INFORMATION AND COMPUTER SCIENCE DEPARTMENT

Microservices Applications Attack and Detection

Project Deliverables R3

Team Members

Abdulrahman AlShehri	G201215060
Omar AlGhamdi	G202307090
Sultan AlHomoud	G202307190

Mentors

Dr. Waleed AlGobi

Submitted as part of the requirements for the SEC 619 for the Professional Mater of Cybersecurity

Term: 241, SEC 619: Project

Table of Contents

7.1 Prerequisites:

- Below are the list of technologies and necssary softwares
  - I. *Kubernetes (local deployment)*
  - II. *Sysdig (open-source version)*
  - III. *Docker (for containerized environments)*
  - IV. *Chisel (Sysdig extension, Lua script)*
  - V. *Python*
  - VI. *Replicawatcher (git repository)*

- Hardware Requirements

  – The following table lists the four machines required for the cluster, along with their CPU, memory, and storage specifications.
  – The following four machines as described in the below table consider as the hardware requirements, such as minimum memory or CPU specs

Name	**Role**	**IP Address**	**CPU**	**RAM**	**Storage**	**OS**
jumpbox	Administration host	192.168.1.30	1	2GB	20GB	Debian 12 (bookworm) AMD
kubernetes-server	Kubernetes server (Control Plane)	192.168.1.0	1	4GB	20GB	Debian 12 (bookworm) AMD
node-0	Kubernetes worker node	192.168.1.199	3	10GB	20GB	Debian 12 (bookworm) AMD
node-1	Kubernetes worker node	192.168.1.48	3	10GB	20GB	Debian 12 (bookworm) AMD

– Mention any hardware requirements, such as minimum memory or CPU specs.

• Installation Instructions:

We have followed famous kubernetes deployment for deploying self-hosted clusters (Kubernetes-the-hard-way). In addition, we have included changes to the implementation, modified the images of all nodes to AMD instead of AARCH64 as well as all downloaded applications listed in the repo AMD instead of AARCH64.

Environment setup

1. **Provision Machines**

   - Install Debian 12 on all machines.
   - Configure SSH access and verify connectivity between all nodes.
   - Set hostnames as kubernetes-server, node-0, and node-1

2. **Set Up Jumpbox**
   Install utilities and Kubernetes bash

3. Set Up Cluster

3.1 Configure Machines and Hostnames

Setting unique hostnames and enabling hostname-based communication between nodes simplifies cluster management. Updating the /etc/hosts file on all machines ensures they can communicate using hostnames instead of IP addresses, which is essential for Kubernetes components to operate seamlessly.

Steps:

1. Assign hostnames:

   ```
   hostnamectl set-hostname <hostname>
   ```

   Example hostnames:

   - kubernetes-server.kubernetes.local for the control plane.
   - node-0.kubernetes.local and node-1.kubernetes.local for worker nodes.

2. Update /etc/hosts on all machines:

   ```
   echo "<IP> kubernetes-server.kubernetes.local kubernetes-server" >> /etc/hosts
   echo "<IP> node-0.kubernetes.local node-0" >> /etc/hosts
   echo "<IP> node-1.kubernetes.local node-1" >> /etc/hosts
   ```

3.2 Generate TLS Certificates

TLS certificates are crucial for securing communication between Kubernetes components, ensuring confidentiality, integrity, and mutual authentication. A Certificate Authority (CA) is created to sign all certificates, and individual certificates are generated for components like the API server, kubelet, and etcd.

Steps:

1. Create a Certificate Authority (CA):

   ```
   openssl genrsa -out ca.key 4096
   openssl req -x509 -new -key ca.key -days 1000 -out ca.crt -subj "/CN=kubernetes-ca"
   ```

2. Generate client server certificates for each component within the cluster:

   Generate the certificates and private keys:

   ```
   certs=(
   "admin" "node-0" "node-1"
   "kube-proxy" "kube-scheduler"
   "kube-controller-manager"
   "kube-api-server"
   "service-accounts"
   )

   for i in ${certs[*]}; do
   openssl genrsa -out "${i}.key" 4096
   ```

```
openssl req -new -key "${i}.key" -sha256 \
-config "ca.conf" -section ${i} \
-out "${i}.csr"

openssl x509 -req -days 3653 -in "${i}.csr" \
-copy_extensions copyall \
-sha256 -CA "ca.crt" \
-CAkey "ca.key" \
-CAcreateserial \
-out "${i}.crt"
done
```

The results of running the above command will generate a private key, certificate request, and signed SSL certificate for each of the Kubernetes components.

3. Distribute certificates to nodes:

Copy the appropriate certificates and private keys to the node-0 and node-1 machines:

```
for host in node-0 node-1; do
  ssh root@$host mkdir /var/lib/kubelet/

  scp ca.crt root@$host:/var/lib/kubelet/

  scp $host.crt \
    root@$host:/var/lib/kubelet/kubelet.crt

  scp $host.key \
    root@$host:/var/lib/kubelet/kubelet.key
done
```

Copy the appropriate certificates and private keys to the kubernetes-server machine:

```
scp \
  ca.key ca.crt \
  kube-api-server.key kube-api-server.crt \
  service-accounts.key service-accounts.crt \
  root@kubernetes-server:~/
```

The kube-proxy, kube-controller-manager, kube-scheduler, and kubelet client certificates will be used to generate client authentication configuration files in the next lab.

3.3 Generating Kubernetes Configuration Files for Authentication

Configure Kubernetes clients to connect and authenticate to Kubernetes API Servers. These configuration files are crucial for enabling secure communication between the Kubernetes API Server and its components. Key steps include:

- **Creating kubeconfig files for each component**: Generate configuration files for kubelet (e.g., node-0.kubeconfig, node-1.kubeconfig), kube-proxy (kube-proxy.kubeconfig), kube-controller-manager (kube-controller-manager.kubeconfig), kube-scheduler (kube-scheduler.kubeconfig), and admin (admin.kubeconfig), using the certificate authority (CA) and client certificates generated during the TLS Certificates lab.

- **Specifying cluster details**: Each kubeconfig specifies the cluster information, credentials, and context for the respective component, ensuring proper authentication and authorization.

- **Distributing kubeconfig files**: Copy the kubeconfig files to their respective worker nodes or controller instances (e.g., /var/lib/kubelet and /var/lib/kube-proxy on worker nodes) to establish secure communication.

- **Setting up data encryption**: Generate a 32-byte encryption key and create an encryption-config.yaml file to encrypt Kubernetes Secrets at rest, ensuring data security within the cluster.

By following these steps, the Kubernetes components are securely configured to communicate with the API server.

3.4 Set Up the etcd Cluster

The etcd cluster serves as the backbone of the Kubernetes control plane, storing cluster state, configuration, and secrets. Setting up a highly available and secure etcd cluster ensures consistent and reliable access to this critical data.
Steps:

Install the etcd Binaries

Extract and install the etcd server and the etcdctl command line utility:

```
{
  tar -xvf etcd-v3.4.34-linux-arm64.tar.gz
  mv etcd-v3.4.34-linux-arm64/etcd* /usr/local/bin/
}
```

Configure the etcd Server

```
{
  mkdir -p /etc/etcd /var/lib/etcd
  chmod 700 /var/lib/etcd
  cp ca.crt kube-api-server.key kube-api-server.crt \
    /etc/etcd/
}
```

Each etcd member must have a unique name within an etcd cluster. Set the etcd name to match the hostname of the current compute instance:
Create the etcd.service systemd unit file:

```
mv etcd.service /etc/systemd/system/
```

Start the etcd Server

```
{
  systemctl daemon-reload
  systemctl enable etcd
  systemctl start etcd
}
```

3.5 Set Up the Kubernetes Control Plane

The control plane is the core of Kubernetes, responsible for managing the cluster and scheduling workloads. Setting up the API server, controller manager, and scheduler ensures the cluster can orchestrate workloads securely and efficiently.

Steps:

Install Kubernetes binaries on the control plane (kubernetes-server):

Connect to the jumpbox and copy Kubernetes binaries and systemd unit files to the kubernetes-server instance:

```
scp \
  downloads/kube-apiserver \
  downloads/kube-controller-manager \
  downloads/kube-scheduler \
  downloads/kubectl \
  units/kube-apiserver.service \
  units/kube-controller-manager.service \
  units/kube-scheduler.service \
  configs/kube-scheduler.yaml \
  configs/kube-apiserver-to-kubelet.yaml \
  root@kubernetes-server:~/
```

The commands in this lab must be run on the controller instance: kubernetes-server. Login to the controller instance using the ssh command. Example:

```
ssh root@kubernetes-server
```

Provision the Kubernetes Control Plane

Create the Kubernetes configuration directory:

```
mkdir -p /etc/kubernetes/config
```

Install the Kubernetes Controller Binaries

Install the Kubernetes binaries:

```
{
  chmod +x kube-apiserver \
    kube-controller-manager \
    kube-scheduler kubectl

  mv kube-apiserver \
    kube-controller-manager \
    kube-scheduler kubectl \
    /usr/local/bin/
}
```

Configure the API Server:

```
{
  mkdir -p /var/lib/kubernetes/

  mv ca.crt ca.key \
    kube-api-server.key kube-api-server.crt \
    service-accounts.key service-accounts.crt \
    encryption-config.yaml \
    /var/lib/kubernetes/
}
```

Create the kube-apiserver.service systemd unit file:

```
mv kube-apiserver.service \
  /etc/systemd/system/kube-apiserver.service
```

Configure the Kubernetes Controller Manager

Move the kube-controller-manager kubeconfig into place:

```
mv kube-controller-manager.kubeconfig /var/lib/kubernetes/
```

Create the kube-controller-manager.service systemd unit file:

```
mv kube-controller-manager.service /etc/systemd/system/
```

Configure the Kubernetes Scheduler

Move the kube-scheduler kubeconfig into place:

```
mv kube-scheduler.kubeconfig /var/lib/kubernetes/
```

Create the kube-scheduler.yaml configuration file:

```
mv kube-scheduler.yaml /etc/kubernetes/config/
```

Create the kube-scheduler.service systemd unit file:

```
mv kube-scheduler.service /etc/systemd/system/
```

Start the Controller Services

```
{
  systemctl daemon-reload

  systemctl enable kube-apiserver \
    kube-controller-manager kube-scheduler

  systemctl start kube-apiserver \
    kube-controller-manager kube-scheduler
}
```

3.3.5 Set Up Worker Nodes

Worker nodes host application containers and handle resource allocation. Configuring kubelet and kube-proxy on each node ensures they can manage workloads and integrate with the control plane.

