Module 1: Overview of CIS Benchmarks

1.1 Understanding the Challenge

If you keep your server open to the internet, you will observe a huge spike in hacking attempts.

```
Jul 26 05:44:42 ip-172-31-62-21 sshd[24735]: Disconnected from 195.54.160.183 port 34646 [preauth]
Jul 26 06:42:35 ip-172-31-62-21 sshd[29819]: Invalid user pi from 31.201.193.88 port 36834
Jul 26 06:42:35 ip-172-31-62-21 sshd[29819]: input_userauth_request: invalid user pi [preauth]
Jul 26 06:42:35 ip-172-31-62-21 sshd[29809]: Invalid user pi from 31.201.193.88 port 36824
Jul 26 06:42:35 ip-172-31-62-21 sshd[29809]: input_userauth_request: invalid user pi [preauth]
Jul 26 06:42:36 ip-172-31-62-21 sshd[29819]: Connection closed by 31.201.193.88 port 36834 [preauth]
Jul 26 06:42:36 ip-172-31-62-21 sshd[29809]: Connection closed by 31.201.193.88 port 36824 [preauth]
Jul 26 07:31:33 ip-172-31-62-21 sshd[1694]: Invalid user user from 195.54.160.180 port 33804
Jul 26 07:31:33 ip-172-31-62-21 sshd[1694]: input_userauth_request: invalid user user [preauth]
Jul 26 07:31:33 ip-172-31-62-21 sshd[1694]: Received disconnect from 195.54.160.180 port 33804:11: Client disconnect
ing normally [preauth]
Jul 26 07:31:33 ip-172-31-62-21 sshd[1694]: Disconnected from 195.54.160.180 port 33804 [preauth]
Jul 26 07:42:57 ip-172-31-62-21 sshd[2705]: Did not receive identification string from 198.98.49.181 port 50738
Jul 26 07:51:34 ip-172-31-62-21 sshd[3466]: Received disconnect from 61.230.7.137 port 42662:11: Bye Bye [preauth]
Jul 26 07:51:34 ip-172-31-62-21 sshd[3466]: Disconnected from 61.230.7.137 port 42662 [preauth]
Jul 26 07:51:35 ip-172-31-62-21 sshd[3468]: Invalid user cirros from 61.230.7.137 port 42800
Jul 26 07:51:35 ip-172-31-62-21 sshd[3468]: input_userauth_request: invalid user cirros [preauth]
Jul 26 07:51:36 ip-172-31-62-21 sshd[3468]: Received disconnect from 61.230.7.137 port 42800:11: Bye Bye [preauth]
Jul 26 07:51:36 ip-172-31-62-21 sshd[3468]: Disconnected from 61.230.7.137 port 42800 [preauth]
```

1.2 Monitoring is the Second Step

Security Monitoring is always the second step.

The first step is to harden your infrastructure.

1.3 Challenge with Organization Level Hardening Guidelines

Depending on the technology that you use and the organization, the hardening steps would differ.

Tools/Technology	Organization X	Organization Y
AWS	30 Hardening Steps	80 Hardening Steps
Linux Server	120 Hardening Steps	40 Hardening Steps
Mac OS	20 Hardening Steps	25 Hardening Steps

1.4 Importance of Standardization

The Center for Internet Security (CIS) provides a standardized set of security benchmarks to identify and refine effective security measures for a specific set of tools and technologies.

1.5 Example CIS Rules - AWS Benchmark

The organization has migrated its infrastructure to AWS.

They want to ensure that their AWS account follows the security best practices.

Rule	Rule	Checklist
1	Avoid the use of “root” account	Yes
2	Ensure CloudTrail log file validation is enabled	Yes
3	Ensure VPC flow logging is enabled in all VPCs	No

1.6 Industry is adopting CIS Strongly

Top Industry providers like AWS provide built-in controls that allow you to monitor whether AWS accounts are following CIS Benchmarks.

1.7 CIS =! Enough

Just by making use of CIS Benchmarks does not mean you are following all the best security practices for a system.

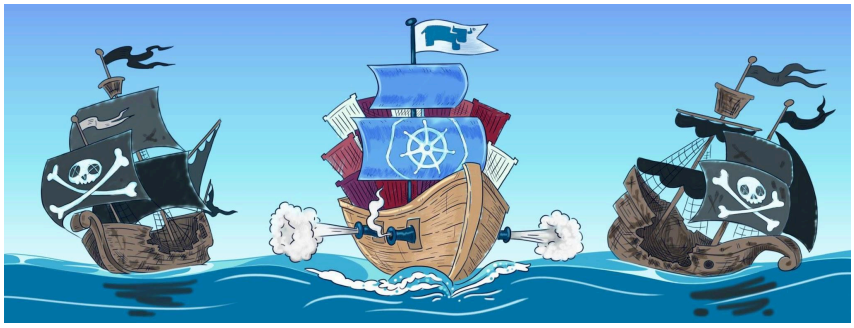
Many organizations make use of a combination of CIS Benchmarks + Custom Hardening Guidelines as part of the overall security.

Module 2: CIS Benchmarks for K8s

2.1 CIS for Kubernetes

The CIS Kubernetes Benchmark is a set of recommendations for configuring Kubernetes to support a strong security posture.

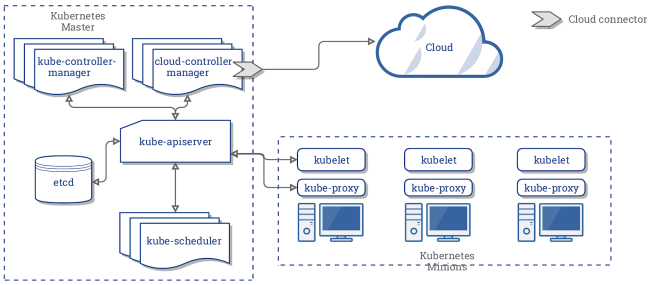
The Benchmark is tied to a specific Kubernetes release.



