"Qealler" a new JAR-based Information Stealer | Zscaler

By Mohd Sadique

Published: 2019-02-06 · Archived: 2026-04-05 23:18:23 UTC

Recently, the Zscaler ThreatLabZ team came across a new type of malware called Qealler, which is written in Java and designed to silently steal sensitive information from an infected machine.

Qealler is a highly obfuscated Java loader that deploys a Python credential harvester.

We first saw this payload hit Zscaler Cloud Sandbox on Jan 21, 2019, and below is a screenshot of the detonation report.

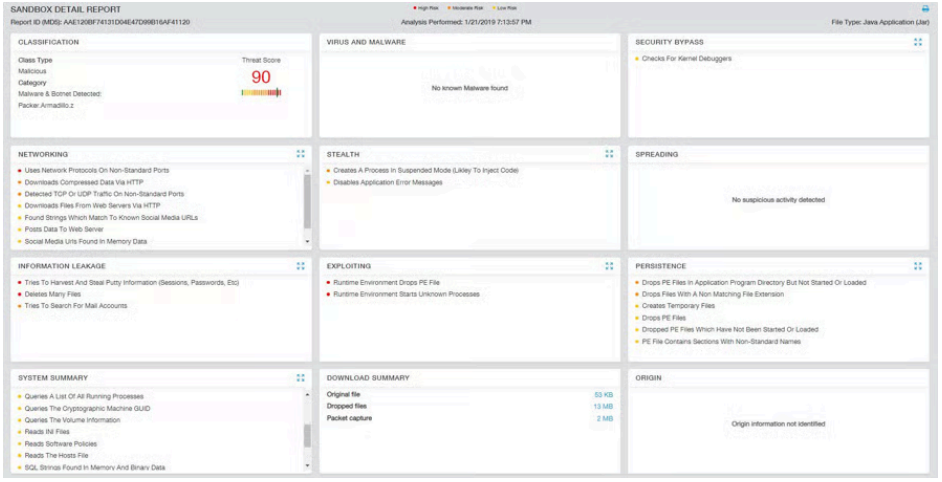

Fig. 1: Zscaler Cloud Sandbox report

This threat makes use of social engineering techniques to initiate the infection, as the malicious JAR file has to be executed by the user. These malicious JAR files are portrayed as invoice-related files, requiring the user to double-click on the file to open it.

We have been monitoring this campaign for the past two weeks, and the malware has been quite active, spiking this week.

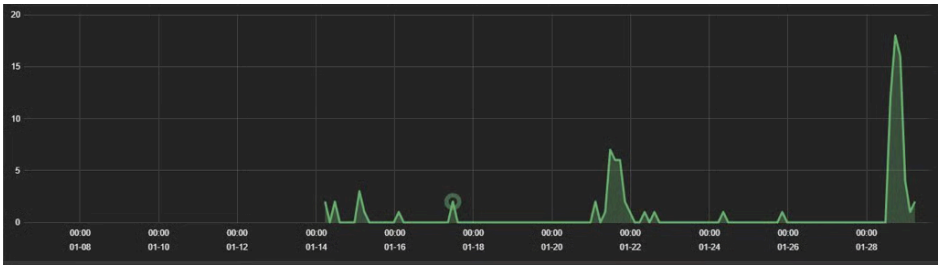

Fig. 2: Hits of Qealler in a week

The malicious JAR file (named Remittance.jar), which we analyzed, was getting downloaded from a compromised site (hiexsgroup.co[.]uk). It is heavily obfuscated with Proguard Java obfuscator. After deobfuscation and decompilation, we saw encrypted URLs that are accessible by a key, as shown in the figure below.

```
public enum C0047if {
    UUID("2a898bc98aaf6c96f2054bb1eadc9848eb77633039e9e9ffd833184ce553fe9b"),
    REPORT_URL("d7c363a2019dac744cf076e11433547a47907e2c2f781e2d1c8f59a40c57dd03"),
    LIB_7Z_URL("8e65457409fea4b2a183125f1c0f552080edb4cefa516b14698cb8d0abf5bb6d"),
    LIB_QEALLER_URL("0e10ad6938994f2466b192d8f29217ad39155b8a3a082b6412048f4a12126b3b");

    public final String f94int;

    private C0047if(String str) {
        this.f94int = str;
    }
}
```

Fig. 3: Accessing encrypted URLs

The sample has a “synchronized” file that contains key-value pairs.

