2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON) | 978-1-6654-0690-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/UEMCON53757.2021.9666641

BotsideP2P: A Peer-to-Peer Botnet Testbed

Adam Beauchaine[1], Orion Collins[1],and Mira Yun[1]
[1]School of Computing and Data Science, Wentworth Institute of Technology, Boston, MA
[2]Department of Computing and Data Science
Email: {beauchainea, collinso, yunm}@wit.edu

***Abstract*— The number of botnet attacks has been rapidly increasing in recent years. The threat of botnets constitutes a major security consideration for institutions and organizations concerned with the prevention of cybercrime. Among the most severe of these concerns are those regarding peer-to-peer (P2P) botnet attacks. These botnets present difficulties in local detection, as well as direct challenges to traditional network security practices due to their decentralized nature. We propose a testbed, named *BotsideP2P*, specialized in the deployment of a P2P botnet with built-in network and endpoint monitoring, so that researchers and industry professionals may gain a better understanding of these botnets in a lab environment. The botnet we have deployed implements a distributed hash table (DHT) to provide bot-to-bot encryption using the Kademlia DHT, and the Asyncio network framework is used for process handling. Emphasis is placed on documenting both the structure of the botnet deployed as well as the logging procedures implemented, and flow data and logs are included in results.**

Keywords—Botnet, P2P Botnet, Testbed, Distributed Hash Table, Kademlia, Stateful Firewall

I. Introduction

Among many disparate forms of cyber-attacks being perpetrated by attackers today, the botnet has proved particularly complex in its design, and detrimental in its application. A botnet, which can be described as a distributed system of nodes controlled by an attacker for malicious means, has grown to become one of the most concerning cyber threats for many organizations.[1] For some time now, the increasing number of botnet cyber-attacks has been cause for concern amongst researchers. Botnet attacks such as Zeus and Mirai have infected up to nearly half a million hosts at the peak of their use [2] and similar attacks continue to be cause for concern. With an ever-growing number of small computing devices such as Internet of Thing (IoT), a subset of devices expected to reach as much as 30 billion by 2030 [3], botnet attacks have more potential hosts, and thus more potential impact than ever. There is now a recognized need to develop intelligent and effective methodologies for botnet detection and prevention in modern distributed computing systems and networks. With the aforementioned increase in botnet effectiveness, the potential consequences of botnet attacks continue to rise in severity. Due to this, common botnet attack methods such as distributed-denial-of-service (DDoS) could become increasingly dangerous to organizations. This could lead to attacks similar in scope to the 2016 attack on DYN (now Oracle) distributed name servers (DNS) via Mirai, that succeeded in bringing down the entire service primarily through the use of IoT, poorly secured, networked devices [2].

Given the massive cybersecurity threat botnet attacks pose, *BotsideP2P*, a specialized botnet testbed, aims to address one of the most severe threats in peer-to-peer (P2P) botnet attacks. P2P Botnets are characterized by their decentralized nature and comparative difficulty in detection opposed to a traditional Command and Control (C&C) botnet model. *BotsideP2P* utilizes a distributed hash table (DHT) enabled P2P botnet, deployed on a secure, contained testbed for monitoring and logging network and endpoint activity. To the best of our knowledge, no similar project exists for P2P botnets. It is our hope that this work can lead to a powerful educational resource for students, in addition to some level of insight into the nature of P2P botnet attacks.

The structure of this paper is as follows: Section II introduces technical background and current methodologies of botnet detection, as well as several similar research projects. Section III provides an introduction and overview of *BotsideP2P* and how our particular botnet is used for testing in our environment. Section IV discusses networking and logging configurations of our testbed. Sections V and VI are our perceived results of this research, and conclusions we have drawn, respectively.

II. P2P Botnet Background and Design

To best understand the severity of the threat posed by P2P botnet attacks, one should first understand the traditional botnet attack model of command and control (C&C). C&C botnet attacks can be modeled using a single command and control server infecting and commanding many compromised nodes for nefarious acts such as DDoS. and keylogging attacks [2]. This model accounts for the majority of botnet attacks today [1], and while practical to execute, the model can have some intrinsic problems for attackers that might inspire the use of a P2P botnet attack. The most immediate of these problems is the capacity for loss of node connection to a C&C server, in the event of a firewall or intrusion prevention system (IPS) preventing such outgoing connections. This would leave a C&C botnet non-functioning due to being unable to receive commands from a centralized server [1]. While nodes would still be infected, they would remain largely harmless until dealt with later. In addition, C&C server demands have the capacity to be computationally expensive on C&C network communications, leading to potential problems in botnet attack scaling [4].

Solutions to the issues of C&C blocking and computational efficiency can exist in the P2P botnet model. In the P2P model, the function of C&C server is instead designated to any individual node, with processes for role election and delegation taking place during botnet initialization. This decentralized framework allows for botnet attacks to scale much faster and offers increased protection against potential system security mechanisms [2]. When any node can fill the role of C&C server, the old mechanisms of isolating and blocking traffic to the C&C server no longer apply. Because work can potentially be distributed amongst an entire infected node cluster, scaling may take place in a much more efficient manner than a traditional botnet model.

