OPEN SOURCE MALWARE LAB

Robert Simmons
ThreatConnect, USA

Email rsimmons@threatconnect.com

ABSTRACT

The landscape of open source malware analysis tools improves every day. A malware analysis lab can be thought of as a set of entry points into a tool chain. The main entry points are a file, a URL, a network traffic capture, and a memory image. This paper is an examination of the major open source tools that satisfy the analysis requirements for each of these entry points. Each tool's output can potentially feed into another tool for further analysis. The linking of one tool to the next in a tool chain allows one to build a comprehensive automated malware analysis lab using open source software.

For file analysis, the three major versions of *Cuckoo Sandbox* will be examined. To analyse a potentially malicious URL, the low-interaction honeyclient *Thug* will be covered. Next, if one has a network capture (PCAP) to analyse, the *Bro Network Security Monitor* is a great option, and will be covered. Finally, for cases in which the analysis target is a memory image, the *Volatility Framework* will be examined. Each of the inputs and outputs of the tools will be reviewed to expose ways in which they can be chained together for the purpose of automation.

INTRODUCTION

There is a large and ever growing selection of tools that a researcher can use to analyse malware today. By identifying the major entry points into the malware analysis process, one can evaluate and choose the best tool to handle the analysis of each entry point. The four major entry points examined here are: a file, a URL, a recording of network traffic, and an image of volatile memory. Each of these entry points is analysed using a freely available, open source tool. To build a complete open source malware lab that can begin analysis with any of the four major entry points, output from each tool must be fed into the next tool in the analysis tool chain.

Before digging deeper into each tool and how they can be tied together, we must spend time explaining why an open source malware lab is needed. First and foremost, it is one of the basic weapons in the armoury of a threat researcher. Of the four stages of malware analysis – fully automated analysis, analysis of static properties, interactive behavioural analysis, and manual code reversing – this lab covers the first two stages, which are considered the easier stages [1]. Additionally, if one is hunting for threats within an enterprise, having a malware lab available through which collected samples can be run immediately is crucial. Finally, a malware lab is an important component of network defence. Network security monitoring tools that carve files from network traffic can send these files to the lab for analysis. Similarly, inbound emails with attachments and URLs can have these two analysed by the lab for signs of malicious activity. Finally, many host-based intrusion detection systems will send unknown executables to their control server for further analysis. These files can in turn be analysed by an open source malware lab.

CUCKOO SANDBOX

The first tool is really more of a large collection of tools all orchestrated together into a single file analysis system. In this case, a file not only includes all types of executables, but also *Microsoft Word*, *Excel*, and *PowerPoint* files, as well as PDFs, Java JAR files, and Flash files, among others. Essentially, any file that can be executed or that can lead to code execution post exploitation is included.

Cuckoo currently has three major flavours: stable (1.2) [2], release candidate (2.0 RC1) [3], and cuckoo modified (a.k.a. *Spender Sandbox*) [4]. Cuckoo modified and the current release candidate have a few important differences from the stable version. First is the normalization of file and registry paths. In cases where a dropped file's path would be different depending on *Windows* OS versions, *Cuckoo* will replace the version-specific portion of the path with a standard environment variable. This allows one to use data collected from multiple versions of *Windows* and still be able to perform analysis and comparisons across these differences. An example of this would be changing C:\Users\WSuser\AppData\bonzo\AIDVFP.jpg to %APPDATA%\bonzo\AIDVFP.jpg. A second important difference is the variety of OS support in the current version of *Cuckoo*. It supports *Mac OS X*, *Linux* and *Android*. Finally, the ability to send outbound malware network traffic to a remote VPN or with Tor anonymization allows the researcher to conceal the IP address of the *Cuckoo* instance, and to adjust the apparent geographic location of the victim workstation to fool location-aware malware.

Two pieces of software that work in concert with *Cuckoo Sandbox* are *Paranoid Fish* (pafish) [5] and *VMCloak* [6]. *Paranoid Fish* detects certain anti-analysis techniques that malware will use to determine whether it is being observed in a VM or in a sandbox. If detected, the malware may behave in a different way from normal, or may not display any behaviour at all. To correct for these detectable features of a sandbox, *VMCloak* is used to obfuscate these features so that they are less detectable by malware. *VMCloak* also helps greatly by building the VMs for use in *Cuckoo* dynamically and installing software versions automatically.

From the *Cuckoo* report, specifically the JSON report, network indicators are collected, such as IP addresses, hostnames, URLs, and hashes of dropped files. Further analysis down the tool chain examined here can be done on the dropped files as well as the captured network traffic, and the URLs visited during the sandbox session. One problem to look out for when automating this process is to watch how many generations of parent-child dropped files are to be analysed automatically. One does not want to end up with an infinite loop of dropped files that are all essentially the same except for dynamically generated padding added before the drop. This can be corrected for by stopping the automated analysis after about six generations. It is always good to add an alert if a sample reaches this point so that manual checks of the malware and the condition of the dropped files can be performed.

