# FINAL PROJECT
# KUBERNETES FORENSICS

**JOHN C. FORD III (JOHN FORD)**

# Contents

# I. INTRODUCTION

In the summer of 2014, Google introduced the world to Kubernetes[1], a “lean yet powerful open-source container manager.” A year later, Kubernetes had its v1 release[2], and Google ceded control and donated the project to the newly formed Cloud Native Computing Foundation[3,4]. Kubernetes development and adoption accelerated rapidly in the following years. The tinkering and hacking community quickly found ways to have Kubernetes installed onto the Raspberry Pi, a small, System-on-Chip (SOC) computing device favored for its price point and low power requirements[5,6]. In the fall of 2017, Microsoft officially launched Azure Kubernetes Service, a managed version of Kubernetes available on its cloud platform[7]. The next year, in the summer of 2018, Amazon followed suit launching Elastic Kubernetes Service on the Amazon Web Services (AWS) cloud platform[8].

While the introduction of Kubernetes has resulted in many success stories[9], it also has many challenges. The Kubernetes homepage hosts an active feed of Common Vulnerabilities and Exposures (CVE)[10]. These CVEs contain information about potential privilege escalation, unauthorized access, and more[11,12,13]. With these challenges in mind,

---

1 https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html
2 https://github.com/kubernetes/kubernetes/releases/tag/v1.0.0
3 https://techcrunch.com/2015/07/21/as-kubernetes-hits-1-0-google-donates-technology-to-newly-formed-cloud-native-computing-foundation-with-ibm-intel-twitter-and-others/
4 https://www.cncf.io/
5 https://kubernetes.io/blog/2015/11/creating-a-raspberry-pi-cluster-running-kubernetes-the-shopping-list-part-1/
6 https://www.raspberrypi.com/products/
7 https://azure.microsoft.com/en-us/blog/introducing-azure-container-service-aks-managed-kubernetes-and-azure-container-registry-geo-replication/
8 https://aws.amazon.com/blogs/aws/amazon-eks-now-generally-available/
9 https://kubernetes.io/case-studies/
10 https://kubernetes.io/docs/reference/issues-security/official-cve-feed/
11 https://github.com/kubernetes/kubernetes/issues/121879
12 https://github.com/kubernetes/kubernetes/issues/113756
13 https://github.com/kubernetes/kubernetes/issues/101435

this project will examine a few of the tools and techniques that are available to monitor the activity and state, as well as means for capturing evidence from containers[14] that are hosted in a Kubernetes environment.

## II. PROJECT ENVIRONMENT

In order to experiment within a Kubernetes environment, I first needed to establish a cluster composed of one or more nodes[15]. A node is a machine, virtual or real hardware, that provides the resources that are needed to run workloads[16]. For this project, I chose to use several pieces of hardware that I already owned.

4 x Raspberry Pi 4B[17]

- Quad core ARM 64-bit SoC @ 1.8GHz
- 8 GB LPDDR4-3200 SDRAM
- 64 GB Samsung Fit Plus USB Storage[18]

3 x HP EliteDesk 800 G2 Mini[19]

- Quad core Intel Core i5 6500T @ 2.5GHz
- 16 GB DDR4-2133 RAM
- 256GB SSD

To provision each node, I downloaded the latest Ubuntu 20.04 Server Edition directly from the Ubuntu releases page[20], and I followed the quick tutorial that Ubuntu provides for installing the server OS[21]. At this stage, it was time to choose a distribution of Kubernetes to install. I chose to use k3s, a lightweight, single binary version of

---

[14] https://www.docker.com/resources/what-container/
[15] https://kubernetes.io/docs/concepts/overview/
[16] https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro
[17] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/
[18]
https://www.samsung.com/us/computing/memory-storage/usb-flash-drives/usb-3-1-flash-drive-fit-plus-64gb-muf-64ab-am/
[19] https://support.hp.com/us-en/product/product-specs/hp-elitedesk-800-35w-g2-desktop-mini-pc/7633266
[20] https://releases.ubuntu.com/focal/
[21] https://ubuntu.com/tutorials/install-ubuntu-server

Kubernetes[22]. The k3s project provides an easy to use, downloadable script to install and setup Kubernetes[23]. While the basic instructions will suffice for a single node installation, I had to modify the environment variables and arguments to suit my setup.

I started out by downloading the install script locally and modifying its permissions to make it executable.

```
curl -sfL  https://get.k3s.io > get-k3s.sh
chmod +x get-k3s.sh
```

With the install script downloaded, I proceeded to initialize the cluster on a single node that would serve as my starting point.

```
K3S_CLUSTER_SECRET=YourAmazingAndUniqueK3sToken \
INSTALL_K3S_VERSION=v1.28.3+k3s2 \
./get-k3s.sh server --cluster-init
```

Once the initial command completed, I proceeded to all the other nodes and connected them into the cluster by pointing to the first node.

```
K3S_CLUSTER_SECRET=YourAmazingAndUniqueK3sToken \
INSTALL_K3S_VERSION=v1.28.3+k3s2 \
./get-k3s.sh server --server https://<FIRST_SERVER_IP>:6443
```

I followed the instructions provided by the k3s team to retrieve the Kubernetes configuration from one of the nodes[24]. Using the *kubectl* CLI tool[25], I was able to check the status of the nodes.

