WHITEPAPER

web use only

DRACO TEAM

Looking Into the Eye of the Interplanetary Storm

Bitdefender

www.bitdefender.com

Contents

Author:
Silvia Pripoae - Security Researcher, Bitdefender

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:

Description	Purpose	ID
info topic	information about the infected system	C0FAh40EtzpBb145aYUcluKo1UJZ-1bRUGg1WJo3OFzWFEKzbShsWFO-wt95Log1eP0gp22hX3PQ2iU5w83a8uoPw-C3WngMgFD1
cmd topic	commands (version `0.0.*`)	6szrvClvhS8QSTs6nq0I28i77MNO1DVh4pCVBvRYxm-5dGHPsOa8sNCMypcPoMFTe5BeXWPnNILo3I6G6Ev1gQLeqz54RoMwW9pPFD1
VPNGate scraper topic	information about VPNGate servers	HyD9c7ZrNrXZqG-M7lVVdOHH5DjmHFz-PBHGk1OdqEpVR-Nn7MXo-bDjX7nMdAnqq-SNSyd6WCYgnu-19uT0rvsK3V7bPN-RSrNkITviS
DDB topic	synchronization messages for the DDB	M7lVVdOHH5DjmHFzPBHGk1OdqEpVR-HyD9c7ZrNrXZqGNn7MXo-bDjX7nMdAnqq19uT0rvsK3V7bPN-SNSyd6WCYgnuRSrNkITviS

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:

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

Node Discovery

The **content routing** interface that libp2p offers can be used for peer discovery. Nodes advertise themselves as providers for certain `CID`s (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: `fmi4kYtTp9789G3sCRgMZVG7D3uKalwtCuWw1j8LSPHQEGVBU5hfbNdnHvt3kyR1fYUlGNAO0zactmIMIZodsOha9tnfe25Xef1`
  CID: `bafkreidcr6e6zr5zs4rqgzbsl5ewsrm4wfgb3jydpk7ad2gcznudglcdqa`
- relay discovery:
  Namespace: `relay:8LSPHQEGVBU5hfbNdnHvt3kyR1fYU1`
  CID: `bafkreigrhwwcynbhu4swhhkd6y6dudznk2wouxilm6sq53u6icinlwyi7q`
- backend discovery:
  Namespace: `proxybackendH0DHVADBCIKQ4S7YOX4X`
  CID: `bafkreidu4wapxwcecwvwhov2wb53d7mzmyivzmj5raobuegule4syo3bti`
- reque discovery:
  Namespace: `requeBOHCHIY2XRMYXI0HCSZA`
  CID: `bafkreifapjcai3rznlq3we4docfeism3kryfwuyxod7ggzksbyekdhd7f4`
- web API discovery
  Namespace: `web-api:kYVhV8KQ0mA0rs9pHXoWpD`
  CID: `bafkreibzlbmu7weuqzkt4dwfwdqlfu2spnrsg3t3qvveilllbhqv4qowzq`
- 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, `Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2` listed the following address (among others):

"/ip4/78.x.x.120/tcp/52202/p2p/QmVoDwmbfwSUPT3ds5ytWRwhoWZkzgE9qFHiYHfJQ5cAnm/p2p-circuit/p2p/Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2"

This means that the node at 78.x.x.120:52202 (storm node with ID `QmVoDwmbfwSUPT3ds5ytWRwhoWZkzgE9qFHiYHfJQ5cAnm`) 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:


```
{
    "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/QrnStmHQXqVcHf8DcWUf1FtIObfCoT3dnmYPWM8gTKmXGhOXu-fdekgniT8VwrgMK4kEWCdBsvI2qy0pVygmu-lJtu96t0nroXaN1"
  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.vpngate.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:

```
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\":\"$(id 2>/dev/null)\",
  \"root_access\":\"$(getprop persist.sys.root_access 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.

