DGA Clustering and Analysis: Mastering Modern, Evolving Threats

DGALab

Alexander Chailytko
Malware Reverse Engineering Team
Check Point Software Technologies
Minsk, Belarus
alexanderc@checkpoint.com

Aliaksandr Trafimchuk
Malware Reverse Engineering Team
Check Point Software Technologies
Minsk, Belarus
aliaksandrt@checkpoint.com

This paper was presented at Botconf 2015, Paris, 2-4 December 2015, www.botconf.eu
It is published in the Journal on Cybercrime \& Digital Investigations by CECyF, https://journal.cecyf.fr/ojs
It is shared under the CC BY license http://creativecommons.org/licenses/by/4.0/.
DOI: 10.18464/cybin.v1i1.10

***Abstract* — Domain Generation Algorithms (DGA) is a basic building block used in almost all modern malware. Malware researchers have attempted to tackle the DGA problem with various tools and techniques, with varying degrees of success. We present a complex solution to populate DGA feed using reversed DGAs, third-party feeds, and a smart DGA extraction and clustering based on emulation of a large number of samples. Smart DGA extraction requires no reverse engineering and works regardless of the DGA type or initialization vector, while enabling a cluster-based analysis. Our method also automatically allows analysis of the whole malware family, specific campaign, etc. We present our system and demonstrate its abilities on more than 20 malware families. This includes showing connections between different campaigns, as well as comparing results. Most importantly, we discuss how to utilize the outcome of the analysis to create smarter protections against similar malware.**

Keywords—malware; DGA; clustering; automation; reverse engineering, threat intelligence

I. INTRODUCTION

Malware that uses Domain Generation Algorithm (DGA) has become more prevalent these days. DGA [1] is an algorithm whose main function is to periodically generate a large number of domain names that can be used by the malware to communicate with a malicious C&C server.

This introduces a lot of new challenges to malware researchers and the antivirus industry as a whole. It is becoming increasingly difficult to seize the operation of malware families using DGAs; sinkholing in its current form is not effective because there are too many domains that need to be blocked every day, and only a small number of them will be contacted by the malware. The sinkholing process cannot be automated. It also often involves contacting domain name registrars, so they can proceed with taking legal actions against malicious domains, which is only feasible for large security companies.

We need more data to effectively counter innovations like DGA. That's why we initiated our DGALab project, in which we produce an exhaustive DGA feed that includes the most popular DGA families seen in the wild.

II. MODULAR SYSTEM & CLUSTERING PROBLEM

DGALab is a highly modular system. We now discuss each module in more detail.

A. *Cuckoo core & VMWare Workstation*

Cuckoo [2] is a malware analysis system that can be customized to achieve almost any goal in automating malware research. We use *Cuckoo* as a foundation to run and control our virtual machines.

We made several modifications in *Cuckoo*, mostly to increase our successful emulation rate and to prevent detection of *Cuckoo* and the virtualized environment. *CuckooMon*, which is the DLL injected into the target processes, was cleaned up and rewritten.

Also, we have decided to use *VMWare Workstation* instead of the more frequently used *VirtualBox* for the sake of reliability.

B. *DGALab modules (data sources)*

There are many DGA implementations in the wild. Most of them, however, can be grouped into 3 major families:

1) Static DGA – a DGA whose generated output is not dependent on date or seed. Most likely this kind of DGA will generate the same domains every time (example: *Expiro*).

2) Date-based DGA – this type of DGA generates domains based on date (example: *Conficker*).

3) Seed-based DGA – this type of DGA utilizes hardcoded seed and\or date to generate domain names. The seed is usually a very large number which cannot be predicted. The seed may also be changed for different targets or campaigns.

Reimplementing the DGA algorithm is not sufficient to deal with that type of DGA; one needs to reverse engineer every new sample for each campaign, to extract the relevant seed or to emulate every such sample (example: *Tinba*).

We designed DGALab to be able to handle successfully each of these families and developed a separate module for each specific family.

