


FritzFrog: A New Generation Of Peer-To-Peer Botnets

 guardicore.com/2020/08/fritzfrog-p2p-botnet-infects-ssh-servers/

By Ophir Harpaz



Executive Summary

- Guardicore has discovered **FritzFrog**, a sophisticated peer-to-peer (P2P) botnet which has been actively breaching SSH servers since January 2020.
- **Golang-Based Malware:** FritzFrog executes a worm malware which is written in Golang, and is modular, multi-threaded and fileless, leaving no trace on the infected machine's disk.
- **Actively Targeting Government, Education, Finance and more:** FritzFrog has attempted to brute force and propagate to tens of millions of IP addresses of governmental offices, educational institutions, medical centers, banks and numerous telecom companies. Among those, it has successfully breached more than 500 servers, infecting well-known universities in the U.S. and Europe, and a railway company.
- **Sophistication:** FritzFrog is completely proprietary; its P2P implementation was written from scratch, teaching us that the attackers are highly professional software developers.
- **Interception:** Guardicore Labs has developed a client program in Golang which is capable of intercepting FritzFrog's P2P communication, as well as joining as a network peer.

- **Attribution:** While we are unable to attribute the FritzFrog botnet to a specific group, we have found some resemblance to a previously-seen P2P botnet named Rakos.

Introduction

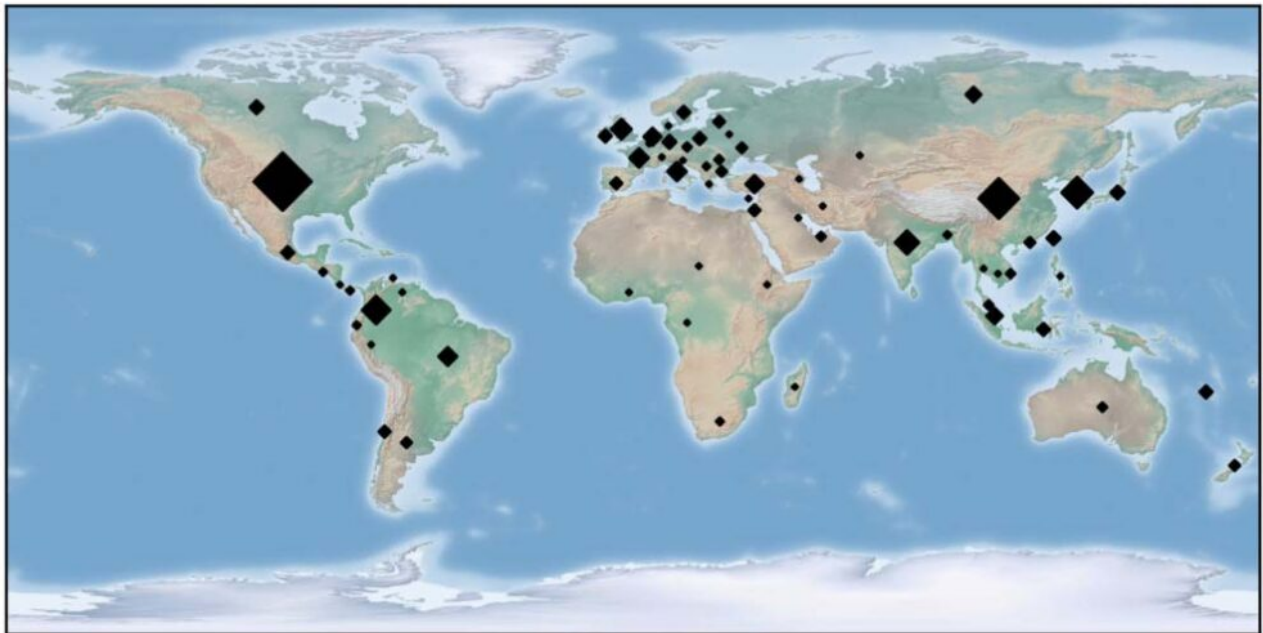
FritzFrog is a highly sophisticated peer-to-peer (P2P) botnet that has been actively breaching SSH servers worldwide. With its decentralized infrastructure, it distributes control among all its nodes. In this network with no single point-of-failure, peers constantly communicate with each other to keep the network alive, resilient and up-to-date. P2P communication is done over an encrypted channel, using AES for symmetric encryption and the Diffie-Hellman protocol for key exchange.