Regardless of variant, the capacity of both C&C and P2P botnet models to cause damage is widely known [2]. For example, the Mirai botnet was first introduced in 2016. The botnet had infected 213,000 devices, shortly after its code was released open source the number of devices grew to 493,000 [1]. The targets of the Mirai attack were often cyber physical systems (CPS) and IoT devices, typically possessing limited security resources. This aptly demonstrates the severity of botnet infection and attack procedures, given the attacks managed such success against major internet services. Another botnet, Zeus P2P Crimeware Toolkit [5], was a closed source program which infected vulnerable HTTP/HTTPS devices and monitored their systems for financial data which would then be captured and sent to its C&C server. It was relatively cheap to purchase and had an interactive PHP webpage with a MySQL database which pooled information stolen from zombie machines.[5] Given the effectiveness and ease of acquisition of these botnet attacks by malicious actors, security tools to aid in their detection and prevention is of the highest priority.

The primary detection mechanism for botnets such as Mirai and Zeus is network analysis. The scale of bots tends to generate a lot of traffic with constant updates and queries to neighboring nodes. Thus, some botnets do not even try to hide their presence. Mirai for example, would have each device continuing to exchange messages. Due to the sheer number of infected devices the fingerprint was easily traceable. These would lead to system administrators creating intrusion detection systems (IDS) or anti-virus routines to stop the spawn of new bots on the network. Zeus on the other hand used encrypted HTTP (Hypertext Transfer Protocol) requests between bots to exchange information. This made it avoid typical IDS or firewall rules which allowed web traffic in and out.

Given the importance of network traffic in botnet detection, the goal of our testbed, *BotsideP2P*, is to not only provide a secure environment for the deployment of P2P botnet attacks, but also to provide several sources of logging both endpoint and network flow data. Several similar projects have preceded ours and successfully expanded upon the technical knowledge of botnet attacks through methods similar to those we aim to employ. In [6], Hyslip et al. builds on existing network analysis methodologies to create a botnet detection system built on the NetFlow protocol. Whereas Ahmed et al. [7] build a similar contained botnet testbed for a variant of Mirai. However, to the best of our knowledge, no project so far has aimed to deploy a P2P botnet in a controlled testbed and offer network analysis on botnet initialization and operation. *BotsideP2P* provides both educational and technical value in developing specified security solutions to P2P botnet attacks.

III. *BOTSIDEP2P*

BotsideP2P deploys a refactored version botnet of the 2018 PythonP2P project [8]. This is a python-based botnet built on top of the Kademlia DHT, and initially used Twisted for asynchronous network event handling. We later upgraded the asynchronous engine to Asyncio. A DHT can be visualized as a hash table with key value pairs dispersed across multiple network nodes being used for the communication of encrypted data. DHTs first emerged in the early 2000s with the likes of Chord [9] and Tapestry [10] being among the first to pioneer the lookup algorithms employed by DHTs of today. Within the DHT, nodes are specified by a combination of key and value pairs which are uploaded to the distributed ledger. The ledger tracks nodes on the network and will be passed to new nodes that may join, which allows them to view and connect with their peers. Due to their lookup speed and fault tolerance, DHT sees widespread use in a variety of P2P networks and applications. For example, the file-sharing protocol like BitTorrent, and botnets such as Gnuman (Gnutella), both use DHTs to store routing information [11].

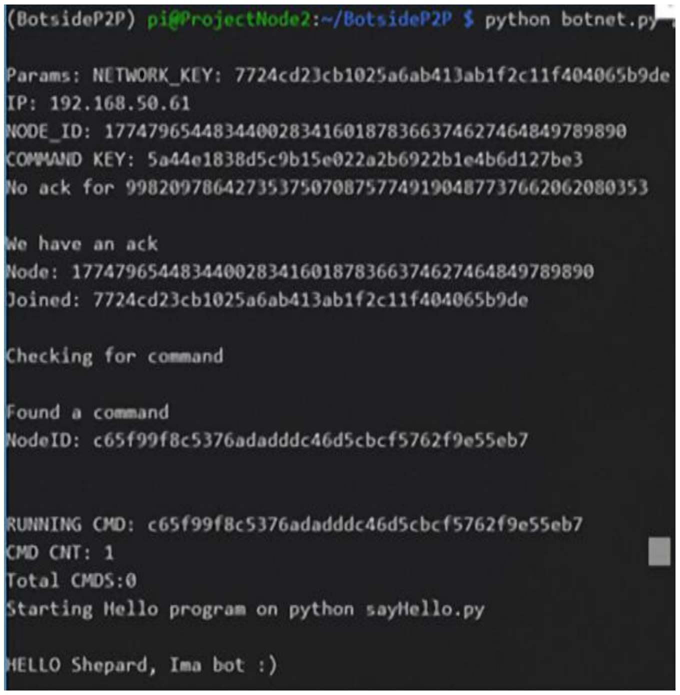
The bot network itself is written with the Kademlia DHT python implementation. In this model the secret key is a hashed network ID which a node is given in initialization that allows it to securely join the bot network. Once given this secret key, the node will contact the bootstrap server, which is hard coded into the program. At which point it will be given a unique node identification (NodeID) key which appends to the existing DHT. The bot will then populate its DHT entry with its command key, which is a hash derived from the NodeID. Kademlia offers several benefits over other P2P routing protocols. Firstly, routing table entries populate because of key lookups which decrease node discovery time. Secondly its asynchronous nature allows it to avoid timeout delays that occur when a node tries to contact a node that has dropped out of the network. Once a node misses check-in its entry is removed from the DHT, preventing duplicate entries for bots that lose connectivity. The XOR metric of its lookup algorithm allows for nodes to find low-latency paths in logarithmic time when trying to reach a destination [12]. Additionally, since NodeIDs are derived from hashes a notion of closeness can be determined since the first nodes a bot contacts will have similar hash values [11].

