Breaking the bank(er): Automated configuration data extraction for banking malware

Author: James Wyke

Affiliation: Sophos

Contact email: james.wyke@sophos.com

Abstract

Despite recent successes against large banking malware botnets such as Gameover Zeus and Shylock, banking malware continues to be a huge threat as new families such as Dyreza, Vawtrak and Dridex have occupied the vacant space in the market.

Not only has the malware itself matured but we have also seen the ecosystem and market model used by the malware authors evolve, as they gain in professionalism and sophistication.

In order to provide holistic protection against these threats and to aid an adequate incident response and forensic post-mortem should a compromise succeed, we must know as much about the malware as possible.

There is a wide variety of information that is useful to us: indicators of compromise, command and control addresses, campaign IDs, botnet names, revision numbers, cryptographic keys, downloaded configuration files that may contain web injects, redirections, further modules, tertiary command and control addresses, and many more.

We can use some of these items of information to aid protection, others to identify infected hosts, decrypt network traffic and identify stolen data, and track threat campaigns to help us assess the overall impact of the threat and provide attribution.

Extracting this information can be a painstaking manual task that takes a great deal of time. A far better solution is to automate the process.

In this paper we outline our sandbox based system that automatically extracts command and control addresses, decrypts and processes network traffic and configuration files, and extracts and stores many other types of valuable data, in a scalable and extensible way.

We describe the architecture of the system, the ease with which new modules can be plugged in to handle new malware families, and how we use this system to track and protect our customers against highly prevalent and damaging malware families including Vawtrak, Dyreza, Dridex and Zeus.

Introduction

In 2014 two highly successful Law Enforcement actions took out the Gameover Zeus botnet [1] and the Shylock botnet [2]. Contrary to many previous actions against financial malware operations, these takedowns effectively eradicated Gameover Zeus and Shylock, representing perhaps the most successful Law Enforcement and Industry actions of their kind.

Despite these successes banking malware is as prevalent and as much of a threat as ever. New and evolved families such as Dridex [3], Dyreza [4] and Vawtrak [5] have taken the place of the families that were wiped out. This waxing and waning of malware families is a pattern we have seen in the past and one we can expect to see continue in the future.

Since banking malware is going to be a threat for the foreseeable future we must find ways to combat it. In this paper we present a system that automatically extracts a wide variety of useful data from banking malware families that can be used to protect against further variants and to aid Incident Response after successful compromises.

Breaking Banking Malware

One way to combat a threat is to attempt to gain as much information about it as possible. This is a strategy we can employ against banking malware. There is a wide variety of data that we can extract from banking malware samples that we can use to block elements of the malware's functionality and to establish what actions the malware carried out when assessing the impact of a compromise.

The data we can extract ranges from the more obvious examples such as command and control addresses, particularly useful when there are backup addresses that may not be evident

during normal execution, to more obscure items such as decryption keys, Domain Generation Algorithm (DGA) seed values, campaign names, build versions, installation artifacts such as file names and registry key names, network traffic such as downloaded configuration files, downloaded modules and commands received from the command and control server.

Some of this data is more directly useful for protecting against further variants of the family, such as network artifacts which can be blocked or used to identify other infected systems, and downloaded updates and further modules that detection can be added for.

Other types of data are more useful in discovering the intentions of the malware authors, which can be useful when attempting to develop more robust protection mechanisms. This includes downloaded configuration files which may contain web page code injections and other valuable information, as well as specific commands that the botnet controllers may issue to infected machines, such as further downloads or system information gathering commands. We may find that these types of data require other key pieces of information to be extracted first, such as decryption keys. Figure 1 shows a snippet of the web page code injections from a Vawtrak sample.

