Uncleanable and Unkillable: The Evolution of IoT Botnets Through P2P Networking

Technical Brief

By Stephen Hilt, Robert McArdle, Fernando Merces, Mayra Rosario, and David Sancho

Introduction

Peer-to-peer (P2P) networking is a way for computers to connect to one another without the need for a central server. It was originally invented for file sharing, with BitTorrent being the most famous P2P implementation. Decentralized file-sharing systems built on P2P networking have stood the test of time. Even though they have been used to share illegal pirated content for over 20 years, authorities have not been able to put a stop to these systems.

Of course, malicious actors have used it for malware for quite a long time as well. Being able to create and manage botnets without the need for a central server is a powerful capability, mostly because law enforcement and security companies typically take down criminal servers. And since a P2P botnet does not need a central command-and-control (C&C) server, it is much more difficult to take down.

From the point of view of defenders, this is the scariest problem presented by P2P botnets: If they cannot be taken down centrally, the only option available would be to disinfect each of the bot clients separately. Since computers communicate only with their own peers, the good guys would need to clean all the members one by one for a botnet to disappear.

Originally, P2P botnets were implemented in Windows, but developers of internet-of-things (IoT) botnets do have a tendency to start incorporating this feature into their creations. This is a big deal because IoT botnets, as they currently stand, primarily affect home routers. Since home users rarely even log into their routers, it is highly unlikely that these infections would ever be cleaned. To make matters worse, often a router belongs to the internet service provider (ISP), so that a concerned user would not even be able to access it as an administrator, let alone update or clean it. A botnet that cannot be taken down and cannot ever be individually disinfected seems to describe an eternal botnet.

Why would a user care if their home router is infected? Even if they do not mind their IP address being used to commit crimes, malicious actors could subvert their network using the router as a starting point. P2P botnets compound this by making sure those infections hang around forever. This is important for both home and corporate users, as it holds relevance in both scenarios.

It should not be surprising that we are seeing more and more P2P botnets surfacing in IoT environments. This may be an indication that a major change in the threat landscape is happening.

Current P2P IoT Botnet Landscape

We covered the general IoT botnet threat landscape in our paper “Worm War: The Botnet Battle for IoT Territory.”[1] In that paper, we described how pieces of IoT botnet malware are at war with one another as they all try to steal bots away from their competitors. We also described how there are certain “baseline” botnet malware families that are either open-source or widely available in the cybercriminal underground. The pieces of botnet malware that form the bulk of the attacks we are seeing today in the IoT threat landscape are variants of these botnet malware families: Mirai, Kaiten, and Qbot. We no longer cover these three in this paper.

In this paper, we focus on botnet malware families that have taken an extra step by implementing P2P features. This is a big step forward for any developer of a botnet to take because it ensures that the botnet, if properly nurtured, can be taken down only one client at a time. In Windows or any desktop environment, mass cleaning is possible to accomplish, although it is quite rare. For example, if within the span of a single week most antivirus vendors add detection for a newly discovered piece of botnet malware, in that short time frame the botnet malware could be significantly impacted. Here is where the IoT environment is very peculiar: When we talk about home routers — or any other smart device, really — the antivirus protection factor is just not there. This means that cleaning enough routers in a short span of time is not going to happen. We already alluded to this “uncleanable” IoT botnet malware concept in our previous work on VPNFilter, a high-profile type of IoT botnet malware that is still hanging on more than 2 years after its supposed takedown.[2]

The next big questions are: How many of these P2P botnet malware families are there? And how common are they? So far, we have seen only five families, and these families are not common at all. We do not have concrete infection numbers, however, because our monitoring capabilities are based on “protected” devices and there are not enough of these to obtain useful statistics on IoT environments.

We surmise that these five botnet malware families are not common because most of the IoT botnet threat landscape at large is completely dominated by Mirai clones or some of the other “baseline” families we described in our previous paper. Beyond the actual numbers, we find worrying the fact that the speed at which P2P botnet malware is surfacing is increasing. That probably means that there is an interest from malicious actors in making P2P botnet malware as resilient as possible to keep enabling their business plans. We review each of these five P2P IoT botnet malware families and their features in the succeeding subsections.

Wifatch

Appearing as far back as November 2014, Wifatch was the first IoT malware with P2P capabilities.[3] As claimed by its original authors, it was designed to close down the Telnet port and detect and disinfect malware.[4] The purported intention of its authors was to protect routers by infecting them with this malware. This infection, in turn, would clean them from other malware. Lastly, Wifatch would close the Telnet port to stop other malware from infecting the “protected” router.

This “Robin Hood” malware was born of the idea of a similar “good” botnet called Carna, but Wifatch has its own P2P protocol implemented in Perl. The P2P protocol Wifatch uses seems to be custom-made and is very straightforward. Since Wifatch's source code is publicly available, one can see the comments and the clarifications left there by its original authors.[5]

```
# connect types
sub H_JOIN ()          {72}    # initial join(?), want seed
sub H_NEIGHBOR_LO () {73}    # connect low priopity (no seeds)
sub H_NEIGHBOR_HI () {74}    # connect high priority
sub H_ACCEPT ()        {75}
sub H_REJECT ()        {76}

# messagetypes
sub M_DISCONNECT () {1}      # disconnect
sub M_FORW_JOIN ()  {2}      # random walk join forward
sub M_SHUFFLE ()    {3}      # update nodes
sub M_WHISPER ()    {4}      # node-to-node
sub M_BROADCAST ()  {5}      # broadcast to all nodes
sub M_PING ()       {6}      # keepalive
```

Figure 1. Basic P2P commands used by Wifatch as described in its source code

Other than the infection routine and the P2P capabilities, the only purpose of this botnet malware seems to be to protect the infected router from further infections. This appears to be a kind of academic malware and is perhaps a proof of concept. It certainly has no ties to any cybercriminal group, nor does it have a money-driven objective.

