Exploring the Human Fingerprints on Malware

Tobias Johansson and Robert M. Lee

Exploring the Human Fingerprints on Malware

By: Tobias Johansson[1] and Robert M. Lee[2]

Abstract

Much of the focus of cyber threat intelligence is countering adversaries and the tools and capabilities they leverage to do target organizations harm. Malware is a popular choice by many adversaries to fulfill their goals such as access development or destructive purposes. Malware contains a wealth of information to analyze for the purpose of cyber threat intelligence. The development, operationalizing, and utilization of malware is performed by humans and these human interactions leave traces of how the malware is leveraged, its configuration data, or even the choice of the malware itself. Malware is often not unique to specific adversaries but these traces, identified in the paper simply as human fingerprints, can be useful in clustering intrusions into sets for structured analysis and satisfying intelligence requirements. This is not a new concept and there are many researchers who take advantage of these practices today. The purpose of this paper is to introduce this concept to a wider audience and also structure it around the Diamond Model as a useful tool for analysis.

Malware as an Adversary Capability

Adversary is a loaded term in cyber threat intelligence simply meant as an abstract to define the attacker. What constitutes an adversary is entirely dependent on the defender and what level of information is useful to them to understand enough to achieve their goals, such as network defense. Some terms such as “threat actor” have been used to broadly describe adversaries in cyber threat intelligence but the term does not mean much more and has a lack of a widely agreed upon formal definition. For the purposes of this paper, the labeling of a threat activity group, or activity group for short, will be used to describe the features of adversary intrusions relevant for analyzing to meet intelligence requirements for network defense.[3]

The Diamond Model

In the concept of an activity group there are four primary features that should be extracted from adversary intrusions to form a set. These features are:

[1] Tobias is an alumni of SANS FOR578 – Cyber Threat Intelligence; this paper is a follow on to the class to extend the topics presented in the class to make them more widely available to the community
[2] Robert M. Lee is CEO of Dragos, Inc. and is the course author of SANS FOR578
[3] Activity Group is a term formally defined in The Diamond Model of Intrusion Analysis written by Sergio Caltagirone, Andrew Pendergast, and Christopher Betz http://www.activeresponse.org/wp-content/uploads/2013/07/diamond.pdf

- Adversary
- Infrastructure
- Capability
- Victim

The Adversary feature

The Adversary feature pertains to data associated with the individual, team, or organization responsible for performing the intrusion and can also relate to the adversary’s customers such as the separate person or group the adversary is serving with the intrusion. To a defender the data may seem to represent one entity and interaction; on the offense’s side though there could be multiple individuals, teams, and alliances that form all the components that go into an intrusion. It is in this feature the distinctive human choices will exist. Because of the lack of visibility the defender has into the adversary’s operations beyond the data in the intrusion it is important to note that the data placed in the Adversary feature is essentially an assessment by the defender. Said simply, so much of intrusions such as the existence of malware and infrastructure is a fact but the adversary’s choices, what they intended to do, what the data means about them, etc. is an assessment. As an example, if an adversary chooses to reuse code from one malware family in the development of another, that choice would be captured in the Adversary feature. Even though it seems to be related to malware which might otherwise be described under Capability; it is the choice of this code reuse and the access to it, as well as the defender’s assessment of what this means, that ties it to the Adversary feature. Simply noting that there is code similarities would be a fact, no assessment required, and thus be placed in the Capability feature. All of the observables abstracted out of intrusions for the Adversary feature are the result of analysis by the analyst performing the intrusion analysis and not facts.

The Infrastructure feature

The Infrastructure feature pertains to the physical or logical mechanisms that the adversary leverages to deliver their capability to the victim and carry out their intrusion. Command and control servers such as IP addresses and domains are some of the easiest infrastructure to identify, and it is possible to identify infrastructure at various steps of the adversary’s intrusion chain. As an example, a PDF file hosting the adversary’s malware is Infrastructure for that Capability in the Weaponization phase of the intrusion kill chain. It is often straightforward to extract out these observables for the Infrastructure phase. As an example, in analyzing malware an analyst may identify hard coded IP addresses the adversary uses to communicate to the malware. It is a fact that the IP address is present and it can be confidently placed in the Infrastructure feature of the Diamond Model. Some of the information gained may be inaccurate or not useful but the presence of them is not the result of analysis by the analyst and instead an observation.

The Capability (or Tactic, Technique, and Procedure) feature

The Capability feature of the Diamond Model is meant to capture the capability of the adversary; this can relate to their tools, such as malware, or their methods. This feature is where malware will most commonly preside. However, it is important to remember that not all capabilities are malware based. As an example, an adversary could leverage native commands and features of a system to perform their operation. In this example, the adversary's tactics, techniques, and procedures (TTPs) are highly interesting and worth documenting but would not reasonably be called malware. Like Infrastructure, the observables related to this feature are facts. The use of PoisonIvy malware by an adversary could be documented as a fact in the Capability feature and on its own is not of particular interest beyond tactical level defense recommendations or indicators of compromise.

The Victim feature

The Victim feature is the target of the adversary and their intrusion. The Victim feature can describe a variety of targets such as people, organizations, digital assets, or even physical locations. The observations, like Capability and Infrastructure, are factual. If an organization is compromised, it is reasonable to state that it is a Victim of the Adversary. However, as it relates to the Adversary, such as whether or not the Adversary intended to target the organization or not, those would be assessments and not facts and thus belong to the Adversary feature.

Using the Diamond Model

The Diamond Model seems straightforward at first to the point that a reasonable analyst would try to automate feature selection; however this cannot be done as the Diamond Model is a flexible model meant to be used differently based on the intelligence requirement the analyst is trying to satisfy. The key thing to note for this paper is that the adversary's choices, or human fingerprints, should be contained in the Adversary feature. They can be observed in the analysis of any other feature. As an example, who the Adversary intends to target by examining commonalities between multiple victims, their preference for types of command and control by analyzing the infrastructure chosen, and preference of malware families or exploits are all analysis that the analyst can do and abstract to the Adversary feature. Given every analyst's limited visibility, i.e. collection gaps, it is impossible to state that commonalities that exist are a factual reality intended by the adversary in the way the defender is perceiving them. Thus analysis is required which accounts for a significant amount of the work performed in cyber threat intelligence.

The rest of the paper will focus heavily on malware as a type of Capability the adversary can leverage and information that can be gathered from the analysis of malware for the purpose of filling out the four features of the Diamond Model. A particular focus will be given to the human fingerprints that make up the Adversary feature and guiding analysts where to look. The end goal will vary depending on the analyst's intelligence requirement. However, one of the goals used in the paper is to cluster different intrusions together to identify unique activity groups. Defender's that focus on activity groups instead of singular intrusions can more readily abstract lessons learned and key areas of focus for proactive defense work as well as anticipating future intrusions and posturing correctly.

Clustering of intrusion data

A simple form of clustering can be illustrated by the overview of PoisonIvy configuration values in Figure01. Configuration values used are CampaignID, Password, DNS Name and Port, the size of the nodes is based on how frequent the value is among the samples analyzed. There appears to be some common, or default, values frequently used by most instances of the malware. However, there are also instances which share no values with the rest of the samples, ending up in isolated islands in the graph. These islands could indicate multiple adversaries utilizing this tool, which would argue that the presence of this malware is probably not enough to tie a compromise to a certain activity group. It could also speak to different adversary choices that relate to different types of Infrastructure of Victims or even a developer shared between different adversaries that operationalize and configure the tool differently. In the case of PoisonIvy it is most likely being used by different adversaries, this is a common reality for tools as widespread as PoisonIvy.

