Who's Behind Your Proxy? Uncovering Bunitu's Secrets

August 5, 2015 by Malwarebytes Labs

Last updated: March 30, 2016

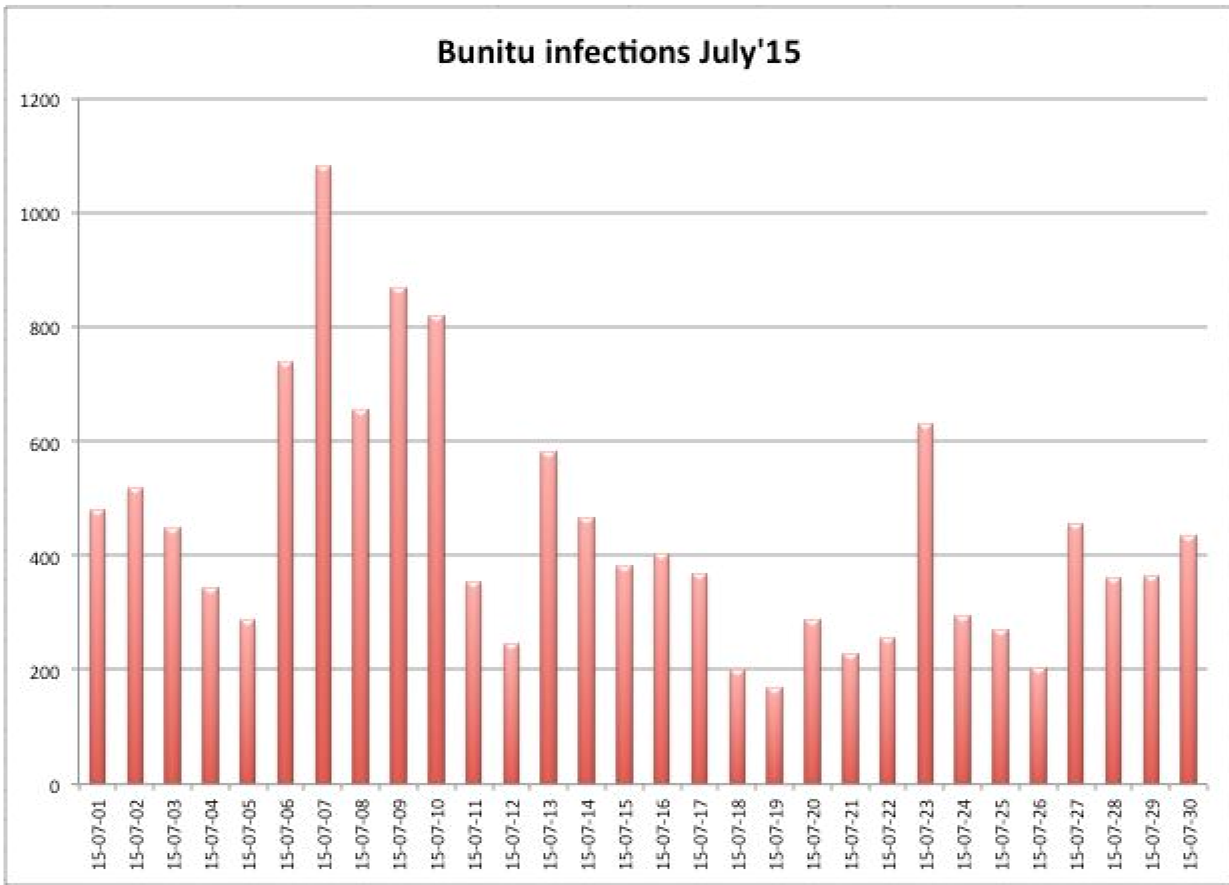
Disclaimer

The following research is the result of a collaboration with ad-fraud fighting firm Sentrant. Analysts from both the Sentrant and Malwarebytes teams have been working on the Bunitu malware and we decided to combine our efforts to provide a more complete study.

Executive summary

In our previous analysis we showed how the Bunitu Trojan was distributed via the Neutrino exploit kit in various malvertising campaigns. After spending more time analyzing the proxy, we realized that the requests we were receiving were not related to ad-fraud activity (as we initially suspected) but instead appeared to be for some sort of VPN service.

We believe that the operators of the Bunitu botnet are selling access to infected proxy bots as a way to monetize their botnet. People using certain VPN service providers to protect their privacy are completely unaware that the backend uses a criminal infrastructure of infected computers worldwide.



Number of Bunitu infections in July based on telemetry data from Malwarebytes Anti-Malware.

Not only that, but all traffic is also unencrypted – ironic for a VPN service – and could be intercepted via a Man-In-The-Middle attack. Malicious actions such as data theft or traffic redirection could therefore easily be performed.

During our research we noticed that a VPN service called VIP72 was heavily involved with the Bunitu botnet and its proxies. VIP72 appears to be a top choice for cybercriminals, as referenced on many underground forums. A recent report from FireEye on Nigerian scammers also mentions VIP72.