2.2 Kubernetes Architecture

There are two important components of a Kubernetes Cluster:

Kubernetes Master Node and Worker Nodes



2.3 Checks for K8s CIS

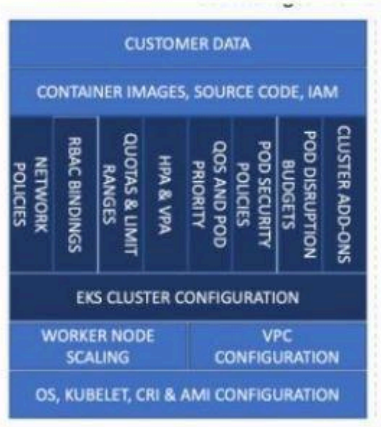
There are various tools like kube-bench that allows us to scan our K8s cluster based on the CIS Benchmark guidelines.

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
[FAIL] 1.1.9 Ensure that the --profiling argument is set to false (Scored)
[FAIL] 1.1.10 Ensure that the --repair-malformed-updates argument is set to false (Scored)
[PASS] 1.1.11 Ensure that the admission control policy is not set to AlwaysAdmit (Scored)
[FAIL] 1.1.12 Ensure that the admission control policy is set to AlwaysPullImages (Scored)
[FAIL] 1.1.13 Ensure that the admission control policy is set to DenyEscalatingExec (Scored)
[FAIL] 1.1.14 Ensure that the admission control policy is set to SecurityContextDeny (Scored)
[PASS] 1.1.15 Ensure that the admission control policy is set to NamespaceLifecycle (Scored)
[FAIL] 1.1.16 Ensure that the --audit-log-path argument is set as appropriate (Scored)
[FAIL] 1.1.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate (Scored)
[FAIL] 1.1.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as appropriate (Scored)
[FAIL] 1.1.19 Ensure that the --audit-log-maxsize argument is set to 100 or as appropriate (Scored)
[PASS] 1.1.20 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 1.1.21 Ensure that the --token-auth-file parameter is not set (Scored)
[FAIL] 1.1.22 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored)
```

2.4 Shared Responsibility Model

Many providers provide managed Kubernetes Cluster (AWS EKS, Digital Ocean, GCP)

Since customers do not have full control over the cluster, the responsibility is divided.

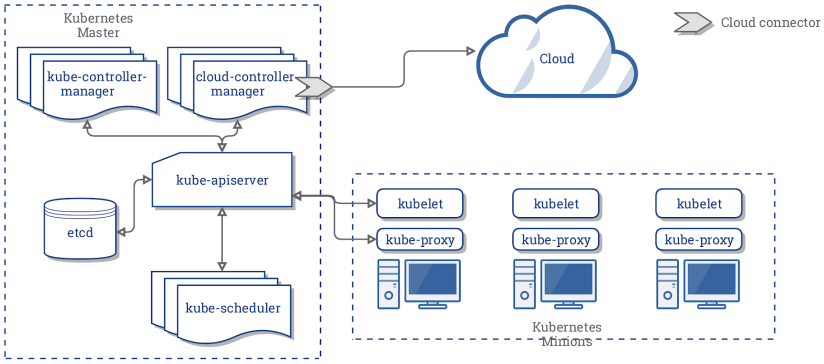


Module 3: CIS Benchmarks for K8s

3.1 Revising ETCD

etcd is a distributed reliable key-value store.

etcd reliably stores the configuration data of the Kubernetes cluster, representing the state of the cluster.



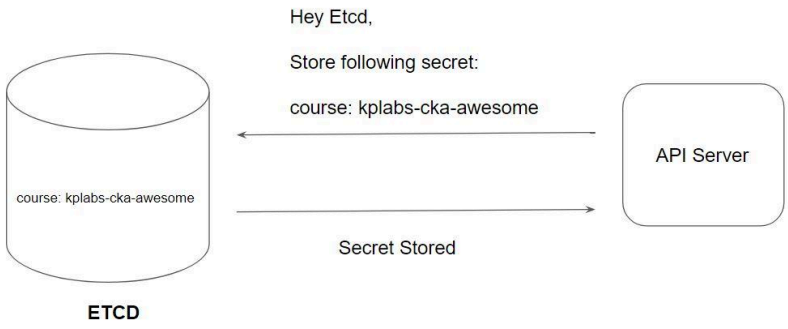
3.2 ETCD Security areas

There are three primary security areas that we will be focussing on:

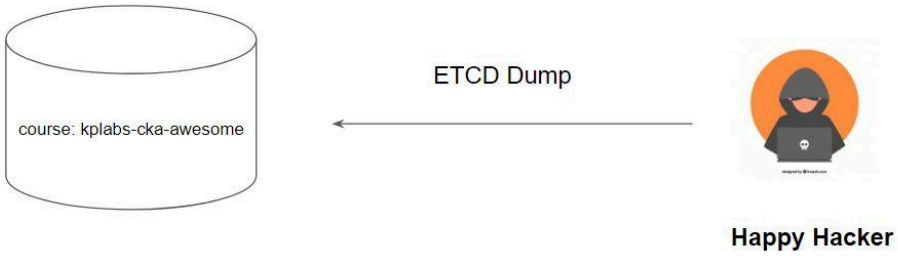
- Plain Text Data Storage
- Transport Security with HTTPS
- Authentication with HTTPS Certificates

3.3 Plain Text Data Storage

