root9B

TECHNICAL FOLLOW UP - APT28

MALWARE ANALYSIS

ROOT9B
AUG 2015

TABLE OF CONTENTS

INTRODUCTION

In May 2015, root9B released an APT28 Threat Defiance report (1) detailing pre-event indicators and threat information regarding a pending attack on several entities. This follow-up report is focused on providing additional insight and technical analysis of a malware sample that was originally reported.

Approximately 45 days after the release of the root9B report, Netzpolitik released a report on a breach of the German Parliament. The Netzpolitik report (2) detailed the malware and methods employed in the breach and attributed the event to APT28. The attack on the German Parliament used similar malware and the same command and control infrastructure that was identified in the original root9B report.

The following information is root9B's malware analysis of the malicious Dynamic Link Library (DLL) noted in our May 2015 report and presents a strong link to the recovered malware sample reported in the German Parliament exploit. Both samples appear to have been created from the same code base and share the same command and control infrastructure. This report provides additional security measures to defend against this variant of the malware.

Throughout the report, "sample 1" refers to the Netzpolitik malware sample which was described in Claudio Guarnieri's report. "Sample 2" refers to the .DLL sample of the malware analyzed by root9B.

Technical analysis and credits follow.

FILE ANALYSIS

STATIC INFORMATION

SAMPLE 1: HASHES OF ANALYZED FILE(S) (NOTED IN NETZPOLITIK REPORT)

Artifact #1

MD5: 77e7fb6b56c3ece4ef4e93b6dc608be0

SHA1: f46f84e53263a33e266aae520cb2c1bd0a73354e

SHA256 5130f600cd9a9cdc82d4bad938b20cbd2f699aadb76e7f3f1a93602330d9997d

Artifact #2

MD5: 5e70a5c47c6b59dae7faf0f2d62b28b3

SHA1: cdeea936331fcdd8158c876e9d23539f8976c305

SHA256: 730a0e3daf0b54f065bdd2ca427fbe10e8d4e28646a5dc40cbcfb15e1702ed9a

1. For more information on root9B's previous report, please see http://www.mediafire.com/view/bdr77piwp0ij0qz/FSOFACY.pdf

2. For more information on the German Parliament exploit event, please see https://netzpolitik.org/2015/digital-attack-on-german-parliament-investigative-report-on-the-hack-of-the-left-party-infrastructure-in-bundestag/

SAMPLE 2: HASHES OF ANALYZED FILE(S) (NOTED IN ORIGINAL ROOT9B REPORT)

MD5: 800AF1C9D341B846A856A1E686BE6A3E

SHA1: 0450AAF8ED309CA6BAF303837701B5B23AAC6F05

SHA256: 566ab945f61be016bfd9e83cc1b64f783b9b8deb891e6d504d3442bc8281b092

SSDEEP: 24576:od9xrou+n5LzE8y//TiOxxbto2D1yvIQnQS7PcBUivF5p/WjYwqr:M9l7+5Lzm2SClQvl+BAvbp/WjYwqr

IMPHASH: f37de4514467bed1e93e37fc7eea4505

PEHASH: b6d0f65b6edb2daecc0a389a9f8159fbb600383c

AUTHENTIHASH: 2d8f43ec069b9141f23e8d6aad847f4cd92abd0ff3df9b66f3ca11c314596e95

SAMPLE 2: PE INFORMATION

Linker Version: 10.0

Compile Date: 14/04/2014 13:13:59 GMT

Architecture: i386

Type: Shared Library (DLL)

Size: 1048064 bytes

Static Libraries: OpenSSL 1.0.1e 11 Feb 2013

PE Sections for Sample 2 DLL Variant MD5: 800af1c9d341b846a856a1e686be6a3e					
Name	VirtAddr	VirtSize	RawSize	MD5	Entropy
.text	0x1000	0xb5f0b	0xb6000	6d15c21fe25b1ef0b0b68c71ef0ed253	6.696818
.rdata	0xb7000	0x34d4b	0x34e00	e54cd9fb9ca40453bb13cb760491a9b3	5.682972
.data	0xec000	0xd970	0x9c00	5aeaaba85d77fc9ff2a7b49c4f94ce9c	5.260893
.reloc	0xfa000	0xae12	0xb000	9073eb2828cce8ce307ed4b9d526db3e	5.645191

