Rust-Based Info Stealers Abuse GitHub Codespaces

By Nitesh Surana, Jaromir Horejsi (words)

Published: 2023-05-19 · Archived: 2026-04-05 21:08:04 UTC

Cloud

This is the first part of our security analysis of an information stealer targeting GitHub Codespaces (CS) that discusses how attackers can abuse these cloud services for a variety of malicious activities.

By: Nitesh Surana, Jaromir Horejsi May 19, 2023 Read time: 5 min (1426 words)

Save to Folio

Cloud-based developer environments allow developers to virtually code from anywhere and start right from their smartphones, tablets, or any device with a browser and an internet connection. GitHub Codespace (CS) is one such feature-rich, cloud-based service from Microsoft that enables developers to build software from anywhere.

After its availability was made public in November 2022, any GitHub user could create at least two active CS instances and use them for free with limits on storage, processing power, and duration. CS instances are isolated virtual machines (VMs) hosted on Azure that can be accessed using the web browser, GitHub CLI, or other integrated developer environments (IDEs) such as VSCode and JetBrains, among others. Since any GitHub user could create CS environments, it did not take long for attackers to find ways to abuse this service.

In January 2023, we shared a proof of concept showing how an attacker could abuse a feature allowing the exposure of ports on GitHub CS to deliver malware with open directories. It should be noted that open directories aren't new and threat actors have been documented using these for serving malicious content such as ransomware, exploit kits, malware samples, and the like.

In relation to this, we recently came across Rustlang-based info stealers targeting Windows. Much like the technical details shared in our previous Twitter thread, these info stealers disguised themselves as applications or platforms. Our investigation showed how these info stealers operate by leveraging exposed ports on a CS instance to exfiltrate credentials from an infected machine. In this blog, we detail one of these info stealers masquerading as a popular computer game. This will serve as the first part of the series, to be followed by another entry analyzing how this info stealer is able to persist on the victim machine after it infects an existing installation of Discord.

