



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

August 5, 2015 by [Malwarebytes Labs](#)

Last updated: March 30, 2016

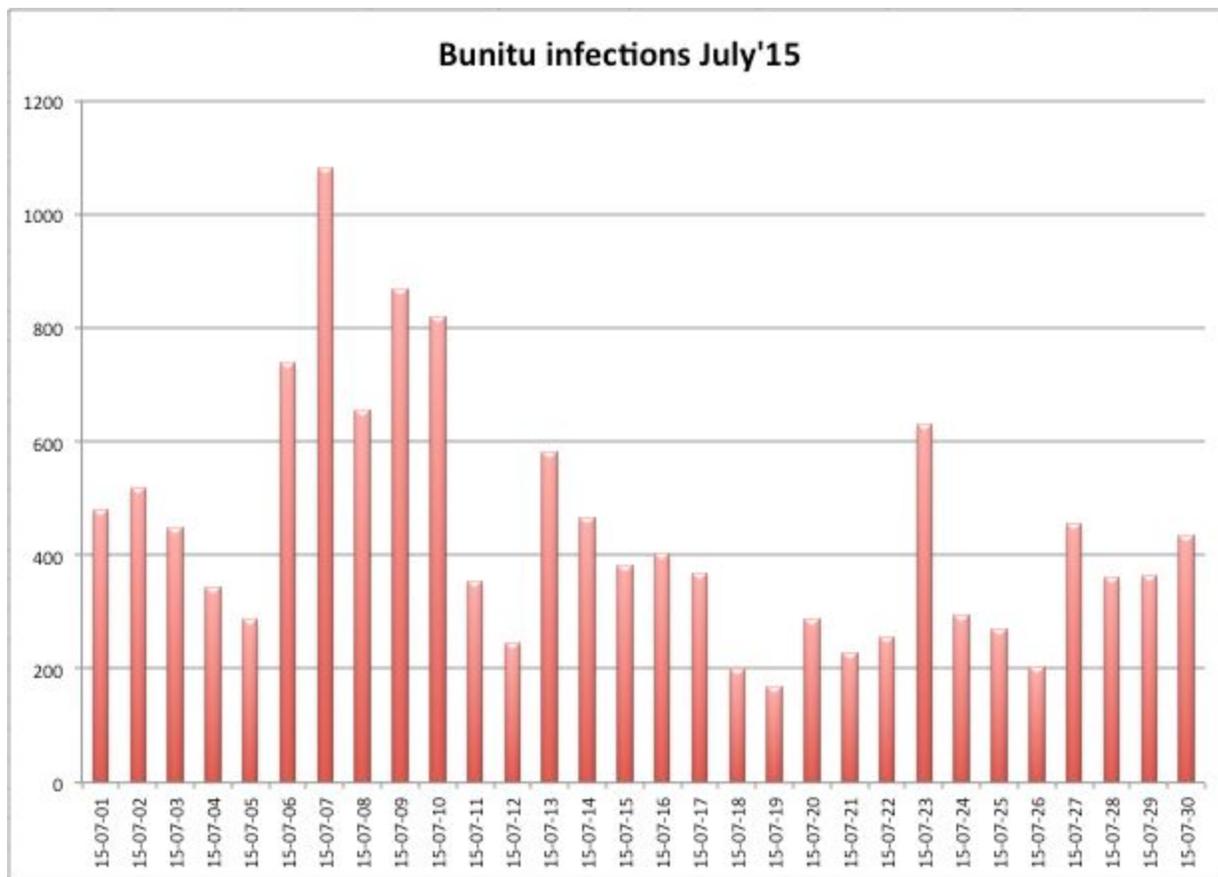
Disclaimer

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

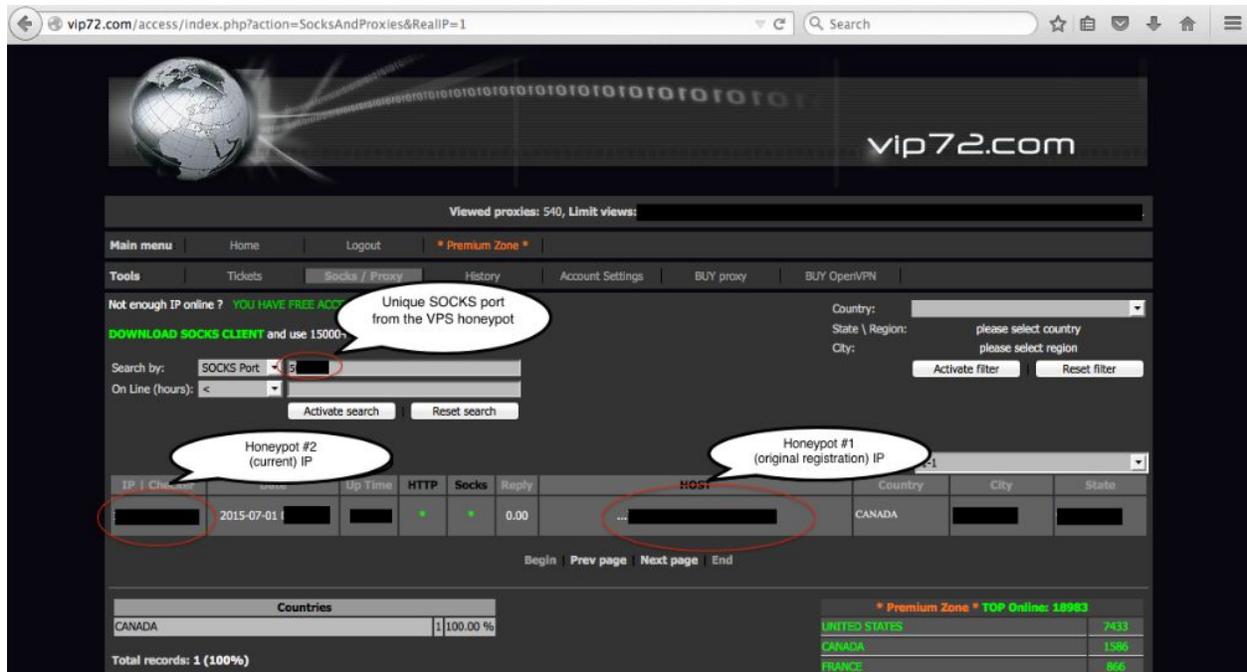
Almost immediately after registering our honeypot we realized that many of the requests we were receiving came from a VPN service known as VIP72.

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 VVIP72 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 