Looking Into the Eye of the Interplanetary Storm
Executive Summary

Bitdefender researchers have found clues that the Interplanetary Storm Golang botnet could be used as highly anonymous proxy-network-as-a-service and potentially rented using a subscription-based model.

While the botnet has been under previous scrutiny from Bitdefender researchers, constant monitoring of the development lifecycle of Interplanetary Storm has revealed that threat actors are both proficient in using Golang and development best practices, and well-versed at concealment of management nodes. While previous research from security researchers has focused on analyzing some of the capabilities of the malware and its network traffic, Bitdefender researchers have provided the full picture as well as focused on finding leads regarding the malware developers’ identity and the potential purpose of the infrastructure.

Interplanetary Storm also has a complex and modular infrastructure designed to seek out and compromise new targets, push and synchronize new versions of the malware, run arbitrary commands on the infected machine and communicate with a C2 server that exposes a web API.

This article offers a glimpse into the inner workings of the Interplanetary Storm botnet, provides an exhaustive technical analysis of the Golang-written binaries along with an overview of the protocol internals and finally, some attribution information.

A series of six specialized nodes that are part of the management infrastructure are responsible for checking for node availability, connecting to proxy nodes, hosting the web API service, signing authorized messages, and even testing the malware in its development phase.

Along with other development choices, this leads us to believe that the botnet is used as a proxy network, potentially offered as an anonymization service.

Mapping the Interplanetary Storm botnet, we estimate the size of the botnet at around 9,000 devices. The overwhelming majority have Android as their operating system and about 1% Linux. However, a very small number of devices have Windows as their OS, but they seem to be running older versions of the malware. In its new iteration, IPStorm propagates by attacking Unix-based systems (Linux, Android and Darwin) that run Internet-facing SSH servers with weak credentials or unsecured ADB servers. We have also seen Darwin only in a few entries that seem to represent the same machine, the one used for development of IPStorm.

In terms of geographic distribution, most victims of this particular botnet seem to be based in Asia, but the botnet has a global footprint, with victims in Brazil, Ukraine, the US, Sweden and Canada just to name a few countries.

Note: The attribution section of this research should be considered as a stepping stone by law enforcement agencies looking to further investigate the malware author and the infrastructure. For further information, Bitdefender researchers are fully committed to helping officials with any other attribution information that may be necessary to build a case and continue the investigation from a law enforcement perspective. If you are a law enforcer from the affected countries and wish to obtain the attribution information we have on this investigation, please reach out to draco@bitdefender.com.
Key findings

• Botnet potentially rented as an anonymous proxy network
• Built to use compromised devices as proxies
• Botnet mapping reveals global presence
• Rented using multi-tier subscription-based pricing model
• More than 100 code revisions to date
• Detailed analysis of the infrastructure behind the Interplanetary Storm botnet

Introduction

This article offers a glimpse into the inner workings of the Interplanetary Storm botnet, provides an exhaustive technical analysis of the Golang-written binaries along with an overview of the protocol internals and finally, some attribution information.

Interplanetary Storm (IPStorm) was first reported by researchers from Anomali in June 2019. In May 2020, we discovered a new campaign of this botnet when it attacked our SSH honeypots. The malware has been in continuous development since then, integrating new features and seeking to blend in with innocuous traffic.

In its new iteration, IPStorm propagates by attacking Unix-based systems (Linux, Android and Darwin) that run Internet-facing SSH servers with weak credentials or unsecured ADB servers.

Its capabilities include backdooring the device (running shell commands) and generating malicious traffic (scanning the Internet and infecting other devices). We have determined that the main purpose of the botnet is turning infected devices into proxies as part of a for-profit scheme.

Botnets with this goal have turned up in the past (for instance: dark_nexus, Ngioweb, Gwmndy) and the assumption was that cybercriminals were selling illegitimate access to devices on underground forums or the Dark Web. This time, however, we have found evidence that the bot herders are posing as a legitimate proxy service on the Clearnet.

Overview

The last two years have seen an increase in malware written in Golang, and Linux botnets make no exception. Emptiness, Liquorbot, Kaiji and Fritzfrog are all examples of Golang bots that target Linux machines using SSH as an attack vector. Multiple features of this language make it desirable for malware authors: portability and the rich codebase being the foremost.

Some of these malware families follow the model of "traditional" Linux botnets, rewriting them in Go, while others have original design. IPStorm belongs in the latter category, as its core functionality is written from scratch. It integrates open-source implementations of various protocols, such as NTP, UPNP and SOCKS5, and bases its peer-to-peer protocol on libp2p. The libp2p library contains a networking stack through which users can interact with the Interplanetary Filesystem (IPFS).

Compared to other Golang malware we have analyzed in the past, IPStorm is remarkable in its complex design due to the interplay of its modules and the way it makes use of libp2p's constructs. It is clear that the threat actor behind
the botnet is proficient in Golang; one consequence of the malware author’s good coding practices, namely their thoroughness in error handling, is that it makes the reverse engineering process easier, as many code sequences are accompanied by relevant logging strings.

The entire functionality of the bot is bundled into a statically linked binary that is packed with UPX. The large size of the binaries - about 7.7M packed and 18M unpacked - is due to the inclusion of the Golang runtime. Although the binaries have been stripped, the debugging information persists in the .gopclntab section of the binaries. This allows us to recover the name of the functions and packages and structure of types.

Each IPStorm version is cross-compiled for multiple CPU architectures and platforms, encompassing the following:

storm_android-386
storm_android-amd64
storm_android-arm7
storm_android-arm64
storm_linux-386
storm_linux-amd64
storm_linux-arm7
storm_linux-arm64
storm_darwin-amd64

Timeline

The evolution of IPStorm can be tracked precisely, as all binaries are versioned using Semantic Versioning. It can be split into three phases:

• Major 0, Minor 0: reported by Anomali last year; targeted Windows exclusively
• Major 0, Minor 1: emerged this year (in May 2020); targeted Unix-derived systems
• Major 0, Minor 2: latest evolution (September 2020); transitioned away from the publish-subscribe model

As of the writing of this article, the most recent version is 0.2.05a.