Hajime

The second P2P botnet malware family surfaced in October 2016. When it was first detected, Hajime was compared to Mirai because they targeted many of the same devices.[6] However, the two are not part of the same codebase. Unlike Mirai, Hajime does not have third-party attacking capabilities. Hajime's main difference from Mirai is its added P2P capabilities. The P2P protocol implemented in Hajime is DHT (Distributed Hash Table), the same protocol responsible for BitTorrent's distributed file system sync-up between disparate nodes, which does not need a centralized server.

The DHT protocol is probably a good choice as a P2P protocol for a botnet. These are some of DHT's features:

- It is very stable and reasonably fast.
- It is open-source, so it can be modified to suit any purpose.
- It is used by many file-sharing applications and for other uses. This means that a detection of this protocol in network traffic does not scream "malicious."

The default DHT protocol implementation as implemented in Kademlia or BitTorrent supports only five types of commands (plus the "error" reply, for a total of six). This is a very straightforward and uncomplicated protocol. It is important to mention that DHT is only the node sync-up mechanism.

```c
#define ERROR 0
#define REPLY 1
#define PING 2
#define FIND_NODE 3
#define GET_PEERS 4
#define ANNOUNCE_PEER 5
```

Figure 2. The DHT protocol P2P commands as mentioned in the Kademlia source code[7]

DHT is relatively simple, so that Hajime uses uTP (uTorrent Transport Protocol) to be able to transmit the config files on top of DHT. As part of uTP, the first thing Hajime does upon connecting to the P2P network is to exchange keys in order to get encrypted data. This ability is a big deal because it means that the commands injected by the botnet owner into the P2P network are protected. It also means that only the real botnet owner can send those commands. Someone trying to infiltrate the network by sending arbitrary commands would need to know the private key as originally designated by the botnet owner.

While it was a big botnet malware family back when it was first seen, Hajime is no longer a dangerous player in this space. In addition, Hajime never had distributed denial-of-service (DDoS) capabilities. Therefore, it was never properly monetized, and we cannot really classify it as a criminal botnet malware family. We do, however, consider it an old contender against Mirai.

Hide ‘n’ Seek

Hide ‘n’ Seek (HNS) was first seen in January 2018. This malware spreads by using vulnerabilities, two of which are specific to some IP cameras. Thus, we can surmise that this botnet malware family aims to infect more than just home routers. It is not very clear what the ultimate objective of the attacker behind this malware is. The HNS agent has a modular plug-in architecture, but so far only two payload modules have been seen: an info stealer and a Monero miner module. The latter module is uncommon and does not seem to be part of a normal HNS infection. In 2018, HNS peaked in activity with a detection of a few thousand infections.[8]

We tried to assess its current size (in September 2020, when we conducted our research) and found that it fluctuated around the low 100 nodes. Given the current low activity level of this malware, it appears as if the owners have abandoned it and are not actively adding new infections to it nor doing much with the few currently infected nodes.

Technically, HNS is interesting because it supports a custom P2P protocol that allows nodes to receive remote instructions from the network. The protocol is very simple, but it works well enough. Each node has a small list of peers that is maintained and kept up to date through frequent requests for new information from the P2P network.

The protocol itself is very simple: It has just a single-letter command, with the letter being lowercase for client-mode requests (from a single node to the P2P network) or uppercase for server-mode requests (from the network to a requesting node). For instance, a node requests a config file from the network by sending the following request to a remote node:

> "h" + the version of the config file an infected device has.

The remote node replies with:

> "H" + the version of the current config file the remote node has.

Of course, since all nodes are equal, the bot infecting a single device will respond to other lowercase, client-mode requests and will do so in uppercase, server mode.

Based on a report by Avast,[9] HNS supports other commands including:

- "y" (receive data from file) and "Y" (send data from file).
- "~" (request an address or a port of another peer) and "^" (respond with a peer's address or port).
- <random 5 to 16 bytes> (protocol challenge, ack response expected) and "O" (acknowledgement).
- "z" (report a device to be scanned).
- "m" (send a file identified by hash to ip:port).

HNS also has other interesting features: It copies itself to init.d in order to achieve persistence, it has an anti-tampering feature, and it blocks the infected machine's Telnet port to avoid other infections. This botnet malware has 10 different binaries to support different hardware architectures, although they all connect to the same botnet. Additionally, HNS can infect devices through the Android Debug (ADB) port. Its config files also contain an Elliptic Curve Digital Signature Algorithm (ECDSA) signature, which helps in mitigating this botnet infection.[10]

Ultimately, HNS could be used for a variety of purposes given the flexibility that its plug-in architecture allows. In practice, however, we have not heard reports of this botnet malware being used for criminal actions. HNS seems to have a high criminal potential that has, fortunately, never been realized.

