NEWLY DISCOVERED ROUTE TAKEOVER AND DNS HIJACKING ATTACKS IN OPENSHIFT

Luiza Nacshon[1] and Martin Ukrop[2]

[1]Senior Security Engineer, Red Hat, Israel
[2] Senior Technical Program Manager, Red Hat, Brno

ABSTRACT

OpenShift uses Route objects to expose web applications to the outside world through HAproxy. One of the challenges of managing web application routing in containerized environments such as OpenShift is securely transferring information and allowing access to the applications running in those environments. This paper will examine two possible attacks discovered during security research on OpenShift networking: Route takeover and DNS hijacking. While writing this paper, we didn't find related works discussing the attacks in containerized environments like Openshift. The novelty of the discovered attacks is the way those attacks are implemented and leveraged in the Openshift environment. The techniques used to gain route takeover and DNS hijacking can work only on Openshift clusters. Next, in the paper, we will briefly present and explain how users can prevent those possible attacks by following specific security practices.

KEYWORDS

Networking, Routes, Containerized Network, Hijacking, Network Security Policies, Route Takeover

1. INTRODUCTION

During our research on the security of OpenShift networking, we found two possible attacks that may occur due to misconfiguration or human error and may lead to route takeover and DNS hijacking attacks by internal attackers in OpenShift. The attacks presented in this paper are recent discoveries in the OpenShift environment. The Route takeover attack is a similar method to the traditional DNS takeover, where the attacker can take advantage of the DNS record of a dandling host (registered with a DNS record, but the host is not in use). When translating the DNS takeover to Route takeover in OpenShift, the attacker can take advantage of a CNAME record of a dangling route. The same issue occurs in DNS hijacking. In the traditional DNS hijacking attack, the attacker can manipulate DNS records to forward all the traffic to the server. When translating the DNS hijacking attack to the OpenShift environment, the attacker can take advantage of misconfiguration in the DNS policy in the cluster and manipulate traffic into the pod.

In this paper, we will go through the attacks, the proof of concept, and the security practices that OpenShift users should apply to prevent those attacks from happening.

First, we will briefly describe the OpenShift router, how OpenShift routes work, and how to configure routing for web applications or services running on the OpenShift cluster.

In an OpenShift v4 cluster, the OpenShift router is a layer 7 load balancer whose controller is registered with the default ingress subdomain for the cluster. A single HAProxy-based OpenShift router is created for each ingress subdomain. HAProxy [1] is an open-source, software-defined load balancer and proxy application. OpenShift uses an HAProxy package that is custom-built for OpenShift, using *dist-git/Brew* [2]. In OpenShift v4, HAproxy router settings can be configured using the API [3].

2. PRELIMINARIES

OpenShift uses HAproxy to route a URL associated with an application and proxies the request into the proper pod [4], where a pod is one (or more) containers deployed together on one host.

OpenShift uses the concept of routes to direct ingress traffic to containerized applications or services deployed on a pod. The containerized applications are deployed in an OpenShift pod. Routes are translated into HAproxy configurations [5] managed by the OpenShift Ingress Controller. If a user wants services and (by extension) pods to be accessible from the external world, the user needs to create a route. A route allows OpenShift users to host web applications at a public URL. It can be either secure or unsecured, depending on the network security configuration of the web application.

William Caban, a global telco chief architect [6], describes OpenShift in a very detailed way, which may help clarify what OpenShift is. An OpenShift namespace is a term used to describe a method to scope resources in a cluster, while projects group and isolate related objects and contain groups on namespaces. All namespaces in a project are based on the root namespace for the project. A namespace contains objects and services that will always contain the prefix of the namespace name in their routes. A service in OpenShift is a set of multiple running pods used to define consumable applications like a database or a microservice. A service has a dedicated IP address.

This paper will focus on Openshift terms; pod, service, namespace, and routing. A pod is a set of running containers; we can define a service as a logical set of pods. A service is an abstracted layer on top of the pod, which provides a single IP and DNS name through which pods can be accessed. The routing exposes the service to the external world by creating and configuring externally reachable hostnames. Routes and endpoints expose the service to the external world, where the user can use the name connectivity (DNS) to access defined applications.

The default OpenShift router HAProxy uses the HTTP header of the incoming request to determine where to proxy the connection. The HAProxy router needs to know which service the client wants to access. The default search domain for the pod will be *.<namespace>.cluster.local*, where the namespace is the location of the pod running the containerized web application. The router first forwards the namespace queries to the master nameserver, which is the default behavior for containerized environments.

