2024
DUBLIN

2 - 4 October, 2024 / Dublin, Ireland

FROM CODE TO CRIME: EXPLORING THREATS IN GITHUB CODESPACES

Nitesh Surana & Jaromír Hořejší
Trend Micro

ABSTRACT

Cloud-based development environments enable developers to work from any device with internet access. Introduced for all GitHub users during the GitHub Universe event in November 2022, *Codespaces* offers a customizable cloud-based IDE, simplifying project development. This cloud-based IDE allows developers and organizations to customize projects by using configuration-as-code features, easing some previous pain points in project development such as environment setups and dependency hell, amongst others.

Thanks to the low threshold for creating Codespaces – which can be done by any *GitHub* user – it didn't take long to observe abuse of this service. We found *GitHub Codespaces* being used by threat actor(s) to develop, test and host infostealer malware written in Rust, NodeJS and C++ targeting *Windows* platforms.

INTRODUCTION

A Codespace is a Docker container that runs on an isolated virtual machine on *Microsoft Azure*. Codespaces cannot be self-hosted. Additionally, any *GitHub* user can create two Codespace instances for free for up to 120 core hours and 15GB of storage.

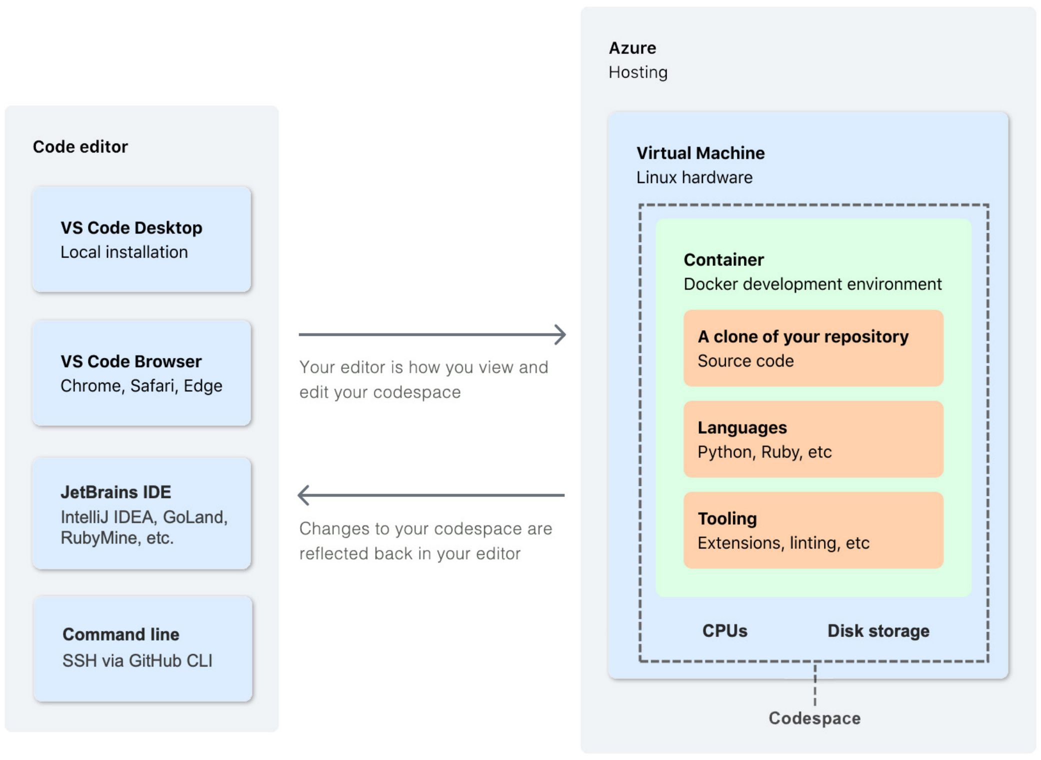


Figure 1: Overview of GitHub Codespaces.
Source: https://docs.github.com/en/codespaces/overview

Codespaces come with tools, libraries, and extensions installed by default. For a consistent environment, Codespaces can be created using configurations in the JSONC format (JSON with Comments), known as development container [1] configurations. Dev containers come with powerful features such as postStartCommand and postCreateCommand, which allow for arbitrary command execution in development environments. To aid developers in testing applications, *GitHub* allows users to expose or forward ports on the Codespace. Codespaces can be deployed to different regions based on the user's preference. The default (and the maximum) retention period for a Codespace is 30 days and the maximum idle timeout is four hours.

It is important to note that, although Codespaces are privileged Docker containers, no two Codespaces share the same Azure VM. A container escape can be done on GitHub Codespaces, but there is no immediate risk associated.

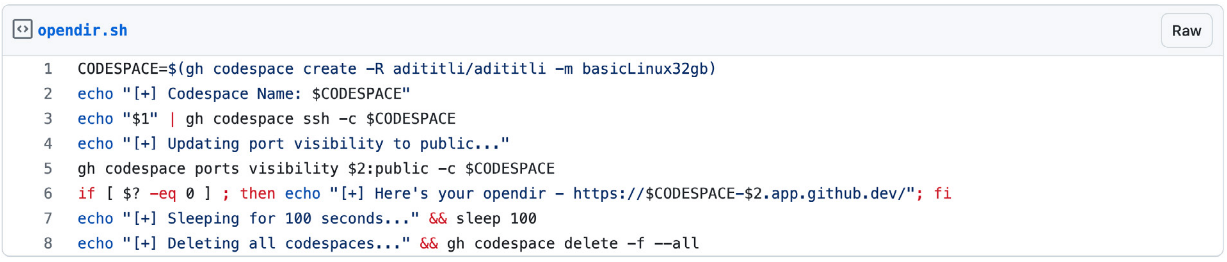
We think such services are important to investigate for the following reasons:

1. The bar to create a *GitHub* Codespace is quite low and ease of use is high, as anyone with a valid *GitHub* account can create two VMs across multiple regions.
2. Unlike public cloud providers like *Azure*, *AWS* and *GCP*, no payment information is required to spin up *Codespaces* since this service is free, with limits on number of hours of compute and storage configurations.
3. To create a *GitHub* account, one can use temporary email services and can create Codespaces without using any personally identifiable information.

4. Developer-centric features like dev containers allow for creation of consistent environments using configuration files, i.e. attackers could have their entire environments defined in a JSONC file and could easily replicate environments.

In a blog post [2] published in January 2023 we explored how one could create malware file servers using dev containers and *GitHub Codespaces*. Codespaces allow for exposing ports on the dev container in specific modes. One of the modes is 'public', using which, the port can be exposed over the internet without any authentication context required.

From a developer's perspective, this is a helpful feature to test and debug applications in real time. However, this is interesting as open directories are frequently abused to deliver malware in environments with traditional detection/protection mechanisms (e.g. IP reputation checks). Since the domain created while exposing a port publicly is a randomized subdomain of *GitHub* (github.dev), it would bypass modern solutions (unless traffic inspection is being performed). Using dev containers with *GitHub Codespaces*, one could automate creation of malware file servers.