A DGALab module is a Python script. There are 4 types of modules that DGALab supports:

1. Static DGA module

This is the simplest type of module. As stated previously, a static DGA generates constant domain names which do not depend on date or seed. We reverse engineer this malware only once and write the module. It will then generate domains automatically for our output feed.

2. Date-based DGA module

As stated previously, the output of a date-based DGA is based on date and/or time. Malware that uses this type of DGA usually has a limited number of variants (usually between 1 and 5). We perform initial reverse engineering of each variant and implement the algorithm in Python. The scheduler then runs this module every N hours (usually N=24). Finally, we get the output DGA feed for every day. We can also pregenerate domains, for example, for 30 days in advance and save them for future use.

3. Emulation DGA module

This is the most complex DGA and, consequently, the most interesting module type. As stated previously, emulation DGA utilizes the hardcoded seed and\or date to generate domains. The major problem is that the seed is usually represented as an integer (4 bytes), which means there are 4,294,967,295 possible variants of the DGA. We cannot predict which of these values will be hardcoded by the malware author into the samples that will appear in the wild. Bruteforcing and storing the results of domain generation for all 4,294,967,295 possible seed variants is definitely not feasible, especially when the malware, for example, outputs 5,000 domains for each variant for each day, resulting in a total of 21,474,836,470,000 domains per day.

We approached this differently. We prepared an environment that consists of our own DNS server on the host machine that accepts all DNS requests issued by the malware, and a bunch of specifically crafted virtual machines which are used for emulation and are controlled using modified version of *Cuckoo*.

Virtual machines are adopted for generating the entire list of domains that malware can produce. To achieve this, we implemented fixes for the following components of the Windows system:

1) NetBIOS name resolving
2) Microsoft Windows Sockets library
3) Different network related DLLs
4) *ZwDelayExecution* API

These fixes also allowed us to generate 4,000 domains in just **3 seconds**, while usual generation would take approximately **3 hours** (Tinba).

We then harvest all the samples (see III. Harvesting Samples for a detailed explanation) from the family that we are interested in, based on various detection parameters. These samples are placed in a queue.

The process of emulation runs in parallel. Each virtual machine takes one sample from the queue at a time. As soon as the emulation is finished, the VM moves on to the next sample in the queue.

When we run a malware sample, the result is a list of domains that are being generated. After performing some filtering, we take the first domain in that list and use it as a name for a category inside that family by appending a letter at the end. For example:

spaines.pw_A

If we encounter another sample that has generated the same first domain, but has other domains that are different, this means that it uses a different seed, but has the same "initialization vector." We create another category for this sample, appending another letter. For example:

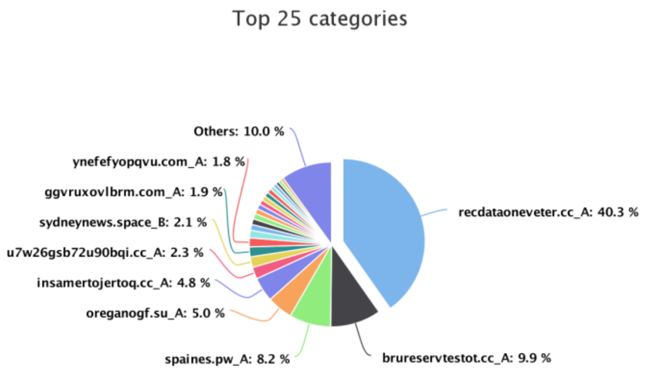
spaines.pw_B

Another possible scenario is if we encounter a sample that generated the same domains that we saw previously in another category. The hash of that sample will be labeled with an existing category.

After we run a number of samples, we end up with a list of categories that we can sort by prevalence, last seen date, and other parameters.

For example, let's look at the Tinba malware family. At the moment of writing this paper, DGALab processed more than 67,000 samples. We've found 115 distinctive categories, for which we have generated domain lists. We believe we fully cover the Tinba family and its derivatives.