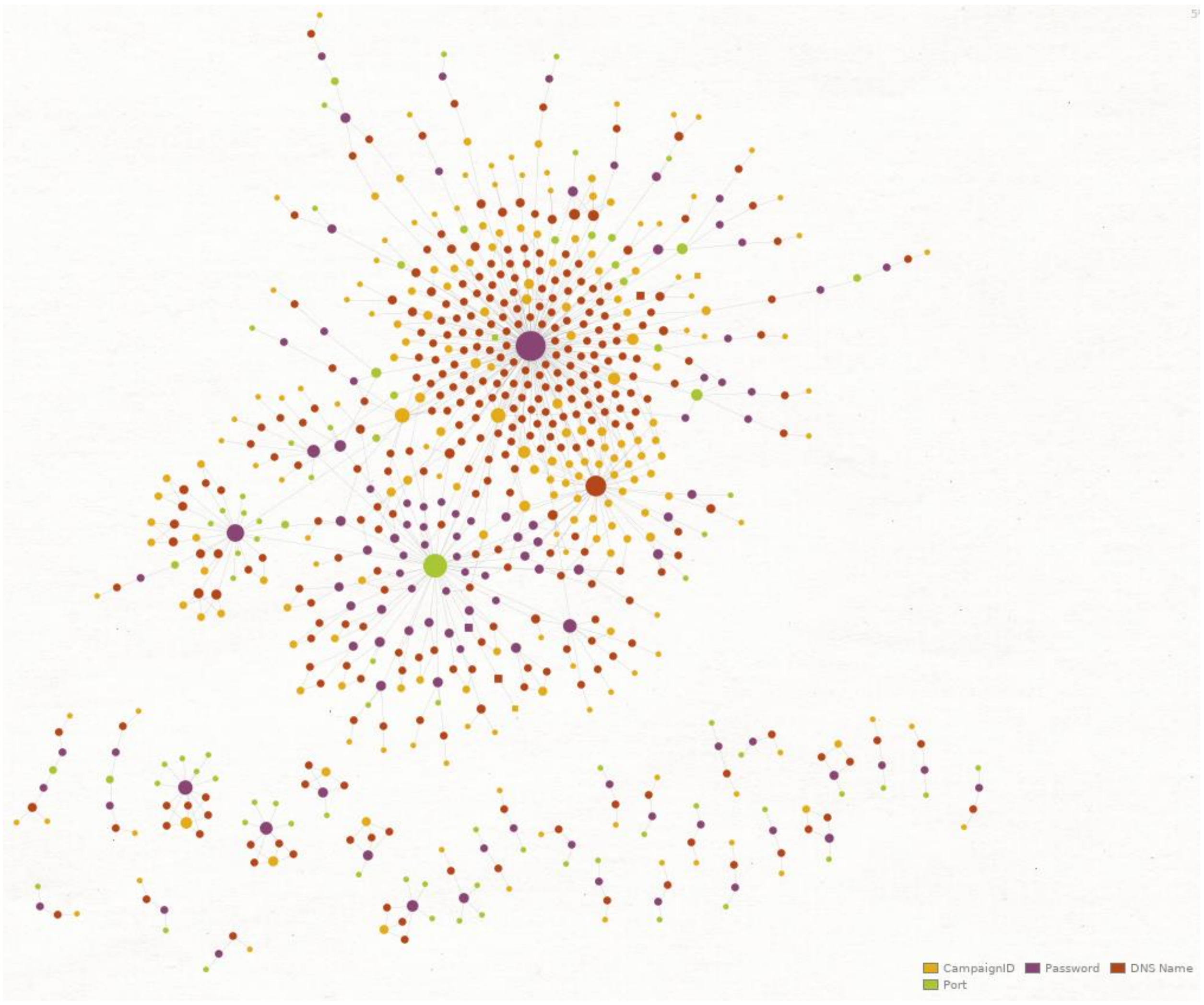


Figure01. Overview of PoisonIvy configurations, courtesy of malwareconfig.com

Three types of artifacts to look for

This section will discuss a few common locations for human fingerprints of the Adversary, on malware. The three categories demonstrated here are: header metadata, code reuse and configuration data. Each category will be presented with examples and how they might add to the Adversary feature.

Header Metadata

When compiling software the compiler leaves artifacts in the code. This could be a PDB path, compiler version etc. Some of them can be disabled, and some of them need to be patched after, or during, compiling and linking in order to be removed. These can sometimes reveal information about the development environment, the developer writing the code, or even internal names of the project itself. The artifacts often appears in various headers of the binary in a structured format.

Code reuse

Malware developers, much like developers of other types of software, may create a certain style over time, along with various utility functionalities helping them save time in new development projects. Code is often reused over multiple types of malware and in multiple versions. The cost of developing malware, with the infrastructure to use it properly, is an investment that the adversary may want to benefit from multiple times to increase return on value. Changing tools and infrastructure when not absolutely necessary is resource intensive and therefore reasonable adversaries must balance how often changes are made. Various concepts of AV evasion, as an example, is often leveraged to deliver the malware in a covert manner rather than rebuilding the logic from scratch.

Configuration data

The choices made by the humans behind the malware can reveal a great deal of information about functionality, intent, and usage of a sample. They can be the consequence of a logistical requirement offloading details from the command and control server to the victim, such as configuration within the malware. They could also be a consequence of the environment hosting the malware, such as Mutex values. They could also define a targeted execution environment such as specific internal IP addresses, applications on the victim machine, a subnet within a company, or perhaps a geographic region which either allows the malware to execute or self deletes. These choices, conscious or not, can reveal patterns to be found. When analyzing malware one must not forget that there are usually at least two parties with interests which form these values; the developers and the operators. In some cases there are also additional teams such as access development teams and infrastructure teams.[4]

Header Metadata

During compilation of Windows PE files the compiler adds some metadata, not always related to the execution of the code, which can reveal information of which compiler was used, what the path of the project is, when it was compiled, etc. Sometimes the developers of malware pass on their malware with PDB strings still in the binaries, this can lead to unintended information leaks

[4] Many intrusion analysts track adversaries almost exclusively off of malware and their configuration data. The risk of tracking adversaries by malware only is that you are more often tracking the malware developer, not the operator of the intrusion. In many complex adversary teams there are infrastructure temas, exploit development teams, malware teams, access development teams, and operators. The adversary should not be thought of as a single binary entity. Analyzing TTPs instead of just malware is an effective way to track the adversary’s operators instead of just the malware developers. This matters as different teams in any given organization can support different operations and simply grouping everything together as one monolithic adversary will make it difficult to analyze motivations, targets, and changing tradecraft.

which can be an opportunity to pivot for more data to analyze. Be careful though as sometimes all it brings is context or even confusion. Many PE file headers contains timestamp information, however not all of the timestamps will be available for any given PE file. This might be due to the structure not being compiled or the adversary might choose to modify the data. Valid timestamps are often expected to be very close to each other, within seconds or minutes to each other.

Compile timestamp and PDB strings

These GravityRAT samples were found with PDB strings left intact:

G1:
SHA256: 9f30163c0fe99825022649c5a066a4c972b76210368531d0cfa4c1736c32fb3a
Compiled: 2016-12-22 06:34:24
PDB: f:\F\Windows Work\G1\Adeel's Laptop\G1 Main Virus\systemInterrupts\gravity\obj\x86\Debug\systemInterrupts.pdb

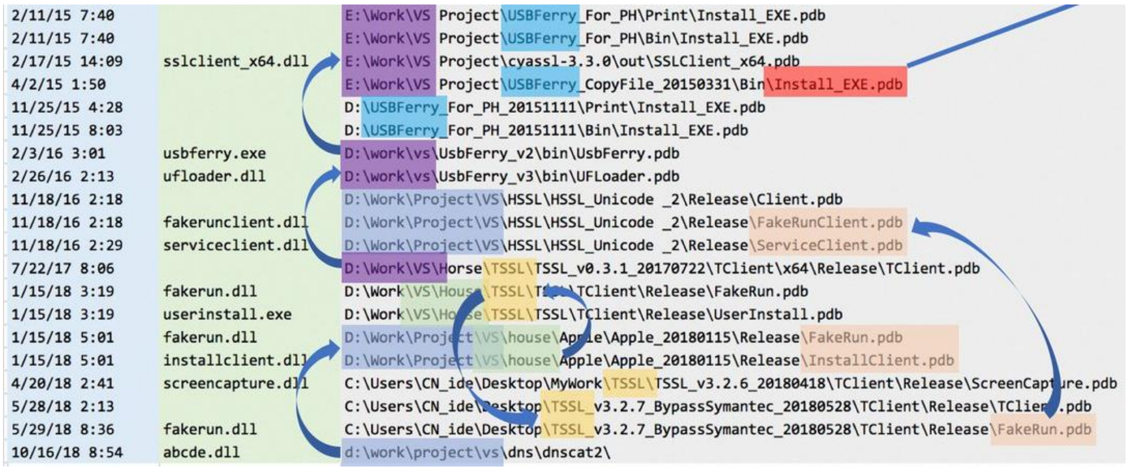
G2:
SHA256: 1993f8d2606c83e22a262ac93cc9f69f972c04460831115b57b3f6244ac128bc
Compiled: 2017-07-31 10:04:20
PDB: e:\Windows Work\G2\G2 Main Virus\Microsoft Virus Solutions (G2 v5) (Current)\Microsoft Virus Solutions\obj\Debug\Windows Wireless 802.11.pdb

G3:
SHA256: 99dd67915566c0951b78d323bb066eb5b130cc7ebd6355ec0338469876503f90
Compiled: 2017-08-21 21:28:31
PDB: F:\Projects\g3\G3 Version 4.0\G3\G3\obj\Release\Intel Core.pdb

GX:
SHA256: 1c0ea462f0bbd7acfdf4c6daf3cb8ce09e1375b766fbd3ff89f40c0aa3f4fc96
Compiled: 2017-12-06 07:52:11
PDB: C:\Users\The Invincible\Desktop\gx\gx-current-program\LSASS\obj\Release\LSASS.pdb

The PDB path can sometimes reveal versions, user name, project names, targeted platform, architecture and other pieces of information. In this case the development appears to be done in different generations, G1 to GX, which can be tied to different periods of time. This could help building a timeline of the development of a malware type, or perhaps bring some insight to how mature a campaign is.

Another great example of when pivoting PDB strings can bring results can be seen in Figure02. This overview illustrates how pivoting of a PDB path, d:\work\project\vs\, can lead to multiple new malware samples to be analyzed.