The nature of *BotsideP2P* setup means that nodes must be quick to initialize and highly available, as such, the use of an asynchronous network programming framework is necessary for node communication. Asynchronous is a method of structuring programs so that code can be scheduled as a task to either be performed immediately or later. This can be further extended by having tasks create separate threads that will accomplish an operation and report when they have completed. A problem encountered early on was that Kademlia DHT is no longer compatible with Twisted framework, the original network programming framework used in *BotsideP2P*. As such, we opted to refactor the project to allow for Asyncio network framework integration, the newer framework utilized by Kademlia.

Several components of *BotsideP2P* are required for functionality, nodes can be grouped as the bootstrapper node, commander node, and worker nodes in botnet structure.

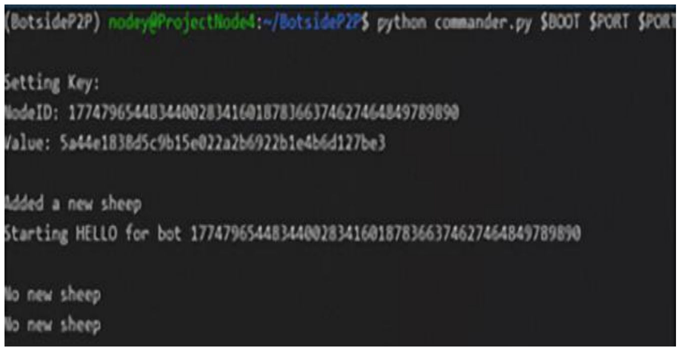
- Bootstrapper - Initialization node within the network whose sole purpose is to register new nodes and append them to the DHT.
- Commander - Commander nodes will send out commands and monitor workers. Maintains its own list of known workers to pass to the next commander when they are elected.
- Worker - Worker nodes will start a listening server and bootstrap the node onto the network if it has not already joined. It will then go into idle state until it receives a route, or payload request.

The bot programs all function within an event loop in which the bot will listen on the specified network port for communication from other nodes on the Kademlia network. This is defined as the idle state. Typically, code executes in a sequential manner on a single thread and will have trouble dealing with multiple Input/Output requests. By leveraging an asynchronous model, the bot program will continue running while it waits for new data from the network. For example, when the bot joins the network, it could be assigned task A and task B from a commander node. Async behavior allows it to start task A and schedule task B. At any point if task A needs to wait, task B will run until task A can be resumed. In this time, it could receive and schedule additional task C. Once all the tasks have been completed the bot will submit the data to the network and go into the idle state. This process can be visualized in Fig. 1, which shows the bot joining the network, populating its DHT entries, and receiving a command. Fig. 2 shows a view from the commander node, when passing a command onto some number of workers.



```
(BotsideP2P) pi@ProjectNode2:~/BotsideP2P $ python botnet.py

Params: NETWORK_KEY: 7724cd23cb1025a6ab413ab1f2c11f404065b9de
IP: 192.168.50.61
NODE_ID: 177479654483440028341601878366374627464849789890
COMMAND KEY: 5a44e1838d5c9b15e022a2b6922b1e4b6d127be3
No ack for 998209786427353750708757749190487737662062080353

We have an ack
Node: 177479654483440028341601878366374627464849789890
Joined: 7724cd23cb1025a6ab413ab1f2c11f404065b9de

Checking for command

Found a command
NodeID: c65f99f8c5376adadddc46d5cbcf5762f9e55eb7


RUNNING CMD: c65f99f8c5376adadddc46d5cbcf5762f9e55eb7
CMD CNT: 1
Total CMDS:0
Starting Hello program on python sayHello.py

HELLO Shepard, Ima bot :)
```

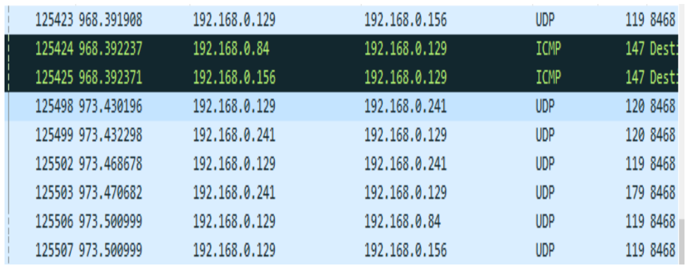
Fig. 1. Worker Node joining and checking for active commands



```
(BotsideP2P) nodey@ProjectNode4:~/BotsideP2P$ python commander.py $BOOT $PORT $PORT

Setting Key:
NodeID: 177479654483440028341601878366374627464849789890
Value: 5a44e1838d5c9b15e022a2b6922b1e4b6d127be3

Added a new sheep
Starting HELLO for bot 177479654483440028341601878366374627464849789890

No new sheep
No new sheep
```

Fig. 2. Commander Node pushes command to worker

Payloads are included in the program directory or can be downloaded remotely by a bot once the "fetch payload" command ID is passed via the DHT. When a bot receives a command, it will create a child thread to execute the payload and schedule a task to check in with the commander node once the process has stopped. From a network standpoint, botnet initialization traffic is further detailed in Section V. Botnet initialization is performed via UDP on port 8468, however any port can be selected. The transfer of the hashed network key generates several packets which were logged and displayed by the packet capture in Fig.3.