Steps:

Install Kubernetes binaries on worker nodes (node-0 and node-1):

Copy Kubernetes binaries and systemd unit files to each worker instance:

```
for host in node-0 node-1; do
  SUBNET=$(grep $host machines.txt | cut -d " " -f 4)
  sed "s|SUBNET|$SUBNET|g" \
    configs/10-bridge.conf > 10-bridge.conf

  sed "s|SUBNET|$SUBNET|g" \
    configs/kubelet-config.yaml > kubelet-config.yaml

  scp 10-bridge.conf kubelet-config.yaml \
  root@$host:~/
done
for host in node-0 node-1; do
  scp \
    downloads/runc.arm64 \
    downloads/crictl-v1.31.1-linux-arm64.tar.gz \
    downloads/cni-plugins-linux-arm64-v1.6.0.tgz \
    downloads/containerd-2.0.0-linux-arm64.tar.gz \
    downloads/kubectl \
    downloads/kubelet \
    downloads/kube-proxy \
    configs/99-loopback.conf \
    configs/containerd-config.toml \
    configs/kubelet-config.yaml \
    configs/kube-proxy-config.yaml \
    units/containerd.service \
    units/kubelet.service \
    units/kube-proxy.service \
    root@$host:~/
done
```

The commands must be run on each worker instance: node-0, node-1. Then Login to the worker instance using the ssh command. Example:

```
ssh root@node-0
```

The kubelet will fail to start if swap is enabled. It is recommended that swap be disabled to ensure Kubernetes can provide proper resource allocation and quality of service.

```
Swapoff -a
```

Create the installation directories:

```
mkdir -p \
  /etc/cni/net.d \
  /opt/cni/bin \
  /var/lib/kubelet \
  /var/lib/kube-proxy \
  /var/lib/kubernetes \
  /var/run/kubernetes
```

Install the worker binaries:

```
{
  mkdir -p containerd
  tar -xvf crictl-v1.31.1-linux-arm64.tar.gz
  tar -xvf containerd-2.0.0-linux-arm64.tar.gz -C containerd
  tar -xvf cni-plugins-linux-arm64-v1.6.0.tgz -C /opt/cni/bin/
  mv runc.arm64 runc
  chmod +x crictl kubectl kube-proxy kubelet runc
  mv crictl kubectl kube-proxy kubelet runc /usr/local/bin/
  mv containerd/bin/* /bin/
}
```

Configure CNI Networking

Create the bridge network configuration file:

```
mv 10-bridge.conf 99-loopback.conf /etc/cni/net.d/
```

Configure containerd

Install the containerd configuration files:

```
{
  mkdir -p /etc/containerd/
  mv containerd-config.toml /etc/containerd/config.toml
  mv containerd.service /etc/systemd/system/
}
```

Configure the Kubelet

Create the kubelet-config.yaml configuration file:

```
{
  mv kubelet-config.yaml /var/lib/kubelet/
  mv kubelet.service /etc/systemd/system/
}
```

Configure the Kubernetes Proxy

```
{
  mv kube-proxy-config.yaml /var/lib/kube-proxy/
  mv kube-proxy.service /etc/systemd/system/
}
```

Start the Worker Services

```
{
  systemctl daemon-reload
  systemctl enable containerd kubelet kube-proxy
  systemctl start containerd kubelet kube-proxy
}
```

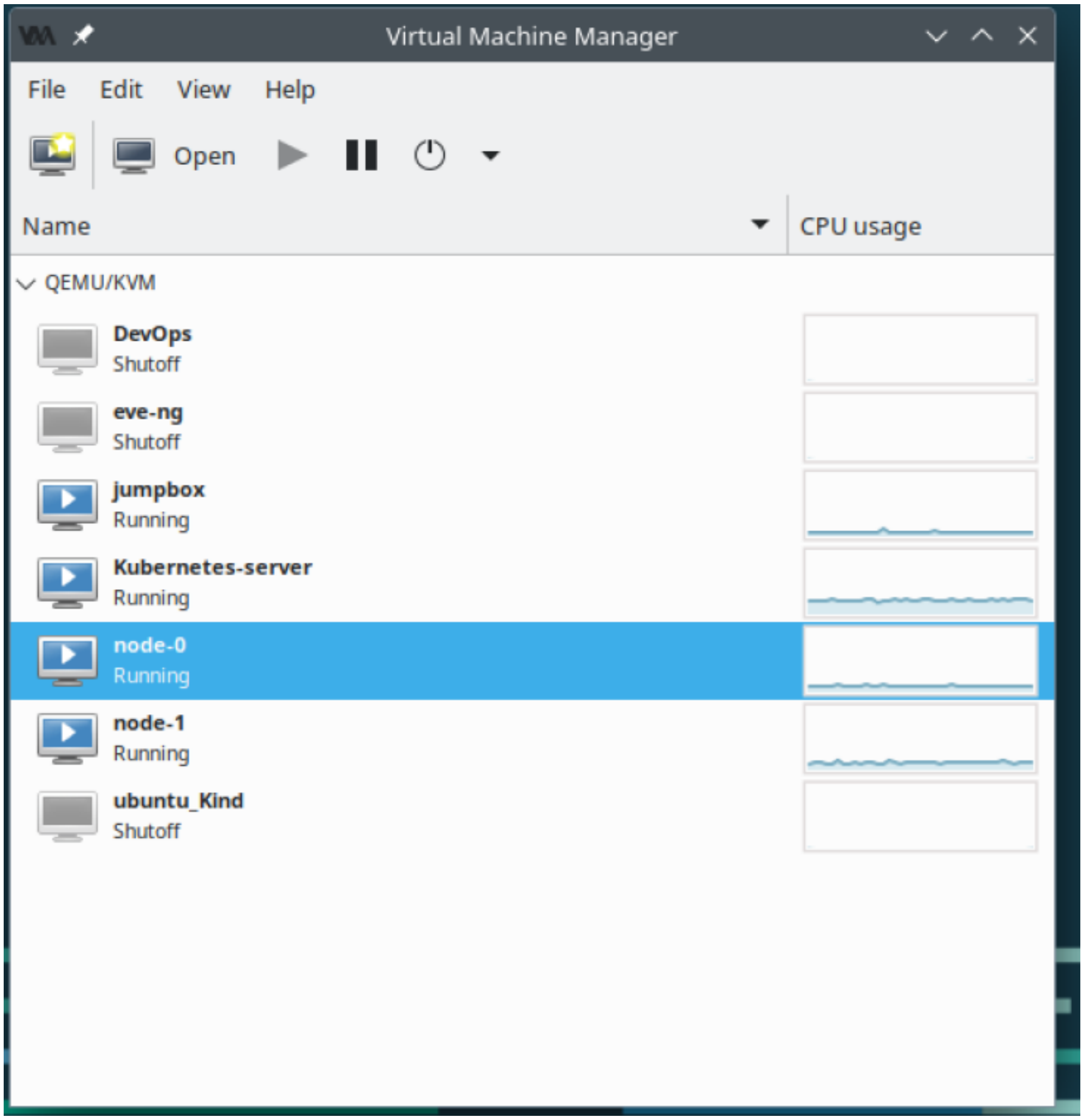
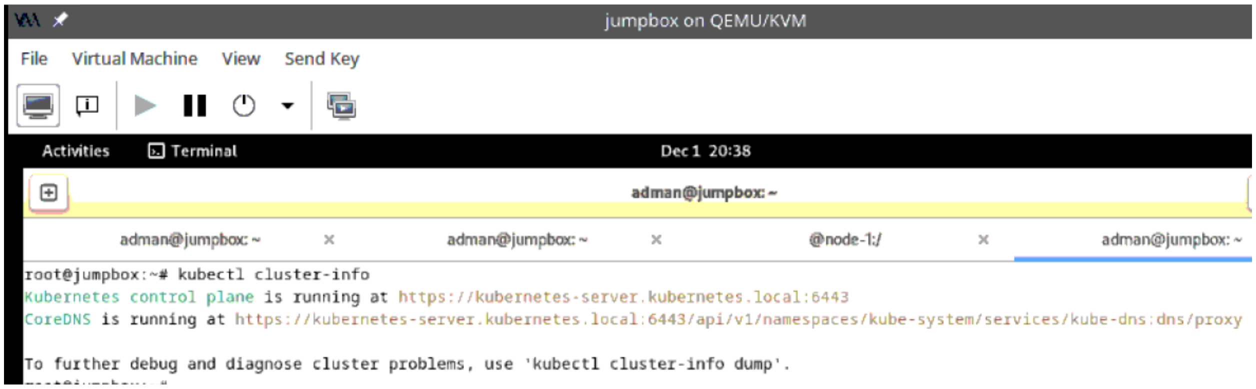
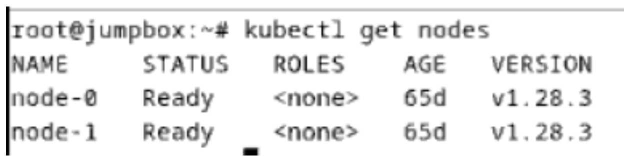