[22] https://k3s.io/
[23] https://docs.k3s.io/quick-start
[24] https://docs.k3s.io/cluster-access
[25] https://github.com/kubernetes/kubectl

```
NAME           STATUS   ROLES                       AGE    VERSION
hp-mini-00     Ready    control-plane,etcd,master   231d   v1.28.3+k3s2
hp-mini-01     Ready    control-plane,etcd,master   230d   v1.28.3+k3s2
hp-mini-02     Ready    control-plane,etcd,master   12d    v1.28.3+k3s2
rpi4b-node-00  Ready    control-plane,etcd,master   12d    v1.28.3+k3s2
rpi4b-node-01  Ready    control-plane,etcd,master   12d    v1.28.3+k3s2
rpi4b-node-02  Ready    control-plane,etcd,master   12d    v1.28.3+k3s2
rpi4b-node-03  Ready    control-plane,etcd,master   12d    v1.28.3+k3s2
```

With all nodes reporting as *Ready*, I could begin my experiments.

## III. AUDITING API TRAFFIC

A Kubernetes installation is composed of several components[26]. A production installation will typically separate the duties of managing the cluster, referred to as the control plane, from running workloads, referred to simply as workers. For our project setup, because there are a small number of nodes and low resource usage, all nodes serve both functions. This means they contain all of the various components.

When sending any create, read, update, or delete (CRUD) request to change the state of the cluster, the request must first pass through the API server. The *kube-apiserver* component is what is responsible for performing this role.[27] The *kube-apiserver* coordinates with the *kube-scheduler* and *kubelet* components to schedule workloads onto the appropriate node that meets the workloads needs[28,29]. All of this traffic can be tracked using native Kubernetes audit logs[30].

Kubernetes audit logs are captured based on an audit policy which determines which events will be recorded[31]. Each request to the API server is composed of stages like *RequestReceived, ResponseStarted, ResponseComplete,* and *watch*, but every request won't necessarily contain all stages[32]. Stages and resource URLs can be ignored to avoid excessive logs.

---

26 https://kubernetes.io/docs/concepts/overview/components/
27 https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/
28 https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/
29 https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/
30 https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/
31 https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/#audit-policy
32
https://www.datadoghq.com/blog/monitor-kubernetes-audit-logs/#how-to-configure-kubernetes-audit-logs

```yaml
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
# do not log requests to the following
- level: None
  nonResourceURLs:
  - "/healthz*"
  - "/logs"
  - "/metrics"
  - "/swagger*"
  - "/version"

# limit level to Metadata so token is not included in the spec/status
- level: Metadata
  omitStages:
  - RequestReceived
  resources:
  - group: authentication.k8s.io
    resources:
    - tokenreviews

# extended audit of auth delegation
- level: RequestResponse
  omitStages:
  - RequestReceived
  resources:
  - group: authorization.k8s.io
    resources:
    - subjectaccessreviews

# log changes to pods at RequestResponse level
- level: RequestResponse
  omitStages:
  - RequestReceived
  resources:
  - group: "" # core API group; add third-party API services and your API
services if needed
    resources: ["pods"]
    verbs: ["create", "patch", "update", "delete"]

# log everything else at Metadata level
- level: Metadata
  omitStages:
  - RequestReceived
```

An example audit policy from the Datadog document on audit logs[33]

33 https://www.datadoghq.com/blog/monitor-kubernetes-audit-logs/#how-to-configure-kubernetes-audit-logs

Following the instructions provided by k3s[34], I proceeded to configure the initial node with the above audit policy.

```
sudo mkdir -p -m 700 /var/lib/rancher/k3s/server/logs
sudo nano /var/lib/rancher/k3s/server/policy.yaml
```

After pasting the policy contents and saving the file, I proceeded to modify the k3s service file to supply audit logs, appending the following lines to the *ExecStart*.

```
'--kube-apiserver-arg=audit-log-path=/var/lib/rancher/k3s/server
/logs/audit.log' \
'--kube-apiserver-arg=audit-policy-file=/var/lib/rancher/k3s/ser
ver/audit.yaml' \
```

Finally, I reloaded the service configs and restarted the k3s service.

```
sudo systemctl daemon-reload
sudo systemctl restart k3s.service
```

Firing a quick request for pods using *kubectl get pod*, I then went to go look for my event in the newly created audit log.

```
sudo cat /var/lib/rancher/k3s/server/logs/audit.log | grep
/api/v1/namespaces/default/pods
```