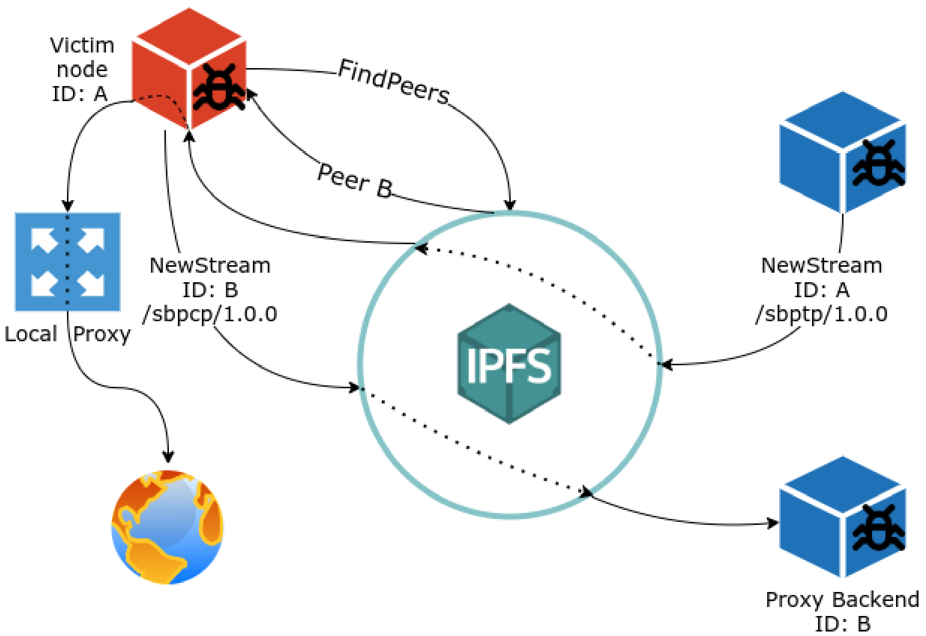
The `proxy backend` is located using either the DDB (older versions) or the node discovery mechanism. The bot then connects to the backend using the `/sbpcp/1.0.0` protocol and periodically pings with its external IP address and latency.

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:

```
type proxy.BidirectionalPipe struct{
        ctx context.Context
        lrw io.ReadWriter
        rrw io.ReadWriter
        pipe1 *proxy.Pipe
        pipe2 *proxy.Pipe
}
type proxy.Pipe struct{
        src io.ReadWriter
        dst io.ReadWriter
}
```

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 (`Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2`). 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:

```
  Length      Date    Time    Name
---------  ---------- -----   ----
     1764  2020-06-05 13:37   linux/install.sh
      397  2020-05-12 09:17   storm_android/install-recovery.sh
     2796  2020-05-12 14:18   storm_android/storm-install.sh
      370  2020-05-11 21:42   storm_android/storm-pslist.sh
```

```
      190  2020-05-11 12:24   storm_android/storm.rc
---------                     -------
     5517                     5 files
```

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.

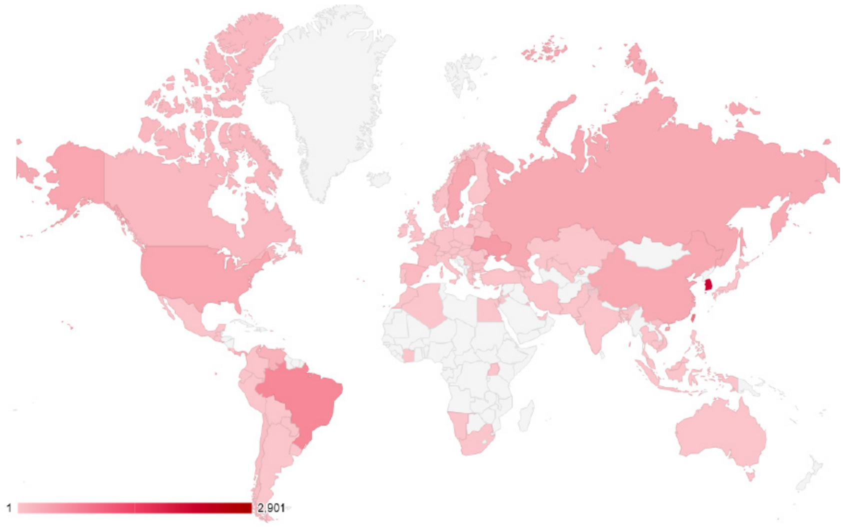
There are still a number of devices that, according to the messages they post on the info topic, run versions of IPStorm prior to `0.1.*` and have Windows as their OS. Since the new campaign focuses solely on Unix systems, it is remarkable that the bots have persisted on these devices since the 2019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