2/11/15 7:40		E:\Work\VS Project\USBFerry_For_PH\Print\Install_EXE.pdb
2/11/15 7:40		E:\Work\VS Project\USBFerry_For_PH\Bin\Install_EXE.pdb
2/17/15 14:09	sslclient_x64.dll	E:\Work\VS Project\cyassl-3.3.0\out\SSLClient_x64.pdb
4/2/15 1:50		E:\Work\VS Project\USBFerry_CopyFile_20150331\Bin\Install_EXE.pdb
11/25/15 4:28		D:\USBFerry_For_PH_20151111\Print\Install_EXE.pdb
11/25/15 8:03		D:\USBFerry_For_PH_20151111\Bin\Install_EXE.pdb
2/3/16 3:01	usbferry.exe	D:\work\vs\UsbFerry_v2\bin\UsbFerry.pdb
2/26/16 2:13	ufloader.dll	D:\work\vs\UsbFerry_v3\bin\UFLoader.pdb
11/18/16 2:18		D:\Work\Project\VS\HSSL\HSSL_Unicode _2\Release\Client.pdb
11/18/16 2:18	fakerunclient.dll	D:\Work\Project\VS\HSSL\HSSL_Unicode _2\Release\FakeRunClient.pdb
11/18/16 2:29	serviceclient.dll	D:\Work\Project\VS\HSSL\HSSL_Unicode _2\Release\ServiceClient.pdb
7/22/17 8:06		D:\Work\VS\Horse\TSSL\TSSL_v0.3.1_20170722\TClient\x64\Release\TClient.pdb
1/15/18 3:19	fakerun.dll	D:\Work\VS\House\TSSL\TSSL\TClient\Release\FakeRun.pdb
1/15/18 3:19	userinstall.exe	D:\Work\VS\House\TSSL\TSSL\TClient\Release\UserInstall.pdb
1/15/18 5:01	fakerun.dll	D:\Work\Project\VS\house\Apple\Apple_20180115\Release\FakeRun.pdb
1/15/18 5:01	installclient.dll	D:\Work\Project\VS\house\Apple\Apple_20180115\Release\InstallClient.pdb
4/20/18 2:41	screencapture.dll	C:\Users\CN_ide\Desktop\MyWork\TSSL\TSSL_v3.2.6_20180418\TClient\Release\ScreenCapture.pdb
5/28/18 2:13		C:\Users\CN_ide\Desktop\TSSL_v3.2.7_BypassSymantec_20180528\TClient\Release\TClient.pdb
5/29/18 8:36	fakerun.dll	C:\Users\CN_ide\Desktop\TSSL_v3.2.7_BypassSymantec_20180528\TClient\Release\FakeRun.pdb
10/16/18 8:54	abcde.dll	d:\work\project\vs\dns\dnscat2\

Figure02. @stvemillertime overview of PDB string pivoting.[5]

Although the PDB string is a compiler artifact, it contains human fingerprints in the form of names of the project and folders in the path chosen by the adversary.
In some cases the purpose of a malware sample may be revealed like this ChChes sample:

SHA256: cb0c8681a407a76f8c0fd2512197aafad8120aa62e5c871c29d1fd2a102bc628
Compiled: 2016-10-28 23:14:12
PDB: D:\Projects\ByPassAV\WinCConnect\Release\WinCConnect.pdb

It would appear as if the origin of this project was the need to evade AV detection. This might bring more context to the malware, and add to the Adversary feature, but can be hard to pivot on in order to gain further insight.

In these examples PDB strings, as well as compilation timestamps, are examples of Adversary choices. A PDB string can contain plenty of useful information about the malware itself, the environment it is compiled in and the developer writing it. It can also be used to pivot to additional samples. A single compilation timestamp will not bring clarity on its own, but when analyzed over multiple intrusions an analyst could assess time zone and working hours of the malware developer.

Windows PE Rich header

In the Microsoft portable executable format there is a undocumented section called the Rich header which contains information about the compilation and linking environment of the binary. In the deobfuscated header the compilers, along with their version and reference count, involved in the compiling and linking of the binary are documented. There are multiple strategies

[5] @stvemillertime tweet on PDB string pivoting.
https://twitter.com/stvemillertime/status/1059650999034503170

to analyzing the Rich header, one of which is by calculating a RichPV hash. A RichPV-hash[6] can be used as a fingerprint of a specific compiling and linking environment. A match in RichPV-hash can be seen as an overlap in build environment between two binaries, which could strengthen the relationship between the two binaries as they would appear to be compiled in the same environment. The concept can be demonstrated by comparing the RichPV-hash of these Shamoon samples.

```
SHA256: 0694bdf9f08e4f4a09d13b7b5a68c0148ceb3fcc79442f4db2aa19dd23681afe
Compiled: 2011-11-28 14:53:13
RichHash: 7d6f2f3f47aefcac530615dc2e24fa21
RichPV: 27c24b83252ccd0f941a78af91221cb7

SHA256: 0975eb436fb4adb9077c8e99ea6d34746807bc83a228b17d321d14dfbbe80b03
Compiled: 2011-11-28 15:50:45
Rich Hash: 97843d6d76b0f404fc4d4b3cde0cc09a
RichPV: 27c24b83252ccd0f941a78af91221cb7

SHA256: c3ab58b3154e5f5101ba74fccfd27a9ab445e41262cdf47e8cc3be7416a5904f
Compiled: 2011-11-28 15:50:59
Rich Hash: d5ab8070feb092df47c98e2fd190ddfb
RichPV: 27c24b83252ccd0f941a78af91221cb7
```

Since the RichPV-hashes matches between these samples they would appear to be compiled in the same environment. The Rich section contains a checksum which can be used for integrity check of the data, an invalid checksum indicates post compilation modifications of the data. The data can be modified with a hex editor, as seen with Olympic Destroyer samples which was seen as a false flag attribution attempt. Because the adversary controls the data in the Rich header it will not always be useful. If the adversary chooses not to tamper with the data it can provide an insight into the build environment of the adversary, it can also be used to associate malware to the adversary.

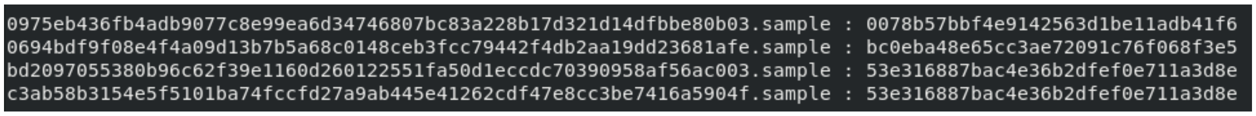
Code reuse

Analysis of code overlap may indicate code reuse and help identify new versions of a malware or even new malware types developed by the same adversary. It can be hard to quickly identify code overlap among malware samples though; sometimes YARA-signatures or automated tools can help identify overlaps but often it takes a trained analyst to perform malware analysis. There are several strategies available for automated code overlap analysis, a subset of them will be presented below.

[6] The concept of RichPV-hash and using the Rich header for Malware detection was defined in Leveraging the PE Rich Header for Static Malware Detection and Linking by Maksim Dubyk https://www.sans.org/reading-room/whitepapers/reverseengineeringmalware/paper/39045

Import hash

In order for PE binaries to leverage external functionality during execution the required function is imported, typically from common Windows DLLs or other 3rd party libraries. Depending on the functionality of the binary different functions will be imported. By hashing the name of the libraries and functions we get what is called an imphash, short for import hash[7]. The imphash can be used to identify related malware samples, these will typically be very similar as the imphash fingerprints the external dependencies of a binary. Figure03 shows the imphash of four Shamoon samples, where two of the samples share imphash. This also shows how a minor adjustment by the adversary, perhaps some new functionality that requires additional external dependencies or refactoring of code or streamlining of logic which removes external dependencies, will result in a different imphash.



```
0975eb436fb4adb9077c8e99ea6d34746807bc83a228b17d321d14dfbbe80b03.sample : 0078b57bbf4e9142563d1be11adb41f6
0694bdf9f08e4f4a09d13b7b5a68c0148ceb3fcc79442f4db2aa19dd23681afe.sample : bc0eba48e65cc3ae72091c76f068f3e5
bd2097055380b96c62f39e1160d260122551fa50d1eccdc70390958af56ac003.sample : 53e316887bac4e36b2dfef0e711a3d8e
c3ab58b3154e5f5101ba74fccfd27a9ab445e41262cdf47e8cc3be7416a5904f.sample : 53e316887bac4e36b2dfef0e711a3d8e
```

Figure03. Imphash calculated on Shamoon samples.

Fuzzy hashing

The digital hash of files is unique and thus finding commonality between two different samples of malware by comparing the digital hash would not be useful. Fuzzy hashing though aims at identifying if the contents of two files are similar, often with a measure of “how similar” the two samples are. Ssdeep[8] is a fuzzy hashing technique which can identify sequences of bytes in the same order that exists in two files. Ssdeep creates a hash which can be used to measure the distance, or similarity, to another ssdeep hash. TLSH[9] is a fuzzy hashing technique which provides another method of fuzzy hashing, which also provides a measure of distance between files. A challenge for fuzzy hashing concepts is that source code can look dramatically different in its compiled form when modifying compiler flags, using a new version of the same, or another, compiler. As with any other form of comparison of binaries they need to be compared in their original form, packed samples and any form of obfuscation will degrade the quality of the measure.