[34] https://docs.k3s.io/security/hardening-guide#api-server-audit-configuration

Here is an abbreviated form of the event that was recorded in the audit log.


```json
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "RequestResponse",
  "auditID": "601dd134-1ea0-4e6b-a94a-b1e6a87ac1fe",
  "stage": "ResponseComplete",
  "requestURI": "/api/v1/namespaces/default/pods?limit=500",
  "verb": "list",
  "user": {
    "username": "system:admin",
    "groups": [
      "system:masters",
      "system:authenticated"
    ]
  },
  "sourceIPs": [
    "192.168.1.177"
  ],
  "userAgent": "kubectl/v1.24.1 (linux/amd64) kubernetes/3ddd0f4",
  "objectRef": {
    "resource": "pods",
    "namespace": "default",
    "apiVersion": "v1"
  }
}
```


The log captured the requested resource, *pods,* the username used to make the request, *system:admin*, the source IP, *192.168.1.177* corresponding to my local machine, and the useragent, *kubectl*. With the flexibility provided by specifying an audit policy that is able to target all aspects of the Kubernetes API, a system administrator can be as fine grained as needed for their use case.

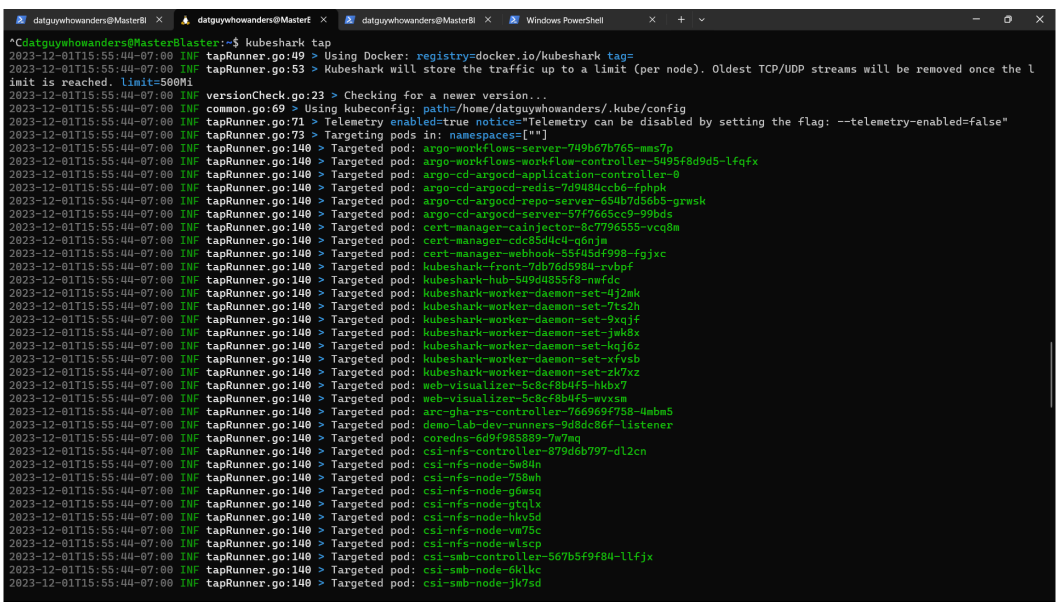
## IV. MONITORING NETWORK TRAFFIC

In addition to communication between Kubernetes components, there is network traffic being relayed in and out of the cluster to support running applications, and new tools are available to monitor and visualize that traffic. The Kubeshark project[35], inspired by Wireshark[36], provides real-time visibility into Kubernetes network traffic.

I installed Kubeshark by following the installation command provided on their GitHub releases page[37].

