Malware Analysis Report: “FritzFrog”

CAP6137 Malware Reverse Engineering: P0x04

Naman Arora
naman.arora@ufl.edu

April 27, 2021

Contents

1 Executive Summary

The malware sample provided has been identified to belong to *FritzFrog* family of trojans/bot-nets. It uses *SSH* brute-force techniques to inject itself into the target machine. Once injected, it opens up a listening port on the machine (*Port 1234*) through which it enters into a *peer-to-peer* swarm of similar victim bot-nets. The communication between the individual bot-nets is encrypted and is routed through *SSH tunnel*. The bot-net uses an internal implementation of *Database* to store peers' information, targets' information as well as binary data it collects. A direct connection to a *bot-master* could not be identified and thus, it is a strong indication that all the commands as well as data ex-filtration is done via a *peer-to-peer* algorithm. On analysis, it shows that the malware sample has backdoor like capabilities like running commands, getting and pushing binary data, downloading binary programs, anti-detection etc.

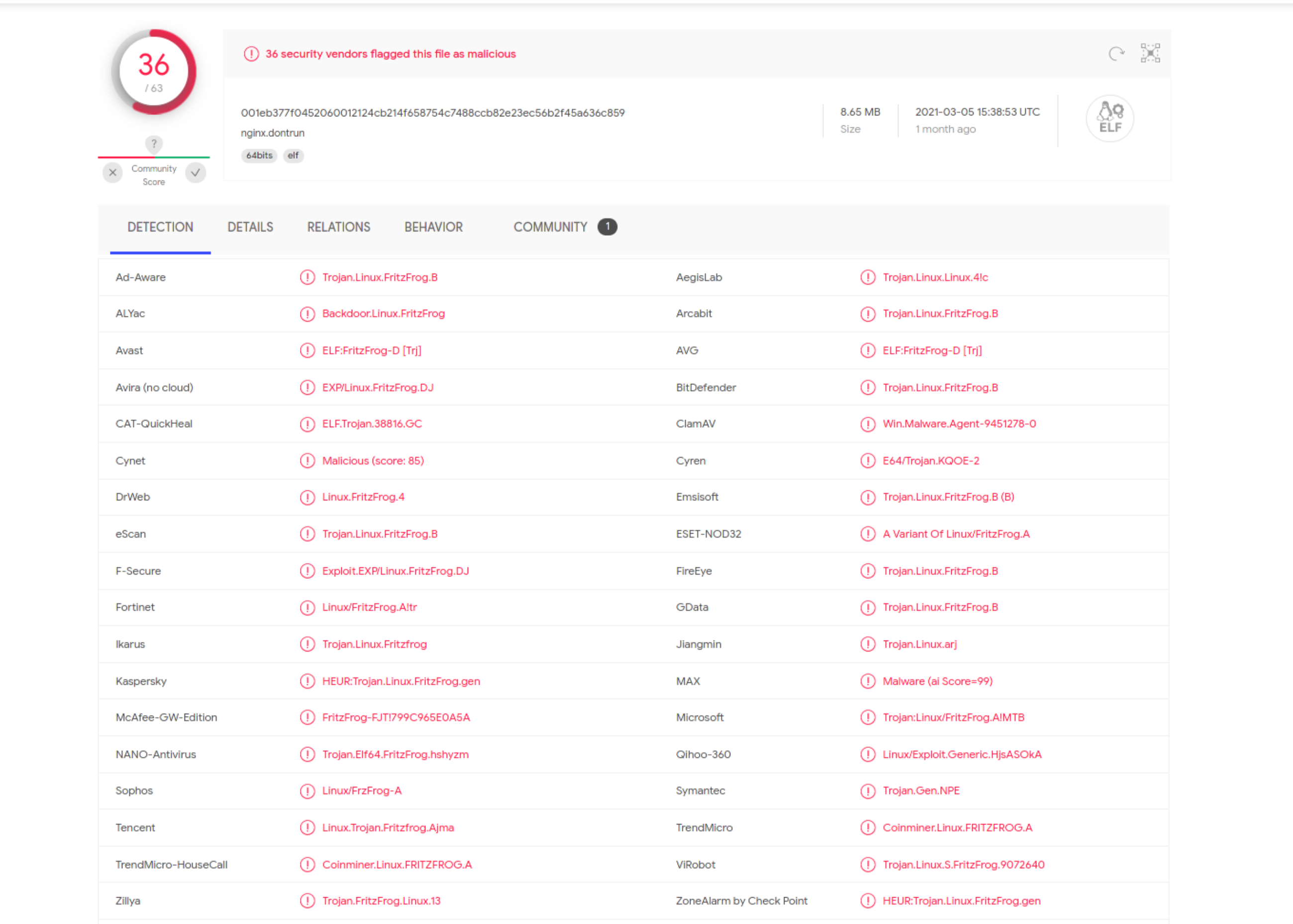
The malware is an example of early samples written in *Go programming language*. *Go* being a system agnostic language might mean the author might target other platforms too (*eg. Windows and Mac*) for extended reach. The malware shows lackluster obfuscation and it takes very little effort to gain access to metadata like function names etc. present within the binary itself. This lack of obfuscation might indicate an early attempt in writing malware in a new programming language on the part of the author.