YARA function matching

A YARA[10] rule can be used to describe the byte pattern of a function in a binary, which can be used to identify the function in other samples without reverse engineering them. By capturing

[7]The concept of import hashing was defined by FireEye in Tracking Malware with Import Hashing https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html

[8] The concept of ssdeep is provided by the ssdeep Project
https://ssdeep-project.github.io/ssdeep/index.html

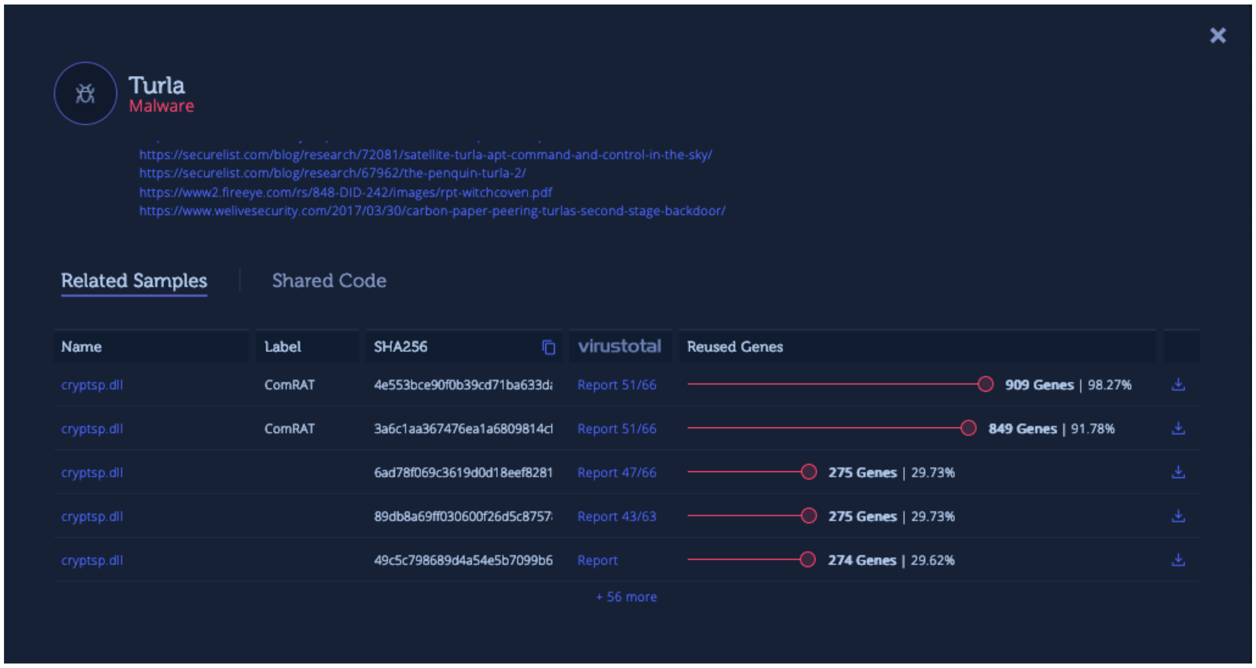
[9] Smart Whitelisting Using Locality Sensitive Hashing
https://blog.trendmicro.com/trendlabs-security-intelligence/smart-whitelisting-using-locality-sensitive-hashing/

[10] YARA was originally developed by Victor Alvarez of VirusTotal
https://virustotal.github.io/yara/

the byte pattern of a function, and wildcarding location dependant items and calls, the yara rule becomes a fingerprint for the sequence of bytes representing the logic of the function. A minor change in the logic of the function or compiler variations, as mentioned under fuzzy hashing, can easily result in zero matches. YARA is currently a popular choice both for matching malware and functionality within malware.

Community tools

Multiple tools for analyzing code overlap, or binary similarity, has been developed. Intezer has developed a service which can help identify code reuse among samples. This is done by disassembling the code into an intermediary language and generating tokens, or “genes”, which can be indexed and queried when comparing samples. This is an automated process, which is many times faster than manual analysis, that can help identifying samples originating from the same developer or team of developers, assuming there is a shared code base. Figure04 shows an example analysis of a Turla malware with code overlap, represented as “Genes”, of other files.



https://securelist.com/blog/research/72081/satellite-turla-apt-command-and-control-in-the-sky/
https://securelist.com/blog/research/67962/the-penquin-turla-2/
https://www2.fireeye.com/rs/848-DID-242/images/rpt-witchcoven.pdf
https://www.welivesecurity.com/2017/03/30/carbon-paper-peering-turlas-second-stage-backdoor/

Name	Label	SHA256	virustotal	Reused Genes
cryptsp.dll	ComRAT	4e553bce90f0b39cd71ba633d	Report 51/66	**909 Genes** | 98.27%
cryptsp.dll	ComRAT	3a6c1aa367476ea1a6809814c	Report 51/66	**849 Genes** | 91.78%
cryptsp.dll		6ad78f069c3619d0d18eef8281	Report 47/66	**275 Genes** | 29.73%
cryptsp.dll		89db8a69ff030600f26d5c8757	Report 43/63	**275 Genes** | 29.73%
cryptsp.dll		49c5c798689d4a54e5b7099b6	Report	**274 Genes** | 29.62%

+ 56 more

Figure04. Intezer analysis of Turla ComRAT[11].

Various utility functionality

What could be classified as utility functionality is often reused among multiple malware samples. Typical examples could be how the actor performs update of file MAC-times, adding firewall rules, enabling “SeDebugPrivilege” access token, writing log output to a file, executing a shell

[11] Intezer analysis of Turla ComRAT
https://analyze.intezer.com/#/files/4e553bce90f0b39cd71ba633da5990259e185979c2859ec2e04dd8efcdafe356/families/3ac8a718-fc2d-47d4-a000-7d6fdc9879cc

command, gaining persistence, etc. Typical utility functionality are the interactions with the OS needed to perform its main purpose, in order to ease development these are often reused once proved successful. These functions adds to the understanding of a “developer style” which can be associated with the activity group and recognized over multiple samples. Intezer[12] released a blog post on their research with McAfee on code reuse among malware samples that has been attributed as North Korean. One of the functions mentioned launches cmd.exe with a net share, which can be identified in multiple samples. This type of code reuse will speed up the development process of new tools for the Adversary, and it will also increase the overlap with previous tools. But, there are caveats with analyzing code overlap. Open-source libraries, snippets from public/private forums etc. may overlap with multiple samples but that might not be unique to the adversary you are interested in.

As discussed in this section, an adversary choice might be to reuse code. A minor code overlap between samples should not result in a definitive decision that they originate from the same Activity group. However, when analyzing malware from multiple intrusions and the reuse of code becomes more and more evident among malware samples, the analyst can assess that the samples are likely to originate from the same activity group. The adversary choice of reusing code can be an opportunity for analysts to cluster intrusions into sets and enrich the Adversary feature with further insights.

Configuration data

Malware executing on a host needs certain resources to perform its task such as how it is supposed to execute, where it should execute or install code, if it should import functionality from the host, ensuring there is only one instance of the malware running on the system, and command and control communications; each of these can contain vital information. Sometimes this results in valuable information on malware samples such as Mutex values, date formatting, or a preferred way of persistence. Most of these would consist of operational requirements potentially from the operators of the intrusion, however some of these would originate from the developer of the malware.

PE’s version info

The version information of a PE, containing “CompanyName”, “FileDescription”, “InternalName”, “OriginalFilename” etc. is often subject to various attempts at diverting suspicions. Some adversaries use these values to make the file blend in, perhaps by copying these values from a legit binary or by creating plausible values. Frequent typos, forgotten copyright symbols, incorrect casing of names, spelling and thematic mistakes can be observed. These values will appear as strings in a structured format in the headers of a binary. These are typically revealed during static analysis.

[12]Examining Code Reuse Reveals Undiscovered Links Among North Korea’s Malware Families https://www.intezer.com/examining-code-reuse-reveals-undiscovered-links-among-north-koreas-malware-families/

Looking at a reported use of a customized QuasarRAT sample we find that the adversary may have intended to blend in among native Windows binaries, but made spelling mistakes:

SHA256: af41e9e058e0a5656f457ad4425a299481916b6cf5e443091c7a6b15ea5b3db3
LegalCopyright: Microsoft Cotporation. All rights reserved.
InternalName: Dark-savage.exe
FileVersion: 2.2.1
ProductName: Microsoft Windows Operation System

Looking at another QuasarRAT sample associated with the same activity group we find string reuse:

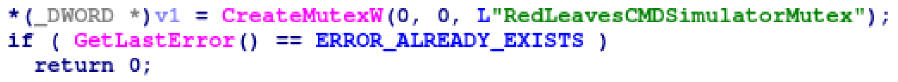
SHA256: 0b3610524ff6f67c59281dbf4a24a6e8753b965c15742c8a98c11ad9171e783d
LegalCopyright: Microsoft Cotporation. All rights reserved.
InternalName: DA.exe
FileVersion: 1.1.2
ProductName: Microsoft Windows Operation System

Findings like these can add to the general understanding of an activity group. Strings like these can allow for pivoting to find new interesting samples. Pivoting of a small typo like this is not guaranteed to give meaningful results, but it may yield samples which may be interesting to analyze. Frequent spelling mistakes on malware by an adversary may suggest that the malware development process lacks quality assurance and includes manual steps. Even this information, collected over time and with other information, could help the defender understand the maturity of the adversary and their operations.