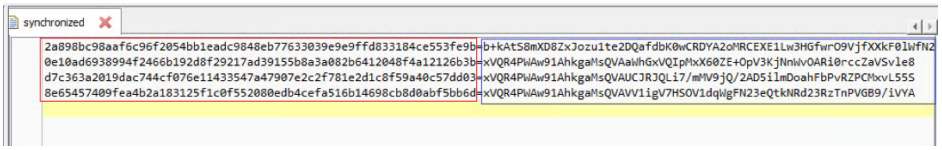

Fig. 4: Key-Value pair of encrypted URLs

On execution, this sample first creates two file paths in *%USERPROFILE%* by checksum of hardcoded strings.

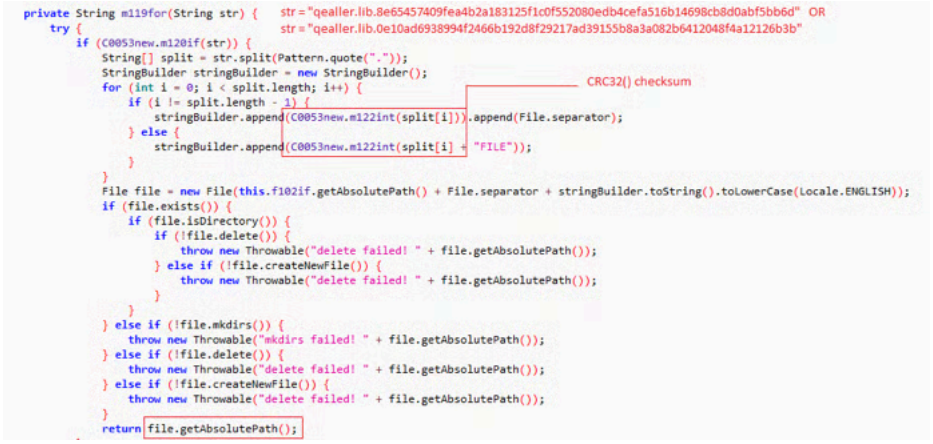

Fig. 5: File Path creation

File path 1:

%USERPROFILE%\\CRC32(“2a890bc98aaf6c96f2054bb1eadc9848eb17633039e9e9ffd833104ce553fe9b”)\\CRC32(“qealler”)\\CRC32(“lib”)\\CRC32(

Equivalent to:

%USERPROFILE%\\a60fcc00\\bda431f8\\a90f3bcc\\83e7cdf9

File Path 2:

%USERPROFILE%\\CRC32(“2a890bc98aaf6c96f2054bb1eadc9848eb17633039e9e9ffd833104ce553fe9b”)\\CRC32(“qealler”)\\CRC32(“lib”)\\CRC32(

Equivalent to:

%USERPROFILE%\\a60fcc00\\bda431f8\\a90f3bcc\\db2bf213

If the above two files don’t exist, the malicious file decrypts the URL, downloads these two files, and stores them in the same place.

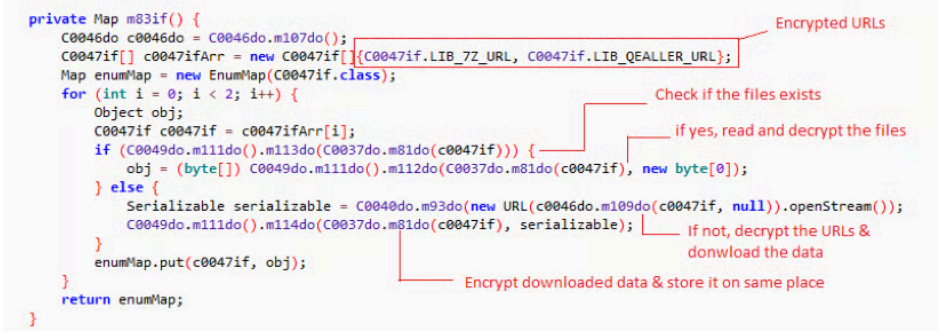

Fig. 6: Encrypts and drops downloaded module

The value of LIB_7Z_URL in the synchronized file is “xVQR4PWAw91AhkgaMsQVAVV1igV7HSOV1dqWgFN23eQtkNRd23RzTnPVGB9/iVYA” which is decoded by BASE64 and decrypted by AES-EBC with the hardcoded key “bbb6fec5ebef0d93”.