Honeyplot #1 is still listed as the proxy "HOST" with the new IP for Honeyplot #2 listed as the current IP.

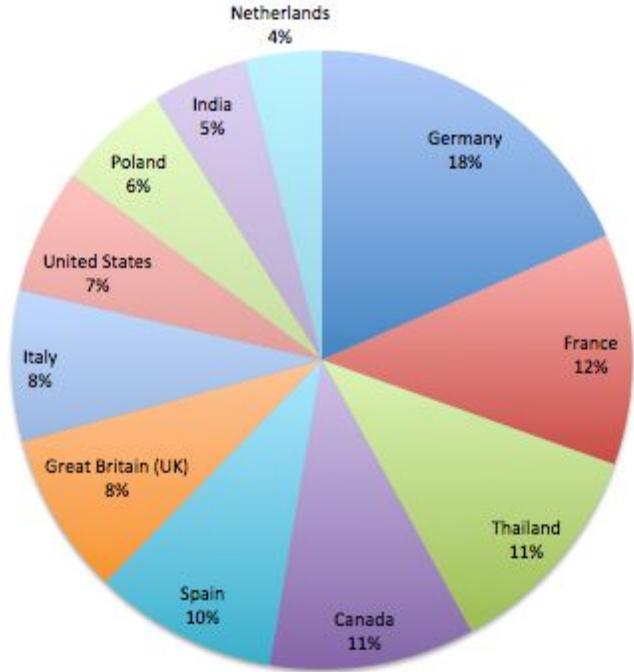
If VIP72 was simply scanning the Internet for open proxies it is possible that they would have identified both our proxies (old and new IP) at different times. However, without having access to the Bunitu C2 server and bot ID there is no way that they could have associated those IPs to the same proxy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

