HW2, Raspberry Pi Network Application Exploration – Media Server (February 2020)

Yaamini Barathi Mohan, Hyoeun Choi, Annamarie Casimes

*Abstract*— This project demonstrates application level development on a Raspberry Pi using Docker containers, cross-compilation, and knowledge management using a GitHub Wiki. Student teams were each assigned a separate application layer server to host. Upon the conclusion of the assignment the all student devices could form a broader network hosting email, http, media, ftp, dhcp, game and a peer-to-peer servers. This article focuses on student team three - a media server (kodi UPnP/Server).

In this project the intent was to setup a runtime environment and a build environment. The runtime environment consisted of a client hosted on a Kali Linux virtual machine running a Kali Linux docker container. The client connects to a Kodi media server instance hosted by an Ubuntu Server docker container running on a Raspberry Pi 4 with an Ubuntu server operating system. The build environment allowed developers to build the applications on an x86-64 bit architecture but cross compile it for an ARM64 bit architecture. Wiki markdown format documents were used for documentation and knowledge management.

This project encountered numerous setbacks and ultimately was unable to implement the build environment for cross or native compilation of source code. The team attempted to use three different client/server software packages: Kodi, Plex, and Gerbera. Kodi failed due to software package and X11 dependencies. Plex server source code is proprietary. Gerbera failed due to a C++ compiler filesystem error.

*Index Terms*—Runtime environment, build environment, media server, Kodi, GitHub.

# introduction

Application development is complicated due to application’s dependencies on underlying system architectures. This project showcased several concepts including: cross-compilation, build vs run-time environments, documentation, and application and network security.

Cross-compilation – compiling software on one architecture for another architecture where (usually) there is a less powerful processor. The intent of the project was to cross-compile the server/client source code on a modern day personal computer with an x-86 architecture for a less-powerful but still impressive Raspberry Pi 4 featuring an ARM architecture. Cross-compilation was ultimately relaxed to native complication due to complexities in the project.

Build vs Runtime environments. Build environments allow engineers to gather all the required dependencies, source code, and compilers which together can take up significant space. Engineers can then make and configure the source code into a binary file to run on a like or unlike architecture. The resulting binary file is then distributed to the systems that need to use it along with any project dependencies.

Runtime environment is where the application will actually function. Docker allows the containerization of applications which enhances portability. Each docker container contains a stripped down supporting architecture, the software or application or data to support its purpose, and any dependencies. Docker has a number of advantages including standardization, compatibility, isolation, encapsulation, simplicity, rapid deployment, portability, and more [1][2]. Docker was new to the majority of our team members. There is a learning curve associated with containers in general and Docker specifically. More information about Docker is on their website [3].

Network Security is a multifaceted domain which includes security throughout an application development lifecycle. The best method to achieve a secure application is to consider security throughout the project rather than bolting it on at the end. Often projects are complicated and engineers causing engineers to struggle to meet deadlines. This added stress often leads engineers to consider functionality over security. Our project is an example of this scenario, however, we attempted to consider security impacts throughout the development process.

# Project Goals

The main project goal is to understand several components of network security focusing on threat modeling, network security policy, and secure network design. The goals of the assignment are:

1. Knowledge Management
2. Runtime Environment build out
3. Build Environment build out
4. Network analysis between server and client

# Project Execution

## Knowledge Management

Knowledge Management is critically important in development. Documentation aids in software/system security approval and validation. Documentation also allows future engineers to understand the current status and build new components or enhancements for the product. Documentation also allows other engineers to replicate previous tasks with less effort so users are not continually “reinventing the wheel.”