In this article we will review the proxy mechanism and expose the underlying infrastructure used by the Bunitu botnet. We are also sharing indicators of compromise so that end users are able to clean up their computers and no longer help to provide free exit nodes for dubious VPN services.

Technical details

Experiments performed

In order to confirm our hypothesis regarding the Bunitu proxies we developed our own Bunitu “honeypot”. We reverse engineered the Bunitu command and control (C2) protocol and developed a script that mimicked the proxy registration request.

We then used the script to register our honeypot to the Bunitu C2 and recorded the URLs of all the requests that were subsequently sent to our honeypot. A copy of the honeypot registration script can be found on our GitHub here: **bunitu_tests**.

Findings

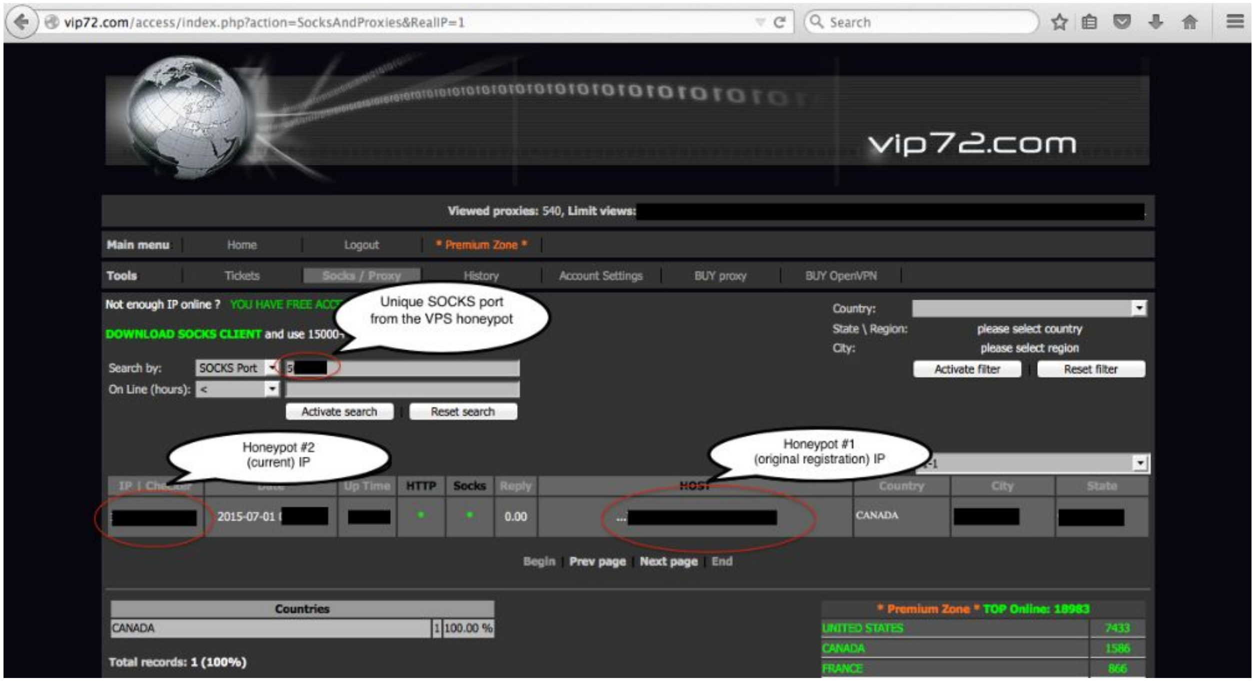
Almost immediately after registering our honeypot we realized that many of the requests we were receiving came from a VPN service known as VIP72.

Since the clients were already connected through a proxy it seemed strange that they would be visiting a second proxy, so we decided to investigate further. We also shut down the honeypot as we did not want to accidentally intercept legitimate requests from people who were unaware that they were using a botnet as a proxy.

We registered an account and logged into VIP72 and were surprised to see our honeypot proxy listed as one of their available exit IP address. Of course this in of itself is not proof that VPIP72 is knowingly using Bunitu botnet proxies.

It could be the case that they were scanning the Internet for open proxies (proxies that are listening on the Internet without requiring authentication) and using them to route traffic. However, we noticed a bug in the proxy registration system. The IP address that the proxy is initially registered from will be maintained in the VIP72 database as the “HOST” and associated with the proxy, even if the proxy moves to a new IP address.

To prove that VIP72 is using Bunitu proxies as their exit points, we registered a Bunitu proxy from one IP (Honeypot #1) then moved it to another IP (Honeypot #2) and registered it again using the same bot ID.



