Indicators of Compromise in Memory Forensics

Chad Robertson
Indicators of compromise in memory forensics

GIAC (GCFA) Gold Certification

Author: Chad Robertson, chad@rocacon.com

Advisor: Tim Proffitt

Accepted: TBA

Abstract

Utilizing memory forensics during incident response provides valuable cyber threat intelligence. By both providing mechanisms to verify current compromise using known indicators and to discover additional indicators, memory forensics can be leveraged to identify, track, isolate and remediate more efficiently.
1.0 Introduction

There has been a recent increase in the availability of intelligence related to malware. New IP addresses, hostnames, MD5 hashes, mutex values, and other attacker artifacts are shared often. Historically, there exist many host-based and network-based standard methods to utilize these artifacts, such as intrusion detection/prevention systems, firewalls, anti-virus, and file whitelisting. While these solutions provide various benefits, they can fall short of confirming an infection.

Consider antivirus for example. In an article published June 11\textsuperscript{th}, 2012, MIT Technology Review boldly proclaimed “The Antivirus Era Is Over”. Mikko Hypponen, Chief Research Officer of F-Secure, wrote that it was “a spectacular failure of our company, and for the antivirus industry in general” (Hypponen, 2012) that they had possessed samples of Flame, one of the most complex pieces of malware ever discovered, for at least two years without examining it closely and developing detection mechanisms. (Hypponen, 2012) Later, in the same article, Mr. Hypponen states, “The truth is, consumer-grade antivirus products can’t protect against targeted malware created by well-resourced nation-states with bulging budgets. They can protect you against run-of-the-mill malware: banking trojans, keystroke loggers, and e-mail worms. But targeted attacks like these go to great lengths to avoid antivirus products on purpose. And the zero-day exploits used in these attacks are unknown to antivirus companies by definition.” (Hypponen, 2012)

SC Magazine’s Tom Cross states, “Although basic controls like anti-virus will always have a place in the security arsenal, they are not up to the task of defending networks against sophisticated, targeted attacks.” (Cross, 2012) “It is going to take human analysts to recognize the subtle and often unpredictable patterns of evidence that sophisticated attacks leave behind. Therefore, the best strategies are going to focus on arming incident responders with the tools that they need to monitor their environments and actively hunt for active attack activity.” (Cross, 2012)

“So what’s the next-generation solution? The future of security lies in shifting toward behavior-oriented scanning, says Dennis Pollutro, president and founder of cloud

Chad Robertson, chad@rocacon.com
security vendor Taasera. While "there will always be a place for signatures," security products have to begin identifying malware by what it's doing, rather than what it looks like, he says.” (Rashid, 2012)

“Several things have to happen before the malware infection results in damage or data theft on the compromised computer, which gives defenders a "couple hundred processes" to monitor for, Pollutro adds. Threat intelligence allows administrators to recognize patterns of behavior, such as creating directories on a file system or communicating with an IP address that had previously been flagged as suspicious.” (Rashid, 2012)

Malware can be identified by seeking out common TTP’s (tools, techniques, and procedures) used during development and infection. These relationships help analyst gain a better understanding of the adversary and develop attacker profiles. From the profiles we can draw inferences to better adapt and respond to attacks.

Categorization techniques vary among researchers. David French, Senior Malware Researcher at CERT suggests a malware-centric approach. “To express such relationships between files, we use the concept of a “malware family”, which is loosely defined as “a set of files related by objective criteria derived from the files themselves.” Using this definition, we can apply different criteria to different sets of files to form a family.” (French, 2012)

Michael Cloppert, a senior member of Lockheed Martin’s Computer Incident Response Team suggests taking a more adversary-centric approach. “The best way to behaviorally describe an adversary is by how he or she does his job…that "job" is compromising data, and therefore we describe our attacker in terms of the anatomy of their attacks.” (Cloppert, 2009). “We as analysts seek the most static of all indicators [those closest to the adversary] but often must settle for indicators further from the adversary until those key elements reveal themselves.” (Cloppert, 2009)

According to Greg Hoglund, Founder of HBGary, “It makes no sense to separate the human from the malware and TTP’s. They are two ends of the same spectrum. This is not a black and white science; it works because humans aren’t perfect. It works because
humans are creatures of habit and tend to use what they know. They use the same tools every day and don’t rewrite their malware every morning. They don’t have perfect OPSEC. They put their digital footprints out on the Internet long ago – and it’s usually just a few clicks away from discovery. There is a reflection of the threat actor behind every intrusion. To discount this is to discount forensic science.” (Hoglund, 2011)