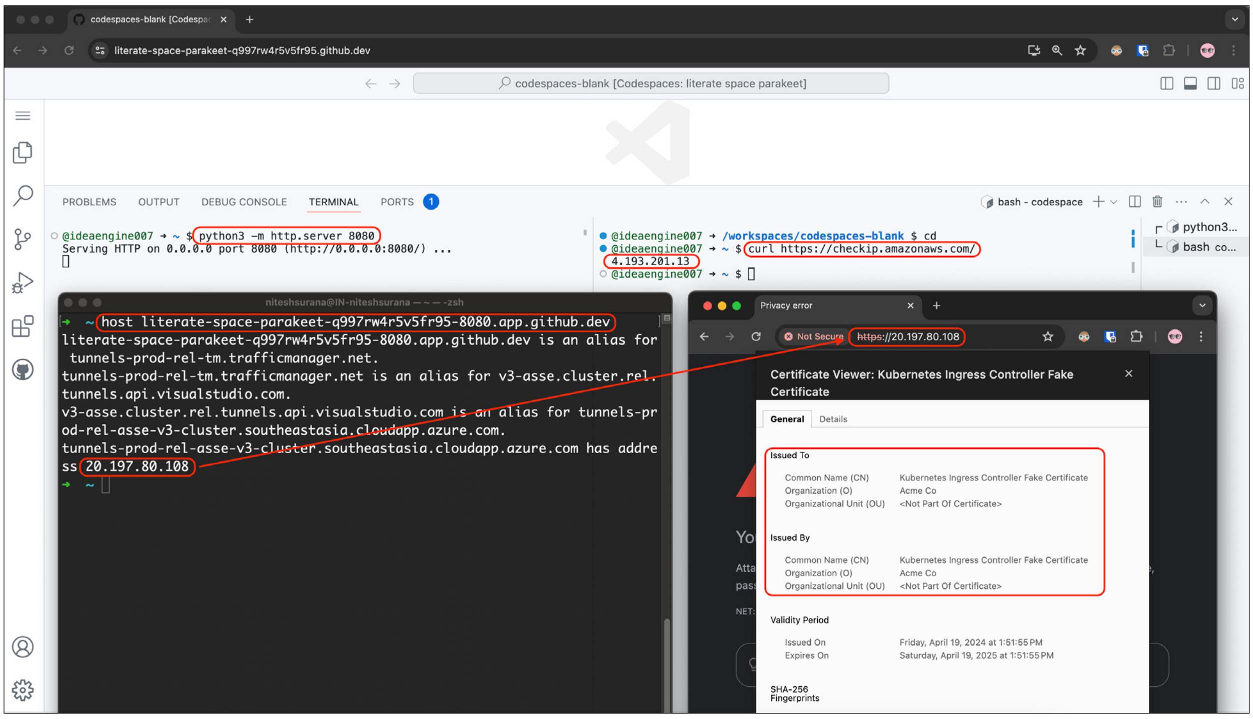
```
CODESPACE=$(gh codespace create -R adititli/adititli -m basicLinux32gb)
echo "[+] Codespace Name: $CODESPACE"
echo "$1" | gh codespace ssh -c $CODESPACE
echo "[+] Updating port visibility to public..."
gh codespace ports visibility $2:public -c $CODESPACE
if [ $? -eq 0 ] ; then echo "[+] Here's your opendir - https://$CODESPACE-$2.app.github.dev/"; fi
echo "[+] Sleeping for 100 seconds..." && sleep 100
echo "[+] Deleting all codespaces..." && gh codespace delete -f --all
```

Figure 2: A simple script to create a Codespaces-based open directory.

You can get the source code of the above-mentioned script in [3] and see a proof-of-concept video in [4]. More details are included in the blog post [2].

STEALTHY OPEN DIRECTORIES & C2

Our exploration led us to an interesting observation when attempting to hunt for exposed ports on *Codespaces* at scale. We noticed that we couldn't access an exposed port on a Codespace via its public IP address. Instead of directly reaching Codespace, we encountered the Kubernetes Ingress Controller inline, as indicated by the SSL certificate and HTTP response. This controller functions as both a reverse proxy and a load balancer, adding an additional layer of traffic routing abstraction. As a result, Codespace itself cannot be accessed solely through its IP address.


codespaces-blank [Codespaces
literate-space-parakeet-q997rw4r5v5fr95.github.dev
codespaces-blank [Codespaces: literate space parakeet]
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
bash - codespace
python3...
bash co...
niteshsurana@IN-niteshsurana — ~ — -zsh
Privacy error
Not Secure
https://20.197.80.108

```
@ideaengine007 → ~ $ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

```
@ideaengine007 → /workspaces/codespaces-blank $ cd
@ideaengine007 → ~ $ curl https://checkip.amazonaws.com/
4.193.201.13
@ideaengine007 → ~ $
```