Unlike other P2P botnets, FritzFrog combines a set of properties that makes it unique: it is fileless, as it assembles and executes payloads in-memory. It is more aggressive in its brute-force attempts, yet stays efficient by distributing targets evenly within the network. Finally, FritzFrog's P2P protocol is proprietary and is not based on any existing implementation.

The malware, which is written in Golang, is completely volatile and leaves no traces on the disk. It creates a backdoor in the form of an SSH public key, enabling the attackers ongoing access to victim machines. Since the beginning of the campaign, we identified 20 different versions of the malware executable.

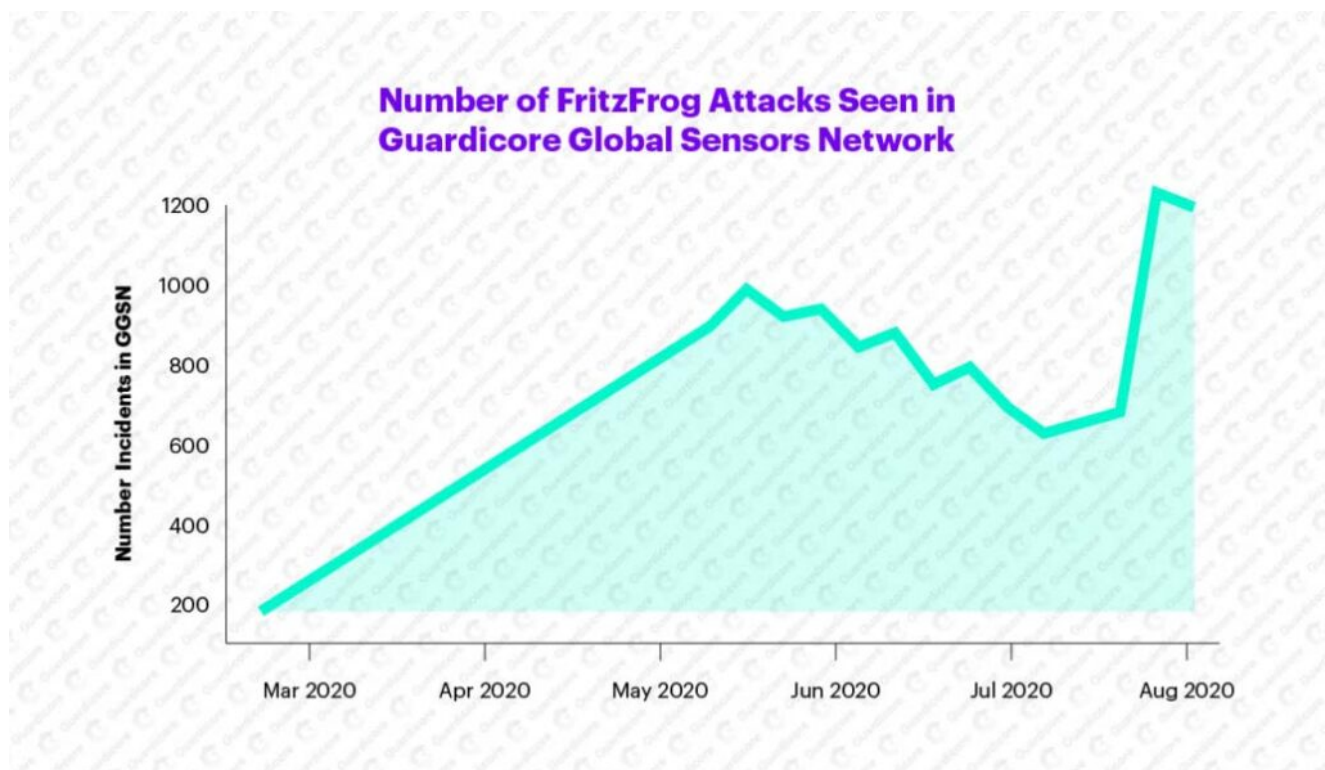
In this report, we will describe how the FritzFrog campaign was discovered, the nature of its P2P network and the malware's inner workings – including the infection process, command encryption and volatile behavior.

