Playing Hide 'N Seek:
World's first IoT Botnet with custom-built p2p communication

Bitdefender

Introduction

Bitdefender researchers have uncovered an emerging botnet that uses advanced communication techniques to exploit victims and build its infrastructure. The bot, dubbed HNS, was intercepted by our IoT honeypot system following a credentials dictionary attack on the Telnet service.

The bot was first spotted on Jan. 10 then faded away in the following days, only to re-emerge on Jan. 20 in a significantly improved form.

Impact

The HNS botnet communicates in a complex and decentralized manner and uses multiple anti-tampering techniques to prevent a third party from hijacking/poisoning it. The bot can perform web exploitation against a series of devices via the same exploit as Reaper (CVE-2016-10401 and other vulnerabilities against networking equipment).

The bot embeds a plurality of commands such as data exfiltration, code execution and interference with a device's operation.

Operation

The bot features a worm-like spreading mechanism that randomly generates a list of IP addresses to get potential targets. It then initiates a raw socket SYN connection to each host in the list and continues communication with those that answer the request on specific destination ports (23 2323, 80, 8080). Once the connection has been established, the bot looks for a specific banner ("buildroot login:") presented by the victim. If it gets this login banner, it attempts to log in with a set of predefined credentials. If that fails, the botnet attempts a dictionary attack using a hardcoded list.

Once a session is established with a new victim, the sample will run through a "state machine" to properly identify the target device and select the most suitable compromise method. For example, if the victim has the same LAN as the bot, the bot sets up TFTP server to allow the victim to download the sample from the bot. If the victim is located on the internet, the bot will attempt a specific remote payload delivery method to get the victim to download and run the malware sample. These exploitation techniques are preconfigured and are located in a memory location that is digitally signed to prevent tampering. This list can be updated remotely and propagated among infected hosts.

The samples identified in our honeypots on Jan. 10 revolved around IP cameras manufactured by a Korean company. These devices seemed to play a major role in the botnet as, out of the 12 IP addresses hardcoded in the sample, 10 used to belong to Focus H&S devices. The new version, observed on Jan. 20, dropped the hardcoded IPs.

Like other IoT bots, the newly discovered HNS bot cannot achieve persistence, and a reboot would bring the compromised device back to its clean state. It is the second known IoT botnet to date, after the notorious Hajime botnet, that has a decentralized, peer-to-peer architecture. However, if in the case of Hajime, the p2p functionality was based on the BitTorrent protocol, here we have a custom-built p2p communication mechanism.

UDP communication mechanism

The bot opens a random port on the victim, and adds firewall rules to allow inbound traffic for the port. It then listens for connections on the open port and only accepts the specific commands described below. Our initial look at the sample revealed an elliptic curve key inside the file that is used to authenticate the command which updates the memory zone where configuration settings are stored, to prevent infiltration or poisoning attempts against the botnet.

The botnet binary can have the following arguments **at runtime**:

'k' + [some_port_number]	Kills all processes running on the specified port at startup.
'l' + [some_port_number]	Listens on the specified port number after startup
's' + path	Reads files starting from the path and loads them into memory. Then, the content can be retrieved by other peers via the 'm', 'y' and 'Y' commands. There's also a sha256-checksums for each file that gets created. As a side effect, this will keep the malicious binary on the system.
'a' + [IP:PORT]	Adds IP:PORT to the list of internally stored IP:PORT
'e' + IP:PORT	Adds a new target to the otherwise randomly generated target list.

At runtime, the bot listens to commands either on the port specified at startup via argument or on a randomly generated port, if not specified. As it receives a command, the bot usually replies to the sender with a packet that follows the pre-established communication protocol. Given that the parsing / processing of data is exactly the same in both receiver and emitter mode, we can conclude that the botnet communication is based on a peer-to-peer architecture.

In addition to the bot capabilities, the samples we analyzed also feature a web-server component that hosts and serves binary files to other potential victims.

Supported commands

To understand the capabilities of the bot, we looked into the messages received from other peers. Below is a list of the supported commands, along with a description of the expected results:

'i' + u32(config_size)

If the config_size received is larger than the size of the current configuration, an acknowledgement message is sent back to the peer. The peer is eventually set as a communication endpoint to get the larger config. If the dht_size is smaller, an 'I' message is sent back to the peer advertising the larger config.

'I' + u32(config_size)

The config_size received should be larger than the size of the current config (following an 'i' message query). An acknowledgement message is sent back and eventually the peer is set as a communication endpoint to get the larger config.

'm' + u8[32](hash) + u32(ip) + u16(port) + u16(seq) + u8(hops) + u8(unk)

This message attempts to find data based on the hash given. It first checks locally if the hash is known and data is available. If data is available, it starts sending it to ip:port via a 'Y' message. If data is not locally available, the bot broadcasts the current 'm' message (with hops decremented, but not more than 5 hops) to all its known peers.