125423	968.391908	192.168.0.129	192.168.0.156	UDP	119	8468
125424	968.392237	192.168.0.84	192.168.0.129	ICMP	147	Dest
125425	968.392371	192.168.0.156	192.168.0.129	ICMP	147	Dest
125498	973.430196	192.168.0.129	192.168.0.241	UDP	120	8468
125499	973.432298	192.168.0.241	192.168.0.129	UDP	120	8468
125502	973.468678	192.168.0.129	192.168.0.241	UDP	119	8468
125503	973.470682	192.168.0.241	192.168.0.129	UDP	179	8468
125506	973.500999	192.168.0.129	192.168.0.84	UDP	119	8468
125507	973.500999	192.168.0.129	192.168.0.156	UDP	119	8468

Fig. 3. Example of botnet generated traffic via a bot network interface

Several problems were encountered during the botnet refactoring process, not strictly limited to the botnet itself. The network architecture presented several issues that will be further discussed in Section IV. Core problems consisted of layer 3 configurations on the Cisco adaptive security appliance (ASA). There were also problems ensuring botnet capability to initialize over topological configuration other than the same local area network segment. This could have been resolved, given more time, by creating a UDP hole-punch function. A UDP hole-punch allows machines to communicate through NAT, and firewalls, by opening a listening server and sending a UDP packet through it to a remote host. It creates a temporary firewall rule which allows the machines to exchange information until the rule expires. In addition, a certain aspect of *BotsideP2P* was the use of a variety of different hardware. This introduced certain challenges as not all components were compatible with the software required. However, this successfully mimicked more closely a standard enterprise environment where hardware compatibility is not guaranteed.

Finally, the code refactoring process allowed the code to be python 3.5+ compatible and use less dependencies, relying solely on Kademlia, and the built-in libraries. This change should ensure long-term stability as well as ease of use in the future, as Asyncio proved to be substantially easier to work with than Twisted framework for our purposes.

IV. *BOTSIDEP2P* HARDWARE AND CONFIGURATIONS

Our *BotsideP2P* testbed components consist of a single Cisco ASA firewall, connected to up to 5 off-the-shelf Raspberry Pi devices as detailed in Table 1 and Table 2, respectively. Our goal was to construct this testbed using primarily off-the-shelf devices for easy reconstruction in an educational environment. We decided early on that a component such as a firewall would be necessary not only for the obvious routing component of our testbed, but for flow data logging and network analysis of our testbed in use. The ASA platform was chosen primarily due to team familiarity and ease of use, in addition to features such as application layer inspection and session logging, both of which are used in botnet initialization and attack monitoring. The gateway router is a standard Linksys WRT54GL wireless router running DD-WRT [13] custom firmware. This router is also capable of session logging, a useful feature specifically for capturing the bootstrapping process.

TABLE I. LAYER 3 DEVICE SPECS

Device Name	Throughput	Concurrent Sessions	Interfaces	Virtual Interfaces
Cisco ASA 5510	Up to 150 MBPS	10,000	8 ports fast ethernet switch	3 (No trunking support) 20 (trunking support)
Linksys WRT54GL	54MBPS	128	4x100 MBPs	N.A.

TABLE II. ENDPOINT DEVICE SPECS

Device Name	SoCt	GPU	Networking	Storage
Raspberry Pi 3 x 3	Broadcom BCM2837	4x ARM Cortex-A53 1.2GHz	1GB LPDDR2 900MHz	Micro SD
Raspberry Pi 2 x 2	Broadcom BCM2837	900MHz quadcore ARM Cortex CPU	1 GB SDRAM	Micro SD

Our initial testbed design consists of 3 separate network segments connected via the Cisco ASA 3310 firewall, with each representing a separate part of our botnet system. As shown in Fig. 4, the host segment (Network ID 192.168.1.0), consisting of infected nodes used to launch an attack, the internet segment, simulating a remote connection to a victim server (Network ID 192.168.2.0), and the victim segment, where the victim server is running (Network ID 192.168.3.0). A single firewall interface is reserved for this test victim web server that may be targeted for DDoS attacks. The bot network is connected to the ASA via a wireless router, simulating a typical home network design. Our endpoint logging server also exists on this segment, running off another raspberry pi.

All Raspberry Pi devices were reformatted with Raspberry Pi OS (32 bit) and configured with XRDP [14] and SSH to allow for remote management via Windows machines natively. Firewall rules were updated to allow for remote connections to be made over port 3389 in addition to 8468 (botnet port used for node initialization). These firewall rules were consistent across all endpoints as well as layer 3 devices to allow for remote access and botnet initialization, respectively. These were the only network security rules configured on the testbed.

The Cisco ASA was configured to not only serve as a logging device for network flow data, but also as a layer 3 router for the endpoint and victim segments. This configuration, albeit atypical, allowed *BotsideP2P* to leverage the ASA's logging capabilities to their fullest, as free flowing traffic across the firewall makes botnet initialization as well as logging a more simplified process.