2 Static Analysis

2.1 Basic Identification

Attribute	Value
Bits	64
Endianess	Little
Operating System	Linux
Class	ELF64
Subsystem	Linux
Size	9254304 Bytes
Compiler	Go
SHA256 Hash	001eb377f0452060012124cb214f658754c7488ccb82e23ec56b2f45a636c859

2.2 Malware Sample Family Identification



36 security vendors flagged this file as malicious

001eb377f0452060012124cb214f658754c7488ccb82e23ec56b2f45a636c859
nginx.dontrun
64bits elf

8.65 MB
Size

2021-03-05 15:38:53 UTC
1 month ago

Ad-Aware	Trojan.Linux.FritzFrog.B	AegisLab	Trojan.Linux.Linux.4!c
ALYac	Backdoor.Linux.FritzFrog	Arcabit	Trojan.Linux.FritzFrog.B
Avast	ELF:FritzFrog-D [Trj]	AVG	ELF:FritzFrog-D [Trj]
Avira (no cloud)	EXP/Linux.FritzFrog.DJ	BitDefender	Trojan.Linux.FritzFrog.B
CAT-QuickHeal	ELF.Trojan.38816.GC	ClamAV	Win.Malware.Agent-9451278-0
Cynet	Malicious (score: 85)	Cyren	E64/Trojan.KQOE-2
DrWeb	Linux.FritzFrog.4	Emsisoft	Trojan.Linux.FritzFrog.B (B)
eScan	Trojan.Linux.FritzFrog.B	ESET-NOD32	A Variant Of Linux/FritzFrog.A
F-Secure	Exploit.EXP/Linux.FritzFrog.DJ	FireEye	Trojan.Linux.FritzFrog.B
Fortinet	Linux/FritzFrog.A!tr	GData	Trojan.Linux.FritzFrog.B
Ikarus	Trojan.Linux.Fritzfrog	Jiangmin	Trojan.Linux.arj
Kaspersky	HEUR:Trojan.Linux.FritzFrog.gen	MAX	Malware (ai Score=99)
McAfee-GW-Edition	FritzFrog-FJT!799C965E0A5A	Microsoft	Trojan:Linux/FritzFrog.A!MTB
NANO-Antivirus	Trojan.Elf64.FritzFrog.hshyzm	Qihoo-360	Linux/Exploit.Generic.HjsASOkA
Sophos	Linux/FrzFrog-A	Symantec	Trojan.Gen.NPE
Tencent	Linux.Trojan.Fritzfrog.Ajma	TrendMicro	Coinminer.Linux.FRITZFROG.A
TrendMicro-HouseCall	Coinminer.Linux.FRITZFROG.A	ViRobot	Trojan.Linux.S.FritzFrog.9072640
Zillya	Trojan.FritzFrog.Linux.13	ZoneAlarm by Check Point	HEUR:Trojan.Linux.FritzFrog.gen

Figure 1: Virustotal: Family Identification

The malware executable can be identified by submitting to Virustotal [4] as belonging to *FritzFrog* malware family.

2.3 Section Headers

The sections within the binary have expected entropy values and do not show any significant deviations from the norm of any regular go program. One interesting thing, though, is the *.gopclntab* section [1]. This section contains contains mapping of individual functions with their line information from the original source files.

This feature has been available since *Go 1.2* and helps in getting author defined function names from the binary. Tools like *Redress* [3] help in gathering the metadata. Also, *Rizin* has been tested to perform similar metadata extraction during this analysis and a script for *Ghidra* performs similar action [2].

2.4 A case against Packing

The malware sample almost certainly shows no obfuscation techniques like packing or encryption. Not only the *Go Lang* standard library functions, but the author generated function names can be recovered using the *.gopclntab* section. Much of the *peer-to-peer* functionality is visible including *struct types* like *main.Database*, *main.DHGroup* and *main.CryptoComm*.