SAMPLE 2: SUSPICIOUS IAT ENTRIES

Samples 1 and 2 both share the same suspicious IAT entries; however, the specific order of functions within the source files is different, which is illustrated by the different IMPHASH values for both sample variants.

accept (Ordinal #1)

bind (Ordinal #2)

closesocket (Ordinal #3)

connect (Ordinal #4)

CreateFileA

CreateFileW

CreateThread

ExitThread

FindFirstFileExA

GetCommandLineA

GetCommandLineW

GetDriveTypeA

GetDriveTypeW

GetModuleFileNameA

GetModuleFileNameW

GetModuleHandleA

GetModuleHandleW

GetProcAddress

GetStartupInfoW

GetTickCount

GetVersionExA

IsDebuggerPresent

listen (Ordinal #13)

LoadLibraryA

LoadLibraryW

recvfrom (Ordinal #17)

recv (Ordinal #16)

send (Ordinal #19)

sendto (Ordinal #20)

Sleep

socket (Ordinal #23)

TerminateProcess

UnhandledExceptionFilter

WriteFile

WSAStartup (Ordinal #115)

SAMPLE 2: PE EXPORTS

Sample 1: MD5: 800AF1C9D341B846A856A1E686BE6A3E, exports the “start” function

Sample 2: MD5: 5e70a5c47c6b59dae7faf0f2d62b28b3, exports the same named “start” function.

SAMPLE 2: OTHER

IP Callback: 176.31.112.10, configurable via command line argument

Port: 443, configurable via command line argument

Secure Protocol: Yes, possibly configurable via command line argument

SAMPLE 2: METHODOLOGY

Run the following commands inside a sandboxed environment:

Rundll32.exe <name_of_malware.dll>,start <arguments>

SAMPLE 2: COMPILATION CHARACTERISTICS

Compiler: Microsoft VC++ Compiler version 10.0

Table 1: Analysis of __security_init_cookie function

	Visual C++ 2013 (12.0)	Visual C++ 2012 (11.0)	Visual C++ 2010 (10.0)	Visual C++ 2008 (9.0)
Hot Patch Bytes Present	No	No	Yes	Yes
Local Variable Count	3	3	2	2
Presence of OR EAX, 0x4711	Yes	Yes	Yes	No

Table 1: Verifies that the binary is consistent with compilation by Visual C++ compiler version 10.0. This information is also in the IMAGE_OPTIONAL_HEADER as 10.0 for Linker Version.

SAMPLE 2: OPENSSL CHARACTERISTICS

Version: 1.0.1e 11 February 2013

Compiled By: Visual C++ Compiler 10.0

root9B analyst generated two IDA signature sets by compiling OpenSSL 1.0.1e two different times with different versions of Visual C++ compiler. These two IDA signature sets were applied against the malware to determine the most likely compiler version. One of the signature sets was compiled with Visual C++ compiler version 10.0 and the other was compiled with Visual C++ compiler version 12.0. When the Visual C++ compiler version 12.0 was applied to the malware, IDA was unable to match libeay32.sig and ssleay32.sig to more than one-third of the library functions. However, when the signatures were generated using the Visual C++ compiler version 10.0, there was over 95% coverage of the library functions. This indicates that Visual C++ compiler version 10.0 most likely compiled the OpenSSL static library contained in the malware. This appears to be consistent with the same compiler used in the malware itself.

COMPARISON TO GERMAN PARLIAMENT MALWARE

Figure 1 shows the IDA code graph of sample 2 as compared against Sample 1. The chief difference noted thus far is that one was developed as an executable (German Parliament analyzed sample 1) while the other was created as a DLL (root9B analyzed sample 2).

