



Behavioral Matrix Deep Dive into an Energetic Bear Tool

Kyle O'Meara

SEI CERT Coordination Center

March 2019

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Copyright 2019 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.


NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon®, CERT® and CERT Coordination Center® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM19-0240



The entire contents of this
presentation are UNCLASSIFIED

Agenda



- **whoami**
- **Motivation**
- **Behavioral Matrix**
- **Analysis Energetic Bear API Hashing Tool**

whoami

- Software Engineering Institute (SEI) CERT Coordination Center (CERT/CC)
 - Threat Analysis Directive -> Malware Analysis Team -> Reverse Engineering (RE) Team
- Adjunct Faculty & Faculty Advisor at Carnegie Mellon University and Adjunct Faculty at Duquesne University
 - Teach undergrad and grad cyber security courses
- Develop and run the BSidesPGH Capture the Flag (CTF)
- Past work includes:
 - FireEye - Sr. Security Consultant dedicated to the SharkSeer Program
 - Private Consulting Firm - Digital Forensics and Incident Response
 - National Security Agency (NSA) - Various roles as a Technical Analyst
- Larger speaking engagements: Blackhat USA & Europe Arsenal, DEF CON, ShmooCon, NatCSIRT, BSidesPGH, FIRST TC, Senate Cyber Threat Intelligence Summit
- Publications can be found on the SEI Digital Library and CERT/CC GitHub

RE Team

The CERT/CC Reverse Engineering (RE) Team is committed being a world leader in advancing the art of malicious code analysis.

These include:

- Studying and analyzing the long or short term evolution of malware families
- Binary RE: Analysis and when appropriate YARA rule generation and/or the extraction of configuration data to include C2s, domains, username & password combinations, unique keys
- RE Tools: Create and maintain a suite of utilities to assist in and speed up the reverse engineering process
- RE Experience Beyond x86/x64: Experience in RE of other architectures and platforms such ARM, MIPS, PowerPC, and Android
- Hardware RE: firmware extraction and analysis

Motivation

- Improve our corpus of knowledge on the RE Team
- Identify gap areas of specific malware tools used by adversaries.
- How to I do that?
 - Create a matrix that maps to internal and open source
 - Pick a piece of malware that has gaps in the matrix
 - Fill in the those gaps

Behavioral Matrix

- Fields
 - Malware Common Names
 - CERT/CC APIAnalyzer Behavior (if appropriate)
 - MITRE ATT&CK Techniques
 - CERT/CC Analysis
 - CERT/CC YARA rules
 - CERT/CC Config Dumper
 - OSINT YARA rules
- This matrix will not only highlight internal gaps but also gaps that exist in the open source community

CERT/CC APIAnalyzer

- ApiAnalyzer is a tool for finding sequences of API calls with the specified data and control relationships.
 - This capability is intended to be used to detect common operating system interaction paradigms like opening a file, writing to it, and the closing it.
- Part of Pharos static binary analysis framework
 - <https://github.com/cmu-sei/pharos>

APT28 Behavioral Matrix (for context)