In this approach, the data is stored in ETCD in plain-text format



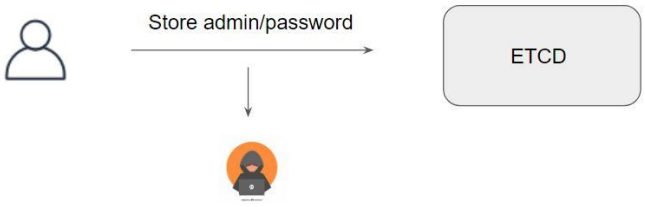
In such a scenario, a quick dump of the ETCD and all the data would be readable by an attacker or a malicious user.



3.4 In-Transit Encryption

The data that is stored in etcd is sensitive in nature.

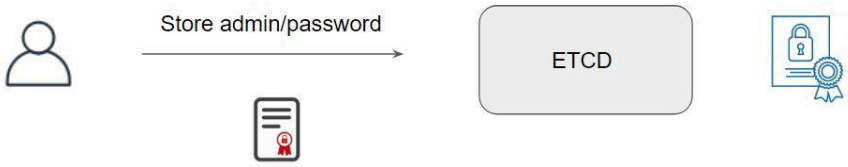
Along with the at-rest encryption, it is important to have in-transit encryption as well.



3.5 Client Authentication

In this approach, etcd will check all incoming HTTPS requests for a client certificate signed by the trusted CA, requests that don't supply a valid client certificate will fail

This option is enabled via --client-cert-auth and --trusted-ca-file flag.



k

Module 4: Revising SSL/TLS

4.1 Overview of HTTPS

HTTPS is an extension of HTTP.

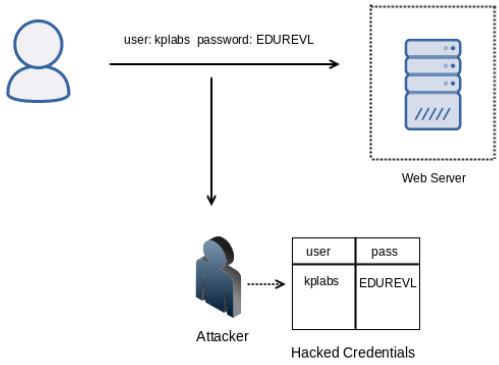
In HTTPS, the communication is encrypted using Transport Layer Security (TLS)

The protocol is therefore also often referred to as HTTP over TLS or HTTP over SSL.

4.2 Scenario 1: MITM Attacks

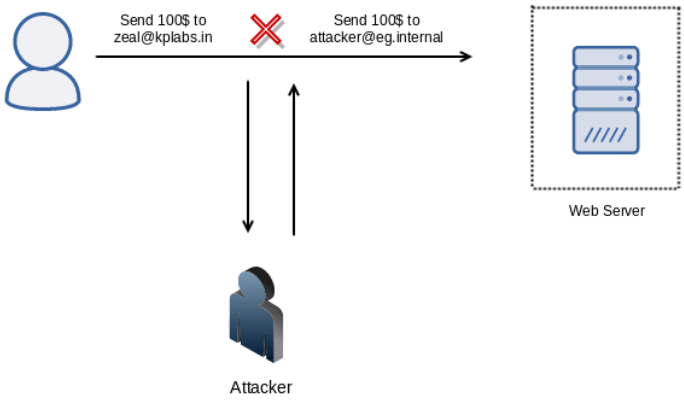
User is sending their username and password in plaintext to a Web Server for authentication over a network.

There is an Attacker sitting between them doing a MITM attack and storing all the credentials he finds over the network to a file:



4.3 Scenario 2: MITM & Integrity Attacks

Attacker changing the payment details while packets are in transit.



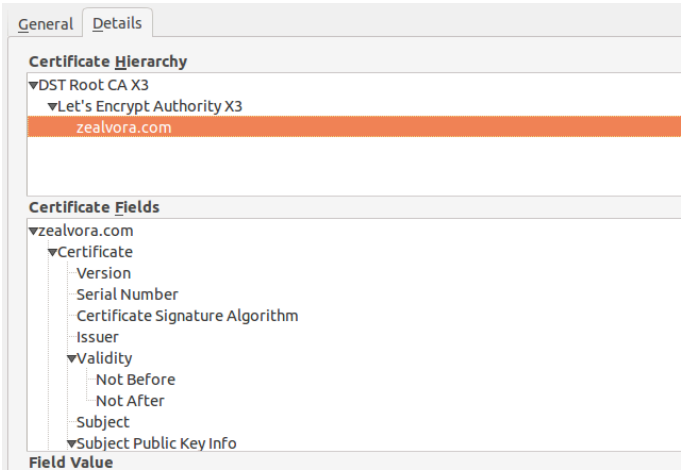
4.4 Introduction to SSL/TLS

To avoid the previous two scenarios (and many more), various cryptographic standards were clubbed together to establish secure communication over an untrusted network and they were known as SSL/TLS.

Protocol	Year
SSL 2.0	1995
SSL 3.0	1996
TLS 1.0	1999
TLS 1.1	2006
TLS 1.2	2008
TLS 1.3	2018

4.5 Understanding it in an easy way

Every website has a certificate (like a passport which is issued by a trusted entity).



The certificate has a lot of details like the domain name it is valid for, the public key, validity, and others.

Browser (clients) verifies if it trusts the certificate issuer.

It will verify all the details of the certificate.

It will take the public key and initiate a negotiation.

Asymmetric key encryption is used to generate a new temporary symmetric key which will be used for secure communication.