2.5 Interesting Imports

Some of the imports from the *Go* standard library as well as external packages are,

- *os/exec* which indicates towards command execution
- *crypto/ssh* which indicates towards *SSH* key exchange using *DiffiHellman*, communication over *SSH* channel etc.
- *encoding/json and encoding/base64* which indicate *JSON* as well as *Base64* data serialization.
- *net/http* which indicates some *HTTP* functionality.

2.6 Interesting Code Constructs

The following functions, established as user functions from *Go .gopclntab* section are interesting, (functions missing sub-points when their name represents exactly what they do)

2.6.1 type struct DHGroup

This is quite likely used to exchange keys within the peers Notable methods are:

1. main.*DHKeyExchange **@0x007ce0d0**
2. main.*DHGroup.ComputeKey **@0x007ee470**
3. main.*DHGroup.G **@0x007edcf0**
    - Generates the variable *G* in creating the keys
4. main.*DHGroup.GeneratePrivateKey **@0x007edea0**
5. main.*DHGroup.P **@0x007edb40**
    - Generates the variable *P* in generating the keys

2.6.2 type struct Database

This type most likely is used to store peer information, data fetched, blacklisted peers as well as peers currently being deployed Following are notable methods:

1. main.NewDatabase **@0x007d4290**
    - Creates a new instance of *Database* type, idiomatic Go.
2. main.*Database.AddBlEntry **@0x007d5890**
    - Inserts a new blacklist entry
    - Internally uses *main.*Database.internalAddBlEntry* **@0x007d5660**
3. main.*Database.AddDeploying **@0x007d500**
    - Possibly inserts a new entry which is currently being deployed and compromised for insertion into the swarm
    - Internally calls a *main.*Database.internalAddDeploying* **@0x007d6620**

4. main.*Database.AddOwned @**0x007d8800**
    - Possibly inerts the information about binary data/blobs into the database
    - Internally uses *main.*Database.internalAddOwned* @**0x007d7640**
5. main.*Database.AddTarget @**0x007d5970**
    - Adds a new target that might possibly convert to a deployed peer (?)
    - Internally uses *main.*Database.internalAddTarget* @**0x007d4f80**
6. main.*Database.AddTPEntry @**0x007d7000**
    - Adds a *target pool* which most likely consists a swarm of targets
    - Internally leverages *main.*Database.internalAddTPEntry* @**0x007d7130**
    - Interestingly, it does not leverage *main.*Database.internalAddTarget* in a loop indicating a deviation of *Target Pool* from a single target
7. main.*Database.GetBlacklist @**0x007d4ac0**
8. main.*Database.GetDeploying @**0x007d4850**
9. main.*Database.GetOwned @**0x007d45e0**
10. main.*Database.GetTargetPool @**0x007d4d10**
11. main.*Database.GetTargets @**0x007d4390**
12. main.*Database.IncreaseDeployFailCount @**0x007d6170**
13. main.*Database.IncreaseFailCount @**0x007d5f40**
14. main.*Database.IncreaseTryCount @**0x007d8940**
15. main.*Database.RemoveDeploying @**0x007d5e10**
16. main.*Database.RemoveOwned @**0x007d5cd0**
17. main.*Database.RemoveTarget @**0x007d5a60**
18. main.*Database.ResetDeployFailCount @**0x007d63c0**
19. main.*Database.ResetSuccFails @**0x007d90e0**

2.6.3 main.Worker

main.Worker is presumably a function that is run after key exchange is successful. If this is the case, then it is very likely it is run as a separate go routine. It basically is an infinite if-else loop with separate functions called as commands.

1. Peer Algorithm related
    1. main.ping @**0x007f1bf0**
        - Uses to send a ping to peer
        - Updates peer status in Database if read is successful
        - Uses *main.CryptComm.Read/Write* to send encrypted pings
    2. main.getpeerstats @**0x007faf90**
        - Possibly sends stats of all the peers, including the ones blacklisted and the ones that have sent blobs **to** the nbor
        - Uses *main.*Database.GetOwned*, *main.*Database.GetDeploying* as well as *main.*Database.GetBlacklist*
    3. main.getvotestats @**0x007fdf80**
        - Sends over *TargetPool* **to** the peer
        - Uses *main.*Database.GetTargetPool* and *main.*Database.GetOwned* internally
    4. main.communicate @**0x007f2150**
        - Possibly used to update a socket/communication method for a peer in Db
        - Has an evasion feature, returns regular errors if a check fails
    5. main.getstatus @**0x007f5200**
        - Possibly sends status of a particular peer **to** nbor
    6. main.putblentry @**0x007fea70**
        - Add a new *blacklist* entry to the database
        - internally uses *main.*Database.AddBLEntry*
    7. main.getdb @**0x007f4f20**
        - Pushes peer database **to** nbor
        - Uses *JSON* Encoding
    8. main.pushdb @**0x007f6670**
        - Fetches peer database **from** nbor
        - Uses *JSON* Encoding
    9. main.getdbzip @**0x007f6cb0**
        - Pushes peer database **to** nbor

