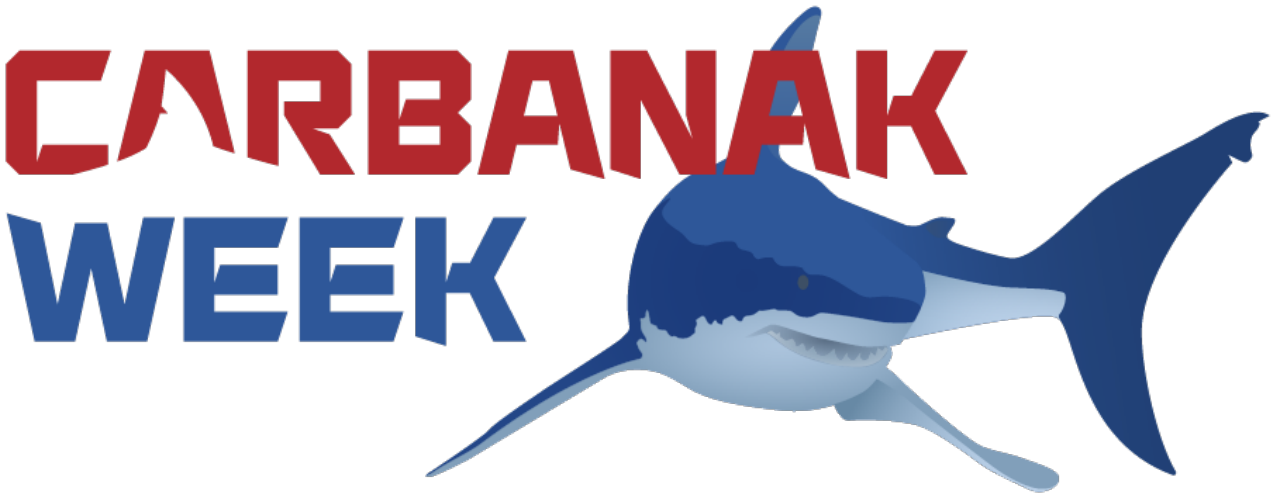
CARBANAK Week Part Two: Continuing the CARBANAK Source Code Analysis

Threat Research

April 23, 2019 | by Michael Bailey, James T. Bennett



Update (April 30): Following the release of our four-part CARBANAK Week blog series, many readers have found places to make the data shared in these posts actionable. We have updated this post to include some of this information.

In the previous installment, we wrote about how string hashing was used in CARBANAK to manage Windows API resolution throughout the entire codebase. But the authors used this same string hashing algorithm for another task as well. In this installment, we'll pick up where we left off and write about CARBANAK's antivirus (AV) detection, AV evasion, authorship artifacts, exploits, secrets, and network-based indicators.

Antivirus Evasions

Source code unquestionably accelerates analysis of string hashes. For example, the function AVDetect in AV.cpp iterates processes to detect AV by process name hash as shown in Figure 1.

```
switch( hash )
{
    case 0x0fc4e7c5: //sfctlcom.exe
    case 0x0946e915: //protoolbarupdate.exe
    case 0x06810b75: //tmproxy.exe
    case 0x06da37a5: //tmpfw.exe
    case 0x0ae475a5: //tmbmsrv.exe
    case 0x0becd795: //ufseagnt.exe
    case 0x0b97d795: //uiseagnt.exe
    case 0x08cdf1a5: //ufnavi.exe
    case 0x0b82e2c5: //uiwatchdog.exe
        return AV_TrandMicro;
    case 0x0d802425: //avgam.exe
    case 0x0f579645: //avgcsrvx.exe
    case 0x0e048135: //avgfws9.exe
    case 0x0c34c035: //avgemc.exe
    case 0x09adc005: //avgrsx.exe
    case 0x0c579515: //avgchsvx.exe
    case 0x08e48255: //avgtray.exe
    case 0x0ebc2425: //avgui.exe
        return AV_AVG;
    case 0x08d34c85: //avp.exe
    case 0x07bc2435: //avpui.exe
        return AV_KAV;

}
```

