RawPOS Malware Rides Again

threatvector.cylance.com/en_us/home/rawpos-malware.html

The BlackBerry Cylance Threat Research Team

Summary

As part of a recent forensics investigation by the Cylance Consulting Services team, we uncovered some new RawPOS malware. This family of POS malware has been widely documented in operation since 2008. Numerous retail operations of various sizes have been compromised with this malware and its variants.

Rather than rehash old malware, our intent is to discuss 'signature fidelity' and explain through technical detail why poorly-written signatures give people a false sense of security. This 'antivirus is dead' argument is often presented, but with little technical detail to highlight specifically why this is the case.

In our example below, the RawPOS variant went undetected for well over 30 days by a legacy antivirus (AV) vendor. By the time the vendor deployed custom DAT files, the only samples identified were in the quarantine directory of CylancePROTECT®. Fortunately, this customer deployed CylancePROTECT in time and prevented any data exfiltration.

At the end of this post, we'll provide an updated yara file for identifying all variants of the RawPOS dumper, as well as some sha256 hashes of the new variant.

RawPOS Overview

RawPOS is documented rather extensively across the web – for example, here and here. The variant we found is slightly updated, and we'll focus on the memory dumper component: (0ca08c10a79cddbb359354f59ba988e77892e16dce873b5ba8e20eb053af8a18).

There also appears to be three open source signatures located on Github:

https://raw.githubusercontent.com/mattulm/sfiles_yara/master/pos/rawpos_POS.yar
https://raw.githubusercontent.com/mattulm/sfiles_yara/master/pos/RawPOS2015_dumper_old.yar
https://github.com/mattulm/sfiles_yara/blob/master/pos/RawPOS2015_dumper.yar

Fig1-RawPOS

Figure 1

The above signature relies mostly on string literals and requires little developmental effort to adjust to evade, as we'll see later.

Code Diff Analysis

This level of analysis will explore the functional changes to the code itself. We'll highlight differences between two variants of the same memory dump component. One sample was compiled in March 2015, and the updated variant was compiled in January 2017. The resulting diff analysis shows little functional change. The new variant is largely similar to the 2015 variant.

This program has an updated naming scheme for where it places the dumped memory files. It also removes the 'help' text from the binary. As of writing, this file was undetected by the legacy AV vendor we used in this instance. Let's compare this version with the previous version that was documented in March 2015 (252C13FE7A7E4F1E9E9227678C7156630A9D15AA457ADD2FF8EA5BCDB3403CD4) to see what changed.

We'll use binary diffing to compare the code changes between the two variants to show the developmental effort the author used to create the new variant. Often when malware is updated, small changes will be added to the code base to enable new functionality or to evade legacy AV signatures. The tool we'll use is diaphora, which is a great open-source resource. Diaphora takes an IDA database as input and compares against a second IDA database to determine what code regions are similar. Some screenshots highlighting some of the analysis between the two hashes are listed below.

Of the 454 functions identified within this binary, 445 are 100% matches with the 2015 (252C13FE7A7E4F1E9E9227678C7156630A9D15AA457ADD2FF8EA5BCDB3403CD4) variant.

The remaining nine functions are partial matches.

Fig2-RawPOS

Figure 2

Three functions appear in the older variant only:

Fig3-RawPOS

Figure 3

Only three unmatched functions are missing in the new variant. 0040146E is the original functional version of the snapshotting module that has been removed (as discussed below). 004013EC is contained within the new variant at location 004012C9, but has no cross-references to this code from within the binary per the IDA disassembly. 004013A0 has been removed completely in the new version, as this was the calling function to 004012C9 in the 2015 variant.

Two new functions appear only in the newer variant:

Fig4-RawPOS

Figure 4

The two unmatched functions are a statically-linked strcpy and a snapshot by pid function located at 00401A6C. The disassembly of the snapshot by pid function is pictured below. This allows the memory dumper to take a snapshot if you provide a pid function. This functionality was contained within the 2015 version and is located at 0x40146E. The disassembly is below.

Fig5-RawPOS

Figure 5

The similarity between the two functions is pretty close, and can further be illustrated with the contrasting graph layout from the 2015 variant of this function at 0x40146E:

Fig6-RawPOS

Figure 6

Fig7-RawPOS

Figure 7

In summary, this variant has roughly no new functionality. It has even removed some functionality, which is rare considering developers code to add features. The big question is, why would a malware author *remove* code from their newer variant? This is most likely an attempt to evade signatures, as evidenced on the code areas that changed. With this in mind, we'll now compare the string layout between the two different files.

String Diff

Capturing a string diff between the two samples is accomplished by using the following code snippet in python. The first and second arguments are output from strings tools (strings.exe from sysinternals on windows boxes was used as input). Piping this string output to a file, and then supplying each of those files as command line arguments to the python script, will give you some differential analysis of the strings:

```
import sys


with open(sys.argv[1], 'r') as f:

 rawpos_newvar_txt = f.readlines()

with open(sys.argv[2], 'r') as f2:

 rawpos_315_txt = f2.readlines()

#print("Common strings\nF1:{}\nF2:{}:\n{}".format(sys.argv[1], sys.argv[2],
"".join(set(rawpos_newvar_txt) & set(rawpos_315_txt))))

#print("New Variant only strings\nF1:{}\nF2:{}:\n{}".format(sys.argv[1], sys.argv[2], "".join(
set(rawpos_newvar_txt) - set(rawpos_315_txt))))

print("Old Variant only strings\nF1:{}\nF2:{}:\n{}".format(sys.argv[1], sys.argv[2],
"".join(set(rawpos_315_txt) - set(rawpos_newvar_txt) )))
```