Mutex/Event/Semaphore values

Malware sometimes need to share resources internally just like non-malicious applications. A Mutex is *"A synchronization primitive that can also be used for interprocess synchronization"*[13], an Event or a Semaphore can be used in the same way. A common use of these are to ensure that there's only one instance of the malware running on a victim system. The string value used can sometimes reveal a theme like keyboard walks, magic/lucky numbers, if they are generated by a logic, or simply the file hash. Mutex values appear as strings in a binary, these are typically revealed during static or dynamic analysis.

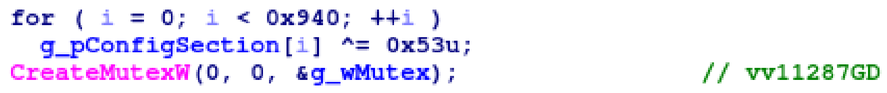
Figure05 and Figure06 shows a sample of RedLeaves that reveals multiple mutexes being registered.



```
*(_DWORD *)v1 = CreateMutexW(0, 0, L"RedLeavesCMDSimulatorMutex");
if ( GetLastError() == ERROR_ALREADY_EXISTS )
  return 0;
```

[13] Microsoft .Net documentation "Mutex Class"
https://docs.microsoft.com/en-us/dotnet/api/system.threading.mutex?view=netframework-4.7.2

Figure05. Mutex value “RedLeavesCMDSimulatorMutex” used.

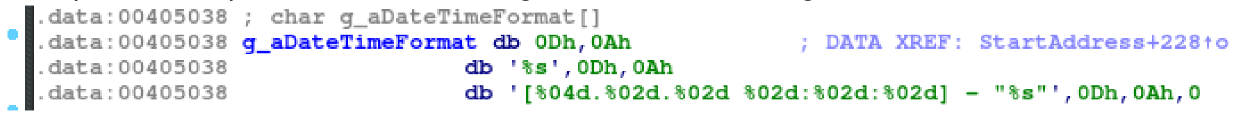

```
for ( i = 0; i < 0x940; ++i )
  g_pConfigSection[i] ^= 0x53u;
CreateMutexW(0, 0, &g_wMutex);                    // vv11287GD
```
Figure06. Mutex value “vv11287GD” used.

The first Mutex string is reused across multiple samples, possibly across multiple different operations. The second mutex string can be unique for this sample, or possibly the operation which the samples belongs to. A strategy like this could be used to prevent multiple infections within the same operation and across operations for this malware.

Formatting of text

Text meant for humans is often formatted to the preference of the creators, this can be log output still in the binary or the structure of command and control messaging. Date formatting can vary significantly depending on the region the style order is intended for. Formatting can be out of convenience so the operator can read the date string in a familiar format or to enable other tools further down in the operator tool chain to correctly parse the date value. Text formatting styles typically appears during static and dynamic analysis.

A sample of KiloAlpha malware shows log file date format being used:


```
.data:00405038 ; char g_aDateTimeFormat[]
.data:00405038 g_aDateTimeFormat db 0Dh,0Ah          ; DATA XREF: StartAddress+228↑o
.data:00405038                   db '%s',0Dh,0Ah
.data:00405038                   db '[%04d.%02d.%02d %02d:%02d:%02d] - "%s"',0Dh,0Ah,0
```
Figure07. Date format string of yyyy.dd.mm in malware.

Using this format string the decrypted log file entries will become human readable:

```
-----------------------------------------------------------------------
[2018.08.13 14:15:13] - "Windows Explorer"
-----------------------------------------------------------------------
[2018.08.13 14:15:22] - "Administrator: C:\Windows\System32\cmd.exe"
whoami
```

This date format is commonly used in Asian countries[14]. A unique, or less common, way of formatting output can add to the understanding of adversary preference. This should not be seen as evidence of the origin of an Adversary, but rather as a clue to the preferences of the developers or operators using the malware which could contribute to a stronger assessment with more information over time.

14 Wikipedia Date format by country
https://en.wikipedia.org/wiki/Date_format_by_country

Method of persistence

There are several methods of gaining persistence such as autostart variants, services, and scheduled tasks. Hiding in plain sight, or imitating other non-malicious software, is common in order to blend in. In order to succeed, file and folder names, service name and description, registry keys etc. needs to seem non-malicious or normal to the system and its users. Although the themes can vary significantly, the type of persistence can often be reused if proven successful. This can lead to code reuse during the installation of a malware. How the malware persists on the victim system is typically revealed during static or dynamic analysis.

Looking at the persistence from two KiloAlpha samples we find a pattern:

File path: %ALLUSERSPROFILE%\Intel\MsMpEng.exe (if victim machine Windows Vista or newer)
File path: %ALLUSERSPROFILE%\Application Data\Intel\MsMpEng.exe (if victim machine is Windows XP or older)
Registry key for autostart: HKLM\Software\Microsoft\CurrentVersion\Run\Windows IME Update

File path: %ALLUSERSPROFILE%\Intel\mscorsw.exe (if victim machine Windows Vista or newer)
File path: %ALLUSERSPROFILE%\Application Data\Intel\mscorsw.exe (if victim machine is Windows XP or older)
Registry key for autostart: HKLM\Software\Microsoft\CurrentVersion\Run\Mozilla Update

These details can be useful for clustering intrusions although it may be hard to pivot off of these strings to find new relevant samples. However, these details can also help focus incident responders on single cases.

Communication details

A common method of operating malware is through callbacks to a command and control server which can forward tasking to the malware. This requires, at some point, communication details to be included in the malware. Domains, IPs, and ports are common indicators found in malware and activity group reporting. These pieces of information are some of the easiest to change for adversaries but still provide valuable information, pivot points, and human fingerprints of the choices made which may represent some new details about the adversary. As with persistence, domains and urls are often chosen to blend in or mimic a legitimate service. The location of the infrastructure used, hacked or bought, can reveal preferences for certain VPS vendors or themes in an operation. How the malware communicates is often revealed during static or dynamic analysis.

Taking another look at one of the GravityRAT samples we find a set of command and control details:

SHA256: 99dd67915566c0951b78d323bb066eb5b130cc7ebd6355ec0338469876503f90
http://coreupdate.msoftupdates.com:46769
http://msupdates.mylogisoft.com:46769
http://updateserver.msoftupdates.eu:46769

http://coreupdate.msoftupdates.com:46769/G3/ServerSide/G3.php
http://msupdates.mylogisoft.com:46769/G3/ServerSide/G3.php
http://updateserver.msoftupdates.eu:46769/G3/ServerSide/G3.php

These plaintext strings reveals three command and control domains with an uncommon port being used. The URLs also indicate there could be an opportunity to pivot to find new command and control servers or samples. Additionally, the choice of the specific domains may reveal a preference by the adversary or have some value to blending in to the victim's expected network traffic.

As discussed in this section, adversary choices can contain plenty of different types of data. Malware might need to contain multiple adversary choices in order to function as intended. Adversary choices in the form of operational requirements may result in specific communication patterns and command and control hosts used. It may also decide which persistence methods will be used or how the malware is supposed to be perceived by a user. These choices can appear as preference, where certain communication ports, protocols or structures are favored.

Case Study: HEXANE

In this section, the activity group HEXANE's malware will be leveraged to demonstrate the concepts previously discussed in the paper. In looking at some DanBot samples associated with the HEXANE activity group we can identify several human fingerprints. In the cluster of intrusions a VBA macro enabled document delivered the DanBot malware which communicates over DNS or HTTP with the command and control infrastructure. The DanBot samples have been observed with different command and control configurations, as standalone applications, and as plugins/modules. Below follows a breakdown of some of the interesting artifacts found in these malware samples, which can be used to build an understanding of the tools, preferences and behaviors of the activity group.

Header Metadata

Most of the samples discovered contains useful compiler information. Table01 shows an overview of the samples analyzed. It should be noted that the PDB string, and compile time, can be patched with a hex editor.