Bot Lifecycle

The startup code of the bot initializes an IPFS node and launches the goroutines (lightweight threads) dedicated to each of the bot’s sub-modules. It sets the oom_adj score for the malware process to -17, ensuring that it will not be killed if the system runs out of available memory. Then, it ensures that only a single instance of the malware runs on the device by periodically scanning the list of processes for the name storm. Any matching process is killed and its executable removed.

A 2048-bit RSA key pair is generated and stored in a writable path on the filesystem. This key belongs to the IPFS node and uniquely identifies it. The node is instantiated and the bootstrap process is started, making it reachable by other nodes in the IPFS network. The connection to other peers in the botnet is ensured by periodically “announcing” itself and looking for peers that broadcast the same announcement (more about this in the “P2p communication” section).
An info ticker collects information about the system and publishes this fingerprint on an IPFS pubsub topic. An example of such an entry we retrieved from the info topic is the following (with some information redacted for privacy):

```
{
  "T" : 1592892637,
  "HostID" : "Qmf4[___________redacted___________]",
  "Version" : "0.1.81a",
  "Platform" : "linux-arm7",
  "SystemInfo" : {
    "GoOS" : "linux",
    "Kernel" : "Linux",
    "Core" : "4.19.97-v7+",
    "Platform" : "unknown",
    "OS" : "GNU/Linux",
    "Hostname" : "raspbx",
    "CPUs" : 4
  },
  "Uid" : "0",
  "Gid" : "0",
  "UserName" : "root",
  "UserDisplayName" : "root",
  "UserHomeDir" : "/root",
  "IsAdmin" : true,
  "ExecutablePath" : "/usr/bin/storm",
  "InstallationPath" : "/usr/bin/storm",
  "ComputerID" : "",
  "LocalIPs" : null,
  "ExternalIP" : "[redacted]",
  "Processes" : null
}
```

As information about other peers is not used by bots themselves, we believe that it is collected only for the interest of the bot herders. Fortunately for our research, prior to version 0.1.92a it was also public for anyone knowing the topic ID.

Another periodic goroutine is tasked with performing an update if a new version of the bot is available. In this case, the updated file is written to the filesystem, the persistence of the malware is re-established and the process is re-launched.

The persistence is handled depending on OS.

- On Linux it uses the open-source daemon package to create a service named storm.
- On Android it remounts the filesystem as read-write and overwrites the `/system/bin/install-recovery.sh` file.
- On Darwin, no persistence method is implemented.
P2P Communication

When it comes to communication between peers, IPStorm makes use of multiple mechanisms provided by libp2p over IPFS:

- topics
- content routing (node discovery)
- libp2p protocols

Different approaches are used for messages intended for all nodes (version updates, file checksums, IDs of nodes with special roles) and for messages intended for certain nodes (scanning targets, proxy requests, shell commands).

In the first approach, messages are published on a topic and all nodes subscribe to that topic and process the messages. In the case of the DDB (distributed database), messages published on the topic serve to sync the DB among all nodes. Although the messages may come out of order due to the way they are propagated through the network, the inclusion of a timestamp enables each node to keep only the most recent value for a given key. To ensure that peers can properly coordinate using timestamps, the bot updates its time by querying a random entry from a list of public NTP servers.

The second method applies for example to the scanning module: a central entity issues scanning commands, distributing the targets to bots. This is achieved by connecting to each bot using a protocol particular to IPStorm.

Topics

Topics are part of libp2p’s implementation of the Publish-Subscribe pattern. The following topics are used by IPStorm:

<table>
<thead>
<tr>
<th>Description</th>
<th>Purpose</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>info topic</td>
<td>information about the infected system</td>
<td>C0FAh40EtzpBb145aYUcluKo1UJZ-1bRUJGg1WJo3OFzWFEDKzbShsWFO-wt95LQg1ep0gp22hX3PQ2iU5w83a8uoPw-C3WngMgFD1</td>
</tr>
<tr>
<td>cmd topic</td>
<td>commands (version 0.0.*)</td>
<td>6szrvClvhS8QSTs6nqOl28i77MNO1Dvh4pCVBvRYxm-5dGHPsOa8sNCMyxpcP0MFTe5BeXWPnNILo3I6G6Ev1gQLeqz54RoMwW9pPFD1</td>
</tr>
<tr>
<td>VPNGate scraper topic</td>
<td>information about VPNGate servers</td>
<td>HyD9c7ZrNrXZqG-M7IVVd0HH5DjmHFz-PBHgk10dqEpVR-Nn7MXo-bDjX7nMdaNqq-SNsyd6WCYgnu-19uT0rsK3V7bPN-RSrNkITviS</td>
</tr>
<tr>
<td>DDB topic</td>
<td>synchronization messages for the DDB</td>
<td>M7IVVd0HH5DjmHFz-PBHgk10dqEpVR-HyD9c7ZrNrXZqG-Nn7MXo-bDjX7nMdaNqq19uT0rsK3V7bPN-SNsyd6WCYgnu-RSrNkITviS</td>
</tr>
</tbody>
</table>

Since version 0.2.*, IPStorm has abandoned these topics in favor of a centralized design using the web API module.

Protocols

libp2p protocols come into play when a peer wants to open a direct connection to another peer. The source dials the destination peer specifying a multiaddress and protocol. The protocol is used to identify which handler is invoked in the destination node, which only accepts connections for the protocols it supports.
IPStorm defines a set of its own protocols:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Module</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sreque/1.0.0</td>
<td>storm.reque.client</td>
<td>receiving commands from the reque server and sending responses</td>
</tr>
<tr>
<td>/shsk/1.0.0</td>
<td>storm.handshake</td>
<td>(obsolete) authenticating other peers</td>
</tr>
<tr>
<td>/sfst/1.0.0</td>
<td>storm.filetransfer</td>
<td>sending samples</td>
</tr>
<tr>
<td>/sbst/1.0.0</td>
<td>storm.backshell</td>
<td>executing shell commands on the victim's machine</td>
</tr>
<tr>
<td>/sbpcp/1.0.0</td>
<td>storm.proxy</td>
<td>communicating with the proxy backend</td>
</tr>
<tr>
<td>/sbptp/1.0.0</td>
<td>storm.proxy</td>
<td>receiving proxy connections</td>
</tr>
<tr>
<td>/strelayp/1.0.0</td>
<td>storm.node</td>
<td>authenticating peers used as relays</td>
</tr>
</tbody>
</table>

Node Discovery

The content routing interface that libp2p offers can be used for peer discovery. Nodes advertise themselves as providers for certain CIDs (content IDs) and likewise search for providers, locating peer nodes.