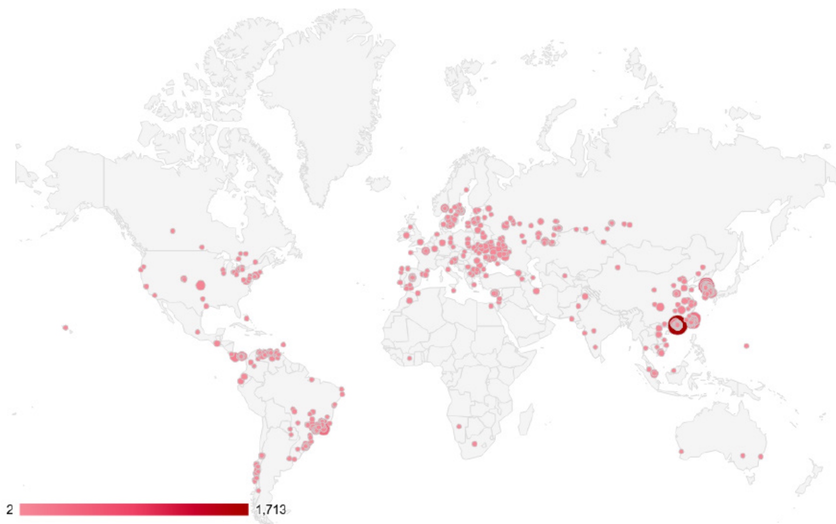
The following figures refer to the geographic distribution of victims, as identified by external IPs. Most are in Asia. Affecting 98 countries in total, the botnet appears powerful in terms of its capacity for scanning the Internet, but through its choice of attack vectors targets classes of devices 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.

Country	# of victims
Hong Kong	2901
Republic of Korea	2125
Taiwan	1018
Brazil	743
Ukraine	524
United States	380
China	353
Sweden	351
Russia	336
Venezuela	242
Panama	217
Canada	159

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: `QmW1ptn27xSAgZqBvJwhaGWmJunjzqAGt1oAj4LdVAm9vM`
  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: `QmddMf2PfNXu6KVKp63rcLhWpNqQdaQdPZzP649dRXS6et`
  Roles: unknown
  Addresses: /ip4/212.x.x.100/tcp/443
- ID: `Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2`
  Roles: trusted, web API backend, reque manager, proxy backend, proxy checker
  Addresses: /ip4/54.x.x.216/tcp/444, /ip4/54.x.x.216/tcp/5554, /ip4/101.x.x.240/tcp/443, /ip4/101.x.x.222/tcp/443, /ip4/111.x.x.85/tcp/443
- ID: `QmViHGaXaG5JzbvH2Xs1Ro19fvoKG1KqpPGMYWLc4ckEAV`
  Roles: proxy backend, reque manager
  Addresses: /ip4/54.x.x.216/tcp/443, /ip4/54.x.x.216/tcp/5555
- ID: `QmShLAfGGVDD32Gsx5Q2YbuhdDm1uHAmV8aozV6uRKAZdW`
  Roles: trusted
  Addresses: unknown
- ID: `QmNWL3UTHbfCxhKA8Lu9aL2XpPGGYQmN4dihXsaCapiyNx`
  Roles: trusted
  Addresses: unknown
- ID: `QmV7UYLoDmUv3XViiN1GqrsC6t8WLPGmCKTMAJ544x2pbA`
  Roles: proxy backend
  Addresses: /ip4/45.x.x.194/tcp/443

- ID: `QmdACwNe1JdkD2N45LRbcFthPpEVQVtfBvuHFHQ4cFMcAz`
  Roles: development
  Addresses: unknown (External IP: 163.172.x.x)
- ID: `QmNoV8qAgLTEo1nQN8wmusi9U9kmArUjVNsKY4TdYS1wvT`
  Roles: development
  Addresses: unknown

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`:

```
QmNeV49LPkgQENkpSmg6q8nB1jypY74jhtWxP9rANUQubd
QmRG9bwpWumxkNwGbceiMXmikAeNDw48kiTRaJSt8rSmzp
QmXPUhUy4e2jg6dqjfsTnseC5m5KHgZvqMr6AwVutdcQGL
QmYmDUkGJJ5K2BQ6JYuJeN2SJB3wfDg2N78th6tMEmHgSF
QmbNwkGiHrK9XcP6n2vMZh9sZMoicvcrhaTbtXMzoC8rp1
QmbcbZb8Jq8u44oCqna5h2ZBjSpRT42dNLDDNCyuSfGu3L
Qmek3KNuJY3eRbSGZ9zm2M8yYLb7bMeA28gMswQXarhbmW
```

