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 Carma, 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.

```plaintext
# connect types
sub H_JOIN () {72}   # initial join(?), want seed
sub H_NEIGHBOR_LO () {73}  # connect low priority (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. 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.
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. 

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, 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.

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. 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. 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. 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.
Run Command
Instruct bot to do something (see command list below).

Ping
Check peer availability.

Pong
Respond to Ping.

Announce
Announce oneself as a peer. It adds this peer to self’s peer list.

PeerUpdate
Let peer update its peer list.

Table 1. The five opcodes used by HEH

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Run Command</td>
<td>Instruct bot to do something (see command list below).</td>
</tr>
<tr>
<td>2</td>
<td>Ping</td>
<td>Check peer availability.</td>
</tr>
<tr>
<td>3</td>
<td>Pong</td>
<td>Respond to Ping.</td>
</tr>
<tr>
<td>4</td>
<td>Announce</td>
<td>Announce oneself as a peer. It adds this peer to self’s peer list.</td>
</tr>
<tr>
<td>5</td>
<td>PeerUpdate</td>
<td>Let peer update its peer list.</td>
</tr>
</tbody>
</table>

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.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Restart</td>
<td>Restart the bot.</td>
</tr>
<tr>
<td>2</td>
<td>Exit</td>
<td>Exit bot.</td>
</tr>
<tr>
<td>3</td>
<td>Attack</td>
<td>Launch an attack (not yet implemented).</td>
</tr>
<tr>
<td>4</td>
<td>Execute</td>
<td>Execute shell command.</td>
</tr>
<tr>
<td>5</td>
<td>Print</td>
<td>(Not yet implemented.)</td>
</tr>
<tr>
<td>6</td>
<td>PeerUpdate</td>
<td>Update peer list.</td>
</tr>
<tr>
<td>7</td>
<td>UpdateBotFile</td>
<td>Bot downloads link and updates this “bot” file.</td>
</tr>
<tr>
<td>8</td>
<td>SelfDestruct</td>
<td>Wipe out the device.</td>
</tr>
<tr>
<td>9</td>
<td>Misc</td>
<td>(Not yet implemented.)</td>
</tr>
</tbody>
</table>

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

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

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
0afbe71c6a8c6151a20f7f873b500cfca77dac7b4f76dfadbc7f68151f01ceb
176094e94dfe0e6591d82d3c5b604bfee7976166b1e8cc6e9f0f346730ff419
1f94083aa9791cd1d2d6bc9d67911e7e60954139c02aed7ad54578412ecba0ad
2701a4630d4dace81115952e01864901fcac223b912877191a752a83d93225d0
2957c4301881d0181b977e6c07881c79b3ec7ea5cd5c682702414001173
29d76a5d80dc98e9689c9bc36c8691ce4b2a4867e6c4e4c397d80f9208b65cb7
2d77ce8ce7ad07b62fa129d2f4dc7f61b54c9e3a15e9235789caf68337b664
376a837cd8774cc1e70027c7f5134b0f9acc79f4dfcf5227601069571a413e7
3784a6cf0e04154c51d0c601742aded2446890c1645da14a8e5ff6bce6968a8
392b4d6b66a9d7c152533a620e650b5ab81e63599e919f2c253beb6ebda6f620
