
COMPUTER SECURITY CONFERENCE
NO HAT
Ghost in the Machine
Live Fire Threat Actor Dissection
GHOST
in the machine
retooling_
01001010
10110101
00111100
11010010

Who we are....

@DrCh40s

silvio

- ❑Co-founder of *Retooling LLC*
- ❑Former Senior Cyber Security Architect @ LEONARDO Spa - Cyber Security Division
- ❑Senior Security Researcher @ EMC/RSA -> DELL – Center of Excellence
- ❑Malware reverse engineer @ Symantec - Security Response
- ❑PhD Network Security @ University of Pisa
- ❑M.Sc. in Computer Engineering

dr.ch40s@stealthcraft.io

Antonio

- ❑Co-founder of *Retooling LLC*
- ❑Former Senior Cyber Security Architect @ LEONARDO Spa - Cyber Security Division
- ❑Cyber Threat Analyst
- ❑PhD System Security
- ❑M.Sc. in Computer Science

@t0nvi

t0@stealthcraft.io

retooling _

Revo

Implants Refresh Graph View Activity

Name	Platform	User	
DC-1	Windows x86-64 (64-bits)	Administ	
http-reflective-DF-TEST-003-reflectiveca...	Windows x86-64 (64-bits)	SYSTEM	
SYSTEM	Windows x86-64 (64-bits)	SYSTEM	
OLD-1	Windows x86-64 (64-bits)	Administ	
redfiber-http-DF-TEST-004-CrashWB-After-...	Windows x86-64 (64-bits)	Administ	
OLD-2	Windows x86-64 (64-bits)	Administrator	DESKTOP-66AK1
DEMO-1	Windows x86-64 (64-bits)	Administrator	DESKTOP-66AK1
redfiber-http-DF-TEST-008-test-screensho...	Windows x86-64 (64-bits)	rf-developer	RF-DEV
rf-http-DF-TEST-006-EXEC-tech2	Windows x86-64 (64-bits)	rf-developer	RF-DEV
redfiber-http-DF-TEST-005-properly-uninstalled	Windows x86-64 (64-bits)	Administrator	WIN-96K8TOLOKSO

Emulation Library Refresh

Name	Actors	Version
Latrodectus	TA577, TA578	v1.0.0
Havoc	Unknown	v1.0.0
Lynx	Unknown	v1.0.0
PlugX	Axiom, DragonOK, APT3, (11 more)	v1.0.0-p5-dbg

PlugX v1.0.0-p5-dbg Refresh Emulate

Readme Specs Module

Aliases

DestroyRAT, Kaba, Korplug, Sogu, TVT, Thoper

Description

PlugX is a remote access tool (RAT) with modular plugins that has been used by multiple threat groups.

Who is using PlugX?

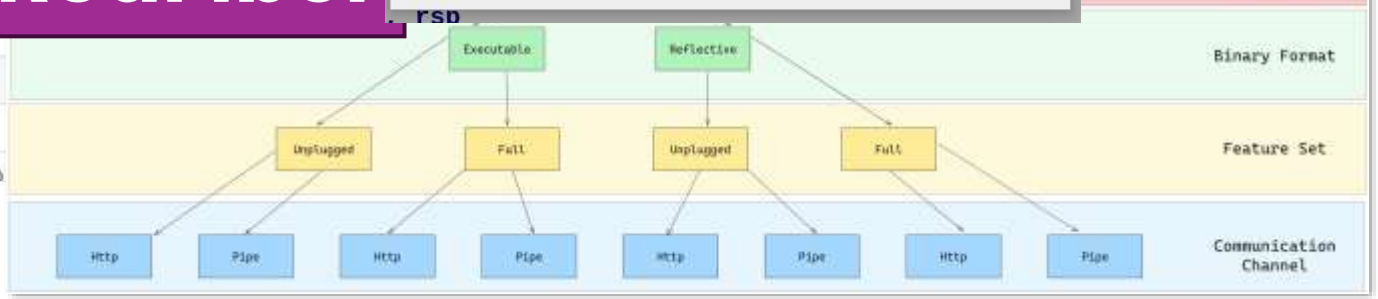
Axiom, DragonOK, APT3, Threat Group-3390, Winnti Group, menuPass, TA459, GALLIUM, APT41, Higaisa, Mustang Panda, LuminousMoth, Cinnamon Tempest, Daggerfly.

Techniques

This emulated version of **PlugX** implements 8 techniques.

Name	Code
Data from Local System	T1005
Query Registry	T1012
Masquerading	T1036
Process Discovery	T1057
File Deletion	T1070.004
File and Directory Discovery	T1083
Modify Registry	T1112
Screen Capture	T1113

Redfiber



MalOpSec Saga

MalOpSec I –

- ❑TTPs used to impair analysis
  - ❑LLVM obfuscation
  - ❑Injection
  - ❑Persistence
  - ❑Loader
  - ❑Anti VM/Debug

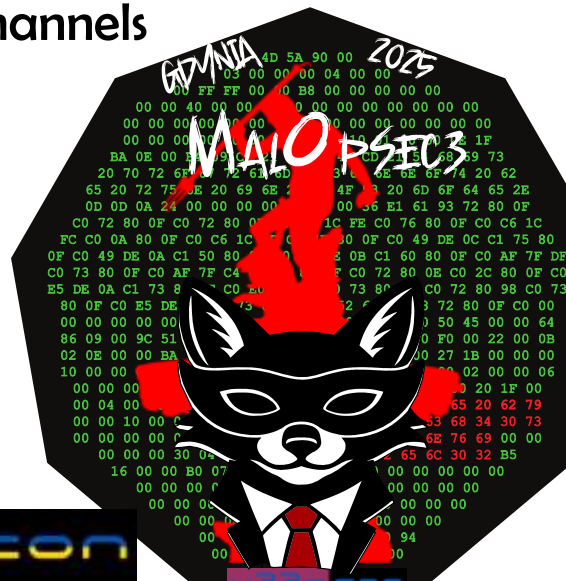
MalOpSec II – Great Escape

- ❑TTPs used to evade EDR and EDR detection points:
  - ❑Minifilter, ELAM, PPL, ETW, notification callback, mem scanners, …
  - ❑Unhooking
  - ❑ROP
  - ❑Vulnerable Drivers
  - ❑Stack spoofing
  - ❑.Net obfuscation