Guardicore Labs provides a [Github repository](#) containing a detection script as well as a list of Indicators of Compromise (IOCs) for this campaign.



Exploring FritzFrog

Guardicore Labs first noticed this campaign as part of its ongoing [Botnet Encyclopedia](#) research. On January 9, new attack incidents appeared where malicious processes named ifconfig and nginx were executed. We started monitoring the campaign's activity, which rose steadily and significantly with time, reaching an overall of 13k attacks on Guardicore Global Sensors Network (GGSN). Since its first appearance, we identified 20 different versions of the Fritzfrog binary.



What was intriguing about this campaign was that, at first sight, there was no apparent command and control (CNC) server being connected to. It was shortly after the beginning of the research when we understood no such CNC existed in the first place.

To intercept the FritzFrog network, Guardicore Labs has developed a client program in Golang, which performs the key-exchange process with the malware and is capable of sending commands and receiving their outputs. This program, which we named frogger, allowed us to investigate the nature and scope of the network. Using frogger, we were also able to join the network by “injecting” our own nodes and participating in the ongoing P2P traffic.

FritzFrog was found to brute-force millions of IP addresses, among which are governmental offices, educational institutions, medical centers, banks and numerous telecom companies. It has successfully breached over 500 SSH servers, including those of known high-education institutions in the U.S. and Europe, and a railway company.