Overview of functions

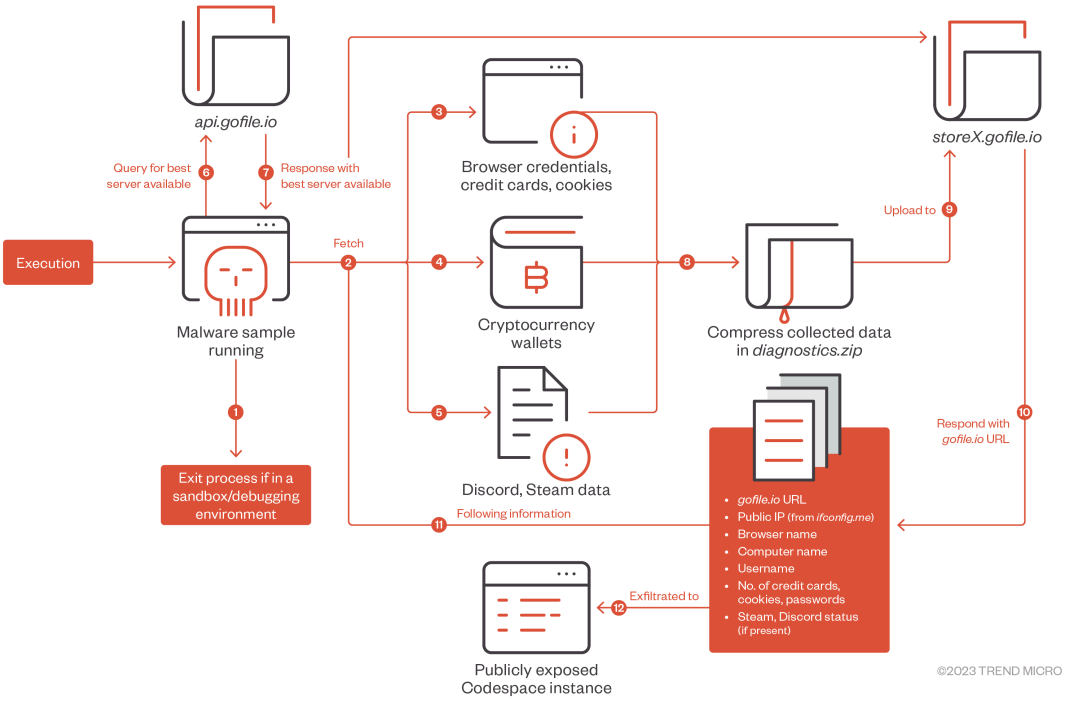

Figure 1. A brief overview of the first section of this info stealer

Analyzing the info stealer sample with a decompiler, we noticed a number of interesting function names, including anti-debugging features and stealing data from web browsers, Discord, Steam, and cryptocurrency wallets, among others.

Figure 2. Suspicious functions (top) and when we decompiled the main function of the sample (bottom)

Functions for anti-debugging and anti-analysis

Initially, the function called *malware::anti_debug::detect::hfc268b042e05af6a()* checks if the sample is running in a controlled environment. The function fetches the username and, later, the current host name to compare it with a list of blocklisted usernames and host namesopen on a new tab that might have been used in sandboxes and debugging environments. If any match is found, the stealer process is terminated. For comparison of a similar method, we found a repository of a Python-based anti-debugger with anti-debugging and anti-analysis procedures implemented.

```
std::env::_var::he40ab40f4ff6571b();
if ( BYTE8(v28) != 3 )
{
  *(_OWORD *)Size = v28;
  *(_OWORD *)Buf2 = v27;
  core::result::unwrap_failed::hc39515d7b13f74e4();
}
v26 = v28;
*(_OWORD *)v25 = v27;                           // Fetch Hostname
whoami::platform::username::hfe34bd2cda1de7c8(Buf2);// Fetch Username
v15 = Buf2[0];
v16 = Size[0];
for ( i = 0i64; i != 54; i += 2i64 )
{
  v18 = (const void *)v11[i];
  if ( !v18 )
    break;
  if ( v11[i + 1] == v16 && !memcmp(v18, v15, v16) )// Compare with a list of blocked usernames
    std::process::exit::h613cee649ef68cef();  // Stop further execution
}
_rust_dealloc();
v19 = v25[0];
v20 = v26;
for ( j = 0i64; j != 56; j += 2i64 )
{
  v22 = (const void *)v13[j];
  if ( !v22 )
    break;
  if ( v13[j + 1] == v20 && !memcmp(v22, v19, v20) )// Compare with a list of blocked hostnames
    std::process::exit::h613cee649ef68cef();  // Stop further execution
}
```

Figure 3. Anti-debug checks implemented by the stealer

Stolen information breakdown

In this section, we enumerate the stolen data and processes we found from the infection routine of the info stealer malware.

Stealing browser data

Once anti-debug checks are done and no sandbox or anti-debug environment is detected, the stealer collects the credentials stored in the victim machine, such as passwords, cookies, and credit card information in the following popular web browsers:

- 360Browser
- Amigo
- Brave
- Chromodo
- Chromunium (sic)

- CocCoc
- Comodo
- Epic Privacy Browser
- Google Chrome
- K-Melon
- Kometa
- Mail.Ru
- Maxthon3
- Nichrome
- Orbitum
- Slimjet
- Sputnik
- Torch
- Uran
- Vivaldi
- Yandex

We observed that “Chromunium” is a typo of “Chromium,” and it does not work. Neither did we find any public mentions of “Chromunium” being a browser. Notably, majority of modern browser codebases are based on Chromium, a free and open-source project, including Microsoft Edge even if it is not found in the stealer’s list for checking.

