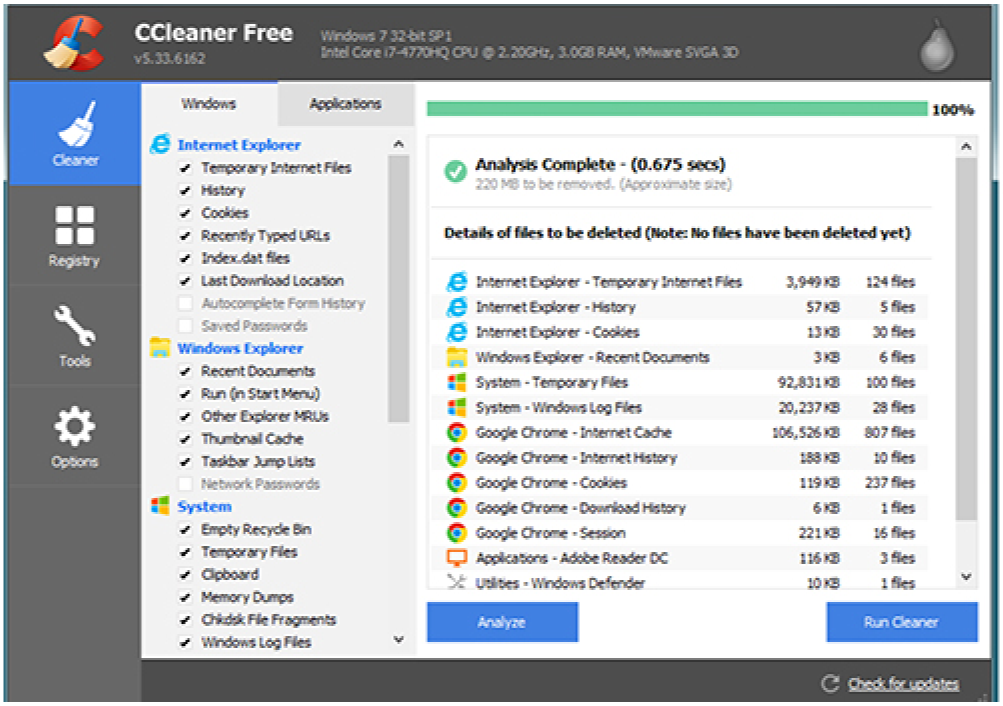
CCleaner Backdoor: Analysis & Recommendations

Karan Sood

October 4, 2017



The term "supply chain attacks" means different things to different people. To the general business community, it refers to attacks targeting vulnerable third-parties in a larger organization's supply chain. A well-known retail chain's massive breach in 2013 is a classic example: Adversaries used a poorly protected HVAC vendor as their gateway to hack into the giant retailer's enterprise network. However, threat researchers have another definition: To them, supply chain attacks can also denote the growing phenomenon in which malicious code is injected into new releases and updates of legitimate software packages, effectively turning an organization's own software supply infrastructure into a potent and hard-to-prevent attack vector. The recent backdoor that was discovered embedded in the legitimate, signed version of CCleaner 5.33, is just such an attack.

To help inform the user community and empower them to better defend against software supply chain attacks, the CrowdStrike® Security Response Team (SRT) conducted a thorough analysis of the CCleaner backdoor. A popular PC optimization tool, the 5.33 version of CCleaner has had widespread distribution across multiple industries, but the embedded code appeared to actually be targeted at specific groups in the technology sector. (More information on targeted industries is available for CrowdStrike customers in our Falcon Intelligence™ portal.) CrowdStrike's threat intelligence team had also previously reported on the malware's C2 (command and control) infrastructure in a recent alert for CrowdStrike customers identifying possible links to Aurora Panda. The report also outlines the potential for additional adversary tactics, techniques and procedures (TTPs).