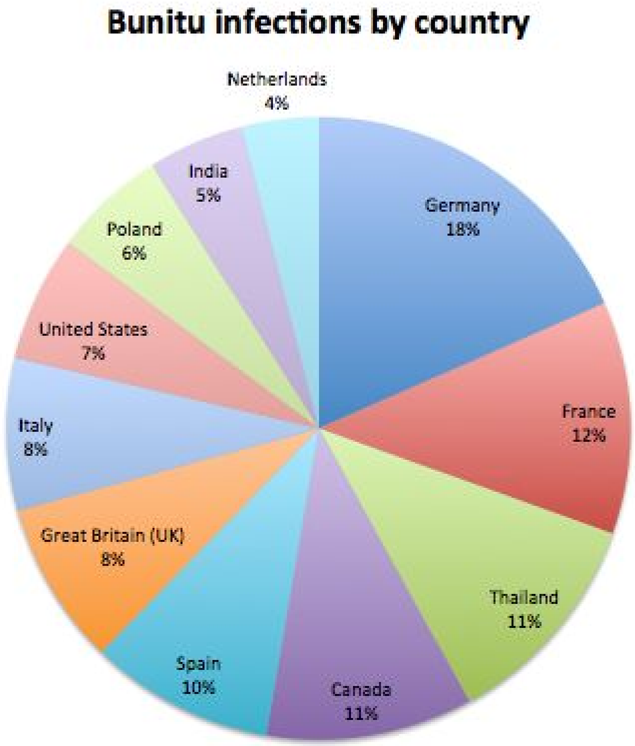
As you can see in the VIP72 proxy list, the IP for Honeypot #1 is still listed as the proxy “HOST” with the new IP for Honeypot #2 listed as the current IP.

If VIP72 was simply scanning the Internet for open proxies it is possible that they would have identified both our proxies (old and new IP) at different times. However, without having access to the Bunitu C2 server and bot ID there is no way that they could have associated those IPs to the same proxy as shown in the screenshot above.

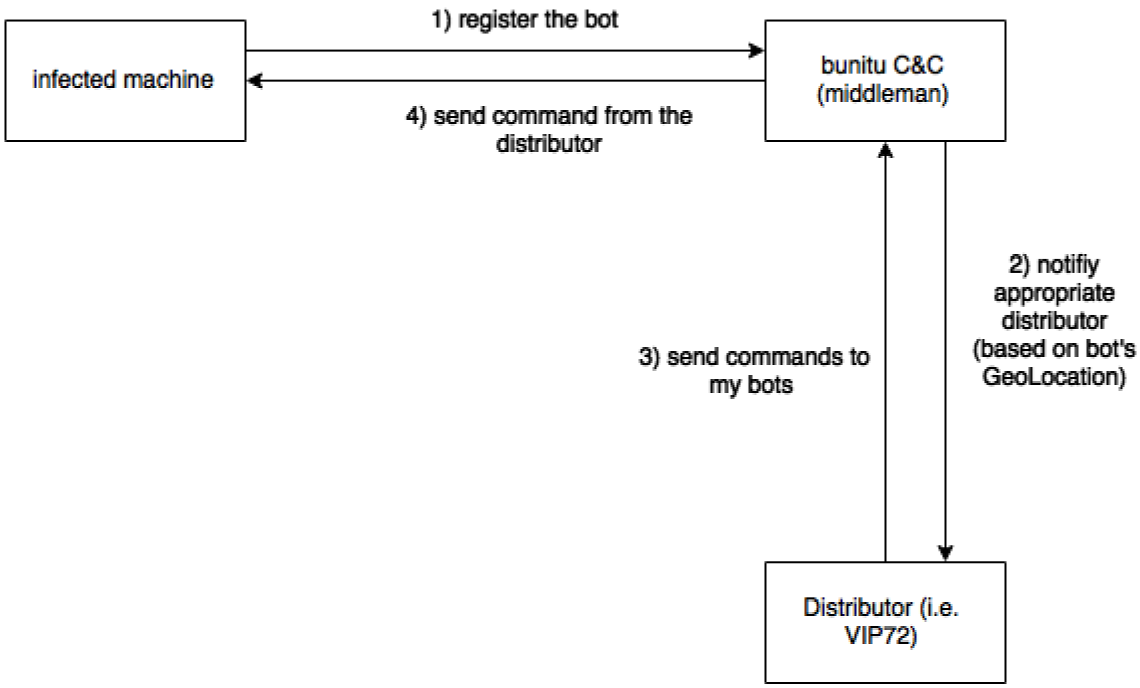
This is proof that the operators of VIP72 also have direct access to the Bunitu botnet server and use Bunitu infected hosts as proxies for their service.

Distributors

Our experiment lead us to the conclusion that distributors are different based on the geolocation of Bunitu infected machines.



In the US. and Canada, the VPN provider is VIP72, but in Central and Eastern Europe characteristics of the traffic are entirely different and suggest another VPN provider which we have not been able to pinpoint yet.



Our hypothesis is that the botnet is operated by a middleman who resells a pool of bots to various providers. Then, the bots are assigned to particular VPN networks according to their geolocation.

Proxy analysis

Two types of proxies are created on an infected machine:

1. **Standard**, by opening ports and passing traffic through them which works if the machine has a public IP address.
2. **Tunneled,** by connecting to C&C #2 and receiving commands through and passing the results back which works even if the infected machine has no public IP address.