While analyzing the function *malware::browsers::steal_data::h8cac638d5caa2249()*, however, we also noticed mentions of a function called *get_chromunium_targets*. In an attempt to look for a related stealer code on GitHub, we came across a repository containing a source code in Rust language, which we examined to be an info stealer sending stolen information to the attacker’s webhook. Based on the similarities of the function code, sequence of browsers, and applications being targeted, the info stealer analyzed in this blog post was likely based on or inspired by the stealer we discovered in the GitHub repository.

```
mov     rcx, 0EC21713F4675F68Dh
xor     rdi, rcx
mov     rcx, 0D58A186E226DC091h
xor     rbx, rcx
mov     [rax], rbx
mov     [rax+8], rdi
mov     qword ptr [rsp+0CB8h+var_698], rax
movups  [rsp+0CB8h+var_698+8], xmm11
lea     rax, loc_B1A420+3
mov     qword ptr [rsp+0CB8h+var_C28], rax
mov     rax, qword ptr [rsp+0CB8h+var_C28]
mov     rcx, 0FFFFFFFF92CD62A2h
xor     rcx, cs:_ZN7malware8browsers22get_chromunium_targets23_XREF_STATIC_MUT_OFFSET17h37e490abe9072e5dE
mov     rdx, rcx
shr     rdx, 0Ch
xor     rdx, rcx
shld    rdx, rcx, 1
rol     rdx, 6
movzx   ecx, dx
mov     rbx, [rax+rcx]
mov     edi, [rax+rcx+8]
movzx   esi, word ptr [rax+rcx+0Ch]
mov     ecx, 0Eh
mov     edx, 1
call    __rust_alloc
test    rax, rax
jz      loc_407036
```

Figure 4. Calling a function named “get_chromunium_targets” in one of the methods from the info stealer