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 `QmW1ptn27xSAgZqBvJwhaGWmJunjzqAGt1oAj4LdVAm9vM`, 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',
```

```
 'PublicKey': 'CAASpgIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCzR75b/L1QhliHV/yFZF
LNQ6MC5JsbU3pvwBQN7yP82+vKNV6gu0bUozOUIVc2O/EZLbMyiS+cKrEWNue72/2FGnM84DfTWBYh
UiYrhJVupHXGGEfBFBfy1LtswXoEaW584up1CQwdKlbKKw4Fqt2NUCEHlX8IWRZS4F34N53xqFBb1RNwgmfa
xdagttsP+20jCI10sLkCk6OnMAasyQCkOB+xvH6t4nakztipe/xgpU6kv5CittTfKFnRq952TS0G6a4m7OruX
TVIJEqiDV/E+avFI2JdJoe7NzKwrtbrfx01nSNCvDSqRNxd4S1HfGu6rpse+3YbmLoMwEDVPVvjAgMBAAE='
}
{
 'Addresses': ['/ip4/10.12.128.139/tcp/444', '/ip4/212.x.x.100/tcp/444'],
 'AgentVersion': 'storm',
 'ID': 'QmW1ptn27xSAgZqBvJwhaGWmJunjzqAGt1oAj4LdVAm9vM',
 'ProtocolVersion': 'ipfs/0.1.0',
 'PublicKey': 'CAASpgIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCw4JxaWchBO9fGmyjNI
OwA/L1G3P6zz3QrsJmbwJV0o3GDYjyLG7RzTeqi1NySvwpUW7mE7IP/XIGrDgbXodNGr+/o5bcHXosJwCccS
EJvxcqL9+ePBO4uwcOIUhxDtWiOjEdPwQ5jTua4S2oeJgNwsPIn3SEz2A5UqdstFB6SJBqth8OT4MGqkXKoWE
uBtA6Qc5pUYnhgUSznQH77rqbygiudM9d9BFg6jlvMoUrBqXj83tC31OZK45AzY2cEH3Lz2UzCgO36DUmUH
BB7qR01Up6v9QNGW9pBQkDolFxrJputv+zavthxcrkyhCbdHvrlBC0kw+1uSRaPD4A/FE/zAgMBAAE='
}
```

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 `QmddMf2PfNXu6KVKp63rcLhWpNqQdaQdPZzP649dRXS6et`). 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.

IP	Port	ID	Last seen
54.x.x.216	444	Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2	7th September 2020
54.x.x.216	443	Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2	29th September 2020
54.x.x.216	443	QmViHGaXaG5JzbvH2Xs1Ro19fvoKG1KqpPGMYWLc4ckEAV	5th September 2020
212.x.x.100	443	QmddMf2PfNXu6KVKp63rcLhWpNqQdaQdPZzP649dRXS6et	6th September 2020
212.x.x.100	443	QmViHGaXaG5JzbvH2Xs1Ro19fvoKG1KqpPGMYWLc4ckEAV	5th September 2020
212.x.x.100	443	QmW1ptn27xSAgZqBvJwhaGWmJunjzqAGt1oAj4LdVAm9vM	29th September 2020
212.x.x.100	444	QmW1ptn27xSAgZqBvJwhaGWmJunjzqAGt1oAj4LdVAm9vM	6th September 2020
212.x.x.100	444	Qmeb3X55MaoKhZfYsUHFgkZWAz3ZFtQCQz6qiaEqamo7a2	7th September 2020

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 Noth 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
```