According to The MITRE Corporation, “to be proactive, cyber defenders need to fundamentally change the nature of the game by stopping the adversary’s advance, preferably before the exploit stage of the attack [which] requires defenders to evolve from a defensive strategy based primarily on after-the-fact incident investigation and response to one driven by cyber threat intelligence.” (The MITRE Corporation, 2012)

Identification of singular characteristics of malware helps to automate future identification tasks. Often malware from the same family share these characteristics, which helps analysts, identify related files. These characteristics can be used by analysts to organize malware within repositories, root out additional infections present on the network, or identify new variants of malware.

Memory forensics allows analysts to “reconstruct the state of the original system, including what applications were running, what files those applications were accessing, which network connections were active, and many other artifacts” (Michael Ligh, 2010). Many free tools are available to analyst for local memory imaging as well as several enterprise solutions for remote imaging. While memory forensics shouldn’t be considered a replacement for any other security technology, it is most certainly a valuable tool and should be considered, as is stated in the Malware Analyst’s Cookbook, “extremely important to incident response” (Michael Ligh, 2010). This paper discusses analyzing malware. It does not describe designing a safe and effective malware analysis environment. If you are new to malware analysis and wish to perform any of the examples within this paper please spend some time researching how to do so safely.

Chad Robertson, chad@rocacon.com
2.0 Scope and Assumptions

The research discussed herein focuses on the creation of signatures to help speed up memory analysis during incident response. Because they are not meant to be used as host-based indicators deployed uniformly there is a greater tolerance for false positives. If, instead, the task was to develop signatures to deploy to all client machines then there would be a much greater need to tune each rule since, in that case, any false positive could have an immediate impact on the client experience. For this research, the resulting signatures will be run by the analyst, for the analyst, to expedite the identification, containment, and eradication of threats.

We will touch on many topics related to malware analysis and memory forensics. To limit the scope of this research, specific malware analysis techniques will not be discussed. We will rely on various publicly available materials from which to understand our topic. If malware analysis is of interest to the reader, they are encouraged to see the reference section for the articles cited and the appendix for a list of additional resources.

There are many tools mentioned throughout this paper. Detailing each tool is out of scope, but links to the tools mentioned will be included within the appendix for reference.

3.0 Finding Indicators

Indicators can come from a variety of sources. There are numerous blogs dedicated to malware reversing and analysis. “Malwaremustdie”, “Joe Security LLC”, and “Hooked on Mnemonics worked for me” to name a few (see Appendix 1). There are also many security companies that publish some of their own internal findings publicly. Organizations such as Mandiant, Fireeye, and Dell Secureworks are great sources of detailed reports that include indicators.

Another option is to acquire malware samples for analysis. Enterprising and ambitious analysts could mine the internet for malware themselves. There are a variety of sites (See Appendix 1) that note active malware sources that can be downloaded manually or scrapped via script (such as maltrieve - See Appendix 1) to automate
collection. Alternatively, one could setup a honeypot (such as Dionea - See Appendix 1) to glean current samples from the internet by allowing attackers access to seemingly vulnerable services.

Another approach is to turn to the numerous websites that provide users access to malware. Sites such as virusshare.com, malwr.com, malware.lu, offensivecomputing.net, and contagiodump offer access to millions of samples (See Appendix 1).

### 3.1 YARA

To help aid in scanning memory samples for indicators of compromise, we will use a tool called YARA. “YARA is a tool aimed at helping malware researchers to identify and classify malware samples. With YARA you can create descriptions of malware families based on textual or binary patterns contained on samples of those families. Each description consists of a set of strings and a Boolean expression which determines its logic.” (YARA in a nutshell, 2013)

YARA provides a mechanism to match indicators such as strings and hexadecimal within memory samples.

### 3.2 Indicators Defined

Michael Cloppert wrote a phenomenal blog post in 2009 on the computer forensic blog on SANS called Security Intelligence: Attacking the Kill Chain. In it he describes classifying indicators into one of three categories: atomic, computed, and behavioral.