Web Inject	Target URL: ^de8of677fyt0b\.cloudfront\.net/adrum-ext\.[a-z0-9]{32}\.js\?[0-9]{8}$ Flags: 0x12 Data before: m.info("Sending CORS Beacon:"+b+"\n"); Data after: s("geoCountry",n.truncate(Data inject: m.info("Sending CORS Beacon:"+b+"\n");if (typeof(est_script)=="undefined"){
Web Inject	Target URL: ^de8of677fyt0b\.cloudfront\.net/adrum-ext\.[a-z0-9]{32}\.js\?[0-9]{8}$ Flags: 0x12 Data before: s("pageType", Data after: s("geoCountry",n.truncate(Data inject: 0);s("baseGUID", null);s("parentGUID", null);s("parentPageUrl", null);s("parer
Web Inject	Target URL: ^de8of677fyt0b\.cloudfront\.net/adrum-ext\.[a-z0-9]{32}\.js\?[0-9]{8}$ Flags: 0x12 Data before: m.info("Sending Beacon:\n"+decodeURIComponent(c.replace(/&/g,"& " Data after: s("geoCountry",n.truncate(Data inject: m.info("Sending Beacon:\n"+decodeURIComponent(c.replace(/&/g,"& "))
Web Inject	Target URL: ^de8of677fyt0b\.cloudfront\.net/adrum-ext\.[a-z0-9]{32}\.js\?[0-9]{8}$ Flags: 0x12 Data before: m.EXT.la(this.Da(h)) Data after: s("geoCountry",n.truncate(Data inject: m.EXT.la(this.Da(h)).replace('?10062015','')
Web Inject	Target URL: ^[a-z]{3,7}\.[a-z\-]{3,14}\.co\.uk/personal/.*/lib/adrum.js\?[0-9]{8}$ Flags: 0x12 Data before: adrumExtUrl:DI.adrum.adrumExtUrl Data after: s("geoCountry",n.truncate(Data inject: adrumExtUrl:DI.adrum.adrumExtUrl+'?10062015'

Figure 1 – Web page code injection example used by Vawtrak

Yet further types of data may be useful to track specific campaigns or the actors behind the campaigns. This is generally the kind of meta-data that will help us differentiate this variant from a similar variant that may have been used by a different group. For example, many families embed a botnet name or a campaign ID into binaries. We can extract these values and use them to track specific variants over time. Figure 2 shows the XML configuration file extracted from a Dridex loader sample that contains the botnet name, in this case botnet “125”, and server addresses.

```
<config botnet="125">
   <server_list>
91.227.18.85:80
66.110.179.66:8080
202.44.54.5:8080
176.31.28.253:80
   </server_list>
</config>
```

Figure 2 – Dridex botnet name and server details

Figure 3 provides a summary of the different types of data we can extract and how they can be useful.

Data Type	Uses
Command and control addresses	Prevent further infection, Identify additional compromises
Downloaded configuration files	Secondary and tertiary command and control addresses, Web page code injections (web injects), Further module download addresses, Redirects, Keylogger processes, More…
Decryption keys	Decrypt other data types
Campaign IDs, build versions, botnet names	Actor tracking, Variant distinction
DGA seed values	Replicate DGA, block domains
Bot updates	Replicate/detect new versions
Issued commands	Further network addresses, Gain understanding of attacker intentions
Installation artifacts – filenames, registry key names etc	Indicators of compromise, Targets to extract from disk
Further cryptographic keys, e.g. signature verification keys	Actor tracking

Figure 3 – A comparison of the different types of extractable data

Solutions

The traditional way to extract this kind of information is through manual analysis. A skilled analyst can reverse engineer a malware sample and work out where this data is stored and how it is encrypted and use that information to decode data from memory or network captures. Perhaps the analyst will write a quick tool to perform the decoding and some of the extraction work on the sample.

However, this is not a solution that scales. Manual analysis is a difficult process that can take a highly skilled researcher several days or longer. When dealing with many thousands of samples every day it is impossible for each one to be handled by a human analyst. To successfully extract data for the volume of samples that we would like, the process must be entirely automated.

