

What We Thought Was SMOKELOADER: Introducing REMUS

Published: 2026-03-27 · Archived: 2026-09-08 02:02:04 UTC

Author: Izan Pérez Perpén (aachum) **Date:** 27 March 2026 **TLP:** TLP:WHITE **Sample:** `Kemus.exe` / SHA256: `cfc21d8df942918f7a74b99f2cccf7e54e2a6dd1ea6de60897ff0026a26b5c4`

Executive Summary

REMUS is a previously undocumented malware family combining infostealer and loader capabilities with a C2 resolution mechanism based on the Ethereum blockchain. At runtime, REMUS queries a public Ethereum RPC endpoint and extracts the C2 address from the `result` field of the JSON-RPC response. The C2 infrastructure is anchored to blockchain state rather than DNS, making traditional takedown and sinkholing ineffective.

The malware targets credentials from Chrome, Firefox, Steam, and the Windows clipboard. It captures screenshots via GDI and exfiltrates data over HTTP POST using a hardcoded Chrome 117 User-Agent. A loader component maps and executes a second-stage payload in memory without writing to disk.

The analysed sample is a 64-bit Windows PE with an internal build timestamp of March 19, 2026. It uses compile-time string encryption across at least five distinct XOR schemes, API hashing for import resolution, and control-flow obfuscation through chained two-entry jump tables.

Key Findings

- C2 address resolved at runtime from an Ethereum smart contract via `eth.llamarpc.com`
 - Strings encrypted using five distinct XOR schemes; 20 strings recovered through static analysis
 - All Windows API calls resolved at runtime via hash; no readable IAT
 - Second stage is a trojanized `sechost.dll` mapped fileless into memory
 - Full Firefox credential extraction: `cert9.db`, `logins.json`, `cookies.sqlite`, `formhistory.sqlite`, `places.sqlite`, `prefs.js`
 - Chrome/Chromium credential extraction via `\Local State` and DPAPI
 - Four-step C2 protocol reconstructed from PCAP; `tag=` parameter carries MT19937-derived hardware ID
-

Technical Analysis

1. Entry Point and Initialization

Execution begins at `_start` (`0x14001d4e`) with two calls to `sub_14002d970` using constants `0x38d75035` and `0xf44e2605`, resolving function pointers through the API hash resolver into globals `data_140038d78` and `data_14003d80`. Control then passes to `sub_1400062d0`.

2. String Encryption

REMUS uses five distinct encryption schemes across its string table, with different constants per string. The dominant scheme operates on UTF-16LE words:

```
key_i = (BASE - i * MULTIPLIER) & 0xFFFF
plaintext_word[i] = ciphertext_word[i] ^ key_i
```

Through analysis of all `__builtin_memcpy` blobs in the binary, the following strings were recovered:

Function	Scheme	Decrypted value
sub_1400062d0	XOR16 0x840e/0x7bf2	\BaseNamedObjects\
sub_140007210	XOR16 0x8594/0x7a6c	Content-Type: application/x-www-form-urlencoded\r\n
sub_140003020	Byte XOR 0x5d	Content-Disposition: form-data; name="
sub_14000c7b0	XOR16 0xd831/0x27cf	\Local State
sub_1400148a0	XOR16 0x8594/0x7a6c	cert9.db
sub_1400148c0	XOR16 0x3bb3/0xc44d	cookies.sqlite
sub_140014900	XOR16 0xa7f3/0x580d	formhistory.sqlite
sub_140014930	Arith 0x5e12	places.sqlite
sub_1400148e0	XOR16 0xf1d0/0x0e30	logins.json
sub_140014970	XOR16 0x1431/0xebcf	\prefs.js
sub_140016f70	XOR16 0xa2d0/0x5d30	rundll32 "
sub_140016f70	Byte arith 0x1d	__COMPAT_LAYER=RunAsInvoker
sub_14001c660	Arith 0x5e12	InstallPath
sub_14001c660	XOR16 0xa7f3/0x580d	\REGISTRY\MACHINE\SOFTWARE\...
sub_14001be90	Additive 0x1954	steam.exe
sub_14001ddd0	Byte XOR 0x17	# REMUS LOG\n\nbuild:\n date:
sub_140012190	Byte XOR 0x7e	dpapi.dll
sub_1400064e2	Arithmetic	&hwidx , tag=
sub_140004b5e	Arithmetic	User-Agent (Chrome 117)

Function	Scheme	Decrypted value
sub_1400064e2	Byte XOR	POST

Two blobs in sub_140016f70 use 32-bit arithmetic masks (0x91ba57dc , 0x1ffe8a56) whose key values depend on runtime register state and were not recoverable through static analysis.