MalOpSec III – False Flag

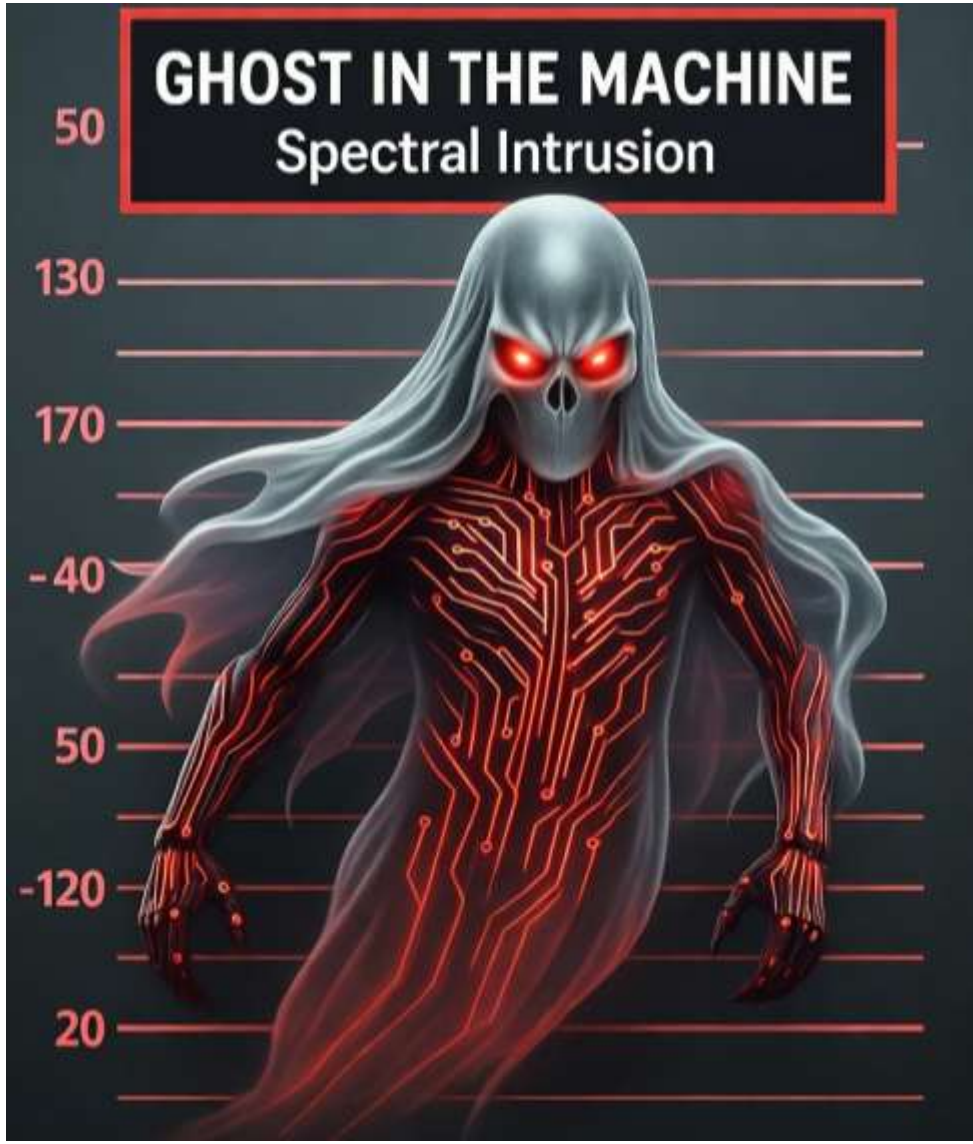
- ❑The real emulation:
  - ❑Mbc-PE-Compilers artifact
  - ❑Static Analysis tool
  - ❑Memory allocation and shellcode
  - ❑Implant as system
  - ❑Ransomware emulation
  - ❑Com. channels

PRIVATE



Ghost in the machine activities....

- This APT tried to infect this year:
  - NATO personal during the Locked Shield – FOR_700
  - RECON organizer during the conference
- The infections were slightly different
  - During RECON the threat actor added sophistication on both the first and second stages... maybe we will talk about it later
- We recovered a new dropper sent to NoHat folks and currently the C2 is still active...



Clone the workshop repo

https://github.com/retooling-io/workshop-nohat25/

Verify that you have **ALL** the required tools

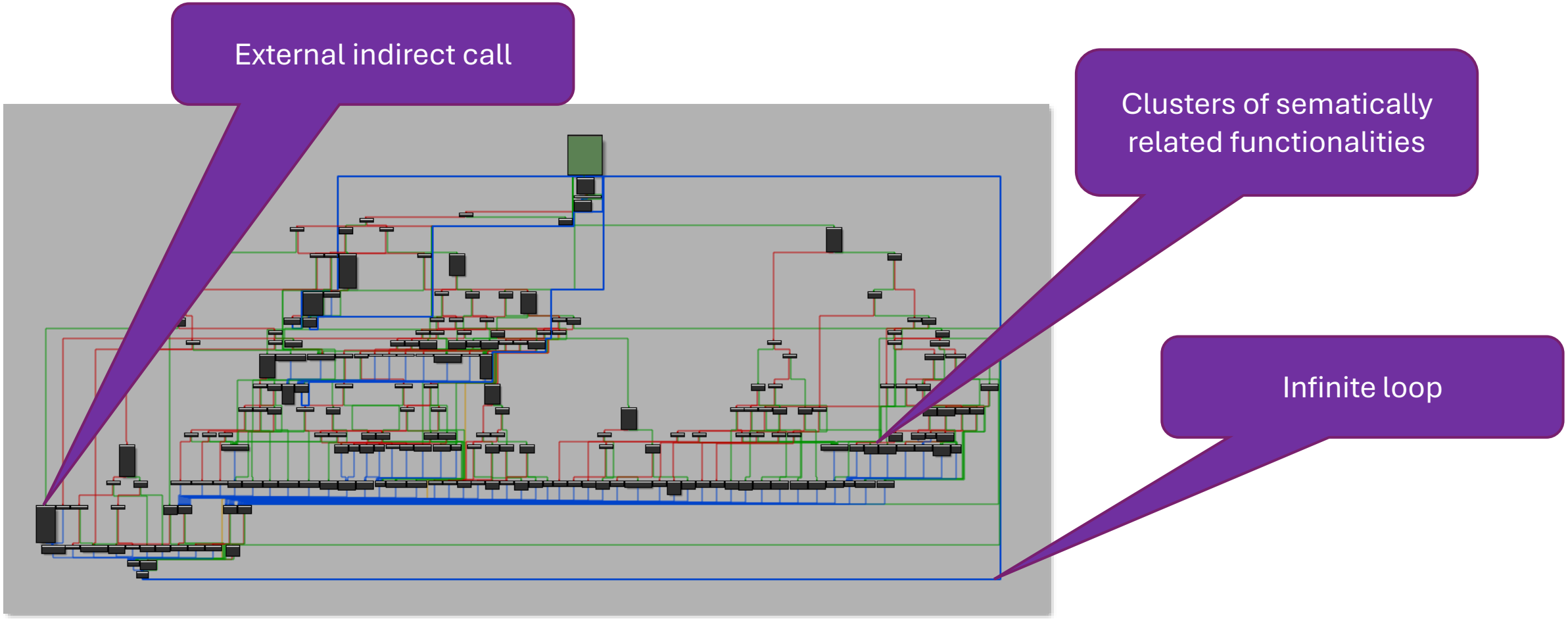
Msedgewebview4!read_input_file

```
__int64 __fastcall read_input_file(struct_pCtx *pCtx, const char *pFileName)
{
  FILE *fp; // rdi
  unsigned int file_size; // ebx
  char v5; // al
  __int64 result; // rax

  fp = fopen(pFileName, "rb");
  fseek(fp, 0, 2);
  file_size = ftell(fp);
  fseek(fp, 0, 0);
  if ( file_size >= 65537
    || (fread(gReadContent, file_size, 1ui64, fp),
        fclose(fp),
        pCtx->g_var_ptr = &unk_140025FF0,
        pCtx->pCurrPtr = gReadContent,
        *(_DWORD *)&gReadContent[file_size - 9] != 'EBEB')
    || (v5 = gReadContent[file_size - 5], (v5 & 2) != 0)
    || (v5 & 1) == 0 )
  {
    exit(1);
  }
  pCtx->pStartData2 = gReadContent;
  result = *(unsigned int *)&gReadContent[file_size - 4];
  pCtx->seed = result;
  return result;
}
```