Malware Common Names	CERT APIAnalyzer Behavior	MITRE ATT&CK Techniques	CERT Analyzed	CERT Known YARA Rules	CERT Config Dumper	OSINT YARA Rules
CORESHELL/SOURFACE/Sofacy	Informational: TerminateSelf	Binary Padding, Custom Cryptographic Protocol, Data Encoding, Obfuscated Files or Information, Registry Run Keys / Startup Folder, Remote File Copy, Rundll32, Standard Application Layer Protocol, System Information Discovery	No	Yes	No	Yes
	Process Manipulation: SpawnProcess					
GAMEFISH/JHUHUGIT/Seduploader/JKEYSKW/Sednit/SofacyCarberp	Informational: TaskList, RaidClipboard, TerminateSelf	Clipboard Data, Component Object Model Hijacking, Data Encoding, Exploitation for Privilege Escalation, Fallback Channels, File Deletion, Logon Scripts, New Service, Obfuscated Files or Information, Process Discovery, Process Injection, Registry Run Keys / Startup Folder, Remote File Copy, Rundll32, Scheduled Task, Screen Capture, Standard Application Layer Protocol, System Information Discovery, System Network Configuration Discovery	Yes	Yes	Yes	Yes
	Networking: RecvFileViaWininet					
X-Agent/CHOPSTICK/SPLM	Informational: FileSearchV2, TerminateSelf	Command-Line Interface, Communication Through Removable Media, Connection Proxy, Fallback Channels, File and Directory Discovery, Input Capture, Modify Registry, Query Registry, Remote File Copy, Replication Through Removable Media, Screen Capture, Security Software Discovery, Standard Application Layer Protocol, Standard Cryptographic Protocol	Yes	No	No	Yes
	Malware: ReverseShellV2					
X-Tunnel/XAPS	Networking: RecvFileViaWininet	Binary Padding, Command-Line Interface, Connection Proxy, Credentials in Files, Fallback Channels, Network Service Scanning, Obfuscated Files or Information, Remote File Copy, Standard Cryptographic Protocol	No	Yes	No	Yes
	Process Manipulation: SpawnProcessV2					
AZZY/EVILTOSS/ADVSTORESHELL/NETUI/Sedreco	Informational: TerminateSelf	Command-Line Interface, Commonly Used Port, Component Object Model Hijacking, Data Compressed, Data Encoding, Data Encrypted, Data Staged, Execution through API, Exfiltration Over Command and Control Channel, File and Directory Discovery, File Deletion, Input Capture, Modify Registry, Obfuscated Files or Information, Peripheral Device Discovery, Process Discovery, Query Registry, Registry Run Keys / Startup Folder, Rundll32, Scheduled Transfer, Standard Application Layer Protocol, Standard Cryptographic Protocol, System Information Discovery	No	No	No	Yes
	Process Manipulation: SpawnProcess, SpawnProcessV2					
OLDBAIT	Malware: ReverseShellV2	Credential Dumping, Masquerading, Obfuscated Files or Information, Standard Application Layer Protocol	No	No	No	No
	Networking: RecvFileViaWininet					
USBStealer	Informational: TerminateSelf, EnumerateDrivesV2, EnumerateDriveInfoV2, FileSearchV2	Automated Collection, Automated Exfiltration, Communication Through Removable Media, Data from Removable Media, Data Staged, Exfiltration Over Physical Medium, File and Directory Discovery, File Deletion, Masquerading, Obfuscated Files or Information, Peripheral Device Discovery, Registry Run Keys / Startup Folder, Replication Through Removable Media, Timestamp	No	No	No	No
	Process Manipulation: DeleteServiceWide, DeleteService					
Zebrocy	Informational: EnumerateDrives, LoadProgramResource	Custom Command and Control Protocol, Remote File Copy, Standard Application Layer Protocol, System Information Discovery	No	No	No	Yes
	Process Manipulation: SpawnProcess					

Energetic Bear Matrix

Malware Common Names	CERT APIAnalyzer Behavior	MITRE ATT&CK Techniques (10)	CERT Analyzed	CERT Known YARA Rules	CERT Config Dumper	OSINT YARA Rules
Havex/Oldrea	Informational: TerminateSelf, FileSearchV2	Credential Dumping, Data Encrypted, Data Obfuscation, Email Collection, File and Directory Discovery, File Deletion, Process Discovery, Process Injection, Registry Run Keys / Startup Folder, System Information Discovery, System Network Configuration Discovery, System Owner/User Discovery	Yes	Yes	Yes	Yes
Sysmain	Informational: EnumeratesDrivesV2, FileSearchV2, TerminateSelf Process Manipulation: SpawnProcess, SpawnProcessV2	None	Yes	Yes	Yes	
API Hashing Tool	None	None	Yes	Yes	Working	Yes
Karagany	Informational: TerminateSelf	Credential Dumping, Data Staged, Process Discovery, Registry Run Keys / Startup Folder, Remote File Copy, Screen Capture, Software Packing	No	No	No	Yes