This is achieved by working directly with CIDs through the interface offered by go-libp2p-kad-dht (routing.FindProviders). go-libp2p-discovery offers an alternative way, using namespaces which can be converted to CIDs (routing.FindPeers, routing.Advertise). For each type of discovery used by IPStorm, we list both:

• general peer discovery (provided by all IPStorm nodes):
  - Namespace: fmi4kYtT9789G3sCRgMZVG7D3uKalwtCuWw1j8LSPHQEGVBU5hfbNd
    nHvtt3kyRIfYULGNA00zactmIMIZods0ha9tnfe25Xef1
  - CID: bafkreidcr6e6zr5z4rsgzbs15ewsm4wfgb3jypk7ad2gznudgcdqa

• relay discovery:
  - Namespace: relay:8LSPHQEGVBU5hfbNdnHvt3kyR1fYU1
  - CID: bafkreighrwwcynbhu4swwhkd6y6dudznk2wouxiilm6sq53u6iclin1wy17q

• backend discovery:
  - Namespace: proxybackendH0DHVADBCIKQ4S7YO4X
  - CID: bafkreidu4wapxwcecwwhov2wb53d7mzmyivzmj5raobuegule4syo3btl

• reque discovery:
  - Namespace: requeBOHCHIY2XRMYXI0HCSZA
  - CID: bafkreibapjcai3rzn1q3we4docfeism3kyfwuyxod7gzkgsyekdhd7f4

• web API discovery
  - Namespace: web-api:kYVhV8KQ0mA0rs9pHXoNpD
  - CID: bafkreibzlblmu7weuqzk74dowfdq1fu2spnrag3t3qvveil1lbhqv4qowzq

• seeder discovery
  - Namespace: stfadv: + checksum

Relays

At some point between 0.1.43a and 0.1.51a, IPStorm introduced support for circuit relays. This may have been implemented to improve reachability for nodes that are behind NAT or as an attempt to conceal the management nodes. Shortly after the feature was implemented, most of the management nodes no longer published their IP addresses, using relay circuits instead. As an example, Qmeb3X55MaoKhhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2 listed the following address (among others):
This means that the node at 78.x.x.120:52202 (storm node with ID QmVoDwmbfwSUPT3ds5ytWRwhoWZkzgE9qFHiiYHFJQ5cAnm) is used as a relay when establishing a connection to Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2.

The use of this feature has diminished since version 0.1.85a and most nodes list their external IPs. However, some of the nodes remain hidden behind relays and, in some cases, the relays do not belong to the botnet.

On a related note, another short-lived attempt at concealing the management nodes involved using domains instead of IP addresses. The domains were generated with DNSPod - a domain registration service - and were used for multiaddresses advertised as proxy backends on the DDB topic:

/dns4/splendidobed.site/tcp/443/ipfs/QmViHGaXaG5JzbvH2Xs1Ro19fvoKG1KqpPGMYWLC4ckEAV
/dns4/spenso.me/tcp/443/ipfs/QmViHGaXaG5JzbvH2Xs1Ro19fvoKG1KqpPGMYWLC4ckEAV

Through their configuration, IPStorm nodes enable both the use of relays for outbound connections (EnableAutoRelay()) and for the node to act as a relay hop (EnableRelay(circuit.OptHop)). However, peers that want to use an IPStorm node as a hop have to complete a handshake:

- connect to the relay using the /strelayp/1.0.0 protocol
- send the string "HSR"
- the relay should respond with "+\n"

Nodes that have this feature enabled (this is not true for all versions) advertise themselves on the DDB or using a specific discovery namespace (see “Node Discovery”).

### Modules

The packages included in IPStorm’s version 0.2.05a are:

Packages:
- main
- storm/backshell
- storm/ddb
- storm/filetransfer
- storm/logging
- storm/malware-guard
- storm/node
- storm/powershell
- storm/proxy
- storm/reque_client
- storm/starter
- storm/statik
- storm/util
- storm/web_api_client
Other packages were included in the past but have been replaced or discontinued:

- storm/avbypass
- storm/bootstrap
- storm/handshake
- storm/identity
- storm/peers_cache
- storm/storm_runtime
- storm/vpngate

In this section, we provide a technical analysis of the modules that encompass its core functionality.

**Malware Guard**

This task executes periodically, looking for competing malware. Processes are deemed suspicious if their name, executable path or command line arguments contain any of the strings from a blacklist. The processes are killed and their executables are removed. The blacklisted strings are:

- /data/local/tmp
- rig
- xig
- debug
- trinity
- xchecker
- zpyinstall
- startio
- startapp
- synctool
- ioservice
- start_
- com.ufo.miner

**Backshell**

This module is used for running shell commands on the infected device. The shell is accessed through a libp2p connection with /sbst/1.0.0 as its protocol ID. The module lacks any authentication or authorization.

**DDB**

The distributed database (DDB) is used by bots to store and share configuration data. For synchronization, each bot periodically publishes the entries in their local database on an IPFS topic. They also subscribe to this topic and process the messages, updating local entries with ones with newer timestamps. In this process, only "trusted" nodes are
authorized to update certain keys. libp2p (pubsub) implements message signing; the bot checks that the public key used to sign the message belongs to a hardcoded list of trusted public keys;

An example of a DDB entry retrieved from the associated topic is:

```json
{
    "Command": "SetWithTTL",
    "Key": "file-checksum/storm_android-amd64",
    "Value": "12c3368e17c04f49cfea139148b63fd1ab1a41e26c113991c2bb0835dd02495b",
    "TTL": 3600000000000,
    "T": 1598897109
}
```

The **Command** refers to the operation that should be performed on the database; in this case, the entry has a time-to-live (TTL), which means that it should be discarded after a time interval (the **TTL** value) has elapsed since its publishing (the **T** timestamp). **Key** and **Value** correspond to the actual data stored in the database.