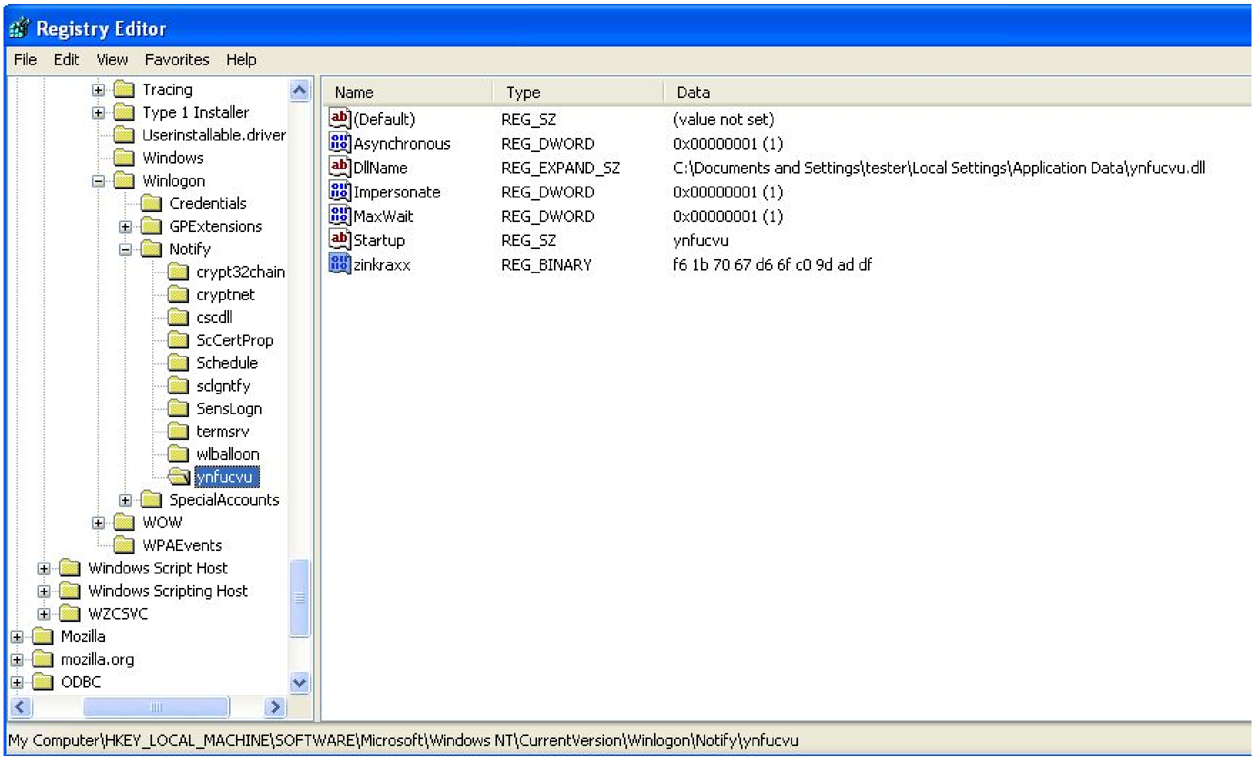
Viewing connections by *tcpview*, we can see:

bunitu...	612	TCP	0.0.0.0	12960	0.0.0.0	0	LISTENING				
bunitu...	612	TCP	0.0.0.0	43879	0.0.0.0	0	LISTENING				
bunitu...	612	TCP	10.0.2.15	1036	95.211.15.37	53	ESTABLISHED	1	37	2	87

- First 2 connections belong to standard proxies – HTTP and SOCKS (listening at **2 randomly chosen ports**).
- Second connection belongs to **C&C#2** (in this case: *95.211.15.37*) at remote port **53** (tunnel).

Connection initialization process:

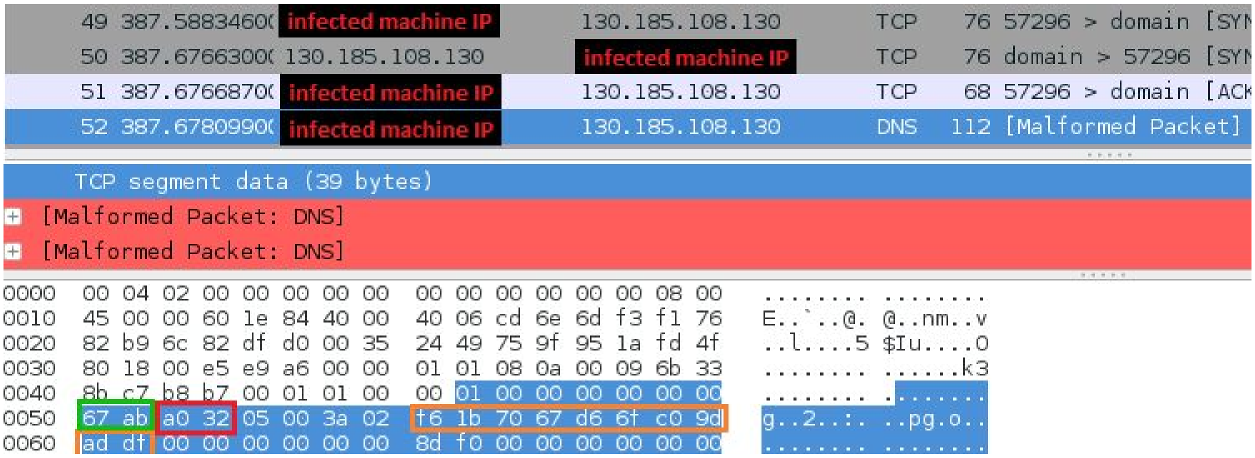
As we mentioned in the previous post about Bunitu, during installation of the Trojan, a unique ID is generated and stored in the registry:



This is an important value sent to the C&Cs and used to identify the particular bot (bot ID). It occurs in each and every packet exchanged between the bot and C&C, often in its truncated version containing only the first 4 byes, i.e.**: fb 1b 70 67** for the above case.

In short, presence of the relevant key in the packet can be used as proof that the packet belongs to the Bunitu protocol.

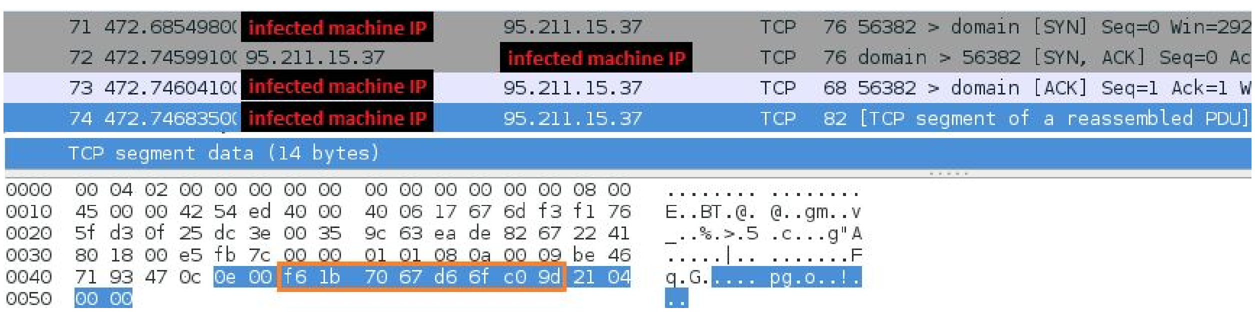
Standard proxy registration (packet sent to C&C#1):



Details:

00 01 01 00 00 01 00 00 00 00 00 00 = header (hard coded)
67 ab = socks proxy port (little endian -> 0xab67 = **43879**)
a0 32 = http proxy port (little endian -> 0x32ab = **12971**)
05 00 = hard coded value
3a = minutes since last reboot
02 = hours since last reboot
fb 1b 70 67 d6 6f c0 9d ad df = bot ID
8d f0 = hard coded unique to each version of the malware

Tunneling proxy registration (packet sent to C&C#2):



Details:

0e 00 = Length of the message (little endian) -> 0x00e0 -> 14
fb 1b 70 67 d6 6f c0 9d = bot ID, truncated (without last WORD)
21 04 00 00 = command (0x0421) "start the proxy"

Communication models: standard proxy vs tunnel:

C&C#1 is used to register standard proxies when the clients have a public IP address.

To keep the connection with C&C#1, the client periodically sends the above registration packet. Due to the fact that the infected machine has a public IP, the role of the C&C is simple: To make sure that the bot is ready to receive commands.

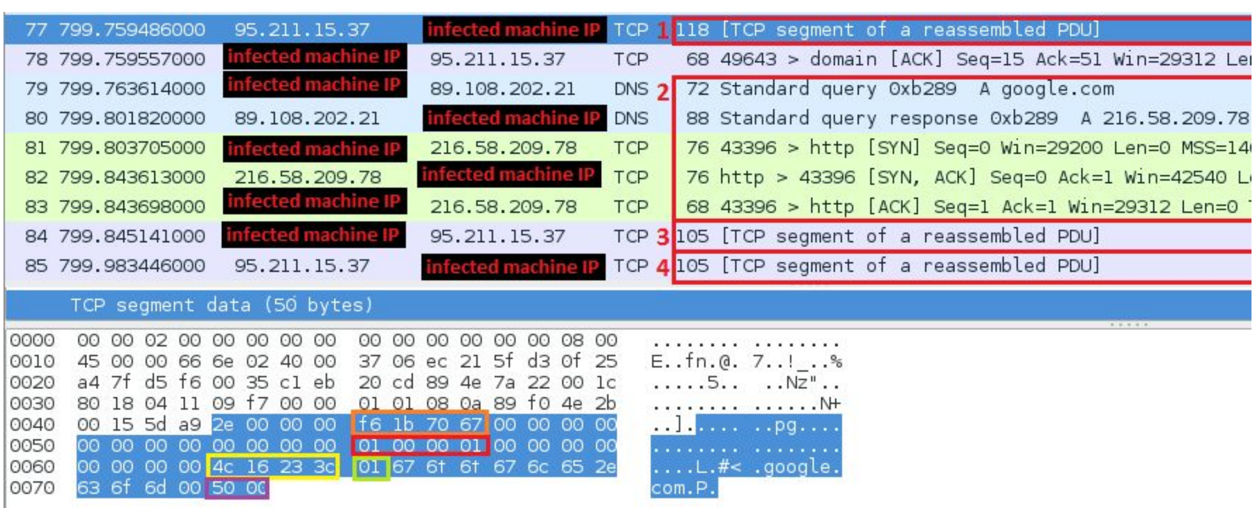
To emulate the bot's behavior, we have implemented the following script: **cnc1_test.py**. The server is just used to receive data from the client, and does not send any special response back and that's why it is not possible to verify whether the given host is a Bunitu C&C#1.