Max file size

marker

Msedgewebview4!huge_switch



Decrypt the payload

```
[redacted]/source/repos/workshop-nohat25/scripts$ python3 decrypt_bytecode.py ../samples/01/SearchHost.bin 0x100b
4c1
Successfully decrypted '../samples/01/SearchHost.bin' using key 0x100B4C1 to '../samples/01/SearchHost.bin.decrypted'
[redacted]/source/repos/workshop-nohat25/scripts$ hexdump -C ../samples/01/SearchHost.bin.decrypted | head
00000000  13 01 01 ff 23 34 11 00  ef 00 40 34 ef 10 40 54  |....#4....@4..@T|
00000010  ef 00 40 3e ef 00 c0 4f  ef 00 40 01 73 00 10 00  |..@>...O..@.s...|
00000020  83 30 81 00 13 01 01 01  67 80 00 00 b7 25 00 00  |.0......g....%..|
00000030  9b 88 05 71 73 00 00 00  67 80 00 00 37 56 00 00  |...qs...g...7V..|
00000040  9b 08 06 e2 73 00 00 00  67 80 00 00 37 55 00 00  |....s...g...7U..|
00000050  9b 08 15 e2 13 05 00 00  73 00 00 00 67 80 00 00  |........s...g...|
00000060  97 26 00 00 03 b6 86 fd  93 05 05 00 63 1e 06 00  |.&..........c...|
00000070  37 55 00 00 9b 08 15 e2  13 05 00 00 73 00 00 00  |7U..........s...|
00000080  13 06 05 00 23 bc a6 fc  03 36 86 01 13 06 06 01  |....#....6......|
00000090  13 05 06 00 03 35 05 00  63 0a c5 00 83 26 85 10  |.....5..c....&..|
[redacted]/source/repos/workshop-nohat25/scripts$
```

Payload encryption

license.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	73	87	B8	49	5B	90	BB	10	5F	46	3A	B9	BE	68	96	BE	s‡¸I[.».‗F:¹¾h–¾
00000010	C1	AC	AB	2C	B7	1B	E6	2F	96	CE	F1	C3	06	E2	41	92	Á¬«,·.æ/–ÎñÃ.âA'
00000020	00	5F	1E	36	BE	4D	2D	C5	D4	A7	6C	50	38	7E	4E	51	.‗.6¾M-ÅÔ§lP8~NQ
00000030	7B	44	2A	9A	98	FC	DC	99	63	D1	95	57	D7	2C	6F	C3	{D*š˜üÜ™cÑ•W×,oÃ
00000040	87	0C	23	0A	E4	6A	5F	B0	2A	ED	33	A0	D1	E7	BD	4F	‡.#.äj‗°*í3 ÑÇ½O

license.bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000021E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000021F0	00	00	00	00	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	00	00	00	00	[illegible]	[illegible]	.....
00002200	00	00	00	00	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	00	00	00	00	[illegible]	[illegible]	.....
00002210	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00002220	00	00	00	00	00	00	00	00	00	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	00	00	.....
00002230	52	45	4C	41	00	46	45	41	54	01	C1	B4	00	01			RELA FEAT.ï¾.Þ

IsEncrypted flag

Encryption key

Marker

Understanding the target architecture

```
[redacted]/source/repos/workshop-nohat25/scripts$ python3 find_shellcode_arch.py ../samples/01/SearchHost.bin.dec
rypted
--- Shellcode Architecture Detection ---
Input shellcode length: 8776 bytes

Analyzing 4096 bytes of shellcode...

  X86 (32-bit): 3 instructions disassembled.
  X64 (64-bit): 3 instructions disassembled.
  ARM (32-bit - ARM mode): 0 instructions disassembled.
  ARM (32-bit - Thumb mode): 0 instructions disassembled.
  AArch64 (64-bit ARM): 0 instructions disassembled.
  MIPS (32-bit - Little Endian): 0 instructions disassembled.
  MIPS (32-bit - Big Endian): 0 instructions disassembled.
  MIPS64 (64-bit - Little Endian): 0 instructions disassembled.
  MIPS64 (64-bit - Big Endian): 0 instructions disassembled.
  RISC-V (32-bit): 27 instructions disassembled.
  RISC-V (64-bit): 27 instructions disassembled.
```

Getting the right architecture

- ❑ Assuming that it is executable code and decode it using e.g. the `capstone` lib
- ❑ Brute-force the architecture and take the best candidate:
- ❑ By running the script, `RISC-V (64-bit)` is selected as the best candidate for the payload

Possible heuristic: take the one with the highest number of correctly decoded `ret` instruction

```python
architectures = {
    "X86 (32-bit)": (CS_ARCH_X86, CS_MODE_32),
    "X64 (64-bit)": (CS_ARCH_X86, CS_MODE_64),
    #...
    "RISC-V (32-bit)": (CS_ARCH_RISCV, CS_MODE_RISCV32),
    "RISC-V (64-bit)": (CS_ARCH_RISCV, CS_MODE_RISCV64),
}
# ...additional code here...
# Initialize Capstone for the current architecture
md = Cs(arch_const, mode_const)
# keep going if the instructions are not decoded correctly
md.skipdata = True

instructions_count = 0
# Disassemble the data
for _ in md.disasm(data_to_analyze, 0): # dummy base address
    if(_.mnemonic != 'db' and _.mnemonic != '.byte'):
        if _.mnemonic == 'ret':
            instructions_count += 1

disassembly_details[arch_name] = instructions_count

# Update if this architecture yielded more instructions
if instructions_count > max_instructions:
    max_instructions = instructions_count
    most_likely_architecture = arch_name
```