THUG

Thug [7] is a low-interaction honeyclient. This is a software system that pretends to be a browser. Its goal is to coax a drive-by URL to deliver its payload. A drive-by is a URL that may try to determine the version of software the visitor is running and then deliver an exploit of some kind. The end result is that malicious code is run on the victim's system without any user action beyond clicking the initial drive-by link. A honeyclient such as *Thug* will try to trigger the drive-by and then capture the payload. It does this by changing its user-agent string to any number of common combinations of operating system and browser, including *iOS* and *Android*. It also is able to present a specific version of a plug-in such as Java or Flash to make sure that the vulnerable version the drive-by is seeking is presented to the malicious URL properly, and the payload is subsequently captured. A second type of URL that is a bit easier to work with is a malware download URL. This type of URL typically hosts a second-stage malware implant, or an updated version of the malware. For these, *Thug* will simply visit the URL and download the payload file for analysis.

Thug's most important outputs are its JSON formatted logs which can be culled for network indicators such as URLs, hostnames, and IP addresses. In addition to these, if the drive-by is triggered and the payload captured, these files can then be automatically analysed by *Cuckoo* as another stage of analysis in this tool chain. Finally, *Bro* can be used to analyse the network traffic produced during the visit to the URL.

BRO

Bro [8] is a network security monitoring (NSM) framework that is used to analyse live packet capture or recorded packet capture in the form of PCAP files. NSM is *Bro*'s primary use, but is not how it is utilized in this open source malware lab. In this case, *Bro* is used to example the captive traffic of a malware sample being run alone in a controlled environment.

Bro consists of a large set of scripts written in the *Bro* scripting language. Each script or set of scripts in turn produces a *Bro* log. This is a text file with data gleaned from aspects of the packet capture. A change to the local.bro configuration file changes the output format of the *Bro* logs to JSON format for ease of automated consumption.

The main *Bro* logs that are important for malware analysis are the conn.log, dns.log, http.log, files.log, and the extract_files folder where files carved from the network traffic are collected for analysis. Network indicators can be collected from conn.log, dns.log, and http.log, whereas file hashes used for file indicators can be gathered from the files.log. URLs can be further analysed using *Thug*, and finally, executables and documents carved from the network traffic can be further analysed using *Cuckoo Sandbox*. Again, take care when analysing the output to make sure that an infinite loop is not created. To correct for this, make sure not to re-analyse a URL or file that has already been analysed, and restrict parent-child analyses to about six generations. Make sure to include alerts when this type of restriction is used so that manual investigation can be performed on the samples.

VOLATILITY

Volatility [9] is a memory analysis framework that is used to extract artifacts from volatile memory. It currently supports *Windows*, *MacOS X* and *Linux* operating systems. One of the main tasks that can be performed with *Volatility* is to find hidden or unusual processes. It is always good to start with a clean memory image that one can compare with one that has been infected with a malware sample. Process lists can be run through diff to find processes that did not exist in the clean image. Two main *Volatility* tools for this purpose are psscan and pslist. Additionally, dumps of these processes can be extracted and then further analysed using *Cuckoo* after import tables have been repaired sufficiently.

In addition to the variety of executables that can be extracted and dumped for further analysis, many network connections made by processes can be analysed and the destination IP address and port collected as a network indicator. Finally, visited URLs can be pulled out of the *IE* browser history, but these are typically found more often in a memory image of a victim rather than a specifically built machine dedicated to malware analysis.

CONCLUSION

The most important concept when lining these tools up into a tool chain is to have the output from one tool feeding the input of the next tool that can consume and analyse that particular output type. Carved and dropped files are analysed with *Cuckoo Sandbox*, URLs with *Thug*, collected PCAP files with *Bro*, and collected memory images with *Volatility*. For this project, *ZeroMQ*, and open source message queue software was used, but any type of message queuing system is appropriate in this case. It was found that using a web server such as *NGINX* to move the files themselves from one tool in the tool chain to another is much more efficient and easier to work with than adapting the message queue to handling these large files itself. In this configuration, the message queue transmits the server and file location at which the next tool in the tool chain can pick up the file that it requires for analysis. For the report output from the tools mentioned, each provides for some type of JSON-based output that is quite easily consumed using Python scripts as the glue that holds together the open source malware lab.

REFERENCES

[1] https://zeltser.com/mastering-4-stages-of-malware-analysis/.

[2] https://github.com/cuckoosandbox/cuckoo/releases/tag/1.2.

[3] https://github.com/cuckoosandbox/cuckoo/releases/tag/2.0-rc1.

[4] https://github.com/spender-sandbox/cuckoo-modified.

[5] https://github.com/a0rtega/pafish.

[6] https://github.com/jbremer/vmcloak.

[7] https://github.com/buffer/thug.

[8] https://www.bro.org/.

[9] https://github.com/volatilityfoundation/volatility.