Technical Analysis

CCleaner

CCleaner is a PC cleaning utility developed by Piriform, which was recently acquired by antivirus (AV) provider Avast in June 2017. The affected version of the utility contains a modified __scrt_common_main_seh function that routes the execution flow to a custom function meant to decode and load the malware. This takes place even before the entry point (EP) of the utility is reached.The new execution flow leads to a function that decodes a blob of data, as reproduced in Python below:

```
def decode(indata):
   key = 0x2547383

   i = 0

   dec = []

   for i in range(0, len(indata)):

       key = ((key * 0x47a6547) & 0xFFFFFFFF) & 0xFF
       dec.append(blob[i] ^ key)

       key = key >> 0x8

   return dec
```

The result of the decoding subroutine is shellcode and the payload (which is missing the IMAGE_DOS_HEADER field). The missing IMAGE_DOS_HEADER is likely to subvert AV solutions that search for MZ (0x4d5a) headers in memory.

Next, the program creates a memory heap with the flag HEAP_CREATE_ENABLE_EXECUTE to allow for execution, and copies the shellcode on the heap, and executes it.

ShellCode

The shellcode is responsible for loading the payload in memory. It attains the PEB (Process Environment Block) of the malware process to load kernel32.dll and find the location of the function **GetProcAddress**. This function is used to retrieve the addresses of functions such as **VirtualAlloc**, **memcpy**, and **LoadLibrary**. It allocates PAGE_EXECUTE_READWRITE memory to which it copies the previously decoded payload (minus the IMAGE_DOS_HEADER) as shown below.

```
0000000: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000010: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000020: 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

```
0000030: 0000 0000 0000 0000 0000 0000 d000 0000 ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000a0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000b0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000c0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000d0: 5045 0000 4c01 0200 c23a 8059 0000 0000 PE..L....:.Y....
00000e0: 0000 0000 e000 0e21 0b01 0600 0020 0000 .......!..... ..
00000f0: 0002 0000 0000 0000 0011 0000 0010 0000 ................
0000100: 0030 0000 0000 0010 0010 0000 0002 0000 .0..............
0000110: 0400 0000 0000 0000 0400 0000 0000 0000 ................
0000120: 0040 0000 0004 0000 0000 0000 0200 0000 .@..............
0000130: 0000 1000 0010 0000 0000 1000 0010 0000 ................
0000140: 0000 0000 1000 0000 0000 0000 0000 0000 ................
0000150: 4c28 0000 dc00 0000 0000 0000 0000 0000 L(..............
0000160: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000170: 0030 0000 c000 0000 0000 0000 0000 0000 .0..............
0000180: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000190: 0000 0000 0000 0000 0000 0000 0000 0000 ................
00001a0: 0000 0000 0000 0000 0010 0000 0001 0000 ................
00001b0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
00001c0: 0000 0000 0000 0000 2e74 6578 7400 0000 .........text…
00001d0: 041e 0000 0010 0000 0020 0000 0004 0000 ......... ......
00001e0: 0000 0000 0000 0000 0000 0000 2000 00e0 ............ …
00001f0: 2e72 656c 6f63 0000 1a01 0000 0030 0000 .reloc.......0..
0000200: 0002 0000 0024 0000 0000 0000 0000 0000 .....$..........
0000210: 0000 0000 4000 0042 0000 0000 0000 0000 ....@..B........
```

Once the payload is copied to the newly allocated memory, the shellcode resolves the needed API's, and calls the OEP (original entry point) of the payload in memory.

Payload

Environment Checks

Once it's loaded, the payload creates a thread that performs the core functionality of the malware. It performs a few checks at the onset of the environment and the user privileges. The malware employs the function **msvcrt.time** to record the current time of the malware. It then uses **IcmpCreateFile** and **IcmpSendEcho** to send an IPv4 ICMP echo to an invalid IP address, with a timeout of 601 seconds. This is meant to delay the execution of the malware by 601 seconds; this delay is then measured by calling **msvcrt.time** again, and ensuring that more than 600 seconds have elapsed between the first and second calls to the function. It should be noted that if the call to **IcmpCreateFile** fails, the malware will just sleep for 600 seconds.These steps are measures against debugging and/or sandboxing. It also invokes **IsUserAnAdmin** to ensure that the current user is member of the administrator's group. If either of these checks fails, the malware exits immediately.

It uses a decoding scheme as the one described above to decode strings during runtime in memory. It is important to note that these dynamically decoded strings are zeroed out in memory before each function using them exits. The strings dynamically decoded throughout the execution of the malware are listed in the Appendix section of this blog.

The malware also checks the privilege levels of its own process; if the process does not have administrative privileges, it uses **AdjustTokenPrivileges** to enable the **SeDebugPrivilege** value for the process. This enables the process to either debug or adjust memory for a process owned by another account.

Registry Checks

The malware checks for the following registry key: **HKLM\SOFTWARE\Piriform\Agomo\TCID**. The key value is supposed to hold a system time value; if the value is greater than the current time, the malware will terminate. It also checks the value of **HKLM\SOFTWARE\Piriform\Agomo\MUID**. If the key does not exist, the malware will set its value using a pseudo-random number derived in the following manner:

```
// Pseudocode to calculate MUID val
DWORD MUID;
unsigned int seed, rand1, rand2;
seed = GetTickCount();
srand(seed);
rand1 = rand();
rand2 = rand() * rand1;
MUID = GetTickCount() ^ rand2;
```

Gathering Victim Information

Once the checks are completed, the malware gathers the following information about the victim machine:

- OS major version
- OS minor version
- OS architecture
- Computer name
- Computer DNS domain

- IPv4 addresses associated with the machine. This information is gathered by calling **GetAdaptersInfo**, and then enumerating through each adapter to search for the IP_ADAPTER_INFO → IpAddressList → IpAddress field.
- Installed applications. The malware accesses the registry key HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall, and enumerates through each key, and compares the *Publisher* value with “Microsoft Corporation.” If there is a match, it moves on to the next value. If not, it will attain the DisplayName value using **SHGetValueA**, and insert it into memory. Each name is prepended with an “S” in memory.
- Full name of the executable image of each running process. The malware calls **WTSEnumerateProcessA** to get a pointer to an array of WTS_PROCESS_INFO structures, which are then used to get the *ProcessName* field for each process. Each process name is prepended with a “P” in memory.

This information is stored in a data structure in memory in the following manner:

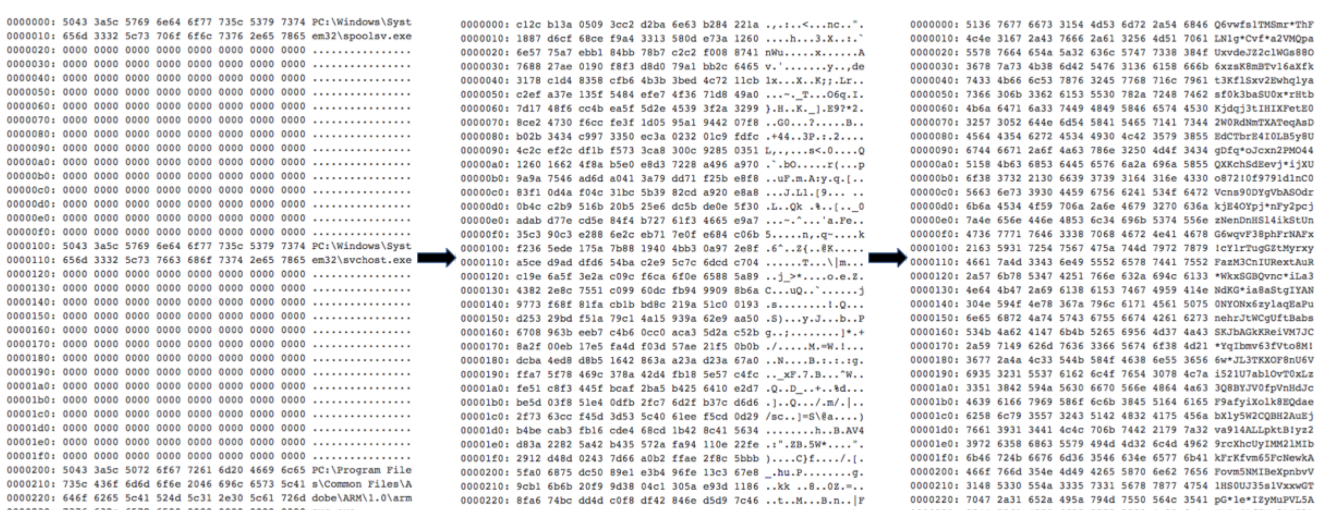
```
00000000 DataStruct       struc ; (sizeof=0x42A0, mappedto_51)