Furthermore, we need to distil the knowledge gained from the analysis carried out by the human into a reusable form that can be applied without modification by a machine to all other samples belonging to the same malware family. We should embed this knowledge into a framework that can be easily extended to perform a similar level of data extraction against new malware families, and we should ensure that the framework is capable of processing many thousands of samples per day. The data we extract must be machine-readable so that it can be passed on to other systems such as URL reputation and sample processing systems.

An ideal location for this framework is an existing high availability, high throughput automated analysis or sandbox system. Many organisations are setting up systems such as this for internal and external or commercial use. Embedding the custom data extraction framework into a sandbox solution means that all the problems of throughput, availability and queue management become problems belonging to the sandbox and we are free to concentrate on the data extraction framework. For our solution we have used the freely available Cuckoo sandbox [6].

Architecture

Figure4 represents a high level overview of the architecture of the system.

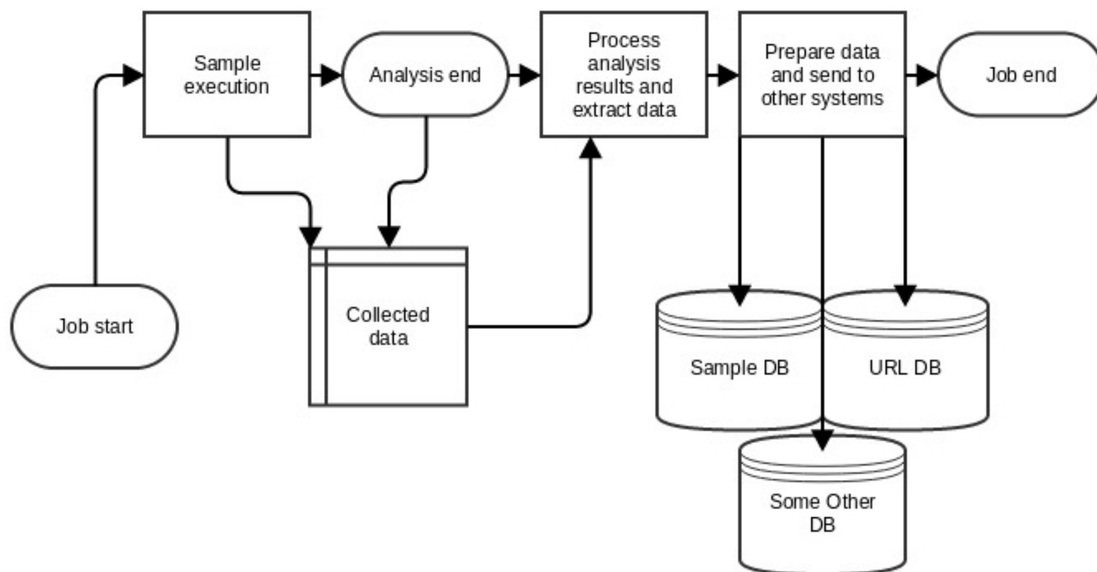

Figure 4 – high level architecture

The key considerations are ensuring that the correct data is collected during the *sample execution* and at the *analysis end* stages and that the logic exists to correctly extract the required information from the raw data during the processing stage. Once the data has been properly extracted it can be packaged and sent on to other systems.

Capturing Data

Important items of information that we need to capture typically include memory dumps, network traffic, and registry and disk changes.

For memory data we have found it advantageous to capture full memory at the end of analysis and also to incorporate inline memory dumping during execution. This is particularly useful when applied to processes when they exit, by initiating a dump from *ExitProcess* or a similar API hook, as these processes and any data they contain will most likely not be available in a full memory dump at the end of the analysis.

Cuckoo includes a facility called *Auxiliary modules* that are used to execute tasks in parallel to analysis jobs [7]. The network capture takes place in an auxiliary module named *sniffer.py*, which uses *tcpdump* to output a *pcap* file. We have also found it useful to intercept and capture encrypted network traffic sent over TLS which can be implemented through a man-in-the-middle proxy and an additional auxiliary module.