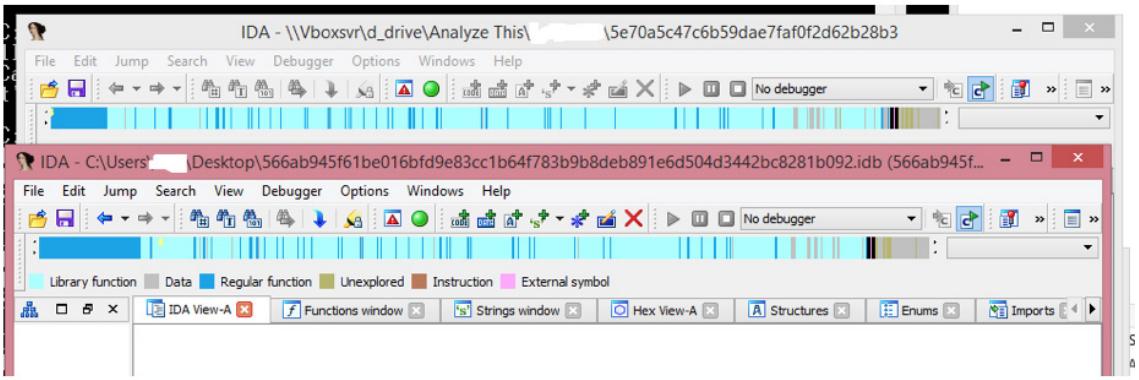


Figure 1: IDA Color Graph Comparison after Signatures (10.0) were applied (Sample 1 top, Sample 2 bottom)

Another similarity is in a large function at the very beginning of the text section. Its only purpose is to move a set of embedded items (4 kilobytes each, 8 total) into memory. The actual contents of these sections are unknown and appear to be encrypted. However, both samples contain the same byte variants. The fact that there is a total of 8kb of identical data moved into memory by both samples, using a nearly identical allocation methodology, indicates a high likelihood that both malware samples are from the same adversary.

Figure 2 shows a comparison of byte distribution in the two segments of allocations (sample 2 only) and an AES256 encrypted blob generated at root9B. A side by side comparison reveals they are very similar in distribution given the size of the data block. Please keep in mind this may not be quite the right threshold to differentiate with 100% accuracy between obfuscation and encryption since the data blobs are very small in size.

The allocation was not a known file format and was not executable code in the current state. This particular allocation is stack-based and the DLL was compiled with Data Execution Prevention (DEP) enabled. Since the allocation is stack-based and DEP is enabled, it cannot be executed unless it is first moved to another portion of executable memory. Based on the form of allocation in use by the code, root9B doubts this code can be directly executed by this process in memory. However, we have not ruled out that this could be an encrypted binary that is written to disk and executed via some other run mechanism. No disassembler used was able to translate this as executable code in its existing state.