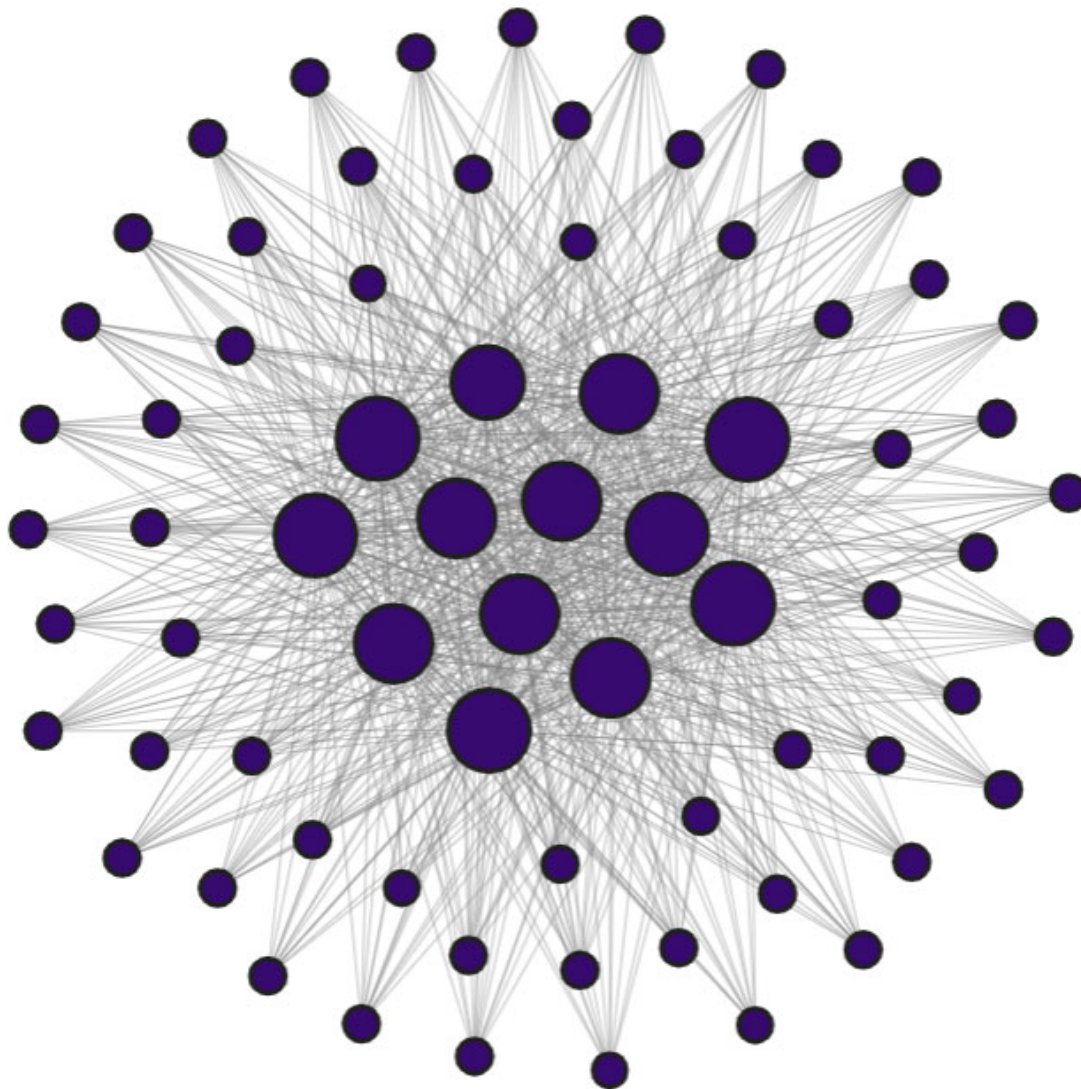
New Generation P2P

Why “New-Generation”?

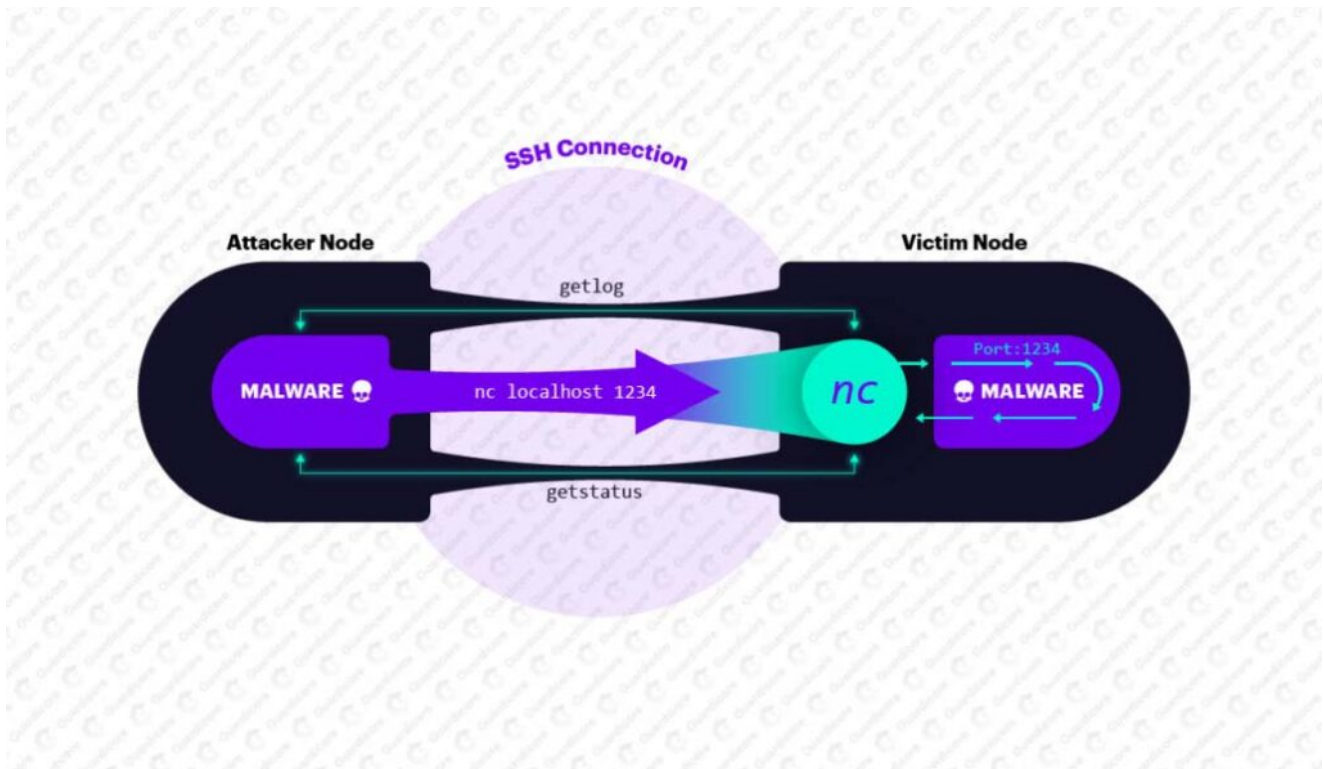
FritzFrog has a special combination of properties, which makes it unique in the threat landscape:

- *Fileless* – FritzFrog operates with no working directory, and file transfers are done in-memory using blobs.
- *Constantly updating* – databases of targets and breached machines are exchanged seamlessly.
- *Aggressive* – Brute-force is based on an extensive dictionary. By comparison, DDG, a recently discovered P2P botnet, used only the username “root”.
- *Efficient* – Targets are evenly distributed among nodes.
- *Proprietary*– The P2P protocol is completely proprietary, relying on no known P2P protocols such as µTP.

Once a victim is successfully breached, it starts running the UPX-packed malware, which immediately erases itself. The malware process runs under the names ifconfig and nginx, to minimize suspicion. As part of its startup process, the malware begins listening on port 1234, waiting for commands. The first commands which a new victim receives are responsible for syncing the victim with the database of network peers and brute-force targets.



Traffic on a non-standard port, such as 1234, can be easily detected and blocked by firewalls and other security products. Thus, FritzFrog's author employed a creative technique to evade detection and stay under the radar. Instead of sending commands directly over port 1234, commands are sent to the victim in the following manner: The attacker connects to the victim over SSH and runs a netcat client on the victim's machine, which in turn connects to the malware's server. From this point on, any command sent over SSH will be used as netcat's input, thus transmitted to the malware.



The FritzFrog attackers implemented an encrypted command channel with over 30 different commands. Command parameters and responses are transferred in designated data structures and serialized (“marshalled”) to JSON format. Prior to sending, the data is encrypted using AES symmetric encryption and encoded in Base64. To agree upon the encryption key, the involved nodes use the Diffie-Hellman key exchange protocol.

FritzFrog’s P2P Commands

Database Operations	Payloads Operations	Administration Operations	
getdb pushdb pushdbzip getdbzip getdbnotargetsgettargets pushtargets forcetargets puttarggetpoolputblentry	resetowned pushowned putowned getownedgetdeploy putdeploying deploystatus	runscript pushbin getbin sharefiles	getstatus getstats getblobstats getpeerstats getvotestats mapblobs getlog pushlog getargs proxy ping comm exit

Nodes in the FritzFrog network keep in close contact with each other. They constantly ping each other to verify connectivity, exchange peers and targets and keep each other synced. The nodes participate in a clever vote-casting process, which appears to affect the distribution of brute-force targets across the network. Guardicore Labs observed that targets are evenly distributed, such that no two nodes in the network attempt to “crack” the same target machine.