Resolve Runtime (RISCV)

```
void ResolveRuntime(void)

{
  undefined8 pKernel32BaseAddr;
  undefined8 pfnLoadLibraryA;
  undefined8 pNtdllBaseAddr;
  undefined8 pWs2_32BaseAddr;
  char *pCurrLibName [13];

                    /* kernel32.dll */
  pKernel32BaseAddr = pfnLoadLibraryHash(0x536cd652);
  pfnVirtualProtect = GetProcAddrByHash(pKernel32BaseAddr,0xb23cae4);
  pfnCreateThread = GetProcAddrByHash(pKernel32BaseAddr,0x3defdc66);
                                                                151b);
  pfnSleep = GetProcAddrByHash(pKernel32BaseAddr,0xffffffffd8a41517);
  pfnGetSystemTimeAsFileTime = GetProcAddrByHash(pKernel32BaseAddr,0xffffffffbec8db07);
                                                                    );
  pfnGetProductInfo = GetProcAddrByHash(pKernel32BaseAddr,0x5e173207);
  pfnLoadLibraryA = GetProcAddrByHash(pKernel32BaseAddr,0xffffffffdf2bb  ec);
  pCurrLibName[0] = s_advapi32.dll_000018c8;
  jEcall(pfnLoadLibraryA,pCurrLibName);
  pfnGetUserNameW = GetProcAddrByHash(0xffffffffb89cdf4b);
  pNtdllBaseAddr = pfnLoadLibraryHash(0xffffffff46857d4);
  pfnWcslen = GetProcAddrByHash(pNtdllBaseAddr,0xffffffffaf2d5b2e);
  pfnMemcmp = GetProcAddrByHash(pNtdllBaseAddr,0x3dbadeb1);
  pfnVirtualFree = GetProcAddrByHash(pKernel32BaseAddr,0xffffffffaf2df57);
  pCurrLibName[0] = s_ws2_32.dll_000018d5;
  pWs2_32BaseAddr = jEcall(pfnLoadLibraryA,pCurrLibName);
  DAT_0000209b = GetProcAddrByHash(pWs2_32BaseAddr,0x6262ee6b);
  DAT_000020a0 = GetProcAddrByHash(pWs2_32BaseAddr,0xffffffff9ca52ed3);
  DAT_000020a8 = GetProcAddrByHash(pWs2_32BaseAddr,0xffffffffe4340368);
  DAT_000020b0 = GetProcAddrByHash(pWs2_32BaseAddr,0xffffffffb762f0e);
  DAT_000020b8 = GetProcAddrByHash(pWs2_32BaseAddr,0x77055568);
  DAT_000020c0 = GetProcAddrByHash(pWs2_32BaseAddr,0xffffffffcf76216a);
  DAT_000020c8 = GetProcAddrByHash(pWs2_32BaseAddr,0xffffffff043bfa1);
  DAT_000020d0 = GetProcAddrByHash(pWs2_32BaseAddr,0xffffffff953071fc);
  DAT_000020d8 = GetProcAddrByHash(pWs2_32BaseAddr,0x4776c8b9);
  DAT_000020e0 = GetProcAddrByHash(pWs2_32BaseAddr,0x47873bb8);
  DAT_000020e8 = GetProcAddrByHash(pNtdllBaseAddr,0xffffffffaaa75494);
```

Load by parsing the PEB

Relevant functions to load and execute external modules

Information retrieval

Networking APIs

The 65599 hash fuction (RISCV)

```
34          curr_hash = 0;
35          do {
36            pbVar4 = pbVar4 + 1;
37            uVar5 = (uint)bVar1;
38            bVar1 = *pbVar4;
39            curr_hash = (long)(int)((int)curr_hash * 65599 + uVar5);
40          } while (bVar1 != 0);
41          if (curr_hash == target_hash) {
42            return ordinal;
43          }
```

RtlHashUnicodeString function (wdm.h)

02/22/2024

The **RtlHashUnicodeString** routine creates a hash value from a given Unicode string and hash algorithm.

Syntax

Copy

ive is **TRUE**, a lowercase and uppercase string hash to the same value.

`[in] HashAlgorithm`

The hash algorithm to use. If *HashAlgorithm* s HASH_STRING_ALGORITHM_X65599, **RtlHashUnicodeString** uses the x65599 hashing algorithm. If *HashAlgorithm* s HASH_STRING_ALGORITHM_DEFAULT, **RtlHashUnicodeString** uses the default algorithm. Currently, the default algorithm is the x65599 hashing algorithm.

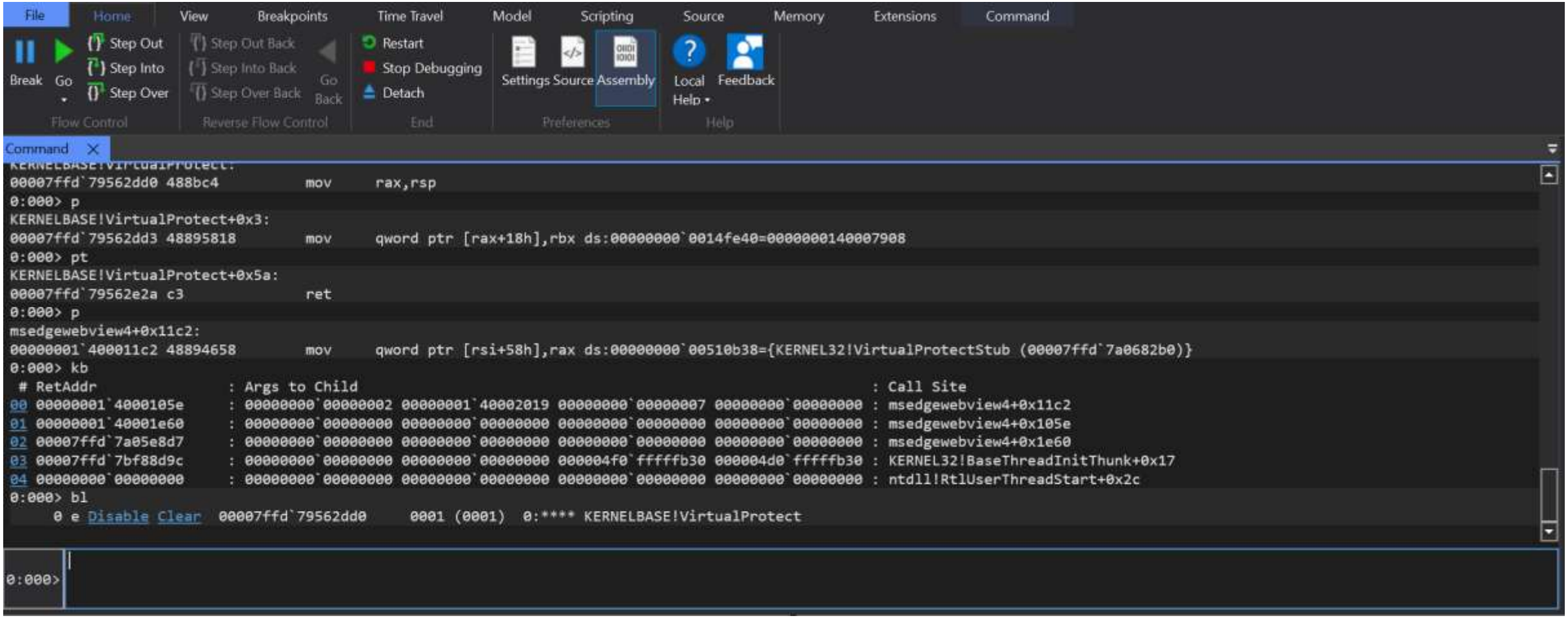
Loader capabilities

```
287   goto LAB_00000e30;
288   while( true ) {
289     curr_offset = curr_offset - 1;
290     puVar12 = puVar12 + 10;
291     if (curr_offset == 0) break;
292 LAB_00000da8:
293     uVar1 = *puVar12;
294     if ((uVar1 <= uVar5) && (uVar5 < puVar12[-1] + uVar1)) {
295       curr_offset = (long)(int)(uVar5 - uVar1) + (long)(int)puVar12[2] & 0xffffffff;
296       goto LAB_00000dd8;
297     }
298   }
299   curr_offset = 0;
300 LAB_00000dd8:
301   status_ = jVirtualProtect(pStartOfMZ,uVar15,0x40,aiStack_29c + 3);
302   if (status_ != 0) {
303     jCreateThread(0,0,pStartOfMZ + curr_offset,0,0,0);
304     FUN_000004d8(0xffffffffffffffff);
305     jVirtualProtect(pStartOfMZ,uVar15,(long)aiStack_29c[3],&uStack_78);
306   }
```