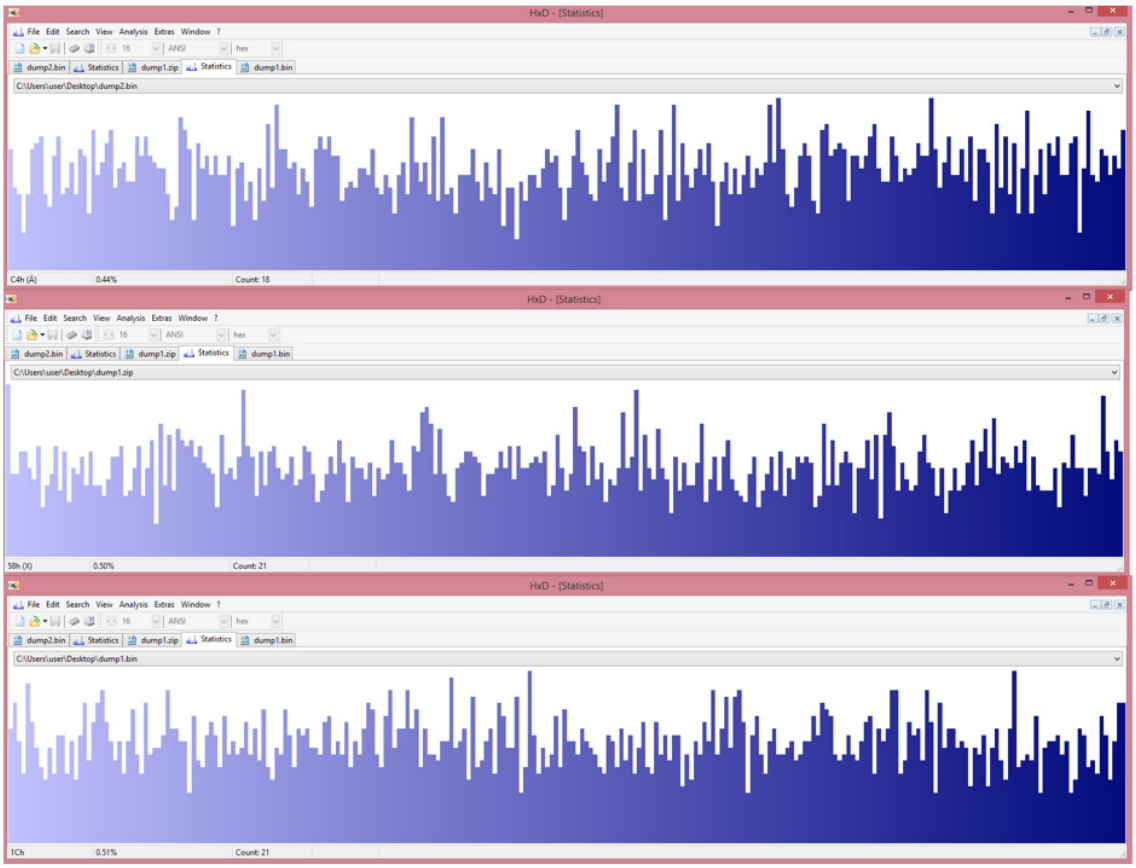


Figure 2: Frequency Analysis of Bytes Values within Data Blocks (top, bottom) versus Encrypted Segment (middle)

Figure 3 shows a comparison of the instructions writing the memory blob; one sample writes 4 bytes at a time while the other sample writes 1 byte at a time. Please take note that in both malware samples the contents are the same.

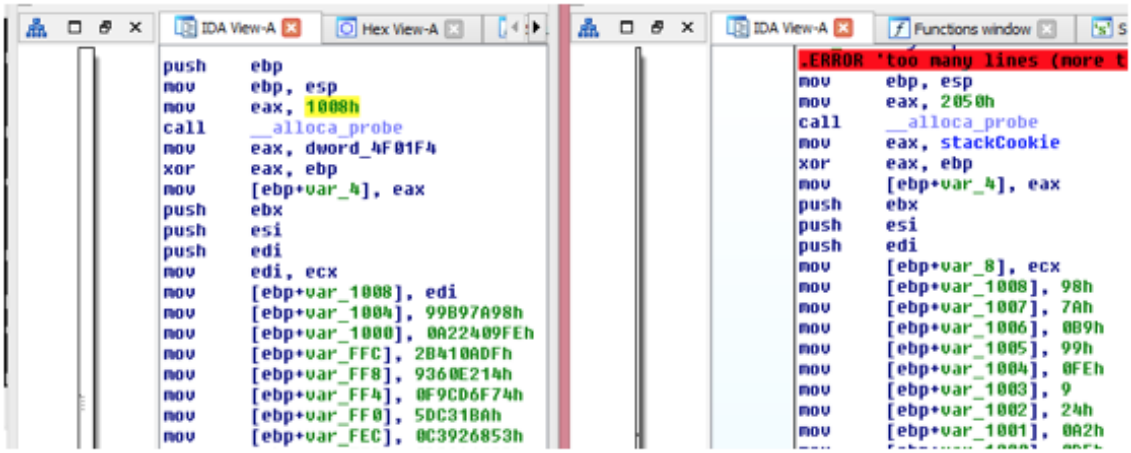


Figure 3: Comparison of large allocation between Sample 1 (left) and Sample 2 (right)

During execution of sample 2, we were able to capture a set of packets that have a single leading DWORD followed by 0x20 bytes of packet data. The packet, which it generated consistently, maintained the same structure throughout several executions of the malware. This is the same format and length in Sample 1. Figure 4 illustrates a hexdump of the beaconing packets received by a Secure Sockets Layer (SSL) listening post that was configured for analysis.