C&C#2 (tunnel) is used when the clients don't have a public IP

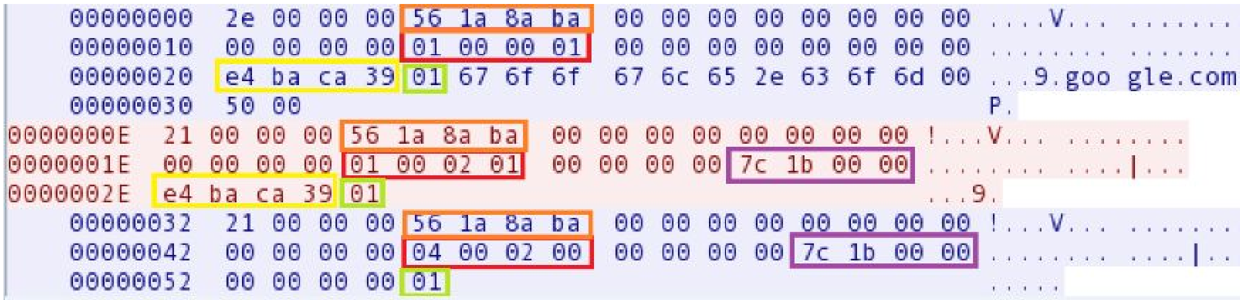
Communication with the tunnel and keeping the connection alive is more complex, as it involves a custom protocol. In this case, the server plays an active and important role: Its responses can be used to test whether a particular host is a Bunitu C&C#2. For such a verification, we have created following script: **cnc2_test.py**

After receiving the registration packet, C&C#2 tests the bot by asking it to execute a DNS query:

1. C&C#2 (IP: ***95.211.178.145***) sends a command to test the connection by querying ***google.com***
2. The bot executes the request by making the DNS query and then testing the connection with the queried IP *216.58.209.70* that belongs to *google.com*
3. The bot reports success (or failure) to C&C#2 (IP: ***95.211.178.145***)
4. C&C#2 confirms receiving the report



Packets exchanged between C&C#2 (blue) and bot (red) during this test:



Every packet exchanged between C&C#2 and a bot is prompted by a DWORD containing the length of the data that follows it (little endian). After that, there is the bot ID (truncated to first 4 bytes).

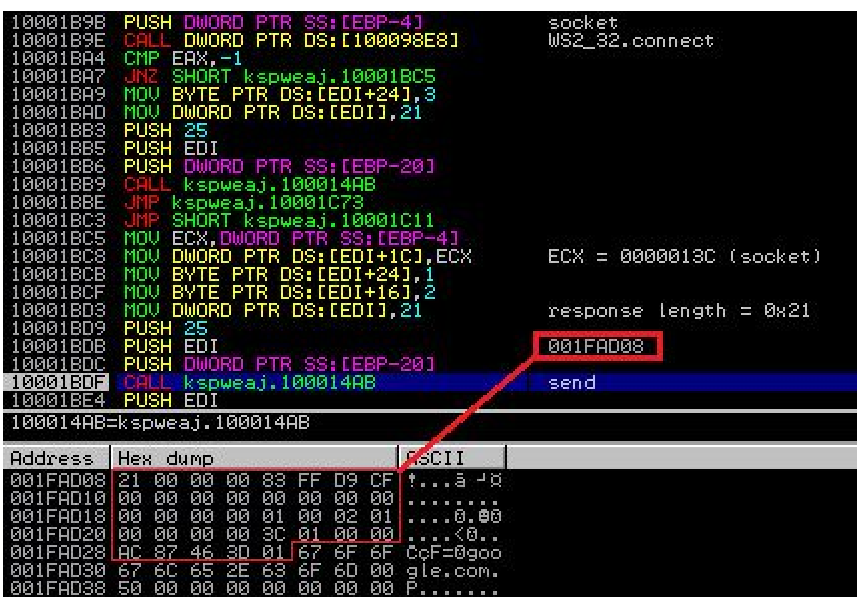
The **6-th DWORD** ***(marked red)*** packet can have the following meanings:

- **01 00 00 01**: “test the given domain”
- **01 00 02 01**: “bot reporting: domain tested”
- **04 00 02 00**: “report accepted”

