Internship Report

Université de Lorraine, 1ère année de Master Informatique

From 2025-06-02 to 2025-07-25

Carbone Team,
Laboratoire Lorrain de Recherche en Informatique et ses Applications

Introduction

The main goal of this internship was to improve the post-processing of *Goatracer*, a dynamic malware analysis platform, in order to find informations about the network communications made by analysed malware.

This resulted in the creation of a Python script, which was then ported as a module of *Goatracer*. The module shows satisfying results, but has some limitations.

This document describes the methodology and work realized during the internship, as well as the difficulties encountered.

Acknowledgements

Fist of all, I want to thank my tutor Jean-Yves Marion for giving me this internship subject and trusting me with an important part of his project.

I also want to thank Pierre Marty, who has always been helpful, kind, and motivating during my time at Carbone, as well as Fabrice Sabatier for dealing with my frequent requests, always with a smile.

Finally, I extend those thanks to everyone in the Carbone Team who have been very nice and welcoming, and made this internship a valuable experience.

Contents

1 Host Structure Presentation

The Carbone Team is a public research team specialized in the study of malware (**Mal**icious soft**ware**) and their ecosystem. It is part of the LORIA (Lorraine Research Laboratory in Computer Science and its Applications), a laboratory shared by researchers from CNRS, INRIA and Université de Lorraine.

The team focuses on three different aspects related to malware:

- Defence against malware, using reverse-engineering and analysis techniques
- Understanding offensive tactics, by building attack chains and other attacking examples
- Studying malicious ecosystems, including organisations and underground economies

As part of the DefMal project (PEPR Cybersecurity - France 2030 - ANR), the team has been working for several years on malware-related projects including the LHS (Laboratoire de Haute Sécurité, high-security laboratory) [1], providing secure servers for teams at LORIA, as well as *Goatracer*, a dynamic malware analysis tool (itself running on the LHS servers).

2 Context and Goals

There are different types of malware which have different goals and inner workings.
However, one recurring feature of malware is the use of networking on infected devices.
Network capabilities can be used to infect other computers, send denial-of-service attacks as part of a botnet, or communicate with a server to send information or receive commands, in which case we call this server a *Command and Control* server, often abbreviated as a *C2* server.

In the context of defense against cybercriminals, understanding how a given malware works is essential to build better security. As part of that work, being able to understand how and where a malware communicates to other devices is a major asset.

In oder to analyse a malware, there are two major techniques:

- Static analysis, in which we disassemble the malware from binary code, and the resulting assembly code is then examined in order to understand how it works.
  This technique does not require any execution and is mostly safe. However, assembly code is in most cases verbose and complex, and very difficult to understand from a human point of view.
- Dynamic analysis, in which we execute the malware (in most cases inside of a virtual machine) and examine the machine behaviour in order to understand how it is being used. This latter technique allows for more informations to be retrieved, but requires a dedicated analysis environment and tools such as debuggers. Dynamic analysis also only retrieves informations about the executed parts of the code, which can be a blessing and a curse, as this skips parts of the code that may be unused but also others that might trigger on a specific context (e.g. a specific argument or hardware).

As of today, dynamic analysis is still the easiest way to analyse binaries that perform self-modifications during runtime. For instance, if a program has encrypted sections, we can access parts of the decrypted assembly code by executing it and waiting for the program to decrypt itself. This is also useful when the binary is compressed with the help of *packer* tools.

The Carbone team has been working for multiple years on a new dynamic analysis tool called *Goatracer* [2], where a user can deposit a PE file, the main executable Windows-exclusive file format, on a web platform. This file is then executed inside of a virtual machine. Then, using techniques of instrumentation (injecting assembly code in the executed file) and introspection (examining the virtual machine from the host perspective), the tool is able to retrieve informations about the assembly code being executed, modifications of the code by itself (which we call waves [3]) and calls to system library functions with their arguments (which we call *syscalls*). This last information is extremely valuable for analysis.

The main objective of this internship is to develop new ways to interpret the output of *Goatracer*, so that new information can be extracted from the analysed file.
We focused mainly on the analysis of network-related information, so that we may be able to find informations such as IP addresses, URLs and the protocols used in the malware analysed by *Goatracer*.

3 Methodology

The main source of network-related informations from *Goatracer* is the list of *syscalls*. Most modern Operating Systems handle the network stack, and offer librairies to allow user-space programs to interact with network features. As such, if we intercept network-related *syscalls* as they are being sent out from the malware to the operating system, we can retrieve a lot of information that can be read from the arguments.
For example, if a malware wishes to send a ping to an address on the Windows OS, it would use the `IcmpSendEcho` system call, which itself requires an IP address string being passed as argument. Using *Goatracer*, we can see this variable being passed as plain text.

As the virtual machines used by *Goatracer* do not have network capabilities yet, network capture was not a considered option. This will be discussed further in Section 5.2.

