseudo code functions, the reverse engineer is forced to work with assembly, which is a lot more time consuming.

Packers

Often also referred to as ‘loaders’ or ‘droppers’, loaders are programs which contain a program in its resource file. During runtime, the *loader* decrypts the actual program and loads it into the memory. This technique is often used to prevent anti-virus inspection, because the loader is not considered malicious by the anti-virus. The other file the anti-virus can see, is the encrypted asset, which is not malicious either. A *dropper* downloads the malicious binary from the web. This way, the anti-virus can not even scan the malicious binary before the execution, because it is not on the device. The *dropper* could download a *loader*, which executes the actual malware. (Schiffman, A Brief History of Malware Obfuscation: Part 2 of 2, 2010) (Lau, 2012) (Malwarebytes Labs, 2016)

Encryption

Frequently used to avoid static detection, encryption remains a strong way to prevent analysts from taking a peek at the malware. Generally, there are two types of encryption found in malware.

The first way is the creation of an algorithm by the author self. Often, the idea behind this is simple: *"If I use an unknown algorithm, it will be really hard to figure out what is being done during the encryption and decryption processes"*. Alas, this is a fallacy. The new technique hasn't been tested in the wild and often contains weaknesses which allows the reverse engineer to fully decrypt the data. Using a modern encryption standard, such as 256-bit AES encryption, is the second approach. This technique is rather hard, if not impossible, to beat, if it is correctly implemented. Luckily for security researchers, malware authors are quite often mistaken and misconfigure the malware's encryption. (Delrue, 2016) (Arora, 2012)

Anti-anti-virus measures

Evading an anti-virus suite is surely possible using certain techniques. Depending on how big of a threat the malware opposes out in the wild, the anti-virus makers respond adequately. Avoiding heuristic analysis can become a complex task, especially if one is writing all the code without using an automated tool. Writing these steps without said tool, the chances to avoid detection increase, as does the required technical knowledge for the author. (Bachaalany, 2015)

Anti-virtualisation techniques

Often, malware is analysed in virtual environments because a single computer can analyse a malware sample, regardless where the reverse engineer is: in an airplane, at work or in the train. With the use of a physical machine, the reverse engineer would need an additional machine to execute the sample on. To avoid analysis, the malware can try to detect if the environment in which it is executed, is a virtual one. A few options to detect a virtualised system could be to check the name of drivers, the size of the hard disk drive, the movement of a mouse, the key presses on a keyboard or checking if the sleep function is patched.

Evading a virtualised system makes analysis harder and delays the time before the sample is analysed. This, in turn, delays the time until the disinfectant routine is created by the anti-virus company which increases the timespan in which the malware is undetected.

Anti-reverse engineering techniques

Evading a virtual system aids in evading mass and/or automatic detection, it is still possible for a reverse engineer to analyse the malware. Causing a decompiler to crash or malfunction would make the analysis harder, if not impossible depending on the knowledge of the analysist.

A technique that is commonly used is the attachment of a debugger to the malware by the malware as a first action. Since a program can only be debugged once, the reverse engineer can't debug the malware without patching the first instructions. Other techniques that are often used are the obfuscation techniques that have been described earlier.

Anti-sandbox techniques

A sandbox is an, often temporary, environment created by the anti-virus suite, in which a program is executed just before it is executed on the physical machine. The anti-virus suite analyses the behaviour of the program in mere seconds and deems the executable either safe or unsafe. If the outcome of the analysis is unsafe, the user will receive a warning and the file will be placed in quarantine. If the result is safe, the program will be executed on the machine.

In the early days of malware development, the sandboxes executed the instructions as if they were ran on a normal computer. The abuse of the sleep function alone, was sufficient to evade the sandbox. To avoid this, the sleep function was sped up in some sandboxes. Nowadays, processors have become increasingly fast, meaning that a second of time in the sandbox can handle more instructions than was possible before. The instructions of most programs, especially malware, have increased less than the speed with which the processor executes instructions, favouring the outcome of sandbox test in the anti-virus' perspective.

Other known techniques to evade the sandbox is to call upon non-existing files or use barely ever used instructions. Calling upon a non-existing file would give a handle in the memory to the program. A sandbox does not have time to actually load the file, so a fake handle is created. If the program would call a file that it knows does not exist, any given handle would confirm the existence of a sandboxed environment. Since the sandbox is an emulated environment, only common instructions are implemented. Instructions that are barely used because of their age or inefficiency will not be implemented. Calling such an instruction will cause the sandbox to malfunction and result in a time out of the sandbox. The program is then free to executed on the normal machine.