SHA256	**Compile time**	**File name**	**PDB String**
aa7ef56643d294b442d60137f5a8de15cd8472ecabdad09c9fd7cf64446a35c0	2018-09-10 09:21:16	UpdateCreator.dll	D:\Bot\Command\UpdateCreator\UpdateCreator\obj\Debug\UpdateCreator.pdb
b767daab16272144f09db405eec72e42f986e7683753a2c1e143cdbe385818e2	2018-09-10 09:24:09	GooglUPDT.exe	D:\Bot\GoogleUpdates\GoogleUpdates\obj\Debug\GooglUPDT.pdb
3588d6a0837409035b4e2ae28fd27224bd487fc8ddf7ed8bf6898a	2018-09-13 08:04:46	GooglUPDT.exe	D:\Bot\GoogleUpdates\GoogleUpdates\obj\Debug\GooglU

3ff3df275f			PDT.pdb
d6c7872e9a8c921c6027a089d2e96424c3846de08e9522319cad1e190b42291d	2018-11-26 07:42:53	UpdateEngine.dll	D:\Bot\Command\UpdateCreator\UpdateCreator\obj\Debug\UpdateEngine.pdb
30eb4698adb0bb690f3b0f8911cede411f99356e1b56d9d8a882ddde105ad83f	2018-12-09 11:24:18	Google_Updates.exe	D:\Bot\GoogleUpdates\GoogleUpdates\obj\Debug\Google_Updates.pdb
ceedc02e6338c7027d82b4a3a4a43ad971a0342a6f8fa27c47a2520d00bc1a1e	2018-12-22 15:44:16	RunCal.dll	D:\Bot\BackDor\Command\UpdateCreator\UpdateCreator\obj\Debug\RunCal.pdb
11c52732d7fde12f5f4c6431f8be876ffd73acdd725c4b908b257be1b007a290	2018-12-23 10:22:38	HPServerManager.exe	C:\Users\T\Desktop\bbbbb\GoogleUpdates\GoogleUpdates\obj\Debug\HPServerManager.pdb
5768d9d503d01331182b980b41b8fb02a269825e89d3e33e08e6de6f5ffa2024	2018-12-25 00:28:29	HPServerManager.exe	D:\Bot\BackDor\GoogleUpdates\GoogleUpdates\obj\Debug\HPServerManager.pdb
4c4cc3473e050b83943e58548a71c72603a934b2daba6d57fd75908323d32776	2018-12-27 13:28:54	vmware_authd.exe	D:\Bot\BackDor2\GoogleUpdates\GoogleUpdates\obj\Release\vmware_authd.pdb
72f78276ea06649556c3beaa5a53f1b3faa5e4b2fe094f1e84cc959c70139c02	2053-03-16 00:54:01	AppTrace.exe	N/A
28d8dd812f6b5f1e6e96ff9824054d2141a715ff65b2d7e15e9407bc376b55ed	2053-03-16 00:54:01	Atrace.exe	N/A
1b9c7363f67537ba3255804c429b349ec2ced8bd03ec4bcd6cf8a81fcf34e20a	2053-03-16 00:54:01	Atrace.exe	N/A
10d0d53f5e5f34c424431492fa4ee95eb2fa4fe6327455384cf508c586dd2851	2060-01-06 14:55:58	AdobeReport.exe	C:\Users\Matt\Desktop\source\New\DanBot\AdobeReport\obj\Debug\AdobeReport.pdb

Table01 Compiler artifacts

The discovered samples would suggest that HEXANE started the development of the DanBot malware at, or prior to, 2018-09-10. The file names are similar to legitimate-looking applications or services and the developer(s) appear to favor D:\Bot\ as project path when compiling. Some of the latter samples appear to be patched by HEXANE, since compile times and PDB paths seem invalid, which could indicate a shift in focus. When attempting to cluster these intrusions, prior to the creation of the HEXANE activity group, analysts could use these observations and assessments about them in the Adversary feature in the Diamond Model which would assist in linking seemingly disparate intrusions together.

When creating a new .Net project in Visual Studio the assembly will be assigned a new Globally Unique ID (GUID), for each build of the project the assembly is assigned a new Module Version ID (MVID). With this in mind, we can attempt to group malware by original project GUID and to identify post-compilation modifications of .Net assemblies.[15] GUID and MVID are also available in VirusTotal analysis of .Net binaries. It should be noted that, as with PDB strings and compile times, these can be patched with a hex editor.

SHA256	**GUID**	**MVID**
aa7ef56643d294b442d60137f5a8de15cd8472ecabdad09c9fd7cf64446a35c0	b142fda2-f6a5-47aa-9ab2-1182bb742504	29b20d67-e381-445d-bef5-c2825f11c8b1
b767daab16272144f09db405eec72e42f986e7683753a2c1e143cdbe385818e2	f7cf0cbf-5779-4269-a15d-c4bac5c47ee0	f46fdd34-3e37-4993-9bc5-cfcee4257d10
3588d6a0837409035b4e2ae28fd27224bd487fc8ddf7ed8bf6898a3ff3df275f	f7cf0cbf-5779-4269-a15d-c4bac5c47ee0	4b45f52c-fac9-4fc2-b60b-1a6de2bda7f5
d6c7872e9a8c921c6027a089d2e96424c3846de08e9522319cad1e190b42291d	e1e4985d-c36d-4315-a05a-61516a20a6ea	dc2e0b65-c6d8-4658-adcc-25e6eca896cc
30eb4698adb0bb690f3b0f8911cede411f99356e1b56d9d8a882ddde105ad83f	b49c031b-8899-456f-b894-adc36cb07419	3859f683-2f13-4fb6-ba1a-066d00abb889
ceedc02e6338c7027d82b4a3a4a43ad971a0342a6f8fa27c47a2520d00bc1a1e	4db75cc7-4910-4c47-af9a-c71e32cfca4a	4d599c20-1f36-4fc1-aba6-e3634072824b
11c52732d7fde12f5f4c6431f8be876ffd73acdd725c4b908b257be1b007a290	a092ab93-b78c-4586-a2b5-f6245844eec3	4c7db0fb-e732-40c2-9506-21014a8f24aa
5768d9d503d01331182b980b41b8fb02a269825e89d3e33e08e6de6f5ffa2024	a092ab93-b78c-4586-a2b5-f6245844eec3	b45ad9bb-6b00-4d09-bc9e-c73ff59e3ac3
4c4cc3473e050b83943e58548a71c72603a934b2daba6d57fd75908323d32776	338b4776-d3af-4f5e-a026-cae0b25f26d8	9a5cad08-223a-4c6f-9176-06d5a4383d6b
72f78276ea06649556c3beaa5a53f1b3faa5e4b2fe094f1e84cc959c70139c02	b63341d9-25ee-4a07-9255-041788e0b98c	2706e42e-cacf-4ea5-810c-710be80b9e95
10d0d53f5e5f34c424431492fa4ee95eb2fa4fe6327455384cf508c586dd2851	06c5f1c6-ebf6-4b5e-8342-291ab23538f1	7ae6bc01-53c0-42b0-bbc5-a6fdb76493ae

Table02 .Net GUIDs and MVIDs

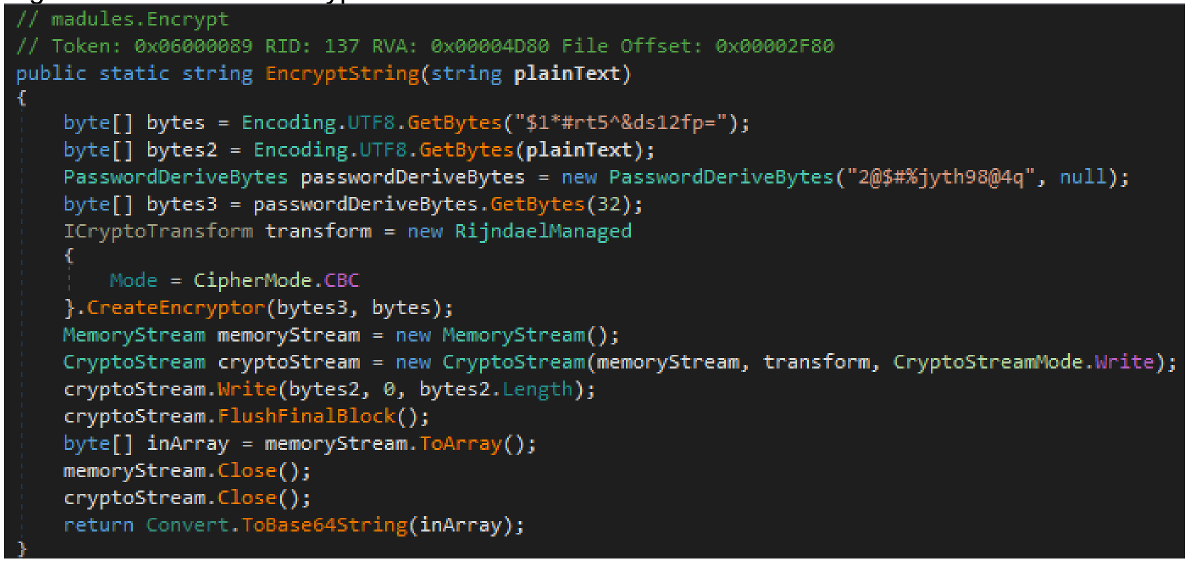
The distribution of GUIDs in the table would suggest some of the samples originate from the same Visual Studio project but with different MVIDs. The amount of unique GUIDs, although plenty of code is shared, would suggest most of the samples are created from new Visual Studio projects with shared code being used as needed.

[15] You, Me, and .NET GUIDs by Brian Wallace
https://threatvector.cylance.com/en_us/home/you-me-and-net-guids.html

Combining the information from Table01 and Table02, an assessment could be made that the samples sharing GUID are possibly builds of the same project. Samples sharing GUID, but compiled in different folders, could suggest there are multiple developers sharing the code or multiple machines were used to build the malware or that it is the same developer using two different computers.

Code reuse

Among the DanBot samples analyzed there is code reuse that could further tie the intrusions together. Among the samples there is also functionality for encryption and decryption of data, Figure08 shows the encryption function.

