It's a BEE! It's a... no, it's ShadowPad.

medium.com/insomniacs/its-a-bee-it-s-a-no-it-s-shadowpad-aff6a970a1c2

asuna amawaka

November 19, 2021



Nov 19, 2021

8 min read

ShadowPad, a malware name that is familiar to many, became widely known since 2017 through its participation in a series of supply-chain attacks in CCleaner, NetSarang and ASUS. There has been a lot of good work describing how clusters of activities by different threat actors (APT41, Fishmonger, Tonto Team, RedFoxTrot, BackdoorDiplomacy are all the "big names" we know) are linked by ShadowPad. I thought to join in the fun of reverse engineering one of the variant of ShadowPad that I found from VirusTotal. So let's begin!

Side-note: Kaspersky calls this variant "ShadowShredder" [1]. This variant has also been documented by PTSecurity [2] in Jan 2021 and was also recently in TeamT5's presentation in VB2021localhost conference [3].

The malware trinity (DLL load-order attack) analyzed in this post:



Legitimate EXE, bdreinit.exe (Bitdefender's Crash Handler) — SHA256: 386EB7AA33C76CE671D6685F79512597F1FAB28EA46C8EC7D89E58340081E2BD

Malicious DLL, log.dll — SHA256: 8D1A5381492FE175C3C8263B6B81FD99AACE9E2506881903D502336A55352FEF

Encrypted Payload, log.dll.dat — SHA256: 0371FC2A7CC73665971335FC23F38DF2C82558961AD9FC2E984648C9415D8C4E I found these files separately, so they may not be originally intended as a package. I managed to piece them together based on their filenames, and they work fine as a set. These files happened to have debugging strings included, which makes the analysis slightly more pleasing to follow.

Let's start with observations from dynamic analysis.

Here's what the log looked like in the debugger upon execution without breakpoints:

DLL Loaded: <u>6FA80000</u> C:\exe\log.dll
DebugString: "BEE[2021-11-18 14:46:17] bdreinit.exe [P:1432,T:2684] ScLoad::path_exe: C:\exe\bdreinit.exe"
DebugString: "BEE[2021-11-18 14:46:17] bdreinit.exe [P:1432,T:2684] ScLoad::path_dll: C:\exe\log.dll"
DebugString: "BEE(2021-11-18 14:46:17) bdreinit.exe (P:1432,T:2684) ScLoad::path_img: C:\exe\log.dll.dat"
DebugString: "BEE[2021-11-18 14:46:17] bdreinit.exe [P:1432,T:2684] ScLoad:KEY(SOFTWARE\Classes\CLSID\{ <u>e3f825af</u> -f27f-5b95- <u>50b2c77deeac30cf</u> }:D572770E)"
DebugString: "BEE[2021-11-18 14:46:19] bdreinit.exe [P:1432,T:2684] ScLoad::ScQueryFromFile() OK"
DebugString: "BEE[2021-11-18 14:46:19] bdreinit.exe [P:1432,T:2684] ScLoad:shellcode()"
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] Inst[0]InstDir: %ALLUSERSPROFILE%\DRM\Test\"
DebugString: "BEE(2021-11-18 14:46:21) bdreinit.exe (P:1432,T:2684) Inst[1]InstExe: Test.exe"
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] Inst[2]InstDll: log.dll"
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] Inst[4]SvcName: MyTest"
DebugString: "BEE(2021-11-18 14:46:21) bdreinit.exe [P:1432,T:2684] Inst[5]SvcDisp: MyTest"
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] Inst[6]SvcDesc: MyTest"
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] Inst[7]KeyPath: SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
DebugString: "BEE(2021-11-18 14:46:21) bdreinit.exe [P:1432,T:2684] Inst[8]KeyName: MyTest"
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] Inst::CopyFileW(C:\exe\bdreinit.exe,C:\ProgramData\DRM\Test\Test.exe)"
DebugString: "BEE(2021-11-18 14:46:21) bdreinit.exe (P:1432,T:2684) DBG_ASSERT(XInstall.cpp:484)
[32]:The process cannot access the file because it is being used by another process."
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] DBG_ASSERT(Sc.cpp:225)
[32]:The process cannot access the file because it is being used by another process."
DebugString: "BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684] SID: S-1-5-21- <u>38464580</u> 15- <u>21202993</u> 29- <u>62090280</u> 9-1000"
DLL Loaded: 70980000 C:\Windows\SysW0W64\apphelp.dll
DebugString: "BEE[2021-11-18 14:46:22] bdreinit.exe [P:1432,T:2684] CreateProcessAsUserW(pid:2748) ok"
DebugString: "BEE[2021-11-18 14:46:22] bdreinit.exe [P:1432,T:2684] CXInject(pid:2748,path:C:\Program Files (x86)\Windows Media Player\wmplayer.exe) OK!"
DebugString: "BEE(2021-11-18 14:46:22) bdreinit.exe (P:1432,T:2684) Inject (C:\Program Files (x86)\Windows Media Player\wmplayer.exe) OK!"
Process stopped with exit code 0x0
Saving database to C:\Program Files\x64dbg\release\x32\db\bdreinit.exe.dd32 0ms
Debugging stopped!

logged by x32dbg

The same debugging information is also written into a file C:\ProgramData\bee.log.