Energetic Bear API Hashing Tool

- In the fall of 2018, the CERT/CC RE Team received a tip from a trusted source about a YARA rule that alerted in VirusTotal that the trusted source had been monitoring.
- YARA rule came from Department of Homeland Security (DHS) Alert TA17-293A [i]
 - This document is associated with Russian activity.
- I believe that this was enough information to warrant further analysis.

Energetic Bear API Hashing Tool

- This YARA rule is allegedly associated with the Energetic Bear group
- Energetic Bear, named by CrowdStrike, conducts global intelligence operations primarily against the energy sector.
- They have been in operation since 2012 [ii].
- This intrusion set is also named Dragonfly (Symantec), Crouching Yeti (Kaspersky), Group 24 (Cisco), and Iron Liberty (SecureWorks), among others [iii].

```
rule APT_malware_2
{
  meta:
    description = "rule detects malware"
    author = "other"
  strings:
    $api_hash = { 8A 08 84 C9 74 0D 80 C9 60 01 CB C1 E3 01 03 45 10 EB ED }
    $http_push = "X-mode: push" nocase
    $http_pop = "X-mode: pop" nocase
  condition:
    any of them
}
```

Analysis Methodology

- Analyzed the YARA rule and initial exemplar (1b17ce735512f3104557afe3becacd05ac802b2af79dab5bb1a7ac8d10dccffd)
 - Analyzed Exemplar in IDA
 - Researched and Applied API Hashing Routine Findings to Exemplar
 - Mapped Research Findings and Analysis to Exemplar in IDA
- Created a tightly scoped YARA rule to discover new exemplars
 - Created API Hash YARA rule to Discover More Exemplars
 - Analyzed New exemplars for the presence of *api_hash_func_slladd1* and *manual_symbol_resolution*
 - Created a tightly scoped YARA rule
- Discovered API hashes found in new exemplars
- Questioned attribution
- Future Work
- Results