Some malware families will not display their full functionality without certain conditions being met during execution. For example, Vawtrak will not contact its command and control server until a browser process makes an outbound network connection. Some malware families such as Tinba [8] use a DGA to calculate command and control server addresses. It make take some time before a live command and control server address is generated so we may want to execute samples belonging to these families for longer than others.

We may also see situations such as with many Zeus variants where the sample file we have does not execute outside of the specific location that it was dropped to on a victim machine. Zeus encodes the pathname and a cryptographic value that is dependent on the target machine into its own executable which prevents normal execution in a sandbox system. If we wish to process these samples we will need to either spoof values or emulate the network activity that the dropper sample would have carried out.

Fortunately Cuckoo provides a feature called *Analysis packages* which we can use to ensure the appropriate circumstances for successful execution are created for each family [9], should we establish that abnormal conditions are required. These analysis packages may be very simple such as the Vawtrak case where we need only launch a web browser after a certain period of execution, but they can also be more complicated, as with Zeus variants, where we dump the sample during execution, extract the information necessary to send a request to the command and control server, and send the data ourselves.

Data Processing

Once we have captured the raw data we must find and extract the information that is relevant to us. Cuckoo provides *Processing modules* that deal with various types of raw data and present it in a structured way to the next layer. We can add a processing module specifically for extracting the configuration data from the raw data captured during execution.

We implemented a parent *ConfigurationData* processing module that takes the data collected during analysis – memory dumps, network captures and any disk and registry artifacts that have been captured – applies a filtering stage against the data to identify which family the sample may belong to and then loads another module that is designed specifically for that malware family.

These family modules are the only part of the framework that needs to be implemented for each family. They can vary according to the specific requirements of the malware family being addressed but they tend to be fairly similar. The dumped files are searched for the markers that indicate where important data structures are located. These data structures are then decoded and added to the results dictionary returned by the outer *ConfigurationData* module.

The data that has been extracted from the memory dumps will likely then be used to decode other artifacts such as network traffic, registry hives or files from the disk. The accumulated data is then added to the results for the analysis task.

This framework is fairly flexible and allows for the use of third party modules for the actual data processing if required. For example, we can use *Volatility* [10] modules against the full memory dump or *ChopShop* modules [11] against the network capture.

Worked Example

To illustrate the concept we will now give a detailed walkthrough of the data extraction module used to extract data from Vawtrak samples.

All the code for dealing with Vawtrak data is contained within a single Python module named *vawtrakconfig.py*, which contains a single class *VawtrakConfig*. All such classes should inherit from the *ConfigData* class which contains initialisation and helper methods. Each new class must at least implement the *getData()* method which returns the extracted data as a Python dictionary. Figure 5 shows a simplified overview of the *VawtrakConfig* class.

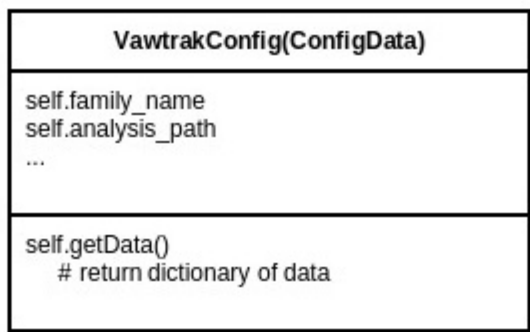

Figure 5 – VawtrakConfig class overview

Inside *getData()* we construct a dictionary object and fill it with the various items of interest that we want to return. These include the command and control addresses, build version, project ID and POST request format string. We also decode any useful data from the network capture, such as downloaded configuration files and commands received from the server.

Vawtrak stores most of its static data (almost everything except data downloaded from the web), inside an encrypted blob embedded in the executable. Inside *getData()* we have further methods to find this blob inside the dumped PE file, decode the data and add it to the results.