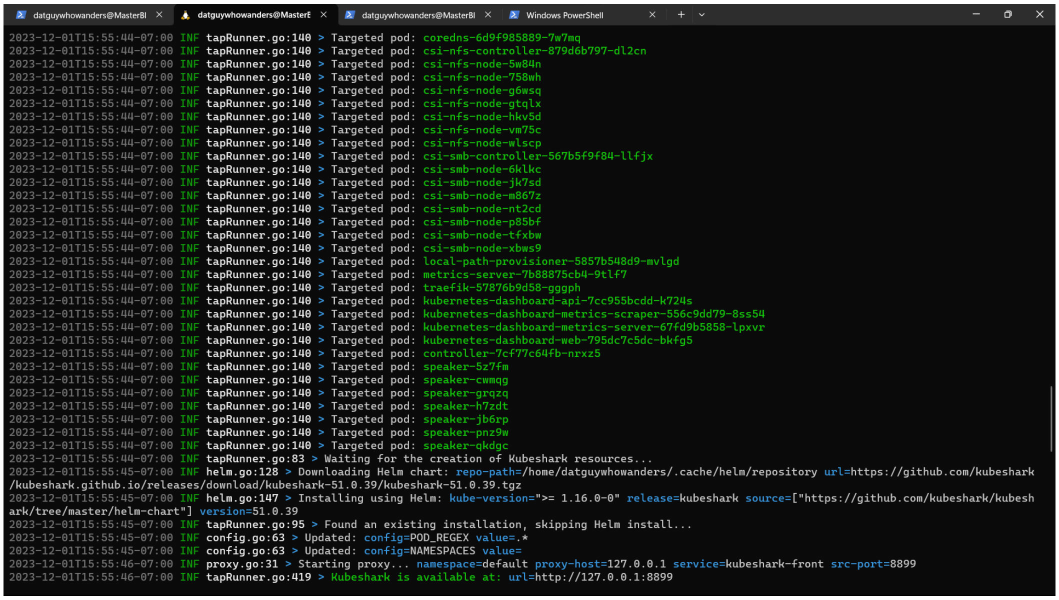
```
curl -Lo kubeshark
https://github.com/kubeshark/kubeshark/releases/download/v51.0.3
9/kubeshark_linux_amd64 && chmod 755 kubeshark
sudo mv kubeshark /usr/local/bin/
```

Using the *kubeshark* CLI for the first time, it uses the *helm* CLI[38] to install the resources it needs into the Kubernetes cluster.

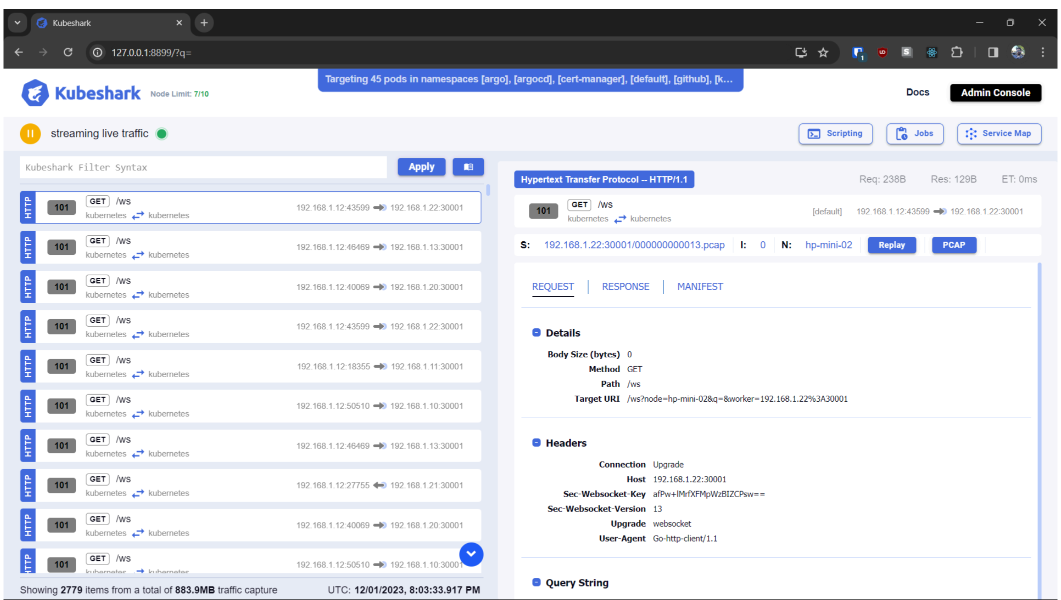


```
^Cdatguywhowanders@MasterBlaster:~$ kubeshark tap
2023-12-01T15:55:44-07:00 INF tapRunner.go:49 > Using Docker: registry=docker.io/kubeshark tag=
2023-12-01T15:55:44-07:00 INF tapRunner.go:53 > Kubeshark will store the traffic up to a limit (per node). Oldest TCP/UDP streams will be removed once the l
imit is reached. limit=500Mi
2023-12-01T15:55:44-07:00 INF versionCheck.go:23 > Checking for a newer version...
2023-12-01T15:55:44-07:00 INF common.go:69 > Using kubeconfig: path=/home/datguywhowanders/.kube/config
2023-12-01T15:55:44-07:00 INF tapRunner.go:71 > Telemetry enabled=true notice="Telemetry can be disabled by setting the flag: --telemetry-enabled=false"
2023-12-01T15:55:44-07:00 INF tapRunner.go:73 > Targeting pods in: namespaces=[""]
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: argo-workflows-server-749b67b765-mms7p
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: argo-workflows-workflow-controller-5495f8d9d5-lfqfx
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: argo-cd-argocd-application-controller-0
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: argo-cd-argocd-redis-7d9484ccb6-fphpk
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: argo-cd-argocd-repo-server-654b7d56b5-grwsk
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: argo-cd-argocd-server-57f7665cc9-99bds
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: cert-manager-cainjector-8c7796555-vcq8m
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: cert-manager-cdc85d4c4-q6njm
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: cert-manager-webhook-55f45df998-fgjxc
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-front-7db76d5984-rvbpf
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-hub-549d4855f8-nwfdc
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-worker-daemon-set-4j2mk
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-worker-daemon-set-7ts2h
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-worker-daemon-set-9xqjf
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-worker-daemon-set-jwk8x
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-worker-daemon-set-kqj6z
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-worker-daemon-set-xfvsb
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: kubeshark-worker-daemon-set-zk7xz
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: web-visualizer-5c8cf8b4f5-hkbx7
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: web-visualizer-5c8cf8b4f5-wvxsm
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: arc-gha-rs-controller-766969f758-4mbm5
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: demo-lab-dev-runners-9d8dc86f-listener
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: coredns-6d9f985889-7w7mq
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-controller-879d6b797-dl2cn
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-node-5w84n
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-node-758wh
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-node-g6wsq
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-node-gtqlx
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-node-hkv5d
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-node-vm75c
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-nfs-node-wlscp
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-smb-controller-567b5f9f84-llfjx
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-smb-node-6klkc
2023-12-01T15:55:44-07:00 INF tapRunner.go:140 > Targeted pod: csi-smb-node-jk7sd
```