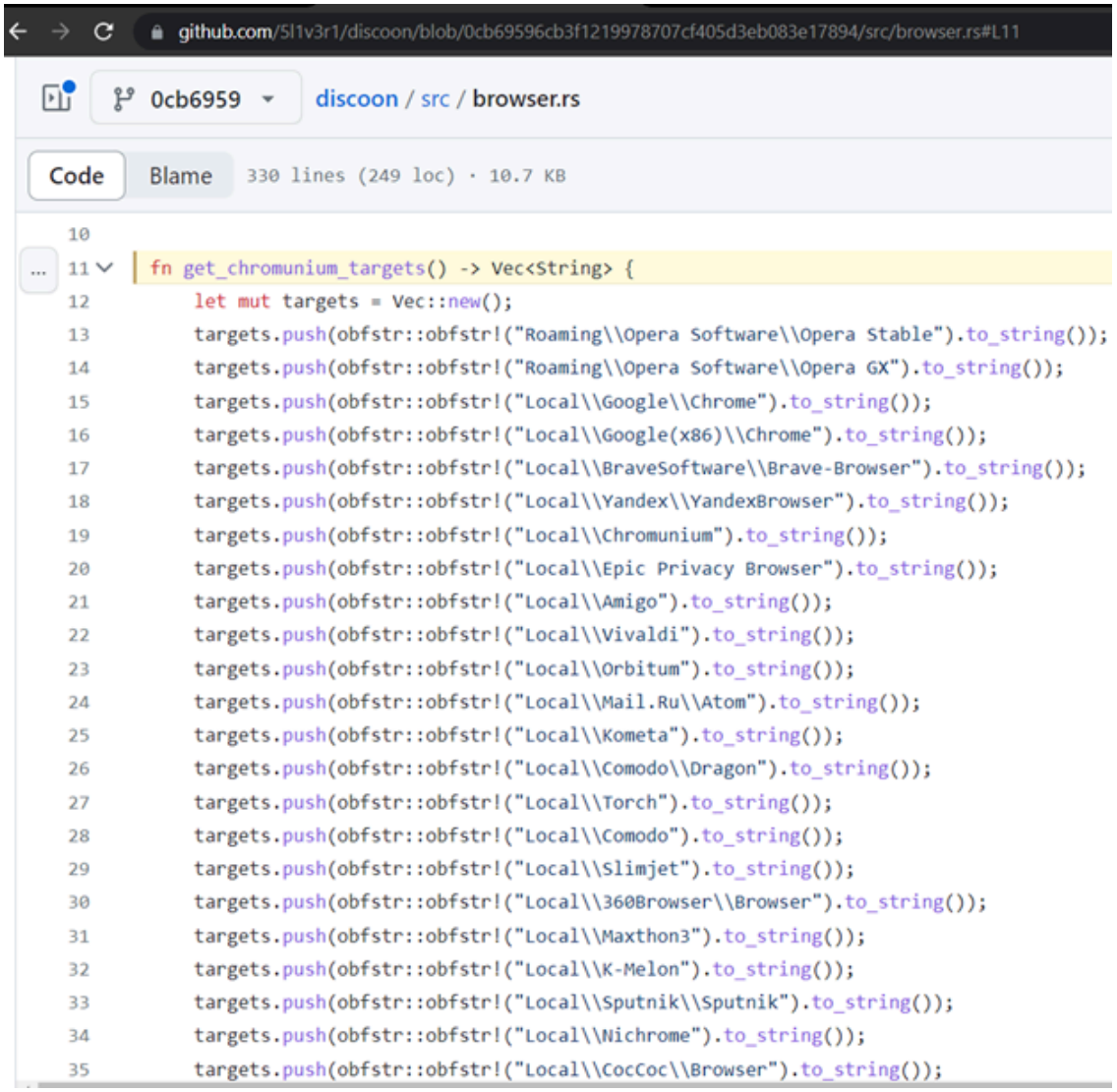

Figure 5. Possible source code related to the info stealer based on function name and capabilities

Meanwhile, the collected credentials for each targeted browser are saved under the following files:

- *%localappdata%\Microsoft\Security\Browsers\<browser_name>\Default\Passwords.tx*
- *%localappdata%\Microsoft\Security\Browsers\<browser_name>\Default\Netscape Cookies.txt*
- *%localappdata%\Microsoft\Security\Browsers\<browser_name>\Default\Credit Cards.txt*

Stealing cryptocurrency wallet data

After collecting the browser credentials, the stealer proceeds to steal information from various cryptocurrency wallets. It then targets known wallets from the paths under the *<%localappdata%>* and *<%appdata%>* folders, as identified here:

- *\Armory*
- *\atomic\Local Storage\leveldb*
- *\bytecoin*
- *\Coinomi\Coinomi\wallets*
- *\com.liberty.jaxx\IndexedDB\file__0.indexeddb.leveldb*

- *\Electrum\wallets*
- *\Ethereum\keystore*
- *\Exodus\exodus.wallet*
- *\Guarda\Local Storage\leveldb*
- *\Zcash*

Stealing Discord data

The stealer also targets the messaging application Discord and looks for Discord tokens. These tokens allow malicious actors to impersonate the victims on the platform once acquired. Once the token is found, it is written to the file *Discord Tokens.txt* located in *<%localappdata%\Microsoft\Security>*. The tokens are scanned from the following paths:

- *%appdata%\discord*
- *%appdata%\discord\Local Storage\leveldb*
- *%appdata%\discordcanary*
- *%appdata%\discordptb*
- *%appdata%\discorddevelopement*
- *%localappdata%\Discord*

Stealing Steam data

The Steam configuration files from *<%programfiles(x86)%\Steam\config\>* are copied to the folder *<%localappdata%\Microsoft\Security\Steam\>* for later exfiltration. Stolen credentials and configuration files are stored in the following paths and files:

- *%localappdata%\Microsoft\Security\Browsers*
- *%localappdata%\Microsoft\Security\Wallets*
- *%localappdata%\Microsoft\Security\Steam*
- *%localappdata%\Microsoft\Security\Discord Tokens.txt*

Exfiltration

The previously collected files are compressed into a file named *diagnostics.zip* and stored in the path *<%localappdata%\Microsoft\diagnostics.zip>*. The stealer uses gofile.io, a file-sharing platform that allows users to upload and share files anonymously. Initially, the stealer fetches the best available gofile.io server by querying *api.gofile.io*. Depending on the response, the best server to send files to or receive files from is used in the subsequent request in the format *storeX.gofile.io*, where “X” is a number (such as “store2” in Figure 6).