Figure 6 shows the piece of code that references the encrypted blob, labelled as *Ptr2EncryptedBlob* in the image.

```
55                  push    ebp             ; GetTickCount
8B EC               mov     ebp, esp
51                  push    ecx
8B 0D 44 21 73 02   mov     ecx, Ptr2EncryptedBlob
8B 01               mov     eax, [ecx]
89 45 FC            mov     [ebp+var_4], eax
85 C0               test    eax, eax
74 42               jz      short loc_270D725
6A 04               push    4               ; keySize
8D 45 FC            lea     eax, [ebp+var_4]
50                  push    eax             ; Key
8D 41 04            lea     eax, [ecx+4]
68 7F 09 00 00      push    97Fh            ; SizeOfData
50                  push    eax             ; Data
E8 19 40 00 00      call    DoVawtrakRC4
A1 44 21 73 02      mov     eax, Ptr2EncryptedBlob
6A 00               push    0
83 20 00            and     dword ptr [eax], 0
E8 12 38 00 00      call    GenRandNumOrKeyVal
8B 0D 44 21 73 02   mov     ecx, Ptr2EncryptedBlob
33 D2               xor     edx, edx
83 C4 14            add     esp, 14h
0F B6 89 10 02 00 00 movzx  ecx, byte ptr [ecx+210h]
```

Figure 6 – Code referencing the Vawtrak encrypted blob

We find this section of code using a regular expression and extract the address of the encrypted blob from the code block. Once the blob has been located we can read it from the dumped file and decrypt it in Python, grabbing the data from the various offsets.

Vawtrak's downloaded configuration file is compressed with *aPLib* [12] and then encrypted with the same algorithm used to encrypt the blob that contains the static details. To make processing of the network traffic easier we first convert the pcap file to a HTTP Archive (HAR) file, then examine each flow for likely command and control traffic, decode it and add the results to the dictionary.

Extending

The process for adding data extraction for new malware families is relatively simple as the framework we have created does not generally need to be modified. A new family-specific module can be added that is just concerned with the details of data extraction for this malware family. Furthermore, since the framework is fairly generic we can apply this process to any types of malware where there is data that we wish to extract that is not immediately evident from normal execution, not just banking malware.

Output and Results

Once the data has been collated we can send it to the various other systems that will consume and act on that data. For this we use Cuckoo's *Reporting Modules*. These are a class of module that take the combined results dictionary and present it in various formats, for example Cuckoo includes a JSON reporting module which dumps the whole results dictionary out to a JSON file, and a *MongoDB* reporting module which outputs all the data into a MongoDB database that also acts as a user-friendly web interface to the results.

We can include a reporting module that extracts all network artifacts from the results and delivers them to a URL reputation system. We can take all extracted PE files and send them to a file processing system. We may choose to send the web injects and other contents from any downloaded configuration files to another database where those specific details are stored for analysis.

It is worth demonstrating the output that can be achieved by showcasing several examples. Figure 7 shows the cryptographic key material and the configuration file URLs for a Citadel sample.

Sample Info

CONFIG VALUE	DATA
family_name	citadel
config_url	[u'http://www.soldelplata.com.ar/wp-admin/file.php|file=config.dll',
citadel_login_key	C1F20D2340B519056A7D89B7DF4B0FFF
variant	01050301
rc4_key	8b6fd115ad2488211e283d4a7821caa53801a99063cdea702d374
citadel_post_key	8f0b49d6

Figure 7 – Citadel data

Figure 8 shows the campaign ID and command and control server addresses for a Dyreza sample.

Sample Info

CONFIG VALUE	DATA
family_name	dyreza
dyreza_campaign_id	2506uk12
config_url	[u'85.192.165.229:443', u'212.37.81.96:4443', u'194.28.190.84:443', u' u'67.207.228.144:443', u'83.168.164.18:443', u'195.34.206.204:443', u u'208.123.135.106:4443', u'176.197.100.182:443', u'31.42.172.36:443' u'176.103.203.166:443', u'80.234.34.137:443', u'178.219.10.23:443', u u'209.193.83.218:4443', u'162.255.126.8:4443', u'193.33.206.47:4443'

Figure 8 – Dyreza campaign ID and command and control servers

Figure 9 shows a variety of information extracted from a Vawtrak sample, including the build version, command and control addresses, the format string for its POST request and the *project ID* used to differentiate between campaigns.