Based on the key, values hold the following meanings:

- `/5aYucluKo1U1/QmStmHQxqVcHf8DcWUF1fTlObfCoT3dnmYPWM8gTKmXGh0Xu-fdekgniT8VwrgMK4kEWCDBs vl2qy0pVymu-Jtu96t0nroXaN1` the most recent version
- `"file-checksum/" + filename` the checksum for the latest bot sample for a specific architecture and OS
- `"seeders:" + checksum` a peer ID of a bot hosting a sample with a specific checksum
- `"seeders-http:" + checksum` the IP and port of an HTTP server hosting a sample with a specific checksum
- `"relays"` a peer ID of a bot that can act as a relay
- `"proxy:backend"` a peer ID of a management node for the proxy module
- `"reque:manager"` a peer ID of a management node for the reque module

The database is an associative array, but there is a distinction to be made from the (key, value) pairs published on the topic. The **Set** and **SetWithTTL** commands store the value associated to the key, replacing the previous one. In the case of the **SAdd** command, the value associated to the key in the database is a list, to which the value from the message is appended.

Although the module is still in use for local storage in the newest sample, the synchronization mechanism has been replaced with queries to management nodes using a web API.

**VPNGate**

This module is used for scraping the API of the public VPN service VPNGate.

The bot performs a request to “http://www.vpntaget.net/api/iphone/” and parses the CSV response. The information obtained about these VPN servers is then published on one of IPStorm’s topics.
The reason for the inclusion of this module in the botnet is probably to overcome certain limitations imposed by VPNGate that restrict the list of servers returned for a request. By scraping in a distributed manner through the botnet, the bot herders can discover a wider selection of VPN servers. The ulterior use of this data is uncertain. However, in light of other discoveries concerning the motivations of the bot herders (see the Attribution section), we hypothesize that these servers may have been (ab)used in the threat actors’ proxy-for-hire infrastructure.

Reque

The *reque* package (standing perhaps for "requests from command-and-control") is used for functionalities related to coordinated scanning for SSH and ADB servers and worm-style infection. The bot connects to an IPFS node referred to as a *reque server* through the `/sreque/1.0.0` protocol. The commands from this server are distributed among a queue of workers. The module is designed to be easily extended to handle new commands. Currently, there are two command handlers, for the *tcp-scan* and *brute-ssh* commands.

The *tcp-scan* command is used to scan an IP range on a set of ports. If the list of targeted ports contains ports 22 or 5555, the module follows the SSH and ADB protocols respectively.

The *brute-ssh* command has as its parameters a list of IP addresses, a port and credentials. If the bot succeeds in obtaining a shell on one of the targeted device, it executes the infection payload, turning the victim into an IPStorm bot. As a honeypot evasion technique, the prompt of the shell is validated using a regular expression before the infection step. The regex matches the "svr04" string, which is the hostname of a Cowrie honeypot.

While in the case of SSH the bot exhibits worm behavior, for ADB the infection phase is not carried out by bots, which only relay the information about the device found back to the reque manager. The actual infection is carried out by one of the bot herder-controlled nodes, which connects to the victim by ADB and issues the infection payload. Based on the data collected from our ADB honeypot, the attackers’ script is equivalent to:

```bash
adb connect $IP:5555
adb root && adb wait-for-device
adb remount && adb wait-for-device
adb shell mount -o rw,remount /data; mount -o rw,remount /system; mount -o rw,remount /
adb shell echo "{
"user":"$(whoami 2>/dev/null)"
,"id":"$(id 2>/dev/null)"
,"machine":"$(uname -m 2>/dev/null)"
,"curl":"$(which curl 2>/dev/null)"
,"wget":"$(which wget 2>/dev/null)"
,"adb":"$(which adb 2>/dev/null)"
,"iptables":"$(which iptables 2>/dev/null)"
,"ipset":"$(which ipset 2>/dev/null)"
,"abi":"$(getprop ro.product.cpu.abi 2>/dev/null)"
,"abi2":"$(getprop ro.product.cpu.abi2 2>/dev/null)"
,"abilist":"$(getprop ro.product.cpu.abilist 2>/dev/null)"
,"abilist32":"$(getprop ro.product.cpu.abilist32 2>/dev/null)"
,"abilist64":"$(getprop ro.product.cpu.abilist64 2>/dev/null)"
,"sdk":"$(getprop ro.build.version.sdk 2>/dev/null)"
}"```
adb push install-recovery.sh /system/bin/install-recovery.sh
adb push storm-install.sh /system/bin/storm-install.sh
adb push sldrgo /system/bin/sldrgo

The fingerprinting information gathered through the echo command serves to adapt the rest of the payload. The last file, sldrgo, is an UPX-packed binary compiled for the CPU architecture of the victim. Its purpose is to download the main bot payload.

**Proxy**

IPStorm proxies function by tunneling the SOCKS5 protocol through libp2p traffic.

The module performs two tasks concurrently: maintaining the connection to a backend and handling incoming streams.

The *proxy backend* is located using either the DDB (older versions) or the node discovery mechanism. The bot then connects to the backend using the `/sbpcp/1.0.0` protocol and periodically pings with its external IP address and latency.

In the initialization phase, it starts a local SOCKS5 proxy on a randomized port using an open source SOCKS5 implementation. A stream handler for the `/sbptp/1.0.0` protocol is instantiated. This allows the node to decapsulate the SOCKS5 proxy traffic from the libp2p stream and forward it to the local proxy. The responses are likewise relayed back to the peer through the libp2p stream. In older versions, this is done using an ad-hoc implementation of bidirectional pipes:

```go
package proxy

import io

// BidirectionalPipe is a bidirectional pipe that decapsulates
// SOCKS5 proxy traffic from a libp2p stream and forwards it back.
type BidirectionalPipe struct{
    ctx context.Context
    lrw io.ReadWriter
    rrw io.ReadWriter
    pipe1 *Pipe
    pipe2 *Pipe
}

// Pipe is a unidirectional pipe that forwards packets from one io.ReadWriter
// to another.
type Pipe struct{
    src io.Reader
    dst io.Writer
}

In newer versions, this is abstracted through the use of the gostream package.