Analyzed the YARA rule and Initial Exemplar

Analyzed Exemplar in IDA

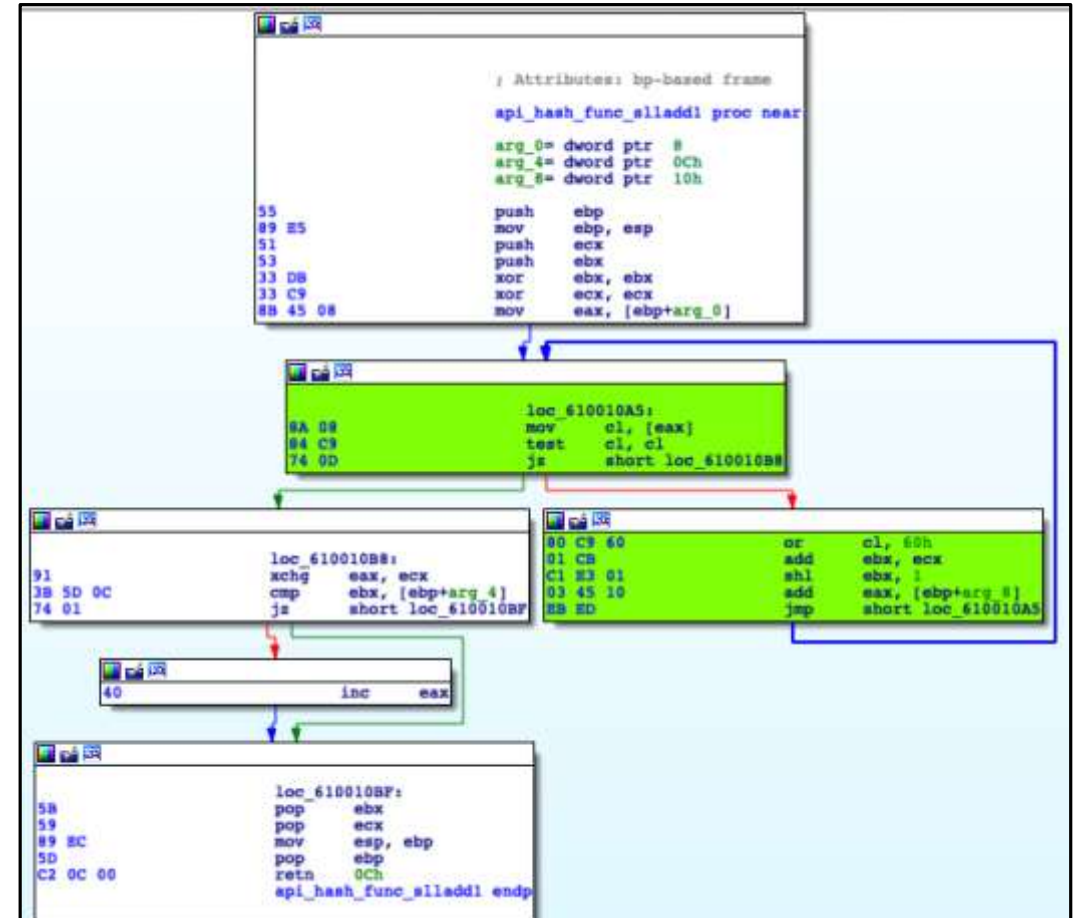
- Interested in understanding the string variables of the YARA rule
- Specifically, the *\$api_hash* variable was not immediately clear to what it represented
- Whereas, the variables *\$http_post* and *\$http_push* appeared to be associated with HTTP header fields.
- Focused my analysis on the *\$api_hash* variable

```
rule APT_malware_2
{
  meta:
    description = "rule detects malware"
    author = "other"
  strings:
    $api_hash = { 8A 08 84 C9 74 0D 80 C9 60 01 CB C1 E3 01 03 45 10 EB ED }
    $http_push = "X-mode: push" nocase
    $http_pop = "X-mode: pop" nocase
  condition:
    any of them
}
```


Analyzed the YARA rule and Initial Exemplar

Analyzed Exemplar in IDA

- After cursory analysis of the initial exemplar, determined that the *\$api_hash* variable was alerting on the routine, highlighted in green
- The key points to highlight are the *or* of 0x60, *shift logical left (shl)* by 1, followed by an *add*, and *jump*
- Determined that this was a Windows API hashing routine based on this information coupled with the *\$api_hash* variable name



Analyzed the YARA rule and Initial Exemplar

Researched and Applied API Hashing Routine Findings to Exemplar

- Used this information and OSINT gathering and found two items:
 - FireEye Flare IDA Plugins Github page that mentioned a plugin called *Shellcode Hashes* [iv]
 - FireEye blog post from 2012 titled “Using Precalculated String Hashes when Reverse Engineering Shellcode” which aided my understanding of API hashing [v]
- In the FireEye Flare IDA plugin, *Shellcode Hashes* script, there was a similar routine that was mentioned on the last slide
 - *for loop* which contains an *or* of 0x60, followed by *add*, and a *shift left by 1*.

```
def sll1AddHash32(inString,fName):  
    if inString is None:  
        return 0  
    val = 0  
    for i in inString:  
        b = ord(i)  
        b = 0xff & (b | 0x60)  
        val = val + b  
        val = val << 1  
        val = 0xffffffff & val  
    return val
```