Figure 1: Cluster Deployment



```
root@jumpbox:~# kubectl cluster-info
Kubernetes control plane is running at https://kubernetes-server.kubernetes.local:6443
CoreDNS is running at https://kubernetes-server.kubernetes.local:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Figure 2: Cluster information



```
root@jumpbox:~# kubectl get nodes
NAME     STATUS   ROLES    AGE   VERSION
node-0   Ready    <none>   65d   v1.28.3
node-1   Ready    <none>   65d   v1.28.3
```

Figure 3: Cluster Working Nodes

3.3.7 Cluster DNS:
The implementation of Kubernetes the hard way does not include ClusterDNS, which enables the one service to reach other services only by the name similar network DNS but on Microservices level. For that, Coredns pods had to be deployed on the cluster as well making Kube-dns service reachable on cluster.

1. Deploying CoreDNS deployment using manifest which will create pods that handle the workload of DNS resolution:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: coredns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/name: "CoreDNS"
spec:
  # replicas: not specified here:
  # 1. Default is 1.
  # 2. Will be tuned in real time if DNS horizontal auto-scaling is turned on.
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  selector:
    matchLabels:
      k8s-app: kube-dns
  template:
    metadata:
      labels:
        k8s-app: kube-dns
    spec:
      priorityClassName: system-cluster-critical
      serviceAccountName: coredns
      tolerations:
        - key: "CriticalAddonsOnly"
          operator: "Exists"
      nodeSelector:
```

```
  kubernetes.io/os: linux
affinity:
   podAntiAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - weight: 100
       podAffinityTerm:
         labelSelector:
           matchExpressions:
             - key: k8s-app
               operator: In
               values: ["kube-dns"]
         topologyKey: kubernetes.io/hostname
containers:
- name: coredns
  image: coredns/coredns:1.8.0
  imagePullPolicy: IfNotPresent
  resources:
    limits:
      memory: 170Mi
    requests:
      cpu: 100m
      memory: 70Mi
  args: [ "-conf", "/etc/coredns/Corefile" ]
  volumeMounts:
  - name: config-volume
    mountPath: /etc/coredns
    readOnly: true
  ports:
  - containerPort: 53
    name: dns
    protocol: UDP
  - containerPort: 53
    name: dns-tcp
    protocol: TCP
  - containerPort: 9153
    name: metrics
    protocol: TCP
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      add:
      - NET_BIND_SERVICE
      drop:
      - all
    readOnlyRootFilesystem: true
  livenessProbe:
    httpGet:
      path: /health
      port: 8080
      scheme: HTTP
    initialDelaySeconds: 60
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 5
  readinessProbe:
    httpGet:
      path: /ready
      port: 8181
      scheme: HTTP
dnsPolicy: Default
volumes:
  - name: config-volume
    configMap:
      name: coredns
      items:
      - key: Corefile
        path: Corefile
```

2. Creating Kube-DNS to make the DNS service only accessible to all services through one ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  annotations:
    prometheus.io/port: "9153"
    prometheus.io/scrape: "true"
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: "CoreDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 192.168.122.10
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
```

```
    port: 53
    protocol: TCP
  - name: metrics
    port: 9153
    protocol: TCP
```

3. Implementing ConfigMap manifest to specify the dns configuration file and content which will be loaded to created coredns pods.

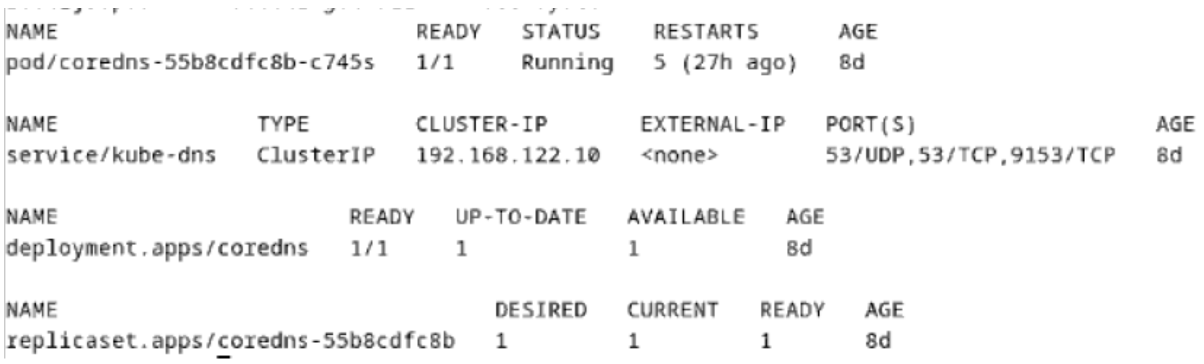
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
        errors
        health {
          lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
          fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
          max_concurrent 1000
        }
        cache 30
        loop
        reload
        loadbalance
    }
```

4. Creating ClusterRole and ClusterRoleBinding to define the access level of DNS pods to Kubernetes API

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:coredns
rules:
- apiGroups:
  - ""
  resources:
  - endpoints
  - services
  - pods
  - namespaces
  verbs:
  - list
  - watch

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:coredns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:coredns
subjects:
- kind: ServiceAccount
  name: coredns
  namespace: kube-system
```

5. Creating ServiceAccount to be used by CoreDNS pods to interact with kubernetes API

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: coredns
  namespace: kube-system
```



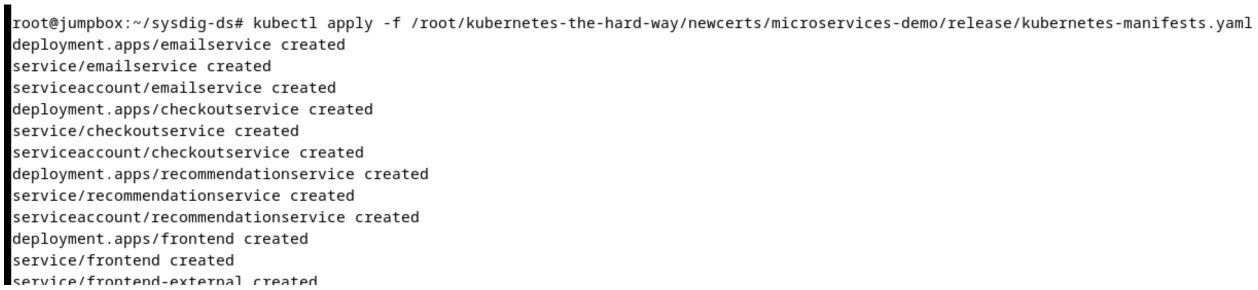
```
NAME                          READY   STATUS    RESTARTS      AGE
pod/coredns-55b8cdfc8b-c745s  1/1     Running   5 (27h ago)   8d

NAME               TYPE        CLUSTER-IP        EXTERNAL-IP   PORT(S)                   AGE
service/kube-dns   ClusterIP   192.168.122.10    <none>        53/UDP,53/TCP,9153/TCP    8d

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/coredns    1/1     1            1           8d

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/coredns-55b8cdfc8b    1         1         1       8d
```

Figure 4: DNS Configuration Result

6. Mounting br_netfilter kernel module

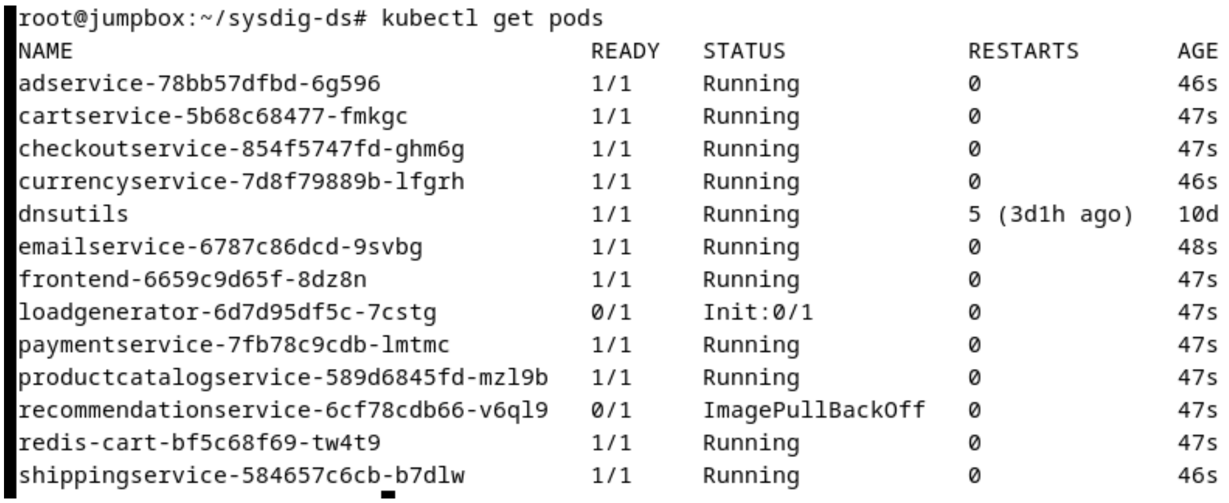
modprobe br_netfilter

4. **Deploy the Online Boutique Application**
    1. Clone Application Repository
       git clone https://github.com/GoogleCloudPlatform/microservices-demo.git
    2. Deploy Application
       kubectl apply -f release/kubernetes-manifests.yaml