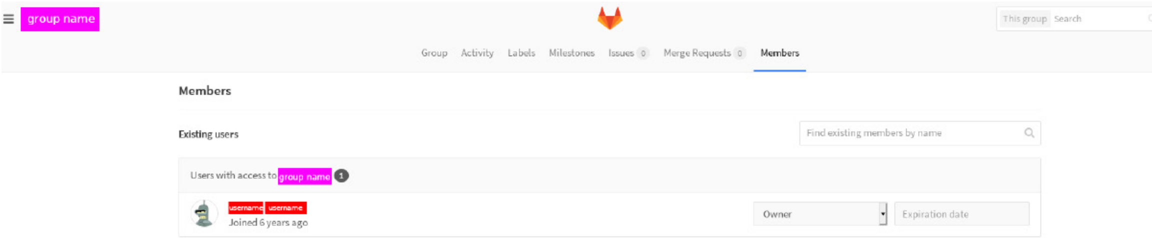
We found several related entries on the info topic. For example:


```
{
    "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).`

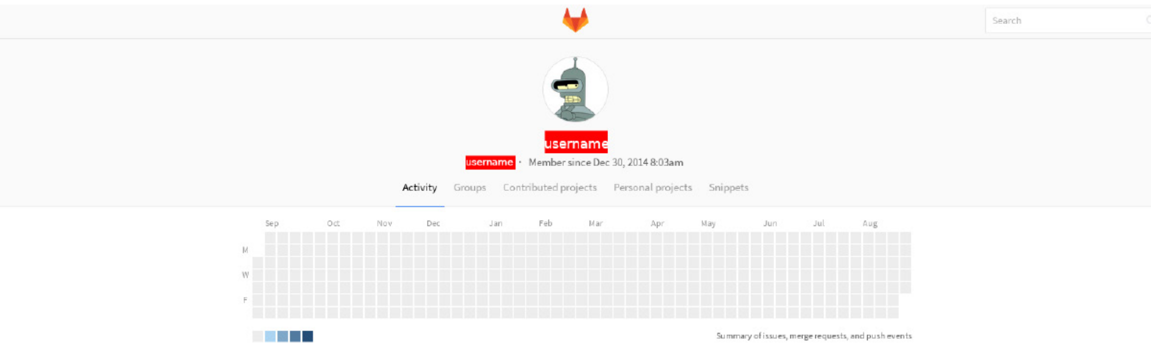
The projects on this Gitlab are hidden, but other information is publicly available. The screen capture, taken in August 2020, shows the existence of a group which contains the same username.

Groups on the Gitlab instance:



Username has been redacted

The user's profile on Gitlab:



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.

Indicators of Compromise (IoCs)

Bot samples (version 0.2.05a):

```
71e7bb56899c7860729119734053409b6e8502f9  "/storm_linux-amd64"
f7b008c30f5555f892211aeaa054d00ba8c093b7  "/storm_linux-386"
dc917a8aa6e8061623163967629db945099062a9  "/storm_linux-arm64"
c442c0a74ec5eddd713e80ea1cb07c8863a4816b  "/storm_linux-arm7"
98c80465c1caf0b59462013875a99b23a43ce73f  "/storm_darwin-amd64"
ea665844b69815d6543cf4c8e6351d0129d7c669  "/storm_android-arm7"
7ae71c05a3dea0165b88ed1a8caa8666b74c03e3  "/storm_android-arm64"
f74c6e810450a31184503e211484f944dda005b3  "/storm_android-amd64"
90d0ecb2489b4d958d1480e9e7d527f0c659c004  "/storm_android-386"
```

Downloaders and persistence:

```
9cc0273d83f0c950c5adfb5e374a55d7503679a5
aee0848450b28bbe1f24ee0287e83c3329955a40
db2d7d829e305feea947073973335a914117ec2a
0546c9b436a87e029480687d6df7b63a19fa87de
```

File names:

```
storm.key
```

Bot samples (older versions):