Analyzed the YARA rule and Initial Exemplar

Researched and Applied API Hashing Routine Findings to Exemplar

- CERT/CC API hashing tool that creates a set of YARA signatures of API hashes for a given set of dynamic link library (DLL) files.
 - This API hashing tool also contains a very similar function that matched the routine mentioned from the exemplar and the FireEye IDA plugin
- Called this routine *sll1Add*
- Ran the CERT/CC API hashing tool on the exemplar for the specific *sll1Add* routine
 - Received an alert for kernel32.dll API hashes

Function	Byte Value (big endian)
LoadLibraryA	86 57 0D 00
VirtualAlloc	42 31 0E 00
VirtualProtect	3C D1 38 00

Analyzed the YARA rule and Initial Exemplar

Mapped Research Findings and Analysis to Exemplar in IDA

- CERT/CC UberFLIRT (“kind of like FLIRT but Uber”)
 - IDA’s Fast Library Identification and Recognition Technology (FLIRT) is meant to find known library functions, can make more signatures but have to manually apply collections, cumbersome to share new signatures, etc.
 - UberFLIRT calculates and stores PIC hashes of arbitrary functions, easily share information via a central database, less false positives, etc.
- Labeled the function shown in previous slides in IDA as *api_hash_func_slladd1* and saved it to the UberFLIRT database
- Examining the exemplar from the entry point:
 - Discovered 2 values that are pushed onto the stack and passed as parameters to a function
 - The 2 values are 0x0038D13C and 0x000D4E88.
 - The value 0x0038D13C is the hash of *VirtualProtect* which matches the values discovered by the CERT/CC API hashing tool on the previous slide
 - The other value, 0x000D4E88 is discussed next

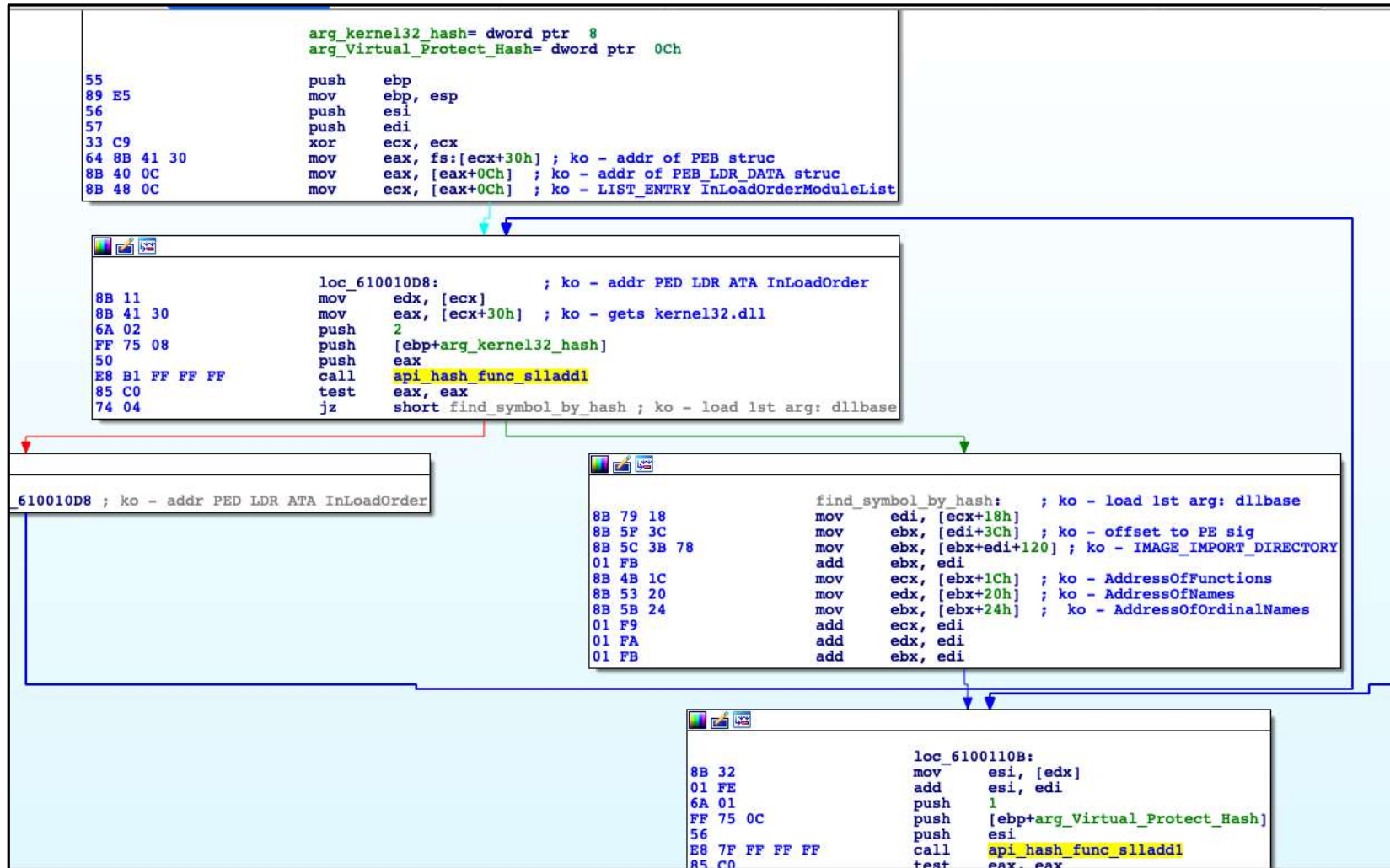
Analyzed the YARA rule and Initial Exemplar