```
→  ~ host literate-space-parakeet-q997rw4r5v5fr95-8080.app.github.dev
literate-space-parakeet-q997rw4r5v5fr95-8080.app.github.dev is an alias for
 tunnels-prod-rel-tm.trafficmanager.net.
tunnels-prod-rel-tm.trafficmanager.net is an alias for v3-asse.cluster.rel.
tunnels.api.visualstudio.com.
v3-asse.cluster.rel.tunnels.api.visualstudio.com is an alias for tunnels-pr
od-rel-asse-v3-cluster.southeastasia.cloudapp.azure.com.
tunnels-prod-rel-asse-v3-cluster.southeastasia.cloudapp.azure.com has addre
ss 20.197.80.108
→  ~
```

Certificate Viewer: Kubernetes Ingress Controller Fake Certificate

General | Details

Issued To

Common Name (CN)	Kubernetes Ingress Controller Fake Certificate
Organization (O)	Acme Co
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	Kubernetes Ingress Controller Fake Certificate
Organization (O)	Acme Co
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	Friday, April 19, 2024 at 1:51:55 PM
Expires On	Saturday, April 19, 2025 at 1:51:55 PM

SHA-256 Fingerprints

Figure 3: Codespace-based open directory cannot be accessed by its IP address.

Exposed ports on a Codespace have the following format: '<codespace-name>-<port>.app.github.dev'. Here, the 'codespace-name' is not predictable by an end-user. In the above picture, we first create a python-based HTTP server listening on port 8080 and forward this port to be publicly accessible. Later, we fetch the IP address of the Codespace by running ‘curl ident.me’. Now, when we try to fetch the IP address of the URL which allows access to the exposed port, we come across an IP address that is different from the Codespaces’ IP address. Finally, we confirm that this IP address is not of the Codespace, but it is of an inline ingress controller that acts as a reverse proxy.

This finding holds significance since it implies that threat intelligence search engines, such as *Shodan*, *Censys* and *BinaryEdge*, cannot index maliciously exposed Codespace instances unless the URLs are already known to be suspicious/malicious. Additionally, in our internal simulations, we were able to use *Codespaces* as a command-and-control server for adversary emulation frameworks.

Typically, when attempting to identify and investigate threat actor infrastructure, researchers pivot and search for artifacts like default SSL certificates and favicon hashes using search engines. However, with the knowledge of this abuse method involving *Codespaces*, threat actors can effectively avoid being fingerprinted, rendering their operations more elusive and stealthier than ever.

EMERGENCE OF INFOSTEALER CAMPAIGNS

During our regular course of threat hunting, we came across a few malicious binaries communicating to domains that resembled exposed ports on a *GitHub* Codespace. Upon further digging, we found that these were infostealers written in various high-level programming languages, such as Rust, NodeJS and C++, masquerading as legitimate applications such as computer games or hacking/cheating utilities. The malicious samples only targeted *Windows* platforms.

Some of the names under which the malicious samples were distributed are listed below:

- Discord Account Generator.exe
- Adventure Island Setup.exe
- Neus Setup.exe
- Cheat Fortnite.exe
- BattleTalent-LaserEyes.exe

ANALYSIS OF THE INFOSTEALER ABUSING GITHUB CODESPACES

Rust variant

The flow of activities performed when the infostealer executes is shown in Figure 4.

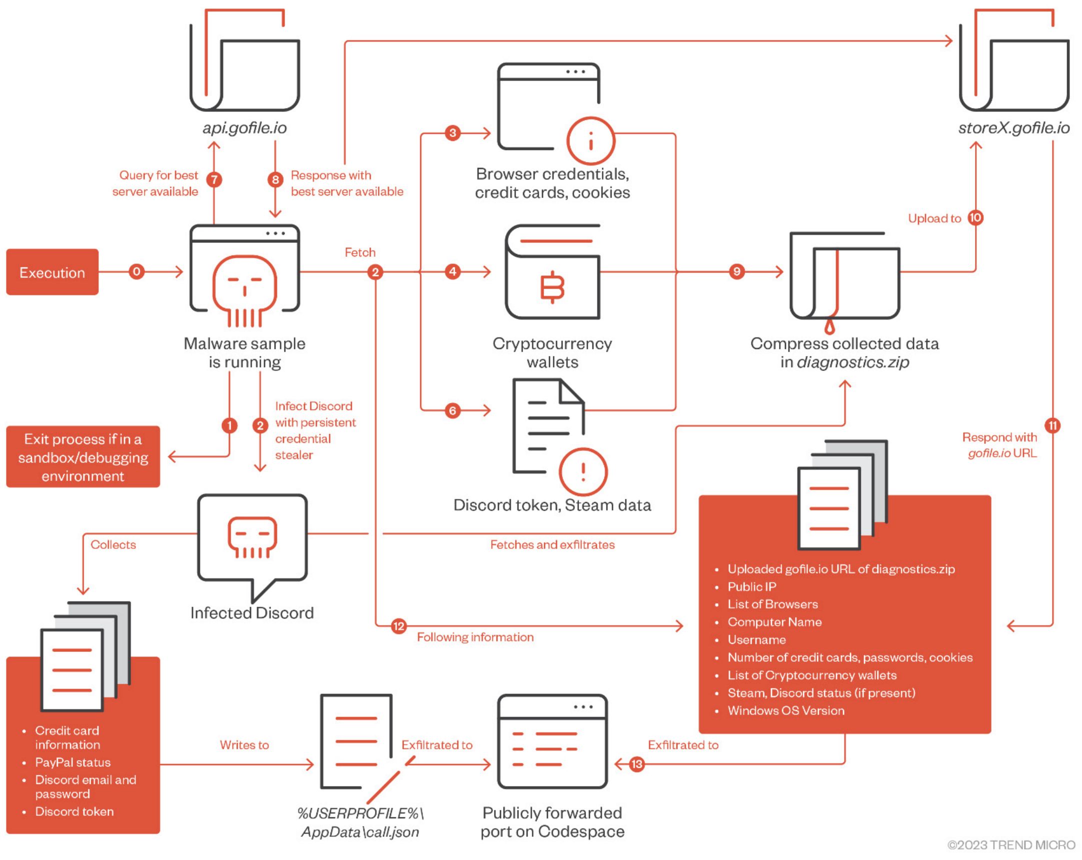
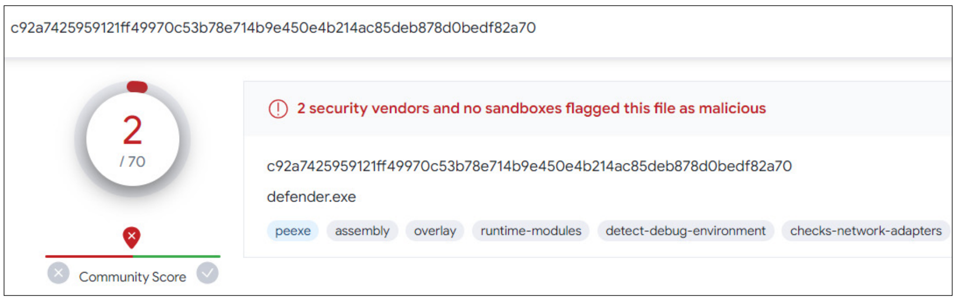

api.gofile.io
storeX.gofile.io
7 Query for best server available
8 Response with best server available
3
Browser credentials, credit cards, cookies
Upload to 10
Execution
0
Fetch 2
4
Cryptocurrency wallets
9
Compress collected data in diagnostics.zip
Malware sample is running
6
Discord token, Steam data
1
Exit process if in a sandbox/debugging environment
2 Infect Discord with persistent credential stealer
Respond with gofile.io URL 11
Collects
Infected Discord
Fetches and exfiltrates
12 Following information
Uploaded gofile.io URL of diagnostics.zip
Public IP
List of Browsers
Computer Name
Username
Number of credit cards, passwords, cookies
List of Cryptocurrency wallets
Steam, Discord status (if present)
Windows OS Version
Credit card information
PayPal status
Discord email and password
Discord token
Writes to
%USERPROFILE%\AppData\call.json
Exfiltrated to
Publicly forwarded port on Codespace
Exfiltrated to 13
©2023 TREND MICRO

Figure 4: Flow of infostealer’s execution.

1. The malware checks if it is running in a sandbox or debugger.
2. The malware patches the *Discord* client, if installed on the infected machine.
3. The malware steals browser credentials and cookies.
4. The malware steals cryptocurrency wallets.
5. The malware steals *Discord* tokens and *Steam* data from web browsers.
6. The malware contacts a third-party cloud storage service to query the best available server.
7. The malware packs the stolen information and uploads it to the third-party cloud storage server.
8. The third-party cloud service responds with a link to the uploaded file. This link, together with additional infected machine information, is exfiltrated to a webhook hosted on *Codespaces*.
9. The infected *Discord* messenger listens for sensitive information and exfiltrates it to the *Codespaces* webhook.

Sandbox evasion

The infostealer checks if it is being run in a controlled environment such as a sandbox. It compares the username and hostname of the machine with a hard-coded list of known usernames and hostnames of sandboxes. For instance, one of the infostealers had extremely low detections on *VirusTotal* on its initial scan:



c92a7425959121ff49970c53b78e714b9e450e4b214ac85deb878d0bedf82a70

2 / 70

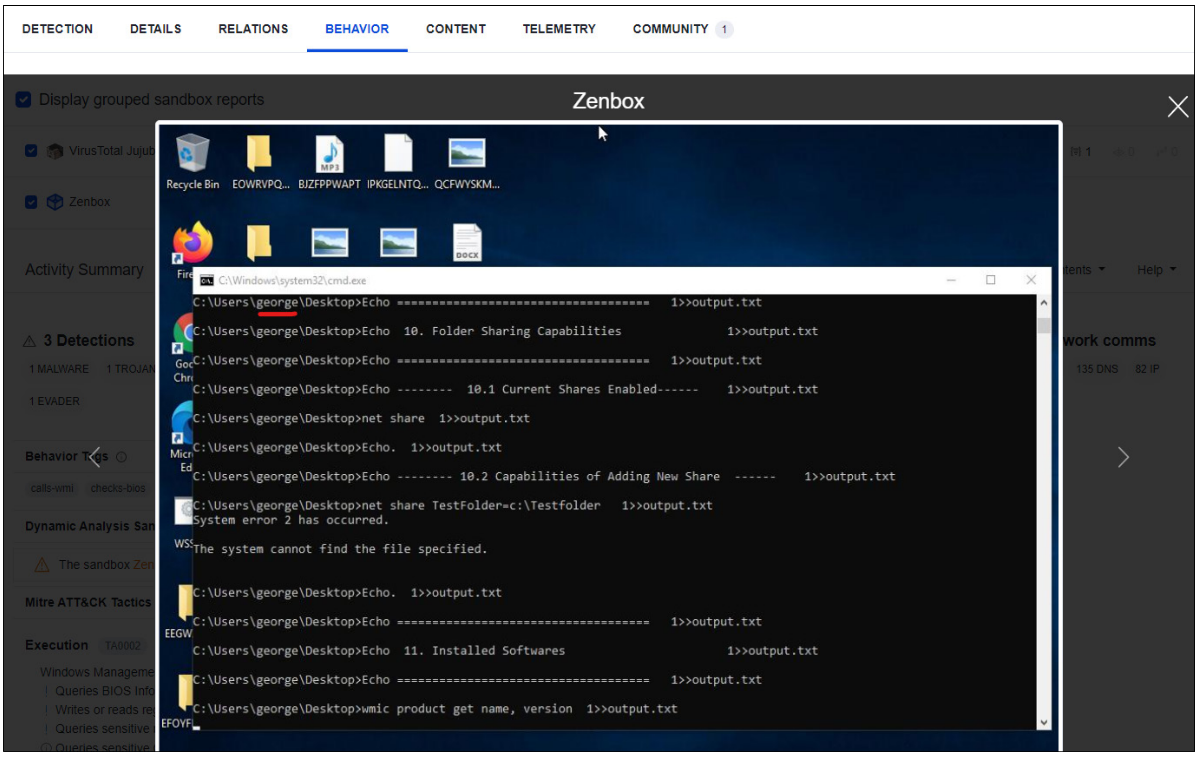
2 security vendors and no sandboxes flagged this file as malicious

c92a7425959121ff49970c53b78e714b9e450e4b214ac85deb878d0bedf82a70

defender.exe

Figure 5: Early VirusTotal detections for an infostealer sample.

Later, we found that *VirusTotal*'s in-house *Windows 10* sandbox, named 'Zenbox', had the hard-coded username 'george', which was being checked by the infostealer upon execution. Such techniques of sandbox evasion are well known among infostealer malware and are documented on *GitHub*.



```
C:\Users\george\Desktop>Echo ==================================   1>>output.txt

C:\Users\george\Desktop>Echo  10. Folder Sharing Capabilities             1>>output.txt

C:\Users\george\Desktop>Echo ==================================   1>>output.txt

C:\Users\george\Desktop>Echo --------  10.1 Current Shares Enabled------    1>>output.txt

C:\Users\george\Desktop>net share  1>>output.txt

C:\Users\george\Desktop>Echo.  1>>output.txt

C:\Users\george\Desktop>Echo -------- 10.2 Capabilities of Adding New Share  ------    1>>output.txt

C:\Users\george\Desktop>net share TestFolder=c:\Testfolder   1>>output.txt
System error 2 has occurred.

The system cannot find the file specified.