Michael's description of each is shown here:

“Atomic indicators are pieces of data that are indicators of adversary activity on their own. Examples include IP addresses, email addresses, a static string in a Covert Command-and-control (C2) channel, or fully-qualified domain names (FQDN’s). Atomic indicators can be problematic, as they may or may not exclusively represent activity by an adversary.” (Cloppert, 2009)

“Computed indicators are those which are, well, computed. The most common
amongst these indicators are hashes of malicious files, but can also include specific data in decoded custom C2 protocols, etc. Your more complicated IDS signatures may fall into this category.” (Cloppert, 2009)

“Behavioral indicators are those which combine other indicators — including other behaviors - to form a profile.” (Cloppert, 2009)

As an example of a complex behavioral indicator, consider the following. An adversary is identified that tends to rely on “spear phished” email attachments. The emails tend to come from a specific range of IP space. When the attachment is executed, the dropper malware gains persistence by writing to specific registry keys and then installs a PoisonIvy variant. From there the attacker uses Windows Credential Editor to move laterally within the network. Once interesting data has been identified, Winrar is used to zip up the files and exfiltrate them to a set of IPs. These indicators could have been identified during separate incident response engagements, and may have little significance alone, but when viewed together build behavioral indicators that can be assigned to specific attacker profiles. We will discuss identifying atomic and computed indicators in depth here, and will leave the creation of behavior indicators to the reader.

Indeed, the identification of atomic and computed artifacts is the easiest step in malware analysis. Sources of unique strings, IP addresses, registry keys, file locations, etc. related to malware are numerous. The task of determining the relevance of each accumulated artifact and its contribution to the attacker profile falls upon the analyst team.

### 3.2.1 Avoiding poor indicators

As we have been discussing, not all indicators are equal. Some artifacts are easily changed by the attacker. For example, consider GhostRat – a popular RAT (remote access tool) used in many recently documented attacks. According to Norton, “the most stable indicator of GhostRat is its network communication. It is well documented and quite distinctive, as it always begins with a “magic word” which in its default configuration is “Gh0st”” (Norman, 2012). However, because the source code for Gh0stRAT is freely available, modifying this “magic word” is simple.
Figure 1

Several known values of Gh0stRAT’s “Magic word” -- Source: http://download01.norman.no/documents/ThemanyfacesofGh0stRat.pdf

4.0 Writing indicator-based YARA rules

Consider the differences between host-based signatures, such as those used by anti-virus, and incident response rules used for YARA. Because many anti-virus platforms have real-time functionality, they must constantly monitor the file system and memory for both static and heuristic based signatures. Thus, system performance is impacted by the simple addition of anti-virus, regardless of the quality of its signatures. If the signatures written for the anti-virus platform are vague, the resulting additional overhead on the system impacts the user experience at best by simply slowing down processing and, at worse, incorrectly identifying files as malicious, quarantining them, and denying the user access until a patch can be deployed.

YARA signatures, specifically those focused on memory forensics, involve only a snapshot of memory, frozen in time and independent of the host machine, thus there is no real-time performance impact to consider. Also, because incident response engagements typically involve a small number of systems, performing matches for less precise characteristics is tolerated and, in some cases, preferred.

To that end, as we create YARA signatures we will test them against a corpus of malware to assess the false positive rate. We will ask ourselves if, in some cases, defining signatures based upon characteristics that might be considered poor form for anti-virus platforms is more tolerated for memory analysis.
4.2 Python and IDA

To help speed up the process of creating YARA rules based on disassembled malware, we turn to IDA Pro and Python. Within idc.py, the Python plugin for IDA, the o_* constants allow for the identification of variable memory addresses within malware. These addresses require the use of wildcards within YARA. A very useful script to automate the substitution of the variable memory addresses with ‘??’ was written by Case Barnes of Accuvant (Barnes, 2012). His script relies on the following operands, o_mem, o_phase, o_displ, o_far, o_near, which are defined within idc.py:

```
o_void    = idaapi.o_void    # No Operand
o_reg     = idaapi.o_reg     # General Register (al,ax,es,ds...) reg
o_mem     = idaapi.o_mem     # Direct Memory Reference (DATA) addr
o_phrase  = idaapi.o_phrase  # Memory Ref [Base Reg + Index Reg] phrase
o_displ   = idaapi.o_displ   # Memory Reg [Base Reg + Index Reg + Displacement] phrase+addr
o_imm     = idaapi.o_imm     # Immediate Value value
o_far     = idaapi.o_far     # Immediate Far Address (CODE) addr
o_near    = idaapi.o_near    # Immediate Near Address (CODE) addr
o_idpspec0 = idaapi.o_idpspec0 # IDP specific type
o_idpspec1 = idaapi.o_idpspec1 # IDP specific type
o_idpspec2 = idaapi.o_idpspec2 # IDP specific type
o_idpspec3 = idaapi.o_idpspec3 # IDP specific type
o_idpspec4 = idaapi.o_idpspec4 # IDP specific type
o_idpspec5 = idaapi.o_idpspec5 # IDP specific type
o_last    = idaapi.o_last    # first unused type
```