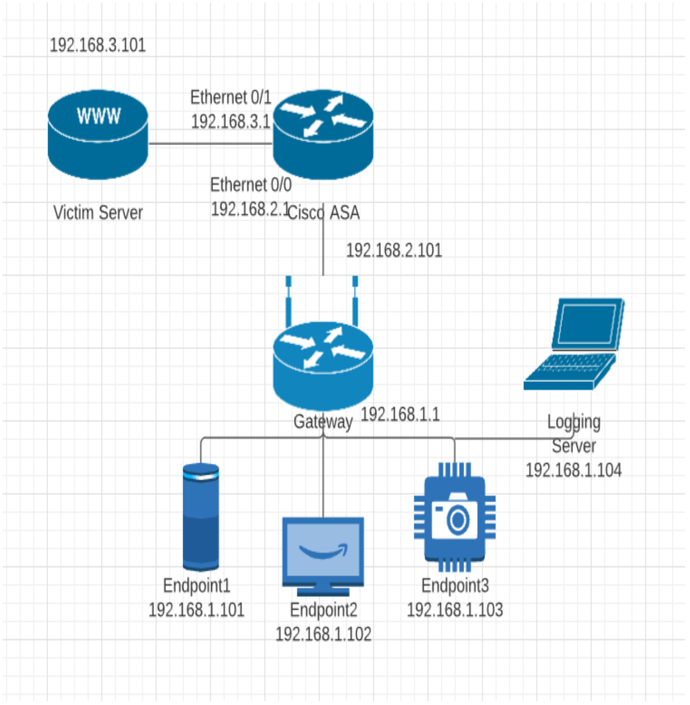
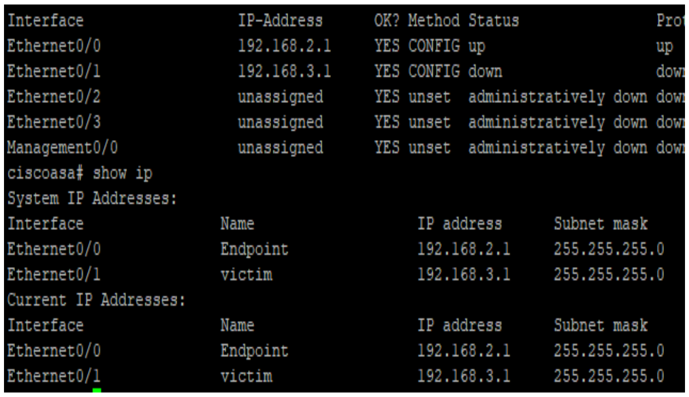


Fig. 4. First Testbed Topological Design

This configuration was accomplished by altering port security values to allow for traffic flow from victim segment to endpoint and adding ACL configuration to permit traffic down the alternate route. The routing rules themselves were configured as static, and the configuration allows for free traffic flow across the firewall. Two rules are configured specifically for logging as shown in Fig. 5 for port 8468, as well as Internet Control Message Protocol (ICMP) in the event of a DDoS.

This logging configuration was considered after a change in topological design after the first design was completed and tested. Fig 6 details a second configuration incorporating a botnet node where the old victim server was located. This second topological design was developed to enable botnet initialization logging in addition to node attack logging. By leveraging the stateful logging capabilities of the ASA to scan for botnet node initialization, early DHT session initializations may be detected. Due to localized botnet initialization requirements, this setup allowed us to produce additional flow data that will be detailed in the following section.



```
Interface              IP-Address      OK? Method Status                Prot
Ethernet0/0            192.168.2.1     YES CONFIG up                    up
Ethernet0/1            192.168.3.1     YES CONFIG down                  dow
Ethernet0/2            unassigned      YES unset  administratively down dow
Ethernet0/3            unassigned      YES unset  administratively down dow
Management0/0          unassigned      YES unset  administratively down dow
ciscoasa# show ip
System IP Addresses:
Interface              Name                  IP address      Subnet mask
Ethernet0/0            Endpoint              192.168.2.1     255.255.255.0
Ethernet0/1            victim                192.168.3.1     255.255.255.0
Current IP Addresses:
Interface              Name                  IP address      Subnet mask
Ethernet0/0            Endpoint              192.168.2.1     255.255.255.0
Ethernet0/1            victim                192.168.3.1     255.255.255.0
```