C:\Users\george\Desktop>Echo.  1>>output.txt

C:\Users\george\Desktop>Echo ==================================   1>>output.txt

C:\Users\george\Desktop>Echo  11. Installed Softwares                     1>>output.txt

C:\Users\george\Desktop>Echo ==================================   1>>output.txt

C:\Users\george\Desktop>wmic product get name, version  1>>output.txt
```

Figure 6: Hard-coded user named 'george' in VirusTotal's Zenbox Windows 10 sandbox.

Credential gathering

The infostealer collects passwords, cookies and credit card information from a plethora of *Chromium*-based browsers (see Appendix). While validating the browsers from the list, we observed that 'Chromunium' is a typo of 'Chromium'. Notably, most modern browsers are based on the *Chromium* project. This typo helped us pivot and discover credential-stealing malware from *Chromium*-based browsers and *GitHub* repos that the infostealer is potentially linked to. Other than the browsers, the infostealer targets cryptocurrency wallets (see Appendix), and clients of *Discord* (the widely used chat application written using the Electron framework) and popular gaming platform *Steam*.

The collected credentials are exfiltrated to gofile.io, a legitimate third-party file-sharing platform that allows users to upload and share files anonymously. Later, the gofile.io response containing the link to the exfiltrated credentials is exfiltrated, along with the information listed below, to a GitHub Codespace-based webhook:

1. List of browsers found.
2. Computer hostname.
3. Number of cookies extracted.
4. 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.

It is worth noting that the webhook is essentially an exposed port on a Codespace. Since the infostealer samples were exfiltrating the above information to an exposed port on a Codespace, this indicates either that the infostealer authors were using *GitHub Codespaces* as their webhook for malicious operations or that they were testing the infostealers' capabilities, adapting to how modern software development is carried out.

Infecting Discord application

The infostealer gains persistence in the victim's environment by patching the *Discord* application. It does so by modifying the ASAR archive (a special format used by applications developed in the Electron framework, which contains the application's source code) of the *Discord* application in a way that lowers the security of authentication. The patch also includes a piece of malicious code, which logs sensitive information like login usernames, passwords, credit cards, *PayPal* details, events of changing password, etc. This patched client terminates any outbound WebSocket connections and disables the password-less login mechanism (which is done using a QR code). As you can see in Figure 7, the QR code is never loaded and the victim is forced to enter their email or phone number and password.

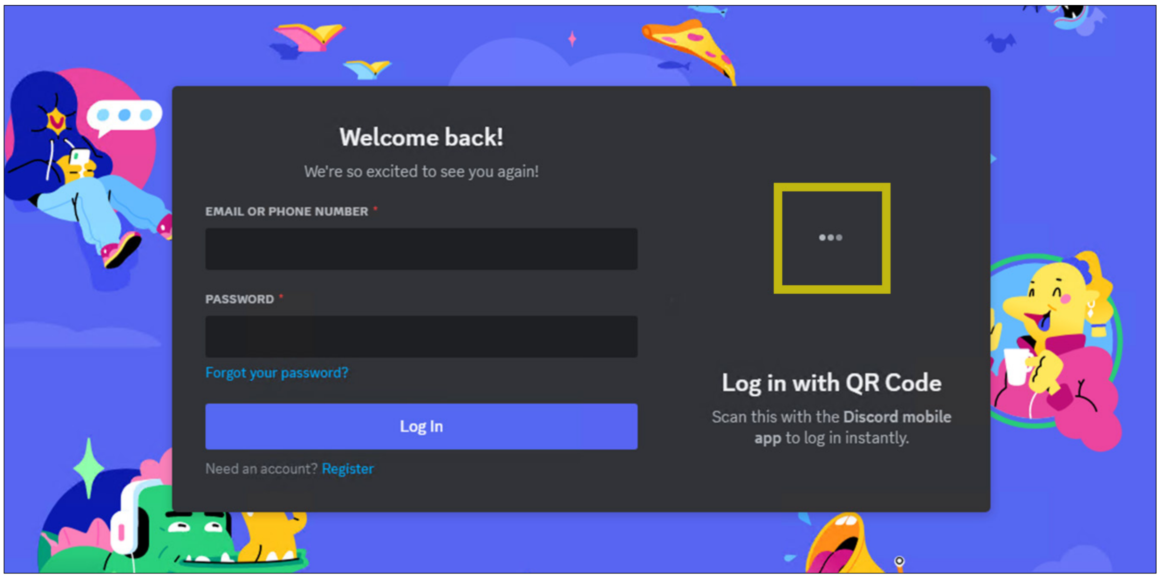
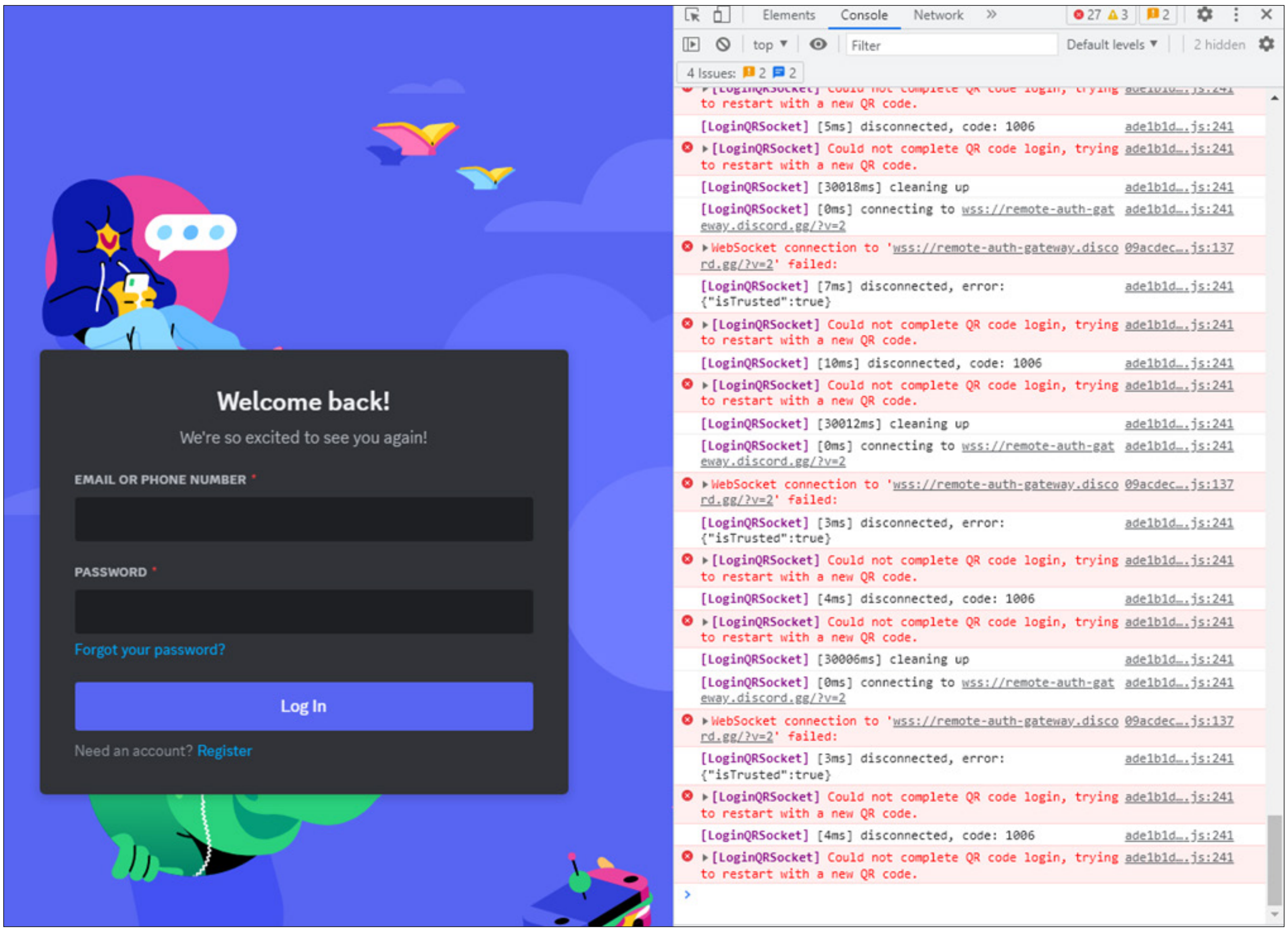

Welcome back!
We're so excited to see you again!
EMAIL OR PHONE NUMBER *
PASSWORD *
Forgot your password?
Log In
Need an account? Register
Log in with QR Code
Scan this with the Discord mobile app to log in instantly.

Figure 7: QR code stuck at loading after infection.

Figure 8 shows a debugging console in the web browser. As WebSocket communication is terminated by the patch, the error message in red states that QR code login could not be completed.