Delving into the Malware

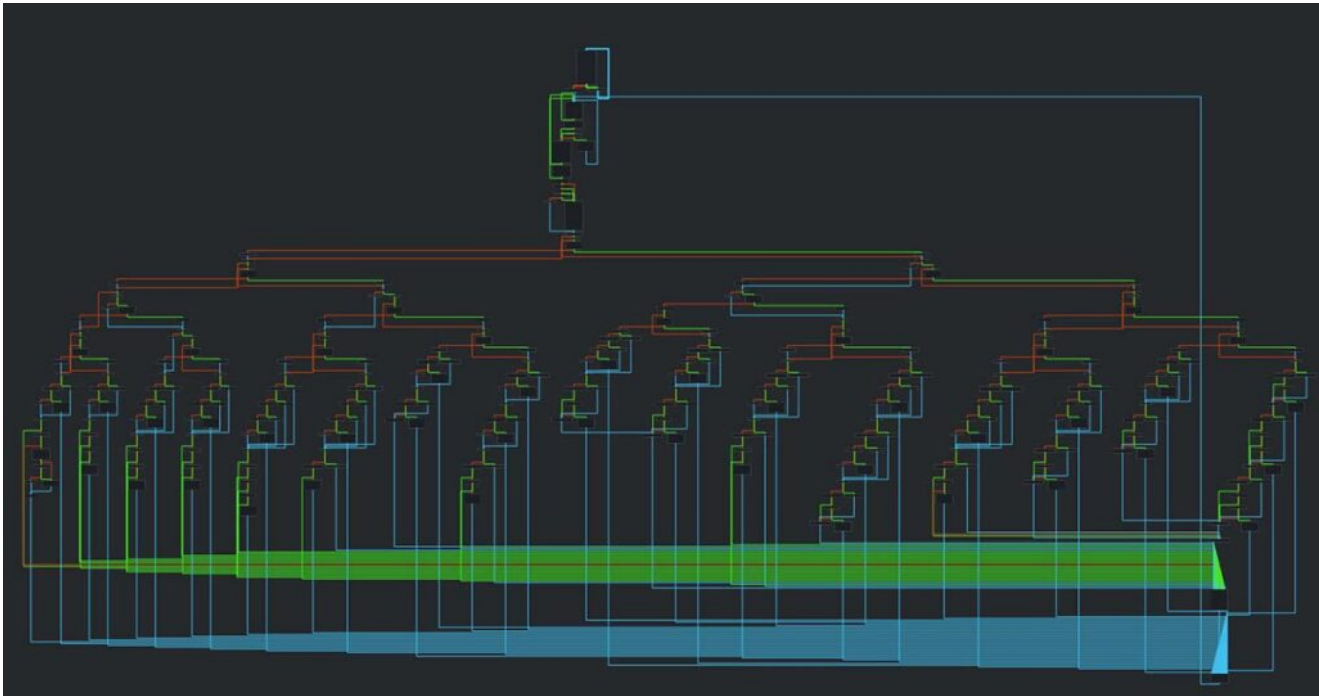
FritzFrog’s binary is an advanced piece of malware written in Golang. It operates completely in-memory; each node running the malware stores in its memory the whole database of targets and peers. The malware spawns multiple threads to perform various tasks simultaneously, as detailed in the table below.

FritzFrog defines the following states with regards to the management of victim and target machines.

1. **Target** – a machine found in the target queue will next be fed to the Cracker module, which in turn will scan and try to brute-force it;
2. **Deploy** – a machine which was successfully breached is queued for malware infection by the DeployMgmt module;
3. **Owned** – a machine which was successfully infected will be added to the P2P network by the Owned module.

MODULE NAME	FUNCTIONALITY
<i>Cracker</i>	Brute force targets
<i>CryptoComm + Parser</i>	Encrypted P2P communication
<i>CastVotes</i>	Voting mechanism for target distribution
<i>TargetFeed</i>	Ingesting targets from peers
<i>DeployMgmt</i>	Worm module, deploying malware on breached (“cracked”) machines
<i>Owned</i>	Connecting to victims after malware deployment
<i>Assemble</i>	In-memory file assembling from blobs
<i>Antivir</i>	Competitors elimination. Kills CPU-demanding processes with the string “xmr”
<i>Libexec</i>	Monero Cryptominer

Each node that runs the malware has a worker thread which is responsible for receiving commands, parsing them and dispatching them to the appropriate function in the code.