```
7ae71c05a3dea0165b88ed1a8caa8666b74c03e3
b61b77c87ed5b0b3d95c975daf5e4ec85632a638
0db8496f0c8a8664c5ab645600a4e11da788ffa3
f0f995fb273d6a7e3fd15aa3797a63790cbfe460
e7d41e75a7faca4669860dd5d38fddda2e4f8c1d
ff872f75d00008444282c18ec91fde0b63196dc4
05be2d82e8a98da991699f88bda9d1160525c957
7afac23e95b4f83769f7ff7df462988d997b964a
b47c25a3e34efb40023eeeaddcefa2ead2a71ba8
d6674f5563f07a4ef2db7e37967cffe78b98e85e
3ccdbd4044623f9639277baa9f3dbec42c66fcf0
d331447e55a99692b196147750132770daf28e5b
a79933eb5fd08745c2742dff4b852d57d4b681e4
c9a570eef414a2511955d704643455ddbfe5d930
d39a2f63dc953c0c1c67cd7a9bec3c5aab9d3628
fcbc3eb70cc8b1bd19b7d990ba360f1ea2a359c3
007dd1362ca43d06b0ca633aa6099493063df7ca
3be8947a898d0539666c98c00b53ebe84c006fcf
086ce30530db7a1b72b9b0b270cd4a1dcc2fa9e6
dcdd9b4f3fb5a713e5e6ac81f5eb0f1f283ee7ea
7fdec673db4fdf3391995cc6adc3895794f8ff02
92b02a4987b360a50f96f86ef3b78a8df2a4d1a8
9d51af0e3602702d5096d108aea2fff620031cd2
88aa38f5c03ffdf7f6c3770f6349fbbc86bd9ab4
161dd2e5634e9f5f85632500ea701886ce49a155
2d522ba28f929922fc80d6e4a3a16c2f1273effe
d32cf21edad6c196ebded79081a8316e84411ad2
1ac42f0bdf5e338c3586be914d9268239e427d14
be3cc9c380cc1aecd562602c7c41183f5865085f
6ec780aecad007d739b0ef90f30e228166a83173
6f9f5d06ea1a34b8022c8314fc38b67e2effae80
4bcff0d3bd6997be6896aedd81211e161c89fad3
9e63567f7e670ea358af584eea6bbc9a68513c9d
b9d2816405979fc528a4eae71b8955242ef251af
5a0f8d0607e20aea0157a5572039ff255c0ae88b
5d1aa62b6c67b5678a3697153afbcdf45932f4af
```

6278fa0cf6c5c1f98f242702ea95ce38a40b79cf
d282272b1eec3aab3956e690e56582792109e822
1c62dfc839e694fd6517dfd736ee8d312ca0ff21
98b540cfb43f1cbfc9ab558bbf55ca8806942d87
2a1e03b568b4e86f36083adf249966aaca610550
3dc474b7f4779e4dce565d7c863e0a01fd17a059
0296faedf44c3c376794b09fa59ead1512712a68
70a010d97e9d4cc64aff0daeefcd2ca44d22b7f4
e266f3ca2ad74a3398d3c0996da1c11c631554cb
baf322bc7a837cd37b0cd132221b8ef2cbfda4f3
bfd425f2af8ab662cc28ad53ca0bd5f0e44f0600
cae869da63ceb8997e140f14fa2ffccec55ac8a6
dbbe855f86059b19bb91b8c78dd2770c9565b733
9454754c054fa94715121062a553d9aac3331065
56534152c27b991b2bf54635027d11cd8287d227
a8415ff9d6b50c8f64034bc433e052a791f61e09
bfd374c09e13fa6623de0b1e00e13b6a1994a7fb
3b4fdb372b5a038b6a3ba71bb123bd651c9b6ea4
b0ce3a6bdc4289302dd2cf781844b5e01c37ddc8
bea2e3a72556c7bf1015f78a77650b03a0953430
7eeff5f162b3280194d630d8dd60aad8ef4d151c
a1bfd79b3ee83a94af1933183b10b849bc047a62
da729079f51ce6082a9e9b370cb651fa6c565e77
d09935c31dd62dfc33e9685719f61b1ed24ecfbc
b45760a4c03f64ee16de794f8ddd601fb3afafcc
0ca0bbafdc2d23a77d3f33aa808bd0a6a4abc551
5f725fcbd9246f15d51dc777d59734654142bcde
fc6a9e873c5f8fd3f2d56eb46143aa92facb987b
be5ba8c384080f251bf204cc4257b98abca5beb7
0b2319d0ee4e231228012df7be2d2a0b4fd23383
eea645610fd4b3d95c543ea69eed84d87a0faafc
98270eb99d3f5d4c49ab447ecffa64da82c38d1a
5a63cdbfd9feed62c7cecc6f075bd01fd4a134b2