00000000 MUID_Val         dd ?
00000004 OS_MajorVersion  db ?
00000005 OS_MinorVersion  db ?
00000006 OS_x64Flag       db ?
00000007 BOOL_Zero        db ?
00000008 ComputerName     db 64 dup(?)
00000048 ComputerNameDnsDomain db 64 dup(?)
00000088 IPAddresses      db 24 dup(?)
000000A0 ApplicationList ApplicationList_Installed ? ; Each application name appened with an 'S'
000024A0 Process_ExecutableImageName RunningProcesses ? ; Each process name appended with a 'P'
000042A0 DataStruct       ends
```

The MUID_Val is used as a unique identifier for the victim machine.

Next, the structure is encoded in memory in two steps:

- Step 1: Aforementioned scheme
- Step 2: Modified version of base64

The image below displays the data structure as it goes through each encoding step.



The custom base64 encoding scheme uses a modified Base64 index table. Rather than the regular table that has the following values: **ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/**; its table has the following values: **abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!***.

C2 Communication

Once the victim machine information has been encoded, the malware queries the registry key **HKLM\SOFTWARE\Piriform\Agomo\NID**. Upon the initial run, the registry key does not exist; however, the malware eventually inserts an IP address computed via a DGA (Domain Generating Algorithm) later in the execution flow. It is interesting to note that even if the registry key exists, the malware extracts