Mapped Research Findings and Analysis to Exemplar in IDA

- Exemplar uses manual symbol loading techniques
 - Very similar to that of shellcode to find and load APIs
- This process reads the Thread Environment Block (TEB) to find the pointer to Process Environment Block (PEB) structure.
- The PEB structure is then parsed to find the DllBase of kernel32.dll.
- This tool also checks to ensure that it has the correct kernel32.dll by using 0x000D4E88 hash value, that is passed to the hashing algorithm, to check for the base name of the kernel32.dll to kernel32.dll name that was found via manual symbol loading.
- The function then continues to parse the portable executable (PE) Export data and then passes the Virtual Protect hash (0x0038D13C) to the hashing algorithm
- The same is done for the remaining hashes
- This process is shown on the next slide with my added comments.
 - Labeled the function *manual_symbol_resolution* and saved it to the UberFLIRT database

Analyzed the YARA rule and Initial Exemplar

Mapped Research Findings and Analysis to Exemplar in IDA



Created a Tightly Scoped YARA Rule to Discover New Exemplars

- Find new, but similar exemplars
- The following was my process to find additional exemplars:
 - Created API Hash YARA rule to Discover Potential New Exemplars
 - Analyzed New exemplars for the presence of *api_hash_func_slladd1* and *manual_symbol_resolution*
 - Created a tightly scoped YARA rule

Created a Tightly Scoped YARA Rule to Discover New Exemplars

- CERT/CC Massive Analysis and Storage System (MASS)
 - In order to conduct research into executable code, we maintain a large archive of potentially malicious software artifacts
 - The archive consists of more than a billion artifacts collected from early 2005 until present day
 - The corpus of artifacts allows us, the analysts, to conduct research on how different malware families and threat actors have evolved over time

Created a Tightly Scoped YARA Rule to Discover New Exemplars

Created API Hash YARA rule to Discover More Exemplars

- Used MASS and this YARA to discover 36 more exemplars for a total of 37
- This YARA rule represents the *push* of the API hash value (0x0038D13C), *push* of the DLL hash value (0x000D4E88), and the *call* to *manual_symbol_resolution*

```
rule api_hashes_2_call
{
  strings:
    (2019-02-22)
    $api_hashes_2_call = { 68 3C D1 38 00 68 88 4E 0D 00 E8 ?? ?? ?? ?? }
  condition:
    uint16(0) == 0x5a4d and $api_hashes_2_call
}
```