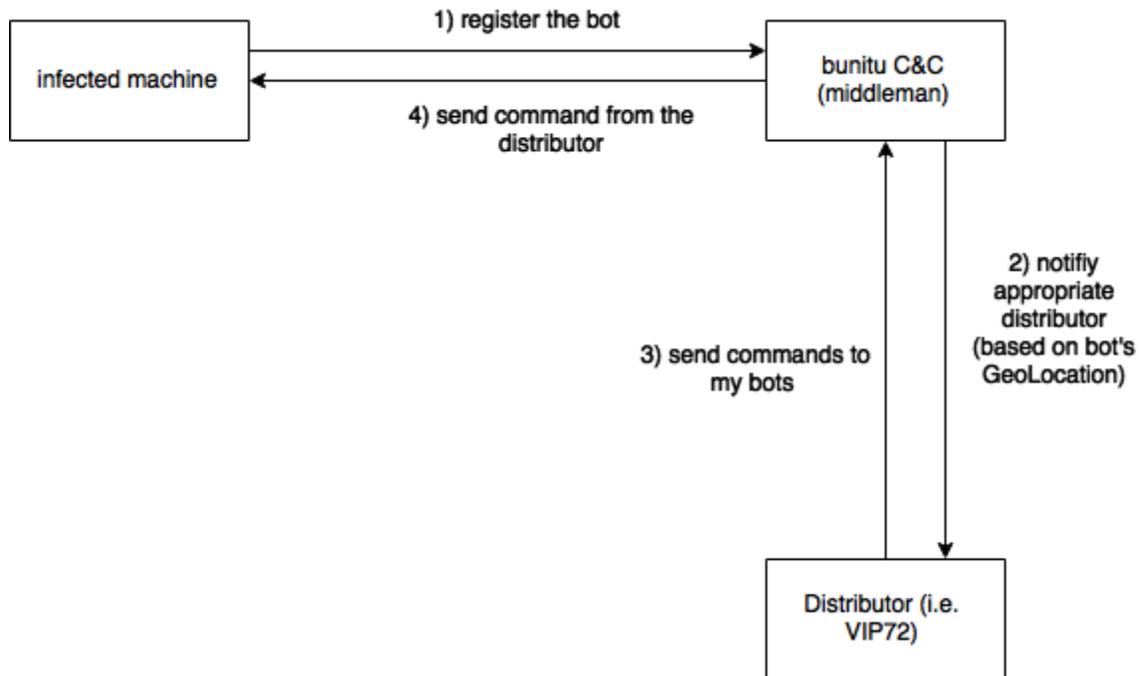
Distributors

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

Bunitu infections by country



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.

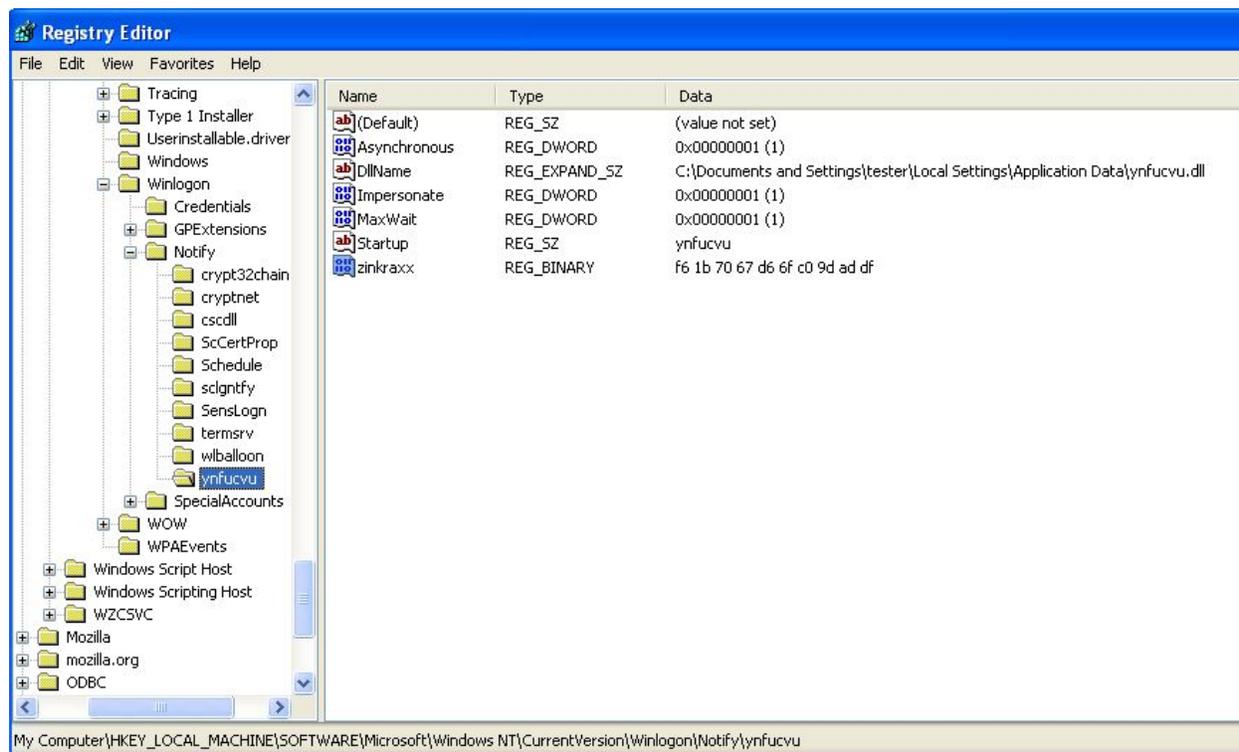
Viewing connections by *tcpview*, we can see:

📄	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 bytes, 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):

```

49 387.5883460( infected machine IP 130.185.108.130 TCP 76 57296 > domain [SYN
50 387.6766300( 130.185.108.130 infected machine IP TCP 76 domain > 57296 [SYN
51 387.6766870( infected machine IP 130.185.108.130 TCP 68 57296 > domain [ACK
52 387.6780990( infected machine IP 130.185.108.130 DNS 112 [Malformed Packet]
.....

TCP segment data (39 bytes)
+ [Malformed Packet: DNS]
+ [Malformed Packet: DNS]
.....
0000 00 04 02 00 00 00 00 00 00 00 00 00 08 00 .....
0010 45 00 00 60 1e 84 40 00 40 06 cd 6e 6d f3 f1 76 E..`..@. @..nm..v
0020 82 b9 6c 82 df d0 00 35 24 49 75 9f 95 1a fd 4f ..l....5 $Iu....0
0030 80 18 00 e5 e9 a6 00 00 01 01 08 0a 00 09 6b 33 .....k3
0040 8b c7 b8 b7 00 01 01 00 00 01 00 00 00 00 00 .....
0050 67 ab a0 32 05 00 3a 02 f6 1b 70 67 d6 6f c0 9d g..2... ..pg.o..
0060 ad df 00 00 00 00 00 00 8d f0 00 00 00 00 00 .....

```

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):

```

71 472.6854980( infected machine IP 95.211.15.37 TCP 76 56382 > domain [SYN] Seq=0 Win=292
72 472.7459910( 95.211.15.37 infected machine IP TCP 76 domain > 56382 [SYN, ACK] Seq=0 Ac
73 472.7460410( infected machine IP 95.211.15.37 TCP 68 56382 > domain [ACK] Seq=1 Ack=1 W
74 472.7468350( infected machine IP 95.211.15.37 TCP 82 [TCP segment of a reassembled PDU]
.....

TCP segment data (14 bytes)
.....
0000 00 04 02 00 00 00 00 00 00 00 00 00 08 00 .....
0010 45 00 00 42 54 ed 40 00 40 06 17 67 6d f3 f1 76 E..BT.@. @..gm..v
0020 5f d3 0f 25 dc 3e 00 35 9c 63 ea de 82 67 22 41 _..%.>.5 .c...g"A
0030 80 18 00 e5 fb 7c 00 00 01 01 08 0a 00 09 be 46 .....|.. .....F
0040 71 93 47 0c 0e 00 f6 1b 70 67 d6 6f c0 9d 21 04 q.G.... pg.o...!
0050 00 00 ..