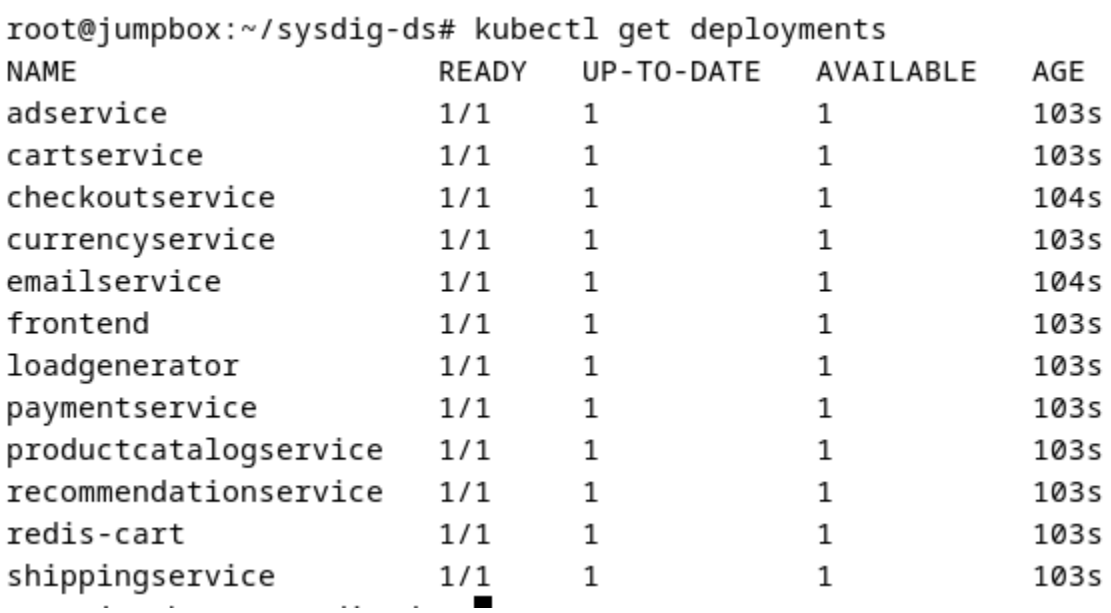
```
root@jumpbox:~/sysdig-ds# kubectl apply -f /root/kubernetes-the-hard-way/newcerts/microservices-demo/release/kubernetes-manifests.yaml
deployment.apps/emailservice created
service/emailservice created
serviceaccount/emailservice created
deployment.apps/checkoutservice created
service/checkoutservice created
serviceaccount/checkoutservice created
deployment.apps/recommendationservice created
service/recommendationservice created
serviceaccount/recommendationservice created
deployment.apps/frontend created
service/frontend created
service/frontend-external created
```

Figure 5: GOB deployment



```
root@jumpbox:~/sysdig-ds# kubectl get pods
NAME                                     READY   STATUS             RESTARTS       AGE
adservice-78bb57dfbd-6g596               1/1     Running            0              46s
cartservice-5b68c68477-fmkgc             1/1     Running            0              47s
checkoutservice-854f5747fd-ghm6g         1/1     Running            0              47s
currencyservice-7d8f79889b-lfgrh         1/1     Running            0              46s
dnsutils                                 1/1     Running            5 (3d1h ago)   10d
emailservice-6787c86dcd-9svbg            1/1     Running            0              48s
frontend-6659c9d65f-8dz8n                1/1     Running            0              47s
loadgenerator-6d7d95df5c-7cstg           0/1     Init:0/1           0              47s
paymentservice-7fb78c9cdb-lmtmc          1/1     Running            0              47s
productcatalogservice-589d6845fd-mzl9b   1/1     Running            0              47s
recommendationservice-6cf78cdb66-v6ql9   0/1     ImagePullBackOff   0              47s
redis-cart-bf5c68f69-tw4t9               1/1     Running            0              47s
shippingservice-584657c6cb-b7dlw         1/1     Running            0              46s
```

Figure 6: Deployed pods



```
root@jumpbox:~/sysdig-ds# kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
adservice               1/1     1            1           103s
cartservice             1/1     1            1           103s
checkoutservice         1/1     1            1           104s
currencyservice         1/1     1            1           103s
emailservice            1/1     1            1           104s
frontend                1/1     1            1           103s
loadgenerator           1/1     1            1           103s
paymentservice          1/1     1            1           103s
productcatalogservice   1/1     1            1           103s
recommendationservice   1/1     1            1           103s
redis-cart              1/1     1            1           103s
shippingservice         1/1     1            1           103s
```

Figure 7: Deployed services

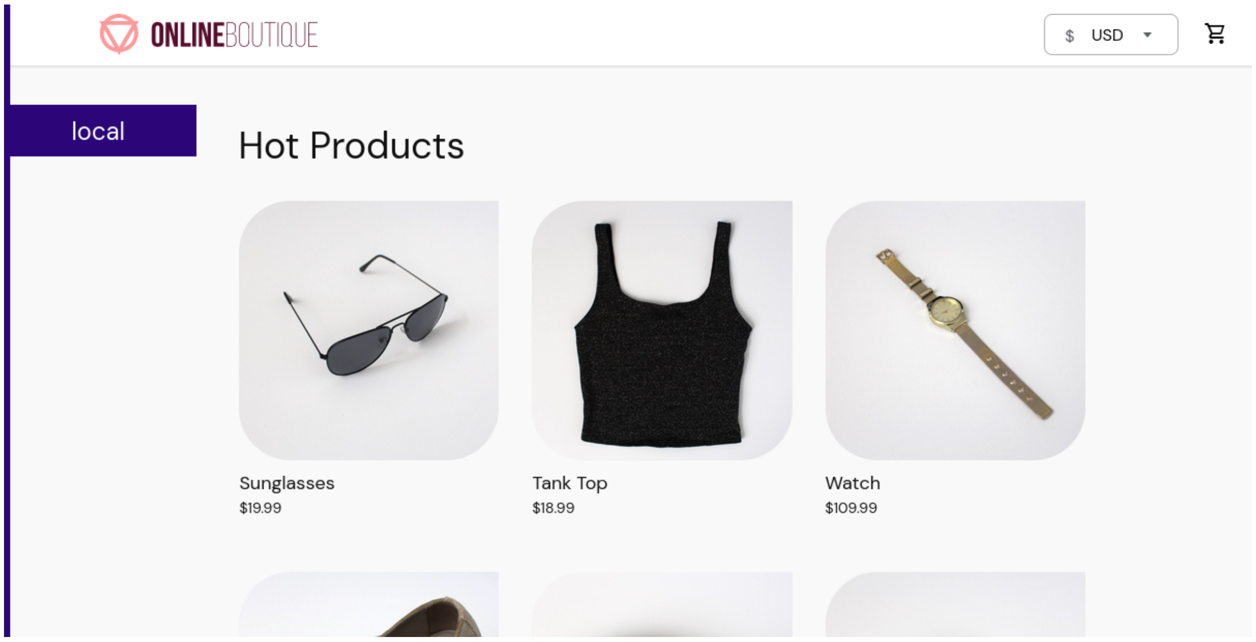


Figure 8: Application Interface (Through frontend service)

Install sysdig & Chisal Scripts

The deployment of sysdig on the Kubernetes cluster was tricky as it was not thoroughly explained as part of the paper. In addition, as sysdig is a free utility that to be used by anyone and deployed as system level application or as container through a cluster, it also has many commercial applications and features part of it.

1. Cloneing ReplicaWatcher repo into jumpbox to acquire chisels scripts

```
Git clone https:// https://github.com/utwente-scs/ReplicaWatcher.git
```

2. Sysdig image by default does not include ReplicaWatcher chisels scripts inside it, so we have to create a configMap to compose chisels script into the pods that will be deployed by daemonset

```
Kubectl create configMap Replicawatcher-chisels –from-file=chisels/*
```

3. created sysdig daemonset to deploy sysdig on every working node and monitor the pods activities on kernel level to get the generated events and save it locally on working nodes. Deployment includes all mounts that sysdig should have access to and also mount of configMap to mount chisels. It also includes the bash commands we used to auomate the collection process.