[35] https://github.com/kubeshark/kubeshark
[36] https://www.wireshark.org/
[37] https://github.com/kubeshark/kubeshark/releases/tag/v51.0.39
[38] https://helm.sh/

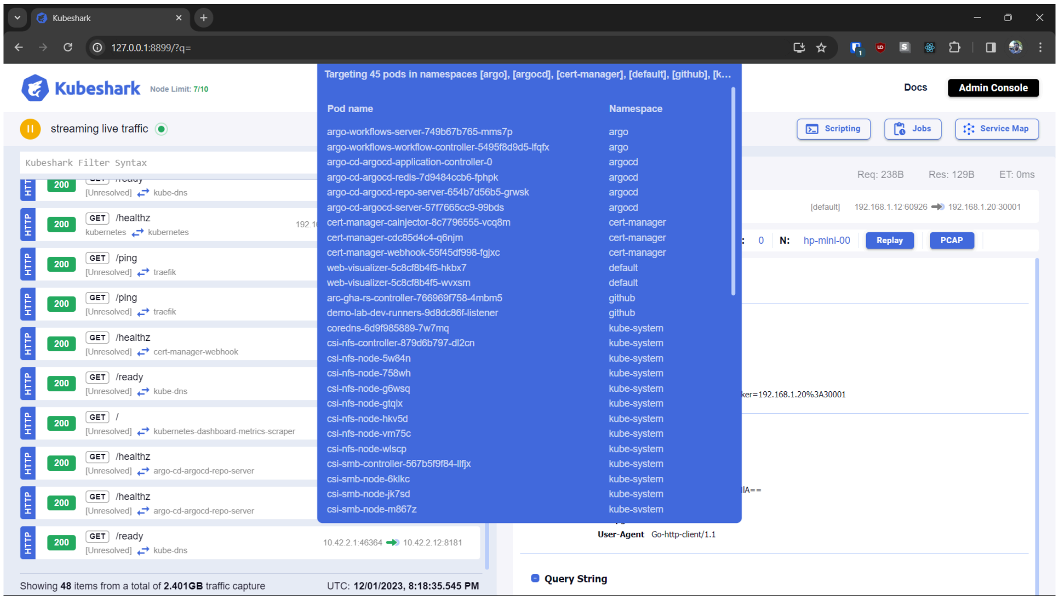



The *kubeshark tap* command defaults to listening on all Kubernetes namespaces[39]. This allows Kubeshark to have visibility to all network traffic within the cluster. Once the necessary resources have been installed, Kubeshark launches a web UI.