3. Anti-Analysis

Dead loops. The pattern `do while (i == 0)` with `i` incremented inside appears throughout the binary. These execute exactly once, producing a computed value that appears data-dependent to decompilers.

Chained jump tables. Control flow is implemented as two-entry jump tables indexed by a condition byte:

```
rcx.b = condition
jump((&data_XXXXXXX)[rcx])
```

Single logical branches require resolving chains of 10 to 15 dispatcher hops.

API hashing. sub_14002d1a0 and sub_14002df90 resolve imports by walking export tables and comparing computed hashes. Identified hashes:

Hash	Function
0xa3b73ba5	CreateMutexW
0x23a50cfe	OpenMutexW
0xd39917fc	Sleep / WaitForSingleObject
0x95871f00	WinHttpCrackUrl
0xe145d93e	WinHttpOpen
0x2cba84c0	WinHttpConnect
0x15a90739	WinHttpOpenRequest
0x92acb581	WinHttpSendRequest
0xc797d48f	WinHttpReceiveResponse
0x86fffd1e	WinHttpCloseHandle
0x4958ad19	WinHttpReadData
REMUS checks for the presence of honey@pot.com.pst in %UserProfile%\Documents\Outlook Files . If found, execution terminates,	

Hash	Function
the file serves as a sandbox environment indicator.	

4. Mutex

sub_1400062d0 decrypts `\BaseNamedObjects\` and creates or opens a named mutex via hash `0xa3b73ba5` with `MUTEX_ALL_ACCESS` (`0x1f0003`). A global flag at `data_140034010` determines the execution branch. When the mutex already exists, the malware opens it with `SYNCHRONIZE | MUTEX_MODIFY_STATE` and continues rather than exiting.

5. COM Initialization

```
CoInitializeSecurity(nullptr, 0xffffffff, nullptr, nullptr,
    RPC_C_AUTHN_LEVEL_DEFAULT, RPC_C_IMP_LEVEL_IMPERSONATE, nullptr, 0, nullptr)
```

`RPC_C_IMP_LEVEL_IMPERSONATE` enables token impersonation through COM interfaces without a direct `AdjustTokenPrivileges` call, consistent with the `SeImpersonatePrivilege` usage in both stages.

6. Victim Identification

sub_1400064e2 initializes MT19937 from a hardware-derived seed. The 624-iteration state expansion uses the standard recurrence constant `0x6c078965`. The resulting state produces the `hwidx` victim identifier; PCAP analysis confirmed the `tag=` parameter carries the MD5 value embedded in the binary (`65b650d78cbf74f17a1f5c139d5ab278`).

7. C2 Communication

7.1 HTTP Stack

All WinHTTP calls are resolved at runtime via hash. The request sequence:

1. `WinHttpOpen` with User-Agent `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.0.0 Safari/537.36`
2. `WinHttpConnect` , `WinHttpRequest` (`POST`)
3. `WinHttpSendRequest` (`Content-Type: application/x-www-form-urlencoded`)
4. `WinHttpReceiveResponse` (timeout: 600s)
5. `WinHttpReadData` (buffer: 13,056 bytes)

On failure, the malware sleeps via hash `0xd39917fc` and retries up to 5 times, updating a global state flag at `data_140034018` after each cycle.

7.2 Blockchain C2 Resolution

The first network contact is a POST to `https://eth.llnampc.com` . The response is parsed by `sub_140001040` , a self-contained JSON parser supporting the full specification including floats, Unicode escapes, and nesting to depth 32. The `result` field contains the C2 address in hex encoding, decoded and stored at `data_140032f0c` and `data_140032f7a` .