Difference between virtualisation and sandboxing

Even though a sandbox is a virtualised system, there is a subtle difference in usage. A virtualised system, such as a virtual machine, is a complete operating system which allows the user to interact dynamically with the system. A sandbox is a closed environment which does not allow input from the user. The environment in the sandbox also differs from the normal virtual machine. Certain functions might be altered, such as the sleep function. The sleep instruction(s) might be skipped to avoid a sleep trap. Upon requesting a library, the handle that is passed to the program is an emulated one, to avoid the loading time.
These differences are the reason that both systems, though technically the same, are explained as if they are different.

About the author

My name is Max ‘Libra’ Kersten and I have an interest in offensive security. To stop current and future attacks, one should think like an attacker. The anti-virus and anti-malware branches will always be too late to stop malware from infecting users. By anticipating what the malicious coders might do, the delay can be minimised.

Acknowledgements

I'd like to thank Pham Duy Phuc for the clarification he's given me whenever I had a problem with a technical aspect. Additionally, I'd like to thank H. Cao for reviewing the English language in the early stages of the paper.

Bibliography

Arghire, I. (2017, 06 29). *NotPetya - Destructive Wiper Disguised as Ransomware*. Retrieved from Security Week: http://www.securityweek.com/notpetya-destructive-wiper-disguised-ransomware

Arora, M. (2012, 07 05). *How secure is AES against brute force attacks?* Retrieved from EETimes: http://www.eetimes.com/document.asp?doc_id=1279619

Atwood, J. (2008, January 3). *Understanding User and Kernel Mode*. Retrieved from CodingHorror: https://blog.codinghorror.com/understanding-user-and-kernel-mode/

Bachaalany, J. K. (2015). *The Antivirus Hacker's Handbook.* Indianapolis, Indiana: John Wiley & Sons, Inc.

Barraco, L. (2013, December 10). *What are the most common types of malware?* Retrieved from AlienVault: https://www.alienvault.com/blogs/security-essentials/what-are-the-most-common-types-of-malware

Beltov, M. (2017, January 13). *New Variant Ploutus Malware Identified*. Retrieved from BestSecuritySearch: https://bestsecuritysearch.com/new-variant-ploutus-malware-identified/

Bisson, D. (2017, 06 28). *NotPetya: Timeline of a Ransomworm*. Retrieved from TripWire: https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/notpetya-timeline-of-a-ransomworm/

Cimpanu, C. (2017, 08 17). *Ransomware Was the Most Prevalent Malware Payload Delivered via Email in Q2 2017*. Retrieved from BleepingComputer: https://www.bleepingcomputer.com/news/security/ransomware-was-the-most-prevalent-malware-payload-delivered-via-email-in-q2-2017/

Cimpanu, C. (2017, 06 27). *Vaccine, not Killswitch, Found for Petya (NotPetya) Ransomware Outbreak*. Retrieved from BleepingComputer: https://www.bleepingcomputer.com/news/security/vaccine-not-killswitch-found-for-petya-notpetya-ransomware-outbreak/

Dalasta, D. (2011, 11 21). *Writing Self-modifying Code Part 1: C Hello world with RWX and in-line assembly*. Retrieved from Infosec Institute: http://resources.infosecinstitute.com/writing-self-modifying-code-part-1/

Delrue, G. (2016, 12 26). *Was AES-256 cracked or not?* Retrieved from Quora: https://www.quora.com/Was-AES-256-cracked-or-not

Doshi, S. S. (2006, April 27). *Five common Web application vulnerabilities*. Retrieved from Symantec: https://www.symantec.com/connect/articles/five-common-web-application-vulnerabilities

Gibbs, A. H. (2017, 05 12). *What is WannaCry ransomware and why is it attacking global computers?* . Retrieved from The Guardian: https://www.theguardian.com/technology/2017/may/12/nhs-ransomware-cyber-attack-what-is-wanacrypt0r-20

Gibbs, S. (2016, November 28). *Ransomware attack on San Francisco public transit gives everyone a free ride | Technology | The Guardian*. Retrieved from The Guardian:

https://www.theguardian.com/technology/2016/nov/28/passengers-free-ride-san-francisco-muni-ransomeware