Fig. 5. ASA Firewall Rules Configuration

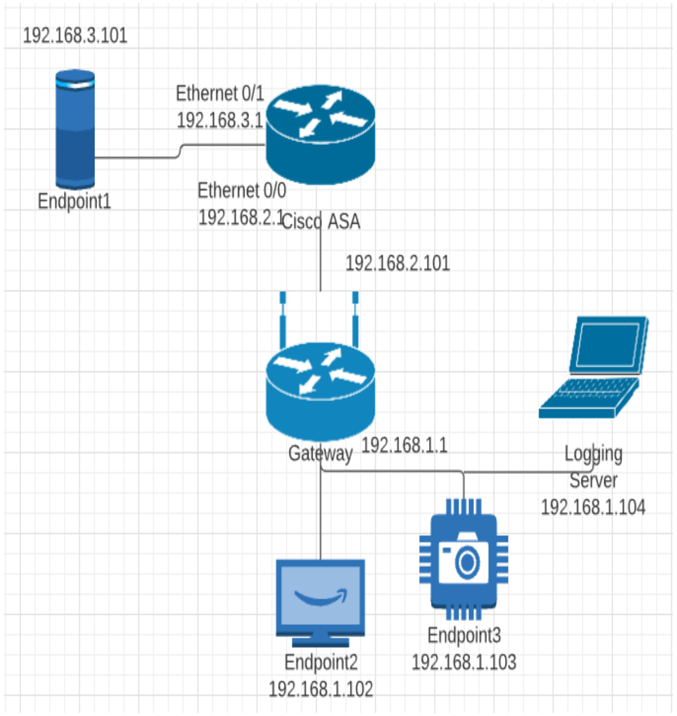
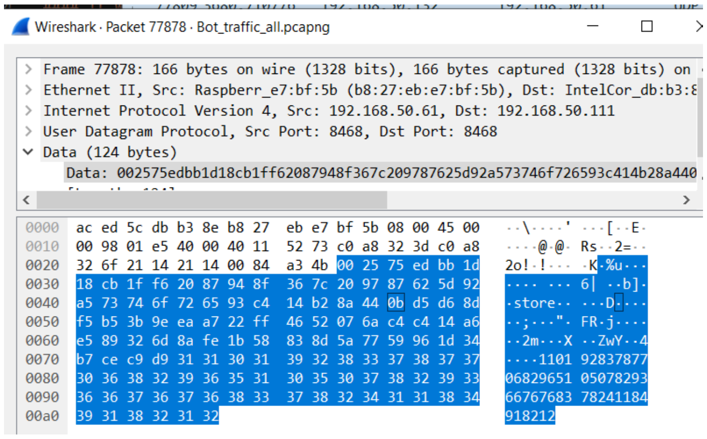


Fig. 6. Second Testbed Topological Design

V. RESULTS

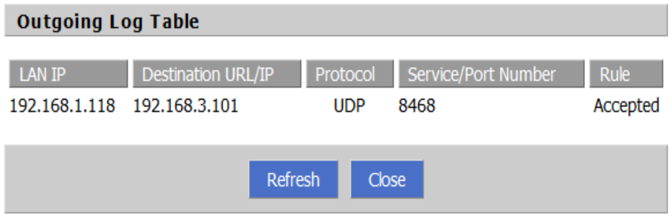
The use of testbed session logging was very much successful in botnet analysis, in part due to the Kademlia node initialization sequence. An example packet containing DHT key sharing info can be seen in Fig.7, packet capturing was performed on both endpoint devices and the gateway router. In addition, the flow data we have gathered indicates that stateful traffic inspection is a remarkably successful way of stopping DHT-enabled P2P botnet attacks. These botnet initialization sessions could be viewed successfully on both gateway router and firewall, as shown in both Fig. 8 and Fig. 9, respectively. While the router is only capable of session viewing, the firewall allows for detailed connection information during node initialization. Incorporating reports such as session monitoring and packet captures into existing logging systems could prove instrumental in identifying botnet activity. These network flow data reports provided on both the ASA and gateway router could potentially be used to identify botnets when communicating hashed values across a network during node initialization. In total, both router and firewall logging systems have the capability to be an effective early identification system for DHT-enabled P2P botnets.



```
> Frame 77878: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on
> Ethernet II, Src: Raspberr_e7:bf:5b (b8:27:eb:e7:bf:5b), Dst: IntelCor_db:b3:8
> Internet Protocol Version 4, Src: 192.168.50.61, Dst: 192.168.50.111
> User Datagram Protocol, Src Port: 8468, Dst Port: 8468
v Data (124 bytes)
    Data: 002575edbb1d18cb1ff62087948f367c209787625d92a573746f726593c414b28a440

0000  ac ed 5c db b3 8e b8 27  eb e7 bf 5b 08 00 45 00   ··\····' ···[··E·
0010  00 98 01 e5 40 00 40 11  52 73 c0 a8 32 3d c0 a8   ····@·@· Rs··2=··
0020  32 6f 21 14 21 14 00 84  a3 4b 00 25 75 ed bb 1d   2o!·!··· ·K·%u···
0030  18 cb 1f f6 20 87 94 8f  36 7c 20 97 87 62 5d 92   ···· ··· 6| ··b]·
0040  a5 73 74 6f 72 65 93 c4  14 b2 8a 44 0b d5 d6 8d   ·store·· ···D····
0050  f5 b5 3b 9e ea a7 22 ff  46 52 07 6a c4 c4 14 a6   ··;···"· FR·j····
0060  e5 89 32 6d 8a fe 1b 58  83 8d 5a 77 59 96 1d 34   ··2m···X ··ZwY··4
0070  b7 ce c9 d9 31 31 30 31  39 32 38 33 37 38 37 37   ····1101 92837877
0080  30 36 38 32 39 36 35 31  30 35 30 37 38 32 39 33   06829651 05078293
0090  36 36 37 36 37 36 38 33  37 38 32 34 31 31 38 34   66767683 78241184
00a0  39 31 38 32 31 32                                  918212
```

Fig. 7. Packet Containing Network Key Data

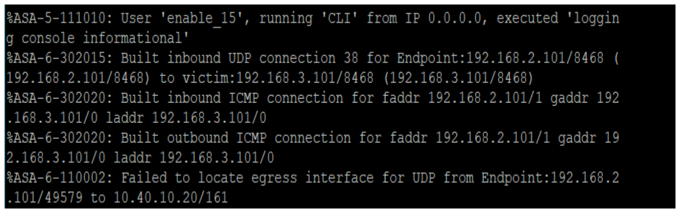
The data shown in Fig. 8 and Fig. 9 is capturable due to layer 3 node initializations, which could effectively be replicated on an IDS system on a local network. However, the capacity of *BotsideP2P* to initialize and launch attacks over multiple network segments also presents new opportunities for log aggregation using syslog output of devices such as the ASA. On a final note, regarding the application of the device, the use of application layer inspection is not presently included in our current configuration. However, this feature could be useful in monitoring and logging botnets with dynamic port assignment features in the future.