🔚 be	elog 🖸	
1	BEE[2021-11-18 14:46:17] bdreinit.exe [P:1432,T:2684]	ScLoad::path_exe: C:\exe\bdreinit.exe
2	BEE[2021-11-18 14:46:17] bdreinit.exe [P:1432,T:2684]	ScLoad::path_dll: C:\exe\log.dll
3	BEE[2021-11-18 14:46:17] bdreinit.exe [P:1432,T:2684]	ScLoad::path_img: C:\exe\log.dll.dat
- 4	BEE[2021-11-18 14:46:17] bdreinit.exe [P:1432,T:2684]	ScLoad:KEY(SOFTWARE\Classes\CLSID\{e3f825af-f27f-5b95-50b2c77deeac30cf}:D572770E)
5	BEE[2021-11-18 14:46:19] bdreinit.exe [P:1432,T:2684]	ScLoad::ScQueryFromFile() OK
6	BEE[2021-11-18 14:46:19] bdreinit.exe [P:1432,T:2684	ScLoad:shellcode()
7	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst[0]InstDir: %ALLUSERSPROFILE%\DRM\Test\
8	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst[1]InstExe: Test.exe
9	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst[2]InstDll: log.dll
10	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst[4]SvcName: MyTest
11	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst[5]SvcDisp: MyTest
12	<pre>BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]</pre>	Inst[6]SvcDesc: MyTest
13	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst[7]KeyPath: SOFTWARE\Microsoft\Windows\CurrentVersion\Run
14	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst[8]KeyName: MyTest
15	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	Inst::CopyFileW(C:\exe\bdreinit.exe,C:\ProgramData\DRM\Test\Test.exe)
16	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	DBG_ASSERT(XInstall.cpp:484)[32]:The process cannot access the file because it is being used by another process.
17	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	DBG_ASSERT(Sc.cpp:225)[32]:The process cannot access the file because it is being used by another process.
18	BEE[2021-11-18 14:46:21] bdreinit.exe [P:1432,T:2684]	SID: S-1-5-21-3846458015-2120299329-620902809-1000
19	BEE[2021-11-18 14:46:22] bdreinit.exe [P:1432,T:2684]	CreateProcessAsUserW(pid:2748) ok
20	BEE[2021-11-18 14:46:22] bdreinit.exe [P:1432,T:2684]	CXInject(pid:2748,path:C:\Program Files (x86)\Windows Media Player\wmplayer.exe) OK!
21	BEE[2021-11-18 14:46:22] bdreinit.exe [P:1432,T:2684]	Inject [C:\Program Files (x86)\Windows Media Player\wmplayer.exe] OK!
22	BEE[2021-11-18 14:46:24] wmplayer.exe [P:2748,T:1268]	OL:MUTEX: Global\PQMIOGCCUOYOYGWYCGMYWCKKK
23	BEE[2021-11-18 14:46:30] wmplayer.exe [P:2748,T:1268]	SID: S-1-5-21-3846458015-2120299329-620902809-1000
24	BEE[2021-11-18 14:46:30] wmplayer.exe [P:2748,T:1652]	CXMgrScreenLog::LogProcEx()
25	BEE[2021-11-18 14:46:34] wmplayer.exe [P:2748,T:1268]	Online(0) = TCP://ti0wddsnv.wikimedia.vip:443
26	BEE[2021-11-18 14:46:35] wmplayer.exe [P:2748,T:1268]	OnlineEx([ti0wddsnv.wikimedia.vip] : 443 : 200, [] : 0 : 0, p: 0)
27		

bee.log

🔚 bee.	log 🖾					
48	BEE[2021-11-18	16:23:32]	wmplayer.exe	[P:2772,T:2320]	OL:MUTEX: Global\PQMIOGCCUOYOYGWYCGMYWCKKK
49	BEE[2021-11-18	16:23:34]	wmplayer.exe	[P:2772,T:2320]	SID: S-1-5-21-3846458015-2120299329-620902809-1000
50	BEE[2021-11-18	16:23:34]	wmplayer.exe	[P:2772,T:2988]	CXMgrScreenLog::LogProcEx()
51	BEE[2021-11-18	16:23:38]	wmplayer.exe	[P:2772,T:2320]	<pre>Online(0) = TCP://ti0wddsnv.wikimedia.vip:443</pre>
52	BEE[2021-11-18	16:23:38]	wmplayer.exe	[P:2772,T:2320]	OnlineEx([ti0wddsnv.wikimedia.vip] : 443 : 200, [] : 0 : 0, p: 0)
53	BEE[2021-11-18	16:23:39]	wmplayer.exe	[P:2772,T:2320]	OnlineEx() Q4=10054
54	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(1) =
55	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	DBG_ASSERT(XOnline.cpp:106)[12006]:*
56	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(2) =
57	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	DBG_ASSERT(XOnline.cpp:106)[12006]:*
58	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(3) =
59	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	DBG_ASSERT(XOnline.cpp:106)[12006]:*
60	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(4) = NULL
61	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(5) = NULL
62	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(6) = NULL
63	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(7) = NULL
64	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(8) = NULL
65	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(9) = NULL
66	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(10) = NULL
67	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(11) = NULL
68	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(12) = NULL
69	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(13) = NULL
70	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(14) = NULL
71	BEE[2021-11-18	16:23:40]	wmplayer.exe	[P:2772,T:2320]	Online(15) = NULL
72	BEE[2021-11-18	16:23:42]	wmplayer.exe	[P:2772,T:2320]	Online(0) = <u>TCP://ti0wddsnv.wikimedia.vip:443</u>
73	BEE[2021-11-18	16:23:42]	wmplayer.exe	[P:2772,T:2320]	OnlineEx([ti0wddsnv.wikimedia.vip] : 443 : 200, [] : 0 : 0, p: 0)
74	BEE	2021-11-18	16:23:42]	wmplayer.exe	[P:2772,T:2320]	OnlineEx() Q4=10054
75	BEE[2021-11-18	16:23:43]	wmplayer.exe	[P:2772,T:2320]	Online(1) =
76	BEE	2021-11-18	16:23:43]	wmplayer.exe	[P:2772,T:2320]	DBG_ASSERT(XOnline.cpp:106)[12006]:*
77	BEE[2021-11-18	16:23:43]	wmplayer.exe	[P:2772,T:2320]	Online(2) =
78	BEE[2021-11-18	16:23:43]	wmplayer.exe	[P:2772,T:2320]	DBG_ASSERT(XOnline.cpp:106)[12006]:*
79	BEE[2021-11-18	16:23:43]	wmplayer.exe	[P:2772,T:2320]	Online(3) =
80	BEE[2021-11-18	16:23:43]	wmplayer.exe	[P:2772,T:2320]	DBG_ASSERT(XOnline.cpp:106)[12006]:*