3.1 General Idea

The first approach we considered to retrieve IP addresses from the list of system call arguments was to use Regular Expressions, allowing for the search of strings formatted in a certain manner. However, while this technique is simple to implement, it is very sensitive to any obfuscation technique being used and does not allow for the retrieval of the protocol being used. Moreover, there are some performance scalability concerns for very large execution traces. As such, this idea was set aside.

The second approach, which we chose to implement, is instead based on the exhaustive enumeration of all first-party (i.e. not external to windows) *syscalls* related to the network stack. Our reason for excluding third-party libraries is that they also have to rely at some point on Windows *syscalls* to handle their communications, therefore we will capture those in any case. Once we have this list of *syscalls*, we know what every one of them does, and what their arguments correspond to. We can then establish a mapping between function names and a list of argument positions, allowing us to retrieve a specific type of arguments (in our case, the addresses.)

3.2 Listing *Syscalls*

To create this list of network-related *syscalls*, we decided to proceed in multiple steps.

1. We created "homemade" binaries using different basic Windows network libraries (winsock, winhttp, wininet) in order to gain familiarity with the format of the arguments being output by *Goatracer* and list some basic *syscalls* with interesting arguments.
2. With the help of the Network TTPs (Techniques, Tactics and Protocols) listed in the MITRE ATT&CK tool [4], we looked for Windows libraries which could be used to interact with certain protocols commonly used by malware. After that, using the Microsoft documentation for those libraries, we were able to list many more *syscalls*.
3. To complete the list with other *syscalls* we missed, we ran *Goatracer* on binaries from the MOTIF malware dataset [5] to manually confirm those programs didn't use anything else to communicate with other devices on the network.

The final list of *syscalls* appears to be pretty complete, but we cannot guarantee at this moment its exhaustivity, as we may find new syscalls we missed during this process.

3.3 Examining Command-Line Network Capacities

One interrogation remained after the listing process. While *syscalls* are a common manner of communicating with the OS, a malware may theorically be able to interact with the network

stack using commands through the command prompt (cmd.exe) or Powershell. While this seemed like a valid concern at first, we were not able to observe any malware having this behaviour. Moreover, there are limited command options on desktop Windows, as tools like `telnet` are disabled for security reasons and others like `ssh` are simply not part of the OS by default. We still found that the system call `GetCommandLineW` was always present in the case of a command execution (for example with the use of the `system` function in C), allowing us to retrieve the content of the command. We also investigated a few methods of system call obfuscations [6], [7] in case we ever need to start retrieving and deobfuscating command line arguments.

4 Implementation

4.1 Creation of a Prototype Extraction Script

The Carbone team has made a python program designed to help the extraction of informations from the result given by the platform (*syscalls*, number of waves, assembly code of waves…) [8] We decided to first implement our network information extraction approach as an extension of this program. After transcribing the list of *syscalls* with their types and argument indexes as a python dictionary, we are able to read the Json file of the Goatracer results and print out the relevant information to standard output, as shown in the Listing 1 example.

```
Found a call to InternetCrackUrlA from WININET.DLL (http). Network-related
arguments:
_IN_ (LPCTSTR) [0x0017ed64] "http://account.protonvpn.store/index.php" (lpszUrl)

Found a call to InternetConnectA from WININET.DLL (http). Network-related arguments:
_IN_ (LPCTSTR) [0x0017ed58] "account.protonvpn.store" (lpszServerName)
_IN_ (DWORD) [0x0017ed5c] 0x00000050 (nServerPort)

Found a call to connect from WS2_32.DLL (tcp/udp). Network-related arguments:
_IN_ (sockaddr*) [0x0017e770] 0x0017eba0 (sockaddr* name)

----
Types of network-related syscalls found:
http
tcp/udp

----
IPs found:
account.protonvpn.store
```

Listing 1: Output of the script on MOTIF_0ae37532a7bbce

4.2 Interpretation of the Extracted addresses

Once we retrieve the IP addresses and URLs, we are able to query APIs in order to get more informations that may be used to further investigate the malware.
We used the IP-API service [9] because it offers an API-keyless way of querying informations. Another option was AbuseIpDB [10] which offered a user-alimented report database, a great resource in our case, however this one required an API key, so we decided to set it aside in the case of a client-side script (as requiring the script user to create an API key was unreasonable). If the queried address is an URL, the service will resolve the name to its current IP attribution. We end up with informations about the country of origin, the ISP (Internet Service Provider), and some flags: Mobile for mobile data communication, Proxy for proxy services, and Hosted for cloud hosting. An example of this information is given in Listing 2:

```
IPs found:
account.protonvpn.store (United States, ISP: Trellian Pty. Limited, Mobile: False,
Proxy: True, Hosting: True)
```

Listing 2: Informations on the URL found in MOTIF_0ae37532a7bbce

4.3 Integration as a Subsystem of *Goatracer*

This script was then ported as a *Goatracer* subsystem by engineers from the team. As such, Goatracer now outputs a new `_network.json` file containing the informations extracted from the *syscalls*.