*Figure 2
Operands within idc.py – Source: Author created*

5.0 Demonstrations

To test our ability to identify indicators in one piece of malware, and then use them to create signatures to identify the same or similar malware across multiple memory dumps, we will turn to three pieces of malware; Stabuniq, X0rb0t, and a sample from APT1. Each sample will be assessed for indicators, and then YARA will be used to scan across several memory images to verify the effectiveness of the rule.

5.1 Stabuniq

Stabuniq is not considered APT, but due to its availability and interesting characteristics it is an excellent example for our purpose.

I will refer to indicators identified within two public analysis posts:


According to Symantec, Stabuniq was first spotted in 2011 on servers belonging to financial institutions, banking firms, and credit unions. (Gutierrez, 2012) Symantec published an analysis in December 2012. During that same month, Mila Parkor of Contagio Malware Dump posted links to samples of the malware (see Appendix 1).

The initial executable contains a second executable that is decoded and dropped at runtime. Because our approach cares only about what eventually ends up in memory, the initial binary is of little interest. Therefore, to identify pertinent indicators, we must review the final executable dropped on the system.

After the malware has been run and the subsequent memory image acquired, similar artifacts to those noted within the online analysis can be seen using Volatility. To follow along, we must identify the injected process PID:

![Figure 3: Injected processes -- Source: Author created](image_url)
The malware uses StabililtyMutexString as its mutex. We can confirm that by using the Volatility mutantscan plugin.

![Figure 4](image-url)

*Output from Volatility’s mutantscan plugin -- Source: author created*

To create a YARA rule to identify this sample by this mutex, we must consider how the data looks within memory. One way to see is to use volshell to display the contents of memory:

![Figure 5](image-url)

*The mutex as stored in process memory -- Source: author created*

As you can see above, the string is encoded with two bytes per character. The ‘wide’ YARA modifier must be used:
Indicators of compromise in memory forensics

Chad Robertson, chad@rocacon.com

Figure 6

YARA rule to match Stabuniq’s mutex - Source: author created

The signature is checked and verified to match:

```
rule Stabuniq_strings
{
    strings: $mutex = "StabilityMutexString" wide
    condition: $mutex
}
```

Figure 7

Results shows match - Source: author created

Because the mutex name is likely easy to change, we return to the quequero.org analysis for a preferred indicator. The analysis also mentions the segment of code used to re-injecting Internet Explorer. Perhaps this function would be more difficult for the malware author to change and thus make a better signature.

```
.text:0040802E  call    dword ptr [eax+0ACh]...:
GetProcessVersion
.text:00408031  mov     [ebp+var_8], eax
.text:00408034  cmp     [ebp+var_8], 0
.text:00408036  jnx     short loc_408071
.text:0040803a  mov     [ebp+var_4], offset injectedThread...
.text:00408068  mov     ecx, [ebp+arg_0]
.text:0040806e  push    ecx
.text:0040806c  call    createRemoteIEThread
.text:00408071  loc_408071:

...; CODE XREF:
threadReinjectIE+38j
.text:00408071  push    1000
.text:00408076  mov     edx, [ebp+arg_0]
.text:00408079  call    dword ptr [edx+0A8h]...: Sleep
.text:0040807f  jmp     short loc_40800E
```

Figure 8

Interesting code block from Stabuniq - Source: http://quequero.org/2013/01/stabuniq-financial-infostealer-trojan-analysis/
Using the Python script mentioned above, we use this bit of code to create a YARA rule:

```
Using Python to create YARA rule - Source: Author created

While process could certainly be done manually, the script automatically inserts wildcards where needed to adjust for memory addressing.
```
To validate the rule, I will use various publicly available memory images, including all those linked to from the Volatility website. I will also include various known-infected memory images acquired during research. These memory images are shown below:

![Sample memory images - Source: author created](image)