Mozi

Mozi was first seen in the wild in September 2019.[11] It is an effective botnet malware family with most of the modern features in the IoT environment. It infects devices with weak credentials by using a hard-coded password list and specific device vulnerabilities. Once it manages to run code in the victim system, it binds local port 14737 to serve as a download site for the victim system to get and run the payload. The file served, called mozi.a, is the basis of this botnet malware's name. There are three known versions: 0s, 1, and v2. Here we describe the behavior of v2.

As usual with botnet malware of its ilk, Mozi blocks SSH (Secure Shell) and Telnet to "protect" the victim from future infections. It uses the DHT protocol to download and verify a config file. From then on, it runs the commands in the file. Inside the regular DHT packets, signaling is customized with specific commands.

The author of Mozi made modifications to the default DHT implementation shown in Figure 2.[12] The modifications are:

- The removal of ANNOUNCE_PEER (command 5), which is not supported.
- The addition of SEND_CONFIG (command 6), one of the two new commands specific to this botnet malware. Upon receiving this request, the node sends back the config file.
- The addition of RECV_CONFIG (command 7), the other new command specific to this botnet malware. This packet contains an encrypted config file.

The config file being shared is encrypted with the ECDSA384 algorithm. Once decrypted, this config file can instruct the node to perform certain actions. There are five “subtask” actions supported by this botnet malware. These include:

- [atk]: DDoS attack.
- [ud]: Update.
- [dr]: Execute payload from specific URL.
- [rn]: Execute system or custom commands.
- [idp]: Report bot info.

Given the number of commands it supports, Mozi is well suited for criminal activity. There is a substantial change between the previous botnet malware families and this one in the level of criminal scenarios this code could be used for. Whereas HNS has a modular architecture that could be used for criminal activities, Mozi appears specifically designed for them. Its main function is to conduct DDoS (the central monetization technique used in most IoT botnets) or to run more malicious code on the victim machine.

HEH

HEH was first seen in October 2020.[13] HEH, at a glance, is another IoT botnet malware family, fashionably developed in the now-popular language Go. HEH scans and infects devices through ports 23 (Telnet) and 2323 (perhaps another Telnet port variant) using brute-force passwords and hard-coded credentials. Two features of its infection procedure stand out as different: It scans for infectable machines by choosing IP addresses at random, and it uses an algorithm to derive the P2P port from a given IP address. This means that it does not have to list IP addresses and ports. Just knowing the IP address of a peer allows HEH to automatically know the related port.

Once the victim has been infected, the bot binds itself to the related port and tries to connect to peer nodes. In addition, HEH stops the local HTTP service and starts one of its own, which serves as the download site for new infections. This also means that the original admin webpage of the router might become inaccessible.

The P2P protocol is customized. It has five opcodes: four for node synchronization and one for receiving external “commands,” as listed in Table 1.

1	Run Command	Instruct bot to do something (see command list below).
2	Ping	Check peer availability.
3	Pong	Respond to Ping.
4	Announce	Announce oneself as a peer. It adds this peer to self's peer list.
5	PeerUpdate	Let peer update its peer list.

Table 1. The five opcodes used by HEH

The four synchronization instructions are as simple as the ones DHT implements (as shown Figure 2). The most interesting part of this list is the "Run Command" opcode. This opcode allows for interesting functionality. Table 2 lists the possible commands that a HEH bot can be instructed to run.

0	Restart	Restart the bot.
2	Exit	Exit bot.
3	Attack	Launch an attack (not yet implemented).
4	Execute	Execute shell command.
5	Print	(Not yet implemented.)
6	PeerUpdate	Update peer list.
7	UpdateBotFile	Bot downloads link and updates this "bot" file.
8	SelfDestruct	Wipe out the device.
9	Misc	(Not yet implemented.)

Table 2. The possible commands a HEH bot can execute

This command list is interesting because it lets us see the potential purpose of the botnet. Not only does it allow any shell command to be run, but in the future, it will have the capability to launch attacks on third parties. It can also cause the device to self-destruct, which is unusual for IoT bots.

We can see that this botnet malware is made to be monetized. Its infections are not casual infections, but rather a financially driven project.

P2P Botnet Evolution: Windows vs. IoT

The peculiarity of P2P malware was that it was being developed and refined as P2P software and protocols for it were being developed. For instance, BitTorrent came out in 2001, and the first P2P-enabled malware was seen in 2003. That same year, DHT appeared with Kademlia. The first malware that used this protocol came 4 years later, in 2007, in the form of Peacomm.