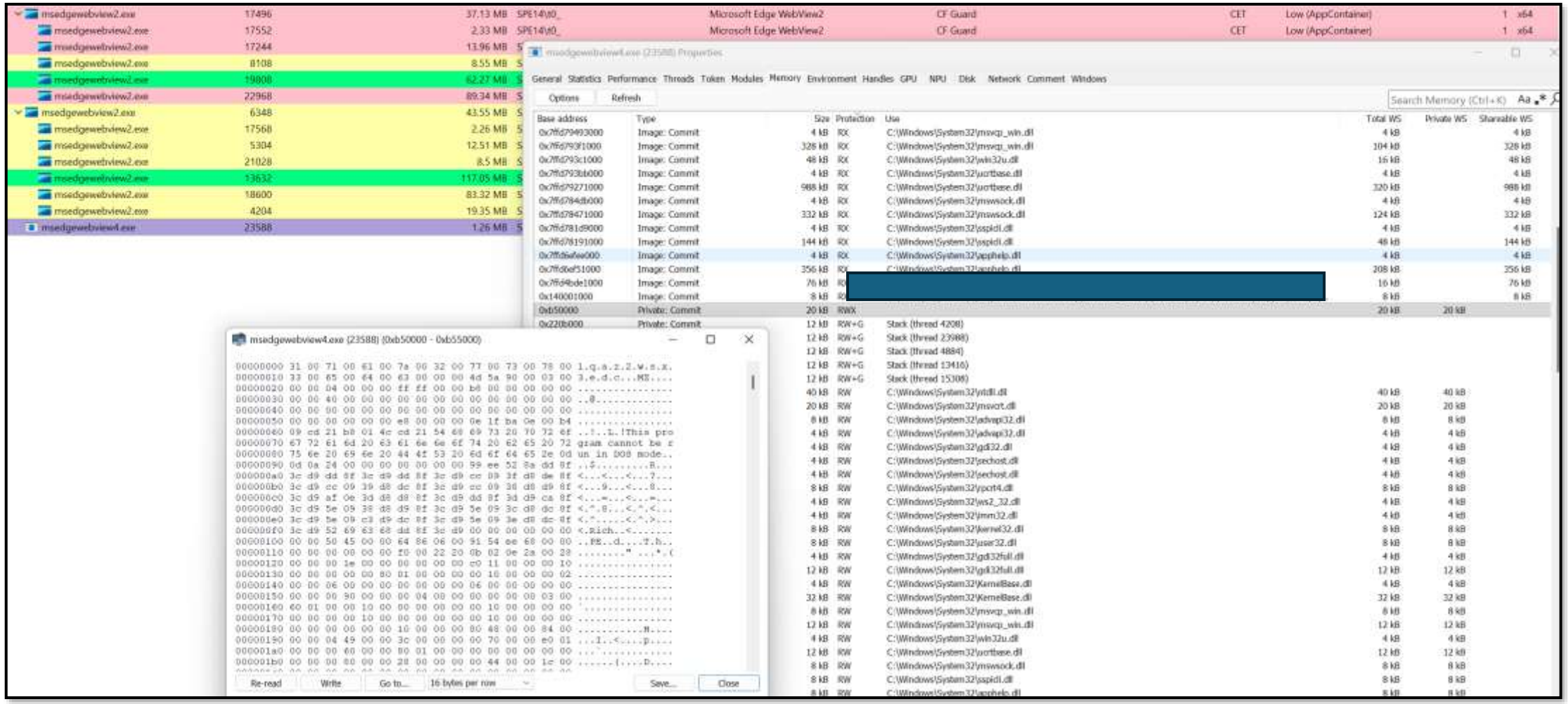
Find dll export

Execute export
(as shellcode)

Towards stage2 : Dump from memory



Towards stage2 : Dump from memory



Automation script server-side -> execution guardrail

- Second stage payload delivered only when the current username corresponds to the expected one
- It was sent only if the account name matched “nohatuser"

Save

```
1   import { main, panic } from "stdlib";
2   import { Implant } from "Implant";
3   import { getfolder, load } from "msg";
4
5   main(async function(){
6       // load the current implant
7       let implant = await Implant.current();
8       console.log(`loaded implant: '${implant.name}'`);
9
10      if (implant.user.match(/<username>/i)) {
11          console.log( `allowed user: ${implant.user}`)
12              let resp2 = await implant.do(load({
13                                              "moduleId": 28,
14                                              "params": "[]",
15                                              "persistence": false,
16                                              "type": 14
17                                              }))
18          if( resp2.error ) {
19              console.log(`error on '${implant.name}': '${resp2.error}' ` )
20          }
21      } else {
22          console.log(`blacklisted user: ${implant.user}`)
23      }
24
25  });
```

The module received over the network...