the IP address from the registry value, but does not do anything with it. After the registry check, it decodes the hard-coded IP address **216.126.225[.]148**, and attempts to send the encoded data struct to it via an HTTP POST request on port 443. It uses **InternetSetOptionA** to set the following option flags on the HTTP handle:

- SECURITY_FLAG_IGNORE_CERT_DATE_INVALID → Ignores bad or expired SSL certificates from the server
- SECURITY_FLAG_IGNORE_CERT_CN_INVALID → Ignores incorrect SSL certificate common names
- SECURITY_FLAG_IGNORE_WRONG_USAGE → Ignores incorrect usage problems
- SECURITY_FLAG_IGNORE_UNKNOWN_CA → Ignores unknown certificate authority problems
- SECURITY_FLAG_IGNORE_REVOCATION → Ignores certificate revocation problems

The malware also calls **HttpAddRequestHeadersA** to append the domain **speccy.piriform[.]com** to the POST request. This is performed to appear inconspicuous and make it harder to detect. It is also likely an attempt to confuse the analyst performing dynamic analysis of the malware. Once the information is sent to the C2, the malware expects to receive a stage 2, which it reads into a locally allocated memory block. Analysis shows that once stage 2 is received, it is decoded using the same custom Base64 and the decoding algorithm. Once decoded, the functions **GetProcAddress** and **LoadLibraryA** are pushed to the stack, and the EP of stage 2 is called. At the time of analysis, stage 2 was not available.

DGA