```
server {
       listen         80;
       server_name    zealvora.com;
       return         301 https://$server_name$request_uri;
}


server {
  server_name zealvora.com;
  listen 443 default ssl;
  server_name zealvora.com;
  ssl_certificate /etc/letsencrypt/archive/zealvora.com/fullchain1.pem;
  ssl_certificate_key /etc/letsencrypt/archive/zealvora.com/privkey1.pem;

  location / {
    root /websites/zealvora/;
    include location-php;
    index index.php;
 }
 location ~ /.well-known {
        allow all;
 }
}
```

.

Module 5: Certificate Authority

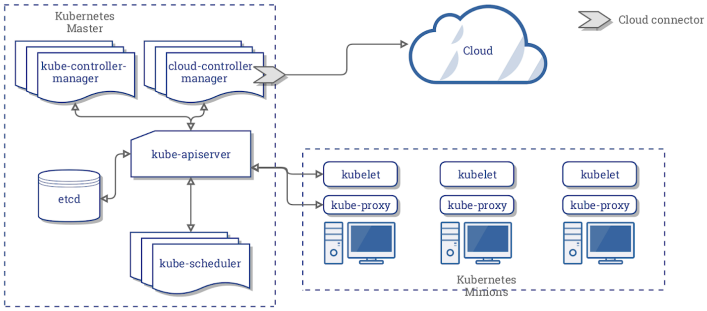
5.1 Importance of Secure Communication

There are multiple components that will be communicating which each other.

When communication is in plain-text, it is prone to a lot of attacks.

Hence it is important to have secure communication between multiple components.

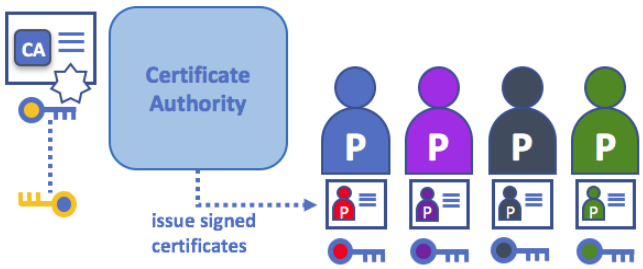
This can be achieved with the help of certificates.



5.2 Overview of Certificate Authority

Certificate Authority is an entity that issues digital certificates.

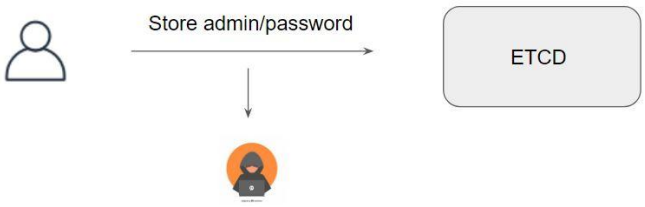
The key part is that both the receiver and the sender trusts the CA.



Module 6: In-Transit Encryption with HTTPS

The data that is stored in etcd is sensitive in nature.

Along with the at-rest encryption, it is important to have in-transit encryption as well.

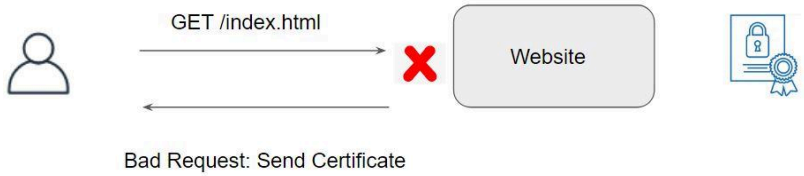


Here are some of the important configuration flags for in-transit encryption at ETCD level.

Argument Flags	Description
--cert-file=<path>	Certificate used for SSL/TLS connections to etcd.
--key-file=<path>	Key for the certificate. Must be unencrypted.
--advertise-client-urls	Addresses etcd server binds to for accepting incoming connections
--listen-client-urls	Addresses etcd clients or other etcd members should use to contact the etcd server

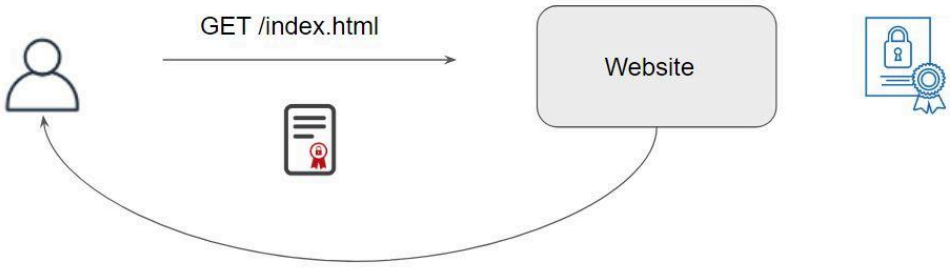
Module 7: Certificate-Based Authentication

Certificate-Based Authentication is used to identify a user/device before granting access to a specific resource like a website.



The following represents the basic authentication workflow.

While requesting, the client needs to also send the certificate that was assigned to him.



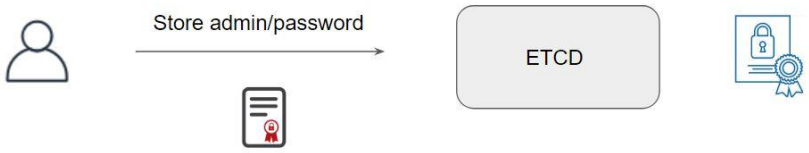
In this type of authentication, every user who needs to access the website will need to have a signed certificate.

Module 8: ETCD Security - Client Authentication

8.1 Overview of Client Authentication in ETCD

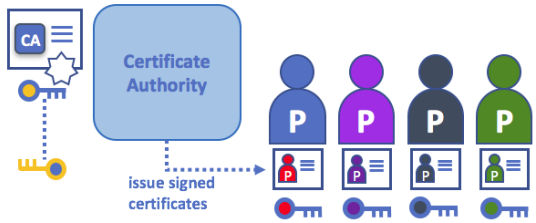
In this approach, etcd will check all incoming HTTPS requests for a client certificate signed by the trusted CA, requests that don't supply a valid client certificate will fail

This option is enabled via --client-cert-auth and --trusted-ca-file flag.



8.2 Certificate Authority Workflow

- Generate the Client CSR
- Sign the Client CSR with the CA to obtain the client certificate.
- Use the Client Certificate to connect to the ETCD

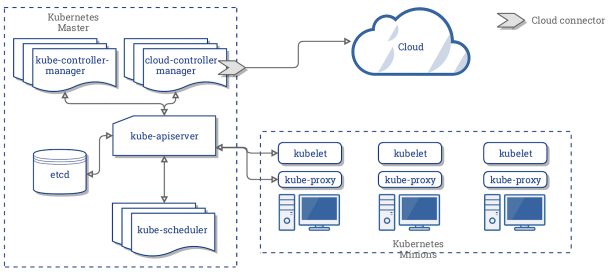


Module 9: API Server Security

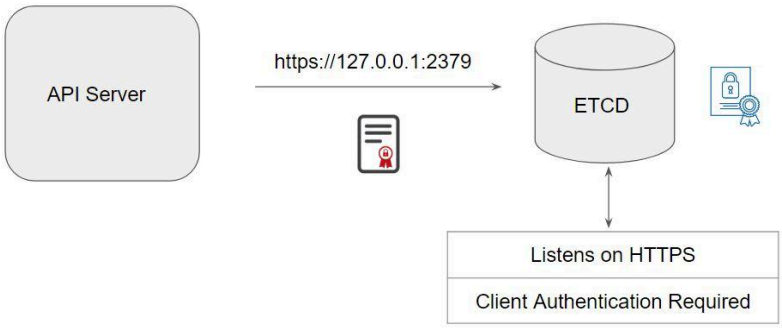
9.1 Revising API Server

API server acts as a gateway to the Kubernetes Cluster.

When you interact with your Kubernetes cluster using the kubectl command-line interface, you are actually communicating with the master API Server component.



9.2 Our Current ETCD Server Setup



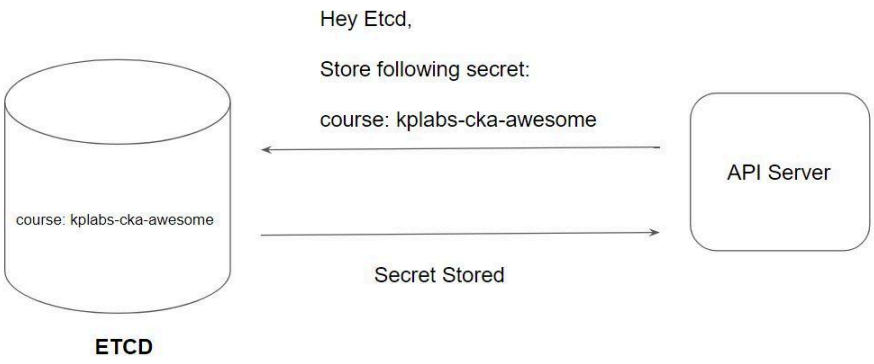
9.3 Necessary Configuration

There are two necessary configurations that are required at kube-apiserver level:

1. Generate Client Certificates for API Server for Client Authentication with ETCD.
2. Connect API Server with ETCD over HTTPS

9.4 Challenge with Plain Text Storage in ETCD

It is also important that the secrets that are stored in ETCD are encrypted.



9.5 Encrypted Secrets Storage

Module 10: Encryption Provider Config

The kube-apiserver process accepts an argument --encryption-provider-config that controls how API data is encrypted in etcd.