- Uses GZIP format
- Uses *compress/gzip* in std library

10. main.pushdbzip @**0x007f67e0**
    - Gets peer database **from** nbor
    - Uses GZIP format
    - Uses *compress/gzip* in std library
11. main.getdbnotargets @**0x007f5090**
    - Possibly pushes blacklisted targets **to** nbor
    - Uses *JSON* encoding

2. Binary related
   1. main.getbin @**0x007f7ed0**
      - Pushs encrypted binary data **to** the nbor
   2. main.pushbin @**0x007f7910**
      - Gets encrypted binary data **from** the nbor
   3. main.sharefiles @**0x007f8b60**
      - Sends a requested file **to** the nbor
      - Possibly has certain evasion features, can send wrong error message if some condition is not met
   4. main.mapblobs @**0x007f8340**
      - Might send blobs and related peer info **to** nbor
   5. main.getblobstats @**0x007fcf80**
      - Send statistics of owned blob **to** nbor
      - Internally uses *main.*Database.GetOwned*
   6. main.getowned @**0x007f4af0**
      - Sends encrpted list of targets owned **to** the nbor
      - Gets all the owned peers from the database using *main.*Database.GetOwned* internally
   7. main.putowned @**0x007f3340**
      - Gets *owned* target **from** nbor
      - adds to Db using *main.*Database.AddOwned*
   8. main.pushowned @**0x007f3a10**
      - Gets all the owned assets **from** the nbor
      - Uses *main.*Database.AddOwned* in a loop internally
   9. main.resetowned @**0x007f4090**
      - Probably resets the attributes of an owned asset
      - Uses *main.*Database.RemoveOwned* before *main.*Database.AddOwned*
      - Removes and re-inserts a target
   10. main.getstats @**0x007f9110**
       - Sends some kind of stats about the *owned* blobs *to* the nbor

3. Target Related
   1. main.gettargets @**0x007f46f0**
      - Iterate over target map and return targets **to** the peer
      - Uses *JSON* encoding
   2. main.puttargets @**0x007f2450**
      - Get targets **from** nbro
      - Internally uses *main.*Database.AddTarget* in a loop
   3. main.pushtargets @**0x007f2880**
      - Receive a list of targets in *JSON* format **from** a nbor
      - Uses *main.*Database.AddTarget* in loop
   4. main.puttargetpool @**0x007f2e10**
      - Adds a whole *targetpool* to database as received *from* nbor
      - Uses *JSON* encoding
      - Uses *main.*Database.AddTPEntry* internally
   5. main.forcetargets @**0x007f2b90**
      - Internally uses *main.*Database.ForceTargets*
   6. main.deploystatus @**0x007f5b90**
      - Most likely used to get status of deployed peers
   7. main.putdeploying @**0x007f36a0**
      - Gets the target info and adds to its database *from* the nbor
      - Uses *JSON* encoding
      - Uses *main.*Database.AddDeploying* internally.

   8. main.getdeploy **@0x007f5550**
      - Uses *main.*Database.SetDeploy* internally

4. Log related
   1. main.getlog **@0x007f1a80**
      - Uses to write encrypted log **to** the nbor
      - Internally uses main.GetLog
   2. main.pushlog **@0x007f9080**
      - Uses to get logs **from** the nbor
   3. main.Log **@0x007e07f0**
      - Possibly creates a new log entry with current time

5. Misc
   1. main.runscript **@0x007fecd0**
      - Runs a script using *os/Exec* module
      - Command is run using *os/Exec.Command* function
      - Uses *os/Exec.*Cmd.StdinPipe* and *os/Exec.*Cmd.StdoutPipe*
   2. main.comm_proxy **@0x007ff390**
      - Possibly used to create a new *CryptComm* connection to a peer
      - Uses *main.NewCryptoCommFromOwned* internally
   3. main.getargs **@0x007ff9b0**
      - Writes yet unknown data to nbor

3 Dynamic Analysis

3.1 Interesting Features

3.2 File System Interaction

3.3 Network Interaction

4 Indicators of Compromise

4.1 Host Based

4.2 *YARA* Rule

5 Appendix A: Screenshots