If the malware cannot connect to the C2, it employs a Domain Generating Algorithm, or DGA, to generate a domain. The DGA is dependent on the current year and month; therefore, it generates a new domain on a monthly basis. Below is the code, reproduced in C, displaying the DGA utilized by the malware.

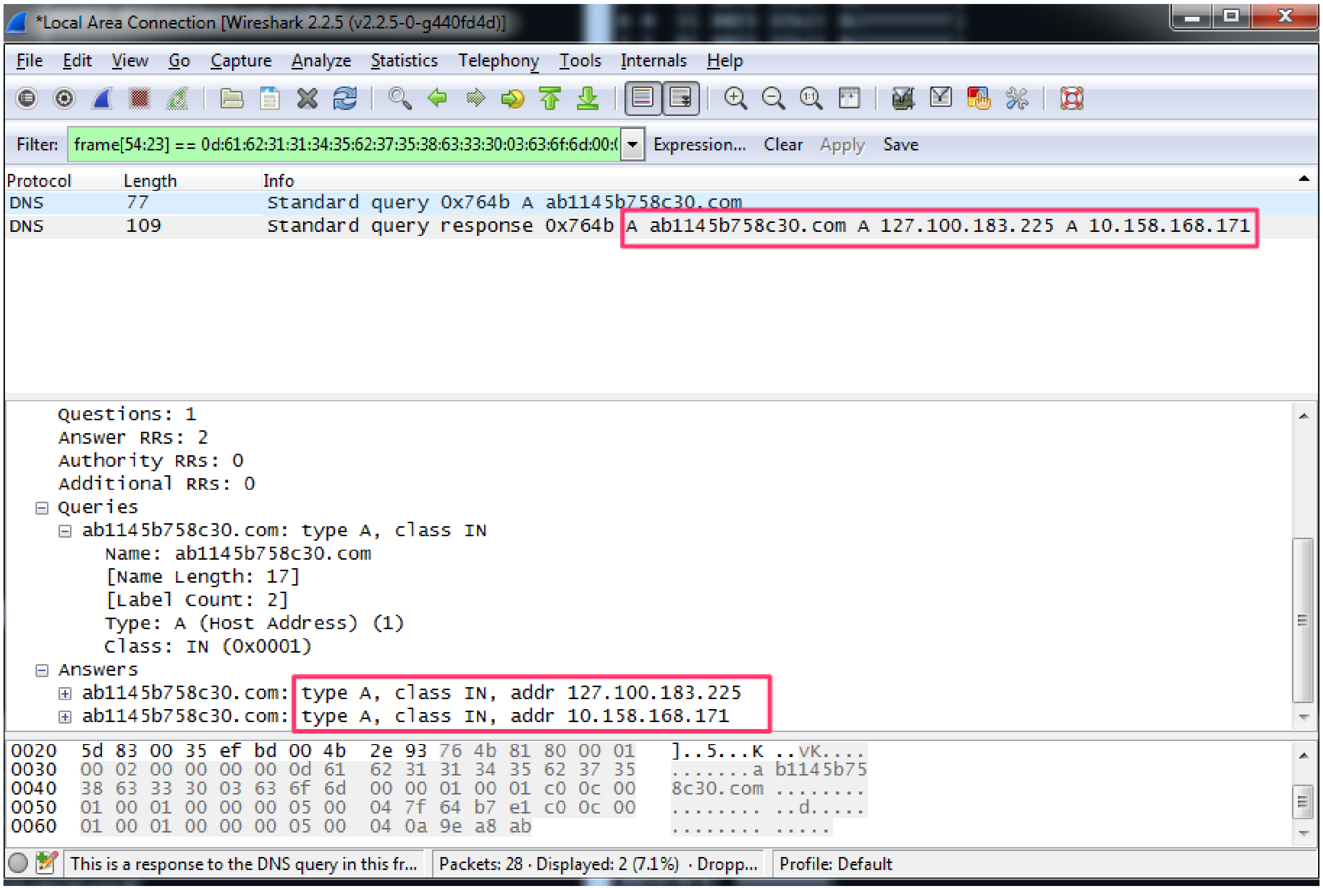
```
#include "stdafx.h"
#include <Windows.h>
#include <stdio.h>
void main()
{
SYSTEMTIME st;
DWORD r1, r2, r3, seed;
char buf[100];
const char *format = "ab%x%x.com";
GetLocalTime(&st);
seed = st.wYear * 10000 + st.wMonth;
srand(seed);
r1 = rand();
r2 = rand();
r3 = rand() * r2;
sprintf_s(buf, format, r3, r1);
}
```

The list of domains calculated for all months in the years 2017 and 2018 are listed in the Appendix.

Once the DGA domain for the current month and year has been calculated, the malware calculates an IP address using that domain in the following steps:

- Get a *hostent* structure by calling **gethostbyname** on the generated domain
- Get the *h_addr_list*, which is a NULL terminated list of IP addresses associated with the domain

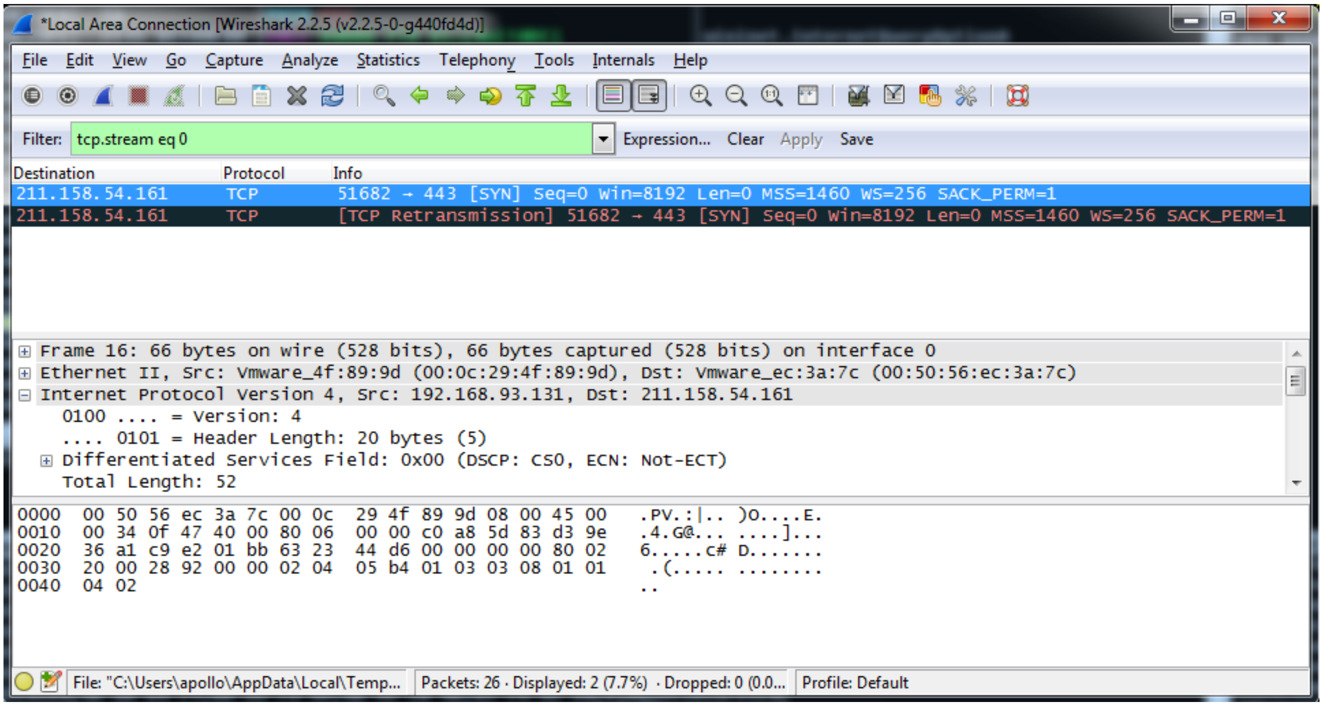
These A records (127.100.183[.]225 and 10.158.168[.]171) for the domain ab1145b758c30[.]com, as highlighted in the PCAP screenshot below will be used to calculate a new C2 IP address. If there are more than two A records, the malware will only utilize the first two on the list.



The Python code below reproduces the algorithm to calculate the new C2 IP address from the A records of the newly generated domain.