For this project, we used GitHub as our main repository for information including code (Dockerfiles and scripts) and for the “how-to” wiki. GitHub uses markdown files which allow special formatting. The figure below is an example of some of the formatting options available. This markdown file is also available on our wiki page at: <https://github.com/guszedd/Network-Security/wiki/_new>

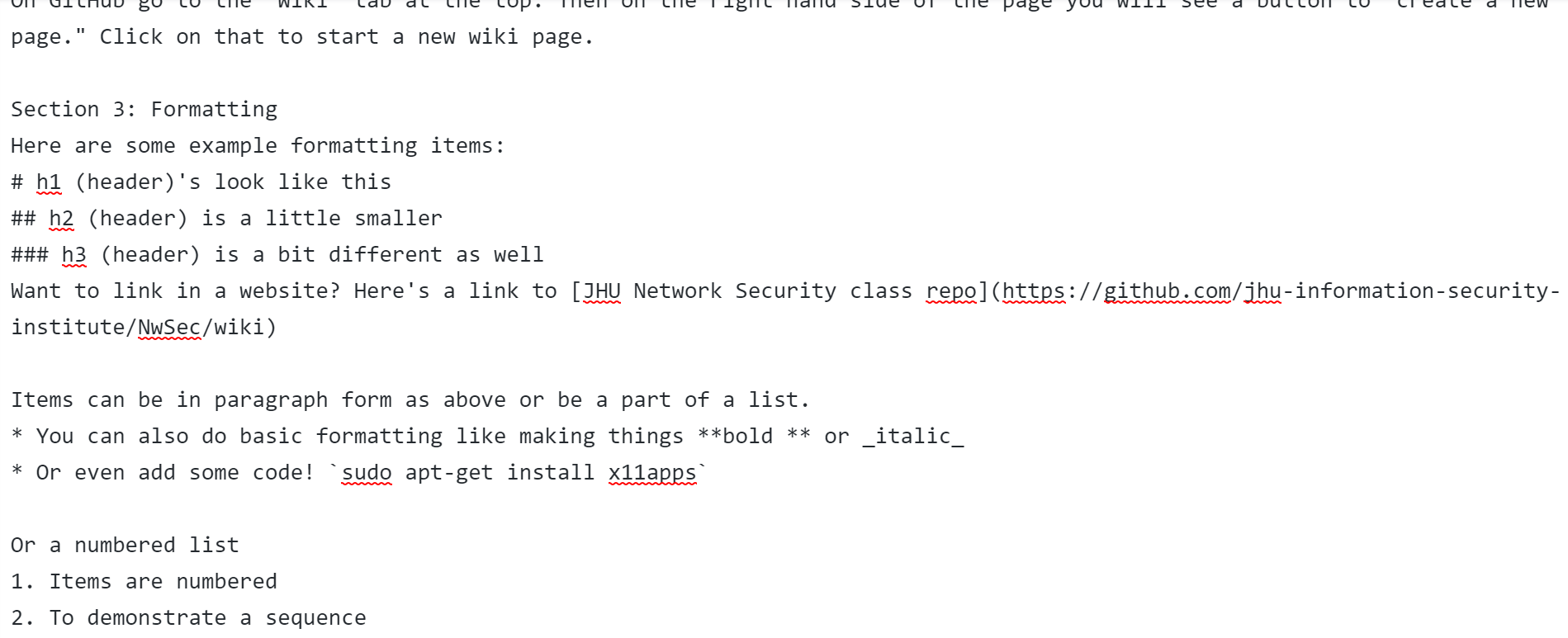


Figure 1. Markdown file (Editor’s view)

GitHub renders the properly formatted markdown file in a user readable way. Here is the above markdown file rendered on GitHub as a wiki.