It took several years for malware writers to digest these new concepts and realize that they could also be used for malicious purposes. In 2007, Julian Grizzard et al. published a study on P2P botnets that included a table summarizing P2P technologies and malware, as shown in Table 3.[14]

Date	Name	Type	Distinguishing Description
12/1993	EggDrop	Non-Malicious Bot	Recognized as early popular non-malicious IRC bot
04/1998	GTbot Variants	Malicious Bot	IRC bot based on mIRC executables and scripts
05/1999	Napster	Peer-to-Peer	First widely used hybrid central and peer-to-peer service
11/1999	Direct Connect	Peer-to-Peer	Variation of Napster hybrid model
03/2000	Gnutella	Peer-to-Peer	First decentralized peer-to-peer protocol
09/2000	eDonkey	Peer-to-Peer	Used checksum directory lookup for file resources
03/2001	Fast Track	Peer-to-Peer	Use of supernodes within the peer-to-peer architecture
05/2001	WinMX	Peer-to-Peer	Proprietary protocol similar to FastTrack
06/2001	Ares	Peer-to-Peer	Has ability to penetrate NATs with UDP punching
07/2001	BitTorrent	Peer-to-Peer	Uses bandwidth currency to foster quick downloads
04/2002	SDbot Variants	Malicious Bot	Provided own IRC client for better efficiency
10/2002	Agobot Variants	Malicious Bot	Incredibly robust, flexible, and modular design
04/2003	Spybot Variants	Malicious Bot	Extensive feature set based on Agobot
05/2003	WASTE	Peer-to-Peer	Small VPN-style network with RSA public keys
09/2003	Sinit	Malicious Bot	Peer-to-peer bot using random scanning to find peers
11/2003	Kademlia	Peer-to-Peer	Uses distributed hash tables for decentralized architecture
03/2004	Phatbot	Malicious Bot	Peer-to-peer bot based on WASTE
03/2006	SpamThru	Malicious Bot	Peer-to-peer bot using custom protocol for backup
04/2006	Nugache	Malicious Bot	Peer-to-peer bot connecting to predefined peers
01/2007	Peacomm	Malicious Bot	Peer-to-peer bot based on Kademlia

Table 3. A descriptive timeline of the early days of P2P technologies and malware

The evolution of P2P botnets in IoT environments has been much faster than in the Windows environment. Since all the concepts are already clear and well developed, malware writers just need to start implementing them into their creations. For instance, it took Peacomm 4 years to start adding P2P features after the publication of Kademlia. In contrast, Wifatch needed only 2 years after the first ever IoT botnet malware family (Linux.Aidra) to do the same.

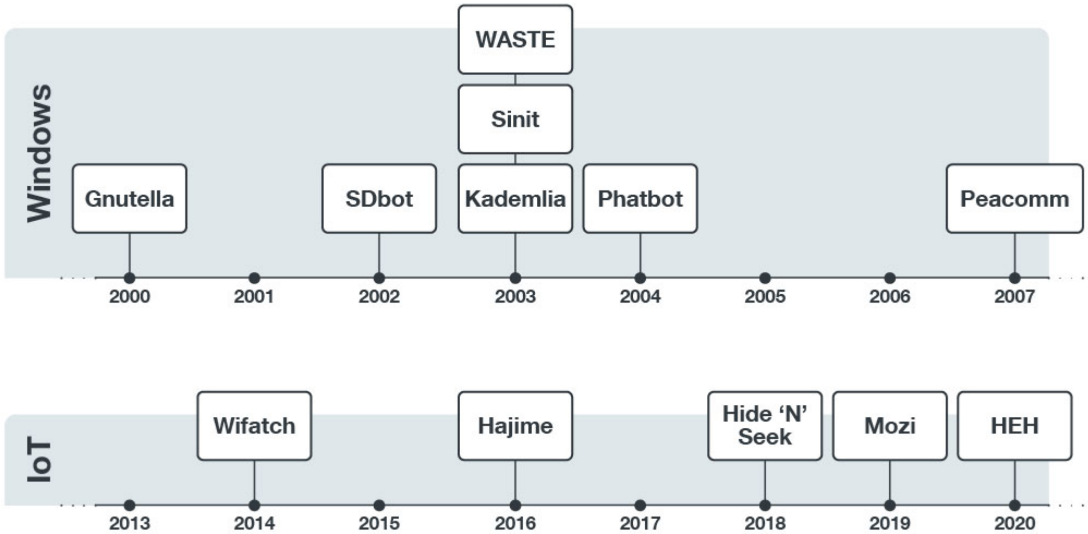

Figure 3. A comparison of the development of P2P malware in Windows and IoT environments

Future Evolution of P2P IoT Botnets

If we don our futurologist hats and try to predict where attacks will go, there are several avenues we can foresee.

First, we know that cybercriminal attacks are heavily motivated by money. Wherever there is a clear and easy monetization method, cybercriminals will follow. Right now, these methods include third-party attacks (commonly DDoS attacks) and virtual private network (VPN) services. There are more exotic monetization methods, but these two are the most popular we have seen. It stands to reason, then, that for these attacks to really take off, cybercriminals would need to find a novel way of making money off infected routers and other IoT devices, or perhaps the old methods would need to become more interesting.

When we try to think of new methods to monetize an infected router, one comes to mind: looking into the infected router's network. A router is the way into a home network. Once a router becomes infected by malware, it would be possible for the malware to do man-in-the-middle (MitM) attacks and steal information or perhaps inject extraneous elements into the return traffic. Dynamic rewriting of webpages could allow router malware to include JavaScript-based cryptocurrency mining or click fraud elements to the pages. The possibilities are plenty. Cybercriminals could sell the desktop infections or the stolen information, or find some other way to monetize the malware.

The attackers do not need to actually intercept the traffic, especially if this proves hard because of TLS (Transport Layer Security) encryption. They can also simply use the router as a beachhead to move laterally to other unsecure devices on the network, such as other poorly configured IoT devices or the home user's corporate laptop. This would be similar to the successful approaches common in today's ransomware or advanced persistent threat (APT) intrusions.

