BIG GAME HUNTING: THE PECULIARITIES IN NATION-STATE MALWARE RESEARCH

Morgan Marquis-Boire
Citizen Lab, University of Toronto

Marion Marschalek
Cyphort Inc.

Claudio Guarnieri
Cuckoo Sandbox Foundation

INTRODUCTION

Over the past few years state-sponsored hacking has received attention that would make a rockstar jealous. Discussion of malware has shifted in focus from ‘cyber crime’ to ‘cyber weapons’, there have been intense public debates on attribution of various high profile attacks, and heated policy discussion surrounding regulation of offensive tools. We’ve also seen the sale of ‘lawful intercept’ malware become a global trade.

While a substantial focus has revolved around the activities of China, Russia, and Iran, recent discoveries have revealed the capabilities of Western nations such as WARRIORPRIDE aka. Regin (FVEY) and SNOWGLOBE aka. Babar (France). Many have argued that digital operations are a logical, even desirable part of modern statecraft. The step from digital espionage to political persecution is, however, a small one. Commercially written, offensive software from companies like FinFisher and Hacking Team has been sold to repressive regimes under the guise of ‘governmental intrusion’ software.

Nation state hacking operations are frequently well-funded, difficult to attribute, and rarely prosecuted even if substantive evidence can be discovered. While efforts have been made to counter this problem, proof is hard to find and even more difficult to correctly interpret. This creates a perfect storm of conditions for lies, vendor lies, and flimsy attribution. We will present a novel approach to binary stylometry, which helps matching binaries of equal authorship and allows credible linking of binaries into the bigger picture of an attack.

Case Study on FinFisher / Hacking Team / VUPEN

Our publications on the FinFisher and RCS commercial spyware opened the field to investigate the systematic use of hacking by Western law enforcement as well as regimes around the world.

Despite numerous reports with reproducible scientific results, spyware vendors like FinFisher and HackingTeam have been consistently denied these discoveries and the weight of the uncovered abuses. Leaked internal documents, however, showed the uncontestable nature of the attribution. Binary attributes, certificates and network behavior have provided systematic ways to monitor the use of such products for an extended period of time.

Case Study on Regin malware

Regin has been one of the most peculiar cases among the trove of nation-state attacks that have come to light in the last few years. With malware samples spanning over a timeframe of more than 10 years, Regin is one of the largest and most sophisticated malware frameworks discovered to date.

Technical analysis by the authors tied with corroborating evidence provided by leaked documents made Regin (aka WARRIORPRIDE) to be one of the first malware kits to be conclusively attributed to a Western nation-state actor[5]. Consequently, it also represented one of the first instances where security vendors had to face their political biases and publicly justify several years of silence on hacking by governments which are their customers.

Case Study on Animal Farm

The recently uncovered Animal Farm malware and its associated families serve as a useful example for our proposed method, as the binaries are straight-forward and easily understood. They come with many of the described features. Additionally, a fairly large set of binaries have been identified, estimated to be developed over a span of several years and by different authors, while showing very different functionality.

The different families of Animal Farm are NBOT, TFC, Bunny, Babar, Casper and Dino; respectively DDoS malware, reconnaissance malware, espionage tools and scripting bots. Their intentions are very different, yet the list of shared features applied through the introduced catalog is long.

By extracting the before mentioned data for the different families we can prove without doubt that NBOT, TFC, Bunny, Babar, Casper and Dino originate from the same group of authors.

PROBLEM STATEMENT

In malware research, especially when dealing with APT and nation state campaigns, we face the challenge of putting the malware at hand into context. Vendors and nation-states are obliged to deny attribution, even if correct. Furthermore, a binary itself does not give away who wrote it, who controlled it, who the infected victims were or what the aim of the operation that involved it was. A standalone binary does not even tell which operation involved it.

We cannot conclude from a binary to its context. What we can do though, is posit from a binary to a related binary, which in most cases helps a great deal in the investigation. Bringing a malicious binary in context with known APT malware often allows credible conclusions about the nature of an attack or in a few limited cases even allows linking a binary with an alleged operator.

In malware research conclusions are very often based on non-scientific research. We present the reader with an approach that helps building credible links between a binary and a given set of binaries from the same author in a measurable way.