Created a Tightly Scoped YARA Rule to Discover New Exemplars

Analyzed New exemplars for the Presence of the Two Functions

- Refined the original YARA rule
- Realized that some of the new exemplars did not alert with the refined YARA rule
- Analyzed this subset of new exemplars that did not hit and discovered two slight variations in the API hashing function
- The first was an addition of 1 extra byte
- The second dealt with 64-bit files

```
rule energetic_bear_api_hashing_tool {
  meta:

    description = "Energetic Bear - API Hashing"
    assoc_report = "DHS Report TA17-293A"
    author = "CERT RE Team"
    version = "1"

  strings:
    $api_hash_func = { 8A 08 84 C9 74 0D 80 C9 60 01 CB C1 E3 01 03 45 10 EB ED }
    $http_push = "X-mode: push" nocase
    $http_pop = "X-mode: pop" nocase

  condition:
    $api_hash_func and (uint16(0) == 0x5a4d or $http_push or $http_pop)
```

Created a Tightly Scoped YARA Rule to Discover New Exemplars

Created a Tightly Scoped YARA rule

- Refined the YARA rule further to incorporate these two variations, for a total of 3 variations

```
rule energetic_bear_api_hashing_tool {
  meta:
    description = "Energetic Bear API Hashing Tool"
    assoc_report = "DHS Report TA17-293A"
    author = "CERT RE Team"
    version = "2"

  strings:
    $api_hash_func_v1 = { 8A 08 84 C9 74 ?? 80 C9 60 01 CB C1 E3 01 03 45 10 EB ED }
    $api_hash_func_v2 = { 8A 08 84 C9 74 ?? 80 C9 60 01 CB C1 E3 01 03 44 24 14 EB EC }
    $api_hash_func_x64 = { 8A 08 84 C9 74 ?? 80 C9 60 48 01 CB 48 C1 E3 01 48 03 45 20 EB EA }

    $http_push = "X-mode: push" nocase
    $http_pop = "X-mode: pop" nocase

  condition:
    $api_hash_func_v1 or $api_hash_func_v2 or $api_hash_func_x64 and (uint16(0) == 0x5a4d or $http_push or $http_pop)
}
```

Discovered API Hashes Found in New Exemplars

- Identified *sll1Add* routine API hash values found in all of the 37 exemplars
- All exemplars had API hash values for functions from kernel32.dll

Function	Byte Value (big endian)
CreateThread	14 F3 0C 00
ExitProcess	6A BC 06 00
GetSystemDirectoryA	E6 B2 9B 06
LoadLibraryA	86 57 0D 00
VirtualAlloc	42 31 0E 00
VirtualFree	8E 18 07 00
VirtualProtect	3C D1 38 00

Discovered API Hashes Found in New Exemplars

- Most of the exemplars had *sll1Add* routine API hash values for functions from `ws2_32.dll`

Function	Byte Value (big endian)
WSAGetLastError	70 71 71 00
WSAStartup	14 93 03 00
connect	7C 67 00 00
recv	C0 0C 00 00
send	D8 0C 00 00
socket	A4 36 00 00

Discovered API Hashes Found in New Exemplars

- There were a few outliers that had *s//1Add* routine API hash values for functions from wininet.dll

Function	Byte Value (big endian)
HttpAddRequestHeadersA	AE 57 5E 36
HttpEndRequestA	DA 03 6D 00
HttpOpenRequestA	DA BB DA 00
HttpQueryInfoA	EE C3 36 00
HttpSendRequestA	DA B3 DA 00
InternetCloseHandle	1A DE BB 06
InternetConnectA	BA 7B D7 00
InternetOpenA	02 F0 1A 00
InternetOpenUrlA	52 87 D7 00
InternetReadFile	62 81 D7 00
InternetSetOptionA	82 28 5E 03