Sample Info

CONFIG VALUE	DATA
family_name	vawtrak
vawtrak_build	0x41
config_url	[u'transfercom.net', u'tankardhier.net', u'jughaus.net', u'incubatenet.com',
vawtrak_format_string	/collection/{PROJECT_ID:HD}/{TYPE:HB}/{BOT_ID:HD}
pubkey_xsum	4fbfd1644940284e5b00d57e1c607ac465a4d539
vawtrak_updatever	0xb
vawtrak_projectid	0x53

Figure 9 – Vawtrak extracted data

When this level of data extraction is employed on a large scale we begin to see certain trends in the data. For example we have used this data to track Vawtrak campaigns in [5] and determine the geographically localised interests of the Vawtrak customers based on the web injects deployed. We can track re-use of encryption keys across Zeus variants and identify samples that are likely to be operated by the same groups. We can also identify similarities between the web injects used in different banking malware families that might indicate that the web injects have been developed by the same individual or team. We can also identify when a new organisation is being targeted by web injects, redirects or some other banking malware technique.

Pitfalls

There are certain pitfalls that are worth mentioning for anyone who is looking to develop a similar project to this.

One snag we have occasionally found is the issue of certain malware families needing slightly different circumstances to execute to their full extent than others. We have already discussed the use of *Analysis packages* to create these circumstances but it can also be advantageous to allow the sample to execute for longer than normal so that maximum opportunity is given to download modules and configuration files.

There is also an increasing trend of malware becoming more modularised and multi-stage. It is useful to execute these samples on multiple platforms as second stage modules will be quite often different depending on the architecture of the victim machine. For example, Dridex will download an x86 version of its main module if running under x86 and an x64 version if executing under x64, so it is important to execute the dropper on both platforms to ensure coverage for all the components.

Conclusion

We have built a system that automates the extraction of a wide range of valuable information from a range of banking malware families. It is extensible and scalable and used for protection as well as longer term analysis purposes.

We have designed a framework that allows analysts to concentrate on the hard part of the process – analysing the malware – and not get bogged down dealing with extraneous details that are typically copy-pasted from one project to another.

We have shown that this system can be put to a variety of uses ranging from blocking and detection of malware artifacts to threat actor tracking and giving assistance to digital forensics and incident response. The system can be continuously improved by adding to the number and range of families that are addressed and this is an ongoing task.

References

[1] "U.S. Leads Multi-National Action Against 'Gameover Zeus' Botnet and 'Cryptolocker' Ransomware, Charges Botnet Administrator", United States Department of Justice, June 2014, http://www.justice.gov/opa/pr/us-leads-multi-national-action-against-gameover-zeus-botnet-and-cryptolocker-ransomware

[2] "Global Action Targeting Shylock Malware", Europol, July 2014, https://www.europol.europa.eu/content/global-action-targeting-shylock-malware

[3] "Dridex Banking Trojan Begins 2015 with a Bang", Ryan Olson, January 2015, http://researchcenter.paloaltonetworks.com/2015/01/dridex-banking-trojan-begins-2015-bang/

[4] "Dyre Banking Trojan", Brett Stone-Gross and Pallav Khandhar, December 2014, http://www.secureworks.com/cyber-threat-intelligence/threats/dyre-banking-trojan/

[5] "Vawtrak – International Crimeware-as-a-Service", James Wyke, December 2014, https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/sophos-vawtrak-international-crimeware-as-a-service-tpna.pdf?la=en

[6] http://www.cuckoosandbox.org/

[7] http://cuckoo.readthedocs.org/en/latest/customization/auxiliary/

[8] http://securityblog.switch.ch/2015/06/18/so-long-and-thanks-for-all-the-domains/

[9] http://cuckoo.readthedocs.org/en/latest/customization/packages/

[10] http://www.volatilityfoundation.org/

[11] https://github.com/MITRECND/chopshop

[12] http://ibsensoftware.com/products_aPLib.html