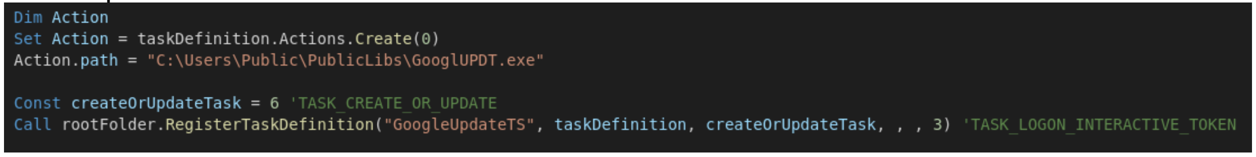


```
// madules.Encrypt
// Token: 0x06000089 RID: 137 RVA: 0x00004D80 File Offset: 0x00002F80
public static string EncryptString(string plainText)
{
    byte[] bytes = Encoding.UTF8.GetBytes("$1*#rt5^&ds12fp=");
    byte[] bytes2 = Encoding.UTF8.GetBytes(plainText);
    PasswordDeriveBytes passwordDeriveBytes = new PasswordDeriveBytes("2@$#%jyth98@4q", null);
    byte[] bytes3 = passwordDeriveBytes.GetBytes(32);
    ICryptoTransform transform = new RijndaelManaged
    {
        Mode = CipherMode.CBC
    }.CreateEncryptor(bytes3, bytes);
    MemoryStream memoryStream = new MemoryStream();
    CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Write);
    cryptoStream.Write(bytes2, 0, bytes2.Length);
    cryptoStream.FlushFinalBlock();
    byte[] inArray = memoryStream.ToArray();
    memoryStream.Close();
    cryptoStream.Close();
    return Convert.ToBase64String(inArray);
}
```

Figure08. Encryption method present in binaries

The iv and password used are shared among the samples observed in these intrusions, allowing an effective detection of these DanBot samples and further linking of the intrusions together. This code block is implemented, with a slightly different name, in all of the binaries.

The preferred method of persistence, and execution of the malware, appears to be as a scheduled task, made to run on intervals when the user is logged on. The DanDrop VBA macro contains code as shown in Figure09. This code, with various minor modifications, is used in the DanDrop documents to schedule the execution of the DanBot malware.



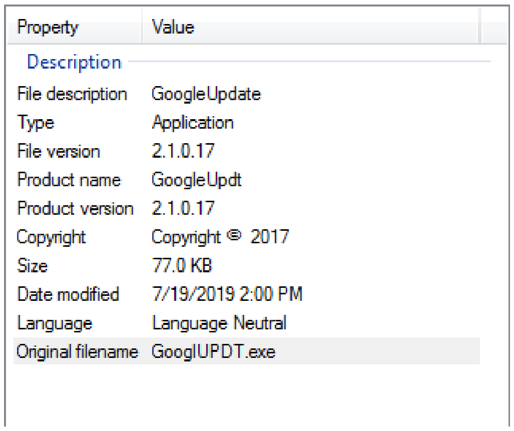
```
Dim Action
Set Action = taskDefinition.Actions.Create(0)
Action.path = "C:\Users\Public\PublicLibs\GooglUPDT.exe"

Const createOrUpdateTask = 6 'TASK_CREATE_OR_UPDATE
Call rootFolder.RegisterTaskDefinition("GoogleUpdateTS", taskDefinition, createOrUpdateTask, , , 3) 'TASK_LOGON_INTERACTIVE_TOKEN
```

Figure09. Register scheduled task persistence. (The code has been refactored and commented for readability)

Configuration data

Observed DanBot sample file names, version info, and other PE attributes may have been selected to make the binaries blend in on a target system. Noteworthy is the usage of shortened names for google and update.

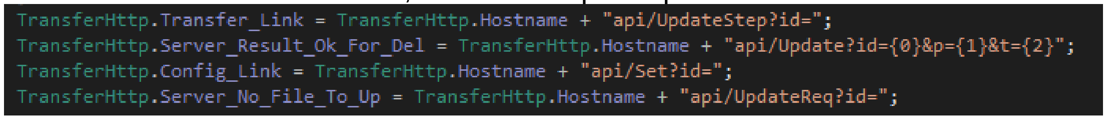


Property	Value
Description	
File description	GoogleUpdate
Type	Application
File version	2.1.0.17
Product name	GoogleUpdt
Product version	2.1.0.17
Copyright	Copyright © 2017
Size	77.0 KB
Date modified	7/19/2019 2:00 PM
Language	Language Neutral
Original filename	GooglUPDT.exe

Figure10. A sample posing as benign Google software.

The preferred installation path for the DanBot malware appears to be a folder under “C:\Users\Public\” where the malware has its working path. This path is constructed in the DanDrop documents.

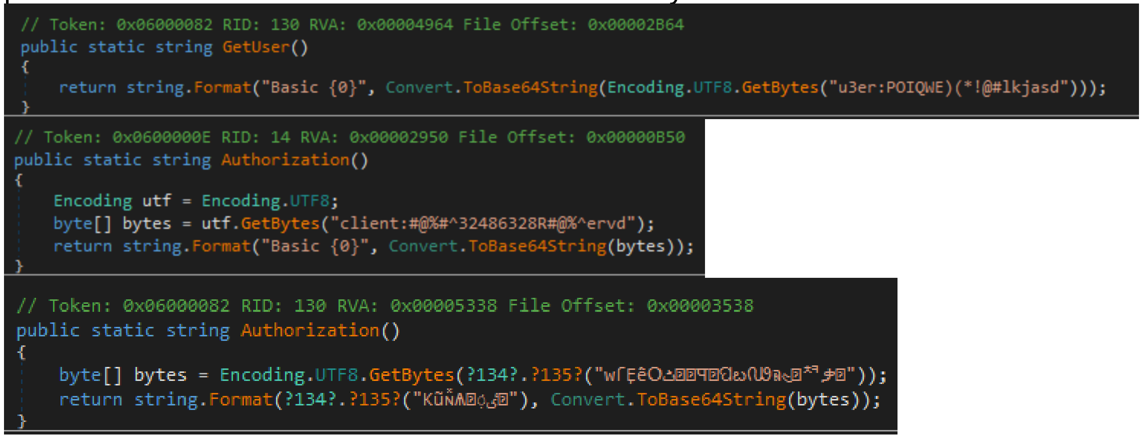
The DanBot samples, which contain an HTTP transport, share hard coded HTTP paths for interactions with the CC server, themed as update parameters.



```
TransferHttp.Transfer_Link = TransferHttp.Hostname + "api/UpdateStep?id=";
TransferHttp.Server_Result_Ok_For_Del = TransferHttp.Hostname + "api/Update?id={0}&p={1}&t={2}";
TransferHttp.Config_Link = TransferHttp.Hostname + "api/Set?id=";
TransferHttp.Server_No_File_To_Up = TransferHttp.Hostname + "api/UpdateReq?id=";
```

Figure11. Hard coded HTTP paths in malware

The DanBot samples analyzed, which contain an HTTP transport, also contains variants of adding an Authorization HTTP header. Whether implemented as a security feature on the CC, or simply as an identifier for separate sets of intrusions, HEXANE’s use of keyboard walks as passwords in the Authorization header is noteworthy.



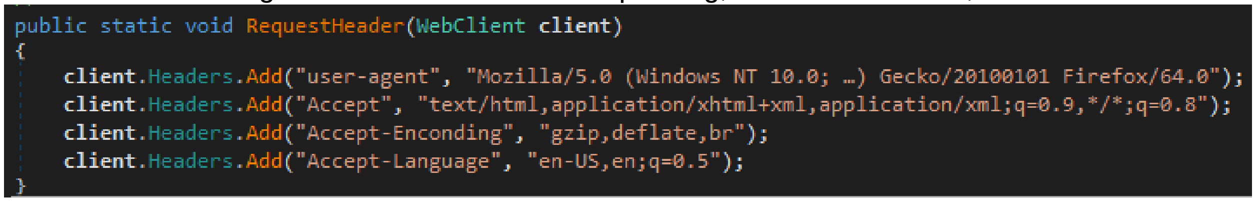
```
// Token: 0x06000082 RID: 130 RVA: 0x00004964 File Offset: 0x00002B64
public static string GetUser()
{
    return string.Format("Basic {0}", Convert.ToBase64String(Encoding.UTF8.GetBytes("u3er:POIQWE)(*!@#lkjasd")));
}
```

```
// Token: 0x0600000E RID: 14 RVA: 0x00002950 File Offset: 0x00000B50
public static string Authorization()
{
    Encoding utf = Encoding.UTF8;
    byte[] bytes = utf.GetBytes("client:#@%#^32486328R#@%^ervd");
    return string.Format("Basic {0}", Convert.ToBase64String(bytes));
}
```

```
// Token: 0x06000082 RID: 130 RVA: 0x00005338 File Offset: 0x00003538
public static string Authorization()
{
    byte[] bytes = Encoding.UTF8.GetBytes(?134?.?135?("wΓḚêO⚠▯▯ꟼ▯Ꮷᴗ᧐Ꮽ₰ᩮ▯*ᵃᵊ϶▯"));
    return string.Format(?134?.?135?("KũN̈ʌ▯ợ▯"), Convert.ToBase64String(bytes));
}
```