3b62c34324901c37d3460c3bc31b4e4c548d440d55cf8566af994dff2739cd25
409940033993976b8a6f8b159dd25acb54244ea5cf42fe9093fc97848551a2
4d20b2c8ce3c955b4ff22eced7f04e6534d9ee4f4be0b7c33ec6d624b2ef68b16
509cbe2589b4a077ca1fcb66ccca471e9e92ab897dc5237218f0f321f56596
51009930be311470d31ab8d43d39023596353ab3693be09c2ed049aef55bee7
51b90d291c7092f0b4403e96eced977d25b029f6b91e7dd0d66de883fa228260
56cb53740eca77ed3b2587da5d7f1d0772769b3a34a9b3e733fb2f0861f5838
56d754f15e342c86df1777a8e9c67da694e02afc4b4dd52a1cc5710d66672dee
57a52249cf3c5b9a72c098e54a4874142e3c29889e5398321a9c0ca250c2ba0b
5d7f7d63a2c7a984b4c44820a91a1990e5cfa190bde43c578b7a8dc1280e3e
5fa72ec89b13aa0e9ff93569ba28990350ae7389492dd6d9c0d216d042384cf1
Hajime hashes can be found on GitHub.\textsuperscript{15}

**Hide ‘n’ Seek Hashes**

07381cbad47445a63f340a9e4c931132043c97c2a2e81eceee07c894cc1d11430

0b05202f4da9bbee1af1811707a765444532824f3c0ac9b353759c86742f4369 MIPS big endian

12ec8c33abae7f0b06533d96f0d820df4ac673e6ff3965a59d982623f4ddc6bb
ARM
1583fd1c6607b77f51411c4ad7c9225324fd1b069645062a348cd885de0ac382 ARM
1a45310fc69d1b9eef00a9f4fd6321a75abb444c6519a0bc7b7eb858590921d
24b89e36e12166f613ed61909d1192dbd918c2eac45d3a75a588ec24a4e2a36 ARM
30eabaeb61a4ecae3ade7d1d4e5c7ad miner
41455b90885ccb027b1b48ce7eba307ae1c16b1b76b345434893b3a581090125
49495c9aa08d7859fec1f99f487560b59d8a8914811746181e4e7edbee85341f x86 64-bit
500dd4c1a5c24495c3bb8173ce5c7b15ba3344ae855090b9b9585b2bfeea974 x86
6399f9463c74ee687c047e3a7e0149421de423de77ee278ce364b68d22eada00
68a8f57b21601ff66980239d6867f72747f5f0e6fabdf07dfe72f9094112f
73df4e952c581af427fa18fa2d0bca409c1814cd872a3ccf05d44f934ce780 MIPS little endian
7e20c6cea8ade6a6c4a08ce48fe4ac2451069b7662a8dda4362a304b4854ec7 ARM
8cb5cb204eab172bhfed5c923c128dd1016c21aaab72e7b31c0359a48d1357e
91606298B13F6C2DAB20EC6D57C71A89C7726F6A9799E8788E904DBEB6DCFB5
97bc283337cabe055e0c79970cf0e62bca5b4d6de2a55c98fe1f69c76990528d
a41c71915ca7c0788c06c20e383454432b85f9aeb4f1f3542dbbba6929e5488
B0FA3682157A315312D79EBA9118AE4F12BF2ED4E2CA6BA6DC93F9643CC4D3A
becad1b9d367e51404475a0a530b57fd5742f3148f23693c349035dbffddd7
c082c39e595c7f23c04ce0d5697567d6e649585d5da49b5bd896e664b712e60d MIPS big endian
d068e8f781879774f0bcccc2a116211d41194b67024fe45966c8272a8038a7a1 ARM
d69ff15c69b2d5698d8bb33de044c34353d74ef801338ee3e67e7d7524f8017e x86
d87ff5118e4a75ba0f802abd55a15d263187aef489a37f505b025dcd1ee66
Ee6fe7f885550af73a6df745d983e16c327ca3253067036744870db17a0c3437

Mozi Hashes
018eaeb1a8ef1bd566a2b87c462316e1071696d0b58f9960e90f4164dce52b0
01d8e2bcf22422e9c995d43c403c63477389f94a141ef3bbd31c85c6ef7e6
HEH Hashes

4c345fdea97a71ac235f2fa9dddb19f05
6be1590ac9e87dd7fe19257213a2db32
bd07315639da232e6bb4f796231def8a
984fd7fbb7d9f20246e580e15fd93ec7
66786509c16e3285c5e9632ab9019bc7
c1b2a59f1f1592d9713aa9840c34cade
6fa68865f1a2dd1cf22f1eba583517c05b6f6c3
43de9c5fbab4cd59b3eab07a81ea8715
6c815da9af17bfa552beb8e25749f313
D302749a080dd73e25673560857495ba14fa382857f64d26138acb044e2d9242
4f9b895a2785f9788fcae8743ab04a24b62e0962b1f8a28dc1206c52327b7916
References