The stealer then uploads the compressed file via a POST request to the endpoint */uploadFile*. The body of the POST request contains the collected credentials from the victim.

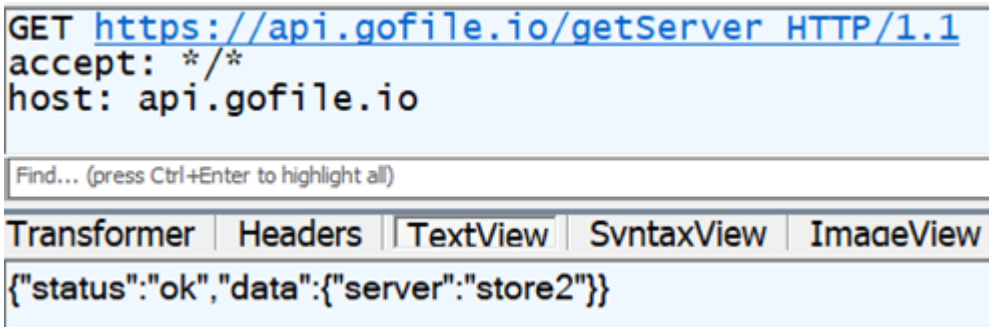

Figure 6. Requesting the best server for upload

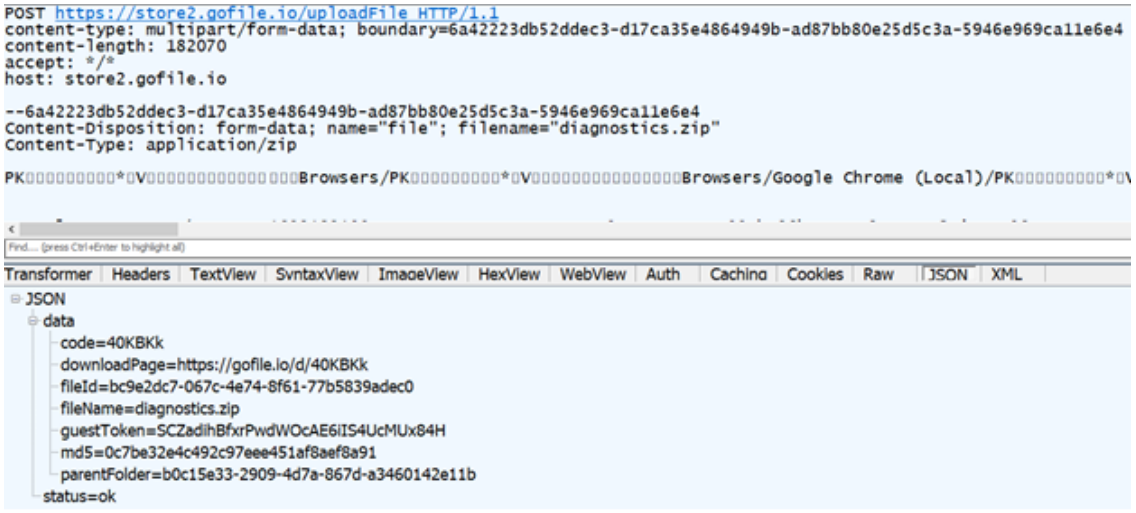

Figure 7. Uploading stolen credentials to the gofile server

In the response, we get the gofile.io URL where the uploaded file is stored. This URL can be accessed by anyone without any authentication. We also get a token in the *guestToken* parameter, which can be used by the uploader to delete the *parentFolder* and *fileId* parameters subsequently. After the gofile.io upload is complete, the query *ifconfig.me* fetches the public IP address of the victim machine.