The final URL after decryption is hxxp://82.196.11[.]96:55326/lib/7z

The value of LIB_QEALLER_URL in the synchronized file is “xVQR4PWAw91AhkgaMsQVAaWhGxVQIpMxX60ZE+OpV3KjNnWvOARi0rccZaVSvle8”, it is also decrypted by the same algorithm with the same key.

The final URL is hxxp://82.196.11[.]96:54869/lib/qealler

The sample downloads the data from these URLs and encrypts it using the AES algorithm with the key generated by SecureRandom() having hardcoded seed value “2a890bc98aaf6c96f2054bb1eadc9848eb17633039e9e9ffd833104ce553fe9b”.

AES key: 39 3e df 7e fc 58 be 20 60 e4 78 bb 4a 91 38 72

After encryption, it stores both files at the below locations to avoid further downloading in the next run:

%USERPROFILE%\\a60fcc00\\bda431f8\\a90f3bcc\\83e7cdf9 (/lib/7z)

%USERPROFILE%\\a60fcc00\\bda431f8\\a90f3bcc\\db2bf213 (/lib/qealler)

```
C:\Users\Insaan\a60fcc00>tree /f
Folder PATH listing
Volume serial number is 00000200 CE8C:8E16
C:.
+---bda431f8
    +---1505df84
    ¦   +---bf396750
    ¦       +---98dd4acc
    ¦               99de3ada [encrypted machine ID]
    ¦
    +---a90f3bcc
            83e7cdf9 [encrypted /lib/7z]
            db2bf213 [encrypted /lib/qealler]
```

Fig. 7: Created path and dropped files

Along with these two files, the virus creates another file path with the following algorithm and stores an encrypted unique machine ID in it. The ID is generated by a random number of system nanoTime.

Machine ID path:

%USERPROFILE%\\CRC32(“2a890bc98aaf6c96f2054bb1eadc9848eb17633039e9e9ffd833104ce553fe9b”)\\CRC32(“qealler”)\\CRC32(“machine”)\\CF

Equivalent to:

%USERPROFILE%\\a60fcc00\\bda431f8\\1505df84\\bf396750\\98dd4acc\\99de3ada

After the downloading and decryption steps are completed, the sample stores a decrypted copy of *83e7cdf9* and *db2bf213* in the *%TEMP%* directory with the name “_.tmp”.

_5025607018550086163005014574876 39.tmp

_5025621654890043005692237335735 35.tmp

_5025607018550086163005014574876 39.tmp (/lib/7z) is again a JAR file that doesn’t have any Java code inside, but contains three PE files inside the libraries as shown in Fig 8.

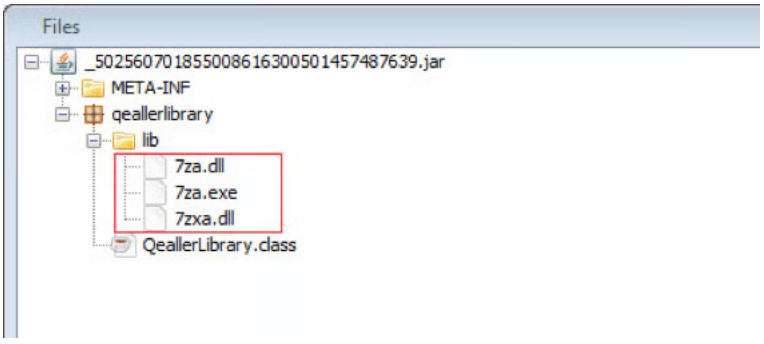

Fig. 8: Content of _5025607018550086163005014574876 39.tmp (/lib/7z)

7za.exe is a repackaged version of 7-zip to ensure the malware executes successfully even if the user does not have it installed by default.

The 7-zip (7za.exe) and its modules (7za.dll, 7zxa.dll) will be extracted from 7z.jar by the main sample and saved in the *%TEMP%* directory with the name “7z_.exe” and “7z_.dll”.

7z_502574395484008643130462441900754.exe

7z_502567545558005642490654395727502.dll

7z_502579570140002751296504101539829.dll

After extraction, the 7-zip executable is called by the main sample with the following command-line options:

%TEMP%\\7z_502574395484008643130462441900754.exe x %TEMP%_502562165489004300569223733573535.tmp -o%TEMP% -p”bbb6fec5ebef0d936db0b031b7ab19b6” -mmt -aoa -y

The downloaded Qealler module *_502562165489004300569223733573535.tmp* (/lib/qealler) is a password-protected file with 7-zip.