Figure 1: Antivirus detection by process name hash

What does CARBANAK do with this information? It evades AV according to what is installed. Figure 2 shows the code for an AVG evasion that the authors disabled by commenting it out. Based on this, it appears as if the AVG evasion was retired, but FLARE team member Ryan Warns confirmed in November 2017 that it still worked with one minor tweak. FLARE disclosed this to AVG immediately upon confirming it. Avast indicates that after our disclosure, they updated the affected DLL to ignore DLL_PROCESS_DETACH and leave its hooks in place.

```
/*
bool AVGUnload()
{
    typedef BOOL(WINAPI *typeDllMain)( _In_ HINSTANCE hinstDLL,_In_ DWORD fdwReason, _In_ LPVOID lpvReserved );
    typeDllMain DllMain;
    HINSTANCE dllAvg = API(KERNEL32, GetModuleHandleA)( _CS_("avghookx.dll") );
    if( dllAvg )
    {
        DbgMsg( "Обнаружен AVG (avghookx.dll)" );
        PIMAGE_NT_HEADERS ntHeaders = PE::GetNTHeaders(dllAvg);
        DWORD addrEP = (DWORD)dllAvg + ntHeaders->OptionalHeader.AddressOfEntryPoint;
        DllMain = (typeDllMain)addrEP;
        DllMain( dllAvg, DLL_PROCESS_DETACH, NULL ); //снятие хуков AVG
        return true;
    }
    else
        DbgMsg( "AVG не обнаружен" );
    return false;
}
*/
```

Figure 2: Commented out source code to unload AVG user-space hooks

In November of 2017, FLARE also disclosed an evasion for Trend Micro's detection of process injection that remained active in the CARBANAK source code. The evasion mirrors a technique used in Carberp that replaces remote heap allocation and a call to CreateRemoteThread with memory mapping and queueing of an asynchronous procedure call via QueueUserAPC. Following our disclosure, Trend Micro indicated that they had updated their behavior monitoring rules and released OfficeScan XG SP1 in December 2017 with a new "Aggressive Event" detection feature that covers this behavior.

Author Characterization

Having source code could pose unique opportunities to learn about the individuals behind the keyboard. To that end, I searched for artifacts in the source code dump that might point to individuals. I found the most information in Visual Studio solution files. Most of these referenced drive O: as the source root, but I did find the following host paths:

- C:\Users\hakurei reimu\AppData\Local\Temp
- C:\Users\Igor\AppData\Local\Temp
- E:\Projects\progs\Petrosjan\WndRec\...
- E:\Projects\progs\sbu\WndRec\...

Unfortunately, these data points don't yield many answers. If they are observed in later artifacts, connections might be inferred, but as of this writing, not much else is known about the authors.

Source Code Survey

The CARBANAK source code contained numerous exploits, previous C2 hosts, passwords, and key material. I decided to comprehensively search these out and determine if they led to any new conclusions or corroborated any previous observations.

Exploits

I wanted to know if the CARBANAK authors wielded any exploits that were not publicly disclosed. To the contrary, I found all the exploits to be well-documented. Table 1 breaks out the escalation code I reviewed from the CARBANAK source code dump.

Name	CVE	Notes
PathRec	2013-3660	Exploit proof of concept (poc) from May 2013
Sdrop	2013-3660	Exploit poc from June 2013
NDProxy	2013-5065	NDProxy.sys exploit originally authored by secniu
UACBypass		UAC bypass by DLL hijacking found in Carberp
COM		UAC bypass by disabling elevation prompts and dialogs via the IFileOperation COM interface
CVE-2014-4113	2014-4113	Win32k.sys exploit derived from code that can be found online
BlackEnergy2		AppCompat shim-based UAC bypass
EUDC	2010-4398	UAC bypass by EUDC exploitation

Table 1: Exploits for elevation found in CARBANAK source code

The CARBANAK source code also contains code copied wholesale from Mimikatz including the sekurlsa module for dumping passwords from lsass.exe and Terminal Services patching code to allow multiple remote desktop protocol connections.

Secrets

My analysis included an audit of passwords and key material found in the source code and accompanying binaries. Although many of these were used for debug versions, I curated them for reference in case a need might arise to guess future passwords based on passwords used in the source code. Table 2 shows recovered passwords used for RC2-

encrypted communications and other purposes along with the corresponding name in the source code and their status as they were encountered (active in source code, commented out, or compiled into a binary).