```
to restart with a new QR code.
[LoginQRSocket] [5ms] disconnected, code: 1006                    ade1b1d….js:241
▸[LoginQRSocket] Could not complete QR code login, trying ade1b1d….js:241
to restart with a new QR code.
[LoginQRSocket] [30018ms] cleaning up                             ade1b1d….js:241
[LoginQRSocket] [0ms] connecting to wss://remote-auth-gat ade1b1d….js:241
eway.discord.gg/?v=2
▸WebSocket connection to 'wss://remote-auth-gateway.disco 09acdec….js:137
rd.gg/?v=2' failed:
[LoginQRSocket] [7ms] disconnected, error:                        ade1b1d….js:241
{"isTrusted":true}
▸[LoginQRSocket] Could not complete QR code login, trying ade1b1d….js:241
to restart with a new QR code.
[LoginQRSocket] [10ms] disconnected, code: 1006                   ade1b1d….js:241
▸[LoginQRSocket] Could not complete QR code login, trying ade1b1d….js:241
to restart with a new QR code.
[LoginQRSocket] [30012ms] cleaning up                             ade1b1d….js:241
[LoginQRSocket] [0ms] connecting to wss://remote-auth-gat ade1b1d….js:241
eway.discord.gg/?v=2
▸WebSocket connection to 'wss://remote-auth-gateway.disco 09acdec….js:137
rd.gg/?v=2' failed:
[LoginQRSocket] [3ms] disconnected, error:                        ade1b1d….js:241
{"isTrusted":true}
▸[LoginQRSocket] Could not complete QR code login, trying ade1b1d….js:241
to restart with a new QR code.
[LoginQRSocket] [4ms] disconnected, code: 1006                    ade1b1d….js:241
▸[LoginQRSocket] Could not complete QR code login, trying ade1b1d….js:241
to restart with a new QR code.
[LoginQRSocket] [30006ms] cleaning up                             ade1b1d….js:241
[LoginQRSocket] [0ms] connecting to wss://remote-auth-gat ade1b1d….js:241
eway.discord.gg/?v=2
▸WebSocket connection to 'wss://remote-auth-gateway.disco 09acdec….js:137
rd.gg/?v=2' failed:
[LoginQRSocket] [3ms] disconnected, error:                        ade1b1d….js:241
{"isTrusted":true}
▸[LoginQRSocket] Could not complete QR code login, trying ade1b1d….js:241
to restart with a new QR code.
[LoginQRSocket] [4ms] disconnected, code: 1006                    ade1b1d….js:241
▸[LoginQRSocket] Could not complete QR code login, trying ade1b1d….js:241
to restart with a new QR code.
>
```

Figure 8: Failed outbound WebSocket requests from Discord client.

Since the QR code never loads, the victim is forced to use their credentials as-is, which are again exfiltrated to an attacker-controlled Codespace. The following artifacts are exfiltrated:

1. *Discord* Token
2. *Discord* login credentials (email address/phone number and password)
3. Updated credentials
4. Credit card details
5. *PayPal* details

More detailed analysis can be found in [5] and [6].

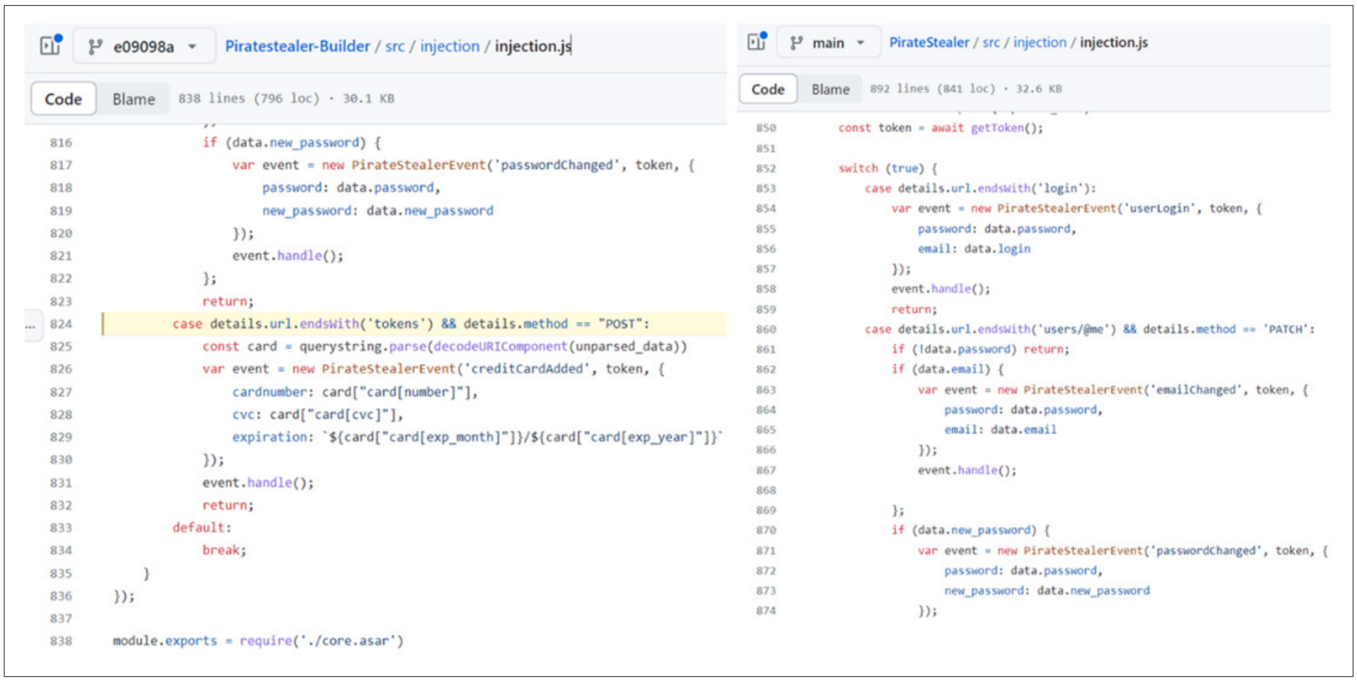
Similarities to other stealers

When searching for related malware code, we noticed that the infostealer has similarities with PirateStealer [7]. This is not surprising, as the source code of malware is often shared and reused. It makes sense for malware developers to reuse existing code (if it at least partially satisfies their requirements) and add new features or modifications where necessary.

Figure 9 shows similarities in the infected *Discord* client with PirateStealer on *GitHub*.

Electron dropper

The Electron variant (developed in the Electron framework) serves as a dropper or installer, which decrypts and runs the previously described Rust variant of the stealer. Applications developed in the Electron framework are usually 'big' projects, containing many files and one specific archive called 'app.asar', containing the source code of the application. More information about Electron-based applications can be found in [8].



e09098a Piratestealer-Builder / src / injection / injection.js

Code Blame 838 lines (796 loc) · 30.1 KB

```
                if (data.new_password) {
                    var event = new PirateStealerEvent('passwordChanged', token, {
                        password: data.password,
                        new_password: data.new_password
                    });
                    event.handle();
                };
                return;
            case details.url.endsWith('tokens') && details.method == "POST":
                const card = querystring.parse(decodeURIComponent(unparsed_data))
                var event = new PirateStealerEvent('creditCardAdded', token, {
                    cardnumber: card["card[number]"],
                    cvc: card["card[cvc]"],
                    expiration: `${card["card[exp_month]"]}/${card["card[exp_year]"]}`
                });
                event.handle();
                return;
            default:
                break;
        }
    });

module.exports = require('./core.asar')
```

main PirateStealer / src / injection / injection.js

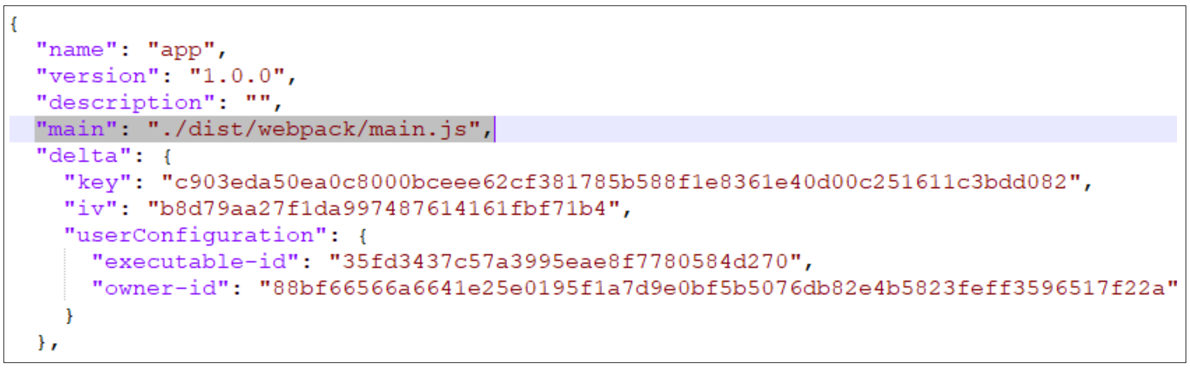
Code Blame 892 lines (841 loc) · 32.6 KB