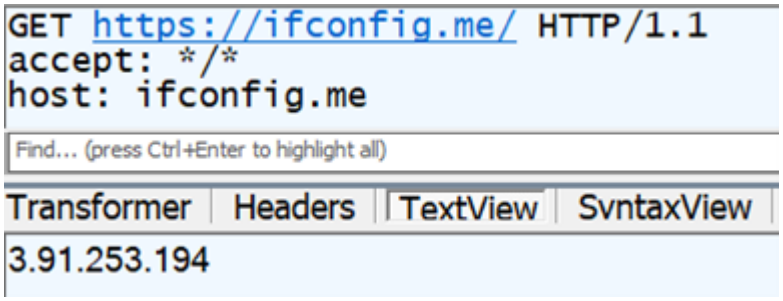

Figure 8. HTTP GET request to get the public IP address of the infected machine

The last step is submitting the stolen information to the Github webhook controlled by the attacker. This is the summary of stolen information exfiltrated by the stealer:

1. List of browsers found
2. Computer name

3. Number of cookies extracted
4. Total number of credit cards extracted
5. Discord status (if Discord is installed or not)
6. Number of passwords extracted
7. Uploaded *gofile.io* URL of *diagnostics.zip*
8. Steam status (if any Steam data was stolen or not)
9. Username of the user running the info stealer
10. List of cryptocurrency wallets extracted
11. Windows operating system version

The stealer then embeds all the pieces of information about the victim into a JSON file and sends this via a POST request to a GitHub CS URL. We saw a POST request attempting to exfiltrate the stolen information to the Github CS endpoint that listens at port 8080. Had the CS been active, port 8080 would have been publicly exposed and, requiring no authentication, the exfiltrated information would have been successfully sent to and received by the attacker.

According to our sample and testing, the exfiltration of the data to the webhook had failed with the status error "302 Moved Temporarily." If we try to access the gofile.io URL, we will see that the file *diagnostics.zip* has been uploaded to the server and can be downloaded by anyone with the URL link because no authorization is required.

Figure 9. Failed exfiltration of stolen data to the Github Codespaces webhook

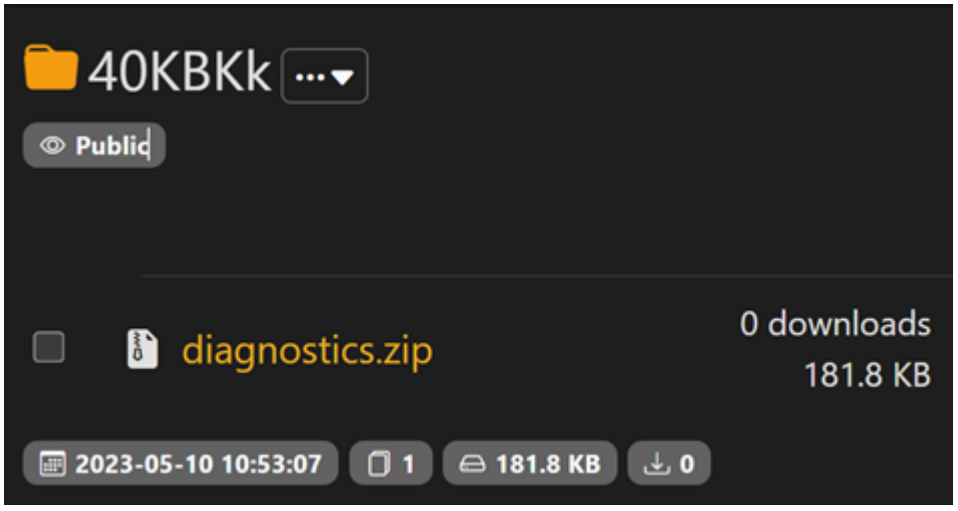

Figure 10. Uploaded file to gofile.io

In the second part of this analysis, we detail our investigation of how this information-stealing malware achieves persistence in the infected machine by modifying the victim's installation of Discord. We also enumerate our security recommendations and insights on how users and security teams can defend their networks and endpoints against this growing threat.

Indicators of Compromise (IOCs)

Download the full list of indicators hereopen on a new tab.

Tags