Enc algo: DES


```
; _BYTE Initial_Permutation_IP[64]
Initial_Permutation_IP db 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20
                                                        ; DATA XREF: DES_+4B↑o
                       db 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40
                       db 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51
                       db 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5
                       db 63, 55, 47, 39, 31, 23, 15, 7
; _BYTE PI[64]
PI                     db 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23
                                                        ; DATA XREF: DES_+2BC↑o
                       db 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13
                       db 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60, 28, 35, 3
                       db 43, 11, 51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58, 26
                       db 33, 1, 41, 9, 49, 17, 57, 25
; _BYTE Expansion_table_E[48]
Expansion_table_E db 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12
                                                        ; DATA XREF: DES_+183↑o
                       db 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21
                       db 20, 21, 22, 23, 24, 25, 24, 25, 26, 27, 28, 29, 28
                       db 29, 30, 31, 32, 1
; _BYTE P_table_post_sbox[32]
P_table_post_sbox db 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18
                                                        ; DATA XREF: DES_+19E↑o
                       db 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22
                       db 11, 4, 25
S_BoxS_8__64_          db 0Eh,4,0Dh,1,2,0Fh,0Bh,8,3,0Ah,6,0Ch,5,9,0,7,0,0Fh,7,4,0Eh,2,0Dh,1,0Ah,6,0Ch,0Bh,9,5,
                                                        ; DATA XREF: DES_+1F9↑o
                       db 0Fh,1,8,0Eh,6,0Bh,3,4,9,7,2,0Dh,0Ch,0,5,0Ah,3,0Dh,4,7,0Fh,2,8,0Eh,0Ch,0,1,0Ah,6,9,0B
                       db 0Ah,0,9,0Eh,6,3,0Fh,5,1,0Dh,0Ch,7,0Bh,4,2,8,0Dh,7,0,9,3,4,6,0Ah,2,8,5,0Eh,0Ch,0Bh,0F
                       db 7,0Dh,0Eh,3,0,6,9,0Ah,1,2,8,5,0Bh,0Ch,4,0Fh,0Dh,8,0Bh,5,6,0Fh,0,3,4,7,2,0Ch,1,0Ah,0E
                       db 2,0Ch,4,1,7,0Ah,0Bh,6,8,5,3,0Fh,0Dh,0,0Eh,9,0Eh,0Bh,2,0Ch,4,7,0Dh,1,5,0,0Fh,0Ah,3,9,
                       db 0Ch,1,0Ah,0Fh,9,2,6,8,0,0Dh,3,4,0Eh,7,5,0Bh,0Ah,0Fh,4,2,7,0Ch,9,5,6,1,0Dh,0Eh,0,0Bh,
                       db 4,0Bh,2,0Eh,0Fh,0,8,0Dh,3,0Ch,9,7,5,0Ah,6,1,0Dh,0,0Bh,7,4,9,1,0Ah,0Eh,3,5,0Ch,2,0Fh,
                       db 0Dh,2,8,4,6,0Fh,0Bh,1,0Ah,9,3,0Eh,5,0,0Ch,7,1,0Fh,0Dh,8,0Ah,3,7,4,0Ch,5,6,0Bh,0,0Eh,
; _BYTE PC1[64]
PC1                    db 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18
                                                        ; DATA XREF: DES_+8C↑o
                       db 10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36
                       db 63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22
                       db 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4
                       db 8 dup(0)
; _BYTE IP_2[48]
IP_2                   db 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19
                                                        ; DATA XREF: DES_+E2↑o
                       db 12, 4, 26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37
                       db 47, 55, 30, 40, 51, 45, 33, 48, 44, 49, 39, 56, 34
                       db 53, 46, 42, 50, 36, 29, 32
; char perm_shift[16]
perm_shift             db 2 dup(1), 6 dup(2), 1, 6 dup(2), 1
```

Encryption Algorithm, Mode, Block size

- 3DES-ECB-8
  - Blocks processed individually (no IV or XOR)
  - Block size 8 bytes

```
    u64Encrypted = DES_(u64Block, *key, 'e');
    u64Encrypted = DES_(u64Encrypted, key[1], 'd');
    u64Encrypted = DES_(u64Encrypted, key[2], 'e');
    v28 = 56LL;
    v29 = v35;
    do
    {
      *v29++ = u64Encrypted >> v28;
      v28 -= 8LL;
    }
    while ( v28 != -8 );
    v16 += 8;
    v4 += 8LL;
    v17 = v31 - 8;
    v15 -= 8;
    v5 += 64;
    v14 = v35 + 8;
    v13 = v36 + 8;
    v12 = v37 - 8;
  }
```

MBC Objective	MBC Behavior
CRYPTOGRAPHY	Encrypt Data::3DES [C0027.004]
DEFENSE EVASION	Obfuscated Files or Information::Encryption-Standard Algorithm [E1027.m05]
DISCOVERY	Code Discovery::Enumerate PE Sections [B0046.001]
PROCESS	Create Thread [C0038]

The key

- ❑Peb walk -> element 1
- ❑Random gen
- ❑Get a Nt* symbol

```
ppeb = (PEB *)ppeb->Ldr;
if ( ppeb )
{
  ppeb = *(PEB **)ppeb->ProcessParameters;
  ldr = (PPEB_LDR_DATA)ppeb->ProcessParameters;
  if ( ldr )
  {
    firstModule = (PLDR_DATA_TABLE_ENTRY)*(unsigned int *)((char *)&ldr[1].InInitializationOrderModuleList.Flink
                                                          + SHIDWORD(ldr->InInitializationOrderModuleList.Blink));
    ppeb = (PEB *)*(unsigned int *)((char *)&firstModule->InMemoryOrderLinks.Blink + (_QWORD)ldr);
    addressOfNames = (PWORD)*(unsigned int *)((char *)&firstModule->InInitializationOrderLinks.Flink + (_QWORD)ldr
    if ( *(_DWORD *)((char *)&firstModule->InMemoryOrderLinks.Blink + (_QWORD)ldr) )
    {
      *(_QWORD *)&i_1 = 0LL;
      ntFunctionCount = 0;
      do
      {
        nameRva = *(unsigned int *)((char *)&addressOfNames[2 * *(_QWORD *)&i_1] + (_QWORD)ldr);
        if ( *((_BYTE *)&ldr->Length + nameRva) == 'N' )
          ntFunctionCount += *((_BYTE *)&ldr->Length + nameRva + 1) == 't';
        ++*(_QWORD *)&i_1;
      }
      while ( ppeb != *(PEB **)&i_1 );
    }
    else
    {
      ntFunctionCount = 0;
    }
    if ( ntFunctionCount )
    {
      *(_QWORD *)&addressOfFunctions = *(unsigned int *)((char *)&firstModule->InMemoryOrderLinks.Blink
                                                        + (_QWORD)ldr
                                                        + 4);
      addressOfNameOrdinals = *(unsigned int *)((char *)&firstModule->InInitializationOrderLinks.Flink
                                                + (_QWORD)ldr
                                                + 4);
      QueryPerformanceCounter(&PerformanceCount);
      randomIndex = 0;
      if ( !*(_DWORD *)((char *)&firstModule->InMemoryOrderLinks.Blink + (_QWORD)ldr) )
        goto LABEL_30;
      nameTable_1 = (char *)addressOfNames + (_QWORD)ldr;
      ordinalTable = (char *)ldr + addressOfNameOrdinals;
      currentIndex = 1LL;
      selectedFunction = 0LL;
      functionTable = (char *)ldr + *(_QWORD *)&addressOfFunctions;
      do
      {
        functionNameRva = *(unsigned int *)&nameTable_1[4 * currentIndex - 4];
        found = 1;
        if ( *((_BYTE *)&ldr->Length + functionNameRva) == 'N'
          && *((_BYTE *)&ldr->Length + functionNameRva + 1) == 't' )
```