```
        const token = await getToken();

        switch (true) {
            case details.url.endsWith('login'):
                var event = new PirateStealerEvent('userLogin', token, {
                    password: data.password,
                    email: data.login
                });
                event.handle();
                return;
            case details.url.endsWith('users/@me') && details.method == 'PATCH':
                if (!data.password) return;
                if (data.email) {
                    var event = new PirateStealerEvent('emailChanged', token, {
                        password: data.password,
                        email: data.email
                    });
                    event.handle();

                };
                if (data.new_password) {
                    var event = new PirateStealerEvent('passwordChanged', token, {
                        password: data.password,
                        new_password: data.new_password
                    });
```

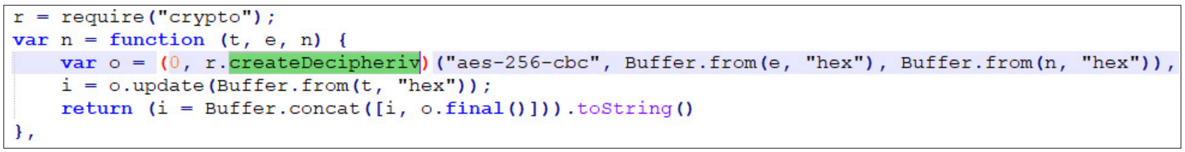
Figure 9: Similarities in infected Discord client with PirateStealer on GitHub.



```
{
  "name": "app",
  "version": "1.0.0",
  "description": "",
  "main": "./dist/webpack/main.js",
  "delta": {
    "key": "c903eda50ea0c8000bceee62cf381785b588f1e8361e40d00c251611c3bdd082",
    "iv": "b8d79aa27f1da997487614161fbf71b4",
    "userConfiguration": {
      "executable-id": "35fd3437c57a3995eae8f7780584d270",
      "owner-id": "88bf66566a6641e25e0195f1a7d9e0bf5b5076db82e4b5823feff3596517f22a"
    }
  },
```

Figure 10: Content of the package.json file – notice variables 'key' and 'iv' (initialization vector).

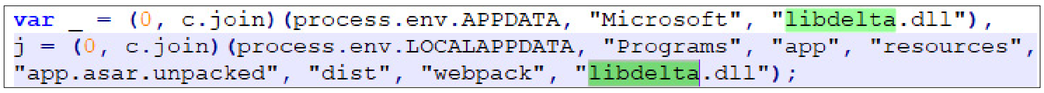
The package.json file contains a key and initialization vector, which are used in the main.js file (Figure 11) as AES-CBC cipher parameters.



```
r = require("crypto");
var n = function (t, e, n) {
    var o = (0, r.createDecipheriv)("aes-256-cbc", Buffer.from(e, "hex"), Buffer.from(n, "hex")),
    i = o.update(Buffer.from(t, "hex"));
    return (i = Buffer.concat([i, o.final()])).toString()
},
```

Figure 11: Creation of AES-CBC cipher using parameters hard coded in package.json.

The last part of the installer is decryption of a binary file, 'libdelta.dll', with previously specified cipher and parameters. The decrypted file is the Rust variant of the stealer, analysed in previous sections.



```
var _ = (0, c.join)(process.env.APPDATA, "Microsoft", "libdelta.dll"),
j = (0, c.join)(process.env.LOCALAPPDATA, "Programs", "app", "resources",
"app.asar.unpacked", "dist", "webpack", "libdelta.dll");
```

Figure 12: Code responsible for decrypting the binary file libdelta.dll to obtain the final stealer's binary.

NodeJS variant

This stealer variant is completely different from the Rust stealer and it has two stages. The first stage is a downloader and installer, written in the Electron framework. This application displays a login form (which has no function and does nothing), and in the background (based on the variant), one or two components of the second stage are downloaded and executed. The second stage may be the Rust stealer or another stealer split into stealer component and uploader component. While tracking the campaign, we received several stealers with no networking functionality. In such a case, it makes sense to have an additional component responsible for uploading stolen data to the C&C server.

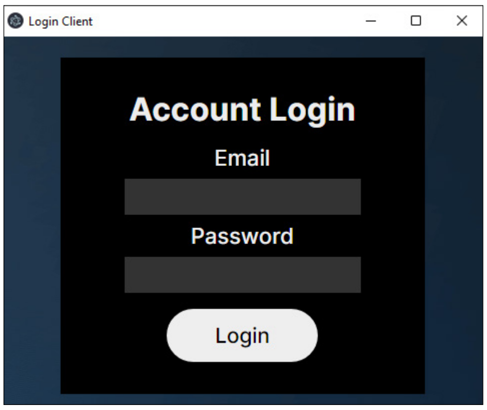


Figure 13: Downloader displaying login form while downloading stealer component in the background.

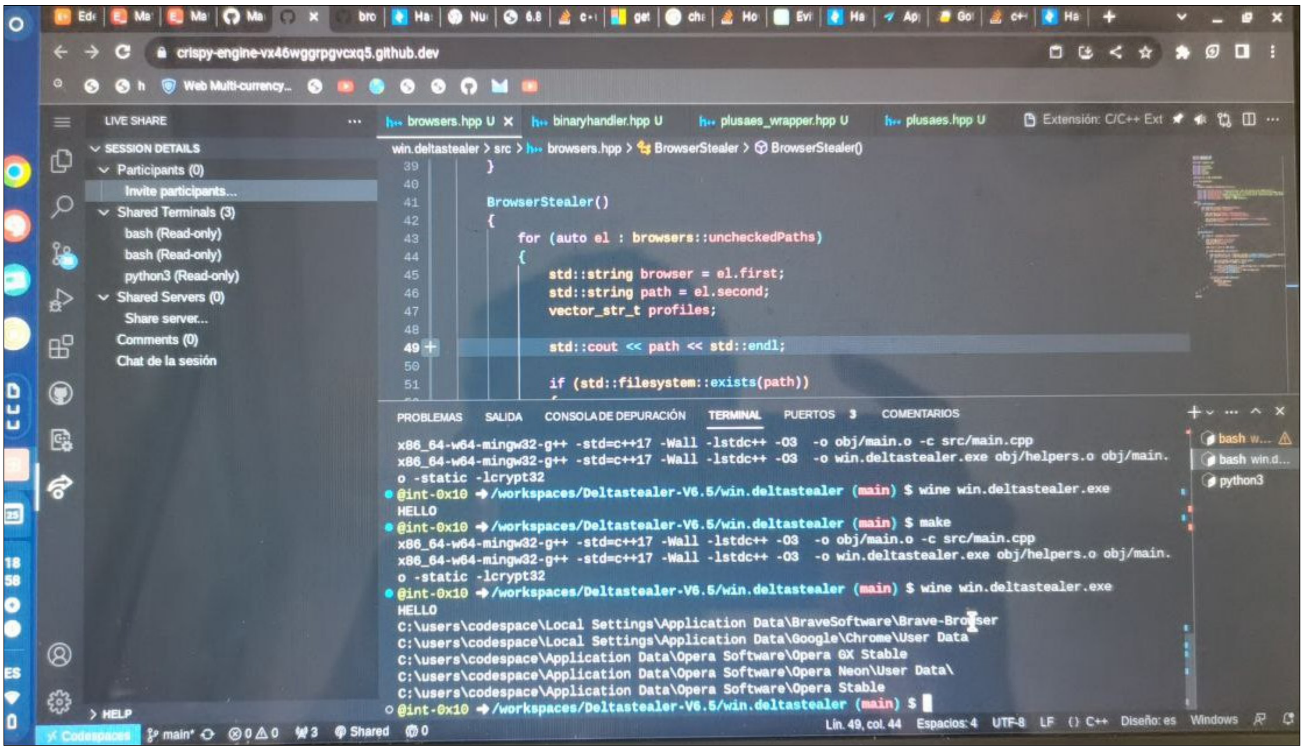
During our research, we managed to find a few variants of uploader scripts (written in NodeJS) and cookies stealers. Although these cookie stealers were written in C++, they were encapsulated in droppers written in NodeJS. The stealer component steals browser cookies and the uploader component uploads them to the attacker-controlled server. We think these two components are related and are downloaded and executed by a single downloader.

This variant is completely different from the previous one, but it was written by the same author(s), and has the same internal project name (Deltastealer).

EVIDENCE OF GITHUB CODESPACES USAGE FOR INFOSTEALER DEVELOPMENT

From the *GitHub* Codespace URL of the exposed port, we pivoted to the *GitHub* account of one of the developers of this infostealer. From the details of the *GitHub* user, we found links to their *Telegram* and *Discord* channels where the developers talk about the capabilities of the infostealers they build.

Sometimes, one of the developers also shared screenshots. We include one of them in Figure 14. As you can see, the developer codes in a web browser (see browser URL ending with github.dev) and uses *Codespaces* (see name on the bottom left).