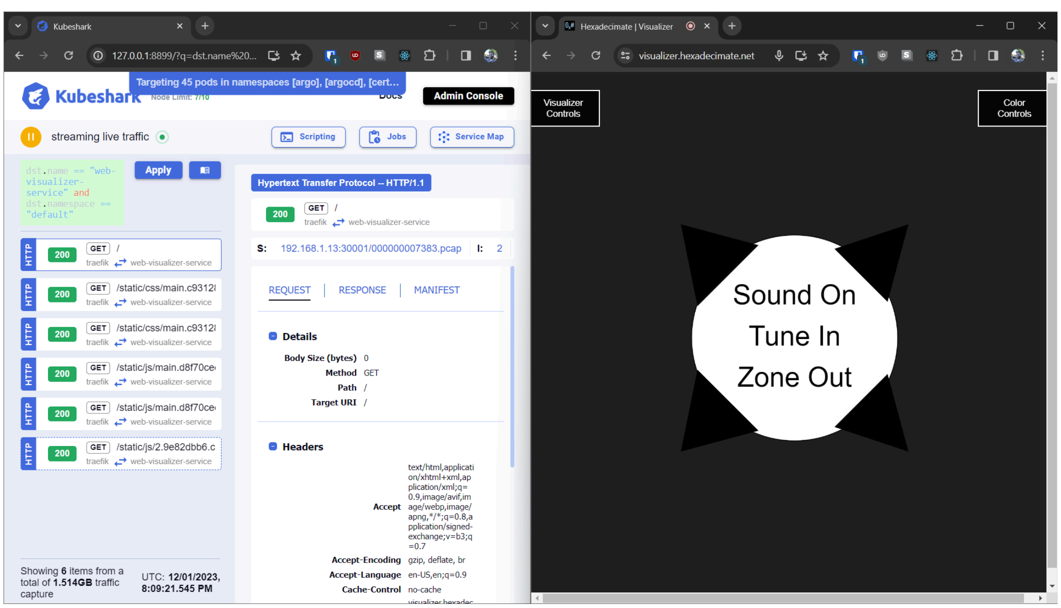

---

[39] https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

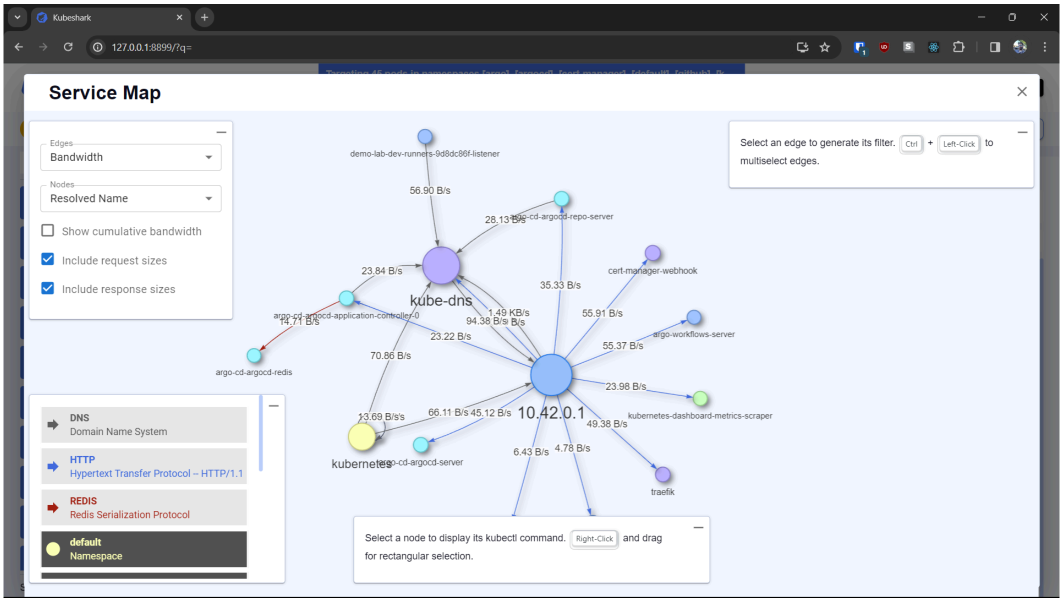



The web UI displays captured traffic on the left, and details about individual captures on the right. Selecting the drop down at the top of the page shows all of the Kubernetes namespaces and pods currently being captured.

Traffic can be filtered on many dimensions, allowing a more targeted approach. I used the *dst.name* filter combined with the *dst.namespace* filter to target a specific web service running with the cluster. Navigating to the associated webpage, the traffic showed the web resources being retrieved from the Kubernetes cluster.




In addition to providing detailed traffic analysis, Kubeshark also provides a high-level visualization of the traffic that is taking place within the Kubernetes cluster.



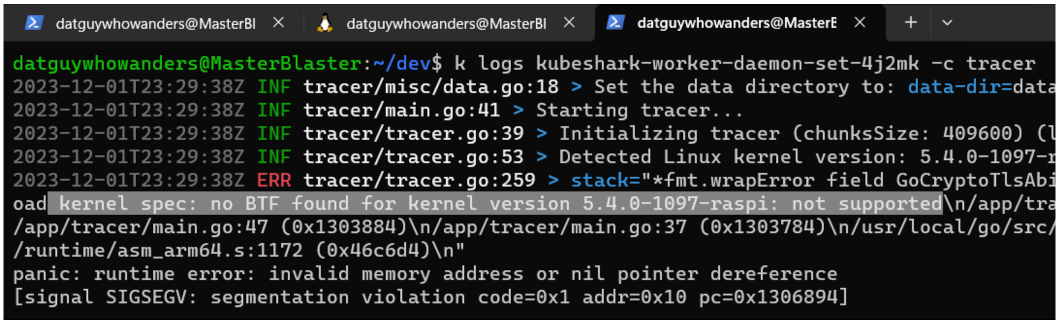

Kubeshark uses the size of both graph nodes and edges to indicate volume and size of traffic, and it separates different traffic protocols by color[40].