The contract address is not present in the binary in any form. TLS 1.3 encryption prevented recovery from PCAP. Recovery requires TLS interception or on-chain analysis if the address surfaces through other means.

7.3 C2 Protocol

PCAP from a sandbox run captured plaintext HTTP to `adveryx.biz:6573` , allowing partial protocol reconstruction:

Step	Body	Response
Registration	<code>access_token=&debug=http://[c2]:[port]</code>	<code>{"success":true}</code>
1	<code>access_token=[token]&step=1</code>	unknown
2	<code>access_token=[token]&step=2</code>	unknown
3	<code>access_token=[token]&step=3</code>	unknown
4	<code>access_token=[token]&step=4</code>	HTTP 404 (sandbox blocked)

The `debug` field tracks which C2 was reached; internal state markers `2-PRE` , `2-EMPTY` , `2-SEND` appear during the step 2 loop. Access token observed: `a009c81b-9c46-4ee1-89ab-e58e16b6993d` .

8. Loader

The dispatcher in `sub_1400062d0` contains a branch that executes `jump(arg1)` , passing a downloaded payload address as a code pointer. No file is written to disk.

9. Information Stealing

Chrome/Chromium. `\Local State` targeted at `sub_14000c7b0` . `dpapi.dll` loaded explicitly at `sub_140012190` .

Firefox. Six profile files targeted: `cert9.db` , `logins.json` , `cookies.sqlite` , `formhistory.sqlite` , `places.sqlite` , `prefs.js` .

Steam. `InstallPath` read from `\REGISTRY\MACHINE\SOFTWARE\Valve\Steam` at `sub_14001c660` . `steam.exe` targeted at `sub_14001be90` .

Clipboard. `OpenClipboard` / `GetClipboardData` / `CloseClipboard` imported directly.

Screenshots. `GetDC` , `CreateCompatibleDC` , `CreateCompatibleBitmap` , `BitBlt` , `GetDIBits` .

10. Execution and Persistence

- `powershell -exec bypass -f "[path]"`
- `rundll32 "[path]"`
- `__COMPAT_LAYER=RunAsInvoker` for UAC bypass
- LNK file manipulation
- `Temp` directory for staging

Dynamic Analysis

Sample detonated on Triage (report `260326-sgfdzshv3q`).

behavioral1 (Windows 10 20H1)

Three C2 domains contacted after blockchain resolution:

Domain	IP	Port
<code>adveryx.biz</code>	<code>76.13.17.11</code>	<code>6573</code>
<code>coox.live</code>	<code>168.231.114.49</code>	<code>28313</code>
<code>padaz.pics</code>	<code>103.211.219.238</code>	<code>4219</code>

Memory dump at `0x000001CC531C0000` (~600KB). Detected: browser credential access, cryptocurrency wallet access, process enumeration.

Second Stage Payload

The stage 1 memory dump contains a trojanized `sechost.dll` :

```

FileVersion: 10.0.22000.434 (Windows 11 21H2)
SHA256:      352721b32ec1c8349985ceccfec8d1ca6e3e6cc12f83350c4ae1a75477588bc2
MD5:        6e22b1331a296a59a16a00cfc36b89c7
SizeOfImage: 0x9e000 (632KB)
PE timestamp: 0x31ec7be5 (1996-07-17, falsified)

```

Malicious code is woven into the legitimate `sechost.dll` structure, presenting a valid export table and version resources alongside the malicious components.

Capabilities identified from the import table, beyond stage 1:

- Credential Manager and LSA secrets: `CredReadW` , `CredWriteW` , `CredReadDomainCredentialsW` , `LsaRetrievePrivateData` , `LsaStorePrivateData` , `LsaQuerySecret`
- Token manipulation: `AdjustTokenPrivileges` , `CreateRestrictedToken` , `RpcImpersonateClient`

- Service persistence: `CreateServiceW` , `StartServiceW` , `DeleteService` , `RegisterServiceCtrlHandlerExW`
- ETW: `StartTraceW` , `ControlTraceW` , `EnableTraceEx2` , `EtwLogSysConfigRundown`
- Custom Base64 alphabet: `ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789#-`

The `Sc13` string in the stage 2 dump connects to `Sc11` , `Sc12` , `Sc14` , `Sc15` markers in stage 1, confirming a shared build system.