Related Research

A number of research projects focus on authorship attribution of code, commonly named code stylometry. The aim of these projects is to identify the origin of a piece of source code or a binary within a set of repositories by known authors.

Code stylometry most certainly does not work on binaries as compilers and disassemblers (or decompilers) wash away most attributes mentioned in recent publications. Caliskan [1] describes attributes, for example, the average number of characters per word, character count, use of special characters, punctuation and percentage of digits as feasible for source code attribution. These cannot be applied to disassembled or decompiled binaries.

Burrows, Uitdenbogerd and Turpin [3] base their authorship attribution research on a feature set including white spaces, operators, literals, keywords, I/O words and function words from

standard C libraries. Again, considering the loss of information through disassembly or decompilation this approach will most certainly fail.

Research published by Rosenblum, Zhu and Miller [4] describes methodologies to automatically detect stylistic features of binary code. Their approach derives features from a control flow graph representation, the instruction sequence of a binary to build a classifier based on support vector machines, as well as the clustering of program authors based on the k-means algorithm. The set of derived features are n-grams and idioms from an instruction level; graphlets, supergraphlets and call graphlets from the control flow graph as well as library calls.

While this is probably the closest to putting two binaries in a context, the approach comes with weaknesses. Regarding APT- or nation state malware, one or more families of malicious binaries do not stem from one author but most likely from a team of developers with switching team members, who have been working on the malware over a long period of time. Additionally, the described technique does not consider standard library code within a binary, deal with obfuscated binaries or show resistance against attribution evasion or false flag techniques. At the same time numerous aspects of malware are not considered for classification, especially with regard to APT and nation state malware.

AN APPROACH TO BINARY STYLOMETRY

We present a novel approach on creating credible links between binaries originating from the same group of authors. The technique is based on features derived from different domains, such as implementation details, applied evasion techniques, classical malware traits or infrastructure attributes. Such features could for example be shared memory allocation habits, obfuscation or custom encryption algorithms, reused anti-simulation tricks, shared system infiltration- or persistence techniques or shared command and control servers.

Deriving features from various domains helps avoid sole reliance on the binary representation of a sample. On the instruction level a program can be altered with something as simple as a change in compiler settings. Also, a broader set of attributes counters deliberate authorship confusion techniques. It is assumed that attributable features are possibly faked, but we rely on the fact that it is difficult to fake all features on all levels. The question finally is, how many features can we gather from compiled binaries and how many do we need to create a credible link?

The following catalog of attributes has proven to be feasible in identifying related binaries. It is important to evaluate the information gained from the dedicated features in order to draw correct conclusions. Primary interest lies in features that are more likely to be introduced by the program author than from outside, from the compiler or linker for example.

String constants

- Error / status messages
- String formatting style
- English grammar mistakes
- C&C commands
- Timestamp formatting

Implementation traits

- Memory allocation habits
- Use of global variables
- Multi-threading model
- Software architecture and design
- Constructor design
- Dynamic API loading technique
- Exception handling
- Usage of public source code
- Programming language and compiler
- Compilation time stamps and time zones

Custom features

- Obfuscation techniques
- Stealth and evasion techniques
- Use of encryption and compression algorithms
- (Shared) encryption keys
- Re-used source code in general
- Malware specific features
- System infiltration
- Propagation mechanisms
- Artifact naming schemes / algorithms
- Data exfiltration techniques
- System / OS version determination technique
- C&C command parsing implementation
- Malware configuration

Infrastructure

- C&C servers
- Countries / languages used for domain hosting and naming
- User agent / beaconing style

- Communication protocol / port
- Communication intervals

PROOF OF CONCEPT: SNOWGLOBE

The proposed catalogue applied to a selection of SNOWGLOBE samples outlines a relation among the families and sums up the most identifying features. Due to size restrictions not all features are shown in the summary. The colors help in easy identification of links; white indicates no relation, light yellow a weak link and dark yellow that a strong link was found. Further more, non-overlapping attributes help understand the differences among related sample and even show a summary of individual traits. These help understand interests and development capabilities of a given threat actor.