Credential Identifier Per Source Code	Password	Status
ADMIN_PASSWORD	1He9Psa7LzB1wiRn	Active
ADMIN_PASSWORD	1234567812345678	Commented out
ADMIN_PASSWORD	cbvhX3tJ0k8HwnMy	Active
ADMIN_PASSWORD	1234567812345678	Commented out
N/A	1234567812345678	Compiled

Table 2: Passwords found in CARBANAK source code and binaries

I found an encrypted server certificate in a debug directory. This seemed like it could provide a new network-based indicator to definitively tie operations together or catch new activity. It was trivial to brute force this container by adapting a publicly available code sample of X509 handling in C# to cycle through passwords in a popular password list. The password was found in less than 1 second because it was the single-character password “1”. The certificate turns out to be for testing, hence the weak password. The certificate is shown in Figure 3, with details in Table 3.

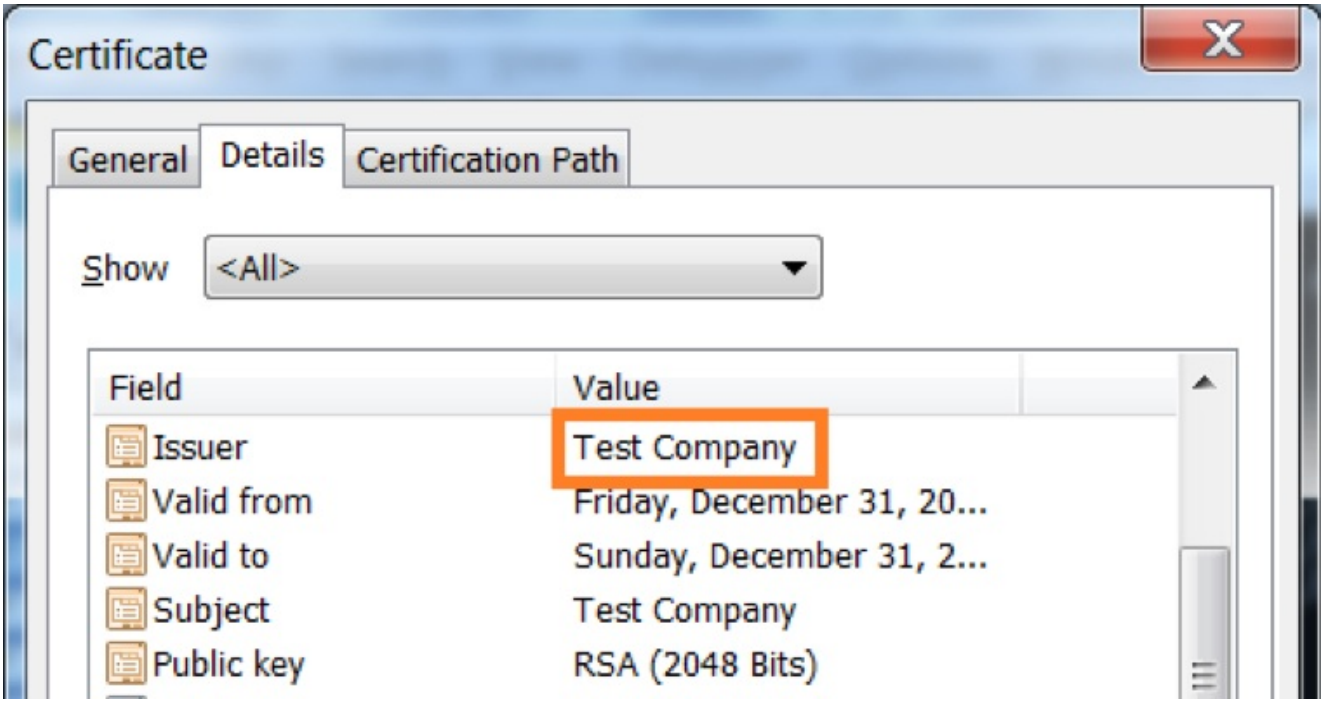


Figure 3: Test Company certificate

Parameter	**Value**
Subject	CN=Test Company
Issuer	CN=Test Company
Serial Number	834C6C3985506D8740FB56D26E385E8A
Not Before	12/31/2004 5:00:00 PM
Not After	12/31/2017 5:00:00 PM
Thumbprint	0BCBD1C184809164A9E83F308AD6FF4DBAFDA22C
Signature Algorithm	sha1RSA(1.3.14.3.2.29)