We decided to keep the IP-API calls as part of the python reader scripts, so the original script was modified to take as input the new json file and do the API calls for the listed addresses. This allows us to keep the calls client-side and avoid the risk of getting rate limited.

5 Results

5.1 Comparison to Other Similar Tools

Goatracer is not the only dynamic analysis platform available to the public. In order to evaluate our ability to find addresses in the analysed binaries, we can to compare it to other tools.
Such tools include Any-run [11] and VirusTotal [12]. They are used as reliable tools in the cybersecurity world, however they are both commercial and closed source services, so we cannot easily find informations as to how they function.

Both of these tools give informations about network communications made by the executable file. An educated guess could point to them using network packets capture instead of our method, as we can see DNS queries in the output, which is necessary to communicate with a domain name but not a deliberate action made by the program. As such, our script retrieves less information than the other tools, but has less noise in the data. We did find however some cases where our script did not find some malicious addresses that the two other tools did. This issue is for now not yet resolved and will be investigated further by the team.

5.2 Current Limitations

Our biggest limitation right now is that the virtual machines *Goatracer* uses do not have any network capabilites. This prevents us from observing malware communications in the case of a successful connection. If *Goatracer* is modified to integrate a technology like fakenets, we could use network packets capture tool in conjunction with our current method to improve our detection. But as *Goatracer* is a tool that tries to be as stealthy as possible to malware, this kind of improvement would have to be carefully implemented, with ethic concerns in mind (as running a malware on a VM with network may cause harm to other external devices).

Another limitation of our current tool is that it relies on the addresses being passed as plain text during certain syscalls. While it is frequent for malware to use those, we think that a malware could theorically obfuscate addresses by hard-coding the address in the struct `sockaddr` passed as argument for certain socket functions, and therefore evade our detection technique. We have yet to observe this behaviour, and detecting those would also require solving some technical challenges to extract the structs in memory passed as arguments during the analysis.[1]

As discussed in Section 5.1, we still are prone to some problems with missed addresses, which will have to be solved in order for the tool to be more reliable.

[1] After writing this report, the team found a malware with this behaviour and implemented the extraction of addresses from the struct, confirming the initial hypothesis.

6 Conclusion

The original goal for this internship was to extract network-related address informations from the system calls arguments observed by *Goatracer*. After examining a first method involving regular expressions and setting it aside due to obfuscation concerns, we decided to exhaustively list all Windows standard libraries that involved network communication, and select the functions in those libraries that had addresses as part of their arguments.

Once this list has been established, we made a python script to print out those arguments, and we used an API to gather more information on those addresses, including the country and ISP. After that, we integrated this script as part of Goatracer.

While our tool had some limitations due to multiple difficulties, we have a satisfying result after testing on the MOTIF malware set and comparing to other similar tools.

Bibliography

[1] "Laboratoire de Haute Sécurité." [Online]. Available: https://lhs.loria.fr/

[2] R. Guittienne, Q. Jacqmin, J.-Y. Marion, P. Marty, and F. Sabatier, "GoaTracer: An Open Service for Advanced PE Tracing," in *Botconf 2025 - 12th Edition*, Angers, France, May 2025. [Online]. Available: https://hal.science/hal-05086241

[3] G. Bonfante, J. Fernandez, J.-Y. Marion, B. Rouxel, F. Sabatier, and A. Thierry, "CoDisasm: Medium Scale Concatic Disassembly of Self-Modifying Binaries with Overlapping Instructions," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, in CCS '15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 745–756. doi: 10.1145/2810103.2813627.

[4] "MITRE ATT&CK®." [Online]. Available: https://attack.mitre.org/

[5] R. J. Joyce, D. Amlani, C. Nicholas, and E. Raff, "MOTIF: A Large Malware Reference Dataset with Ground Truth Family Labels." [Online]. Available: https://arxiv.org/abs/2111.15031

[6] W. Beukema, "Windows Command-Line Obfuscation." [Online]. Available: https://www.wietzebeukema.nl/blog/windows-command-line-obfuscation

[7] D. Bohannon, "DOSfuscation: Exploring the Depths of Cmd.exe Obfuscation and Detection Techniques." [Online]. Available: https://services.google.com/fh/files/misc/exploring-the-depths-of-cmd-exe-obfuscation-wp-en.pdf

[8] "Reader-CFG-CG-JSON Repository." [Online]. Available: https://gitlab.com/defmal1/dynamicanalysishybridplatform/reader-cfg-cg-json

[9] "IP-API Website." [Online]. Available: https://ip-api.com/

[10] "AbuseIpDB Website." [Online]. Available: https://www.abuseipdb.com/

[11] "Any-run Website." [Online]. Available: https://any.run/

[12] "VirusTotal Website." [Online]. Available: https://www.virustotal.com/