If this route of infection becomes a reality, an attacker would not need to decide between infecting a router and infecting an individual computer. Compromising a router could be the gateway to taking over any individual computer inside the network.

How far-fetched or difficult would it be to implement these attacks? It would not be easy, but it is not an impossibility. The malware would need to be able to intercept the traffic coming from inside the network and inject arbitrary elements into every webpage it returns. From a technical perspective, this would probably entail messing with the router's protocol stack, which is within the realm of possibility. Looking at the logs of webpages accessed in order to steal valuable information is also something that could be done relatively easily.

This is possibly the future of home attacks because the Wi-Fi router is the most vulnerable element of a home network. Also, it should be noted that hitting the user's home nowadays is much more dangerous than in pre-Covid-19 times. It is no longer possible to differentiate personal or consumer attacks from attacks against organizations, as attackers would now be targeting the weakest spot to get into a corporation in today's work-from-home (WFH) environments. In fact, companies should care more about this problem than individual users. It would not be surprising for companies to start mandating protection for their employees' WFH setups — including routers — at some point.

To make matters worse, new cybercrime trends tend to explode rapidly and not grow gradually. Malicious actors tend to copy the most successful attacks because there is no lawsuit to worry about for copying competitor strategies. This means that the success of one group in this area could lead to an overnight change — turning IoT botnets from being an annoyance into being one of the largest and most difficult problems that enterprises might have to deal with.

We predict that once one P2P botnet takes the next step and succeeds, all the other botnets will start adding P2P capabilities and these botnets will become the new baseline. Likewise, if this “criminal home hub” concept materializes, this kind of attack can go from purely hypothetical to defining the next decade of IoT attacks.

Conclusion

IoT botnets based on home routers are on the rise. The main reason for this is that these devices are normally unmanaged and home users barely ever log into them to check their status after they are installed. Not only do they not get updated often (enabling infections) but, once infected, they also do not get cleaned up. The fact that some uncleanable botnets are going the P2P route only compounds this problem. Since these infections cannot reliably be dealt with by the user from the inside, security companies, computer emergency response teams (CERTs), and law enforcement agencies become responsible for taking down C&C servers and other criminal networking infrastructure. P2P IoT botnets render those cleaning actions from the outside impossible. They are both uncleanable and unkillable — a threat that could remain unsolved for a very long time.

As most cybercriminals start implementing P2P features into them, these botnets could become a painful problem in the future. All that is needed for them to become a real problem is a better way of monetizing router infections. The likeliest scenario would be for cybercriminals to start attacking from the router in as opposed to the traditional way of making money from the router out through DDoS attacks and VPN access. This “from the router in” scenario would involve looking at the user data and perhaps modifying traffic data to further infect individual computers within the router’s internal network. This is what we call the “criminal home hub” attack.

If both of our predictions come true, the resulting IoT landscape would be much more different from the current one. Uncleanable and unkillable botnets coupled with additional “from the router in” monetization techniques could radically change IoT infections as we know them.