The peer for the incoming connections need not be the same in the logic of the module, but they are both referred to (and tagged in the DDB) as *proxy backend*. As per our observations on our instrumented IPStorm bot, the proxy module receives connection requests shortly after it starts. The source of the connections is the same node that poses as the *proxy backend* (@meb3X55MaoKhZfYsUHFgkZWAz3ZFTQCQz6qiaEqamo7a2). We assume that these are due to the proxy checker mechanism and have not investigated whether the proxy eventually receives real traffic.
Filetransfer

IPStorm hosts the malware binaries in a distributed manner, with each bot "seeding" for one or more samples. This functionality is implemented in the filetransfer package.

The bot regularly checks for updates: it retrieves the latest version number, either from the DDB or through the web API. If a new version is available, it is downloaded and the bot is killed and respawned using the new binary. This process generates a new key pair and therefore a new peer ID.

While the sample matching the native architecture and OS is stored in the filesystem, additional samples are stored in RAM. Depending on available RAM, a number of other samples are downloaded and served on a local HTTP server.

This server is opened on a random port at bot startup and is advertised in the DDB. For each hosted sample, the entries "seeder:" + checksum and "seeder-http:" + checksum are formatted with the bot's peer ID or its external IP and port of the HTTP service, respectively. In newer versions, this is replaced with POSTing the equivalent data to 2 web API endpoints.

The process of downloading the sample has several steps:

1. retrieving the checksum of the latest sample for a specific CPU architecture and OS
2. looking for seeders for the file with that checksum
3. connecting to the peer, either through HTTP or IPFS (protocol /sfst/1.0.0) and downloading the sample
4. validating the checksum

The statik module is used to store a zip file into memory from which files may be retrieved individually. The archive contains:

<table>
<thead>
<tr>
<th>Length</th>
<th>Date</th>
<th>Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1764</td>
<td>2020-06-05</td>
<td>13:37</td>
<td>linux/install.sh</td>
</tr>
<tr>
<td>397</td>
<td>2020-05-12</td>
<td>09:17</td>
<td>storm_android/install-recovery.sh</td>
</tr>
<tr>
<td>2796</td>
<td>2020-05-12</td>
<td>14:18</td>
<td>storm_android/storm-install.sh</td>
</tr>
<tr>
<td>370</td>
<td>2020-05-11</td>
<td>21:42</td>
<td>storm_android/storm-pslist.sh</td>
</tr>
</tbody>
</table>
The first script, `linux/install.sh`, is used as a downloader/dropper for the main payload in SSH infections. The script is customized by the bot ad-hoc, adding a variable containing recent “seeders” for the payload. The scripts in `storm_android` are used for re-configuring persistence on Android devices when the bot is updated.

**WebAPI**

Since version 0.1.92a IPStorm started transitioning away from the PubSub model to a more centralized design. Bots would no longer coordinate using messages posted by other bots. Instead, all information is aggregated by one or more C2 nodes which expose a Web API.

The HTTP protocol is layered on top of libp2p connections using the `go-libp2p-http` package. The services referred to in the IPStorm code as “web API backends” are addressed by their peer IDs and are discovered using the peer discovery mechanism.

The reason for this transition is not clear. Some possible explanations are that the developer(s) have realized that other parties can read and potentially interfere with the topics and desired more control, or that the synchronization was unreliable. The latter would explain the paradox of why we are seeing multiple bots stuck using old malware versions although the code is designed to automatically update to the latest available version.

In addition to ensuring that p2p synchronization is overseen by the bot herder-controlled nodes, the messages remain authenticated and authorization is enforced using the same set of trusted keys as in the case of the DDB.

The API exposes the following endpoints:

- **POST /nodes**
  Parameters:
  - i: fingerprint of the infected system equivalent to the info topic

- **GET /version**

- **GET /files/checksum**
  Parameters:
  - f: filename

- **GET /files/seeders-http**
  Parameters:
  - c: checksum

- **POST /files/seeders-http**
  Parameters:
  - c: checksum
  - s: seeder IP and port
Mapping the Botnet

In our efforts to monitor the botnet we have used data from multiple sources:

- publicly available "peer info" (ID, public key, addresses and versioning information)
- messages published on topics: the DDB topic, the info topic
- querying the DHT for peers that are part of the botnet

In the first approach, we used the database of peer information collected by our IPFS crawler. The nodes that belong to the botnet are easily identified by their AgentVersion, which is analogous to the User-Agent from the HTTP protocol. For applications built using go-libp2p, the AgentVersion is set by default to the name of the main package. IPStorm nodes have the AgentVersion set to storm. This property enabled us to find nodes that don’t announce themselves on other channels, such as the info topic.

Secondly, we used the same mechanism that enables IPStorm bots to find peers: querying the DHT for specific content IDs provided by different classes of nodes within the botnet (regular nodes or ones with special roles).

Information about the IDs of bots is also available in the info topic and the DDB topic. We used the data in the DDB topic for tracking the versions and the roles of the peers controlled by the threat actors. The data from the info topic gave insight into the distribution of victims by country, device type and OS.

In estimating the size of the botnet, we faced the problem that there is no clear way of discerning whether two node IDs represent the same infected device. The ID can be changed through version updates or reinfection. The external IP is not a good identifier either, because it can change over time or because multiple nodes can be behind NAT.

Based on the number of different IDs seen in a week, averaged over several weeks, we estimate the size of the botnet at around 9,000 devices. The overwhelming majority have Android as their operating system and about 1% use Linux. We have seen Darwin only in a few entries on the info topic which clearly represent the same machine, the one used for development of IPStorm.

There are still a number of devices that, according to the messages they post on the info topic, run versions of IPStorm prior to 0.1.* and have Windows as their OS. Since the new campaign focuses solely on Unix systems, it is remarkable that the bots have persisted on these devices since the 2019 campaign.