```

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

77	799.759486000	95.211.15.37	infected machine IP	TCP	1	118 [TCP segment of a reassembled PDU]
78	799.759557000	infected machine IP	95.211.15.37	TCP		68 49643 > domain [ACK] Seq=15 Ack=51 Win=29312 Len=
79	799.763614000	infected machine IP	89.108.202.21	DNS	2	72 Standard query 0xb289 A google.com
80	799.801820000	89.108.202.21	infected machine IP	DNS		88 Standard query response 0xb289 A 216.58.209.78
81	799.803705000	infected machine IP	216.58.209.78	TCP		76 43396 > http [SYN] Seq=0 Win=29200 Len=0 MSS=14
82	799.843613000	216.58.209.78	infected machine IP	TCP		76 http > 43396 [SYN, ACK] Seq=0 Ack=1 Win=42540 L
83	799.843698000	infected machine IP	216.58.209.78	TCP		68 43396 > http [ACK] Seq=1 Ack=1 Win=29312 Len=0
84	799.845141000	infected machine IP	95.211.15.37	TCP	3	105 [TCP segment of a reassembled PDU]
85	799.983446000	95.211.15.37	infected machine IP	TCP	4	105 [TCP segment of a reassembled PDU]

TCP segment data (50 bytes)	
0000	00 00 02 00 00 00 00 00 00 00 00 00 00 08 00
0010	45 00 00 66 6e 02 40 00 37 06 ec 21 5f d3 0f 25
0020	a4 7f d5 f6 00 35 c1 eb 20 cd 89 4e 7a 22 00 1c
0030	80 18 04 11 09 f7 00 00 01 01 08 0a 89 f0 4e 2b
0040	00 15 5d a9 2e 00 00 00 f6 1b 70 67 00 00 00 00
0050	00 00 00 00 00 00 00 00 01 00 00 01 00 00 00 00
0060	00 00 00 00 4c 16 23 3c 01 67 6f 6f 67 6c 65 2e
0070	63 6f 6d 00 50 00

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

```

00000000 2e 00 00 00 56 1a 8a ba 00 00 00 00 00 00 00 00 ...V...
00000010 00 00 00 00 01 00 00 01 00 00 00 00 00 00 00 ...
00000020 e4 ba ca 39 01 67 6f 6f 67 6c 65 2e 63 6f 6d 00 ...9.goo gle.com.
00000030 50 00 P.....
0000000E 21 00 00 00 56 1a 8a ba 00 00 00 00 00 00 00 00 !...V...
0000001E 00 00 00 00 01 00 02 01 00 00 00 00 7c 1b 00 00 ...|...
0000002E e4 ba ca 39 01 ...9.
00000032 21 00 00 00 56 1a 8a ba 00 00 00 00 00 00 00 00 !...V...
00000042 00 00 00 00 04 00 02 00 00 00 00 00 7c 1b 00 00 ...|...
00000052 00 00 00 00 01

```

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):