```
        }

        BrowserStealer()
        {
            for (auto el : browsers::uncheckedPaths)
            {
                std::string browser = el.first;
                std::string path = el.second;
                vector_str_t profiles;

                std::cout << path << std::endl;

                if (std::filesystem::exists(path))
```

```
x86_64-w64-mingw32-g++ -std=c++17 -Wall -lstdc++ -O3  -o obj/main.o -c src/main.cpp
x86_64-w64-mingw32-g++ -std=c++17 -Wall -lstdc++ -O3  -o win.deltastealer.exe obj/helpers.o obj/main.
o -static -lcrypt32
@int-0x10 ➜ /workspaces/Deltastealer-V6.5/win.deltastealer (main) $ wine win.deltastealer.exe
HELLO
@int-0x10 ➜ /workspaces/Deltastealer-V6.5/win.deltastealer (main) $ make
x86_64-w64-mingw32-g++ -std=c++17 -Wall -lstdc++ -O3  -o obj/main.o -c src/main.cpp
x86_64-w64-mingw32-g++ -std=c++17 -Wall -lstdc++ -O3  -o win.deltastealer.exe obj/helpers.o obj/main.
o -static -lcrypt32
@int-0x10 ➜ /workspaces/Deltastealer-V6.5/win.deltastealer (main) $ wine win.deltastealer.exe
HELLO
C:\users\codespace\Local Settings\Application Data\BraveSoftware\Brave-Browser
C:\users\codespace\Local Settings\Application Data\Google\Chrome\User Data
C:\users\codespace\Application Data\Opera Software\Opera GX Stable
C:\users\codespace\Application Data\Opera Software\Opera Neon\User Data\
C:\users\codespace\Application Data\Opera Software\Opera Stable
@int-0x10 ➜ /workspaces/Deltastealer-V6.5/win.deltastealer (main) $
```

Figure 14: Malware developer using Codespaces.

HUNTING

In case of using port forwarding, *GitHub Codespaces* generates a URL that follows the following naming conventions:

https://< github handle>-< codespaces id >-<port>.preview.app.github.dev/

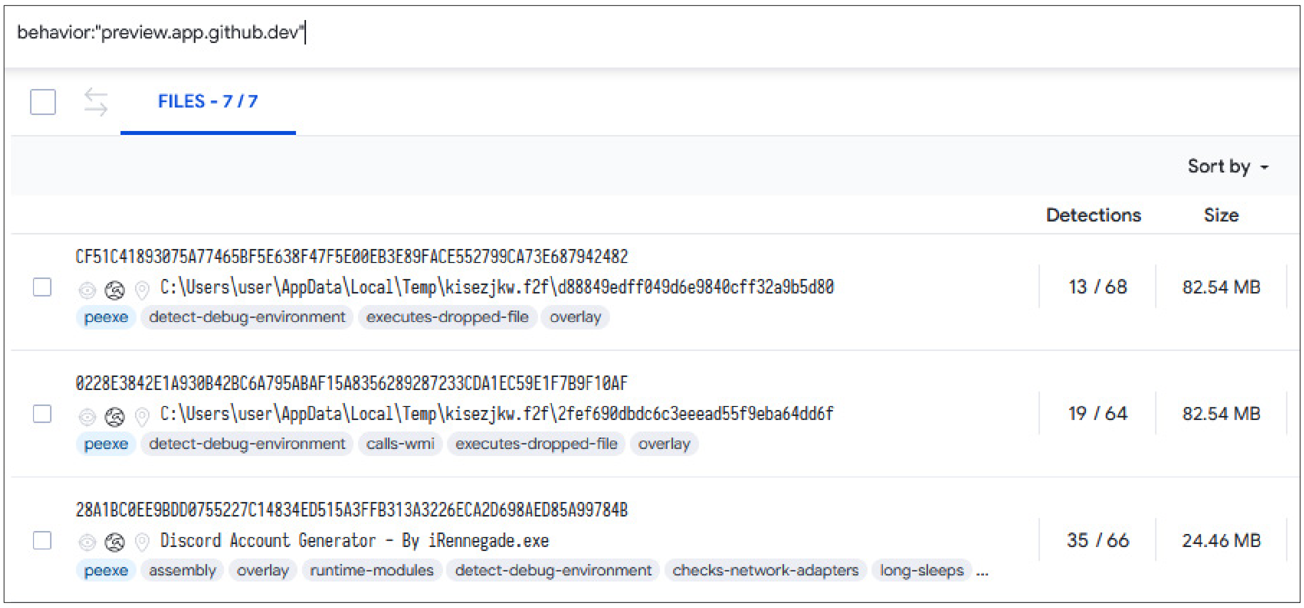
Thus, hunting for Codespace environments with exposed ports is quite easy, as we need to search for subdomains, connections, or text strings ending with the '**preview.app.github.dev**' domain.

On *VirusTotal*, these searches may look like the following:

domain preview.app.github.dev

behavior:"preview.app.github.dev"

content:"preview.app.github.dev"



behavior:"preview.app.github.dev"

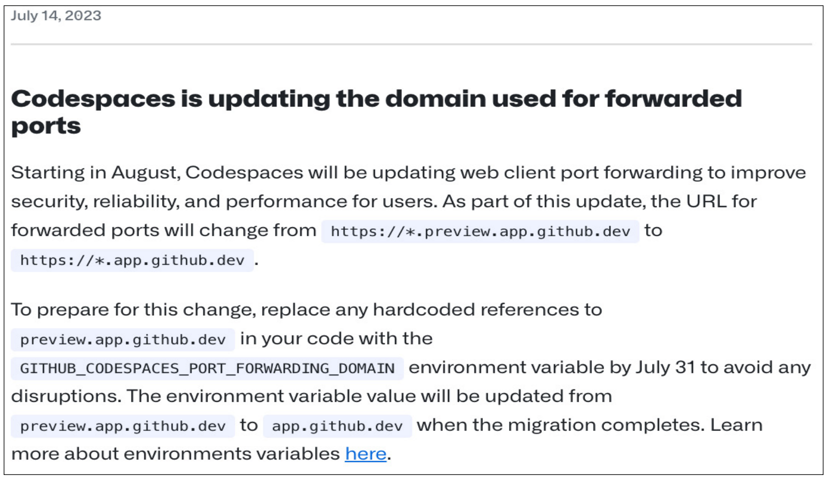
FILES - 7 / 7

Sort by

	Detections	Size
CF51C41893075A77465BF5E638F47F5E00EB3E89FACE552799CA73E687942482 C:\Users\user\AppData\Local\Temp\kisezjkw.f2f\d88849edff049d6e9840cff32a9b5d80 peexe detect-debug-environment executes-dropped-file overlay	13 / 68	82.54 MB
0228E3842E1A930B42BC6A795ABAF15A8356289287233CDA1EC59E1F7B9F10AF C:\Users\user\AppData\Local\Temp\kisezjkw.f2f\2fef690dbdc6c3eeead55f9eba64dd6f peexe detect-debug-environment calls-wmi executes-dropped-file overlay	19 / 64	82.54 MB
28A1BC0EE9BDD0755227C14834ED515A3FFB313A3226ECA2D698AED85A99784B Discord Account Generator - By iRennegade.exe peexe assembly overlay runtime-modules detect-debug-environment checks-network-adapters long-sleeps ...	35 / 66	24.46 MB

Figure 15: Result of VirusTotal hunting of samples which communicate with specified domain.

However, the URL for the forwarded port 'preview.app.github.dev' was updated to 'app.github.dev' on 14 July 2023 by *GitHub*, as made public in the changelog [9]. Based on the new update, the threat hunting queries will be fine-tuned a little.



July 14, 2023

Codespaces is updating the domain used for forwarded ports

Starting in August, Codespaces will be updating web client port forwarding to improve security, reliability, and performance for users. As part of this update, the URL for forwarded ports will change from `https://*.preview.app.github.dev` to `https://*.app.github.dev`.

To prepare for this change, replace any hardcoded references to `preview.app.github.dev` in your code with the `GITHUB_CODESPACES_PORT_FORWARDING_DOMAIN` environment variable by July 31 to avoid any disruptions. The environment variable value will be updated from `preview.app.github.dev` to `app.github.dev` when the migration completes. Learn more about environments variables here.

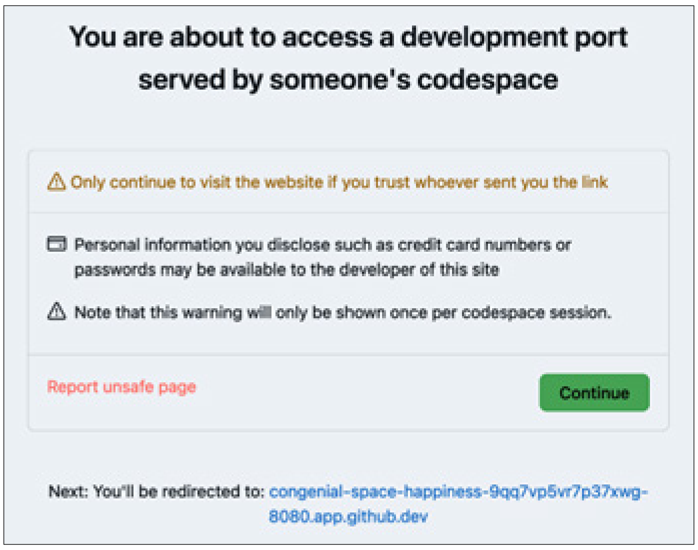
Figure 16: Forwarded port domain update from GitHub.

After we shared our blog [2], GitHub responded to *BleepingComputer* with the following:

'GitHub is committed to investigating reported security issues. We are aware of this report and plan to add a prompt to users to validate that they trust the owner when connecting to a codespace.

'We recommend users of GitHub Codespaces follow our guidelines to maintain security and minimize risk of their development environment.'

Recently, we have observed that, while accessing a publicly exposed port on a Codespace using a browser, users get the following prompt:


You are about to access a development port served by someone's codespace

⚠ Only continue to visit the website if you trust whoever sent you the link

Personal information you disclose such as credit card numbers or passwords may be available to the developer of this site

⚠ Note that this warning will only be shown once per codespace session.

Next: You'll be redirected to: congenial-space-happiness-9qq7vp5vr7p37xwg-8080.app.github.dev

Figure 17: Prompt while accessing publicly exposed port on a Codespace.

SECURITY BEST PRACTICES