```
import struct
import socket
a1 = 0xE1B7647F  # Addresses are returned in network byte order
a2 = 0xABA89E0A def mod_record(rr):
    rr1 = (((rr & 0xff000000) / 0x1000000) ^ (rr & 0xff)) * 0x1000000
    rr2 = (((rr & 0xff0000) / 0x10000) ^ ((rr & 0xff00) / 0x100)) * 0x10000
    rr3 = rr & 0xff00
    rr4 = rr & 0xff
    return (rr1 | rr2 | rr3 | rr4)
newa1 = mod_record(a1)
newa2 = mod_record(a2) newIP =  (newa2 & 0xffff0000) | (newa1 >> 0x10) #
newIP = 0xA1369ED3 print socket.inet_ntoa(struct.pack("<L", newIP))  #
Output is 211.158.54.161
```

The new C2 IP address derived from the records of the domain ab1145b758c30[.]com is 211.158.54[.]161. The malware will attempt to connect to this C2 as shown below. If the connection is successful, it will subsequently send the encoded data structure and await stage 2.

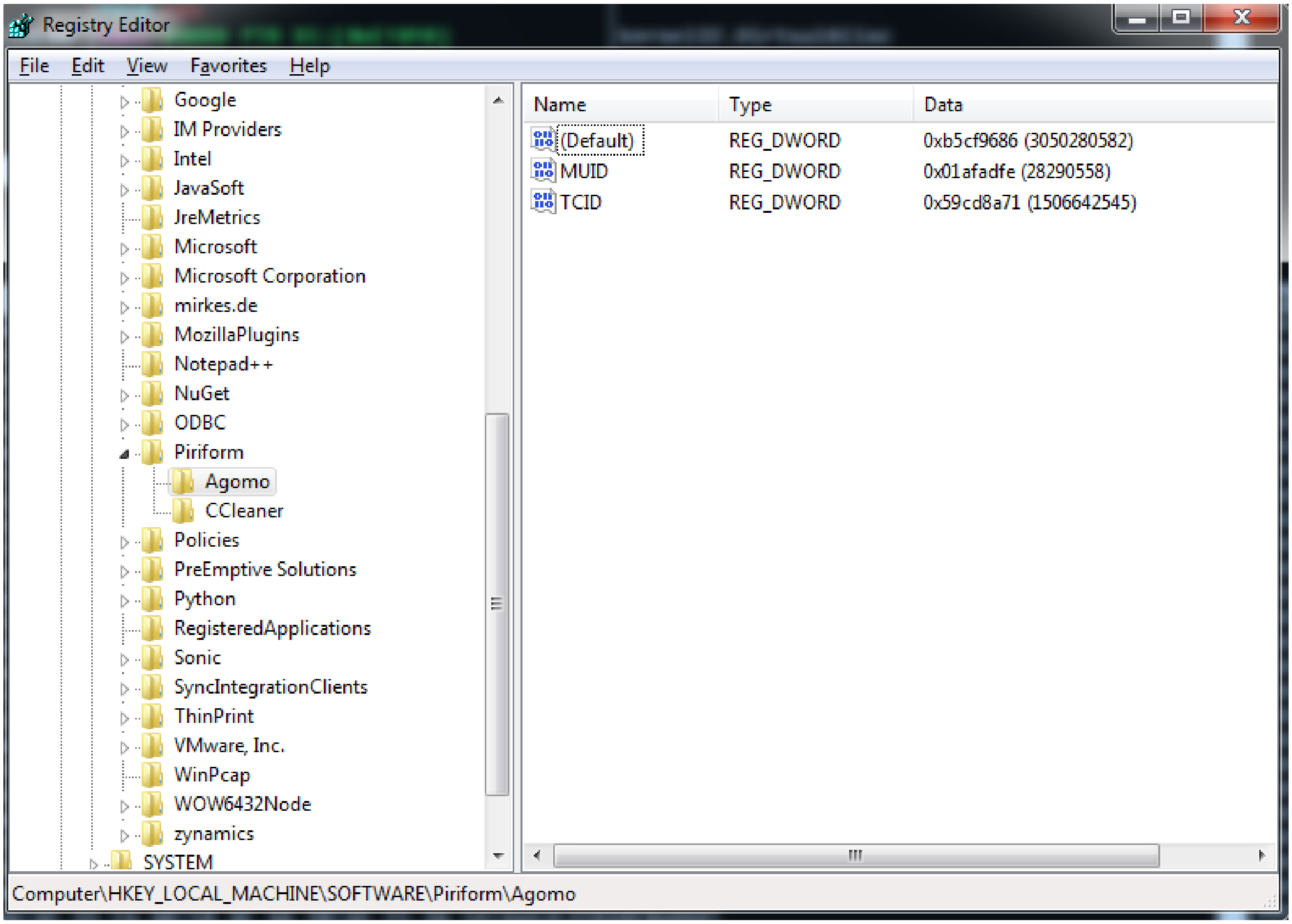


Initial (Buggy) Registry Modifications

Once the C2 communication subroutine has ended, the malware makes two registry modifications:

> Encodes the newly calculated C2 IP address and attempts to save it in **HKLM\SOFTWARE\Piriform\Agomo\NID**. The encoding scheme is the same as the one mentioned before. Analysis shows that before the registry key string is built, a function is called to change the endianness of 0x44494E (DIN) to 0x4E4944 (NID). However, due to a bug in the code the function incorrectly changes it to 0x004E4944 (prepended with a NULL value). Subsequently, function **SHSetValueA** is called with the following parameters:
>
> - **hKey** = HKEY_LOCAL_MACHINE
> - **Subkey** = "SOFTWARE\Piriform\Agomo"
> - **Value** = ""
> - **ValueType** = REG_DWORD
> - **Data** = …
> - **DataLength** = 0x4

The parameter **Value** should be "NID"; however, since the string is incorrectly prepended with a NULL value, the function doesn't read the string at all. The C2 IP address is instead saved in **HKLM\SOFTWARE\Piriform\Agomo\Default** as shown below.



Takes the current time value as determined by the earlier call to **msvcrt.time** and adds 172,800 seconds (2 days) to the value. Saves the new value in HKLM\SOFTWARE\Piriform\Agomo\TCID.

Recommendations

Falcon Endpoint will notify you of any additional activity through our Falcon Intelligence detections. The intent behind the malicious packages was to collect an initial set of reconnaissance data; we urge you to block the known IP address and domains at your network perimeter to prevent any communication to the collection server. In addition, we recommend you update to the latest version of the Avast CCleaner software to ensure the embedded malicious code is removed.