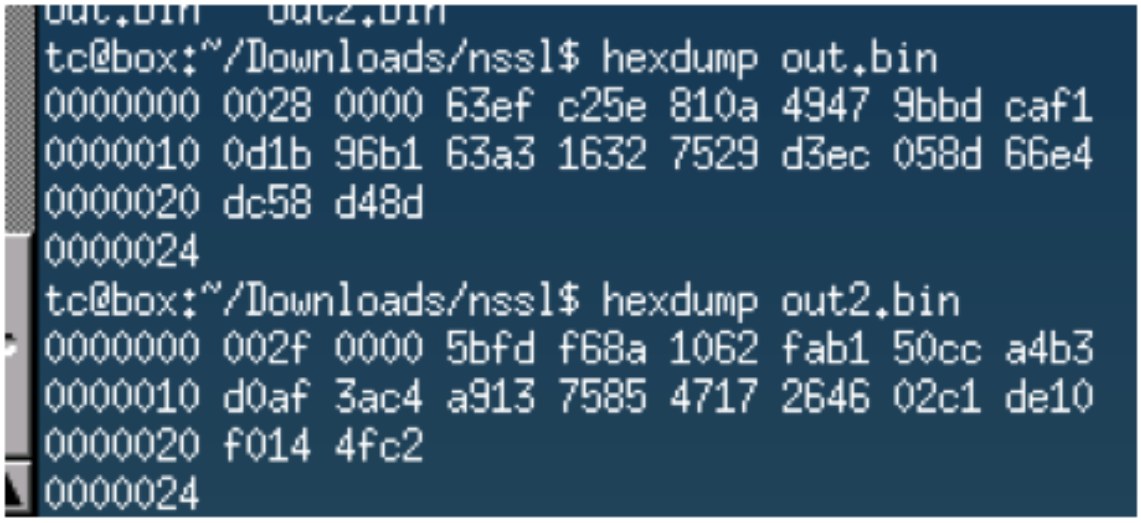


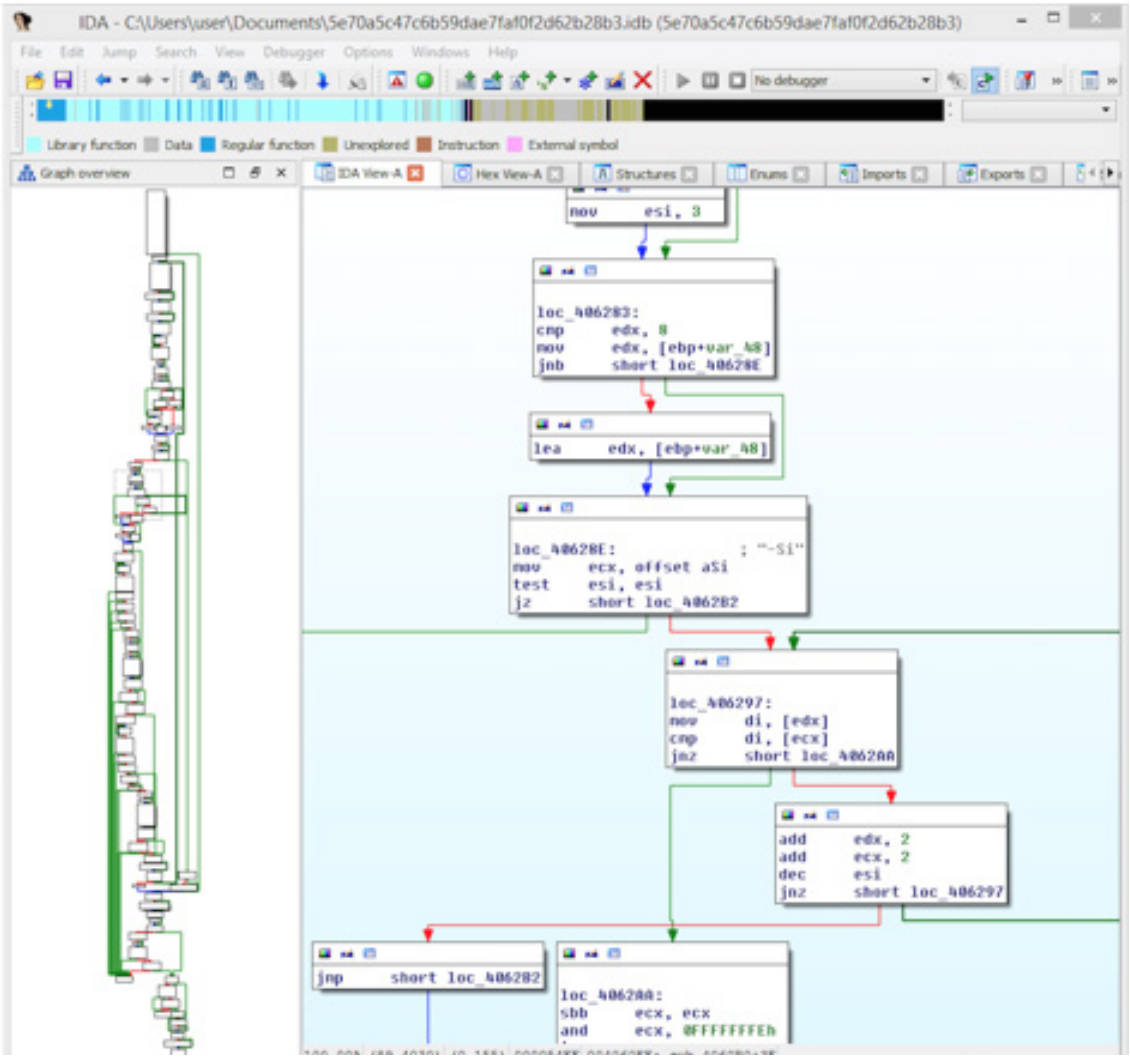
Figure 4: Command Packet Data

The malware only attempted to establish a connection to the Command and Control server (176.31.112.10) over SSL when a total of 4 arguments were supplied. In order to execute the DLL variant, “start” was used as the entry point for running with rundll32. Figure 5 demonstrates the command line parameters used to achieve execution.

```
c:\Users\user\Desktop>rundll32.exe malware.dll",start "-0 -1 -2 -3"
```

Figure 5: Execution Command

Sample 1 had plaintext flags in the code, which were visible within IDA directly after a call to GetCommandLineW. Whereas, Sample 2 had a much more obfuscated method for flag retrieval and argument parsing. This makes recovering what those switches are much more difficult.