Outgoing Log Table

LAN IP	Destination URL/IP	Protocol	Service/Port Number	Rule
192.168.1.118	192.168.3.101	UDP	8468	Accepted

Fig. 8. Endpoint router logging configuration session over layer



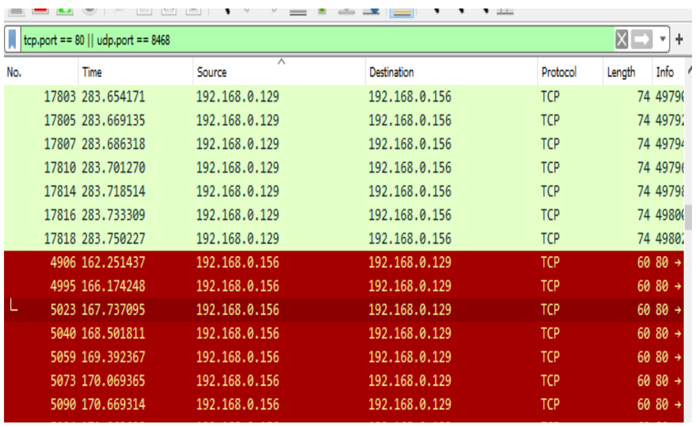
```
%ASA-5-111010: User 'enable_15', running 'CLI' from IP 0.0.0.0, executed 'loggin
g console informational'
%ASA-6-302015: Built inbound UDP connection 38 for Endpoint:192.168.2.101/8468 (
192.168.2.101/8468) to victim:192.168.3.101/8468 (192.168.3.101/8468)
%ASA-6-302020: Built inbound ICMP connection for faddr 192.168.2.101/1 gaddr 192
.168.3.101/0 laddr 192.168.3.101/0
%ASA-6-302020: Built outbound ICMP connection for faddr 192.168.2.101/1 gaddr 19
2.168.3.101/0 laddr 192.168.3.101/0
%ASA-6-110002: Failed to locate egress interface for UDP from Endpoint:192.168.2
.101/49579 to 10.40.10.20/161
```

Fig. 9. Firewall session logging on ASA

In terms of scalability, *BotsideP2P* had no issues within our small testbed, however, given the node initialization process, as well as regular update messages sent across the DHT. Scalability could be difficult, however across a mesh style of network in which routing vectors could become costly and time consuming.

When launching attacks, the botnet performed as expected, and DDoS attacks were performed successfully on the victim server as shown via a packet capture screencap in Fig. 10 with a single node attacking. The possibility exists to add additional attack types in the future such as keylogging, click fraud, and cryptocurrency mining. We would like to explore all of these in subsequent uses of the testbed. A running version of the tested featuring two nodes and the ASA may be viewed in Fig. 11.

We identified several vulnerable factors of P2P botnets as deployed in our testbed design, the most prominent one being the bootstrap server. Because the bootstrap servers are often hard coded, a potential point of failure is created for some P2P botnets, in the same vein as a traditional C&C botnet. If a bootstrap server were compromised, connectivity could be disrupted, or, more catastrophically, fake nodes could be added to the network and begin to sniff connections from peers. This could be circumnavigated by adding functionality to handle the loss of a bootstrap server, likely by reinstating a new device to fulfill the role.



```
tcp.port == 80 || udp.port == 8468

No.    Time        Source         Destination    Protocol  Length Info
17803 283.654171  192.168.0.129  192.168.0.156  TCP       74 4979
17805 283.669135  192.168.0.129  192.168.0.156  TCP       74 4979
17807 283.686318  192.168.0.129  192.168.0.156  TCP       74 4979
17810 283.701270  192.168.0.129  192.168.0.156  TCP       74 4979
17814 283.718514  192.168.0.129  192.168.0.156  TCP       74 4979
17816 283.733309  192.168.0.129  192.168.0.156  TCP       74 4980
17818 283.750227  192.168.0.129  192.168.0.156  TCP       74 4980
 4906 162.251437  192.168.0.156  192.168.0.129  TCP       60 80 →
 4995 166.174248  192.168.0.156  192.168.0.129  TCP       60 80 →
 5023 167.737095  192.168.0.156  192.168.0.129  TCP       60 80 →
 5040 168.501811  192.168.0.156  192.168.0.129  TCP       60 80 →
 5059 169.392367  192.168.0.156  192.168.0.129  TCP       60 80 →
 5073 170.069365  192.168.0.156  192.168.0.129  TCP       60 80 →
 5090 170.669314  192.168.0.156  192.168.0.129  TCP       60 80 →
```

Fig. 10. Example of HTTP Traffic During a DDoS

VI. Conclusion

The focus on peer-to-peer botnets and the exigency of continued research in the field is not misplaced. Their decentralized nature and difficulty to detect and prevent prove the severity of this emerging threat. Compounded with the proliferation of internet connected devices in the form of IoT, the severity of botnet attacks should not be understated. This is compounded by a growing number of novel botnet attacks. In this paper, we deployed a P2P botnet, *BotsideP2P*, in a controlled testbed. By providing network analysis on botnet initialization and operation, *BotsideP2P* represents an applicable form of P2P botnet detection and prevention. *BotsideP2P* configuration details provide both educational and technical value in developing specified security solutions to P2P botnet attacks.