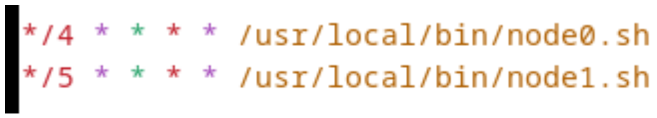
```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: sysdig
  labels:
    app: sysdig
spec:
  selector:
    matchLabels:
      name: sysdig
  template:
    metadata:
      labels:
        name: sysdig
    spec:
      volumes:
      - name: containerd-sock
        hostPath:
         path: /var/run/containerd/containerd.sock
         type: Socket
      - name: dev-vol
        hostPath:
         path: /dev
      - name: proc-vol
        hostPath:
         path: /proc
      - name: boot-vol
        hostPath:
         path: /boot
      - name: modules-vol
```

```
        hostPath:
         path: /lib/modules
      - name: usr-vol
        hostPath:
          path: /usr
      - name: chisels
        configMap:
          name: replicawatcher-chisels
      - name: sysdig-collection
        hostPath:
          path: /root/sysdig/collection
      - name: tmp-vol
        hostPath:
          path: /tmp
      hostNetwork: true
      hostPID: true
      containers:
      - name: sysdig
        image: sysdig/sysdig:0.38.1
        command: ["/bin/bash","-c","cd /host/tmp/; while true; do sysdig -M 10 -c
replicawatcher.lua 'cartservice sysdig_capture.json';sleep 590;done"]
        securityContext:
         privileged: true
        env:

        volumeMounts:
        - mountPath: /host/var/run/containerd/containerd.sock
          name: containerd-sock
          readOnly: false
        - mountPath: /host/dev
          name: dev-vol
          readOnly: false
        - mountPath: /host/proc
          name: proc-vol
          readOnly: true
        - mountPath: /host/boot
          name: boot-vol
          readOnly: true
        - mountPath: /host/lib/modules
          name: modules-vol
          readOnly: true
        - mountPath: /host/usr
          name: usr-vol
```

```
  readOnly: true
- mountPath: /host/tmp
  name: tmp-vol
  readOnly: false
- mountPath: /usr/share/sysdig/chisels
  name: chisels
  readOnly: false
```

Install Replicawatcher tool

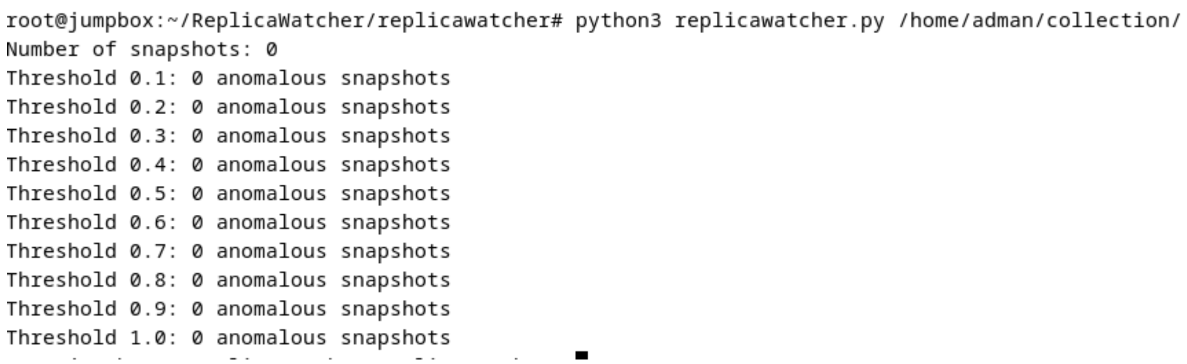
1. Collection of generated logs to monitoring node (jumpbox) through cronjob scripts to collection directory /home/adman/collection/



```
*/4 * * * * /usr/local/bin/node0.sh
*/5 * * * * /usr/local/bin/node1.sh
```

Figure 9: Collection cronjobs from both nodes

2. Running the detection scripts on the collected logs
   python3 replicawatcher.py /home/adman/collection



```
root@jumpbox:~/ReplicaWatcher/replicawatcher# python3 replicawatcher.py /home/adman/collection/
Number of snapshots: 0
Threshold 0.1: 0 anomalous snapshots
Threshold 0.2: 0 anomalous snapshots
Threshold 0.3: 0 anomalous snapshots
Threshold 0.4: 0 anomalous snapshots
Threshold 0.5: 0 anomalous snapshots
Threshold 0.6: 0 anomalous snapshots
Threshold 0.7: 0 anomalous snapshots
Threshold 0.8: 0 anomalous snapshots
Threshold 0.9: 0 anomalous snapshots
Threshold 1.0: 0 anomalous snapshots
```

Figure 10: Running Replicawatcher on collected logs

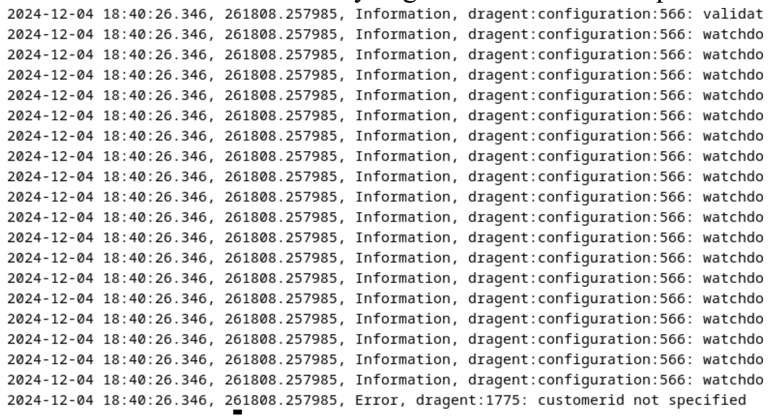
- ## Execution Steps:

- ## Troubleshooting:

Challenge	Status / Resolution
1. Initial we built the application with different accounts at google cloud (GKE) but it is costly (cost around 40 $ in two days)	Resolved

	• Moving to self-hosting Kubernetes
2. All images and application packages in Kubernetes-the-hard-way was aarch64 which is not compatible with AMD	Resolved • Locate all corresponding amd packages. • Insert them into the automation process
3. Kubernetes-the-hard-way is missing the appropriate steps in modifying IP address range which results in misconfigurations of PKI	Resolved • Modification of IP address in configuration file • Rebuild PKI
4. Kubernetes-the-hard-way dose not include DNS configuration that result in application services communication issues	Resolved • Configure complete DNS
5. Working Nodes does not include configuration needed for DNS resolution	Resolved • Loading br_netfilter kernel module
6. Kubernetes-the-hard-way dose not include permanent configuration of static routes	Resolved • Configured static route permanently in interfaces file
7. Paper includes only the chisels filtering code and anomaly detection script without any proper details about the structure of the solution	Resolved • Trying many agent types and different configurations then construct an architecture for collection depending on the information provided in paper and github.
8. Paper mentions the use of sysdig, however sysdig is a commercial company that has many products	Resolved • Used sysdig the opensource utility which is mentioned in replicwatcher github
9. Paper dose not include the daemonset configuration of sysdig	Resolved

	• We created daemonset based on sysdig github repo script and modified the running command to run to run the collection command
10. Paper dose not include how chisels is integrated with sysdig daemonset	Resolved We created configMap to include the chisels scripts in container in order to run the sysdig with scripts
11. Paper mentions the data is collected to an anlyzing host where the replicawatcher.py is ran however nothing about the collection method used.	Resolved • We used cronjob to transfer the output files to monitoring station.
12. Paper dose not include the details of attacks used	Resolved • We modeled the attacked based on given CWEs

13. Collected json capture of sysdig are empty	Not Resolved
	
	We suspect that sysdig have many agents, many of which (sysdig/sysdig or sysdig/agent) sysdig/sysdig is the opensource version which collects the raw capture of system calls and enable the usage of chisels. On the other hand, sysdig/agent enable k8s metadata enrichment which enable the system calls to be mapped in Kubernetes details such as (pod_name, services,… etc). Upon trying the sysdig/agent an error raised that it is missing the CusomerID which enable the service to be integrated with backend sysdig products.
	
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: validat`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Information, dragent:configuration:566: watchdo`
	`2024-12-04 18:40:26.346, 261808.257985, Error, dragent:1775: customerid not specified`

	Which means in order to have replicawatcher running we need the chisels to run and the chisels can run only if the output of sysdig is parsed against Kubernetes details.

7.2 Code Understanding

• Annotated Code Walkthrough:

- Sysdig

Sysdig captures system calls and other operating system events by deploying a kernel module or eBPF probe, which intercepts these calls directly from the Linux kernel. This approach allows Sysdig to monitor system activities in real-time, providing comprehensive insights into process behaviors, file operations, and network interactions. https://github.com/draios/sysdig?utm_source=chatgpt.com

- Chiesl scripts

In the ReplicaWatcher project, custom Chisel scripts are developed to capture relevant events from replicas, focusing on system calls that reflect operational states and send it to replicawathcher in json format .