The master nameserver will answer queries on the *.<namespace>.cluster.local*.

If the request fails, the router will look for the next nameserver answers with *<service>.<namespace>.svc.cluster.local*, which would resolve to the service address of a service on the X namespace.

If a user wants to expose a service externally, an OpenShift route allows it to associate a service with an externally reachable hostname. The defined hostname is then used to route traffic to the

service. If a hostname is not provided as part of the route definition, then OpenShift automatically generates a hostname of the form *<route-name>[-<namespace>].<suffix>*
(for example, nohostname-mynamespace.router.default.svc.cluster.local).

Pod's DNS Config allows users to control the DNS settings for a pod. DNS policies can be set on a per-pod basis. Using the pod specification, we can configure the *DNS policy (dnsPolicy)* field [7].

As we will explain more deeply in the following sections of the paper, weak management of DNS policies or DNS configurations may lead to route takeover or DNS hijacking attacks. We will also explain how the cluster superuser can prevent those attacks.

Both DNS takeover and DNS hijacking attacks are common attacks and may happen in different environments. Marco et al. [8] presented, the DNS takeover attack is increasingly frequent. In 2020, they found about 887 web applications vulnerable to DNS takeover. Also, the statistics gathered by Fireeyes [9] show a wave of DNS hijacking that has affected dozens of domains belonging to the government. Kaur et al. [10] show similar results. These reports confirm that DNS attacks are very common and dangerous attacks that may lead to data leakage or to threat campaigns that redirect all user’s data to the attacker’s host.

Liu et al. [13] present dangling DNS records as Dares; they address the possible threats that may accrue through the Dares, like full control of subdomains and usage of fake certificates signed. Another interesting research [14] described how attackers could leverage the dangling DNS records and perform phishing campaigns pretending to be the originally pointed sites in the CNAME record. Another related work [15,16] discussed advanced techniques used by attackers to hijack DNS records and showed that attacks against DNS infrastructures are growing.

Some related works also discussed the DNS threats that may happen in cloud environments and Kubernetes clusters. Satam et al. [18] show how DNS attacks in a cloud environment, like DNS hijacking, can lead to compromising user's cloud accounts and stored information. The other related works discuss security issues in Kubernetes networking [20,21,22]. Yang et al. [20] discuss the challenges of securing containerized environments in the cloud and how the complexity of containerized systems may increase the attack surface. Wong et al. [21] show the importance of securing the networking communication between the microservices to prevent unexpected DNS and networking attacks.

In addition, we reference a few related works discussing novel approaches to identifying and preventing DNS attacks. Jinyuan et al. [11] presented a graph-based approach using deep learning-based algorithms to detect and prevent DNS attacks. Rigved et al. [12] presented a novel framework to detect DNS takeover by utilizing the enterprise's inside information. Jia et al. [17] presenting a novel approach to detect DNS attacks using graph-based algorithms. In future work, we would like to test proposed solutions on OpenShift clusters and analyze the success of detecting and preventing DNS attacks in OpenShift clusters.

3. ROUTE TAKEOVER ATTACK IN OPENSHIFT

In this section, we will present how a user can create a custom route to define the external hostname for the web application. We will also show how weak management of the DNS configuration may lead to a route takeover attack by an internal malicious actor.

Table 1. Route Takeover Attack Conditions in OpenShift

	User	**Attacker**
Cluster	same cluster	same cluster
Tenant	same/different tenant	same/different tenant
User privileges	regular/super user	regular user
Project level	different/same project	different/same project
Namespace	different/same namespace	different/same namespace
Environment	on-prem	on-prem

3.1. Update DNS CNAME Record for Custom Routes in OpenShift

Once a custom route is created [Figure 1], the user may update the DNS provider by creating a canonical name (CNAME) record (if the user wants to expose this route externally). The CNAME record should point the custom domain to the OpenShift router as the alias.

If the CNAME is not removed when the route is deleted, we are dealing with a dangling route. A malicious internal actor may take advantage of this human error behavior and take over the route.

Let's first explain why we may use CNAME for the web application running in OpenShift. For example, we have a web application hosted on OpenShift that has a route with a long UR, e.g., myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com [Figure 2]. We want to give end users a shortened URL, e.g., foo.multicats.org, which hides the OpenShift URL structure and is easier to remember.