Public Key	Algorithm: RSA Length: 2048 Key Blob: 30 82 01 0a 02 82 01 01 00 e4 66 7f d2 e1 01 53 f9 6d 26 a6 62 45 8b a8 71 ea 81 9a e6 12 d4 1c 6f 78 67 6d 7e 95 bb 3a c5 c0 2c da ce 48 ca db 29 ab 10 c3 83 4e 51 01 76 29 56 53 65 32 64 f2 c7 84 96 0f b0 31 0b 09 a3 b9 12 63 09 be a8 4b 3b 21 f6 2e bf 0c c1 f3 e4 ed e2 19 6e ca 78 68 69 be 56 3c 1c 0e a7 78 c7 b8 34 75 29 a1 8d cc 5d e9 0d b3 95 39 02 13 8e 64 ed 2b 90 2c 3f d5 e3 e2 7e f2 d2 d1 96 15 6e c9 97 eb 97 b9 0e b3 be bc c3 1b 1e e1 0e 1c 35 73 f4 0f d9 c3 69 89 87 43 61 c9 9e 50 77 a2 83 e4 85 ce 5a d6 af 72 a9 7b 27 c5 f3 62 8d e7 79 92 c3 9b f7 96 ed 5c 37 48 0a 97 ee f7 76 69 a2 b9 25 38 06 25 7d 8a e4 94 b2 bb 28 4a 4b 5d c5 32 0d be 8e 7c 51 82 a7 9e d9 2c 8e 6b d8 c7 19 4c 2e 93 8d 2d 50 b4 e0 a4 ed c1 65 a4 a1 ba bf c7 bf 2c ec 28 83 f4 86 f2 88 5c c4 24 8b ce 1d 02 03 01 00 01 Parameters: 05 00

Private Key	Key Store: User Provider Name: Microsoft Strong Cryptographic Provider Provider type: 1 Key Spec: Exchange Key Container Name: c9d7c4a9-2745-4e7f-b816-8c20831d6dae Unique Key Container Name: 5158a0636a32ccdadf155686da582ccc_2bb69b91-e898-4d33-bbcf-fbae2b6309f1 Hardware Device: False Removable: False Protected: False

Table 3: Test Company certificate details

Here is a pivot shared by @mrdavi51 demonstrating how this self-signed certificate is still hosted on several IPs.

> Great findings, loving the series! Did you know the public cert in part two you found is still hosted on two servers? https://t.co/zZYRgPvHVr
>
> — mrdavi5 (@mrdavi51) April 24, 2019

FireEye has observed the certificate **most recently** being served on the following IPs (Table 4):

IP	Hostname	Last Seen
104.193.252.151:443	vds2.system-host[.]net	2019-04-26T14:49:12
185.180.196.35:443	customer.clientshostname[.]com	2019-04-24T07:44:30
213.227.155.8:443		2019-04-24T04:33:52
94.156.133.69:443		2018-11-15T10:27:07

185.174.172.241:443	vds9992.hyperhost[.]name	2019-04-27T13:24:36
109.230.199.227:443		2019-04-27T13:24:36

Table 4: Recent Test Company certificate use

While these IPs have not been observed in any CARBANAK activity, this may be an indication of a common developer or a shared toolkit used for testing various malware. Several of these IPs have been observed hosting Cobalt Strike BEACON payloads and METERPRETER listeners. Virtual Private Server (VPS) IPs may change hands frequently and additional malicious activity hosted on these IPs, even in close time proximity, may not be associated with the same users.

I also parsed an unprotected private key from the source code dump. Figure 4 and Table 5 show the private key parameters at a glance and in detail, respectively.