The above command will extract the Qealler module in the *%TEMP%* directory with the password: bbb6fec5ebef0d936db0b031b7ab19b6

-mmt: use multithreading mode

-aoa: set overwrite mode

-y: assume yes for all the prompts

The Qealler module is the key component of this malware.

The extracted Qealler module contains Python 2.7.12 with the installed packages to ensure the malware will execute even if the user does not have it installed by default.

The Qealler also has a directory named QaZaqne. It is a custom version of the open source project called LaZagne. LaZagne is used to retrieve lots of passwords stored on a local computer. This is the same functionality of QaZagne, which finds and steals credentials of the most commonly used software from local machines.

Fig. 9: Content of extracted *_502562165489004300569223733573535.tmp* (/lib/qealler)

After extraction, the main sample (Remittance.jar) executes a Python file of QaZagne (main.py) with the following option and takes the JSON output:

%TEMP%\\qealler\\python\\python.exe %TEMP%\qealler\qazaqne\main.py all

```
def get_categories():
    category={'chats':{'help':'Chat clients supported'},'sysadmin':{'help':'SCP/SSH/FTP/FTPS clients supported'},'database':
        {'help':'SQL/NoSQL clients supported'},'svn':{'help':'SVN clients supported'},'git':{'help':'GIT clients supported'}
        ,'maven':{'help':'Maven java build tool'},'php':{'help':'PHP build tool'},'mails':{'help':'Email clients supported'}
        ,'wifi':{'help':'Wifi'},'browsers':{'help':'Web browsers supported'},'windows':{'help':'Windows credentials (
        credential manager, etc.)'},'games':{'help':'Games etc.'}}
    return category
def get_modules():
    moduleNames=[ApacheDirectoryStudio(),Autologon(),Dbvisualizer(),Chrome(),CocCoc(),CoreFTP(),Cyberduck(),Filezilla(),
        FtpNavigator(),GitForWindows(),IE(),Jitsi(),MavenRepositories(),Mozilla(),Composer(),Credman(),OpenSSHForWindows(),
        Opera(),Outlook(),Pidgin(),Puttycm(),RDPManager(),Robomongo(),Tortoise(),Skype(),SQLDeveloper(),Squirrel(),
        Unattended(),Vault(),Wifi(),WinSCP(),Cachedump(),Hashdump(),LSASecrets()]
    return moduleNames
```

Fig. 10: Stealer functions in QaZaqne module

This will get the credentials of all the software shown in the figure below:

	Windows
Browser	Chrome, firefox, IE, Opera
Chats	Jitsy, Pigdin, Skype
Databases	DBVisualizer, Postgresql, Robomongo, Squirrel, SQLdevelopper
Games	GalconFusion, Kalypsomedia, RogueTale, Turba
Git	Git for Windows
Mails	Outlook, Thunderbird
Dumps from memory	Keepass, Wdigest (mimikatz method)
SVN	Tortoise
Sysadmin	Apache Directory studio, CoreFTP, CyberDuck, fileZilla, FTPNavigator, OpenSSH, OpenVPN, PuttyCMRDPManager, WinSCP, Windows Subsystem for Linux
Wifi	Wireless Network
Internal mechanism passwords storage	.NET Passport, Generic Network Hashdump (LM/NT), LSA secret

Fig. 11: Qealler steals credentials of the software in this table

The output of the QaZagne on an infected Windows machine is shown in Fig 12. It is in JSON format and contains the credentials of CoreFTP and a Windows credential manager. It always starts with *#fs#* and ends with *#ff#*.

```
#fs#
[
    {
        "Passwords": [
            [
                {
                    "Category": "Coreftp"
                },
                [
                    {
                        "Host": "*.*.122.44",
                        "Login": "admin",
                        "Password": "adm1n32",
                        "Port": 21
                    }
                ]
            ],
            [
                {
                    "Category": "Credential manager"
                },
                [
                    {
                        "Login": "Insaan",
                        "Password": "Password123$",
                        "URL": "server.company.com"
                    }
                ]
            ]
        ],
        "User": "Mohd Sadique"
    },
    {
        "User": "Insaan"
    }
]
#ff#
```