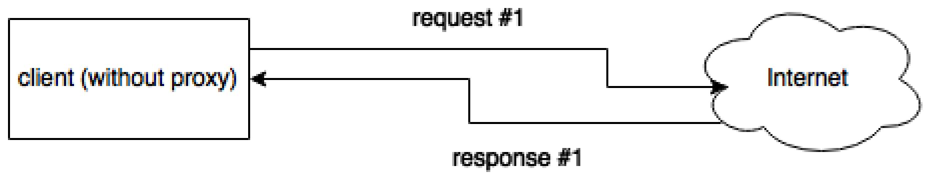
The **8-th DWORD** ***(marked purple)*** is the socket number via which the bot performed a request (to *google*)

The **9-th DWORD** ***(marked yellow)*** is a unique value generated by the C&C#2

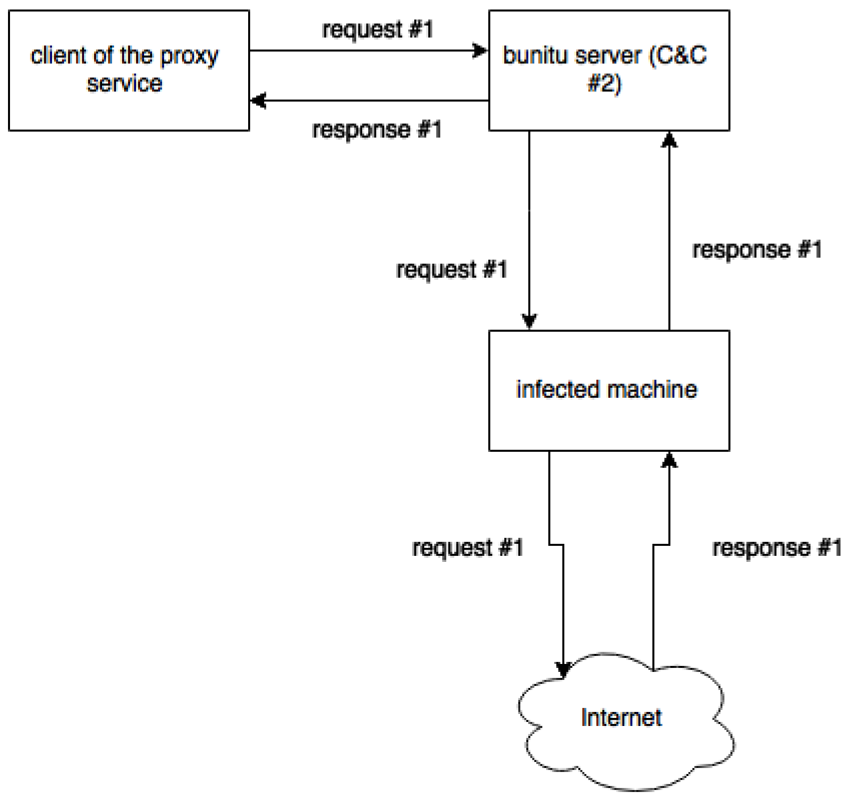
The bot tests the connection with *google*, and then builds the response for the C&C#2 (based on the request and changing the appropriate fields):



Tunneling communication process for the client



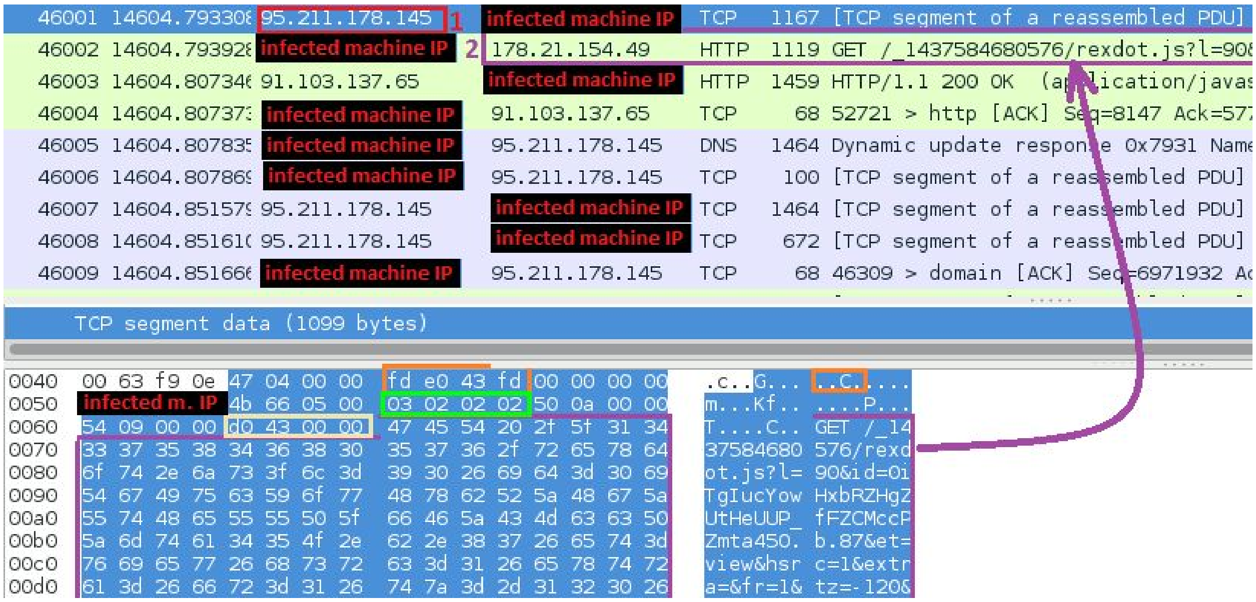
Bunitu proxy communication schema (simplified)