'^' + u8(flags) + u16(port) + u32(ip)

Signals a new peer at ip:port. The new peer gets added to the list of known peers (it can also replace a peer if too many are present).

'~'

Requests a new peer. Upon receiving this, a random peer is selected from the list of known peers and sent via a '^' message

'Y' + u16(chunk_index) + u16(seq) + u8[](data)

This is the data received message. The data is copied in chunks of at most 0x100 bytes (i.e. the current offset is chunk_index * 0x100). The chunk_index is the current offset being read from and it should correspond to the current size of a communicated buffer.

There are multiple possible outcomes once the file is downloaded:

- If the data is a new config, it gets verified (via ECDSA - the elliptic curve key mentioned above) and, if OK, it will replace the current config
- The file can get dropped and executed (i.e. an update mechanism)

'y' + u8[32](hash) + u16(chunk_index) + u16(seq)

This command attempts to read from chunk_index the data corresponding to the cache via a 'Y' message. If, for whatever reason, the data is not available locally, an 'm' message gets broadcast.

'O' + u8(checksum)

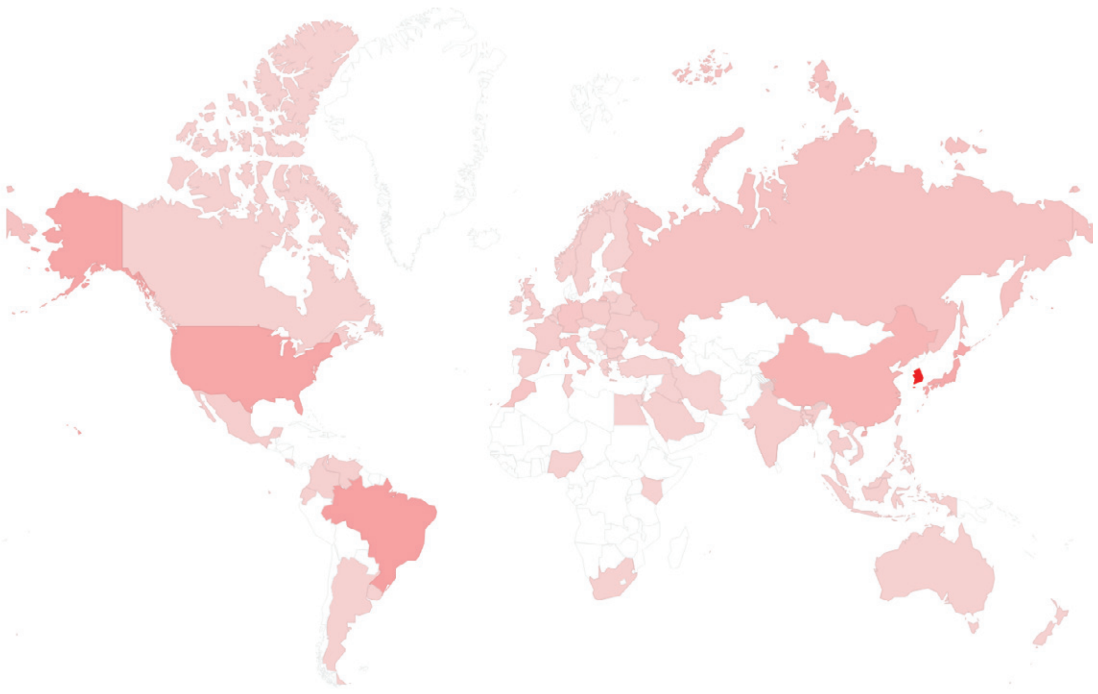
This is an acknowledgement message. The checksum is done on the received message.

'i' , 'I' (first version) 'h', 'H' (second version)	Commands for configuration updates
'm','Y','y'	Data exfiltration mechanism ("m" communicates a hash, while "Y" and "y" move data around)
"z"	scanning component (sends to a peer valid credentials that have been found via dictionary attack)
"O"	ack on "z" credentials received OK (however, the same ack can be sent in several other occasions)
"^"	add a new peer to the list of known peers
"~"	send a peer IP as a response (when queried for a peer IP)

Conclusions

While IoT botnets have been around for years, mainly used for DDoS attacks, the discoveries made during the investigation of the Hide and Seek bot reveal greater levels of complexity and novel capabilities such as information theft - potentially suitable for espionage or extortion.

It is also worth noting that the botnet is undergoing constant redesign and rapid expansion.



Known hashes

efcd7a5fe59ca8223cd282bfe501a2f92b18312c
05674f779ebf9dc6b0176d40ff198e94f0b21ff9

References

1. https://nvd.nist.gov/vuln/detail/CVE-2016-10401
2. https://www.exploit-db.com/exploits/25978/
3. http://www.h-online.com/security/news/item/Treacherous-backdoor-found-in-TP-Link-routers-1822720.html