Discovered API Hashes Found in New Exemplars

- These API hashes found in the exemplars from ws2_32.dll and wininet.dll show that these exemplars have potential network communications
 - The use of two different DLLs for network communications draws conclusions to the existence of at least 2 different versions of the API hashing tool
- Statically analyzed and/or debugged the exemplars to identify any network-based IOCs
- 33 of 37 exemplars, identified 29 unique IP address and port combinations
 - Includes private IP space

Discovered API Hashes Found in New Exemplars

- 4 of 37 the exemplars had structure outbound POST request.
- For 2 of these 4, I captured the requests in a pcap using FakeNet [vii]
- Inferred the outbound POST request structure from strings for the remaining 2

```
POST / HTTP/1.1
X-mode: pop
X-id: 0x00000000,0x5547a48a
User-Agent: Mozilla
Host: 187.234.55.76:8080
Content-Length: 0
Connection: Keep-Alive
Cache-Control: no-cache
```

```
X-mode: push\r\nX-type: more\r\nX-id:
0x00000000,0x523fe61c\r\n
X-mode: push\r\nX-type: last\r\nX-id:
0x00000000,0x523fe61c\r\n
X-mode: pop\r\n\r\nX-id: 0x00000000,0x523fe61c\r\n
Mozilla
POST
```

Questioned Attribution

- Tried to identify other public reporting or research related to this Energetic Bear API hashing tool
- I did not find any reporting or research

Questioned Attribution

- Since there was a link to Energetic Bear, I thought it could be a remote access trojan (RAT) like Havex, which is also attributed to this particular group
- Decided to explore the potential RAT association
- Discovered research by Veronica Valeros on “A Study of RATs: Third Timeline Iteration” [viii].
 - Contacted her directly and ask if she recalled any of the RATs use an API hashing technique
 - She could not recall, but mentioned that it could have been missed because she was not explicitly looking for this technique.
- Used her research to attempt to identify more RATs that use an API hashing technique
- I have not found any that use this technique

Future Work

- This brings me to a couple outstanding questions:
 - Why is this API hashing tool linked to Energetic Bear?
 - Who actually wrote the YARA rule from Figure 1 found in DHS Alert TA17-293A?
 - Can they provide more insight to this problem?
- I hope by publicly discussing that analysis that I can encourage information sharing and allow us, as a community, to urge for more detailed threat reporting.
- Lastly, I will be reaching out to MITRE ATT&CK team to ask for an additional technique, API Hashing, to be added to their framework.
 - During my analysis I could not find such explicitly technique listed

Results

- Expanded the corpus information from a trusted partner regarding DHS Alert TA17-293A
 - This information includes:
 - a more concise YARA rule, shown in Figure 8
 - Additional exemplars shown in Table 2 of the Appendix
 - Network communications, shown in Table 3 of Appendix
 - existence of at least 2 different versions
- If the attribution is correct and based on my research and analysis, this is may also be the **first** publicly documented report of API hashing technique being used by a nation state actor
- Hope that my questions about attribution push the cyber security community to start to ask for more better, more detailed threat reporting

Contact Information

Presenter

Kyle O'Meara

Sr. Member of the Technical Staff

Email: komeara@cert.org

Twitter: @cool_breeze26

Blog post is forthcoming. You will be able to find it on SEI's website

References

- i] [Department of Homeland Security \(DHS\) Alert \(TA17-293A\)](#)
- [ii] [CrowdStrike Global Threat Report: 2013 Year in Review](#)
- [iii] [APT Groups and Operations](#)
- [iv] [FireEye Flare IDA Plugins Github](#)
- [v] [Using Precalculated String Hashes when Reverse Engineering Shellcode](#)
- [vi] Sikorski, Michael and Honig, Andrew; Practical Malware Analysis; 2012
- [vii] [FakeNet](#)
- [viii] [A Study of RATs: Third Timeline Iteration](#) by Veronica Valeros