REQUEST (C&C#2 to bot)

The tunneled C&C receives the requests from the connected clients. It wraps them in the internal protocol and sends them to an infected machine.

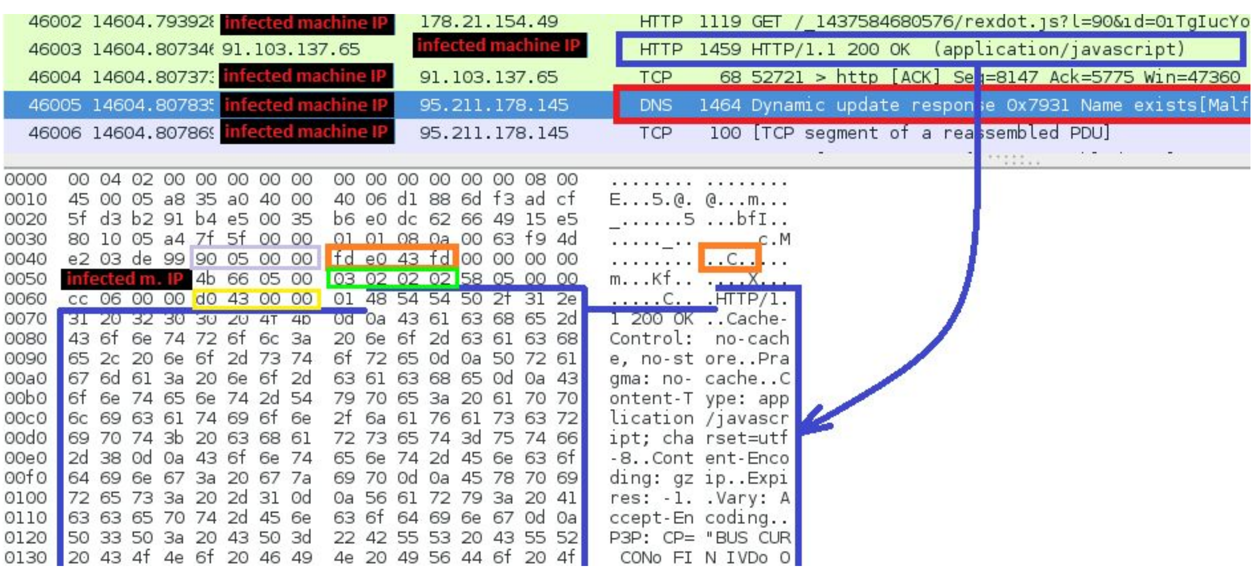
1. C&C#2 (IP: ***95.211.178.145***) gives an order to make a particular request (demanded by the proxy user)
2. The bot performs the request



RESPONSE (bot to C&C#2)

The infected machine carries out the requested operations and its IP address is visible from the outside. After fetching the results, it packs them in the internal protocol and sends them back to the C&C (tunnel).

1. The bot gets the response from the appropriate server
2. The bot passes the response to C&C#2 (IP: ***95.211.178.145***), wrapped in the internal protocol and then C&C#2 passes it to the proxy user



During the communication process, C&C#2 may request the bot to connect to additional IPs.

Here is a command from C&C#2 instructing the bot to connect to a new IP and setup the tunnel SOCKS proxy:

```
00000057  15 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000067  00 00 00 00 33 42 c7 e5  fb                      ....3B.. .
```

Details:

15 00 00 00 - message size
33 - command for “connect to new IP”
42 c7 e5 fb - new IP address (little endian)

Conclusion

Bunitu shows us how versatile malware can be, especially when compromised systems are tied together towards the same goal. While we have analyzed its main components, there is still much more that is unknown about this threat and in particular the extent of its reach or the list of VPN providers using it.

We hope that this research will help others to identify Bunitu related infections and eventually reduce the size of the botnet. We also invite security firms and law enforcement to get in touch with us via the contacts provided below so we can share with them additional intelligence.

Analyzed samples:

- Original sample (installer) md5=542f7b96990de6cd3b04b599c25ebe57 ; payload (ynfucvu.dll) md5=1bf287bf6cbe4d405983d1431c468de7
- Original sample (installer) md5=ac4e05a013705fd268e02a97c15d6f79 ; payload (lyhbyjo.dll) md5=b71832a8326b598208f49bf13e5b961f

Acknowledgements/contacts

We would like to thank the following contributors to this report:

Sentrant: Sergei Frankoff

Malwarebytes: hasherezade