more of bee.log

It seems that I've picked up a test sample. The C2 domain configured within is a subdomain of wikimedia[.]vip, which has been associated with Funnydll and other ShadowPad samples.

I'm curious about the author's choice of name — What does "BEE" stand for?

I extracted all the interesting filename strings, for reference if anyone is interested at guessing the full suite of capabilities within ShadowPad:

XMgrService.cpp XMqrScreen.cpp XMgrScreenLog.cpp XMgrShell.cpp XMqrProcess.cpp XMgrDisk.cpp XMgrKeyLogger.cpp XMgrRegister.cpp XMgrPortMap.cpp XMgrRecentFiles.cpp XSo.cpp XSoClass.cpp XSoTcp.cpp XSoUDP.cpp XSoRTP.cpp XSoPipe.cpp XJoinSvr.cpp XJoin.cpp XEveryone.cpp XHandle.cpp XService.cpp XInterface.cpp XDebug.cpp XOnline.cpp XFireWall.cpp XImpUserService.cpp XImpUser.cpp XInstall.cpp XInject.cpp XStream.cpp XStreamFile.cpp XProxy.cpp XPacket.cpp XPktMap.cpp XConfig.cpp XString.cpp XDIBBitmap.cpp Sc.cpp User.cpp

source filenames

And here's the list of debugging messages found in memory:

CXInstall::Uninst():delete file2: %s CXInstall::Uninst():delete file1: %s CXInstall::InstDeleteCallback():delete file2: %s CXInstall::InstDeleteCallback():delete file1: %s UninstSvc(State:%d) wait OnlineEx([%s] : %d : %d, [%s] : %d : %d, p: %d) OnlineEx() Q0=%d OnlineEx() Q1=%d OnlineEx() Q2=%d OnlineEx() O3=%d OnlineEx() Q4=%d OnlineEx() Q7 OnlineEx() Q8 Online(%d) = %sOnline(%d) = NULLInst::CopyFileW(%s,%s) Inst[8]KeyName: %s Inst[7]KeyPath: %s Inst[6]SvcDesc: %s Inst[5]SvcDisp: %s Inst[4]SvcName: %s Inst[2]InstDll: %s Inst[1]InstExe: %s Inst[0]InstDir: %s SoShutdown(soType:%d): ERROR NOT SUPPORTED SoCreate (id:%d): ERROR NOT SUPPORTED SoConnect(soType:%d): ERROR NOT SUPPORTED SoSend(soType:%d): ERROR NOT SUPPORTED SoRecv(soType:%d): ERROR NOT SUPPORTED SoClose (soType:%d): ERROR NOT SUPPORTED DoPacket() Unknow Cmd: %8.8X recvfrom() = %d CXSoRTP::OnRecv(pkt->wCd=%4.4X)... CXSoRTP::OnRecvData()-EXPROID!!! OnRecvSyncAck()-EXPROID!!! ImpUserSession::0... ImpUserSession::1... ImpUserSession::2... ImpUserSession::WaitForSingleObject(pid:%d, exit:%d(%8.8X)) ok ImpUserSession::ImpUserCreateProcess(%d) ImpUserService(%s) quit... ImpUserService(%s) start... ImpUserSession::session changed (%d:%d)... ImpUserSessionUser::WaitForSingleObject(pid:%d, exit:%d(%8.8X)) ok ImpUser(%s)... x:%d,y:%d,flaq:%d Inject [%s] ERROR: %d Inject [%s] OK! XMgrScreenLog::LogProcEx()... XMgrScreenLog::LogProcEx() quit... CreateProcessAsUserW(pid:%d) ok QueryPLP(): NOT FOUND PLP OL:MUTEX: %s

debugging messages The processes created are as such:

Monitored Processes	-	ņ	x
🗟 - 🗔 🖂 🕸 📭 🚰 📋			
E. C:\exe\bdreinit.exe - PID: 2168 - (Terminated)			
C:\ProgramData\DRM\Test\Test.exe - PID: 316 - (Terminated)			
C:\Program Files (x86)\Windows Media Player\wmplayer.exe - PID:	500)	
C:\Windows\SysWOW64\dllhost.exe - PID: 888			
C:\Windows\SysWOW64\dllhost.exe - PID: 1588			

logged with APIMonitor

"Test.exe" is a copy of bdreinit.exe and is part of the persistency mechanism of the malware — The EXE and DLL are started via Services, and the encrypted payload is loaded from Registry. I supposed if service installation fails, then the malware shall persist through Run regkey.