Now we need to redirect foo.multicats.org external DNS to the OpenShift route. For this purpose, we will use CNAME records in the DNS provider and host field in the application config [Figure 2]. Note that the OpenShift router must accept the route to be selected. Once the router is selected and known, an external DNS provider will use this router's hostname as the target for the CNAME record.

OpenShift is not controlling external DNS records. Therefore, it is up to users to control and ensure that once the router is deleted, they also remove its CNAME from the DNS zone and DNS provider. In the case of weak management, we are left with a dangling DNS CNAME record, which leads to a route takeover attack.

3.2. Attack Vector Description and Proof of Concept (POC)

In this subsection, we will describe how an internal attacker can take advantage of dangling routes and take over a route. For the POC, let's assume that our DNS provider is Google.

An OpenShift route is a way to expose a service by giving it an externally reachable hostname, such as http://www.multicast.org. A router can consume a defined route and the endpoints identified by its service to provide named connectivity that allows external clients to reach the applications.

We tested the attack on an OpenShift v4.10 cluster in our lab and used two development accounts: devuser1 and devuser2.

Our devuser1 is creating a project and application called myguestbook. As we can see in Figure 1, the steps are to create a project -> route -> application. Using the ‘oc expose svc myguestbook’ command, we expose the myguestbook web application to the public world, meaning that everyone can access the myguestbook web application by typing *http://myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com* in the browser [Figure 2].

```
%  oc new-project my-route-project
Now using project "my-route-project" on server "https://api.testclusterroute.lab.pnq2.cee.redhat.com:6443".
%  oc create deployment myguestbook --image=ibmcom/guestbook:v2
deployment.apps/myguestbook created
%  oc get pods
NAME                          READY   STATUS              RESTARTS   AGE
myguestbook-c884989f7-gj2fq   0/1     ContainerCreating   0          13s
%  oc expose deployment myguestbook --type="NodePort" --port=3000
service/myguestbook exposed
%  oc get svc
NAME          TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
myguestbook   NodePort   172.30.94.64   <none>        3000:32251/TCP   20s
%  oc expose svc myguestbook
route.route.openshift.io/myguestbook exposed
% oc get routes
NAME          HOST/PORT                                                                 PATH   SERVICES      PORT
myguestbook   myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com          myguestbook   3000
None
```

Figure 1. Deployment of myguestbook web application

```
$ curl myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com
<!DOCTYPE html>
<html>
        <head>
                <title>Guestbook - v2</title>
        </head>
        <body>
                <h1>Guestbook POC</h1>
        </body>
</html>
```

Figure 2. Browsing myguestbook application in a web browser

In Figure 3, we can see the spec file of the created route. The ‘host’ field is currently pointing to the URL of the myguestbook web application.

```
% oc get route -o yaml
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    annotations:
      openshift.io/host.generated: "true"
    creationTimestamp: "2022-06-15T08:43:52Z"
    labels:
      app: myguestbook
    name: myguestbook
    namespace: my-route-project
  spec:
    host: myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com
    port:
      targetPort: 3000
    to:
      kind: Service
      name: myguestbook
      weight: 100
    wildcardPolicy: None
  status:
    ingress:
    - conditions:
      - lastTransitionTime: "2022-06-15T08:43:52Z"
        status: "True"
        type: Admitted
      host: myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com
      routerCanonicalHostname: router-default.apps.testclusterroute.lab.pnq2.cee.redhat.com
      routerName: default
      wildcardPolicy: None
```

Figure 3. Description of myguestbook web application spec

Devuser1 wants the guestbook to be publicly accessible, with the more straightforward domain name xxxxx.com. Thus, devuser1 registers the myguestbook application route into the Google DNS provider and links a subdomain to the myguestbook application's route, such as:

foo.multicats.org -> CNAME ->
http://myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com/.
Now, devuser1 has registered the route and foo.multicats.org is a CNAME that points to *myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com*. That means now we can browse the myguestbook web application by browsing http://foo.multicats.org [Figure 4].

```
dig +short foo.multicats.org
Myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com.

$ curl foo.multicats.org
<!DOCTYPE html>
<html>
        <head>
                <title>Guestbook - v2</title>
        </head>
        <body>
                <h1>Guestbook POC</h1>
        </body>
</html>
```

Figure 4. browsing myguestbook web application through the shortened URL

Now, devuser1 does not need to use the myguestbook application anymore and decides to delete the application, its route, and the project [Figure 5].