Stage 1: `Kemus.exe` (223KB)

SHA256: `cfc21d8df942918f7a74b99f2cccf7e54e2a6dd1ea6de60897ff0026a26b5c4`

Blockchain C2 resolution, credential theft, loader

|

└─ Stage 2: trojanized `sechost.dll` (632KB, fileless)

SHA256: `352721b32ec1c8349985ceccfec8d1ca6e3e6cc12f83350c4ae1a75477588bc2`

LSA/Credential Manager access, service persistence, ETW manipulation

MITRE ATT&CK

Tactic	Technique	ID	Detail
Defense Evasion	Obfuscated Files or Information	T1027	Five-scheme string encryption, API hashing
Defense Evasion	Masquerading	T1036	Trojanized <code>sechost.dll</code> ; Chrome 117 User-Agent
Defense Evasion	Virtualization/Sandbox Evasion	T1497	Dead loops, chained dispatchers
Defense Evasion	Deobfuscate/Decode Files or Information	T1140	Runtime string decryption
Defense Evasion	Indicator Removal	T1562	ETW manipulation in stage 2
Command and Control	Web Service	T1102	C2 address in Ethereum smart contract
Command and Control	Application Layer Protocol: Web Protocols	T1071.001	HTTP POST via WinHTTP
Command and Control	Non-Standard Port	T1571	Ports 4219, 5902, 6573, 6782, 28313, 48261

Tactic	Technique	ID	Detail
Credential Access	Credentials from Password Stores	T1555	Chrome, Firefox, Credential Manager, LSA
Credential Access	Credentials from Password Stores: Windows Credential Manager	T1555.004	Stage 2: CredReadW , LsaRetrievePrivateData
Collection	Clipboard Data	T1115	GetClipboardData
Collection	Screen Capture	T1113	GDI screen capture
Privilege Escalation	Abuse Elevation Control Mechanism	T1548	RunAsInvoker UAC bypass
Privilege Escalation	Access Token Manipulation: Token Impersonation	T1134.001	SeImpersonatePrivilege via COM
Execution	Command and Scripting Interpreter: PowerShell	T1059.001	powershell -exec bypass
Execution	System Binary Proxy Execution: Rundll32	T1218.011	rundll32
Persistence	Shortcut Modification	T1547.009	LNK manipulation
Persistence	Create or Modify System Process: Windows Service	T1543.003	Stage 2 service installation
Discovery	System Information Discovery	T1082	GetComputerNameA , hardware ID derivation
Discovery	Process Discovery	T1057	Process enumeration to Processes.txt
Discovery	Query Registry	T1012	Steam registry path

YARA Rule

```

rule aachum_REMUS
{
  meta:
    author      = "Izan Perez Perpen (aachum)"
    description = "Detects REMUS infostealer/loader. Covers stage 1 binary and stage 2 memory"
    date        = "2026-03-27"
    sha256_s1   = "cfcb21d8df942918f7a74b99f2cccf7e54e2a6dd1ea6de60897ff0026a26b5c4"
    sha256_s2   = "352721b32ec1c8349985ceccfec8d1ca6e3e6cc12f83350c4ae1a75477588bc2"