	A	B	C	D	E
1			**NBOT/TFC**	**Bunny**	**Babar**
2	**String constants**				
3		Error / status messages	No	Many	Many
4		String formatting style	All plain, commands/config all caps, no special charact	Partially plain, config encrypted, config all caps in XML f	All plain, config all caps, enclosed in '%' characters
5		English grammar mistakes	No	Many	Many
6		C&C commands	PING,EXEC,HTTPF,ASPFLOOD,TCPFLOOD,WEBFLOOD,POSTFL	mainfrequency,getconfig,ftpput,ftpget,sendfile,getfile,u	N/A
7		Timestamp formatting	Time APIs _time64, _mktime64; '%02d:%02d:%02d', Time	Time API GetSystemTime(), 'timestamp %04d-%02d-%02d	N/A
8	**Implementation traits**				
9		Memory allocation habits	direct calls to _malloc/_free, no wrappers	GetProcessHeap()/HeapAlloc()/HeapFree() in large num	direct calls to _malloc/_free, no wrappers
10		Use of global variables	Few	Few, storing of event handles, strings, global flags usec	Few, storing of event handles, strings
11		Multi-threading model	Simple, main thread with several worker threads	Simple, main thread with several worker threads	Complex, multi-threading in various instances coordina
12		Software architecture and design	Standalone executable, classical bot structure	Standalone executable, classical bot structure, integrat	DLL, designed to run in context of arbitrary process, mai
13		Constructor design	MSVC++ default	MSVC++ default	MSVC++ default with complex object dependencies
14		Dynamic API loading technique	Present, subset of APIs only, per API, API name identifie	Present, subset of APIs only, per API, API name identifie	Present, subset of APIs only, per API, API name identifie
15		Exception handling	C++ EH and unhandled exception filter: ExitThread()	C++ EH and unhandled exception filter: ExitThread() (dyr	C++ EH default
16		Usage of public source code	None (known)	Lua engine, C/Invoke bindings	Keylogger from codeproject.com, OpencoreAMR library, M
17		Programming language and compiler	C++ / MSVC++ 8.0	C++ / MSVC++ 8.0	C++ / MSVC++ 8.0
18		Compilation time stamps and time zones	2010:03:11 17:55:03+01:00 2010:02:16 18:05:54+01:00 2010:05:06 15:47:37+02:00	 2011:10:25 21:28:39+02:00 2011:10:25 21:28:00+02:00	2011:08:29 15:02:29+02:00 2011:08:29 13:48:42+02:00 2011:07:06 15:50:11+02:00
19	**Custom features**				
20		Obfuscation techniques	Obfuscation of subset of API names that are to be loade	Obfuscation of subset of API names that are to be loade	Obfuscation of subset of API names that are to be loade
21		Stealth and evasion techniques	Obfuscation of subset of APIs	Emulator check,Containing directory name check,Payloa	Obfuscation of subset of APIs,Infection ‚strategy' based
22		Use of encryption and compression algorithms	API name obfuscation custom algorithm	API name obfuscation custom algorithm	API name obfuscation custom algorithm, adaption of SH
23		(Shared) encryption keys	XOR key AB34CD77h	XOR key AB34CD77h, keys for command/data en-/decrypt	128bit AES, 24 FE C5 AD 34 56 F7 F8 12 01 00 AE B6 7C DE A
24		Re-used source code in general	Timestamp generation, API name hashing and loading,	API name hashing and loading, infection strategy and A	infection strategy and AV product enumeration through
25		Malware specific features	DDoS bot for flooding of network packets	Lua scripted bot for automation of tasks	Espionage malware and userland rootkit
26		System infiltration	Designed to be loaded as service, running in context of	Loaded by a registry key on startup, running in context o	Loaded through registry key which invokes regsvr32.exe
27		Propagation mechanisms	N/A	N/A	N/A
28		Artifact naming schemes / algorithms	Config value naming prefix NBOT_	Internal name bunny 2.3.2, payload name netmgr.exe, re	Internal name Babar64, payload dump21cb.dll, directory
29		Data exfiltration techniques	Statistics file sent to C&C on demand	Log-/dumpfile regularly pushed to C&C	Dumpfiles regularly pushed to C&C (assumption)
30		System / OS version determination technique	N/A	N/A	N/A
31		C&C command parsing implementation	Encrypted command received from C&C, handed over as	Encrypted command received from C&C, command parsir	N/A
32		Malware configuration	hardcoded / plaintext	hardcoded / encrypted	hardcoded / encrypted
33	**Infrastructure**				
34		C&C servers	http://callientefever.info/, http://fullapple.net/	http://le-progres.net/, http://ghatreh.com/, http://usthb	http://www.horizons-tourisme.com/, http://www.gezelir
35		Countries / languages used for domain hosting	US/English	US/French, US/Iranian, US/Algerian	US/Algerian, US/Turkish
36		User agent / beaconing style	User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows	User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Win32)	User-Agent: Mozilla/4.0 (compatible; MSI 6.0; Windows I
37		Communication protocol / port	HTTP/80	HTTP/80	HTTP/80
38		Communication intervals	On demand	Regular, interval configurable	Regular (assumption)