MyTest Properties (Local Computer)									
General Log On	Recovery Dependencies									
Service name:	MyTest									
Display name:	MyTest									
Description:	MyTest									
Path to executable: C:\ProgramData\DRM\Test\Test.exe										
Startup type:	Automatic									
Help me configure	service startup options.									
Service status:	Stopped									
Start	Stop Pause Resume									
You can specify the from here.	ne start parameters that apply when you start the service									
Start parameters:										
	OK Cancel Apply									

The EXE and DLL are copied to C:\ProgramData\DRM\Test (specified by configuration), while the DAT payload file is deleted after the first execution. For subsequent executions, the payload is read from Registry.

The payload is re-encrypted by the malware before it is being written into the Registry for persistency. For some reasons, the initialization value used in the re-encryption algorithm is the compilation timestamp of the malicious DLL file, while the initialization value used in the original encryption algorithm is the first 4 bytes within the dat file, hence the encrypted data seen in the registry differs from the log.dll.dat file.

Registry Editor	11 mm 40 10 1mm		2010 Y 2014 CR P 4 1 1 1 2 2 2 2					
File Edit View Favorites Help								
> - 📙 {E3D5D93C-1663-4A78-A1A7-22375DFEBAEI 🔺	Name	Туре	Data					
E3E1D967-0829-48AC-B3AD-C5AE4CA171C	(Default)	REG_SZ	(value not set)					
e3e478d6-a2f2-4791-89a3-21f5c78dc3ec}	88 D572770E	REG_BINARY	23 f0 a3 d4 93 b3 00 46 09 6f 89 91 92 53 aa e8 dc f5 03 72 31 a6 1d 72 55 26 b2 a2 f4 e					
E413D040-6788-4C22-957E-175D1C513A34								
E4206432-01A1-4BEE-B3E1-3702C8EDC574								
▶ - 1 {E423AF7C-FC2D-11d2-B126-00805FC73204 ▼								
< III >								
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Class	es\Wow6432Node\CL	SID\{e3f825af-f27f-5b9	5-50b2c77deeac30cf}					
(E3ELD967-0829-48AC-B3AD-C5AE4CA171C (e3e478d6-a2f2-4791-89a3-21f5c78dc3ec) (e3f825af-f27f-5b95-50b2c77deac30cf) (e3f825af-f27f-5b95-50b2c77deac30cf) (e3f825af-f27f-5b95-50b2c77deac30cf) (e3f825af-f27f-5b95-50b2c77deac30cf) (e3f825af-f27f-5b95-50b2c77deac30cf) (e3f825af-f27f-5b95-50b2c77deac30cf) (e3f825af-f27f-5b95-50b2c77deac30cf) (e4206432-01A1-4BEE-B3E1-3702C8EDC574) (E423AF7C-FC2D-11d2-B126-00805FC73204 * (computer\HKEY_LOCAL_MACHINE\SOFTWARE\Class	(Default) D572770E res\Wow6432Node\CL:	REG_SZ REG_BINARY SID\{e3f825af-f27f-5b9	(value not set) 23 f0 a3 d4 93 b3 00 46 09 6f 89 91 92 53 aa e8 dc f5 03 72 31 a6 1d 72 55 26 b2 a2 f4 e 5-50b2c77deeac30cf}					

encrypted payload in registry

log.dll.d	lat X																
¥ Ed	it As: H	lex∨	·	Run S	cript	\sim	Rur	1 Tem	plate	~							
	ò	1	2	3	4	5	é	?	8	9	A	B	ċ	Ď	Ę	F	0123456789ABCDEF
0000h:	89	97	96	84	B1	54	87	29	21	55	57	83	04	77	D9	2C	‱—— <i>"</i> ±T‡)!UWf.wÙ,
0010h:	63	ЗB	04	AC	57	A 7	2E	B1	0B	BC	78	AF	4F	6A	87	D3	c;.¬₩§.±.Կx¯Oj‡Ó
0020h:	83	ED	AE	5F	0D	25	FO	F5	B2	7F	60	81	F3	DO	C9	97	fí⊗%ðõ°.`.óÐÉ—
0030h:	41	58	53	24	38	FE	67	66	4A	43	AC	E0	C7	F6	DC	FA	AXS ^{\$} 8þgfJC¬àÇöÜú
0040h:	25	EB	53	E8	A 5	DC	F7	01	96	2A	15	1A	FD	25	B1	FC	%ëSè¥Ü÷*ý%±ü
0050h:	5F	87	B7	09	96	BF	4D	D6	C1	AF	95	92	1F	0B	C0	1A	_ + ·¿MÖA •′À.
0060h:	32	E8	7F	46	5B	C4	10	89	5D	E7	A1	11	B 3	28	3F	80	2è.F[Ä.‰]ç;.³(?€
0070h:	73	75	77	73	23	20	F6	8E	9C	05	67	FB	ЗE	31	BF	F3	suws# öŽœ.gû>1¿ó
0080h:	D5	DC	03	23	E4	0F	AF	66	4C	EE	E7	95	24	5F	C1	CA	ŐÜ.#ä. fLîç•\$ ÁÊ

encrypted payload file

Here is the python script I used to decrypt the payload file log.dll.dat:

SHADOWPAD-analysis/decrypt_payload_dat.py at main · asunaamawaka/SHADOWPAD-analysis

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below...

github.com

And here is the script for decrypting the payload from registry:

<u>SHADOWPAD-analysis/decrypt_payload_reg.py at main · asuna-</u> <u>amawaka/SHADOWPAD-analysis</u>

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below...