```
cat > encryption-at-rest.yaml
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: ${ENCRYPTION_KEY}
      - identity: {}
```

Module 11: Transport Security for API Server

11.1 In-Transit Encryption

If the communication between the client and kube-apiserver is happening based on an in-secure port, it leads to additional risks.

11.2 API Server Ports and IPs

By default the Kubernetes API server serves HTTP on 2 ports:

Ports	Description
Localhost Port	Intended for Testing. No TLS default is port 8080, change with --insecure-port flag. default IP is localhost, change with --insecure-bind-address flag. request bypasses authentication and authorization modules.
Secure Port	use whenever possible uses TLS. cert with --tls-cert-file and key with --tls-private-key-file flag. default is port 6443, change with --secure-port flag. request handled by authentication and authorization modules.

11.3 Important Pointer

To avoid unauthenticated access to the master node a best practice is to set the

--insecure-port flag to 0

Module 12: Integration with systemd

12.1 Overview of Challenges

As of now, we have been deploying K8s components from CLI.

Although this is a good approach for an initial start and testing, for the production it is not an ideal approach.

```
[root@k8s-security ~]# /usr/local/bin/kube-apiserver --advertise-address=143.110.250.107 --etcd-cafile=/root/certifi
cates/ca.crt --etcd-certfile=/root/certificates/apiserver-etcd-client.crt --etcd-keyfile=/root/certificates/apiserve
r-etcd-client.key --etcd-servers=https://127.0.0.1:2379 --encryption-provider-config=/var/lib/kubernetes/encryption-
at-rest.yaml --tls-cert-file=/var/lib/kubernetes/kube-api.crt --tls-private-key-file=/var/lib/kubernetes/kube-api.ke
y --insecure-port 0
Flag --insecure-port has been deprecated, This flag will be removed in a future version.
W1202 06:42:05.686463   15454 services.go:37] No CIDR for service cluster IPs specified. Default value which was 10.
0.0.0/24 is deprecated and will be removed in future releases. Please specify it using --service-cluster-ip-range on
 kube-apiserver.
I1202 06:42:05.686541   15454 server.go:625] external host was not specified, using 143.110.250.107
W1202 06:42:05.686558   15454 authentication.go:484] AnonymousAuth is not allowed with the AlwaysAllow authorizer. R
esetting AnonymousAuth to false. You should use a different authorizer
```

12.2 Systemd Based Approach

Systemd provides a system and service manager that runs as PID 1 and starts the rest of the system

You can specify a service file with all the configuration and settings and systemd will manage the process accordingly.

```
[root@ip-172-31-54-201 certificates]# systemctl status etcd
● etcd.service - etcd
   Loaded: loaded (/etc/systemd/system/etcd.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2020-12-02 07:01:33 UTC; 3s ago
     Docs: https://github.com/coreos
 Main PID: 15813 (etcd)
    Tasks: 10 (limit: 23800)
   Memory: 11.1M
   CGroup: /system.slice/etcd.service
           └─15813 /usr/local/bin/etcd --cert-file=/root/certificates/etcd.crt --key-file=/root/certificates/etcd.k

Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: ClientTLS: cert = /root/certificates/etcd.crt, key = /ro
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: listening for peers on 127.0.0.1:2380
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: raft2020/12/02 07:01:35 INFO: 8e9e05c52164694d is starti
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: raft2020/12/02 07:01:35 INFO: 8e9e05c52164694d became ca
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: raft2020/12/02 07:01:35 INFO: 8e9e05c52164694d received
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: raft2020/12/02 07:01:35 INFO: 8e9e05c52164694d became le
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: raft2020/12/02 07:01:35 INFO: raft.node: 8e9e05c52164694
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: published {Name:default ClientURLs:[https://127.0.0.1:23
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: ready to serve client requests
Dec 02 07:01:35 ip-172-31-54-201.ec2.internal etcd[15813]: serving client requests on 127.0.0.1:2379
```

12.3 Component Logs

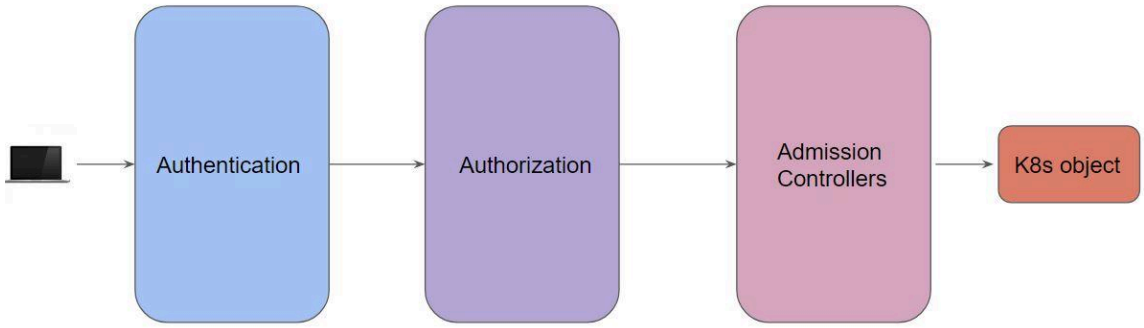
If we are making use of systemd, then we can view the component level logs with journalctl

It provides insight into what is happening inside a cluster, such as what decisions were made by the scheduler or why some pods were evicted from the node.

```
[root@ip-172-31-54-201 ~]# journalctl -u etcd
-- Logs begin at Tue 2020-12-01 13:32:53 UTC, end at Wed 2020-12-02 07:07:31 UTC. --
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal systemd[1]: Started etcd.
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: [WARNING] Deprecated '--logger=capnslog' flag is set;
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: [WARNING] Deprecated '--logger=capnslog' flag is set;
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: etcd Version: 3.4.10
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: Git SHA: 18dfb9cca
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: Go Version: go1.12.17
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: Go OS/Arch: linux/amd64
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: setting maximum number of CPUs to 2, total number of
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: the server is already initialized as member before, s
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: name = default
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: data dir = /var/lib/etcd
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: member dir = /var/lib/etcd/member
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: heartbeat = 100ms
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: election = 1000ms
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: snapshot count = 100000
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: advertise client URLs = https://127.0.0.1:2379
Dec 02 07:01:33 ip-172-31-54-201.ec2.internal etcd[15813]: initial advertise peer URLs = http://localhost:2380
```

Module 13: Access Control

When a request reaches the API, it goes through several stages, illustrated in the following diagram:



Step 1: Authentication

There are multiple ways in which we can authenticate. Some of these include:

Authentication Modes	Description
X509 Client Certificates	Valid client certificates signed by trusted CA.
Static Token File	Sets of bearer token mentioned in a file.

Step 2: Authorization

After the request is authenticated as coming from a specific user, the request must be authorized.

Multiple authorization modules are supported.

Authorization Modes	Description
AlwaysDeny	Blocks all requests (used in tests).
AlwaysAllow	Allows all requests; use if you don't need authorization.
RBAC	Allows you to create and store policies using the Kubernetes API.
Node	A special-purpose authorization mode that grants permissions to kubelets

Module 14: Downsides of Token Authentication

The token-based authentication utilizes static tokens to authenticate requests to the apiserver

The tokens are stored in clear-text in a file on the apiserver, and cannot be revoked or rotated without restarting the apiserver.

Hence, do not use static token-based authentication

```
Dem0Passw0rd#,bob,01,admins
```

Module 15: X509 Client Certificates - Authentication

A request is authenticated if the client certificate is signed by one of the certificate authorities that is configured in the API server.

Module 16: Downside of X509 Certificate Authentication

Following are some of the disadvantages of X.509 certificate-based authentication:

The private key is stored on insecure media (local disk storage).

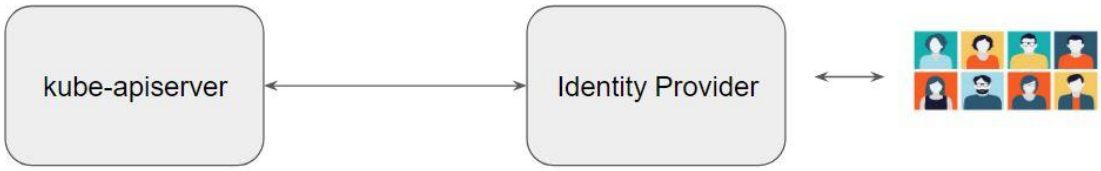
Certificates are generally long-lived. Kubernetes does not support certificate revocation related area.

Groups are associated with Organization in the certificate. If you want to change the group, you will have to issue a new certificate.

Module 17: OpenID Connect Authentication

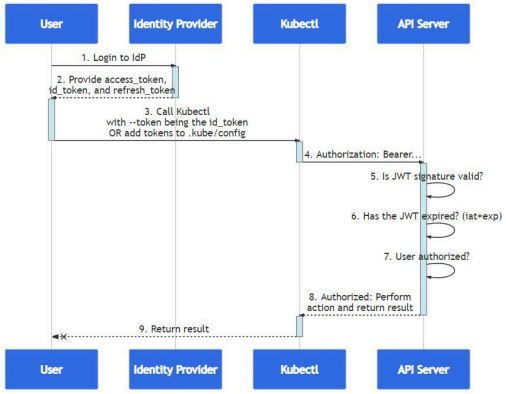
17.1 Overview of OIDC

In this approach, an Identity provider is used for authentication. API server must trust the identity provider.



17.2 Basic Working Steps

- Login to the Identity Provider.
- Identity Provider provides access_token, id_token and refresh_token.
- Make use of the id_token with --token flag while making use of kubectl.
- kubectl sends your id_token in a header called Authorization to the API server
- API servers take care of verification



17.3 Integration with the API Server

There are various additional configurations needed for the API server as part of the integration.

--oidc-issuer-url	The URL of the OpenID issuer
--oidc-username-claim	The OpenID claim to use as the user name
--oidc-client-id	client ID for the OpenID Connect client