Although the fingerprint contains no specific information about the model of the device, clues can be gathered from the OS, kernel version and, sometimes, the hostname and user names. Based on this, we have identified various models of routers, NAS devices, UHD receivers and multi-purpose boards and microcontrollers (such as Raspberry Pi), which may belong to IoT devices.

The following figures refer to the geographic distribution of victims, as identified by external IPs. Most are in Asia. Affecting 98 countries in total, the botnet appears powerful in terms of its capacity for scanning the Internet, but through its choice of attack vectors targets classes of devices that are more prevalent in certain countries. Our honeypots saw on average 3,500 attacks/day from this botnet, which amounted to a large share of the traffic.
Bitdefender DracoTeam • WHITEPAPER
Looking Into the Eye of the Interplanetary Storm

<table>
<thead>
<tr>
<th>Country</th>
<th># of victims</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hong Kong</td>
<td>2901</td>
</tr>
<tr>
<td>Republic of Korea</td>
<td>2125</td>
</tr>
<tr>
<td>Taiwan</td>
<td>1018</td>
</tr>
<tr>
<td>Brazil</td>
<td>743</td>
</tr>
<tr>
<td>Ukraine</td>
<td>524</td>
</tr>
<tr>
<td>United States</td>
<td>380</td>
</tr>
<tr>
<td>China</td>
<td>353</td>
</tr>
<tr>
<td>Sweden</td>
<td>351</td>
</tr>
<tr>
<td>Russia</td>
<td>336</td>
</tr>
<tr>
<td>Venezuela</td>
<td>242</td>
</tr>
<tr>
<td>Panama</td>
<td>217</td>
</tr>
<tr>
<td>Canada</td>
<td>159</td>
</tr>
</tbody>
</table>

Distribution of IPStorm victims by country:

Distribution of IPStorm victims by precise location:
Infrastructure

Although we didn’t manage to get ahold of binaries from the management infrastructure, they are likely developed in the same project, and we can find some clues related to them in the bot binaries. The fact that the management nodes use the same `AgentVersion`, which is set based on the path of the main package, supports this assumption.

The package tree contains some additional packages from which no code (besides some initialization functions) is used by the bots:

- `storm/commander/web_app/router`
- `storm/proxy/checker`
- `storm/util/storm_runtime`

These are perhaps the packages corresponding to a web interface for managing the victims and an automated checker for the availability of the proxies.

The special roles assigned to nodes from the management infrastructure - that we know of - are:

- proxy backend; the bot's proxy module pings these nodes to prove its availability
- proxy checker; a node that connects to a bot proxy
- reque manager; a node that issues scanning and brute-forcing commands (see the reque module)
- web API backend; a node that hosts a web API (used in newer versions)
- trusted node; a node whose public keys is included in a trust list (can sign authorized messages)
- development node; used by the threat actors for development purposes

The following nodes have been observed:

- **ID:** QmW1ptn27xSAg2qBvJwhaGWmJunjzqAGt1oAj4LdVAm9vM
  Roles: trusted, web API backend
  Addresses: /ip4/212.x.x.100/tcp/444, /ip4/212.x.x.100/tcp/5554, /ip4/88.x.x.34/tcp/444

- **ID:** QmdmBf2PfNXu6KVkp63rcLhWpNqQdFZp649dRXS6et
  Roles: unknown
  Addresses: /ip4/212.x.x.100/tcp/443

- **ID:** Qmeb3X55MaoKh2fsYsUHFgkZWAz3ZftQCQz6qiAEqamo7a2
  Roles: trusted, web API backend, reque manager, proxy backend, proxy checker

- **ID:** QmViHGaxG5JzbvH2Xs1Ro19fvoKG1KqpPGMYWLc4ckEAV
  Roles: proxy backend, reque manager
  Addresses: /ip4/54.x.x.216/tcp/443, /ip4/54.x.x.216/tcp/5555

- **ID:** QmShLAfGGVDD32Gux5Q2YvuhdDmluHAmV8aozV6uRKAZdW
  Roles: trusted
  Addresses: unknown

- **ID:** QmNL3UTHbfCxhrkA8Lu9aL2XpPGGYQmN4dihXsaCapiyNx
  Roles: trusted
  Addresses: unknown

- **ID:** QmV7ULoDmUv3XviN1GqrsC6t8WLPgmCTMAJ544x2pbA
  Roles: proxy backend
  Addresses: /ip4/45.x.x.194/tcp/443
Some of these nodes have changed their addresses and roles over time. The listed information is sourced from publicly available peer information, collected over recent months. We have only included multiaddresses which contain an external IP and port. For some nodes, our data only contains relay circuit addresses and their address is therefore listed as unknown.

An interesting side-development is that a group of seven nodes appeared on 17 September 2020, announcing themselves as providers for the web API CID:

QmNeV49LPkgQENkpSmg6q8nBljypY74jhtWxP9rANUQubd
QmRG9bwpWumxxNwGbcexMrkkAvneNDw48kiTRaJSt8rSmzp
QmXPUHuy4e2jg6dqjfsTnseC5m5KHgZvqMr6AwVutdcQQL
QmYmDUKgJJ5K2BQ6JYuJeN2SJB3wfDg2N78th6tMEMHgSF
QmW1ptn27xSAGZqBvJwhaGmJtnjzqAGtIoA4LdVAm9vM
Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2
Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2

They are not reachable and therefore their origin and purpose cannot be ascertained. One theory is that they may belong to an attempt to take down the botnet with a Sybil attack: the bots fruitlessly try to reach the false web API nodes and the functionality that depends upon the module is impeded. Otherwise they may belong to fellow researchers, in which case, Hello!

**Attribution**

While some management nodes were a later addition to the botnet's infrastructure, the nodes hosted on two particular IPs have played central roles since we started this investigation.