- Replicawatcher

This data is then fed into ReplicaWatcher anomaly detection logic, which compares the behavior of identical replicas to identify deviations. By assuming that replicas performing the same tasks should exhibit similar behaviors, ReplicaWatcher analyzing discrepancies among replicas performing similar functions, it identifies anomalies without relying on pre-established baselines

Replicawatcher Logic(how it detect anomaly in technical sound)

Replicawatcher designed to detect anomalies in replicas

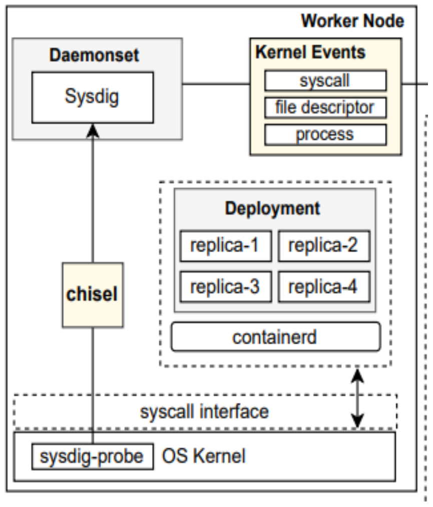
The logic of Replicawatcher consist of three main stages:

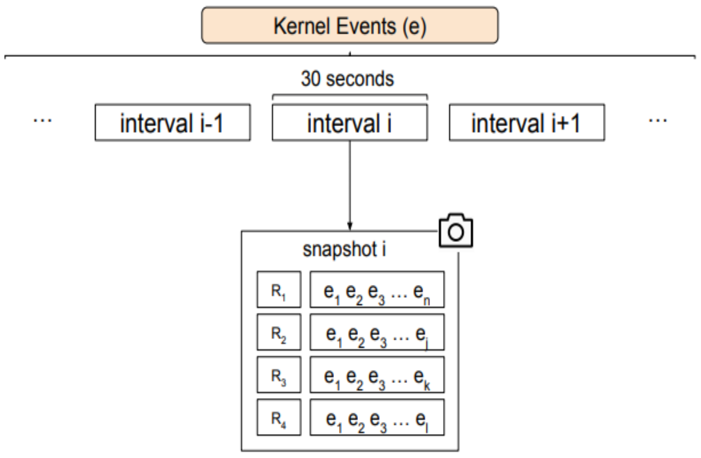
- **Event Chunking**
- **Event Encoding**,
- **Anomaly Detection**.

Each stage uses specific algorithms and techniques to process replica activity data without relying on a training phase.

Event Chunking

In this stage, Replicawatcher uses the captured syscalls from Sysdig with a customized Chisel script to analyze kernel-level events. These events include filesystem interactions, executed commands, and system calls. The captured events are divided into time-bound snapshots, each describing the activities of replicas during a predefined interval (τsnapshot)




Kernel Events (e)
30 seconds
...
interval i-1
interval i
interval i+1
...
snapshot i
R_1 $e_1\, e_2\, e_3 \dots e_n$
R_2 $e_1\, e_2\, e_3 \dots e_j$
R_3 $e_1\, e_2\, e_3 \dots e_k$
R_4 $e_1\, e_2\, e_3 \dots e_l$

- Initialization callback to initializes the event fields and sets filters for monitoring a specific deployment

```
function on_init()
   f_pod_name = chisel.request_field(fields.pod_name)
   f_timestamp = chisel.request_field(fields.timestamp)
   f_syscall_name = chisel.request_field(fields.syscall_name)
   f_syscall_category = chisel.request_field(fields.syscall_category)
   f_file_path = chisel.request_field(fields.file_path)
   f_file_name = chisel.request_field(fields.file_name)
   f_file_directory = chisel.request_field(fields.file_directory)
   f_file_operation = chisel.request_field(fields.file_operation)
   f_proc_name = chisel.request_field(fields.proc_name)
   f_proc_cmdline = chisel.request_field(fields.proc_cmdline)
   f_proc_cwd = chisel.request_field(fields.proc_cwd)
   f_proc_exe = chisel.request_field(fields.proc_exe)
   f_proc_args = chisel.request_field(fields.proc_args)

   -- Set filter
   deployment_filter = "k8s.deployment.name=" .. deployment
   chisel.set_filter(deployment_filter)

   return true
end
```

- Event parsing callback. Captures and processes each kernel event, extracting details such as pod name, timestamp, and syscall name.

```
function on_event()
    table.insert(list_of_pods, evt.field(f_pod_name))
    table.insert(events, {
        evt.field(f_pod_name),
        evt.field(f_timestamp),
        evt.field(f_syscall_name),
        evt.field(f_syscall_category),
        evt.field(f_file_path),
        evt.field(f_file_directory),
        evt.field(f_file_name),
        evt.field(f_file_operation),
        evt.field(f_proc_name),
        evt.field(f_proc_cmdline),
        evt.field(f_proc_cwd),
        evt.field(f_proc_exe),
        evt.field(f_proc_args)
    })
    return true
end
```

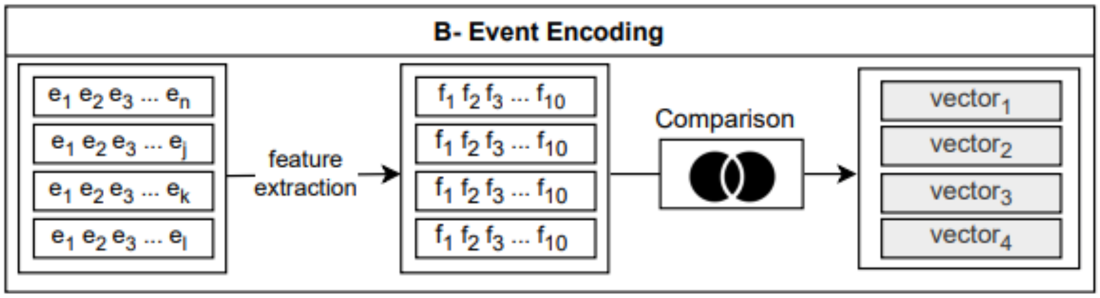
- Capture end callback to Encodes the accumulated events into JSON format and writes the output to a specified file.

```
function on_capture_end()
    local result = parse.parse(events, list_of_pods)
    local success, replicasEvents = pcall(json.encode, result, { indent = true })
    if not success then
        print("Error encoding JSON: " .. replicasEvents)
        return false
    end

    --print(replicasEvents)

    -- Writing to the output file
    local success, err = cutils.write_to_file(output_file, replicasEvents)
    if not success then
        print(err)
        return false
    else
        print("File '" .. output_file .. "' has been successfully generated.")
    end
    return true
```

```
end
```

Event Encoding

This stage processes the captured events from the previous stage and encodes them into structured representations. The primary goal is to extract meaningful patterns by defining disimlirity scores from raw data that allow replicas to be compared effectively and to ensures that Replicawatcher can identify anomalies while minimizing false positives.


B- Event Encoding
$e_1\ e_2\ e_3 \dots e_n$
$e_1\ e_2\ e_3 \dots e_j$
$e_1\ e_2\ e_3 \dots e_k$
$e_1\ e_2\ e_3 \dots e_l$
feature extraction
$f_1\ f_2\ f_3 \dots f_{10}$
$f_1\ f_2\ f_3 \dots f_{10}$
$f_1\ f_2\ f_3 \dots f_{10}$
$f_1\ f_2\ f_3 \dots f_{10}$
Comparison
vector_1
vector_2
vector_3
vector_4

The stage will take the captured json file to multiple steps to process it :

- Preprocess involves extracting key attributes from the raw event data for subsequent analysis, such as syscalls, file operations, and process metadata. This step consider as feature selection to ensure the data filtered from noise , duplicate features , and normalize paths.

```
def preprocess_data(data):
aspects = ["syscalls", "directories", "filenames"]
for replica in data:
for key in aspects:
 if key in replica: replica[key] = list(set(replica[key]))
```

- Dissimilirity caculation :in this step Replicawatcher ReplicaWatcher calculates dissimilarity scores for each feature using **Jaccard Similarity**. This score measures how similar the features of a replica to other replicas.

  Jaccard Similarity Formula:

$$DS_{f_k}(R_i) = 1 - \frac{1}{n-1} \sum_{j=1\ j\neq i}^{n} \frac{|R_{i_{f_k}} \cap R_{j_{f_k}}|}{|R_{i_{f_k}} \cup R_{j_{f_k}}|} \qquad (3)$$

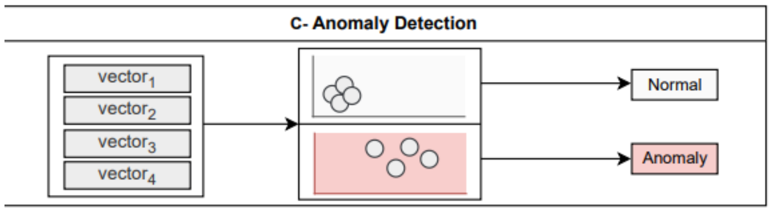
  Here how it is implemented in python: **utils.py**

```
def jaccard_similarity(set1, set2):
    intersection = set1.intersection(set2)
    union = set1.union(set2)
    return len(intersection) / len(union)

def dissimilarity(current_replica_set, other_replica_sets):
    if not current_replica_set:
        return 0
    similarity_scores = [jaccard_similarity(current_replica_set, s) for s in
other_replica_sets]
    avg_similarity_score = sum(similarity_scores) / len(similarity_scores)
    return 1 - avg_similarity_score
```

- Vector representation : then For each replica, the dissimilarity scores of all features are combined into a **dissimilarity vector**:

```
def calculate_replicas_vectors(self, data):
        """Calculate vectors for each replica based on the pre-defined aspects."""
        num_replicas = len(data)
        replicas_vectors = np.zeros((num_replicas, len(ASPECTS)))

        for i, aspect in enumerate(ASPECTS):
            replica_sets = self.calculate_replicas_sets(data, aspect)
            for j, current_set in enumerate(replica_sets):
                other_sets = replica_sets[:j] + replica_sets[j+1:]
                replicas_vectors[j, i] = dissimilarity(current_set, other_sets)

        return replicas_vectors
```

Anomaly detection



This is the final stage to give decision on the replicas behavior by evaluating the dissimilarity vectors. It assumes that normal replicas have similar behaviors, and anomalies are identified as replicas whose vectors deviate significantly from the expected pattern, as measured by their distance from the origin of the vector space or from other replicas.