The results are shown here:

```
Stabuniq [] /usr/share/memory_dumps/public//WINDOWSXP-stabuniq.raw
Stabuniq [] /usr/share/memory_dumps/public//WINDOWSXP-stabuniq-2.raw
```

*Stabuniq YARA signature matched - Source: author created*

As you can see, the rule matched the two Stabuniq-infected memory dumps, but none of the other dumps in the folder. To further test the resilience of the rule, we can extract the malware from memory by using Volatility’s malfind command:
Indicators of compromise in memory forensics

Chad Robertson, chad@rocacon.com

Figure 12

**IEXPLORE.EXE** injected process #1 - Source: author created

```
Process: IEXPLORE.EXE Pid: 1856 Address: 0x160000
Vad Tag: Vad5 Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 7, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x001610000 55 0b cc 83 ec 14 53 56 57 e8 00 00 00 00 5b b1 U.....SVW.....[
0x001610010 eb 0e 2b 40 00 89 5d fc 0b 45 00 0b 0d 0c 09 08 n+@...E.E.M...
0x001610020 14 02 00 00 c7 45 f4 00 00 00 00 5b 55 f4 6b d2 ......E......U.K.
0x001610030 28 0b 45 00 33 c9 66 0b 0c 10 32 12 00 00 05 c9 (.E.3.f...2......

0x160000 55 PUSH EBP
0x160001 8bec  MOV EBP, ESP
0x160003 83ec14  SUB ESP, 0x14
0x160006 53  PUSH EBX
0x160007 56  PUSH ESI
0x160008 57  PUSH EDI
0x160009 e80000000 CALL 0x16000e
0x16000e 55  POP EBX
0x16000f 81eb6e2b4000  SUB EBX, 0x402b6e
0x160015 89dfc  MOV [EBP-0x4], EBX
0x160018 8b4508  MOV EAX, [EBP+0x8]
0x16001b 8b4dfc  MOV ECX, [EBP+0x4]
0x16001e 89814020000  MOV [EAX+0x20], ECX
0x160024 c745f40000000  MOV DWORD [EBP-0xc], 0x0
0x16002b 8b55f4  MOV EDX, [EBP-0xc]
0x16002e 6b228  IMUL EDX, EDX, 0x28
0x160031 8b4500  MOV EAX, [EBP+0x0]
0x160034 33c9  XOR ECX, ECX
0x160036 660b8c1032120000  MOV CX, [EAX+EDX+0x1232]
0x160038 83c9  TEST ECX, ECX
```

Figure 13

**IEXPLORE.EXE** injected process #2 - Source: author created

```
Process: IEXPLORE.EXE Pid: 1856 Address: 0x160000
Vad Tag: Vad5 Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 7, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x001610000 55 0b cc 83 ec 14 53 56 57 e8 00 00 00 00 5b b1 U.....SVW.....[
0x001610010 eb 0e 2b 40 00 89 5d fc 0b 45 00 0b 0d 0c 09 08 n+@...E.E.M...
0x001610020 14 02 00 00 c7 45 f4 00 00 00 00 5b 55 f4 6b d2 ......E......U.K.
0x001610030 28 0b 45 00 33 c9 66 0b 0c 10 32 12 00 00 05 c9 (.E.3.f...2......

0x160000 55 PUSH EBP
0x160001 8bec  MOV EBP, ESP
0x160003 83ec14  SUB ESP, 0x14
0x160006 53  PUSH EBX
0x160007 56  PUSH ESI
0x160008 57  PUSH EDI
0x160009 e80000000 CALL 0x16000e
0x16000e 55  POP EBX
0x16000f 81eb6e2b4000  SUB EBX, 0x402b6e
0x160015 89dfc  MOV [EBP-0x4], EBX
0x160018 8b4508  MOV EAX, [EBP+0x8]
0x16001b 8b4dfc  MOV ECX, [EBP+0x4]
0x16001e 89814020000  MOV [EAX+0x20], ECX
0x160024 c745f40000000  MOV DWORD [EBP-0xc], 0x0
0x16002b 8b55f4  MOV EDX, [EBP-0xc]
0x16002e 6b228  IMUL EDX, EDX, 0x28
0x160031 8b4500  MOV EAX, [EBP+0x0]
0x160034 33c9  XOR ECX, ECX
0x160036 660b8c1032120000  MOV CX, [EAX+EDX+0x1232]
0x160038 83c9  TEST ECX, ECX
```
If we now run the same scan against the dumped binaries we see the same results:

```
Stabuniq [] /usr/share/malware//process.0x815f/029.0x160000.dmp
Stabuniq [] /usr/share/malware//process.0x81752c10.0x160000.dmp
```

**Figure 14**

*Stabuniq YARA rule matched - Source: author created*

As a final verification, we can use Virus Share's malware corpus. Scanning 131073 samples returns zero results.