<u>github.com</u>

The malware has keystroke logging capabilities, and the keystroke log file is encrypted and saved to a random-looking filepath in %PROGRAMDATA% like this:

C:\ProgramData\MOO\JKISIQ\EWGSE\LECIIOGK

Decryption of the keystroke log file is done in the same manner as the configuration data (I'll talk about how to get this in awhile):

SHADOWPAD-analysis/decrypt_keystroke_log.py at main · asunaamawaka/SHADOWPAD-analysis

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

Anti-reverse engineering obfuscation

Now, here comes details on how I handled the reverse engineering. This is slightly harder than usual, because of the obfuscation technique used in the malware. The original malware binary is "shredded" into pieces, with 1 instruction per piece, and put back together with a "jumper" function as glue. Oh, and there are some junk "cmp" followed by "jb" instructions just to make your eyes hurt.

Let's see what it looks like in IDA.

Up till this point at address 0x100011D6, everything is normal.



disassembly at beginning of malware logic Then came this jmp:

📕 🛃 🖾	
.text:10001738	
.text:10001738	
.text:10001738	; Attributes: thunk
.text:10001738	
.text:10001738	sub_10001738 proc near
.text:10001738 E9 8F 66 00	0 00 jmp <mark>loc_10007DCC</mark>
.text:10001738	sub_10001738 endp
.text:10001738	

Followed by another call:

.text:10007DCC	
.text:10007DCC	
.text:10007DCC	loc_10007DCC: ; CODE XREF: sub_10001738†j
.text:10007DCC E8 94 C6 FF FF	call jumper_10004465
.text:10007DCC	
.text:10007DD1 46	db 46h ; F
.text:10007DD2 5A	db 5Ah; Z
.text:10007DD3 00	db 0
.text:10007DD4 00	db 0

If we follow the instructions starting from 0x10004465, you will see something like this:

0x10004465	xchg dword ptr	ss:[esp], eax
0x10004468	jb 0x10012C6F	
0x1000446E	nop	
0x1000446F	xchg dx, dx	
0x10004472	jae 0x10012C6F	
0x10012C6F	pushfd	
0x10012C70	je 0x1000F507	
0x10012C76	jne 0x1000F507	
0x1000F507	add eax, dword	ptr ds:[eax]
0x1000F509	jno 0x1000AD76	
0x1000AD76	popfd	
0x1000AD77	jae 0x10012C5E	
0x10012C5E	xchg dword ptr	ss:[esp], eax
0x10012C61	jbe 0x10006EC3	
0x10012C67	ja 0x10006EC3	
0x10006EC3	ret	

within "jumper" function

It looks terribly complicated, but all it does is to read the next dword after the call and add it to original intended return address.



illustrate jumper (1)

And these "bits and pieces" of instructions occur from here onwards, throughout the whole malware. The "real" instruction that is part of the malware's logic is the single instruction before the call to jumper. Trying to recover these instructions makes me feel like I am picking

up the pieces from the shredding machine and gluing them back.

.text:1000D817	
.text:1000D817 55	push ebp
.text:1000D818 E8 48 6C FF FF	call jumper_10004465 ; 0x10012766
.text:1000D818	
.text:1000D81D 49	db 49h ; I
.text:1000D81E 4F	db 4Fh ; 0
.text:1000D81F 00	db 0
.text:1000D820 00	$db 0 \qquad 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\$
illustrate jumper (2)	
.text:10012766	;
.text:10012766 88 EC	mov ebp, esp
.text:10012768 E8 F8 1C FF FF	call jumper_10004465 ; 0x1000e876
.text:10012768	3
.text:1001276D 09	db 9
.text:1001276E C1	db 0C1h ; Á
.text:1001276F FF	db 0FFh ; ÿ
.text:10012770 FF	db 0FFh ; ÿ 0x1001276D + 0xFFFFC109 = 0x1000E876
illustrate jumper (3)	
.text:1000E876	
.text:1000E876 81 FC 54 C9 00 00	cmp esp, 0C954h
.text:1000E87C E8 E4 58 FF FF	call jumper 10004465 : 0x1000918f
.text:1000E87C	····· / ····· / ······
.text:1000E881 0E	db ØEh
.text:1000E882 A9	db 0A9h : ©
.text:1000E883 FF	db ØFFh : ÿ
.text:1000E884 FF	db ØFFh ; ÿ
illustrate jumper (4)	
.text:1000918F	
.text:1000918F 0F 82 63 3B 00 00	ib loc 1000CCF8
.text:10009195 E8 CB B2 FF FF	call jumper 10004465 : 0x1001153c
.text:10009195	· · · · · · · · · · · · · · · · · · ·
.text:1000919A A2	db 0A2h ; ¢
.text:1000919B 83	db 83h ; f
.text:1000919C 00	db 0
.text:1000919D 00	db 0
illustrate jumper (5)	
.text:1001153C	
.text:1001153C 83 E4 F8	and esp. 0FFFFFF8h
.text:1001153F E8 21 2F FF FF	call jumper 10004465 ; 0x10007894
.text:1001153F	· · · · · · · · · · · · · · · · · · ·
.text:10011544 50	db 50h ; P
.text:10011545 63	db 63h ; c
.text:10011546 FF	db 0FFh ; ÿ
.text:10011547 FF	db ØFFh ; ÿ

illustrate jumper (6)

After doing away with the jumper calls, here's a snippet of recovered "shreds" of instructions:

.text:1000D817	55					push ebp	
.text:10012766	8B	EC				mov ebp, esp	
.text:1000E876	81	FC	54	С9	00	cmp esp, 0C954h	
.text:1000918F	ØF	82	63	3B	00	jb loc_1000CCF8	
.text:1001153C	83	E4	F8			and esp, 0FFFFFF8h	
.text:10007894	81	FC	A0	F6	60	cmp esp, 0F6A0h	
.text:10011444	ØF	82	E1	EC	FF	jb loc_1001012B	
.text:10006591	81	EC	34	04	00	sub esp, 434h	
.text:10007433	8B	4D	0 8			mov ecx, [ebp+8]	
.text:10009284	53					push ebx	
.text:100067E8	56					push esi	
.text:1000D612	57					push edi	
.text:10002CD0	8D	44	24	28		lea eax, [esp+28h]	
.text:100115C5	81	FC	6A	98	00	cmp esp, 986Ah	
.text:100087BA	ØF	82	AØ	3D	60	jb loc_1000C560	
.text:1000C358	33	F6				xor esi, esi	
.text:100130DC	50					push eax	
.text:100037A9	51					push ecx	
.text:10002CEB	89	74	24	30		mov [esp+30h], esi	
.text:10015CDD	89	74	24	34		mov [esp+34h], esi	
.text:1000999E	89	74	24	3C		mov [esp+3Ch], esi	
.text:1000884B	89	74	24	38		mov [esp+38h], esi	
.text:10005A84	E8	A4	83	<u>00</u>	<u>00</u>	call loc_1000DE2D	

I greyed out the junk "cmp" and "jb" instructions. Up till this point, the process of recovery is very manual, with abit of help from this IDAPython script I wrote:

SHADOWPAD-analysis/ida_get_next_instr.py at main · asunaamawaka/SHADOWPAD-analysis

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

<u>github.com</u>

The python script is not perfect — at some point it will fail to work as intended, and I will have to apply the script again at the position where it failed. Furthermore, this manner of "advancing" through the disassembly in IDA didn't feel very efficient. As it turns out, using the "trace" feature in the debugger produced the expected results with ease.

🐋 Trace into...

Break Condition:	Example: eax == 0 && ebx == 0			
Log Text:	0x{p:cip} {i:cip}			
Log Condition:	Example: eax == 0 && ebx == 0			
Command Text:	Example: eax=4;StepOut			
Command Condition:	Example: eax == 0 && ebx == 0			
Maximum trace count:	500000			
Switch Condition:	Example: mod.party(dis.branchdest(cip)) == 1			
	Log <u>File</u> <u>OK</u> Cancel			
	Hint: History is available in every text field with the Up/Down arrows!			

trace in x32dbg

14084	0x10009347	call 0x10010CA0
14085	0x10010CA0	call jumper_0x10004465
14086		
14087	0x1000416C	push ebp
14088	0x1000416D	call jumper_0x10004465
14089		
14090	0x1000E2C2	mov ebp, esp
14091	0x1000E2C4	call jumper_0x10004465
14092		
14093	0x10013B80	push esi
14094	0x10013B81	call jumper_0x10004465
14095		
14096	0x100048C3	mov esi, edx
14097	0x100048C5	call jumper_0x10004465
14098		
14099	0x100097B2	cmp esp, 0xBF44
14100	0x100097B8	call jumper_0x10004465
14101		
14102	0x1000265C	jb 0x100093BB
14103	0x10002662	call jumper_0x10004465
14104		
14105	0x100053C9	test ecx, ecx
14106	0x100053CB	call jumper_0x10004465
14107		

X

```
14108
       0x100024D4 jle 0x10004C22
       0x100024DA call jumper_0x10004465
14109
14110
       0x10012309 push ebx
14111
       0x1001230A call jumper 0x10004465
14112
14113
       0x100138C2 push edi
14114
       0x100138C3 call jumper_0x10004465
14115
14116
       0x1000621B mov edi, dword ptr ss:[ebp+0x8]
14117
       0x1000621E call jumper_0x10004465
14118
14119
       0x1000F8A1 cmp esp, 0x16C
14120
       0x1000F8A7 call jumper 0x10004465
14121
14122
       0x10009EA8 jb 0x1000C880
14123
       0x10009EAE call jumper 0x10004465
14124
```