At this stage Replicawatcher will decide the legitimately of the behavior based on the following:

1- Encoded each replicas into a dissimilarity vector where each element corresponds to the dissimilarity score for a specific feature such as syscalls, directories accessed.
2- Using Euclidean distance to measure the deviation of a replica's dissimilarity vector from the origin (representing normal behavior) where a higher distance indicates greater deviation from expected behavior, suggesting an anomaly.

3- Classifying replicas as anomalous or normal, we apply a threshold to the calculated distances.

```
def detect_anomalies(self, replicas_vectors, threshold):
        """Detect anomalies in the replicas vectors based on a threshold."""
        centroid = np.zeros((1, replicas_vectors.shape[1]))
        distances = cdist(replicas_vectors, centroid, 'euclidean').flatten()
        return np.any(distances > threshold)

    def run_detection(self):
        """Run the anomaly detection process for all JSON files in the folder."""
        print(f"Number of snapshots: {len(self.json_files)}")
        for threshold in np.arange(0.1, 1.1, 0.1):
            files_with_anomalies = sum(self.process_file(f, threshold) for f in self.json_files)
            self.num_files_with_anomalies[threshold] = files_with_anomalies
            print(f"Threshold {threshold:.1f}: {files_with_anomalies} anomalous snapshots")
```

• Extensions or Insights:

– Propose potential improvements, such as:

- Optimizing memory usage.
- Enhancing detection accuracy.
- Adding additional functionality to handle edge cases or improve runtime performance.

ReplicaWatcher significantly enhances detection accuracy through its innovative, training-less anomaly detection approach. Here is how:

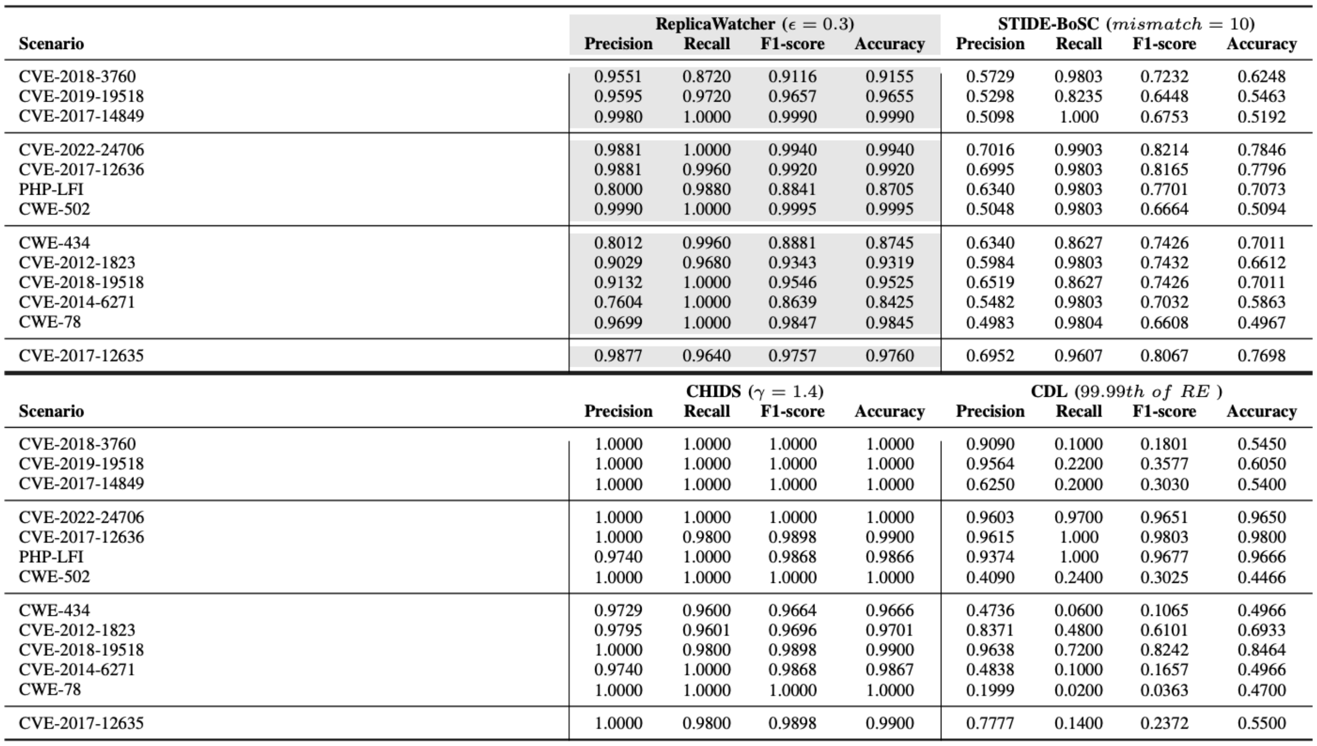
- Elimination of Training Dependence: Unlike state of the art anomaly detection systems for containerized microservice environment, ReplicaWatcher does not rely on pre-trained baselines, which are prone to obsolescence in dynamic environments. Instead, it leverages runtime comparisons of identical replicas to detect deviations, eliminating the need for retraining and ensuring robust performance over time.

- Replica Behavior Analysis: The approach assumes that replicas, under normal operations, exhibit similar behavior. By monitoring their activities, anomalies caused by attacks or misconfigurations stand out as deviations, which are promptly flagged.

- Noise Management: The system is designed to handle operational noise, such as fluctuations in user interactions or workloads, by focusing on features resilient to these variabilities. This ensures a low rate of false positives.

- Efficiency and Scalability: ReplicaWatcher imposes minimal processing overhead, leveraging carefully selected features such as syscalls, file operations, and process metadata to maintain a balance between computational efficiency and detection capabilities. This makes it well-suited for high-traffic, real-world deployments.

- Superior Detection Metrics: Achieving an average precision of 91.08% and a recall of 98.35%, ReplicaWatcher matches the performance of leading training-based solutions without the challenges of model maintenance and retraining.



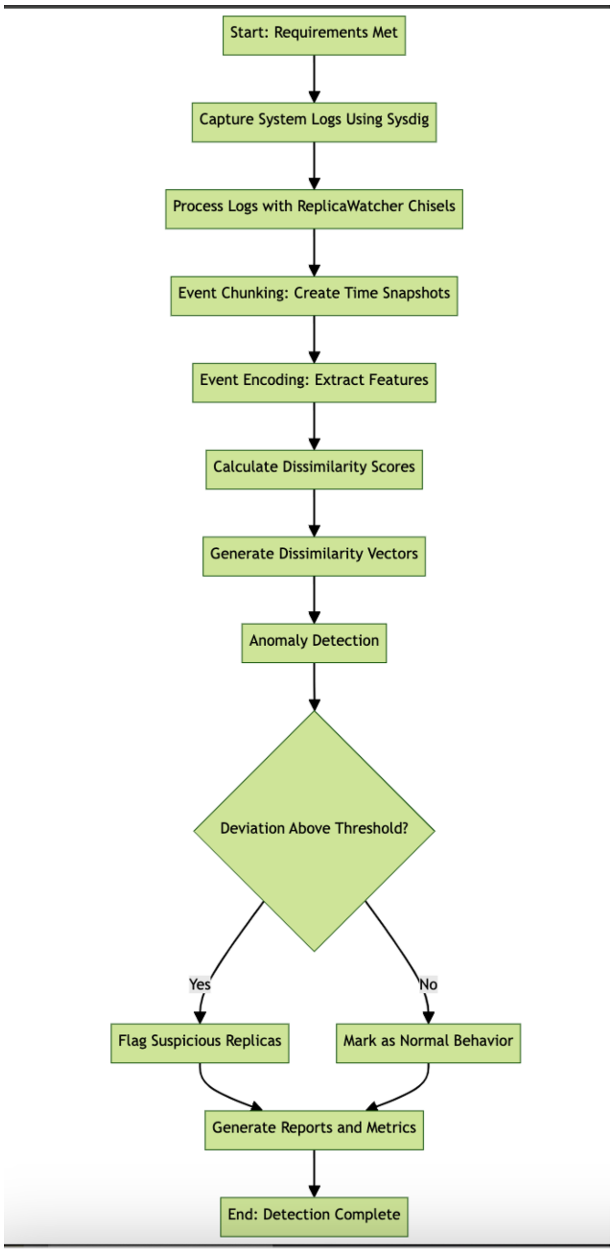
	ReplicaWatcher ($\epsilon = 0.3$)				**STIDE-BoSC** ($mismatch = 10$)			
Scenario	**Precision**	**Recall**	**F1-score**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Accuracy**
CVE-2018-3760	0.9551	0.8720	0.9116	0.9155	0.5729	0.9803	0.7232	0.6248
CVE-2019-19518	0.9595	0.9720	0.9657	0.9655	0.5298	0.8235	0.6448	0.5463
CVE-2017-14849	0.9980	1.0000	0.9990	0.9990	0.5098	1.000	0.6753	0.5192
CVE-2022-24706	0.9881	1.0000	0.9940	0.9940	0.7016	0.9903	0.8214	0.7846
CVE-2017-12636	0.9881	0.9960	0.9920	0.9920	0.6995	0.9803	0.8165	0.7796
PHP-LFI	0.8000	0.9880	0.8841	0.8705	0.6340	0.9803	0.7701	0.7073
CWE-502	0.9990	1.0000	0.9995	0.9995	0.5048	0.9803	0.6664	0.5094
CWE-434	0.8012	0.9960	0.8881	0.8745	0.6340	0.8627	0.7426	0.7011
CVE-2012-1823	0.9029	0.9680	0.9343	0.9319	0.5984	0.9803	0.7432	0.6612
CVE-2018-19518	0.9132	1.0000	0.9546	0.9525	0.6519	0.8627	0.7426	0.7011
CVE-2014-6271	0.7604	1.0000	0.8639	0.8425	0.5482	0.9803	0.7032	0.5863
CWE-78	0.9699	1.0000	0.9847	0.9845	0.4983	0.9804	0.6608	0.4967
CVE-2017-12635	0.9877	0.9640	0.9757	0.9760	0.6952	0.9607	0.8067	0.7698