The malware is transient – it does attempt to survive system reboots. However, a backdoor is left to enable future access to the breached victim, whose login credentials are saved by the network peers. The malware adds a public SSH-RSA key to the `authorized_keys` file. This simple backdoor allows the attackers – who own the secret private key – for passwordless authentication, in case the original password was modified. FritzFrog has used only a single public key which is given in the box below.

```
ssh-rsa  
AAAAB3NzaC1yc2EAAAADAQABAAQDJYZIsncBTFc+iCRHXkeGfFA67j+kUVf7h/IL+sh0RXJn7yDN0vEXz7i
```

The malware file runs various shell commands on the local machine, some of them periodically, to monitor the system state. For example, it runs `free -m` to check for available RAM, `uptime`, `journalctl -S @0 -u sshd` to monitor SSH logins, and other commands which output the CPU usage statistics. These statistics are available for other nodes in the network to consume, and are used to determine, for example, whether to run a cryptominer or not.

The malware runs a separate process – named `libexec` – to mine the Monero coin. The miner is based on the popular XMRig miner and connects to the public pool `web.xmrpool.eu` over port 5555.

An Evil Torrent-Like Network

Fritzfrog relies on the ability to share files over the network, both to infect new machines and run malicious payloads, such as the Monero cryptominer.

To share and exchange files between nodes, Fritzfrog uses a stealthy, fileless approach. Files are split into blobs – bulks of binary data – which are kept in memory. The malware keeps track of the available blobs by storing them in a map together with each blob’s hash value.

When a node A wishes to receive a file from its peer, node B, it can query node B which blobs it owns using the command `getblobstats`. Then, node A can get a specific blob by its hash, either by the P2P command `getbin` or over HTTP, with the URL `https://:1234/`. When node A has all the needed blobs – it assembles the file using a special module named Assemble and runs it.

```
fritzfrog>frogger.exe -frog=[redacted] -command=getblobstats
2020/08/10 17:25:46 Connecting to [redacted]:1234
2020/08/10 17:25:46 Successfully connected to fritzfrog
2020/08/10 17:25:46 Successfully sent my public key
2020/08/10 17:25:50 Sending command getblobstats
Peer blob stats
=====
[462]java[770]386[770]amd64[770]arm[770]libexec[770]mips
[redacted]:22 super super ] Data age: 10s [ ] [*****] [*****] [*****] [ ] [*****]
[redacted]:22 root 1234 ] Data age: 1m36s [ ] [ ] [*****] [ ] [ ] [ ]
[redacted]:22 test test ] Data age: 27s [ ] [*****] [*****] [*****] [ ] [*****]
[redacted]:22 a 123456 ] Data age: 1m43s [ ] [*****] [*****] [*****] [****] [*****]
[redacted]:2222 root root123456 ] Data age: 49s [ ] [*****] [*****] [*****] [****] [*****]
[redacted]:22 tbox tbox ] Data age: 1m42s [ ] [ ] [*****] [ ] [ ] [ ]
[redacted]:22 uftp 123456 ] Data age: 52s [ ] [*****] [*****] [*****] [****] [*****]
[redacted]:22 admin admin ] Data age: 3m47s [ ] [*****] [*****] [*****] [****] [*****]
[redacted]:22 steam steam ] Data age: 40s [ ] [*****] [*****] [*****] [****] [*****]
end
```

Attribution

Tracking the operators of a P2P botnet is a complicated task; due to its distributed nature, commands can be sent to and from any node in the network. Still, we’ve attempted to compare it to previous P2P botnets seen in the threat landscape.