b280647aa1d942559d81ba56be33dc3853fcdffc
533711e615d90b01278493843310 8f7cf962cef7
a427ca3e1dfd683fb4a56eb9958cf893fa9e7512
d5c27609dc21cafc1b44b4692f84aa27c9e2aff6
1fc3ded2cbf7e181f1dec0d75ee077d54274c4ff
e11dea2894fb28a10db837b021c20eff9c6d7539
51dbbcc4e6f9c3815dc2758decb0750b6db51558
02d6ac68de883c7fdc25b471ba84859ef826571c
1b275bfe61d0b4ddf7c1366e168d5065a82f9a76
8e5304a17c91b0ded1116f675de68f2ddc4bbdf4
8e646769b515c011a0a746ee61f11e68f1f0445c
dc37278e3122ef7e721445d94b50024c847f0183
d2976d17c69000b1a81b38550e96a209e1c3a917
34c6d6646f0e80e31764c523695f00c872dac3ea
d737d7eb1aed496bfd76b552114d27921c92f1dd
45ab3734abfca90006278995270b996a2be9ef5e
fdddfae458e513ab5457340451293fc55c932fdc
274b913133345ec718591efd83526b3d7d265f6d
46672c46a535d788de8016a93f40a8ecaa1b98cb
2662e117dd69e9614f171fa2f17a214875638585
2154a52282b357ccd82837130dc4196bb40b3a23
1445aa3ca578aff92b2939ddd2df4b99b9348687
95debaaf3488667a2c89def803b3cfd6fb79a66d
9ab0a141a3f5ecba411ec4e32b2305834c4615ee
3e7b844c1a765a4e81fe22e1fabbaa96174181ca
7fd7ce92968eb86a6bfa6bfbeacf1815ac1159e9
4d871c0ef41349a63b435779871d4bf14fa513c5
199e506e646ae49f4cb723b1d9d39d555008124b
119c4a83b4263364e924967c2704c63ef420dc3e
f3a9bd2d22414628bdd9279c62f424e6f2b2dc73
3db7f1d52970791bb52baa8563447590b811e1e2
4a171325f32e659db92e894df7dc8e53c097158f
61da7e22c378da8339bee44514a77419e3f0b7e0
f0b5db87fe8af473a51d83a3f355224525ec00b8
a1b20819fc93ab2d15762a283f5a27d0758c4ec6
90f700f8e31d92d44dca00ca7114e4344df168fc
2d7d1f4f5deb93d03a02dc3ee3ccec7f8932b40a
581dfb3449bc31ffa87252a51c72447bc6db427e
537f9b03bb1b4d7c2ef940fbaab5bf4bf0403668
b35d2338697864d0d0b5127111ef90ef71362398
da0ada6c49cb4e4bf556e8d5f16cb836de171c5b
38281afadd91013b3ace2ebe796555f41f843915
bae9b4770a4b146a1216ad6147b3f3fed1e1de11
737b3613910c7db0b4a3e245f459163cfe89ba9a
71779adf9e82d9eb3ef5933f9ad1bdcc35ffa299
2d610af2b55cf5392575027bdd0b012dcba18449
e642a07e26492026a26470c5322e6ce75548dbc7
a132a710972f3cb4a501eaedc952d2453225ab92
fc3d9518b01e9e6f98c9d7f2d281c76601b3c981
e1bc68243abf498e68db74f5a5dd9ac73da1da41
f0cbbddcc89471c024af63f6a8d9117c4f864312
b5251072d5bc267801c31c62a1bf6176a1bde738
baffc4e73da782d961734d11cc0ecdb3046b5101

09fa327389241b6174a4f77d445df98d791d7d97
b11b3eeb75a5352f2e99a87e11a113fc284e2ea9
92b77fb0c1292f2a23189cb43ef885d2d59bc51d
d7d4b21f5fb15a2ff2e17319611f5d99455841d2
2d5eab261879563742a7f748081649de57139bbb
fb68a62e671504f8b075a1c0015a89ef11ef02c1
8e6db1b0cc5fc00099e8f53769bb5504d46d09b3
3bc08cee63773c7fd14488c0e2d8861580c182a9
973d73363aafec06846c96e43b278bfc5dffab3b
4f4ba789965da2915ffbd3431c2c92e1e1a39127
4e224f4ec06c9aef7cb2aa2650a9bb2c90b1a1d3
0fb75afbda64d0f924b2da65eeb13a11162f9694
c001d30c0d8c58925a37fdcbe0166425d7989939
4af5c4031dc8b0d1809e9b5ed0d81fe20eaeaf4c
26de1e314e6090f35f2fc167b5eb35be82078a99
3c667347c4a4bf3e568356ff3236727ab1205e1b