	CHIDS ($\gamma = 1.4$)				**CDL** ($99.99th\ of\ RE$)			
Scenario	**Precision**	**Recall**	**F1-score**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Accuracy**
CVE-2018-3760	1.0000	1.0000	1.0000	1.0000	0.9090	0.1000	0.1801	0.5450
CVE-2019-19518	1.0000	1.0000	1.0000	1.0000	0.9564	0.2200	0.3577	0.6050
CVE-2017-14849	1.0000	1.0000	1.0000	1.0000	0.6250	0.2000	0.3030	0.5400
CVE-2022-24706	1.0000	1.0000	1.0000	1.0000	0.9603	0.9700	0.9651	0.9650
CVE-2017-12636	1.0000	0.9800	0.9898	0.9900	0.9615	1.000	0.9803	0.9800
PHP-LFI	0.9740	1.0000	0.9868	0.9866	0.9374	1.000	0.9677	0.9666
CWE-502	1.0000	1.0000	1.0000	1.0000	0.4090	0.2400	0.3025	0.4466
CWE-434	0.9729	0.9600	0.9664	0.9666	0.4736	0.0600	0.1065	0.4966
CVE-2012-1823	0.9795	0.9601	0.9696	0.9701	0.8371	0.4800	0.6101	0.6933
CVE-2018-19518	1.0000	0.9800	0.9898	0.9900	0.9638	0.7200	0.8242	0.8464
CVE-2014-6271	0.9740	1.0000	0.9868	0.9867	0.4838	0.1000	0.1657	0.4966
CWE-78	1.0000	1.0000	1.0000	1.0000	0.1999	0.0200	0.0363	0.4700
CVE-2017-12635	1.0000	0.9800	0.9898	0.9900	0.7777	0.1400	0.2372	0.5500

- Adaptability to Shifting Norms: The system is inherently resilient to normality shifts, such as those caused by software updates, which can often degrade the performance of conventional systems. ReplicaWatcher adapts effortlessly, maintaining consistent detection accuracy without manual recalibration.

• Flow Diagram:

high-level flow diagram of the code’s workflow:


Start: Requirements Met
Capture System Logs Using Sysdig
Process Logs with ReplicaWatcher Chisels
Event Chunking: Create Time Snapshots
Event Encoding: Extract Features
Calculate Dissimilarity Scores
Generate Dissimilarity Vectors
Anomaly Detection
Deviation Above Threshold?
Yes
No
Flag Suspicious Replicas
Mark as Normal Behavior
Generate Reports and Metrics
End: Detection Complete

7.3 Input and Output Demonstrations

• Detection Test Cases:

The objective of the detection test is to simulate and detect attacks using ReplicaWatcher. Each attack will introduce an anomalies which suppose to trigger detection mechanisms.

Test Case 1 (CWE-89 SQL Injection):

Description:
Preparing an exploitation on Google online bootique input fields with malicious SQL commands to bypass authentication and access the database that contain a sensitive data. ReplicaWatcher should detects anomalies in database syscalls like connect, read, and write, flagging unusual patterns such as sudden SQL surges or unexpected table access. Sqlmap and Python requests will be used to craft HTTP requests manually if needed.

Execution:

1. Identify a vulnerable input field (e.g., login form, search bar). Use sqlmap to automate the SQL injection attack (bash):

```
sqlmap -u "http://<your_service_url>/login" --
    data="username=admin&password=pass" --dbs
```

- Use the --dbs flag to list databases after successful exploitation.

Test Case 2 (CWE-307 Brute Force):

Description:
Preparing an exploitation on Google online boutique which is brute force attack such as using automated tools "Hydra" to repeatedly attempt logins and crack passwords through rapid combination testing. ReplicaWatcher should detects the high-frequency failed login syscalls like open, close, and recvfrom, which will detect the abnormal calls in authentication patterns across replicas. A passwords list will be used to conduct this attack.

Execution:

1. Use a brute-force tool "Hydra":

```
hydra -l admin -P /path/to/passwords.txt http-post-form
"/login:username=^USER^&password=^PASS^:F=invalid credentials"
```

   - Replace /login with the endpoint of the login service.
   - Use a dictionary (passwords.txt) with common weak passwords.

Test Case 3 Directory Traversal (CVE-2019-5418, CVE-2018-3760, CVE-2017-14849):

Description:

Preparing an exploitation on Google online boutique which is a directory path vulnerabilities to access files outside intended directories, such as /etc/passwd. Anomalies in file access syscalls like open and read are flagged when accessing files outside the application's root, which will triggering alerts for sensitive directories like /etc or /var.

Execution:

1. Send a crafted HTTP request with manipulated headers:
   ```
   curl -X GET "http://<your_service_url>" -H "Accept: ../../../../etc/passwd"
   ```

2. Exploit the file parameter in a vulnerable API:
   ```
   curl "http://<your_service_url>/readfile?path=../../../../etc/passwd"
   ```

Test Case 4 (File Inclusion CVE-2022-24706):

Description:
Preparing an exploitation by Injecting malicious files through Google online boutique uploads to execute arbitrary code or escalate privileges. ReplicaWatcher should detects abnormal file uploads and execution patterns through syscalls like execve, open, and fopen, which will flagging sudden execution of user-uploaded files as anomalous.

Execution:

1. Upload a malicious PHP file:
   ```
   <?php
   system($_GET['cmd']);
   ?>
   ```

2. Access the uploaded file and execute commands (Bash):
   ```
   curl "http://<your_service_url>/uploads/shell.php?cmd=ls"
   ```

Test Case 5 (Remote Code Execution CWE-502):

Description:
Preparing an attack by exploiting insecure deserialization to inject malicious commands in Google online boutique, such as Python Pickle library RCE with crafted payloads. New processes from unexpected sources will be flagged as anomalous by ReplicaWatcher, with syscalls like execve and fork linked to unrecognized commands.

Execution:

1. Create a malicious Python pickle payload:
```
import pickle
payload = pickle.dumps({"__reduce__": (os.system, ("whoami",))})
with open("malicious.pkl", "wb") as f:
f.write(payload)
```

2. Exploit the vulnerable service by uploading or sending the payload.

Test Case 6 (Command Injection CVE-2012-1823, CWE-78):

Description:
Preparing an exploitation by Injecting shell commands through vulnerable APIs or input fields on Google online boutique, such as passing cat /etc/passwd to a compromised endpoint. ReplicaWatcher should detects abnormal syscall patterns, especially for execve and sh, which will triggering alerts of sudden spikes in command activity across replicas.

Execution:
1. Exploit a vulnerable API that uses shell commands (bash):
```
curl http://<your_service_url>/run?cmd=ls
```

2. Gradually escalate payload complexity to execute harmful commands (bash):
```
curl "http://<your_service_url>/run?cmd=cat+/etc/passwd"
```

Test Case 7 (Privilege Escalation CVE-2017-12635):

Description:
Preparing an exploitation by CouchDB misconfigurations to inject admin keys or escalate user privileges. Unauthorized access or manipulation of user roles is detected via database syscalls like connect and write, the ReplicaWatcher will flagging the replicas that showing inconsistent roles, such as unexpected admin behavior.

Execution:

1. Exploit CouchDB with default admin credentials (bash):
```
curl -X PUT "http://<couchdb_url>:5984/_users/org.couchdb.user:attacker" \
H "Content-Type: application/json" \
-d '{"name": "attacker", "roles": ["_admin"], "type": "user", "password":
"password"}'
```

2. Add an admin user or escalate privileges.