The key (II)

```
UINT64 u64Key[3] = 0000DEADB8D18B4C017FDEAD082504F6C32EDEAD050F0375;
```

- Overwrite syscall ID and other bytes with 0xDEAD

```
if ( selectedFunction )
{
  memset(syscallStub, 0, sizeof(syscallStub));
  for ( copyIndex = 0LL; copyIndex != 24; ++copyIndex )
    syscallStub[copyIndex] = selectedFunction[copyIndex];
  for ( qwordIndex = 0LL; qwordIndex != 3; ++qwordIndex )
    *(_QWORD *)(key + 8 * qwordIndex) = *(_QWORD *)&syscallStub[8 * qwordIndex];
  *(_QWORD *)&index = 0LL;            // re-used var
  do
    *(_WORD *)(key + 8LL * (*(_QWORD *)&index)++ + 4) = 0xDEAD // overwrite syscall ID
  while ( *(_QWORD *)&index != 3LL );
```

```
                        ZwCreateEventPair proc near             ; DATA XREF: .rdata:000000018013594C↓o
                                                                ; .rdata:off_18016BC78↓o ...
4C 8B D1                        mov     r10, rcx                ; NtCreateEventPair
B8 AE 00 00 00                  mov     eax, 0AEh
F6 04 25 08 03 FE 7F 01         test    byte ptr ds:7FFE0308h, 1
75 03                           jnz     short loc_1800A1555
0F 05                           syscall                         ; Low latency system call
C3                              retn
                        ; ---------------------------------------------------------------------------

                        loc_1800A1555:                          ; CODE XREF: ZwCreateEventPair+10↑j
CD 2E                           int     2Eh                     ; DOS 2+ internal - EXECUTE COMMAND
                                                                ; DS:SI -> counted CR-terminated command string
C3                              retn
```

Paddings...

- There were 3 main situations where padding was applied during encryption due fix size of the struct element containing the encrypted strings:
  - String < 3DES block size (8 bytes): Standard PKCS#7 padding was applied (adds N bytes of value N to reach block boundary) and then will be applied the next point
  - String < 16 bytes: The string "Out!Mars" was concatenated to fill the remaining space
  - String > 16 bytes: only the first 16 bytes were processed

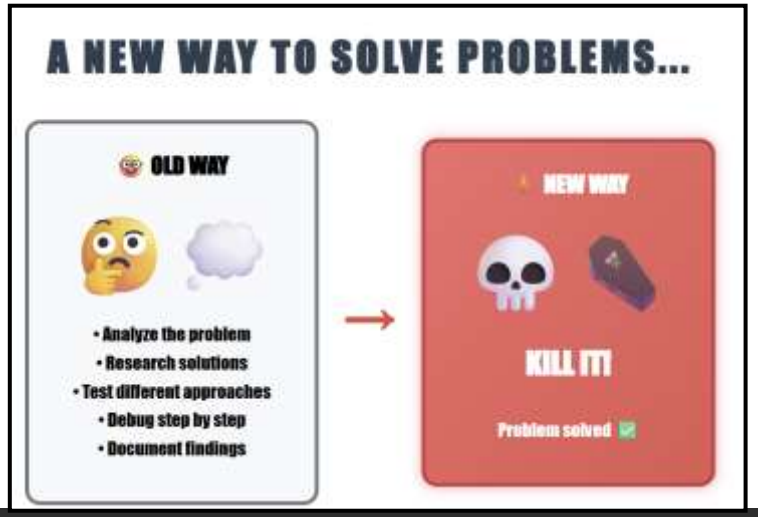
```
{
  _BYTE *outVal; // [rsp+28h] [rbp-50h]
  _BYTE *encBuffer; // [rsp+38h] [rbp-40h]
  __int64 i; // [rsp+40h] [rbp-38h]
  size_t *encLen; // [rsp+58h] [rbp-20h] BYREF

  encBuffer = (_BYTE *)TDES(cleanText, (__int64)&key, &encLen);
  if ( (unsigned __int64)encLen >= 16 )
    return encBuffer;
  outVal = VirtualAlloc(0LL, 16uLL, 0x3000u, 4u);
  if ( !outVal )
    return 0LL;
  for ( i = 0LL; ; ++i )
  {
    outVal[i] = encBuffer[i];
    if ( i == 7 )
      break;
  }
  *((_QWORD *)outVal + 1) = 'sraM!tuO';
  VirtualFree(encBuffer, 0LL, MEM_RELEASE);
  return outVal;
}
```

Killit hidden APIs

- The following APIs were encrypted and resolved at runtime using a custom implementation of `GetProcAddress`:
  - `NtQuerySystemInformation` (ntdll.dll): used to list the processes running on the machine
  - `OpenProcess` (kernel32.dll): used to open target processes
  - `NtTerminateProcess` (ntdll.dll): used to get its address and fill the shellcode
  - `WriteProcessMemory` (kernel32.dll): used to patch NtClose with shellcode
  - `NtClose` (ntdll.dll): API to patch with the kill shellcode