```
--oidc-issuer-url=https://accounts.google.com \
--oidc-username-claim=email \
--oidc-client-id="61743321-jl6b754a.apps.googleusercontent.com" \
```

Module 18: RBAC Authorization

18.1 Overview of Authorization

After the request is authenticated as coming from a specific user, the request must be authorized.

Multiple authorization modules are supported.

Authorization Modes	Description
AlwaysDeny	Blocks all requests (used in tests).
AlwaysAllow	Allows all requests; use if you don't need authorization.
RBAC	Allows you to create and store policies using the Kubernetes API.
Node	A special-purpose authorization mode that grants permissions to kubelets

18.2 Two Important Concepts

A role contains rules that represent a set of permissions

A role binding grants the permissions defined in a role to a user or set of users.

Role Binding:

ReadSome
- zeal
- john
- alice

Role ReadSome

Allow:
Read pods
Read services
Read secrets

18.3 Important Pointers - Certificates

Within a certificate, there are two important fields:

Common Name (CN) and Organization (O)

openssl req -new -key alice.key -subj "/CN=alice/O=admins" -out alice.csr

The above commands create CSR for the username alice belonging to admins group.

Default Cluster Role and Cluster Role Binding

18.4 Default Cluster Role and Cluster Role Binding

Default ClusterRole	Default ClusterRoleBinding	Description
cluster-admin	system:masters group	Allows super-user access to perform any action on any resource.
admin	None	Allows admin access, intended to be granted within a namespace using a RoleBinding
edit	None	Allows read/write access to most objects in a namespace.
view	None	Allows read-only access to see most objects in a namespace.

Module 19: Implementing Auditing

19.1 Overview of Auditing Functionality

Auditing provides a security-relevant, chronological set of records documenting the sequence of actions in a cluster.

The cluster audits the activities generated by users, by applications that use the Kubernetes API, and by the control plane itself.

- what happened?
- when did it happen?
- who initiated it?
- on what did it happen?
- from where was it initiated?
- to where was it going?