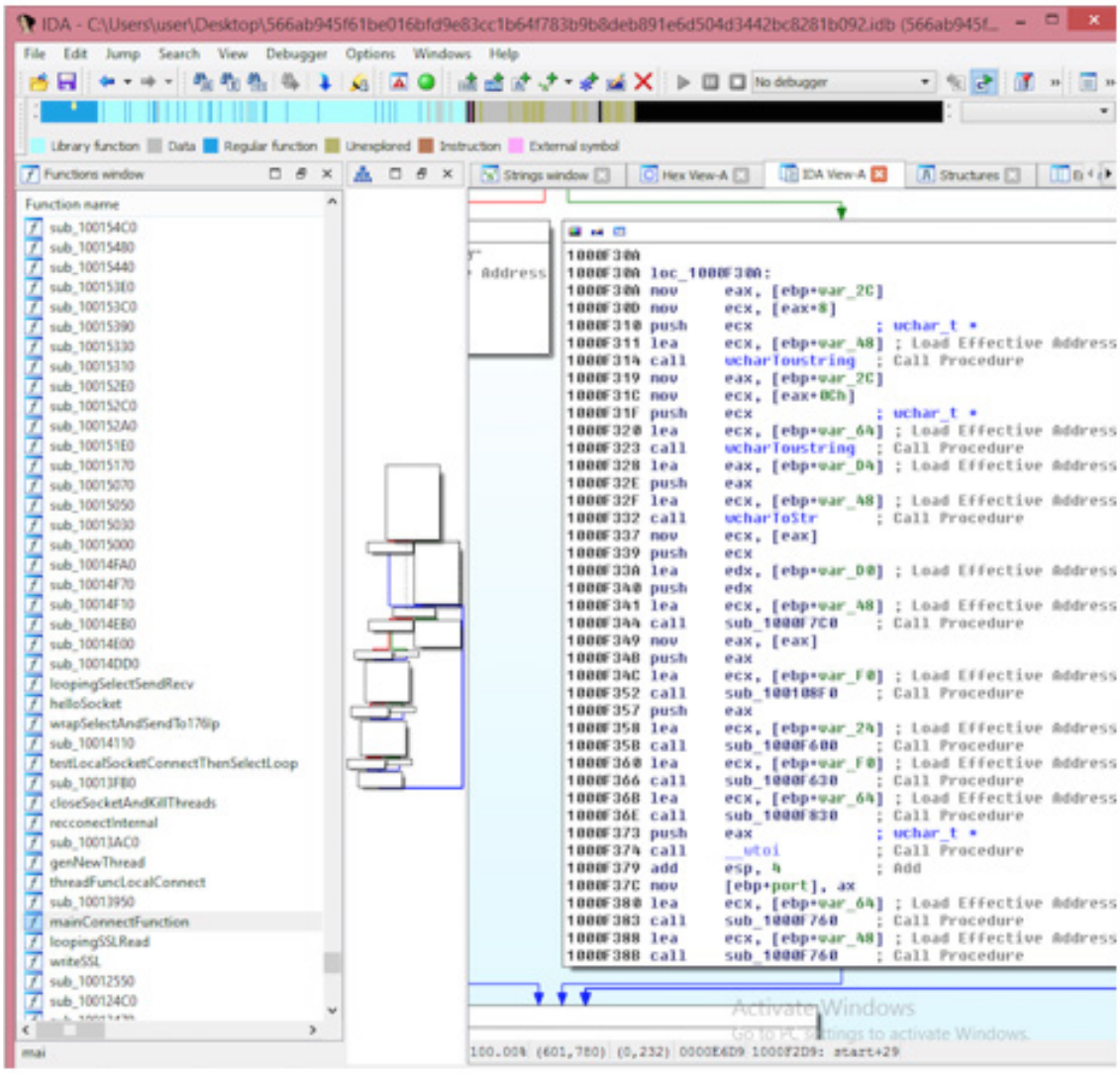


Figure 6: Comparison of IDA Output Netpolitik sample (left) and root9B sample (right)

DETECTION

The YARA signature from Claudio Guarnieri has been modified to include additional offsets to help identify specific binary data. Fortunately there is a sizeable function at the beginning of the text section, which has the same embedded items as Sample 2. In both artifacts, this function was placed at the very beginning of the .text section. Please note that while the effect remains the same, the assembly is slightly different (see figure 3 for a comparison). The following change to Claudio's rule should identify the similarity in both malware variants. The uint32 types in the YARA signature have accounted for endianness.

```
rule apt_sofacy_xtunnel_modified : sofacy openssl tunnel
{
   meta:
      author = "Claudio Guarnieri"

   strings:

      $variant11 = "XAPS_OBJECTIVE.dll" nocase
      $variant12 = "Xtunnel.exe" nocase
      $variant13 = "start" nocase
      $variant22 = "is you live?" nocase
      $mix1 = "176.31.112.10"
      $mix2 = "error inselect, errno% d" nocase
      $mix3 = "no MSG" nocase
      $mix4 = "is you live?" nocase
      $mix5 = "127.0.0.1"
      $mix6 = "err% d" nocase
      $mix7 = "i`m wait" nocase
      $mix8 = "hello" nocase
      $mix9 = "OpenSSL 1.0.1e 11 Feb 2013" nocase

   condition:
      ((uint16 (0) == 0x5A4D) or (uint16 (0) == 0xCFD0)) and ((uint32(0x41D)
== 0x85c6eff8 and uint32(0x421) == 0xFFFFC698) or ((uint32(0x422) ==
0x85c7effc and uint32(0x426) == 0xFFFF7a98) or
      (3 of ($variant*)) or (6 of ($mix*))))
}
```

CONCLUSION

The DLL sample 2 version of the malware analyzed and reported by root9B (1) in May 2015 has striking similarities to the EXE sample 1 version employed in the May 2015 German Parliament breach. For additional analysis and full documentation of the malware used in the German Parliament breach see Claudio Guarnieri's report. (2)

FOOTNOTES

(1)https://www.root9b.com/sites/default/files/whitepapers/R9b_FSOFACY_0.pdf

(2)https://netzpolitik.org/2015/digital-attack-on-german-parliament-investigative-report-on-the-hack-of-the-left-party-infrastructure-in-bundestag/