Appendix

Wifatch Hashes

02171868a19aa4a2b223d07b5534bfe78df167e05b249ed90c762c5173ea4093

0afbfe71c6a8c6151a20f7f873b50fcfcaa7dac7b4f76dfadbc7f68151f01ceb

176094e94dfe0e6591d82d3c5b604bfeea796166b1e8ccc6e9f0f346730ff419

1f94083aa9791cd1d26dbc9d67911e7e60954139c02aed7ad54578412ecba0ad

2701a4630d4daec81115952e01864901fcac223b912877191a752a83d93225d0

2957c4301881ed612d0181b977e6c07881c79b3ec7ea5cd5c682702414001173

29d76a5d80dc98e9689c9bc36c8691ce4b2a4867e6c4e4c397d80f9208b65cb7

2d77ce8ce7ad07b62fa129ddb2f4dc7f61b54c9e3a15e9235789caf68337b664

376a837cdb7574cc1e70027cf75134b0f9acc79f4dfcf5227601069571a413e7

3784a6cf0e04154c51d0c601742aded2446890c1645da14a8e5ff6bcce6968a8

392b4d6b66a9d7c152533a620e650b5ab81e63599e919f2c253beb6ebda6f620

3b62c34324901c37d3460c3bc31b4e4c548d440d55cf8566af994dff2739cd25

4099400333993976b8a6f8b159ddd25acb54244ea5cf42fe9093fc97848551a2

4d20b2c8ce3c955bbff22ecd71f04e6534deea4fbe0b7c43ec6d624b2ef68b16

509cebe2589b4a077ca1fcb66cccaa471e9e92ab897dc5237218f0f321f56596

51009930be311470d31ab8d43d398023596353ab3693be09c2ed049aef55bee7

51b90d291c7092f0b4403e96eded977d25b029f6b91e7dd0d66de883fa228260

56cbd53740eca77ed3b2587da5d7f1d0772769b3a34a9b3e733fb2f0861f5838

56d754f15e342c86df1777a8e9c67da694e02afc4b4dd52a1cc5710d66672dee

57a52249cf3c5b9a72c098e54a4874142e3c29889e5398321a9c0ca250c2ba0b

5d7f7d63a2c7a984b4c44820af91a1990e5cffa190bde43c578b7a8dc1280e3e

5fa72ec89b13aa0e9ff93569ba28990350ae7389492dd6dbc0d216d042384cf1

670575d06bae93f5d04b2017c7f1ac399fabd85fa24dc37e22e55647b21ac842

67e05be8bcee387718c112b989eac533ee9cd9f522790f91af112cf64a7e1c1e

68dd21b8e973ea8e24f5aa2eca233c13dcaefb49c82a8bc02790f310135c9145

6afe47f0a7936286cff89a3235d2af30a0ff7e4fe937e9c35ae1f2398246bf6f

6d626bdec14f14dc5cc07ea91289fb6fbcc8b35b923bc2dad243127de15970b8

6df4769f6b884d73fba79f9b6c99b928a5a333cd85ab50e75f9e9d789d3d6280

6e51bcd9c422adcde5f4abaee78a03d335b472a553ca592640391ea0d7d68166

7a3eb0c78f3fb7c58b8764486ac242946aba534d30b0ef0c363bc0a78b1abf77

7f0d27001ffdbeea8971b9d4de276ac7fb5ae3a25c9e48ac99f4e334c626a8af

80a566d6dc9c823a8dd349ecbacb584a2dc10086bfd91813135defab84c7d7f9

85690e596c675ade24d0a2009a23f74e83f6fa60db5a54507133ce6b61f6b08b

85947563ffc7fe8c78361b919c3da684b6369511b7dcd71498cf5c474dd540ea

8a7e78c104f9e1bfa72ff578d8802b03e71bcf8518f4efcc51d64b608faa68f9

91501c3739e9156b664dad1a6578e70c4b124ad1095d0e7d1f3dde9d9c5b032f

9253e072613063576c3e4e1a8220298817a831bf82a34280b1f29d236e2307fa

93af34be900e566aa2118eb8a049166f084b4cde34331b55951bf717ae70d4c5

a1adca16784843a2bafd1474cb7fea51ef2bab4b41359725f3cf7cdfa9c3410f

afa16fdbd78e747aceaeaee89b398c7719da08744b46192cbdb892b079703faf

b34d662d06f5832f2bf26cdc76540bcf56ae04a81776d01c03ae149a0e911c1c

b4f1a67decfa55f2461b7dac43a0d1598c6eac1db4b14590e85f42ee7effe918

b6190e77008a619daf47532d6dcf2d8ee475f2736c0e36a22d202659512f4ff9

b96a50cdd6a0a6a78e75daa5b8100c11f80a2df033b68bc7959cc4b5a6dbd0a9

ba1d0e26fdc66a7c58e2611e6fc788216bb32234791f4f1d7791b42e4cd1037e

c21c58bfb57724d8b2f1ea1c388b00b7c7aeb3ee60a463f8b872cf8f267c9de5

c38e4c9ab1cde65ef2a53517eff1b2b89ce3a8d94b09bacf0f5aa214ac18b8c9 c3cac02b3feea8267d664a923d86ea62e33915f98c4c1988293893d24a995da3

c4ae8b5a2aeb08ef4615988825f21b48859c997460f06f27f1c9f8ad6bebd180

cdaaa7a0edd4ff8670ffa3bcb466e58c079a1e1e69b1f783880c7541e4e17b1f

d0eff9565324afdd83c9d275c5f7e60eddb08328020009d406ff3fbd4157ec0f

d202d553c1747cef032dd40d848b8ff6836ca3aee31809990251842b1dbffee4

dbf4bf167012f39aa841abaabf52e5a2a9e0806a1d952139d964bfc16c7bf1a0

dd906adc6e1ff81d732eb37c7ef86d9e826df3288a97184e8b7a9c58a03dce36

df1c98789f1c1f21d73448e309f33150c3dbe575455cc6a84d1debd942bb5ca4

e1ac82e7fa04d2a3ddf0d29da34d85d72e6c3fad808d6aed429f5de4e0b7eee4

e22698d2eac2887c9360c45c1576b89b5a3d31a0451cdb7e4d84f1b620bcfde9

e4dc45f955a1303887921235ef5aa30e74e67ed31750be9cfcde837485e43354

e89790699dac00300d158167d0238db176a9618cfcaf8ea18e049cbc2f977efe

eb0fbaa93f5825a28140dbc0580850cf16b6e31d419d5b89b245cf5a3236428e

f08f5aa313600eb048197ac2c3ff87533922e3d0fa130be53142571cf53a0cc9

f1cad26dca04e0d2b5cdc268d9cd3b4bce6c5adbe34bcb8c167f8fbe56ea06ee

f573ce6a8fbe31b56aa270ed7dd34d307a7e47153765b359cb4f472d14d0bb35

f8953dfa979ea179c503f0f8569fc47356996cc2d114aa748a2a7432925d5227

fa7bac8c53a26777df00bcb49da5dc76e063c93d941c2697e8f3333a4ed239d0

ff680aa77698ed3526ac7f0a488944ab6eabb0e0e38c46f024ae6ca4d1796250

Hajime Hashes

Hajime hashes can be found on GitHub.[15]