### 5.2 X0rb0t

Next, we take a look at one of the public analysis posted by Malware.lu called X0rb0t. The analysis can be found here:

1. [http://code.google.com/p/malware-lu/wiki/en_x0rb0t_analysis](http://code.google.com/p/malware-lu/wiki/en_x0rb0t_analysis)

This malware sample uses XOR for encoding. Let’s focus on that function to create a YARA rule to test.

The function is shown below:

```
.text:00406CDE  cnr  [ebp+var_C], 7
.text:00406CE2  jnb  short loc_406CFF
.text:00406CE4  mov  eax, [ebp+Str]
.text:00406CE7  add  eax, [ebp+var_8]
.text:00406CEA  mov  ecx, [ebp+var_C]
.text:00406CED  mov  dl, [eax]
.text:00406CEF  xor  dl, byte_40EE60[ecx]
.text:00406CF5  mov  eax, [ebp+Str]
.text:00406CF8  add  eax, [ebp+var_8]
.text:00406CFB  mov  [eax], dl
.text:00406CFD  jmp  short loc_406CDF
```

**Figure 15**

*Related xor function - Source: author created*
and the resulting YARA rule:

```
DISASSSEMBLY:
jnb short loc_406CF
mov eax, [ebp+Str]  
add eax, [ebp+var_B] 
mov ecx, [ebp+var_C] 
mov dl, [eax] 
xor dl, [byte_40E60|ecx] 
mov eax, [ebp+Str] 
add eax, [ebp+var_B] 
mov [eax], dl 
jmp short loc_406CD5

BYTES:
73 1b 86 45 fc 03 45 f8 8b 4d f4 6a 10 32 91 60 ee 40 00 8b 45 fc 03 45 f8 80 10 eb d6

BYTES PER LINE:
73 1b
86 45 fc
03 45 f8
8b 4d f4
6a 10
32 91 60 ee
40 00
8b 45 fc
03 45 f8
88 10
eb d6

WILD-CARDED MEMORY BYTES:
73 ?? 8b ?? ?? 03 ?? ?? 8b ?? ?? 8a ?? 32 ?? ?? ?? ?? ?? ?? ?? ?? 8b ?? ?? 03 ?? ?? 80 ?? ?? ?? ??

YARA SIGNATURES:
rule Sig
{
    strings: $hex_string = (73 ?? 8b ?? ?? 03 ?? ?? 8b ?? ?? 8a ?? 32 ?? ?? ?? ?? ?? ?? ?? ?? 8b ?? ?? 03 ?? ?? 88
    condition: $hex_string
}
```

**Figure 16**

Output of Python script in IDA - Source: author created

When tested against the same set of memory dumps as was used for Stabuniq, with the addition of a memory dump from an x0rb0t infection, the Yara rule matches only the known x0rb0t-infected dump:

```
X0rb0t [] /usr/share/memory_dumps/public//WIND0WSXP-x0rb0t.raw
```

**Figure 17**

X0rb0t YARA rule matched - Source: author created

When run against a corpus of 131073 files, the rule matches 8 times. Taking a closer look at one of the matches (503783a11080777a35b6349099fb3c3d) revels that the same function is utilized within both samples.
Below, on the right is the segment from x0rb0t, and the left the same code segment from 503783a11080777a35b6349099fb3c3d:

![Figure 18](image)

Output of Python script in IDA - Source: author created

As expected, the same instructions are present in both binaries.

### 5.3 APT1

Soon after Mandiant released their report on APT1 (See appendix 1), Alienvault released several YARA signatures to identify malware seemingly associated with that group. Both the YARA signatures and the malware samples are easily found online (See Appendix 1). To test the effectiveness of utilizing the YARA rules to identify APT1 activity within memory, a sample from the set is selected (8934aeed5d213fe29e858ee616a6ec7), executed, and the memory from the compromised host dumped to disk.