Let's assume that devuser1 forgot to clear around the DNS records and remove the CNAME record pointing to the myguestbook route we have created for external access (in both HAProxy and in Google DNS provider). In such a case, we are dealing with a dangling route. This issue may lead to route takeover. In such a case, a malicious internal actor could potentially re-create the route's hosted zone and gain control via the still-active delegation belonging to the OpenShift user.

```
$ oc whoami
devuser1
$ oc get projects
NAME              DISPLAY NAME   STATUS
my-route-project                 Active
$ oc delete project my-route-project
project.project.openshift.io "my-route-project" deleted
$ oc get projects
No resources found
```

Figure 5. devuser1 deletes the project where myguestbook was running

Let us assume that devuser2 is a malicious user on the same cluster and a different project who was able to find out that there is a dangling route on the OpenShift cluster. It also has a CNAME on the Google DNS provider, which is pointing to the dangling route (there are many open source tools that attackers use to find dangling a CNAME, so this step is really simple).

Now, devuser2 can create a route on the dev account and point the host field of the route to "foo.multicats.com," which is the subdomain on the Google DNS provider pointing to the route deleted by devuser1 [Figure 6].

Using that domain, devuser2 can create phishing sites or malicious web activities using the takeover route [Figure 7], while the end user believes they are accessing devuser1's web application.

```
$ oc whoami
devuser2
$ oc get routes
No resources found in devuser2-project namespace.
$ oc expose svc myguestbook --hostname=foo.multicats.org
route.route.openshift.io/myguestbook exposed
$ oc get routes
NAME          HOST/PORT           PATH   SERVICES      PORT   TERMINATION   WILDCARD
myguestbook   foo.multicats.org          myguestbook   3000                 None
$ dig +short -t CNAME foo.multicats.org
myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com.
```

Figure 6. devuser2 takeover of devuser1's dangling route

```
$ curl foo.multicats.org
<!DOCTYPE html>
<html>    <head>
                  <title>Controlled by devuser2</title></head>
          <body>
                  <h1>Controlled by devuser2</h1></body>
           </html>
```

Figure 7. devuser2 controlling the web application created by devuser1

This issue typically won't affect OpenShift clusters installed on a cloud provider if the cloud provider is deleting the CNAME records (for example, Amazon deleting CNAME in route53). However, this may affect multi-tenant clusters and clusters with different accounts.

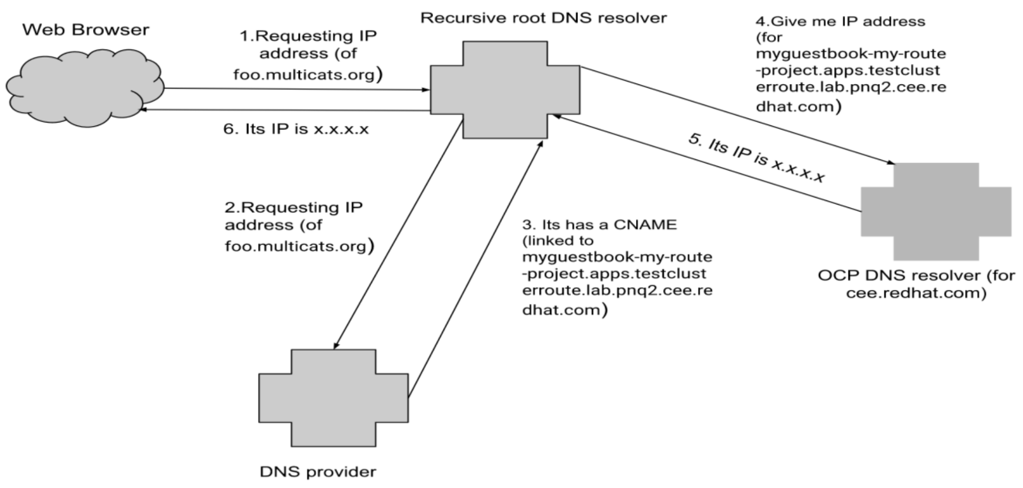
Web Browser
Recursive root DNS resolver
1.Requesting IP address (of foo.multicats.org)
6. Its IP is x.x.x.x
4.Give me IP address (for myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com)
5. Its IP is x.x.x.x
2.Requesting IP address (of foo.multicats.org)
3. Its has a CNAME (linked to myguestbook-my-route-project.apps.testclusterroute.lab.pnq2.cee.redhat.com)
OCP DNS resolver (for cee.redhat.com)
DNS provider

Figure 8. High-level description of why route takeover attack works

There are two issues for the OpenShift superuser to manage, as seen in Figure 8. The route was not removed from the DNS records when deleted. Also, there is still a live CNAME pointing to the deleted route.