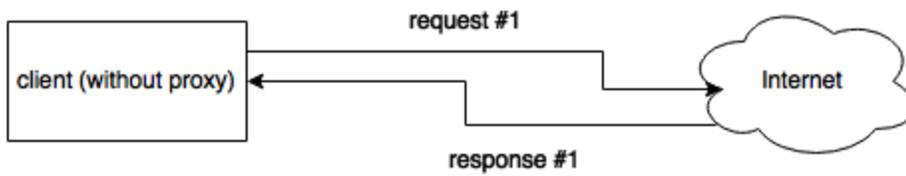
```

10001B98 PUSH DWORD PTR SS:[EBP-4] socket
10001B9E CALL DWORD PTR DS:[100098E8] WS2_32.connect
10001BA4 CNP EAX,-1
10001BA7 JNZ SHORT kspweaj.10001BC5
10001BA9 MOV BYTE PTR DS:[EDI+24],3
10001BAD MOV DWORD PTR DS:[EDI],21
10001BB3 PUSH 25
10001BB5 PUSH EDI
10001BB6 PUSH DWORD PTR SS:[EBP-20]
10001BB9 CALL kspweaj.100014AB
10001BBE JMP kspweaj.10001C73
10001BC3 JMP SHORT kspweaj.10001C11
10001BC5 MOV ECX,DWORD PTR SS:[EBP-4]
10001BC8 MOV DWORD PTR DS:[EDI+1C],ECX ECX = 0000013C (socket)
10001BCB MOV BYTE PTR DS:[EDI+24],1
10001BCF MOV BYTE PTR DS:[EDI+16],2
10001BD3 MOV DWORD PTR DS:[EDI],21 response length = 0x21
10001BD9 PUSH 25
10001BDB PUSH EDI
10001BDC PUSH DWORD PTR SS:[EBP-20]
10001BD7 CALL kspweaj.100014AB send
10001BE4 PUSH EDI
100014AB=kspweaj.100014AB

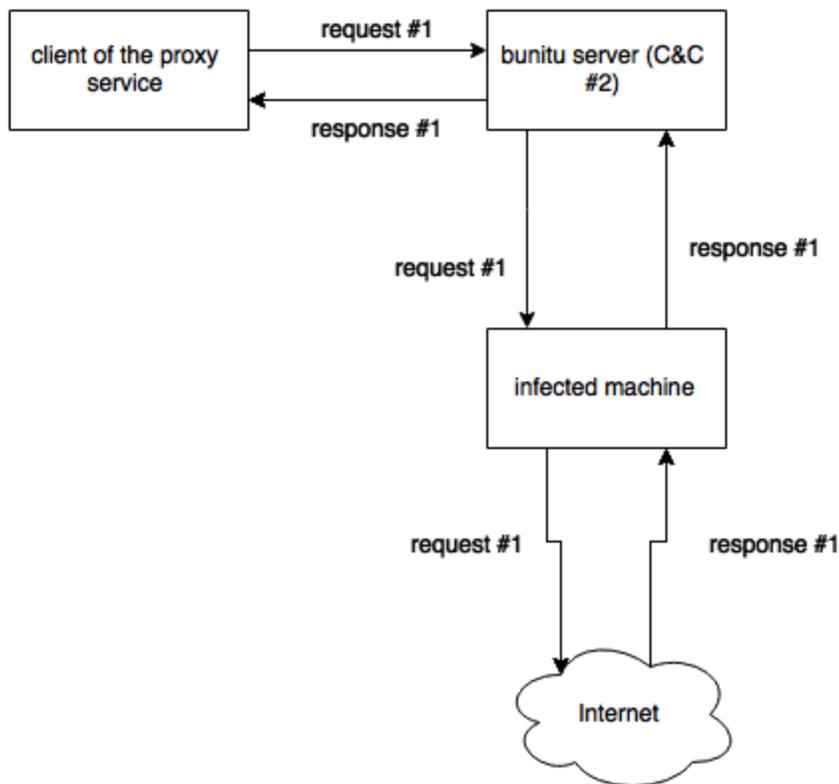
```

Address	Hex dump	ASCII
001FAD08	21 00 00 00 83 FF D9 CF	?...ã~ð
001FAD10	00 00 00 00 00 00 00 00
001FAD18	00 00 00 00 01 00 02 01	...0.00
001FAD20	00 00 00 00 3C 01 00 00	...<0..
001FAD28	AC 87 46 3D 01 67 6F 6F	ÿCF=0goo
001FAD30	67 6C 65 2E 63 6F 6D 00	gle.com.
001FAD38	50 00 00 00 00 00 00 00	P.....

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

```

46001 14604.793306 95.211.178.145 1 infected machine IP TCP 1167 [TCP segment of a reassembled PDU]
46002 14604.793926 infected machine IP 2 178.21.154.49 HTTP 1119 GET /_1437584680576/rexdot.js?l=90&id=01TgIucYow
46003 14604.807346 91.103.137.65 infected machine IP HTTP 1459 HTTP/1.1 200 OK (application/javascript)
46004 14604.807373 infected machine IP 91.103.137.65 TCP 68 52721 > http [ACK] Seq=8147 Ack=5775 Win=47360
46005 14604.807835 infected machine IP 95.211.178.145 DNS 1464 Dynamic update response 0x7931 Name exists[Malformed]
46006 14604.807865 infected machine IP 95.211.178.145 TCP 100 [TCP segment of a reassembled PDU]
46007 14604.851575 95.211.178.145 infected machine IP TCP 1464 [TCP segment of a reassembled PDU]
46008 14604.851616 95.211.178.145 infected machine IP TCP 672 [TCP segment of a reassembled PDU]
46009 14604.851666 infected machine IP 95.211.178.145 TCP 68 46309 > domain [ACK] Seq=6971932 Ack=6971932 Win=0 Len=0

TCP segment data (1099 bytes)
0040 00 63 f9 0e 47 04 00 00 fd e0 43 fd 00 00 00 00 .c..G... ..C....
0050 infected m. IP 4b 66 05 00 03 02 02 02 50 0a 00 00 m...Kf... ..P...
0060 54 09 00 00 00 43 00 00 47 45 54 20 2f 5f 31 34 T...C.. GET /_14
0070 33 37 35 38 34 36 38 30 35 37 36 2f 72 65 78 64 37584680 576/rexd
0080 6f 74 2e 6a 73 3f 6c 3d 39 30 26 69 64 3d 30 69 ot.js?l= 90&id=01
0090 54 67 49 75 63 59 6f 77 48 78 62 52 5a 48 67 5a TgIucYow HxbRZHqZ
00a0 55 74 48 65 55 55 50 5f 66 4e 5a 43 4d 63 63 50 UtHeUUP fFZCMccP
00b0 5a 6d 74 61 34 35 4f 2e 62 2e 38 37 26 65 74 3d Zmta450. b.87&set=
00c0 76 69 65 77 26 68 73 72 63 3d 31 26 65 78 74 72 view&hsrc=l&extr
00d0 61 3d 26 66 72 3d 31 26 74 7a 3d 2d 31 32 30 26 a=&fr=1& tz=-120&