As a starting point, we searched for the nodes holding the trusted keys hardcoded in the binary. We found Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2 and QmW1ptn27xSAGZqBvJwhaGmJtnjzqAGtIoA4LdVAm9vM, which had at that time the following peer infos:

```
{
    'Addresses': ['/ip4/54.x.x.216/tcp/444'],
    'AgentVersion': 'storm',
    'ID': 'Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2',
    'ProtocolVersion': 'ipfs/0.1.0',
}
Looking Into the Eye of the Interplanetary Storm

Starting from their public IPs, 54.x.x.216 and 212.x.x.100, we have located two other nodes hosted on each of these IPs on port 443 (QmViHGaXaG5JzbvH2xS1Ro19fvoKG1KqpPGMYWLc4ckEAV and QmdmF2PfNxu6KVkp63rcLhWpNgQdaQdPZzP649dRXS6et). The IPs and port hosting each node has varied as depicted in the following table, but their association with the botnet (and management nodes in particular) has been constant.

<table>
<thead>
<tr>
<th>IP</th>
<th>Port</th>
<th>ID</th>
<th>Last seen</th>
</tr>
</thead>
<tbody>
<tr>
<td>54.x.x.216</td>
<td>444</td>
<td>QmViHGaXaG5JzbvH2xS1Ro19fvoKG1KqpPGMYWLc4ckEAV</td>
<td>7th September 2020</td>
</tr>
<tr>
<td>54.x.x.216</td>
<td>443</td>
<td>QmViHGaXaG5JzbvH2xS1Ro19fvoKG1KqpPGMYWLc4ckEAV</td>
<td>29th September 2020</td>
</tr>
<tr>
<td>54.x.x.216</td>
<td>443</td>
<td>QmViHGaXaG5JzbvH2xS1Ro19fvoKG1KqpPGMYWLc4ckEAV</td>
<td>5th September 2020</td>
</tr>
<tr>
<td>212.x.x.100</td>
<td>443</td>
<td>QmdmF2PfNxu6KVkp63rcLhWpNgQdaQdPZzP649dRXS6et</td>
<td>6th September 2020</td>
</tr>
<tr>
<td>212.x.x.100</td>
<td>443</td>
<td>QmdmF2PfNxu6KVkp63rcLhWpNgQdaQdPZzP649dRXS6et</td>
<td>5th September 2020</td>
</tr>
<tr>
<td>212.x.x.100</td>
<td>443</td>
<td>QmdmF2PfNxu6KVkp63rcLhWpNgQdaQdPZzP649dRXS6et</td>
<td>6th September 2020</td>
</tr>
<tr>
<td>212.x.x.100</td>
<td>443</td>
<td>QmdmF2PfNxu6KVkp63rcLhWpNgQdaQdPZzP649dRXS6et</td>
<td>6th September 2020</td>
</tr>
<tr>
<td>212.x.x.100</td>
<td>443</td>
<td>QmdmF2PfNxu6KVkp63rcLhWpNgQdaQdPZzP649dRXS6et</td>
<td>7th September 2020</td>
</tr>
</tbody>
</table>

We assert that these two IPs are unquestionably in the ownership of the bot herders. The fact that the management nodes -with the exception of the "development" nodes- have never posted on the info topic proves that they do not run the same bot as the victims. Two of the nodes hosted here possess "trusted" keys and have played privileged roles in the p2p botnet (issued scanning commands, managed proxies or hosted the web API). Another argument is that the nodes hosted on these two IPs have been operational for several months, while regular bots change their IDs when they update.

Furthermore, 54.x.x.216 is the source IP of the attacks over ADB seen in our honeypots.
DNS records from RiskIQ show that these IPs have been linked with two subdomains of a domain that we will refer as "the domain".

DNS A record of the first sub-domain links with IP 212.x.x.100 (seen from 2019-07-24 14:25:41 to 2020-09-09 10:51:24)
DNS A record of the second sub-domain links with IP 54.x.x.216 (seen from 2020-05-04 23:09:57 to 2020-09-09 05:40:33)

Whois records confirm that the ownership of these IPs has remained constant in this period.

The domain is a paid SOCKS5 proxy service. They advertise over 7,000 proxies from all over the world and claim to be "highly anonymous." An interesting thing mentioned in their FAQ is that:

"Every hour we have about 3-10% of new IPs."

The website of the proxy service seems to be offering four standard pricing packages on a monthly subscription that ranges from $74 to $259, depending on the number of concurrent TCP connections – from 150 for the lowest tier offering to 3,000 for the highest tier. However, pricing seems flexible, as the website also offers tier plans that are either geolocation-specific, such as proxies from Europe or North America, world mixed, or even for developers.

The pricing diagram covers anything from hourly tariffs that start from $2 and peak at $14, to daily, weekly or monthly. The highest monthly tier pricing involves a monthly Premium package for $499 and offers mixed proxies from all over the world, including Europe, CIS, and North America, and seems to involve access to the entire live victim infrastructure.

We claim that whoever is behind the domain is also behind IPStorm and uses the infected devices to back their proxy service for financial gain. And the evidence goes further.

More information about these particular IPs and domains can be freely provided to law enforcement agencies by reaching out to draco@bitdefender.com.

The Developer

We have noticed some artifacts in binaries from versions versions 0.0.* and 0.1.* respectively:

/Users/[redacted]/go/src/storm
/Users/dummy/Documents/GoLandProjects/storm

These are the paths of the IPStorm Go project that was stored on the machines where the malware was developed and compiled. While the username “dummy” is common, the other username (which has been redacted) is distinctive enough to investigate further. It is plausible that the developer took note of the fact that their username was mentioned in the Anomali article and sought to be more “anonymous”.

Similar paths are present in the binary loaders used for infection over ADB:

/Users/dummy/Documents/CppProjects/loader

We found several related entries on the info topic. For example:

```
{
  "T" : 1597849326,
  "HostID" : "QmdACwNe1JdkD2N45LRbcFthPpEVQVtfBvuHFHQ4cFMcaZ",
  "Version" : "0.1.90a",
  "Platform" : "darwin- amd64",
  "SystemInfo" : {
    "GoOS" : "darwin",
    "Kernel" : "Darwin",
    "Core" : "19.5.0",
    "Platform" : "x86_64",
    "OS" : "Darwin",
    "Hostname" : "MacBook-Pro-16.local",
    "CPUs" : 16
  },
  "Uid" : "501",
  "Gid" : "20",
  "UserName" : "dummy",
  "UserDisplayName" : [redacted],
  "UserHomeDir" : "/Users/dummy",
  "IsAdmin" : false,
  "ExecutablePath" : "/Users/dummy/Documents/GoLandProjects/storm/storm",
  "InstallationPath" : "/Users/dummy/Documents/GoLandProjects/storm/storm",
  "ComputerID" : "",
  "LocalIPs" : null,
  "ExternalIP" : "163.172.x.x",
  "Processes" : null
}
```

These matched what we knew so far about the development environment: the path of the main project and the username [redacted]. The presence of these entries on the info topic, while the management nodes lack the feature, indicates that these nodes are used for development and testing. Although we have not managed to link the IP 163.172.x.x to any subdomain, we noticed that it is from the same hosting provider as other IPs linked with them and may be part of their cloud infrastructure. The interesting association is that it is in the same ASN (AS12876) as the IP of the subdomain that hosts a Gitlab instance (163.172.x.y).

The projects on this Gitlab are hidden, but other information is publicly available. The screen capture, taken in August 2020, shows the existence of a group which contains the same username.
Groups on the Gitlab instance:

![Gitlab Members](image)

Username has been redacted

The user’s profile on Gitlab:

![Gitlab Profile](image)

Username has been redacted

All this evidence indicates that the threat actors behind the Interplanetary Storm malware are the same people behind the SOCKS5 proxy-as-a-service infrastructure that’s rented in exchange for virtual currency.

More information about this threat actor can be freely provided to law enforcement agencies by reaching out to draco@bitdefender.com.

Conclusions

It is because of this thorough investigation that we believe the Interplanetary Storm botnet not only has the capabilities to act as an anonymization proxy-for-hire infrastructure, but the malware has a highly active development cycle. The main purpose of the botnet seems to be financial, as the code went through a considerable number of revisions to create a reliable and stable proxy infrastructure.
Bitdefender DracoTeam • WHITEPAPER
Looking Into the Eye of the Interplanetary Storm

Indicators of Compromise (IoCs)

Bot samples (version 0.2.05a):

71e7bb56899c7860729119734053409b6e8502f9
f7b008c30f555f5f892211aeaa054d00ba8c093b7
dc917a8a6e80616231639676929db945099062a9
c422c0a74ec5edd713e80ea1c0b7c8863a4816b
98c80465c1ca0f54b59462013875a99b23a43ce73f
ea665844b69815d6543cf4c8e6351d0129d7c669
7ae71c05a3de0165b88ed8a8ca866674c03e3
f746e810450a31318450e211484f944daa005b3
90d0ecb2489b4d958d180e9e7d527f0c659c004

“/storm_linux-amd64”
“/storm_linux-386”
“/storm_linux-arm64”
“/storm_linux-arm7”
“/storm_darwin-amd64”
“/storm_android-arm7”
“/storm_android-arm64”
“/storm_android-amd64”
“/storm_android-386”

Downloaders and persistence:

9cc0273d83f0c950c5adfb5e374a55d7503679a5
aee0848450b28bbe1f24ee02878e3c3329955a40
db2d7829e305ffea947073973335a914117ec2a
0546c9436a87e0294806876d76f7b63a19fa87de

File names:

storm.key

Bot samples (older versions):

7ae71c05a3dea0165b88ed1a8caa8666b74c03e3
b61b77c87ed5b0b3d95c975df5e4ec85632a638
0db8496f0c8a8664c5ab645600a4e11da78f83a
f0f9f5fb273d63a1f5a03797a63790cbeb460
e7f417e57a7faca4669860dd5d38fdada2e4f8c1d
ff87a2f75d0008455828c218ec91fde06b63196d4c
05ebe2d2e8a98da991699f88bda9d1160525957
7afac2e9594f83767f9f77f7d462989897b964a
b47c25a3e34efb40023e6addcfe2e2a721ba8
d6674f5563f07a4ef2db7e37967cfefe8b98e85e
3ccbd4044623f9639277baaf9f3d6bec42c66fcc0
3d31447e55a9692b196147750132770dfaf8e5b
a79933ebf0d8745c2742df4bbf8525d74db8e14
c9a570ef4f4a2511955d704643455dab2d5930
b99933ebf0d8745c2742df4bbf8525d74db8e14
c9a570ef4f4a2511955d704643455dab2d5930
a290a93e20f623d9530c01c67cd7a9b9c3c5ab9d3628
f9c33e6eb700c81b1d97d990ba3b986b74a593c9
007dd1362ca43d065b0ca633a6099493063df7ca
3be8947a898d0539666c98c0053be84c006fdfc

086ce30530db7a1b729b9b2b70cd4a1dccc2fa9e6
dcdd9b4f3fb5a713e5e6ac81f5eb0ff1f283ee7ea
7fdec673db4fdd339195cc6aad389579ff8ff02
92b02a4987b360a50f96f86ef3b78a8df2ad1a8
9d51af0e3602702d5096d108a2ffe620031cd2
88aaa8f5c03ffdf76c3770f6349fbb8ed9ab4
161dd2e5634ef9f585632500ea7018864e9a155
2d522ba28f92922fc80dce4a3a16c2f1273effe
d32cf21edac6c196ebded79081a8316e844112ad2
1ac42f0d5f53e3586e9149268239e427d14
be3cc9c3800c1aecd562620cc741183f856085f0
6ec780aeced007d739b0ef9f03ce228166a83173
6f9ff5d06ea1a34b8022c8314fc3b8672effae08
4b03ff0b369efbe695ad8d8121e161c89fad3
9e63567f7e670a358af584e66bca968513c9d
b9d2816405979fc528a4e71b8955242ef251af
5a0f8d0607e20aee0157a5572039ff255c0ae88b
5d1aa62b6cb7b5678a3697130afbcdf45932f4af
WHY BITDEFENDER?

UNDISPUTED INNOVATION LEADER.
38% of all cybersecurity vendors worldwide integrated at least one Bitdefender technology. Present in 150 countries.

WORLD’S FIRST END-TO-END BREACH AVOIDANCE
The first security solution to unify hardening, prevention, detection and response across endpoint, network and cloud.

#1 RANKED SECURITY. AWARDED ACROSS THE BOARD.