Hide ‘n’ Seek Hashes

07381cbad47445a63f340a9e4c931132043c97c2a2e81ecee07c894cc1d11430

0b05202f4da9bbe1af1811707a76544453282c4f3c0ac9b353759c86742f4369 MIPS big endian

12ec8c33abae7f0b06533d96f0d820df4ac673e6ff3965a59d982623f4ddc6bb

1583fd1c6607b77f51411c4ad7c9225324fd1b069645062a348cd885de0ac382 ARM

1a45310fc69d1b9efef00a9f4fd6321a75abb444c6519a0bc7b7eb858590921d

24b89e36e12166f613edb61909d1192dbd918c2eac45d3a75a588ec24a4e2a36 ARM

30ebaaeb61a4ecae3ade7d1d4e5c7adb miner

41455b90885cbb027b148cec7eba307ae1c16b1b76b345434893b3a581090125

49495c9aa08d7859fec1f99f487560b59d8a8914811746181e4e7edbee85341f x86 64-bit

500dd4c1a5c24495c3bb8173ce5c7b15ba3344aef855090b9b9585b2bfeea974 x86

6399f9463c74ee687c047e3a7e0149421de423de77ee278ce364b68d22eada00

68a8f57b21601ff66980239d6867f72747f5ffc0fe6fabfd07dfe72f90941f2f

73df4e952c581afc427fa18fa2d0bcfa409c1814cd872a3ccf05d44f934ce780 MIPS little endian