```
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "Metadata",
  "auditID": "388ca4e1-c368-45b4-aca9-652e32baf8e1",
  "stage": "RequestReceived",
  "requestURI": "/api/v1/namespaces/default/secrets?limit=500",
  "verb": "list",
  "user": {
    "username": "bob",
    "groups": [
      "system:masters",
      "system:authenticated"
    ]
  },
  "sourceIPs": [
    "127.0.0.1"
  ],
  "userAgent": "kubectl/v1.19.0 (linux/amd64) kubernetes/e199641",
  "objectRef": {
    "resource": "secrets",
    "namespace": "default",
    "apiVersion": "v1"
  },
  "requestReceivedTimestamp": "2020-12-03T16:55:10.357789Z",
  "stageTimestamp": "2020-12-03T16:55:10.357789Z"
}
```

19.2 Audit Policy

Audit policy defines rules about what events should be recorded and what data they should include.

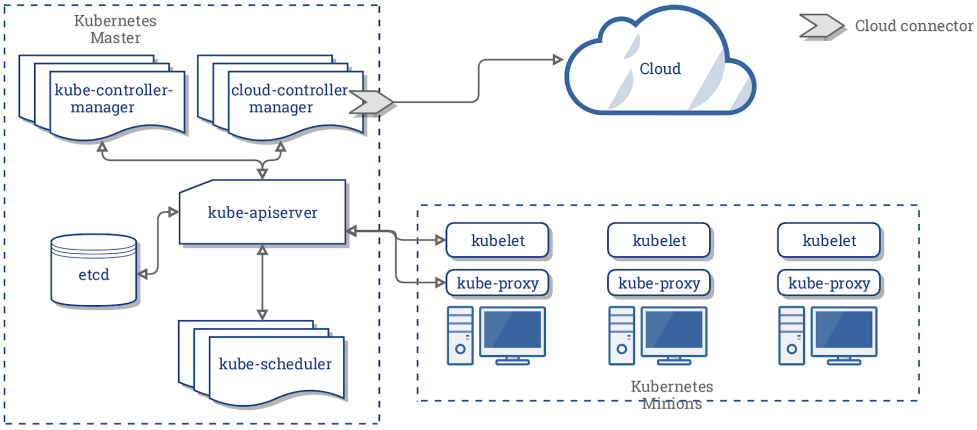
Audit Levels	Description
None	don't log events that match this rule.
Metadata	Log request metadata (requesting user, timestamp, resource, verb, etc.) but not request or response body.
Request	Log event metadata and request body but not response body.
RequestResponse	Log event metadata, request and response bodies.

19.3 Important Flags

Audit Configuration	Description
-audit-policy-file	Path to the file that defines the audit policy configuration.
-audit-log-path	Specifies the log file path that log backend uses to write audit events.
--audit-log-maxage	Maximum number of days to retain old audit log files
--audit-log-maxbackup	Maximum number of audit log files to retain
--audit-log-maxsize	Maximum size in MB of the audit log file before it gets rotated

Module 20: Setting up kubeadm

Kubeadm allows us to quickly provision a secure Kubernetes cluster.



We will be using kubeadm for the following purpose:

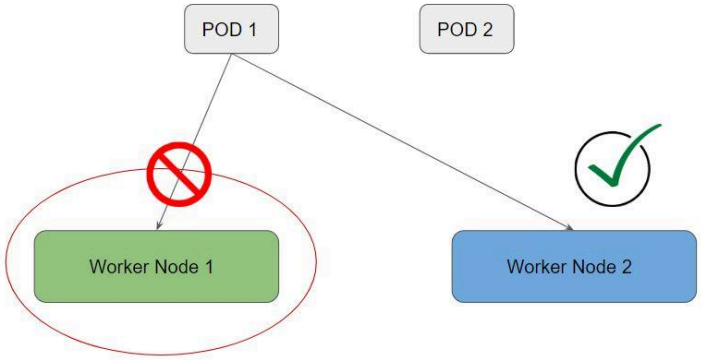
Primarily to explore various security-related configurations that are configured.

Experiment and Learn new interesting security-related topics.

Module 21: Revising Taints and Tolerations

21.1 Understanding Taints:

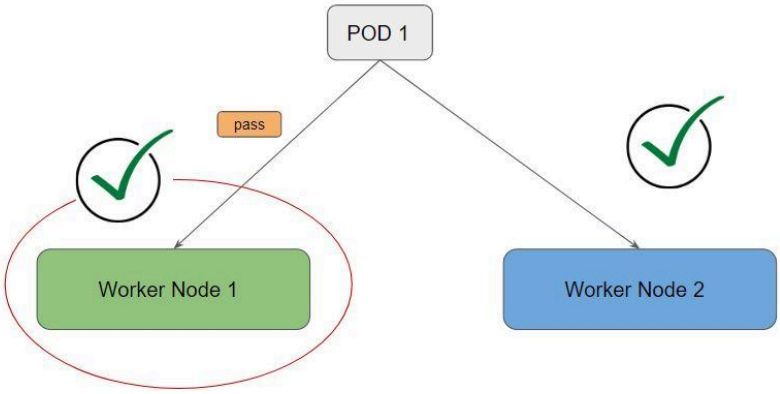
Taints are used to repel the pods from a specific node.



21.2 Understanding Toleration:

In order to enter the taint worker node, you need a special pass.

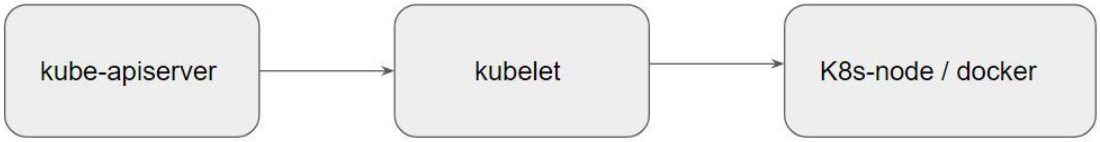
This pass is called toleration.



Module 22: Kubelet Security

22.1 Revising Kubelet

One of the primary tasks of a kubelet is managing the local container engine (i.e. Docker) and ensuring that the pods described in the API are defined, created, run, and remain healthy; and then destroyed when appropriate.



22.2 Important Flags

Important Configuration	Description
--anonymous-auth	Requests that are not rejected by other configured authentication methods are treated as anonymous requests.
--authorization-mode	AlwaysAllow, Webhook
--client-ca-file	start the kubelet with the --client-ca-file flag, providing a CA bundle to verify client certificates
--read-only-port	Associated with ReadOnlyAPI

22.3 Node Authorizer

Node authorization is a special-purpose authorization mode that specifically authorizes API requests made by kubelets.

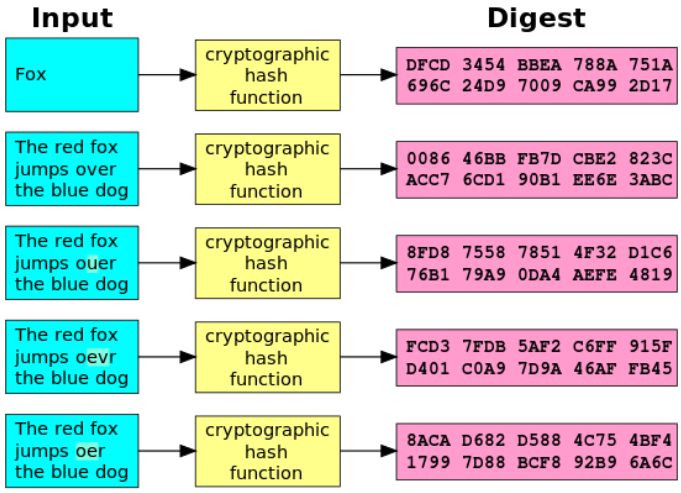
In order to be authorized by the Node authorizer, kubelets must use a credential that identifies them as being in the system:nodes group, with a username of system:node:<nodeName>

Operations	Endpoints
Read	services, endpoints, nodes, pods, secrets, and others.
Write	nodes and node status, pods and pod status, events
Auth-Related	R/W to CSR for TLS Bootstrapping

Module 23: Verifying Platform Binaries

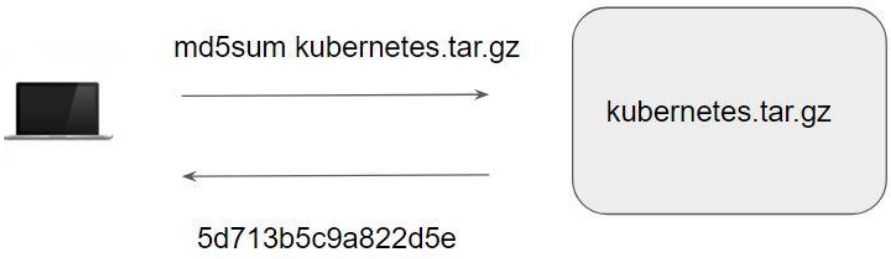
23.1 Overview of Cryptographic Hash Function

Hashing is a one-way function that maps data of arbitrary size (often called the "message") to a bit array of a fixed size (message digest)



23.2 Verify Platform Binaries

To verify the integrity of the binaries, you can take a hash of the archive file and compare it with the hash posted on the official website.



Join Our Discord Community

We invite you to join our Discord community, where you can interact with our support team for any course-based technical queries and connect with other students who are doing the same course.

Joining URL:

http://kplabs.in/chat