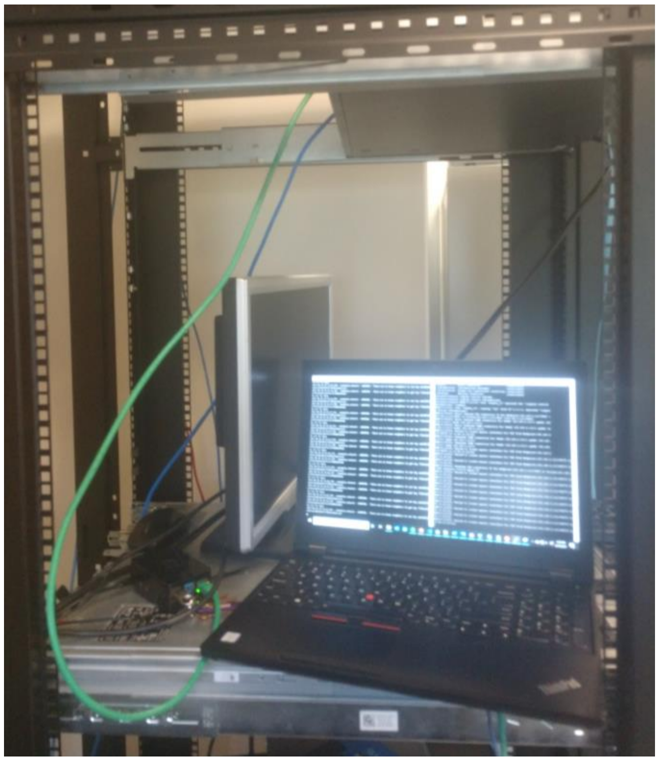


Fig. 11. Botnet Running with Multiple Nodes

There are several improvements that could be made to the testbed in its current form, with both log aggregation and alternative inspection functions being the most immediately obvious. In addition, botnet functionality should be expanded upon to include attacks such as keylogging, click fraud, and cryptocurrency mining. Application layer inspection should be leveraged in the future to test dynamic port assignment features on infected nodes. Finally, logging an increased number of nodes presents an exciting test of endpoint and network logging scalability with our current configuration.

References

[1] W. Zhang, Y. -J. Wang and X. -L. Wang, "A Survey of Defense against P2P Botnets," 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, 2014, pp. 97-102, doi: 10.1109/DASC.2014.26.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “DDoS in the IoT: Mirai and Other Botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[3] Abdur Razzaq, Mirza & Habib, Sajid & Ali, Muhammad & Ullah, Saleem. (2017). Security Issues in the Internet of Things (IoT): A Comprehensive Study. International Journal of Advanced Computer Science and Applications. 8. 10.14569/IJACSA.2017.080650.

[4] P. Wang, B. Aslam, and C. C. Zou, “Peer-to-peer botnets,” *Handbook of Information and Communication Security*, pp. 335–350, 2010

[5] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, “On the analysis of the Zeus botnet crimeware toolkit,” *2010 Eighth International Conference on Privacy, Security and Trust*, 2010.

[6] T. Hyslip and J. Pittman, “A Survey of Botnet Detection Techniques by Command and Control Infrastructure,” *Journal of Digital Forensics, Security and Law*, 2015.

[7] Z. Ahmed, S. M. Danish, H. K. Qureshi, and M. Lestas, “Protecting IoTs from Mirai Botnet Attacks Using Blockchains,” *2019 IEEE 24th*

International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2019.

[8] https://github.com/jhoward321/PythonP2PBotnet

[9] Stoica, I., et al. “Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications.” *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, 2003, pp. 17–32., https://doi.org/10.1109/tnet.2002.808407.

[10] Zhao, B.Y., et al. “Tapestry: A Resilient Global-Scale Overlay for Service Deployment.” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, 2004, pp. 41–53., https://doi.org/10.1109/jsac.2003.818784.

[11] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, “BotViz: A memory forensic-based botnet detection and visualization approach,” *2017 International Carnahan Conference on Security Technology (ICCST)*, 2017

[12] P. Maymounkov and D. Mazières, “Kademlia: A Peer-to-Peer information system based on the XOR METRIC,” *Peer-to-Peer Systems*, pp. 53–65, 2002.

[13] ddwrt.com

[14] xrdp.org

[15] Shetu, Syeda & Saifuzzaman, Mohd & Nessa, Nazmun & Salehin, Md.musfaq-Us. (2019). A Survey of Botnet in Cyber Security. 174-177. 10.1109/ICCT46177.2019.8969048.

[16] M. Thangapandiyan and P. M. R. Anand, "An efficient botnet detection system for P2P botnet," 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2016, pp. 1217-1221, doi: 10.1109/WiSPNET.2016.7566330.