Fig. 12: JSON output of QaZaqne module

The main sample parses this output, fetches below system information, and encrypts it using an AES-EBC algorithm with key “bbb6fec5ebef0d93”.

```
public C0045new() {
    Runtime runtime = Runtime.getRuntime();
    this.f86for.put("osName", System.getProperty("os.name", "none"));
    this.f86for.put("osVersion", System.getProperty("os.version", "none"));
    this.f86for.put("osArch", System.getProperty("os.arch", "none"));
    this.f86for.put("javaHome", System.getProperty("java.home", "none"));
    this.f86for.put("userName", System.getProperty("user.name", "none"));
    this.f86for.put("userHome", System.getProperty("user.home", "none"));
    this.f86for.put("availableProcessor", runtime.availableProcessors());
    this.f86for.put("freeMemory", runtime.freeMemory());
    this.f86for.put("totalMemory", runtime.totalMemory());
}
```

Fig. 13: Fetch and encrypt system info

The final information scraped from the infected machine before encryption is shown below.

```
{
   "output": {
      "systemInfo":
      [{"value": "1", "key": "availableProcessor"},
      {"value": "3142808", "key": "freeMemory"},
      {"value": "C:\\Program Files\\Java\\jre1.8.0_201", "key": "javaHome"},
      {"value": "x86", "key": "osArch"},
      {"value": "Windows 7", "key": "osName"},
      {"value": "6.1", "key": "osVersion"},
      {"value": "18370560", "key": "totalMemory"},
      {"value": "C:\\Users\\Insaan","key": "userHome"},
      {"value": "Mohd Sadique","key": "userName"}],
      "credentials": [{
         "password": "adm1n32", "website": "N/A", "appname": "Coreftp", "details": "Port:21Host:*.*.122.44", "login": "admin"
      },
      {
         "password": "Password123$", "website": "server.company.com", "appname": "Credential manager", "details": "",
         "login": "Insaan"
      }]
   },
   "qealler": {
      "machine_id": "b203375b1a8d4dca382dbb2216cadc36",
      "uuid":
      "cipaX3K7vgiTin_GDMS8f98sc_hBd6GHVXr5f-oAjv9FMSHVIm1r6kyzVVPjkh2ZoxxV-4yENMAZomq9f72w7Ixt_cl9rFkqYRCJV1eZn280-tOcDVPvrCwB
      xkv6Qlbm"
   },
   "time": 1548503549296
}
```


Fig. 14: Scrapped data from an infected machine

Here, machine_id is a unique ID generated by system nanoTime and uuid is encrypted in a synchronized file.

This output is encrypted and encoded with BASE64 and sent to the command-and-control (C2) server, whose URL is an encrypted value of the key “d7c363a2019dac744cf076e11433547a47907e2c2f781e2d1c8f59a40c57dd03” in a synchronized file.

C2 URL: hxxp://82.196.11[.]96:56636/qealler-reloaded/ping

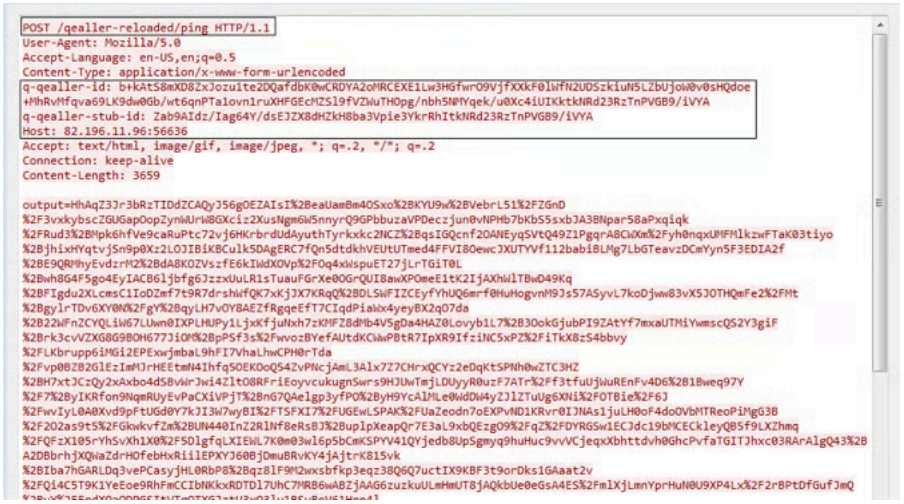

Fig. 15: Data sent to C2

In the post headers, *q-qealler-id* is the encrypted machine ID and *q-qealler-stub-id* is the encrypted hash of the machine ID and system time.

The request body contains encrypted and encoded system information and stolen credentials.