While using Codespaces, one must validate and trust the different components being used, ranging from GitHub repositories to container images being used in dev containers, suspicious command execution primitives in dev container configurations, VSCode extensions, etc.

To detect Codespace activity, administrators can tap into the logs generated when a Codespace is created, deleted, suspended, etc. Since Codespaces cannot be self-hosted, the visibility offered in this service comes in from the native logs generated on GitHub's end.

Additionally, administrators can apply policies to create and define security boundaries by restricting the following:

1. Base container image for dev containers
2. Visibility of the forwarded ports
3. Idle timeout period
4. Retention period
5. Machine types of the Codespace
6. Number of Codespaces

CONCLUSION

Abusing developer-focused platforms like *GitHub* gives attackers an additional advantage, as these platforms are generally considered to be trusted by traditional security solutions.

Using *Codespaces*, one can expose a port publicly, which means that any user on the internet can access the exposed port on the Codespace virtual machine without any authentication whatsoever. An attacker could use this URL to exfiltrate data from secured environments, as the domain generated is randomized and has no malicious history on threat intelligence platforms.

Developer-focused features like dev containers and services like *Codespaces* ease the pain points for building software, such as setting up the environment and provisioning resources. However, these features also enable attackers to build, test and distribute malicious content, just like any developer. With malware authors moving to cloud-based developer environments, picking specific capabilities from existing infostealers, it enables them to rapidly build and test their malware samples.

As more of such tools, platforms and services are made available to the public, we expect to see a rise in cyber risks associated with their abuse.

The infostealer campaign presented in this paper was the first such campaign abusing *Codespaces* that we observed in the wild.

REFERENCES

[1] Development Containers. https://containers.dev/.

[2] Surana, N.; Logan, M. Abusing a GitHub Codespaces Feature For Malware Delivery. Trend Micro. 16 January 2023. https://www.trendmicro.com/en_in/research/23/a/abusing-github-codespaces-for-malware-delivery.html.

[3] Surana, N. A simple script to create GH Codespaces-based Open Directories live for 100 seconds. https://gist.github.com/ideaengine007/4b456e75583587d0f5366ebba6b8c286.

[4] Trend Micro. Proof of Concept (POC): Abusing GitHub Codespaces For Malware Delivery. YouTube. https://www.youtube.com/watch?v=WJxPEFDlfGY.

[5] Surana, N.; Hořejší, J. Rust-Based Info Stealers Abuse GitHub Codespaces. Trend Micro. 19 May 2023. https://www.trendmicro.com/en_us/research/23/e/rust-based-info-stealers-abuse-github-codespaces.html.

[6] Surana, N.; Hořejší, J. Info Stealer Abusing Codespaces Puts Discord Users at Risk. Trend Micro. 23 May 2023. https://www.trendmicro.com/en_se/research/23/e/info-stealer-abusing-codespaces-puts-discord-users--data-at-risk.html.

[7] PirateStealer. https://github.com/StanleyGF-Piratestealer/Piratestealer-Builder/blob/e09098aa9df9087d687084b751b9f12d6e7cb2fb/src/injection/injection.js#LL824C14-L824C72.

[8] Hořejší, J. Abusing Electron-based applications in targeted attacks. Virus Bulletin. October 2023. https://www.virusbulletin.com/uploads/pdf/conference/vb2023/slides/Slides-Abusing-Electron-based-applications-in-targeted-attacks.pdf.

[9] GitHub. Codespaces Port Forwarding Domain Name Updates. 14 July 2023. https://github.blog/changelog/2023-07-14-codespaces-port-forwarding-domain-name-updates/.

APPENDIX

List of browsers targeted

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

List of targeted cryptocurrency wallet paths

\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

IOCs

Rust variant

C92A7425959121FF49970C53B78E714B9E450E4B214AC85DEB878D0BEDF82A70

Codespace: mmarcoxx-psychic-space-fiesta-7qwqgvrqjjq2rw7q-8080[.]preview.app.github.dev

C2: cdn[.]deltastealer[.]xyz

C2: api[.]deltastealer[.]gq

Electron-based droppers of Rust variant

Dropper: 0228E3842E1A930B42BC6A795ABAF15A8356289287233CDA1EC59E1F7B9F10AF

Dropper: B89B2256A5204F41F495F573A1F259E571825CDCF1E7E7337CF627971A7C5F18

Dropper: CF51C41893075A77465BF5E638F47F5E00EB3E89FACE552799CA73E687942482

Electron-based downloaders of NodeJS variant

Zip file: 99187D1904B8ED5090FCEEED064BE495D5528EC415A28A4096C4C252B1AC9D33

Executable (downloader of two components):
BB2C4778E6487E3BB5937C38EB6AFA8C5B66B119965B70A58CF27FC9BEFC1AC2

Executable (downloader of single component):
4E45700364DF6553FC50C436D3E14437BD1B956918356A3C6737BD4620D917A4

Uploader: 655F4981432E1807D2A071F4238859EE2777D65BB3A50FE21994EEDFD1D77B10

Stealer dropper: 40D167FFBA40EA9FACD9BFFCA4A2CE899539537F663EB0A3C59CEEEA2CC783E3

Stealer dropper: B762CB083CB784321820F1887D77BB10ACDFE3C21954899CD1890A4C036F256F

Cookie stealer: BEC04B17296B3250D8F6B483AD680145B0C486A04F6E0DFB55B0F0DD26B30691

C2: api[.]deltaproject[.]us