Kaspersky. (2017, 09 06). *Zeus Trojan Malware*. Retrieved from Kaspersky: https://usa.kaspersky.com/resource-center/threats/zeus-trojan-malware

Kevin Savage, P. C. (2015, August 6). *The evolution of ransomware*. Retrieved from Symantec: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf

Lau, H. (2012, 04 26). *Trojan.Dropper*. Retrieved from Symantec: https://www.symantec.com/security_response/writeup.jsp?docid=2002-082718-3007-99

Lord, N. (2012, October 12). *Common Malware Types: Cybersecurity 101*. Retrieved from VeraCode: https://www.veracode.com/blog/2012/10/common-malware-types-cybersecurity-101

Malwarebytes. (2017, 09 06). *Payload*. Retrieved from Malwarebytes: https://blog.malwarebytes.com/glossary/payload/

Malwarebytes Labs. (2016, 06 9). *Trojan Dropper*. Retrieved from Malwarebytes: https://blog.malwarebytes.com/threats/trojan-dropper/

Microsoft. (2017, 04 11). *Access Control Lists*. Retrieved from MSDN: https://msdn.microsoft.com/en-us/library/windows/desktop/aa374872(v=vs.85).aspx

nhinkle♦. (2010, November 11). *What's the difference between an Application, Process, and Services?* Retrieved from SuperUser: https://superuser.com/questions/209654/whats-the-difference-between-an-application-process-and-services

Piscitello, D. (2016, February 18). *What is Privilege Escalation?* Retrieved from ICANN: https://www.icann.org/news/blog/what-is-privilege-escalation

QNX. (n.d.). *Remote and local attacks*. Retrieved January 24, 2017, from QNX: https://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.neutrino.user_guide%2Ftopic%2Fsecurity_Remote_Local.html

RedHat. (2017, 04 11). *Chapter 20. Access Control Lists*. Retrieved from RedHat.com: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/ch-acls.html

Regalado, D. (2017, January 11). *New Variant of Ploutus ATM Malware Observed in the Wild in Latin America*. Retrieved from FireEye: https://www.fireeye.com/blog/threat-research/2017/01/new_ploutus_variant.html

Schiffman, M. (2010, 02 15). *A Brief History of Malware Obfuscation: Part 1 of 2*. Retrieved from Cisco: https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2

Schiffman, M. (2010, 02 22). *A Brief History of Malware Obfuscation: Part 2 of 2*. Retrieved from Cisco: https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_2_of_2

SentinelOne. (2016, May 11). *Malware and Exploits: An Introduction to Two Prominent Attack Vectors*. Retrieved from SentinelOne: https://www.sentinelone.com/wp-content/uploads/2016/05/MalwareVsExploit_WP_051116.pdf

Sherr, I. (2017, 05 19). *WannaCry ransomware: Everything you need to know*. Retrieved from CNET: https://www.cnet.com/news/wannacry-wannacrypt-uiwix-ransomware-everything-you-need-to-know/

TechoPedia. (2017, 09 06). *Payload*. Retrieved from Techopedia: https://www.techopedia.com/definition/5381/payload

Tend Micro. (2015, 06 24). *ZEUS*. Retrieved from Tend Micro: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/zeus

University, M. A. (2008, May 13). *5. The Trojan War*. Retrieved from Epos: https://epos.wordpress.com/epic-poetry-lessons/5-the-trojan-war/

Urbelis, A. (2017, 05 14). *WannaCrypt ransomware attack should make us wanna cry*. Retrieved from CNN: https://edition.cnn.com/2017/05/14/opinions/wannacrypt-attack-should-make-us-wanna-cry-about-vulnerability-urbelis/index.html

US Government. (2017, 05 19). *Indicators Associated With WannaCry Ransomware*. Retrieved from US-CERT: https://www.us-cert.gov/ncas/alerts/TA17-132A

Williams, C. (2016, November 27). *Passengers ride free on SF Muni subway after ransomware infects network, demands $73k*. Retrieved from The Register: https://www.theregister.co.uk/2016/11/27/san_francisco_muni_ransomware/

Yim, I. Y. (2010). *Malware Obfuscation Techniques: A Brief Survey.* Retrieved from SemanticsScholar: https://pdfs.semanticscholar.org/0401/3804edf3e86218d15868269a355dd3501c0f.pdf