For additional information on CrowdStrike's threat intelligence offerings, visit the Falcon Intelligence product page.

Appendix

Hashes

Information regarding the CCleaner binaries that were affected:

Size: 9791816
SHA256: `1A4A5123D7B2C534CB3E3168F7032CF9EBF38B9A2A97226D0FDB7933CF6030FF`
Compiled: Tue, Dec 29 2015, 21:34:49 UTC – 32 Bit EXE
Version: 5.33.00.6162
Signature Valid
Subject: Piriform Ltd
Issuer: Symantec Class 3 SHA256 Code Signing CA

Size: 7680216
SHA256: `6F7840C77F99049D788155C1351E1560B62B8AD18AD0E9ADDA8218B9F432F0A9`
Compiled: Thu, Aug 3 2017, 9:25:13 UTC – 32 Bit EXE
Version: 5, 33, 00, 6162
Signature Valid
Subject: Piriform Ltd
Issuer: Symantec Class 3 SHA256 Code Signing CA

Size: 7781592
SHA256: `36B36EE9515E0A60629D2C722B006B33E543DCE1C8C2611053E0651A0BFDB2E9`
Compiled: Thu, Aug 3 2017, 9:37:49 UTC – 32 Bit EXE
Version: 5, 33, 00, 6162
Signature Valid
Subject: Piriform Ltd
Issuer: Symantec Class 3 SHA256 Code Signing CA

The following is the information about the decoded payload in memory:
Size: 16384
SHA256: `FA8A55A05CA9E6587C941354628A0E818DCBF42ED3D98C40689F28564F0BFA19`
Compiled: Tue, Aug 1 2017, 8:24:34 UTC – 32 Bit DLL

Network Artifacts

The following is the infrastructure associated with the CCleaner backdoor:

Infrastructure	Connection Type	Description
216.126.225[.]148	Port 443 / TCP	C2

DGA Domains

Month, Year	Domain	Month, Year	Domain
January, 2017	abde911dcc16[.]com	January, 2018	ab3c2b0d28ba6[.]com
February, 2017	ab6d54340c1a[.]com	Feburary, 2018	ab99c24c0ba9[.]com

March, 2017	aba9a949bc1d[.]com	March, 2018	ab2e1b782bad[.]com
April, 2017	ab2da3d400c20[.]com	April, 2018	ab253af862bb0[.]com
May, 2017	ab3520430c23[.]com	May, 2018	ab2d02b02bb3[.]com
June, 2017	ab1c403220c27[.]com	June, 2018	ab1b0eaa24bb6[.]com
July, 2017	ab1abad1d0c2a[.]com	July, 2018	abf09fc5abba[.]com
August, 2017	ab8cee60c2d[.]com	August, 2018	abce85a51bbd[.]com
September, 2017	ab1145b758c30[.]com	September, 2018	abccc097dbc0[.]com
October, 2017	ab890e964c34[.]com	October, 2018	ab33b8aa69bc4[.]com
November, 2017	ab3d685a0c37[.]com	November, 2018	ab693f4c0bc7[.]com
December, 2017	ab70a139cc3a[.]com	December, 2018	ab23660730bca[.]com

Dynamically Decoded Strings

The following are the strings that are dynamically decoded during the malware's execution. It should be noted that each string is promptly zeroed out in memory after use.

```
SOFTWARE\Piriform\Agomo
kernel32.dll
IsWow64Process
SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
Publisher
Microsoft Corporation
DisplayName
QueryFullProcessImageFileNameA
SeDebugPrivilege
%u.%u.%u.%u
ab%x%x.com
speccy.piriform.com
```