traced log

After some cleaning up, here's the code logic that decrypts the payload from registry, for a taste of what we can see after doing away with the jumper calls and junk cmp-jbs.

```
0x1001082D mov edi, dword ptr ss:[ebp+0x8]
0x100075B6 call 0x10010247
->
      0x10013ECE push esi
      0x10003506 test edi, edi
      0x10015384 jne 0x10013F36
      0x100063D1 movzx eax, word ptr ds:[edi]
      0x1000248D xor eax, 0xE3798186
      0x1000E4BB cmp eax, 0xE379DBCB
      0x10008B35 je 0x10011115
      0x10006E89 mov esi, dword ptr ds:[edi+0x3C]
      0x1000A63C mov ecx, dword ptr ds:[edi+esi*1]
      0x1001027C xor ecx, 0xCD5D5126
      0x1000A9CF cmp ecx, 0xCD5D1476
      0x1000C57B je 0x1000D957
      0x10010E32 mov eax, dword ptr ds:[edi+esi*1+0x8]
      0x10013F90 pop esi
      0x1000583D ret
0x10012F94 mov edi, dword ptr ss:[ebp-0x8]
0x10009650 mov esi, dword ptr ss:[ebp-0x14]
0x10013AA4 xor eax, 0xA7847046
0x1000CA8D push edi
0x1000A5F1 mov edx, edi
0x1000DE14 mov ecx, esi
0x10009347 call 0x10010CA0
      0x1000416C push ebp
->
      0x1000E2C2 mov ebp, esp
      0x10013B80 push esi
      0x100048C3 mov esi, edx
      0x100053C9 test ecx, ecx
      0x100024D4 jle 0x10004C22
      0x10012309 push ebx
      0x100138C2 push edi
      0x1000621B mov edi, dword ptr ss:[ebp+0x8]
      0x10007D86 sub edi, esi
             0x1000B41E mov edx, eax
      ->
             0x10015F00 add edx, edx
             0x10006429 lea eax, ds:[eax+edx*8+0x107E666D]
             0x1000667B mov edx, eax
             0x10007AB0 shr edx, 0x18
             0x10005A26 mov ebx, eax
             0x10001C9D shr ebx, 0x10
             0x1000F523 add dl, bl
             0x10008B8E mov ebx, eax
             0x1000E383 shr ebx, 0x8
             0x100027F9 add dl, bl
             0x1000566A add dl, al
             0x1000FDAC xor dl, byte ptr ds:[esi+edi*1]
             0x10013BB5 inc esi
             0x10012E95 dec ecx
             0x10004594 mov byte ptr ds:[esi-0x1], dl
             0x10005D43 jne 0x10009E6E
      . . .
```

traced instructions for decrypting payload from registry **Malware Configuration**

In order to recover the configuration data, it helps to know what it looks like from older variants of ShadowPad (without the shredding obfuscation) — so that we can recognize it in memory. I used APIMonitor to look out for memory copies, because I knew that was what ShadowPad will do with its configuration.

Summary	/ 12,415 of 25,3	332 calls	50% filtered out	9.02 MB used bdreinit.exe					
🗢 🐴	🗢 🚓 🎫 + 😋 🧠 🗉 📖 🐶 📆 🔣 🕘 - 🗇								
#	Time of Day	Thread	Module	API Q	Return Value	Error ^			
4626	4:53:06.189 PM	1	kernel32.dll	_strcmpi ("KERNEL32.dll", "twain_32.dll")	-1				
4630	4:53:06.189 PM	1	KERNELBASE.dll	wcschr ("C:\exe\bdreinit.exe", "\")	0x005a2c98				
4631	4:53:06.189 PM	1	KERNELBASE.dll	wcsrchr ("\exe\bdreinit.exe", "\')	0x005a2ca0				
4632	4:53:06.189 PM	1	KERNELBASE.dll	wcsncmp ("C:\exe\bdreinit.exe", "C:\exe;C:\Windows\system32;C:\Window	0				
4633	4:53:06.189 PM	1	KERNELBASE.dll	LdrLoadDII ("C:\exe;C:\Windows\system32;C:\Windows\system;C:\Windo	STATUS_SUCCESS				
4636	4:53:06.189 PM	1	KERNELBASE.dll	LdrGetProcedureAddress (0x77070000, 0x002fc620, 0, 0x002fc634)	STATUS_SUCCESS				
4637	4:53:06.189 PM	1	KERNELBASE.dll	GetProcAddress (0x76d60000, "memcpy")	0x76d69910				
4639	4:53:06.189 PM	1	KERNELBASE.dll	LdrGetProcedureAddress (0x76d60000, 0x002fc97c, 0, 0x002fc990)	STATUS_SUCCESS				
4644	4:53:06.189 PM	1	KERNELBASE.dll	memcpy (0x005c6510, 0x00edc609, 2198)	0x005c6510				
4725	4:53:06.189 PM	1	kernel32.dll	_strcmpi ("KERNEL32.dll", "twain_32.dll")	-1				
4729	4:53:06.189 PM	1	KERNELBASE.dll	wcschr ("C:\exe\bdreinit.exe", "\")	0x005a2c98				
4730	4:53:06.189 PM	1	KERNELBASE.dll	wcsrchr ("\exe\bdreinit.exe", '\')	0x005a2ca0				
4731	4:53:06.189 PM	1	KERNELBASE.dll	wcsncmp ("C:\exe\bdreinit.exe", "C:\exe;C:\Windows\system32;C:\Window	0	-			
•						P			
Hex Buf									
🚽 🖬	2 1 2	4 ⁸ 8 4	e Ba						
0000	00 00 00 00 0	0 00 00	00 00 00 00 00 00	0 00 19 00 26 00 46 00 53 00 5f 00 6f 00 7a 00	&.F.So	.z. ^			
001c	85 00 90 00 c	2 00 cd	00 02 01 07 01 00	2 01 2e 01 54 01 59 01 5e 01 00 00 00 00 00 00	T.Y.^	···· 🔲			
0038	08 08 08 08 08 0	8 08 08	00 00 00 00 00 00 00	0 00 00 00 00 00 63 01 73 01 83 01 93 01 08 08	c.s				
0070	ff ff ff ff f	f ff 00	00 00 00 00 00 00	0 00 00 00 00 00 00 00 00 00 00 00 00 0					
008c	00 00 00 00 0	0 00 00	00 00 00 4d e7 15	5 00 d1 d6 3a 3d 45 b0 10 ab 96 b3 0e 1d 3e 04	M	.>.			
00a8	20 07 f1 8d 5	3 fe 66 d c0 4e	87 e7 09 00 a2 cc	$1 = 3 = 95 = 29 = 53 = 15 = 41 = 32 = 91 = 7 = 10 = 00 = 45 = a = 85 = S. f 5 = 5d = d5 = 49 = 0a = 74 = 41 = 31 = 71 = 41 = 03 = c6 = b2 = 7 = 09 = 00 = \pm m = N$)S.O≻E + 1 T ±01 0	E			
00e0	f1 15 84 98 0	b 9c 7d	b7 72 ca e7 08 00) 52 cc 39 96 db 41 06 78 df e7 0c 00 c9 1f 90}.r.	R.9A.x				
00fc	c2 41 95 6d a	7 f0 2d	03 6b 00 e8 07 00) 95 ae 90 e3 2a 2b ca 20 e8 07 00 b7 62 18 f2 .A.mk.	*+	b			
0118	08 d6 50 40 e	8 07 00	51 26 81 c2 ee 92	2 d6 8c e8 2e 00 7e ce 5c 64 f9 48 40 f6 3d a7 P@Q&.		.=.			
0150	79 e7 13 9d 1	d b0 e9	80 ac e8 07 00 02	2 bc f7 24 a3 39 97 e8 e8 31 00 6a 96 d1 2d b7 v	x\$.91.i.	y			
016c	3e f3 43 cb b	f c6 db	50 ed 98 28 88 37	1 1d 1d 6f 56 cd 63 19 dc c5 81 45 7d 14 15 48 >.CP	(.7oV.cE)	н			
0188	da 52 c4 e2 3	5 22 b5	8c 54 08 c4 ba 2c	25 d5 37 14 e9 01 00 3e 32 e9 01 00 3e 63 e9 .R5"T.	,8.7>2	>c.			
01c0	15 9d d1 66 9	6 D4 67 4 e9 22	da 21 25 33 a7 f8 00 93 f7 d1 c4 67	3 /e 1/ /c 4e e9 cc UD 38 I9 8a 69 e8 UA UC 56[g.]% 7 14 14 bb 54 5e 5b ce b5 c9 ea e9 c4 3c 73 64f."	s~. N81. gT^[∨ <sd< td=""></sd<>			
01dc	0c 85 31 75 2	f b5 b4	35 e1 35 47 2e 1d	i 8e a9 e9 01 00 2d bd e9 01 00 82 d0 e9 01 001u/5.50	G				
01f8	c7 54 ea 0c 0	0 c7 01	f0 28 47 e7 8a f8	6 6 5 6 5 db b5 ea 0 c 00 58 72 d1 54 d8 f5 5 e .T (G	m][Xr.T	<u>.</u>			
0214	16 54 b0 d9 8 45 39 49 b0 d	4 15 eb	0c 00 21 45 68 1a	a al 68 ec 69 55 db 56 96 76 eb 0c 00 69 66 4a .T!	EhU.V.v	J			
024c	00 00 00 00 00	0 00 00	00 00 00 00 00 00						
00.00	~~ ~~ ~~ ~~ ~		~~ ~~ ~~ ~~ ~~ ~~						