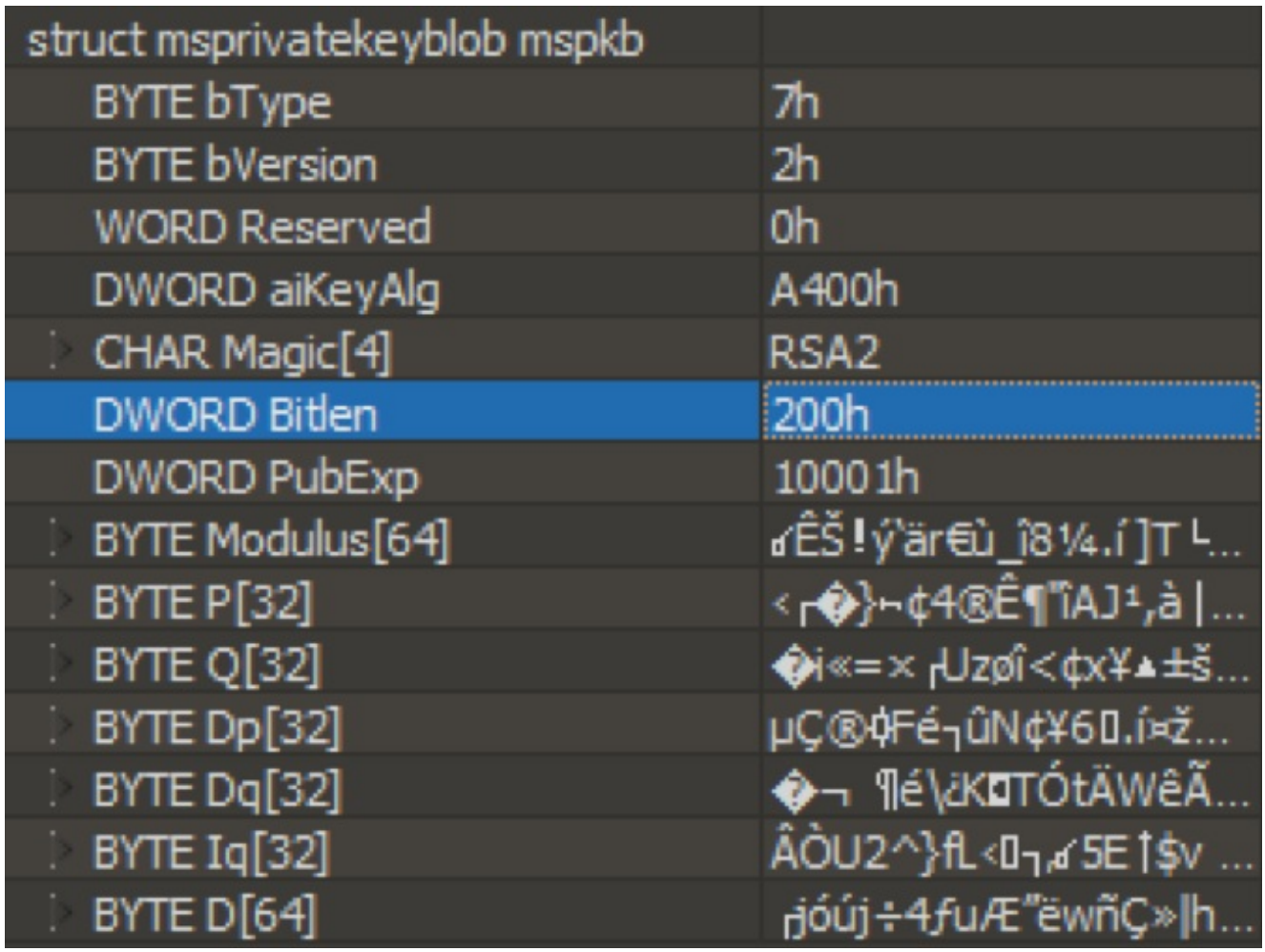


Figure 4: Parsed 512-bit private key

Field	Value
bType	7
bVersion	2
aiKeyAlg	0xA400 (CALG_RSA_KEYX) – RSA public key exchange algorithm
Magic	RSA2
Bitlen	512
PubExp	65537
Modulus	0B CA 8A 13 FD 91 E4 72 80 F9 5F EE 38 BC 2E ED 20 5D 54 03 02 AE D6 90 4B 6A 6F AE 7E 06 3E 8C EA A8 15 46 9F 3E 14 20 86 43 6F 87 BF AE 47 C8 57 F5 1F D0 B7 27 42 0E D1 51 37 65 16 E4 93 CB
P	8B 01 8F 7D 1D A2 34 AE CA B6 22 EE 41 4A B9 2C E0 05 FA D0 35 B2 BF 9C E6 7C 6E 65 AC AE 17 EA
Q	81 69 AB 3D D7 01 55 7A F8 EE 3C A2 78 A5 1E B1 9A 3B 83 EC 2F F1 F7 13 D8 1A B3 DE DF 24 A1 DE
Dp	B5 C7 AE 0F 46 E9 02 FB 4E A2 A5 36 7F 2E ED A4 9E 2B 0E 57 F3 DB 11 66 13 5E 01 94 13 34 10 CB
Dq	81 AC 0D 20 14 E9 5C BF 4B 08 54 D3 74 C4 57 EA C3 9D 66 C9 2E 0A 19 EA C1 A3 78 30 44 52 B2 9F