Even when compared with past P2P botnets, FritzFrog appears unique; it does not use IRC like *IRCflu*, it operates in-memory unlike DDG, and runs on Unix-based machines – as opposed to the *InterPlanetary Storm* botnet. If any, it bears some resemblance – especially with regards to function naming and version numbers – to *Rakos*, a P2P botnet written in Golang and analyzed by ESET back in 2016.

Detection & Mitigation

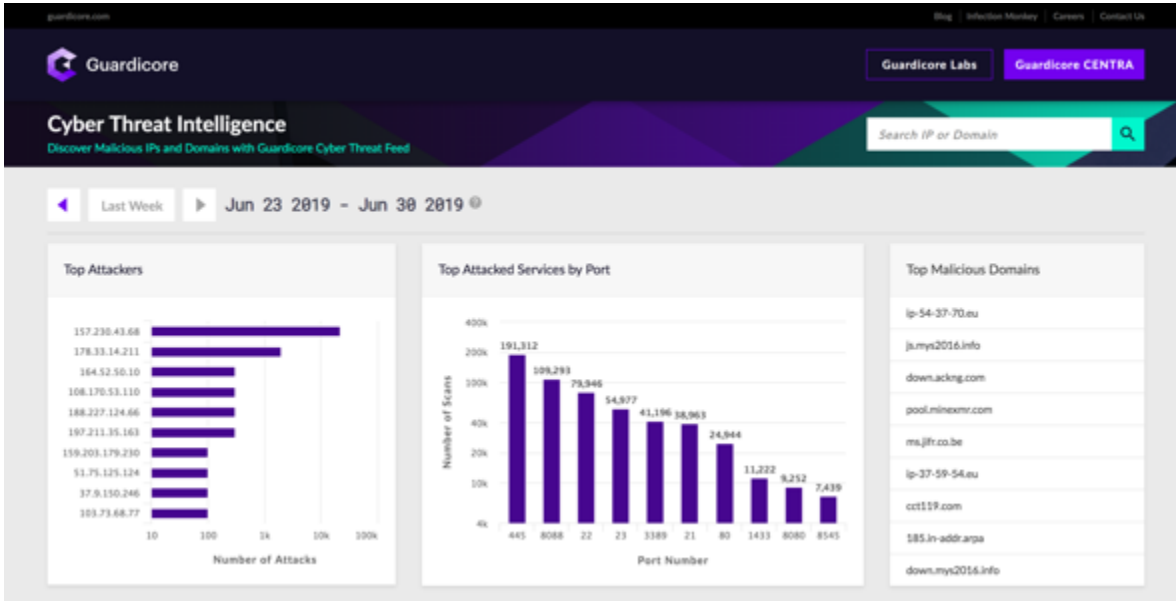
Guardicore Labs provides a [FritzFrog detection script](#) to run on SSH servers. It looks for the following FritzFrog indicators:

- Running processes `nginx`, `ifconfig` or `libexec` whose executable file no longer exists on the file system (as seen below)
- Listening port 1234

In addition, TCP traffic over port 5555 can indicate network traffic to the Monero pool.

```
ubuntu@ip-111-11-11-11:~$ ./detect_fritzfrog.sh
FritzFrog Detection Script by Guardicore Labs
=====
```

- [*] Fileless process nginx is running on the server.
- [*] Listening on port 1234
- [*] There is evidence of FritzFrog's malicious activity on this machine.



Learn More About Threat Intelligence Firewall

[VIEW NOW](#)

FritzFrog takes advantage of the fact that many network security solutions enforce traffic only by the port and protocol. To overcome this stealth technique, process-based segmentation rules can easily prevent such threats.

Weak passwords are the immediate enabler of FritzFrog's attacks. We recommend choosing strong passwords and using public key authentication, which is much safer. In addition, it is crucial to remove FritzFrog's public key from the *authorized_keys* file, preventing the attackers from accessing the machine. Routers and IoT devices often expose SSH and are thus vulnerable to FritzFrog; consider changing their SSH port or completely disabling SSH access to them if the service is not in use.