```

```
reference = "Tria.ge 260326-sgfdzshv3q"
tlp       = "TLP:WHITE"

strings:
// Stage 2: custom Base64 alphabet (#- instead of +/-)
$b64_alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789#-" ascii

// Internal build marker
$remus_log = "# REMUS LOG" ascii

// Blockchain C2 endpoint (hardcoded)
$rpc = "eth.llamarpc.com" ascii wide

// C2 protocol parameters
$access_token = "access_token=" ascii
$debug_param  = "&debug=" ascii
$step_param   = "access_token=&step=1" ascii

// PST targeting artifact
$pst = "honey@pot.com.pst" ascii

// UAC bypass
$runasinvoker = "__COMPAT_LAYER=RunAsInvoker" ascii

// Execution artifacts
$ps1 = "powershell -exec bypass -f \"\"\" ascii
$rdll = "rundll32 \"\"\" ascii

// Multipart file exfiltration headers
$multipart_file = "Content-Disposition: form-data; name=\"file\"; filename=\"\"\" ascii
$multipart_ct   = "Content-Type: multipart/form-data; boundary=" ascii

// Variable-key XOR decryption loop over UTF-16LE words
$xor_loop = {
    0F B7 44 [2]
    66 [2-8]
    66 31 [1-3]
    FF C? [0-4]
}

// Two-entry jump table dispatch
$dispatch = {
    0F B6 C?
    48 8D [2-6]
    FF 24 C?
}
```

```

// MT19937 recurrence constant
$mt_const = { 65 89 07 6C }

// API hash constants
$hash_winhttpopen = { BA 3E E1 45 0E } // 0xe145d93e WinHttpOpen
$hash_winhttpsend = { BA 81 B5 AC 92 } // 0x92acb581 WinHttpSendRequest
$hash_openmutex   = { BA FE 0C A5 23 } // 0x23a50cfe OpenMutexW

condition:
uint16(0) == 0x5A4D
and uint32(uint32(0x3C)) == 0x00004550
and (
    ($remus_log and $rpc)
    or ($access_token and $debug_param and $pst)
    or ($b64_alpha and $remus_log)
    or ($xor_loop and $dispatch and $mt_const)
    or (2 of ($hash_winhttpopen, $hash_winhttpsend, $hash_openmutex)
        and 1 of ($rpc, $access_token, $remus_log))
    or ($ps1 and $runasinvoker and $multipart_file)
)
}

```

Indicators of Compromise

Network

C2 infrastructure rotates across samples. The following domains and IPs have been observed; not all appear in every infection.

Domain	IP	Port	Samples
adveryx.biz	76.13.17.11	6573	1
coox.live	168.231.114.49	28313	1, 2, 3+
padaz.pics	103.211.219.238	4219	1, 3+
nitroca.biz	62.72.32.156	6782	2, 3+
baxe.pics	65.21.104.235	48261	3+
zadno.run	103.211.219.238	4219	3+
ropea.top	unknown	unknown	3+
chalx.live	62.72.32.156	5902	3+

Domain	IP	Port	Samples
buccstanor.pics	94.231.205.229	48261	3+
gluckcreek.online	94.231.205.229	48261	3+
eth.llamarpc.com	172.67.167.200	443	all

Observed C2 ports across samples: 4219 , 5902 , 6573 , 6782 , 28313 , 48261 .

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.0.0 Safari/537.36

Binary

Type	Value	Note
SHA256	cfc21d8df942918f7a74b99f2cccf7e54e2a6dd1ea6de60897ff0026a26b5c4	Stage 1
SHA256	352721b32ec1c8349985ceccfec8d1ca6e3e6cc12f83350c4ae1a75477588bc2	Stage 2 memory dump
MD5	65b650d78cbf74f17a1f5c139d5ab278	Internal tag= value in stage 1
String	# REMUS LOG	Internal build marker
String	honey@pot.com.pst	Anti-sandbox check

Conclusion

REMUS is a two-stage infostealer and loader with credential theft coverage across Chrome, Firefox, Steam, Outlook, and the Windows clipboard. Blockchain-based C2 resolution keeps the operator's infrastructure effectively out of reach of conventional takedown methods. The second stage extends persistence and anti-forensics capabilities through service installation and ETW manipulation.

Infrastructure patterns across samples point to active deployment: consistent use of non-standard TCP ports, TLD diversity across campaigns (.biz , .live , .pics , .run , .top , .online), and IP reuse suggest a managed, evolving operation rather than one-off activity.

The only unrecovered IOC from this analysis is the Ethereum contract address. Its recovery would enable monitoring of C2 rotations and potential coordination with node operators.

Analysis: FLARE FLOSS, Binary Ninja Personal 5.2, Tria.ge sandbox, manual PCAP analysis.

Source: https://aachum.github.io/website/remus_analysis