4bb4494d2a58e15a5a537f55840136e85ac985fc
a5add423db3629cae38a246b06ea5889f1a78da2
ea144f019eda5a1d83b9d247ddd80747e69c2702
a3551e1f4b53d4f97080f7f442954a253cc57a53
29f9bab64c3667f2fc87f2b7554168157b7d7260
9492bfa56f21829ba929b4b8fea3ac74831ed4be
5a5cc088238c91b03e65d3d39383adbb4cf65b8c
f070e4efc5ca80d956d2cab6298940a795b5a621
8375020321b46d02774c02b8e19c8ef46de60d4c
10eab9950e39b85b201c4d4bf5ccda5c82c99df1
eeb8fb7f2c7b5ca510c791d883a791ed431646c3
9d2dfa28ff352705cb525281ce18585bbf48a207
da594340da5de25b11187565c4aed6266150466c
c9ba9b7f8dbd90dc08e14539dee59817500108d9
8399d8e11f5af9bfa3965ffc0d1dabb0b27eefd6
564564027b395ab155a9d0e5c4d9728fc95c68b5
4ae5083464b3baf0214cd27a7c147edeb830f36a
96dafffc9fab010c5bcd7f6f375799a6b53fbade
b361aa83de65bff5f55de1e0f564662d3a53a11b
93411047fbc6404ff4e9e469f72be8c4fcbcd8b5
a5e177cecd2afd4b055bb0fcf8527dca0fb160da
f550f3901a2a3c0345f288164596a94e6a5c5cfd
f3a680de2e4226995cbadba109be04bb495a2b73
8b12011f3ba8bd36315b9980e50481ce724c7c90
f353c88ea2eaac2157db38351fe25dcd30fc1607
5bf446b78c71e609ad9350e0925ec1d36a51b128
60739397b97527770064fb2998782cf0c388a964
ae55e5701e5560028c525d4c57211a0b824cdba5
598593c6c88006e4d7379f5cb5a805ababbe9f48
92846e00539fe9654e54cc0ca28ac7b96dc3a2f5
622ec1444499a3f76cb1baf26cae53cc6b0c5657
d0ec9e9404f9dc59bb136c5311cf94bc51b52700
c17faff79b8e6f80afb114969af2ffbe50b88e07
83e019501ee62cdaae04bab82f60e0945612233a
cddfdc1ef0bceb5b9d9bcb34308f58293e32312e
4ef95e9e8d5dab734eeeea238e7800ca82e3e59d
35c03de7a669391f68f4c42f323bb42c82cf815e
2c16ca05dabae9422bbf89a21d07517527610293
f90ae87f035346ecff239915f3623966553200b4
eb971068c496d8cc29400e718be156df7df965fc
a0aeb67dec68d7eb310f93b5ff9cab1d7fb39d0d
df95ff6b2f359ed1860299fa9301e6c4fe32a06e
52d70a9b81236b7eb5d4380452366a1d2de9a891
0e9e5c3e292228959a7253f2ff52b82417e3ff0f
15f73933637f7c2d67163b980261b88f93ad42cf
008f3585aa8e3719a8ed7273f4ba5648f4c4f31d
4312f491e117d1fb5ae0945fedaca3e56ce65cde
663965a84ede49e1970d7845b7bf5bceae5dbac2
2e425ed8b5c8322670fe385dcb1680694bee0409
91b98df50e7d646a137208f6883af942b88b7ce2
2da48a7180f939c357ef26f180bebdddb28c11c0
bfa3f622796397c03bf7583d9bfc684cf73231e8
fc727dd4035f8a19a73843ec9522bc8cd57d7888
cc94e961753baffbae344c20c42bf04168db3b6e
17980530ec77921ff2492df0d6057375feff7751
fa0ddb6cb2172ed86ff145a87ab44e1c06c4555d
322812970a13dfa07fb9779804c604a705a22f69
78f92b825337b9660d6067b3fc4fd31454eb39f2
e7915be0a297c14a72c7e30ff0cd231311a5967f
39a275a7e4707223b3702dbe4402b2fc0ebd6692
66867d86b08a07b8395c5715093dfbdb015b1b01
79682ed288592a8ff15aa5f9daede791296dec95
304fbc3c16db599cedec6bbbb083daa6adc9b91d
36a47234b5be77efb887029967b344758a7647a9

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.

Bitdefender

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-DT-Whitepaper-IPStorm-WebUseOnly-creat4861-en_EN
07/10/20October 15, 2020 1:08 pm10/15/20