Below is the graph with the most prevalent categories of the Tinba malware family.



4. *Feed providers DGA module*

Modules of this type parse publically available data sources, such as *CriticalStack Intel Feed* [3]. Each module parses a single data source. The results are then aggregated, post-processed, filtered and added to the final output.

C. Scheduler

The scheduler is a complex module that is responsible for performing various tasks. That's how the scheduler works:

- For date-based DGAs: Generates the cache of domains for at least 30 days in advance. After each day has passed, it will then generate the data for an additional day, so it operates as a "sliding window."
- For DGAs that require emulation: Gets new samples from available data sources, (skipping already processed files) and adds them to the execution queue in the virtualized environment.
- For DGA feed providers: Executes modules for each domain feed.

Finally, scheduler gathers all the results of each module type with the required meta information and composes a database that can be exported.

D. Aggregator

The aggregator is a module that takes domains from each of the 4 DGALab data sources and adds some meta information such as the date when a specific domain category was initially discovered, last seen date, prevalence of the domain category, number of domains in each category, etc.

III. HARVESTING SAMPLES

To effectively operate our *DGALab*, we need a large number of malware samples. We tried different sources, but ended up with *VirusTotal* (VT) as our main source.

We start by gathering all possible names and aliases for the families in which we are interested. This step is performed manually and only once, when we add a new family to the *DGALab*. Then, we make a request to VT for each alias of that family, so it will return the list of malware hashes. We perform post-processing (parsing of the reports) and compile them to one list. This list contains all hashes for a malware family, which has at least one or more detections with the same name according to VT. This step filters out a lot of samples and helps to make sure that the sample actually belongs to our family of interest.

Finally, we download all samples in the list and add them to the emulation queue.

IV. VM DETECTIONS PROBLEM

The most significant problem with emulating malware is that it might detect the emulation environment and stop running. We therefore try to be as noninvasive as possible.

From the beginning, we noted that *VirtualBox*, which is used in conjunction with *Cuckoo*, does not meet our requirements. After some testing, we decided to use *VMWare Workstation* instead.

First, we removed almost all the usermode hooks in the *CuckooMon* DLL. Basically, we don't really need them, as we gather all the required information from our own DNS server.

We then wrote a kernel mode driver that hides all virtual machine artifacts. This includes devices, files, registry keys, etc.

Finally, we adjusted the processor features and cleared the hypervisor flag in the processor, so the malware cannot detect the fact that it is running under the hypervisor (using *CPUID* assembler command).

All of these steps raised our successful detection rate by approximately 45%. We successfully generated domains even for the samples that failed to execute on *VirusTotal*.

V. USING OUR FEED

There are problems with sinkholing the large number of domains that modern malware can generate. For example, consider a malware that generates 20,000 domains per day but uses only 10 of them randomly. Sinkholing all of the domains is very ineffective, since 19,990 domains will be removed from legitimate registration for no reason. It's also quite a lengthy process to sinkhole a large number of domains, and it cannot be easily automated.

Therefore, the common approach is to inspect the traffic, track the domain names that are being accessed, identify malware and block its communications with C&C.

We have built an automated process that connects the DGALab with our Threat Intelligence knowledge base, so that every domain that is produced in our DGALab is automatically fed into our cloud infrastructure. It's then propagated to all security devices that deploy our technology in customers' networks.

When a DNS request is issued by a client, the security device checks if the domain name is marked as malicious. If it is malicious, the connection will be dropped and an alert is raised. This effectively stops data leaks from inside the network, as actual C&C communication does not occur.

ACKNOWLEDGMENTS

We would like to thank a member of our team, Stanislav Skuratovich, for helping us out with this project.

REFERENCES

[1] https://en.wikipedia.org/wiki/Domain_generation_algorithm

[2] http://www.cuckoosandbox.org/about.html

[3] https://intel.criticalstack.com/