Figure12. Examples of constructing auth header, the obfuscated string value decodes to cl:QAZwsx!@#$%#!@QAZ

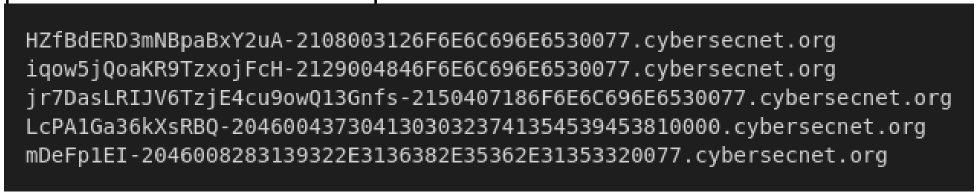
Other headers that stand out are “Accept-Enconding”, which would appear to be a typo, and lower case “user-agent” header, with an invalid user agent. Along with the rest of the headers in Figure13, this function can be found, with varying user agent, in multiple DanBot samples. These headers are great candidates for further pivoting, network detection, and data for the



```
public static void RequestHeader(WebClient client)
{
    client.Headers.Add("user-agent", "Mozilla/5.0 (Windows NT 10.0; …) Gecko/20100101 Firefox/64.0");
    client.Headers.Add("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
    client.Headers.Add("Accept-Enconding", "gzip,deflate,br");
    client.Headers.Add("Accept-Language", "en-US,en;q=0.5");
}
```

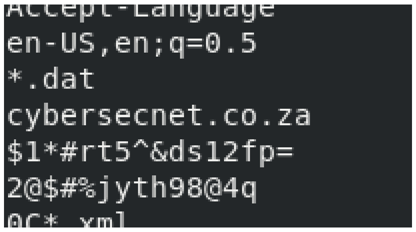
Figure13. Typo in HTTP headers

Some of the DanBot samples contains a second communications channel using a type of DNS tunneling, which is using the domain-string as a data channel. Repeated DNS queries with long subdomains allows DanBot to communicates with the CC server. Figure14 shows the DNS-requests generated, which results in plenty of high entropy subdomain dns-requests. Figure15 shows one of the DNS host names used by HEXANE in this campaign, it can be found in plaintext on the DanBot samples.



```
HZfBdERD3mNBpaBxY2uA-2108003126F6E6C696E6530077.cybersecnet.org
iqow5jQoaKR9TzxojFcH-2129004846F6E6C696E6530077.cybersecnet.org
jr7DasLRIJV6TzjE4cu9owQ13Gnfs-2150407186F6E6C696E6530077.cybersecnet.org
LcPA1Ga36kXsRBQ-204600437304130303237413545394538100000.cybersecnet.org
mDeFp1EI-2046008283139322E3136382E35362E31353320077.cybersecnet.org
```

Figure14. Examples of DNS requests by malware



```
Accept-Language
en-US,en;q=0.5
*.dat
cybersecnet.co.za
$1*#rt5^&ds12fp=
2@$#%jyth98@4q
@C*.xml
```

Figure15. DNS host name in DanBot sample

A cyber threat intelligence analyst’s job, depending on their intelligence requirements, will require them to analyze individual intrusions, cluster those intrusions confidently together, and make observations and assessments to satisfy their goal. The Diamond Model is one method to cluster intrusions together to satisfy such requirements. Identifying human fingerprints to make assessments about the adversary and place them confidently in the Adversary feature is one effective way of linking intrusions. Finding connections in intrusions with one feature, such as Adversary, is useful but be sure not to rely on a single feature or a single source of information. As an example, to cluster intrusions together to make the HEXANE activity group the analysts also considered the Victim, Infrastructure, and Capability features. While this paper explored the human fingerprints in malware you can find such human fingerprints across a wide variety of digital evidence.

As an example, the command and control domain hosting chosen by HEXANE for this set of intrusions are typically themed as general IT-services companies hosted on bought infrastructure at VPS hosting providers. Typically, domains will host their own authoritative name server in order to ensure crafted responses from DNS command and control beacons.

The domain hosting moves over time, frequently using the same provider.

Domain	**VPS Hosting**
cybersecnet[.]org; ascycle[.]space; Excsrvcdn[.]com; Online-analytic[.]com	OVH
Site-issues[.]com; Bsolutions-cloude[.]com	OVH
Web-traffic[.]info; Web-statistics[.]info; Jointence[.]net	OVH
munabulk[.]com; Normalbalk[.]com	Hostkey BV
dobokutpc[.]com; Googlmail[.]net; Gnaii[.]com; Security-patch[.]gr[.]jp; Mentalhealthpromote[.]jp	IDC Frontier Inc

Table03. VPS Hosting providers

The HEXANE victimology also appeared to focus heavily on achieving a goal of reaching industrial organizations such as oil and gas companies. All of these elements together can provide strong correlation of intrusions to create an activity group. The activity group’s motivations, goals, and their human fingerprints will all fit solidly under the Adversary feature and extracting these as cyber threat intelligence analysts can add significant confidence to the satisfying of intelligence requirements.

Last Updated: November 24th, 2019

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Bangalore 2019	**Bangalore, IN**	**Nov 25, 2019 - Nov 30, 2019**	**Live Event**
SANS Cyber Threat Summit 2019	**London, GB**	**Nov 25, 2019 - Nov 26, 2019**	**Live Event**
SANS Nashville 2019	**Nashville, TNUS**	**Dec 02, 2019 - Dec 07, 2019**	**Live Event**
SANS San Francisco Winter 2019	**San Francisco, CAUS**	**Dec 02, 2019 - Dec 07, 2019**	**Live Event**
SANS Security Operations London 2019	**London, GB**	**Dec 02, 2019 - Dec 07, 2019**	**Live Event**
SANS Paris December 2019	**Paris, FR**	**Dec 02, 2019 - Dec 07, 2019**	**Live Event**
SANS Frankfurt December 2019	**Frankfurt, DE**	**Dec 09, 2019 - Dec 14, 2019**	**Live Event**
SANS Cyber Defense Initiative 2019	**Washington, DCUS**	**Dec 10, 2019 - Dec 17, 2019**	**Live Event**
SANS Austin Winter 2020	**Austin, TXUS**	**Jan 06, 2020 - Jan 11, 2020**	**Live Event**
SANS Threat Hunting & IR Europe Summit & Training 2020	**London, GB**	**Jan 13, 2020 - Jan 19, 2020**	**Live Event**
SANS Miami 2020	**Miami, FLUS**	**Jan 13, 2020 - Jan 18, 2020**	**Live Event**
Cyber Threat Intelligence Summit & Training 2020	**Arlington, VAUS**	**Jan 20, 2020 - Jan 27, 2020**	**Live Event**
SANS Tokyo January 2020	**Tokyo, JP**	**Jan 20, 2020 - Jan 25, 2020**	**Live Event**
SANS Amsterdam January 2020	**Amsterdam, NL**	**Jan 20, 2020 - Jan 25, 2020**	**Live Event**
SANS Anaheim 2020	**Anaheim, CAUS**	**Jan 20, 2020 - Jan 25, 2020**	**Live Event**
MGT521 Beta Two 2020	**San Diego, CAUS**	**Jan 22, 2020 - Jan 23, 2020**	**Live Event**
SANS Las Vegas 2020	**Las Vegas, NVUS**	**Jan 27, 2020 - Feb 01, 2020**	**Live Event**
SANS Vienna January 2020	**Vienna, AT**	**Jan 27, 2020 - Feb 01, 2020**	**Live Event**
SANS San Francisco East Bay 2020	**Emeryville, CAUS**	**Jan 27, 2020 - Feb 01, 2020**	**Live Event**
SANS Security East 2020	**New Orleans, LAUS**	**Feb 01, 2020 - Feb 08, 2020**	**Live Event**
SANS London February 2020	**London, GB**	**Feb 10, 2020 - Feb 15, 2020**	**Live Event**
SANS Northern VA - Fairfax 2020	**Fairfax, VAUS**	**Feb 10, 2020 - Feb 15, 2020**	**Live Event**
SANS New York City Winter 2020	**New York City, NYUS**	**Feb 10, 2020 - Feb 15, 2020**	**Live Event**
SANS Dubai February 2020	**Dubai, AE**	**Feb 15, 2020 - Feb 20, 2020**	**Live Event**
SANS Brussels February 2020	**Brussels, BE**	**Feb 17, 2020 - Feb 22, 2020**	**Live Event**
SANS Scottsdale 2020	**Scottsdale, AZUS**	**Feb 17, 2020 - Feb 22, 2020**	**Live Event**
SANS San Diego 2020	**San Diego, CAUS**	**Feb 17, 2020 - Feb 22, 2020**	**Live Event**
Open-Source Intelligence Summit & Training 2020	**Alexandria, VAUS**	**Feb 18, 2020 - Feb 24, 2020**	**Live Event**
SANS Training at RSA Conference 2020	**San Francisco, CAUS**	**Feb 23, 2020 - Feb 24, 2020**	**Live Event**
SANS Tokyo November 2019	**OnlineJP**	**Nov 25, 2019 - Nov 30, 2019**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**