This will take strings output from the two binaries and compare them using Python sets. Analyzing the differences/similarities between these two sample sets, we can observe how the string literal composition has changed between variants. The print lines will give you strings that are common to both old/new variants, strings that are in the new variant only, and strings that are only in the old variant. Comparing this output between the open source signatures released for this family, we notice the following differences:

The regex portion seems to have changed slightly based on the open source signatures:

“((B(([0-9]{13,16})” (oldver)
“(B(([0-9]{13,16})” (newvar)

The string literal for the track data has changed:

"Found track data at %s with PID %d" (oldver)
“Found some data at %s with process id %d!” (newver)

The prompt for which PID to dump has changed:

“Enter Process Id:” (oldver)
“Enter PID:” (newver)

The how to help strings have been removed:

“" Dump private process memory by PID" (oldver)

The extra info has been removed as well. This string was contained in the 004103A0 code that is no longer there:

“Dumping private memory for pid %s to %s.dmp..." (oldver)

The how to help strings have been removed:

“Full private dump of all running processes” (oldver)

The naming convention of the dump has changed:

“memdump\\%s-%d.dmp” (oldver)
“memdump\quota.prNam-%s-pID-%d.dmp” (newver)

The command string has also been elongated to avoid this signature:

“mkdir memdump >NUL 2>NUL” (oldver)
“mkdir memdump >NUL 2>NUL” (newver)

Last, but not least, we have the presence of the following string in the new variant. It’s a surprising positive note to the security community:
iknowyouwatchingthisandilikeyou.exe

The effort required to evade these signatures that were built on string literals is rather low. Changing the amount of prints used in 40146E (old variant), removal of the help banner, and some simple string manipulation was all that was required to evade these signatures. The second open source signature has the right idea by using $_main to generate a signature off the function that performs time-check, which has not changed. This still misses the mark as the function has changed, however, the signature needs adjustment to catch this version that has the time-check.

Conclusion

The above analysis shows how little effort has to be invested to bypass signature-based technologies. This shows that malware with roughly the same functional capabilities can adopt minor tweaks and evade legacy AV vendors.

Relying on signature-based detection mechanisms as a core component of your security stack can lull one into a false sense of security. The alerts will be generated on variants that are known/documented and won't be future-proofed for any similar familial variants post signature deployment. The level of development effort that this author had to commit to avoid this signature has been shown to be pretty low.

Yara File:

```
rule RawPOS_dumper

{

 meta:

 author = "Cylance Inc."

 date = "2017-01-24"

 description = "Used to detect all RawPOS RAM dumper(s)"

              strings:

                            $time_func = { 55 8b ec 81 c4 ?? ?? ?? ?? 53 56 57 8b ?? ?? 8b ?? ?? 6a 00 e8 ?? ?? ?? ?? 59 a3 ?? ?? ?? ?? 6a 00 e8 ?? ?? ?? ?? 59 3d ?? ?? ?? ?? 7e ?? 33 c0 e9 ?? ?? ?? ??}

 $enum_proc_func = { 55 8b ec 81 c4 ?? ?? ?? ?? 50 81 c4 ?? ?? ?? ?? 53 56 57 be c8 b9 42 00 8d ?? ?? ?? ?? ?? b9 41 00 00 00 f3 ?? 8d ?? ?? 50 68 a0 0f 00 00 8d ?? ?? ?? ?? ?? 52 e8 ?? ?? ?? ?? 85 c0 0f ?? ?? ?? ?? ??}

 $open_proc_func = { 8b f0 85 f6 0f ?? ?? ?? ?? ?? 8d ?? ?? 50 6a 04 8d ?? ?? 52 56 e8 ?? ?? ?? ?? 85 c0 74 ?? 68 04 01 00 00 8d ?? ?? ?? ?? ?? 51 ff ?? ?? 56 e8 ?? ?? ?? ?? 56 e8 ?? ?? ?? ??}

              condition:
```

```
        $enum_proc_func or $time_func or $open_proc_func

}
```

SHA256's of New Variant:

a2e720a2c538347144aee50ae85ebfdaf3fdffcfc731af732be5d3d82cd08b18

fe8637ef9be609951aa218942d46a535ba771236668a49a84512b18b02e9fbee

0ca08c10a79cddbb359354f59ba988e77892e16dce873b5ba8e20eb053af8a18

4bd1cc0a38117af7d268c29592ef754e51ce5674e26168c6bb613302f3c62fb8

967fcbc7abcb328afb1dbfd72d68636c478d7369e674d622799b8dfd66230112

The other function is for _strcpy, which has been statically linked in the new variant where the old variant chose to use _strncpy. Why did the malware authors implement this change? If we look at some open source signatures for this specific variant of RawPOS, we may uncover the answer. Below are some signatures that are available:

https://github.com/mattulm/sfiles_yara/blob/master/pos/rawpos_POS.yar *(good, but still fails)*
https://github.com/mattulm/sfiles_yara/blob/master/pos/RawPOS2015_dumper_old.yar

(bad, online)

If you use our endpoint protection product, CylancePROTECT®, you were already protected from this attack. If you don't have CylancePROTECT, contact us to learn how our AI based solution can predict and prevent unknown and emerging threats.

The BlackBerry Cylance Threat Research Team

About The BlackBerry Cylance Threat Research Team

The BlackBerry Cylance Threat Research team examines malware and suspected malware to better identify its abilities, function and attack vectors. Threat Research is on the frontline of information security and often deeply examines malicious software, which puts us in a unique position to discuss never-seen-before threats.