7e20c6cea88ade6a6c4a08ce48fe4ac2451069b7662a8dda4362a304b4854ec7 ARM

8cb5cb204eab172befcdd5c923c128dd1016c21aaab72e7b31c0359a48d1357e

91606298B13F6C2DABE20EC6D57C71A89C7726F6A9799E8788E904DBEB6DCFB5

97bc283337cabe055e0c79970cf0e62bca5b4d6de2a55c98fe1f69c76990528d

a41c71915ca7c0788c0b6c20e383454432b85f9aeb4f1f3542dbbba6929e5488

B0FA3682157A3153122D79EBA9118AE4F12BF2ED4E2CA6BA6DC93F9643CC4D3A

becad1b9d3b67e51404475a0a530b57fd5742f3148f23693c349035dbffddd7

c082c39e595c7f23c04ce0d6597657d6e649585d5da49b5bd896e664b712e60d MIPS big endian

d068e8f781879774f0bcc1f2a116211d41194b67024fe45966c8272a8038a7a1 ARM

d69ff15cff8bd25698d8bb33de044c34353d74ef801338ee3e67e7d7524f8078 x86

d87ff51186c4a75ba0fb802abd55a15d263187aefd489a37f505b025ddc1ee66

Ee6fe7f885550af73a6df745d983e16c327ca3253067036744870db17a0c3437

Mozi Hashes

018eaeb1a8ef1bd566a2b87c462316e1071696d0b58f99f60e90f4164dce52b0

01d8e2bcf22422e9c995d43c403c63477389fc9f4a141ef3bbd31c8f5c6ef7e6

06fcc8e49dd2570a56318d255404540ed380e284ad00866e0ce0f3052be4bd58

084ab317f916d03022ea12b7009540a0b799b987c7c41003d97d4414f3b82bd9

0a9a5cc107e7225529bc04a8ec1c880af34155c2ff778d90efa7118bc2e5414d

0ebf64c87066480180b7490ad6770a9b80e7c0e2cb6ba823fc885c4f34e5f166

16aac527ebc47f734b5aec6408c464246c4d0ca7429e842e39f6f57400865877

190c13563be0e74e5ffccea34bd63103177ecc4c8a764e1d6c6898a54e9ae70a

1e7ca3e32d11f96a8b112175973a0869f16449077365f7a51bb09b4d3375861a

1f7f81494c4a9508e247ea78db9ec184aa0a74be7712d9a643cea3b087c566ee

2c281d35251dcf6e3b0f7536894a1368a1270a26e17bf189a1e6eb8e4b2def5f

3768eb4b9101258f86d5f1cea1138c6df1f2ef6b13c82ed0186f2554bf0b04a1

3a31793b73636f0a26518dee30e5b354e8fa03276c636c06c4a64cdcb1ace534

5ecfe9b180ad21bb522a0028dc03e6c3c235ef7d2c401e899a246206544ad490

73bfb21fe61b184a6914b83b0c742164618db2a4bab5fe504ca311b6d9b6834a

756fe8cf9a6a34c0f047d067cf7ace367fd1667a9f64cadf06eb88a4d5ec8d0e

7ac12520c1f294001aa4fd43b5e103e883738089877c94dbd5f62bb955173a8f

82267baa5ec4fca4f39ec61d85aae8f90e92ccba821b9ce92d74804127e1bf71

832fb4090879c1bebe75bea939a9c5724dbf87898febd425f94f7e03ee687d3b

83441d77abb6cf328e77e372dc17c607fb9c4a261722ae80d83708ae3865053d

8accd39e03fce77ebe73107ade5bac4adc63c9ebe0c600181d8f6f7dcc5215fd

aee85cf36c53dc0345d75cffd2ce8bdaac23907af102cdf4f9820bd7db2349ec

b9148379ed5d8a4b8ad58ec9f2e755ddef9d90a16522c7df00702ae73272a6f8

bba18438991935a5fb91c8f315d08792c2326b2ce19f2be117f7dab984c47bdf

c67195ff6814d2a2c8fdf235841fb9442e66a10d6096336ef3d51c2decb48ff7

c6f6ca23761292552e6ea5f12496dc9c73374be0c5f9d0b2142ca3ae0bb8fe14

d546509ab6670f9ff31783ed72875dfc0f37fa2b666bd5870eecaaed2ebea4a8

e15e93db3ce3a8a22adb4b18e0e37b93f39c495e4a97008f9b1a9a42e1fac2b0

e3ee24ce5e90ceeeb100163ae760ffa77844bbf8c37de87fed1840c5fe2404ab

e90046ff173b57c5d2f16a1fbc45fe132bd843ec5f333d6c2a82a098b8528740

f37c7a78166735816e66fa00886b9d81592731601823fbb76f2285cde62ecc03

fabece475f5a63d9c58ce5f7fb1f8d4e9c7171ac5d603b7b1ec31b0932008cd3

fdf2889d0da4e4bb6b4f6ba6358e194f21650385338e3402302990646c0478bc

HEH Hashes

4c345fdea97a71ac235f2fa9ddb19f05

6be1590ac9e87dd7fe19257213a2db32

bd07315639da232e6bb4f796231def8a

984fd7ffb7d9f20246e580e15fd93ec7

66786509c16e3285c5e9632ab9019bc7

c1b2a59f1f1592d9713aa9840c34cade

6fa68865f1a2ddd1cf22f1eba583517c05b6f6c3

43de9c5fbab4cd59b3eab07a81ea8715

6c815da9af17bfa552beb8e25749f313

D302749a080dd73e25673560857495ba14fa382857f64d26138acb044e2d9242

4f9b895a2785f9788fcae8743ab04a24b62e0962b1f8a28dc1206c52327b7916

References

[1] Stephen Hilt et al. (July 15, 2020). *Trend Micro*. "Worm War: The Botnet Battle for IoT Territory." Accessed on March 3, 2021, at https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/caught-in-the-crossfire-defending-devices-from-battling-botnets.

[2] Stephen Hilt and Fernando Merces. (Jan. 19, 2021). *Trend Micro*. "VPNFilter Two Years Later: Routers Still Compromised." Accessed on March 3, 2021, at https://www.trendmicro.com/en_us/research/21/a/vpnfilter-two-years-later-routers-still-compromised-.html.

[3] Eduard Kovacs. (Oct. 7, 2015). *SecurityWeek*. "Developers of Mysterious Wifatch Malware Come Forward." Accessed on March 3, 2021, at https://www.securityweek.com/developers-mysterious-wifatch-malware-come-forward.

[4] The White Team. (Oct. 5, 2015). *GitLab*. "linux.wifatch." Accessed on March 3, 2021, at https://gitlab.com/rav7teif/linux.wifatch.

[5] The White Team. (Oct. 5, 2015). *GitLab*. "linux.wifatch." Accessed on March 3, 2021, at https://gitlab.com/rav7teif/linux.wifatch.

[6] Stephen Herwig et al. (Feb. 24, 2019). *NDSS Symposium*. "Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet." Accessed on March 3, 2021, at https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_02B-3_Herwig_paper.pdf.

[7] Juliusz Chroboczek. (Dec. 28, 2020). *GitHub*. "dht." Accessed on March 3, 2021, at https://github.com/jech/dht/blob/master/dht.c.

[8] MalwareTech. (Jan. 9, 2019). *MalwareTech*. "Tracking the Hide and Seek Botnet." Accessed on March 3, 2021, at https://www.malwaretech.com/2019/01/tracking-the-hide-and-seek-botnet.html.

[9] Adolf Středa and Jan Neduchal. (Dec. 4, 2018). *Avast*. "Let's play Hide 'N Seek with a botnet.." Accessed on March 3, 2021, at https://blog.avast.com/hide-n-seek-botnet-continues.

[10] Adolf Středa and Jan Neduchal. (Dec. 4, 2018). *Avast*. "Let's play Hide 'N Seek with a botnet.." Accessed on March 3, 2021, at https://blog.avast.com/hide-n-seek-botnet-continues.

[11] Black Lotus Labs. (April 13, 2020). *Lumen*. "New Mozi Malware Family Quietly Amasses IoT Bots." Accessed on March 3, 2021, at https://blog.lumen.com/new-mozi-malware-family-quietly-amasses-iot-bots/.

[12] Alex Turing and Hui Wang. (Dec. 23, 2019). *Netlab 360*. "Mozi, Another Botnet Using DHT." Accessed on March 3, 2021, at https://blog.netlab.360.com/mozi-another-botnet-using-dht/.

[13] JiaYu. (Oct. 6, 2020). *Netlab 360*. "HEH, a new IoT P2P Botnet going after weak telnet services." Accessed on March 3, 2021, at https://blog.netlab.360.com/heh-an-iot-p2p-botnet/.

[14] Julian Grizzard et al. (Jan. 2007). *ResearchGate*. "Peer-to-Peer Botnets: Overview and case study." Accessed on March 3, 2021, at https://www.researchgate.net/publication/234760214_Peer-to-Peer_Botnets_Overview_and_case_study.

[15] Ioannis Profetis. (March 28, 2018). *GitHub*. "hajime_hashes." Accessed on March 3, 2021, at https://github.com/Psychotropos/hajime_hashes.