There are possible ways in OpenShift to prevent the Route takeover attack from happening. The first step in deleting a route is removing the DNS record of unused applications. The cluster admin needs to prevent the deletion of a route and associated application in the case of a CNAME that was not deleted. The route owner needs to make sure to delete the CNAME record from the external DNS provider records. Also, there is an option to clear the DNS records cache periodically.

4. DNS HIJACKING ATTACK IN OPENSHIFT

Pods with the default DNS policy (`dnsPolicy`) set to "ClusterFirst" [Figure 9] or pods with host network and DNS policy "ClusterFirstWithHostNet" would have search paths like *<namespace>.svc.cluster.local*, service.namespace.svc.cluster.local and *cluster.local <cluster domain>* by default.

Table 2. DNS hijacking Attack Conditions in OpenShift

	User	**Attacker**
Cluster	same cluster	same cluster
Tenant	same tenant	same tenant
User privileges	regular/super user	regular user
Project level	different/same project	different/same project
Namespace	different/same namespace	different/same namespace
Environment	on-prem or cloud	on-prem or cloud
DNS Policy	ClusterFirst	N/A

DNS policies can be set on a per-pod basis. By default, a client pod's DNS search list includes the pod's own namespace and the cluster's default domain.

If dns Policy is not explicitly specified, "ClusterFirst" will be used by default.

```
$ oc get pods -o yaml
    terminationMessagePath: /dev/
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/
      name: default-token
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  imagePullSecrets:
  - name: default-dockercfg
  nodeName: test-node
  priority: 0
```

Figure 9. DNS settings are supposed to be provided using the dnsConfig field in the Pod Spec

The DNS policies are specified in the DNS policy field of a pod spec and have several options:

- "Default": The pod inherits the name resolution configuration from the node that the pods run on.
- "**ClusterFirst**": Any DNS query that does not match the configured cluster domain suffix is forwarded to the upstream nameserver inherited from the node.
- "**ClusterFirstWithHostNet**": For pods running with hostNetwork.
- "None": Allows a pod to ignore DNS settings from the OpenShift environment.

In our Proof of Concept (POC) [Figure 10], we can see that a malicious internal attacker adds a namespace called "org" with a service called "aisca2023."Pods configured with ClusterFirst DNS policy would look up "aisca2023.org" by trying *aisca2023.org.<namespace>.svc.cluster.local*, which would fail, and then *aisca2023.org.svc.cluster.local* which would resolve to the service address of the service from the "org" namespace, managed by the malicious attacker.

This issue happens because the superuser in the cluster does not properly configure the DNS policy for the pods and uses the ClusterFirst DNS policy. In such cases, those pods look into the internal cluster resolution domains before looking into DNS resolutions. Since the malicious internal user used “org” TLD for the namespace name and “aisca2023” for the service name on their “com” namespace, the pod configured with ClusterFirst DNS policy will look for "aisca2023.org.svc.cluster.local." The connection is accepted with the DNS resolution available in the cluster.

```
apiVersion: v1
kind: Namespace
metadata:
  name: org
apiVersion: v1
kind: Service
metadata:
  name: aisca2023
  namespace: org
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: fake-aisca2023
  type: ClusterIP
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: fake-aisca2023
  name: fake-aisca2023
  namespace: org
spec:
  containers:
  - args:
    - TCP4-LISTEN:8080,reuseaddr,fork,crlf
    - "SYSTEM:echo HTTP/0.9 200 OK ; echo ; echo You have reached Fake aisca2023."
    command:
    - /bin/socat
    image: openshift/origin-node
    name: fake-aisca2023
    ports:
    - containerPort: 8080
      protocol: TCP
```

Figure 10. aisca2023.org DNS hijacking POC

```
% oc create -f ~/src/openshift-examples/com-namespace-test.yaml
namespace/org created
service/aisca2023 created
pod/fake-aisca2023 created
% oc -n openshift-ingress rsh -c router deploy/router-default curl -s http://aisca2023.org/
You have reached Fake aisca2023.
```