In order to maintain the practical use of this approach the derived features need to be as normalized as possible. The preferred data format is JSON, due to its flexibility in adding and extending nodes. The proposed format is as follows:

```
StringConstants:
	StatusMessages:
		Present (BOOL) | Messages (ARRAY)
	StringFormatting (TEXT)
	Typos:
		Present (BOOL) | Mistakes (ARRAY)
	CnCCommands (ARRAY)
	Timestamp:
		Formatting (TEXT) | ApiList (ARRAY)
Implementation:
	MemoryAllocation:
		ApiList (ARRAY) | Wrappers (BOOL) | SpecialObservations (TEXT)
	GlobalVariables:
		Usage [No/Few/Many] (TEXT) | SpecialObservations (TEXT)
	ThreadingModel:
		Complexity [simple/complex] (BOOL) | Description (TEXT) |
		CoordinationMechanisms (ARRAY)
	SoftwareArchitecture:
		BinaryType [exe/dll/sys] (TEXT) | Standalone (BOOL) |
		ArchitectureDescription (TEXT)
	Constructors:
		Default (BOOL) | SpecialObservations (TEXT)
	DynamicApiLoading:
		Present (BOOL) | Description (TEXT)
	ExceptionHandlingObservations (TEXT)
	PublicSourceCode:
		Present (BOOL) | SourcesList (ARRAY)
	LanguageCompiler (TEXT)
	CompilationTime (ARRAY)
CustomFeatures:
	ObfuscationTechniques (ARRAY)
	EvasionTechniques (ARRAY)
	UsedAlgorithms (ARRAY)
	UsedKeys (ARRAY)
	ReusedFeatures (ARRAY)
	MalwareFeatures (ARRAY)
	PersistenceTechniques (ARRAY)
	InfiltrationTechniques (TEXT)
	PropagationTechniques (ARRAY)
	Artifacts:
		ArtifactNames (ARRAY) | SpecialObservations (TEXT)
	DataExfiltration:
		Present (BOOL) | Description (TEXT)
	OsRecognition:
```

```
            APIList (ARRAY) | Description (TEXT)
      CommandParsing:
            Present (BOOL) | Description (TEXT)
      Configuration:
            Present (BOOL) | Hardcoded (BOOL) | SpecialObservations (BOOL)
Infrastructure:
      CnCServers:
            Domains (ARRAY) | SpecialObservations (TEXT)
      GeoData (ARRAY)
      Beaconing (TEXT)
      ProtocolPort (ARRAY)
      CommunicationIntervals (TEXT)
```

Feature extraction and comparison is an overall tedious process and for most features has to be performed by an analyst by hand. Automation to a small extent is possible, for example in extracting API lists or detecting timestamp formatting strings. Most attributes though require deep understanding of the malware and cannot be accomplished by machine logic. Once the data is complete and abstracted it describes the most important features of a malware sample, needed to compare two or more samples or even malware families.

GAINED VALUE

The presented approach enables analysts to credibly link related binaries and prove or deny a relation among families of binaries. As experienced by real-world case studies, by being able to prove the different families stem from the same group of authors, one can conclude the various interests of the malware operator by considering the capabilities of the malware. Also it is interesting to derive a higher level of information, such as increasing coding capabilities of the authors or added code features over time. Furthermore, the technique can help proving that the same malware is being used in different operations, thus show that one actor has expanded interests or the same malware strain is being used by different actors.