If the C2 server is active and data is successfully sent to the server, it will respond with the encrypted status, which looks like the following after decryption:

{"status":"2000","message":"success","extended":[],"time":1548096059}

IOCs:

hiexsgroup.co[.]uk/?_sm_nck=1

lcbodywowrksltd[.]online

willsonsolicitors[.]biz

willsonsolicitors[.]online

willsonsolicitors[.]store

mcneilspecs[.]com

mcneilspecs[.]org

mcneilspecs[.]net

prestigebuildersltd[.]com

prestigebuildersltd[.]net

larrgroup.co[.]uk/remittance%20advice.jar

prestonbuildersltd.co[.]uk/remittance%20advice.jar

otorgroup.co[.]uk/remittance%20advice.jar

ultrogroup.co[.]uk/remittance%20advice.jar

stgeorgebuildltd.co[.]uk/remittance%20advice.jar

gregoryteebuilders.co[.]uk/remittance%20advice.jar

txjxgroup.co[.]uk/remittance.jar

kingagroup.co[.]uk/remittance%20advice.jar

hiexgroup.co[.]uk/remittance%20advice.jar

salmogroup.co[.]uk/remittance%20advice.jar

4f77bf588e0b721e68971059b0cefe21 (Remittance Advice.jar)

b0ba5d6fdd26d81a6a2f050600ade3f0 (Remittance Advice.jar)

d742beba17f7893b2b4989661652a66f (Remittance Advice.jar)

61ecd8f17d405fa1c29dd78008011250 (Remittance Advice.jar)

ccac2b99cb4b72bc7728a8fc42ccc4ad (Remittance Advice.jar)

76e87575e76b2ea28e1bb49e4c280152 (Remittance Advice.jar)

7854ccf3208f805da7ec19a067ae3abe (Remittance Advice.jar)

ca741116466d5ddbcb76df00748bb885 (Remittance Advice.jar)

9b7ebeff190cef02a7c22072d3d26ab3 (Remittance Advice.jar)

639865eb7fac1b405b223cb4b7fe9ada ({E60A953D}-Remittance Advice.jar)

e6fdc2140f6047fad60720cdf2157f9c (Remittance.jar)

aae120bf74131d04e47d99b16af41120 (Remittance.jar)

3d43a83b1c8877e782ff69650ec00449 (Remittance.jar)

4d433929f175c6df366aed139bf34f85 (Remittance.jar)

2ed3b8cdc87a11437f5a15302ce047d6 (Remittance.jar)

8e0f4cb12c6f2fef3a8ff731c195843d (Remittance.jar)

fc20f0068b71cc74e9061a0ea2b5d45a (Cred_Adv043H3272.jar)

791217f372c347f53003ae8a26a2fe54 (Cred_Adv043H3272.jar)

a593cb286e0fca1ca62e690022c6d918 (7z.jar)

8d2c718599ed0aff7ab911e3f1966e8c (qealler.jar)

5a8915c3ee5307df770abdc109e35083 (main.py)

82.196.11[.]96:54869/lib/qealler

82.196.11[.]96:443/lib/qealler

128.199.60[.]13:443/lib/qealler

37.139.12.136:443/lib/qealler

192.81.222[.]28:41210/lib/qealler

37.139.12[.]169:23980/lib/qealler

37.139.12[.]169:16901/lib/qealler

176.58.117[.]125:8676/lib/qealler

176.58.117[.]125:8796/lib/qealler

146.185.139[.]123:6521/lib/qealler

159.65.84[.]42:10846/lib/qealler

159.65.84[.]42:12536/lib/qealler

139.59.76[.]44:4000/lib/qealler

128.199.60[.]13:47222/lib/7z

128.199.60[.]13:443/lib/7z

128.199.60[.]13:46061/lib/7z

82.196.11[.]96:54869/lib/7z

82.196.11[.]96:443/lib/7z

37.139.12[.]136:443/lib/7z

192.81.222[.]28:39871/lib/7z

176.58.117[.]125:8650/lib/7z

176.58.117[.]125:8796/lib/7z

159.65.84[.]42:11268/lib/7z

82.196.11[.]96:56636/qealler-reloaded/ping

37.139.12[.]136:36561/qealler-reloaded/ping

128.199.60[.]13:56636/qealler-reloaded/ping

192.81.222[.]28:46871/qealler-reloaded/ping

176.58.117[.]125:5797/qealler-reloaded/ping