To start, we test the Alienvault YARA rule against the malicious binary to verify the rule works:

![Figure 19](image)

The known malicious binary matches the rule – Source: Author created
The Alienvault rule matches the malware as expected. Now, the malicious binary is executed on a test system, and the memory from the test system acquired. The same YARA rule is then used to scan the memory dump:

![YARA rule matching the memory dump](image)

*Figure 20*

The *Alienvault YARA rule matches the memory dump – Source: Author created*

With very little effort we were able to take publically available indicators contained within YARA rules and make them actionable against the test machine. The same techniques can be utilized to identify compromised systems across production environments during incident response.

**Conclusion**

Incident response is evolving. Malware authors are constantly coming up with new ways to compromise systems. They have the ability to adapt quickly and often circumvent our slower moving, often signature-based, archaic security solutions. As such, we must begin to consider how we can achieve similar agility in the way we respond. Perhaps the best way to meet today’s threats is to stop relying solely on these legacy tools and begin to focus more on the analyst, our human capital, to piece together
malware artifacts.

The research conducted during the writing of this paper set out to identify how memory forensics can be leveraged to aid incident response. We began by discussing the limitations of standard security platforms. Next, we discussed how behavioral indicators can be created based on the tools, techniques, and procedures utilized by attackers. Then, we discussed how we might find data to be used as indicators, how YARA can be used to aid us, and how we might label and group our identified artifacts. Finally, we took a quick look at two pieces of malware and how we can use readily available analysis data to identify them using YARA.

With the techniques discussed here, an analyst can begin to accumulate and utilize indicators from both publicly disclosed sources and private research. These indicators can then grouped together to form behavioral indicators to move our overall intelligence closer to the adversary. Volatility provides analysts an unprecedented capability to analyze memory images and acquire intelligence. Finally, YARA can be leveraged to identify those indicators across binary data and memory images, allowing for rapid response capabilities.
References


Chad Robertson, chad@rocacon.com
Appendix A

Blogs, companies, and tools mentioned within this paper:

http://malwaremustdie.blogspot.jp/ - Malware Must Die!
http://joe4security.blogspot.ch - Joe Security LLC
http://hooked-on-mnemonics.blogspot.com - Hooked on Mnemonics worked for me
http://www.mandiant.com/ - Mandiant
http://www.fireeye.com/ - FireEye
http://www.secureworks.com/cyber-threat-intelligence/blog - Dell Secureworks
https://github.com/technoskald/maltrieve - Maltrieve
http://dionaea.carnivore.it – Malware honeypot
http://virusshare.com/ - Virus Share
http://malwr.com/ - Malwr
http://offensivecomputing.net/ - Offensive Computing / Open Malware
http://contagiodump.blogspot.com/ - Contagio Malware Dump
http://code.google.com/p/yara-project/ - YARA Project
http://www.threatexpert.com/ Threat Expert automated threat analysis
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Munich September 2020</td>
<td>Munich, DE</td>
<td>Sep 14, 2020 - Sep 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Australia Spring 2020</td>
<td>, AU</td>
<td>Sep 21, 2020 - Oct 03, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Antonio Fall 2020</td>
<td>San Antonio, TXUS</td>
<td>Sep 28, 2020 - Oct 03, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS FOR500 Milan 2020 (In Italian)</td>
<td>Milan, IT</td>
<td>Oct 05, 2020 - Oct 10, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Amsterdam October 2020</td>
<td>Amsterdam, NL</td>
<td>Oct 05, 2020 - Oct 10, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Dallas Fall 2020</td>
<td>Dallas, TXUS</td>
<td>Oct 19, 2020 - Oct 24, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Tel Aviv November 2020</td>
<td>Tel Aviv, IL</td>
<td>Nov 01, 2020 - Nov 05, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Sydney 2020</td>
<td>Sydney, AU</td>
<td>Nov 02, 2020 - Nov 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Rocky Mountain Fall 2020</td>
<td>Denver, COUS</td>
<td>Nov 02, 2020 - Nov 07, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Krakow November 2020</td>
<td>Krakow, PL</td>
<td>Nov 02, 2020 - Nov 07, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Gulf Region 2020</td>
<td>Dubai, AE</td>
<td>Nov 07, 2020 - Nov 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Diego Fall 2020</td>
<td>San Diego, CAUS</td>
<td>Nov 16, 2020 - Nov 21, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich November 2020</td>
<td>Munich, DE</td>
<td>Nov 16, 2020 - Nov 21, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Amsterdam November 2020</td>
<td>Amsterdam, NL</td>
<td>Nov 16, 2020 - Nov 21, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>OnlineUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>