APIMonitor log of memcpy

One visual characteristic of the configuration data is that it will start and end with many zeroes, and it is not very long (the part that looks like encrypted data is approx. ~400 bytes long). In this particular sample, the memory size expected for the configuration is 2198 bytes long. This can perhaps be a helping value to look out for the memory call dealing with the configuration.

Decryption of the configuration data uses the exact same algorithm as what is used to decrypt the keystroke log file. Earlier I shared the standalone python script used to decrypt the keystroke log file. Here's the IDAPython version to handle the configuration data in IDA:

<u>SHADOWPAD-analysis/ida_decrypt_config.py at main · asuna-</u> amawaka/SHADOWPAD-analysis

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

<u>github.com</u>

And here's the decrypted configuration:

8/22/2021 9:42:57 PM		
exchange		
%ALLUSERSPROFILE%\DRM\Test\		
Test.exe		
log.dll		
log.dll.dat		
MyTest		
MyTest		
MyTest		
SOFTWARE\Microsoft\Windows\CurrentVersion\Run		
MyTest		
%ProgramFiles%\Windows Media Player\wmplayer.exe		
%windir%\system32\svchost.exe		
TCP://ti0wddsnv[.]wikimedia[.]vip:443		
SOCKS4		
SOCKS4		
SOCKS5		
SOCKS5		

ShadowPad configuration decrypted

Hmmm~ it looks like someone was testing his BEE some 3 months ago, based on that timestamp in the configuration.

Well then, that is all I have. Come chat with me on Twitter if you have any idea how I can automate this analysis; I seem to have done most stuff in the painful way :|

Network IOC:

ti0wddsnv[.]wikimedia[.]vip:443

Host IOCs:

log.dll — SHA256: 8D1A5381492FE175C3C8263B6B81FD99AACE9E2506881903D502336A55352FEF

log.dll.dat — SHA256: 0371FC2A7CC73665971335FC23F38DF2C82558961AD9FC2E984648C9415D8C4E

Scripts mentioned in this post are here:

GitHub - asuna-amawaka/SHADOWPAD-analysis

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

<u>github.com</u>

[1] https://securelist.com/apt-trends-report-q3-2021/104708/

[2] Higaisa or Winnti? APT41 backdoors, old and new - PTSecurity (<u>https://ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/higaisa-or-winnti-apt-41-backdoors-old-and-new</u>)

[3] Evolution after prosecution: Psychedelic APT41 - TeamT5 at VB2021 (<u>https://vblocalhost.com/conference/presentations/evolution-after-prosecution-psychedelic-apt41</u>)

~~

Asuna | https://twitter.com/AsunaAmawaka