I faced some difficulties with Kubeshark running on the Raspberry Pi devices. Using *kubectl* to query for pods, it was apparent that not all of the resources were able to start successfully.

```
NAME                                READY   STATUS             RESTARTS          AGE
kubeshark-front-7db76d5984-rvbpf    1/1     Running            0                 174m
kubeshark-hub-549d4855f8-nwfdc      1/1     Running            0                 174m
kubeshark-worker-daemon-set-4j2mk   1/2     CrashLoopBackOff   38 (3m59s ago)    174m
kubeshark-worker-daemon-set-7ts2h   1/2     CrashLoopBackOff   37 (2m14s ago)    174m
kubeshark-worker-daemon-set-9xqjf   2/2     Running            0                 174m
kubeshark-worker-daemon-set-jwk8x   1/2     CrashLoopBackOff   38 (4m37s ago)    174m
kubeshark-worker-daemon-set-kqj6z   1/2     CrashLoopBackOff   38 (4m34s ago)    174m
kubeshark-worker-daemon-set-xfvsb   2/2     Running            0                 174m
kubeshark-worker-daemon-set-zk7xz   2/2     Running            0                 174m
```

A status of CrashLoopBackOff is an indicator that the underlying containers within a pod have failed to start so many times that the cluster is waiting for exponentially longer periods of time before retrying[41]. A further examination of the logs from the failing container indicated the problem.

40 https://docs.kubeshark.co/en/service_map
41 https://sysdig.com/blog/debug-kubernetes-crashloopbackoff/




The Raspberry Pi kernel isn't supported at this time.

## V. CONTAINER STATE & IMAGES

Capturing network traffic is one way to understand the behavior of what is going on inside the Kubernetes cluster, but it doesn't cover what is happening within container workloads. With the release of Kubernetes v1.25, forensic container checkpointing was introduced as an alpha feature[42]. This checkpointing functionality relies upon Checkpoint/Restore In Userspace (CRIU), a project that freezes the full state of a container and saves it to disk[43].

As this is an alpha feature, it's currently enabled through feature gates which can be used to unlock functionality on the *kube-apiserver*[44]. Further, as Kubernetes is designed to work with multiple container runtimes[45], the support for this feature varies across those runtimes. The CRI-O container runtime released support for checkpointing with its version 1.25.0[46]. However, the containerd container runtime still has its support for checkpointing pending in a pull request[47]. This made testing in k3s impossible at the current stage of this feature, as k3s defaults to the containerd runtime and only provides additional support for the docker container runtime[48].

[42] https://kubernetes.io/blog/2022/12/05/forensic-container-checkpointing-alpha/
[43] https://criu.org/Main_Page
[44] https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/
[45] https://kubernetes.io/docs/setup/production-environment/container-runtimes/
[46] https://github.com/cri-o/cri-o/releases/tag/v1.25.0
[47] https://github.com/containerd/containerd/pull/6965
[48] https://docs.k3s.io/advanced#using-docker-as-the-container-runtime

While container state capture continues to progress, there are other options for scanning the images that the containers run. Snyk is one company that provides security tooling that can be used to detect CVEs within a container[49]. While this won't help with the running state of a containerized workload, it can help detect security threats before they are released into a production environment.

I started a quick evaluation of the *snyk* CLI by downloading it based on their instructions[50].

```
curl https://static.snyk.io/cli/latest/snyk-linux -o snyk
chmod +x ./snyk
sudo mv ./snyk /usr/local/bin/
```

Before utilizing the CLI, it requires authentication with Snyk's servers[51].

```
export SNYK_TOKEN=TOKEN_FROM_SNYK
snyk auth $SNYK_TOKEN
```

Once authenticated, scanning a common public container image was straightforward. I started off by scanning the latest image for NodeJS, a popular programming language for web applications[52]. Running the command *snyk container test node:latest* produced the following summary.