```

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

```

46002 14604.793926 infected machine IP 178.21.154.49 HTTP 1119 GET /_1437584680576/rexdot.js?l=90&id=01TgIucYow
46003 14604.807346 91.103.137.65 infected machine IP HTTP 1459 HTTP/1.1 200 OK (application/javascript)
46004 14604.807373 infected machine IP 91.103.137.65 TCP 68 52721 > http [ACK] Seq=8147 Ack=5775 Win=47360
46005 14604.807835 infected machine IP 95.211.178.145 DNS 1464 Dynamic update response 0x7931 Name exists[Malformed]
46006 14604.807865 infected machine IP 95.211.178.145 TCP 100 [TCP segment of a reassembled PDU]

0000 00 04 02 00 00 00 00 00 00 00 00 00 00 08 00 .....
0010 45 00 05 a8 35 a0 40 00 40 06 d1 88 6d f3 ad cf E...S.@. @...m...
0020 5f d3 b2 91 b4 e5 00 35 b6 e0 dc 62 66 49 15 e5 _.....5 ..bfI..
0030 80 10 05 a4 7f 5f 00 00 01 01 08 0a 00 63 f9 4d ....._...c.M
0040 e2 03 de 99 90 05 00 00 fd e0 43 fd 00 00 00 00 .....C....
0050 infected m. IP 4b 66 05 00 03 02 02 02 58 05 00 00 m...Kf... ..X...
0060 cc 06 00 00 00 43 00 00 01 48 54 54 50 2f 31 2e .....C... HTTP/1.
0070 31 20 32 30 30 20 4f 4b 0d 0a 43 61 63 68 65 2d l 200 OK ..Cache-
0080 43 6f 6e 74 72 6f 6c 3a 20 6e 6f 2d 63 61 63 68 Control: no-cach
0090 65 2c 20 6e 6f 2d 73 74 6f 72 65 0d 0a 50 72 61 e, no-st ore..Pra
00a0 67 6d 61 3a 20 6e 6f 2d 63 61 63 68 65 0d 0a 43 gma: no- cache..C
00b0 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 61 70 70 ontent-T ype: app
00c0 6c 69 63 61 74 69 6f 6e 2f 6a 61 76 61 73 63 72 lication /javascr
00d0 69 70 74 3b 20 63 68 61 72 73 65 74 3d 75 74 66 ipt; cha rset=utf
00e0 2d 38 0d 0a 43 6f 6e 74 65 6e 74 2d 45 6e 63 6f -S..Cont ent-Enco
00f0 64 69 6e 67 3a 20 67 7a 69 70 0d 0a 45 78 70 69 ding: gz ip..Expi
0100 72 65 73 3a 20 2d 31 0d 0a 56 61 72 79 3a 20 41 res: -1. .Vary: A
0110 63 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 0d 0a ccept-En coding..
0120 50 33 50 3a 20 43 50 3d 22 42 55 53 20 43 55 52 P3P: CP= "BUS CUR
0130 20 43 4f 4e 6f 20 46 49 4e 20 49 56 44 6f 20 4f CONo FI N IVD0 0

```

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](#)