Iq	C2 D2 55 32 5E 7D 66 4C 8B 7F 02 82 0B 35 45 18 24 76 09 2B 56 71 C6 63 C4 C5 87 AD ED 51 DA 2ª
D	01 6A F3 FA 6A F7 34 83 75 C6 94 EB 77 F1 C7 BB 7C 68 28 70 4D FB 6A 67 03 AE E2 D8 8B E9 E8 E0 2A 0F FB 39 13 BD 1B 46 6A D9 98 EA A6 3E 63 A8 2F A3 BD B3 E5 D6 85 98 4D 1C 06 2A AD 76 07 49

Table 5: Private key parameters

I found a value named PUBLIC_KEY defined in a configuration header, with comments indicating it was for debugging purposes. The parsed values are shown in Table 6.

Field	**Value**
bType	6
bVersion	2
aiKeyAlg	0xA400 (CALG_RSA_KEYX) – RSA public key exchange algorithm
Magic	RSA1
Bitlen	512
PubExp	65537
Modulus	0B CA 8A 13 FD 91 E4 72 80 F9 5F EE 38 BC 2E ED 20 5D 54 03 02 AE D6 90 4B 6A 6F AE 7E 06 3E 8C EA A8 15 46 9F 3E 14 20 86 43 6F 87 BF AE 47 C8 57 F5 1F D0 B7 27 42 0E D1 51 37 65 16 E4 93 CB

Table 6: Key parameters for PUBLIC_KEY defined in configuration header

Network Based Indicators

The source code and binaries contained multiple Network-Based Indicators (NBIs) having significant overlap with CARBANAK backdoor activity and FIN7 operations previously observed and documented by FireEye. Table 7 shows these indicators along with the associated FireEye public documentation. This includes the status of each NBI as it was encountered (active in source code, commented out, or compiled into a binary). Domain names are de-fanged to prevent accidental resolution or interaction by browsers, chat clients, etc.

NBI	**Status**	**Threat Group Association**
comixed[.]org	Commented out	Earlier CARBANAK activity
194.146.180[.]40	Commented out	Earlier CARBANAK activity
aaaabbbbccccc[.]org	Active	
stats10-google[.]com	Commented out	FIN7
192.168.0[.]100:700	Active	
80.84.49[.]50:443	Commented out	
52.11.125[.]44:443	Commented out	
85.25.84[.]223	Commented out	
qwqreererwere[.]com	Active	

akamai-technologies[.]org	Commented out	Earlier CARBANAK activity
192.168.0[.]100:700	Active	
37.1.212[.]100:700	Commented out	
188.138.98[.]105:710	Commented out	Earlier CARBANAK activity
hhklhlkhkjhjkjk[.]org	Compiled	
192.168.0[.]100:700	Compiled	
aaa.stage.4463714.news.meteonovosti[.]info	Compiled	DNS infrastructure overlap with later FIN7 associated POWERSOURCE activity
193.203.48[.]23:800	Active	Earlier CARBANAK activity

Table 7: NBIs and prevously observed activity

Four of these TCP endpoints (80.84.49[.]50:443, 52.11.125[.]44:443, 85.25.84[.]223, and 37.1.212[.]100:700) were new to me, although some have been documented elsewhere.

Conclusion

Our analysis of this source code dump confirmed it was CARBANAK and turned up a few new and interesting data points. We were able to notify vendors about disclosures that specifically targeted their security suites. The previously documented NBIs, Windows API function resolution, backdoor command hash values, usage of Windows cabinet file APIs, and other artifacts associated with CARBANAK all match, and as they say, if the shoe fits, wear it. Interestingly though, the project itself isn't called CARBANAK or even Anunak as the information security community has come to call it based on the string artifacts found within the malware. The authors mainly refer to the malware as "bot" in the Visual Studio project, filenames, source code comments, output binaries, user interfaces, and manuals.

The breadth and depth of this analysis was a departure from the usual requests we receive

on the FLARE team. The journey included learning some Russian, searching through a hundred thousand of lines of code for new information, and analyzing a few dozen binaries. In the end, I'm thankful I had the opportunity to take this request.

In the next post, Tom Bennett takes the reins to provide a retrospective on his and Barry Vengerik's previous analysis in light of the source code. Part Four of CARBANAK Week is available as well.

Previous Post
Next Post