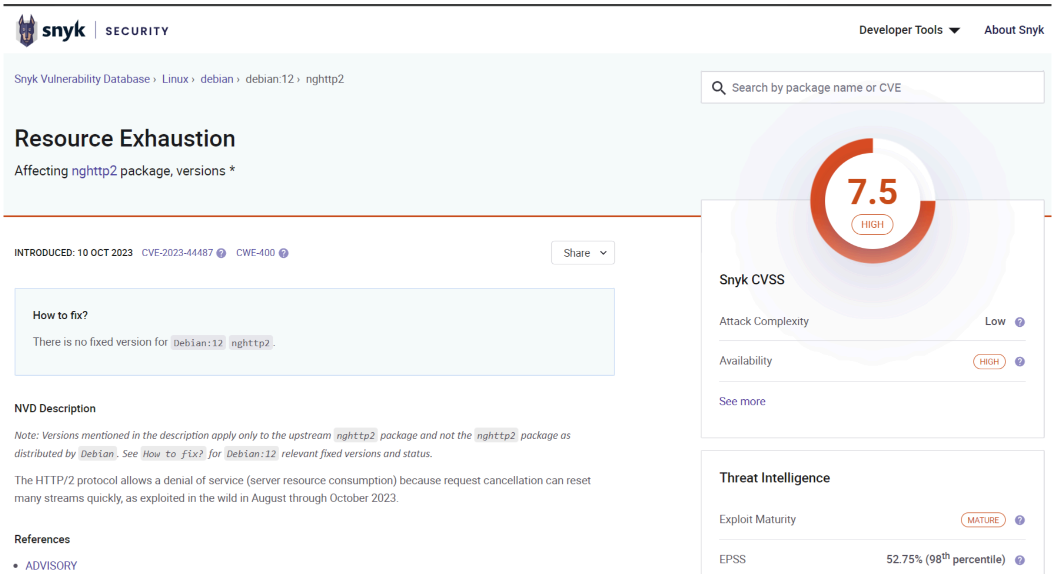
[49] https://snyk.io/learn/container-security/container-scanning/
[50] https://docs.snyk.io/snyk-cli/install-or-update-the-snyk-cli
[51] https://docs.snyk.io/snyk-cli/authenticate-the-cli-with-your-account
[52] https://nodejs.org/en

```
Organization:      johnford2002-cWSzkitkNPpnWvE7vXLTaN
Package manager:   deb
Project name:      docker-image|node
Docker image:      node:latest
Platform:          linux/amd64
Base image:        node:21.2.0-bookworm
Licenses:          enabled

Tested 413 dependencies for known issues, found 160 issues.

Base Image            Vulnerabilities  Severity
node:21.2.0-bookworm  160              1 critical, 1 high, 6 medium, 152 low

Recommendations for base image upgrade:

Alternative image types
Base Image             Vulnerabilities  Severity
node:21-bookworm-slim  32               1 critical, 0 high, 0 medium, 31 low


Learn more: https://docs.snyk.io/products/snyk-container/getting-around-the-
snyk-container-ui/base-image-detection
```

Even in this well-maintained public image, Snyk was able to identify several vulnerabilities, with a couple of them being rather serious. The results included more detail about each separate vulnerability.

```
✗ High severity vulnerability found in nghttp2/libnghttp2-14
  Description: Resource Exhaustion
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN12-NGHTTP2-5953379
  Introduced through: curl@7.88.1-10+deb12u4, git@1:2.39.2-1.1
  From: curl@7.88.1-10+deb12u4 > curl/libcurl4@7.88.1-10+deb12u4 > nghttp2/l
ibnghttp2-14@1.52.0-1
  From: git@1:2.39.2-1.1 > curl/libcurl3-gnutls@7.88.1-10+deb12u4 > nghttp2/
libnghttp2-14@1.52.0-1

✗ Critical severity vulnerability found in zlib/zlib1g
  Description: Integer Overflow or Wraparound
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN12-ZLIB-6008963
  Introduced through: zlib/zlib1g@1:1.2.13.dfsg-1, zlib/zlib1g-dev@1:1.2.13.
dfsg-1
  From: zlib/zlib1g@1:1.2.13.dfsg-1
  From: zlib/zlib1g-dev@1:1.2.13.dfsg-1
```

Clicking into the results loads a web synopsis of the vulnerability which quickly shows that some vulnerabilities are still present because there isn’t currently a fix upstream for the base operating system within the image.




Snyk also provides monitoring capabilities to track vulnerabilities over time[53]. This allows individuals and teams to get alerts when fixes are released so that they can remediate previously identified CVEs.

[53] https://docs.snyk.io/snyk-cli/commands/monitor

## VI. CONCLUSIONS

The Kubernetes ecosystem continues to evolve, and the available tooling for security and forensics does as well. With Kubernetes' built-in auditing support, it is possible to monitor and alert on activity related to events happening within the cluster. However, the amount of events generated can be overwhelming, and a proper audit policy needs to be appropriately scoped to gather useful insights. Traffic happening within the cluster can be captured to analyze communication between services, and Kubeshark is one tool that provides powerful filtering capabilities to zero in on interactions with specific protocols, services, or applications. Finally, full state capture of running containers is coming in the near future. This new capability will provide deeper insights into what is happening inside of workloads, not just what they're communicating externally. In the meantime, security scanning software is available to inspect the images supporting the containers. Snyk is a company that provides a wide array of services to help teams monitor and fix vulnerabilities, preferably before they ever reach a production environment.

**Dated: 01 December 2023**

**John C. Ford III**