Figure 11. innocent user browsing aisca2023.org will get the faked web page

The DNS hijacking attack in OpenShift may be prevented by correctly managing the DNS policies. The values of the search option in */etc/resolv.conf* are used to expand DNS queries.

```
nameserver 10.1.0.10
search <namespace>.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

In the case that /etc/resolv.conf contains expanded search, there will be a lookup for *aisca2023.<namespace>.svc.cluster.local* (where the namespace is org). Also, with the DNSPolicy set to "ClusterFirst," an internal, unauthorized user can forward all aisca2023.org into their pod.

The superuser of the OpenShift cluster should check and change the default DNS configuration to ensure that */etc/resolv.conf* does not contain expanded paths when the DNS policy is set to ClusterFirst. Another recommendation is to prevent namespaces called with TLDs.

5. CONCLUSIONS

Good management of DNS records and policies is important for securing OpenShift clusters. It is also important to clear all unused DNS records and deleted routes and to check all network and DNS policies defined per pod or for the cluster. In future work, we would like to test proposed research frameworks on the detection of the takeover and hijacking attacks in OpenShift clusters and analyze the success of detection and mitigation.

ACKNOWLEDGEMENTS

We would like to thank Thibault Guittet for providing a lab environment and helping to build a POC. Thanks to Fabio Leite and RaTasha Tillery for reviewing the paper.

REFERENCES

[1] www.haproxy.org/#docs
[2] https://pkgs.devel.redhat.com/cgit/rpms/haproxy/tree/haproxy.spec?h=rhaos-4.10-rhel-8
[3] https://github.com/openshift/api/blob/master/operator/v1/types_ingress.go
[4] https://docs.openshift.com/online/pro/architecture/core_concepts/pods_and_services.html
[5] /documentation/en-us/openshift_container_platform/4.7/html/networking/configuring-routes
[6] Caban, W. (2019). Architecting and Operating OpenShift Clusters: OpenShift for Infrastructure and Operations Teams. Apress.
[7] https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/
[8] Squarcina, M., Tempesta, M., Veronese, L., Calzavara, S., & Maffei, M. (2021). Can I Take Your Subdomain? Exploring {Same-Site} Attacks in the Modern Web. In 30th USENIX Security Symposium (USENIX Security 21) (pp. 2917-2934).
[9] Hirani, M., Jones, S., & Read, B. (2019). Global DNS hijacking campaign: DNS record manipulation at scale.
[10] Kaur, D., & Kaur, P. (2016). Empirical analysis of web attacks. Procedia Computer Science, 78, 298-306.
[11] Jia, J., Dong, Z., Li, J., & Stokes, J. W. (2021, June). Detection of malicious dns and web servers using graph-based approaches. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2625-2629). IEEE.
[12] Jayaprakash, Rigved, and Vishnu Kalariyil Venugopal. "A Novel Framework For Detecting Subdomain State Against Takeover Attacks." (2022).
[13] Liu, D., Hao, S., & Wang, H. (2016, October). All your dns records point to us: Understanding the security threats of dangling dns records. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 1414-1425).
[14] Baby, R. T., Ebenezer, V., & Karthik, N. Magnum Opus Of Phishing Techniques.
[15] Hudaib, A. A. Z., & Hudaib, E. A. Z. (2014). DNS advanced attacks and analysis. International Journal of Computer Science and Security (IJCSS), 8(2), 63.
[16] Braun, B. (2016). Investigating dns hijacking through high frequency measurements (Doctoral dissertation, UC San Diego).
[17] Jia, J., Dong, Z., Li, J., & Stokes, J. W. (2021, June). Detection of malicious dns and web servers using graph-based approaches. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2625-2629). IEEE.
[18] Satam, P., Alipour, H., Al-Nashif, Y., & Hariri, S. (2015, September). Dns-ids: Securing dns in the cloud era. In 2015 International Conference on Cloud and Autonomic Computing (pp. 296-301). IEEE.

[19] Andersen, M. F., Pedersen, J. M., & Vasilomanolakis, E. (2022, August). Detecting DNS hijacking by using NetFlow data. In 2022 IEEE conference on communications and network security, CNS 2022. IEEE Communications Society.

[20] Yang, Y., Shen, W., Ruan, B., Liu, W., & Ren, K. (2021, December). Security Challenges in the Container Cloud. In 2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA) (pp. 137-145). IEEE.

[21] Minna, F., Blaise, A., Rebecchi, F., Chandrasekaran, B., & Massacci, F. (2021). Understanding the security implications of kubernetes networking. IEEE Security & Privacy, 19(05), 46-56.

[22] Wong, A. Y., Chekole, E. G., Ochoa, M., & Zhou, J. (2021). Threat Modelling and Security Analysis of Containers: A Survey. arXiv preprint arXiv:2111.11475.