The kill that does not look like a kill

- if a target process was found, its `NtClose` memory was overwritten with the following shellcode where "Exit" will point to the `NtTerminateProcess` address, making the target process exit gracefully




```
if ( v7 )
{
  OpenProcess = (__int64 (__fastcall *)(__int64, _QWORD, _QWORD))GetProcAdd((__int64)v7, (__int64)apiName[3]);
  if ( OpenProcess )
  {
    hObject = (HANDLE)OpenProcess(0x1FFFFFLL, 0LL, a1);
    if ( hObject )
    {
      v10 = NtCurrentPeb();
      if ( v10 )
      {
        Ldr = v10->Ldr;
        if ( Ldr )
        {
          Flink = (__int64)Ldr->InMemoryOrderModuleList.Flink->Flink[2].Flink;
          if ( Flink )
          {
            ntclose_ = (void *)GetProcAdd(Flink, (__int64)apiName);
            if ( ntclose_ && (NtTerminateProcess = GetProcAdd(Flink, (__int64)apiName[1])) != 0 )
            {
              *(_QWORD *)((char *)&loc_7FF95A8E0456 + 2) = NtTerminateProcess;
              writeprocessmem = (unsigned int (__fastcall *)(HANDLE, void *, __int64 (__fastcall *)(), __int64, _BYTE *))GetProcAdd((__int64)v7, (__int64)apiName[4]);
              if ( writeprocessmem(hObject, ntclose_, shellcodeAdd, 19LL, v14) )
              {
                v6 = 1;
```

```
                                        ; void shellcodeAdd()
                                        shellcodeAdd    proc near
48 31 C9                                                xor     rcx, rcx
48 31 D2                                                xor     rdx, rdx

                                        loc_7FF95A8E0456:
48 B8 00 00 00 00 00 00 00 00                           mov     rax, 0
FF E0                                                   jmp     rax
                                        shellcodeAdd    endp
```

Killit bugs

- There were 2 different bugs in the Killit implementation:
  - Some hardcoded processes in the list were saved with capital letters, but before the check, a lowercase algorithm was applied to the process name, making the target impossible to match
  - The index used to traverse the array was wrong, so only the first 3 processes in that list could be targeted

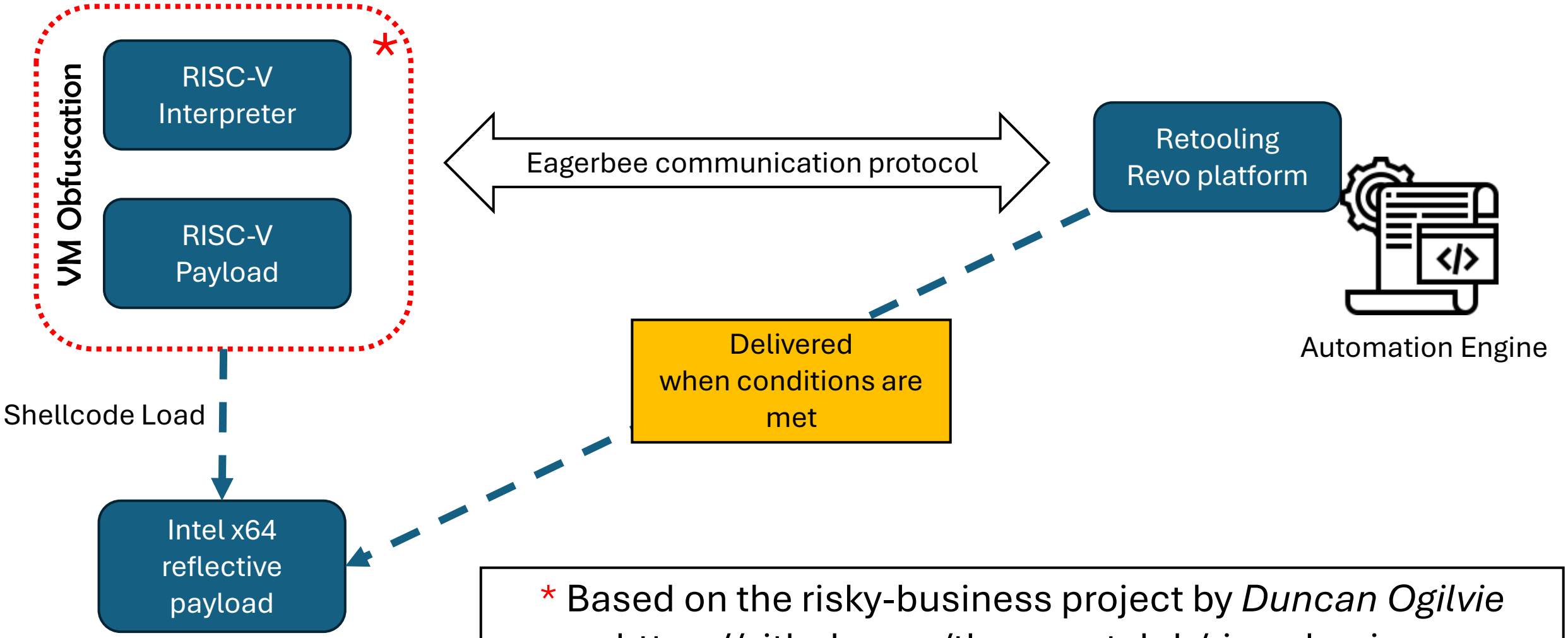
```
WideCharToMultiByte(0, 0, lpWideCharStr, -1, lpMultiByteStr, 260, 0LL, 0LL);
v35 = 0LL;
v5 = *lpMultiByteStr;
if ( *lpMultiByteStr )
{
  v20 = lpMultiByteStr;
  for ( j = v5; ; j = v6 )    // Inline tolower case
  {
    v30 = v20;
    if ( (unsigned __int8)(j - 'A') < 26u )
      *v30 = j + ' ';
    v31 = v30 + 1;
    v6 = v30[1];
    if ( !v6 )
      break;
    v20 = v31;
  }
}
v32 = enc3DES_wrapper((__int64)lpMultiByteStr);
v3 = 1;
v21 = 0LL;
while ( 2 )
{
  v15 = v21;
  for ( k = 0LL; ; k = v33 )
  {
    v16 = k;
    if ( TargetProcessList[v15][k] != *((_BYTE *)v32 + k) )
      break;
    v33 = v16 + 1;
    if ( v16 == 15 )
    {
      v4 = v3;
      goto LABEL_25;
    }
  }
  v34 = v15 + 1;
  if ( v15 != 7 )
  {
    v3 = v15 < 7;
    v21 = v34;
    continue;
  }
  break;
}
```

```
1  ida.exe
2  ida64.exe
3  dbgx.shell.exe
4  x64dbg.exe
5  ghidra.exe
6  binaryninja.exe
7  MoneyTrain.exe
8  FireBall.exe
9  EasyJob.exe
10 notepad.exe
```

Kill_it (the module over the network...)

- All the strings were encrypted using 3DES-ECB with 8-byte blocks
  - The key is retrieved by accessing a random Nt* API and getting the opcode, then overwriting some of them with the `0xDEAD` value
- To resolve the module base address, it uses PEB walking
  - Ntdll is accessed directly using ID 1
  - Kernel32 is accessed by comparing the decrypted name with the module name retrieved from PEB
- Main functionality → <u>Kill processes specified in a hardcoded encrypted list</u>

Overall architecture



* Based on the risky-business project by *Duncan Ogilvie*
https://github.com/thesecretclub/riscy-business

retooling_

Thanks to all participants!

See you next year with something worse...
maybe

www.retooling.io