On the other hand, in reality it is often even more useful to prove the opposite, namely that two malware strains do not carry similar handwriting.

RESTRICTIONS

This research does not focus on authorship attribution, which in itself is an extraordinarily tough objective when basing conclusions solely on binaries. Our purpose is only to build credible links among binaries from one author or a group of authors.

We do not plan to apply machine learning as the amount of known nation state malware binaries is considerably small, thus the data set very limited. Hands-on work is involved in deriving features, automation in the comparison step can be achieved by maintaining an appropriate threat data base which allows basic data analytics.

The presented technique shall not be understood as binary similarity research, as we aim to identify binaries from the same authors rather than to identify binaries with similar functionality.

SOURCES AND SUPPORTING FILES

[1]
Source Code Authorship Attribution -
http://www.ieee-security.org/TC/SP2013/posters/Aylin_Caliskan_Islam.pdf

[2]
Supervised Source Code Authorship Attribution -
https://www.cs.drexel.edu/~ac993/papers/Aylin_Oakland_2013_poster.pdf

[3]
Application of Information Retrieval Techniques for Source Code Authorship Attribution -
http://www.researchgate.net/profile/Alexandra_Uitdenbogerd/publication/220787332_Application_of_Information_Retrieval_Techniques_for_Source_Code_Authorship_Attribution/links/0912f51181dd371820000000.pdf

[4]
Who Wrote This Code? Identifying the Authors of Program Binaries -
ftp://ftp.cs.wisc.edu/paradyn/papers/Rosenblum11Authorship.pdf

[5]
Secret Malware In European Union Attack Linked To U.S. And British Intelligence -
https://firstlook.org/theintercept/2014/11/24/secret-regin-malware-belgacom-nsa-gchq/

[6]
Meet Babar: A New Malware Almost Certainly Created By France
http://motherboard.vice.com/read/meet-babar-a-new-malware-almost-certainly-created-by-france

[7]
Babar: Suspected Nation State Spyware In The Spotlight -
http://www.cyphort.com/babar-suspected-nation-state-spyware-spotlight/

[8]
For Their Eyes Only: The Commercialization of Digital Spying -
https://citizenlab.org/storage/finfisher/final/fortheireyesonly.pdf

[9]
Secret Malware in European Union Attack Linked to U.S. and British Intelligence -
https://firstlook.org/theintercept/2014/11/24/secret-regin-malware-belgacom-nsa-gchq/

[10]
When Governments Hack Opponents: A Look at Actors and Technology -
https://www.usenix.org/conference/usenixsecurity14/speaker-or-organizer/morgan-marquis-boire-citizen-lab

ABOUT THE AUTHORS

Morgan Marquis-Boire
Citizen Lab, University of Toronto

Morgan Marquis-Boire is a Senior Researcher at the Citizen Lab, University of Toronto. He is the Director of Security for First Look Media and a contributing writer for The Intercept. Prior to this, he worked on the security team at Google. He is a Special Advisor to the Electronic Frontier Foundation in San Francisco and an Advisor to the United Nations Inter-regional Crime and Justice Research Institute. In addition to this, he serves as a member of the Freedom of the Press Foundation advisory board and as an advisor to Amnesty International.

Marion Marschalek
Cyphort, Inc.

Marion Marschalek is a Senior Malware Researcher on duty for Cyphort, Inc., focusing on the analysis of emerging threats and exploring novel methods of threat detection. She teaches malware analysis at University of Applied Sciences St. Pölten and frequently appears as speaker at international conferences. Two years ago Marion won Halvar Flake's reverse engineering challenge for females, since then she set out to threaten cyber criminals. She practices martial arts and has a vivid passion for taking things apart. Preferably, other people's things.

Claudio Guarnieri

Claudio Guarnieri is the creator and lead developer of Cuckoo Sandbox, a prominent open source automated malware analysis system. Claudio also runs the Malwr.com website and spends most of his time finding and analyzing malware. In recent years, he has devoted his attention to issues of privacy and surveillance and published numerous investigative reports on surveillance vendors with the Citizen Lab.