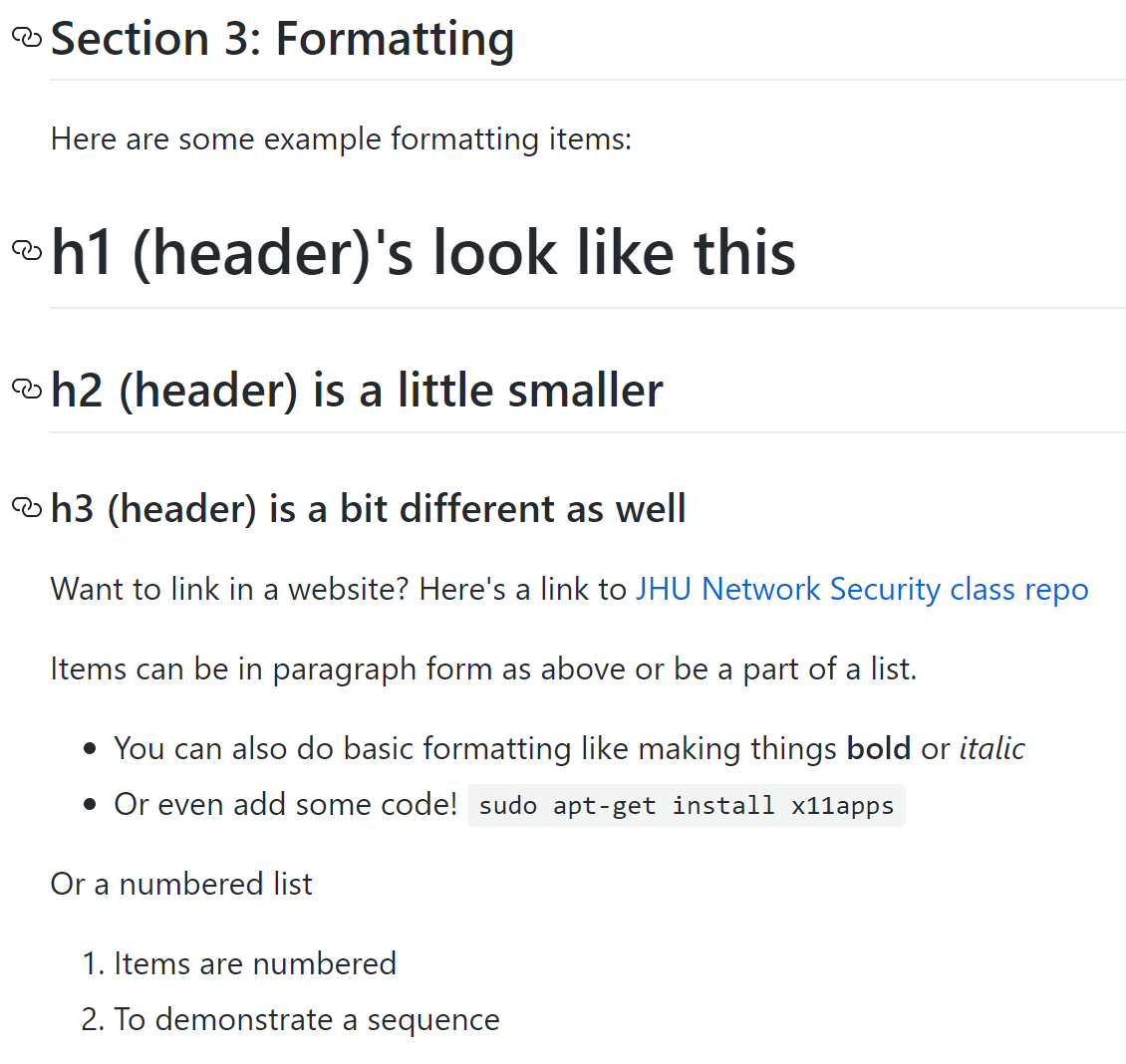


Figure 2. Markdown file (Reader's view)

For this project we created many versions of documentation. Each path we went down had its own sequence of events. In order to avoid confusion only the “final” versions of the documentation were uploaded to GitHub. However, it is important to note that, in some cases, it is useful to upload a working copy (with appropriate markings), to provide intermediate or in-progress information to other developers or customers. Our wiki page used another method – announcing future additions but not providing in-progress copies to help control information dissemination and versioning.

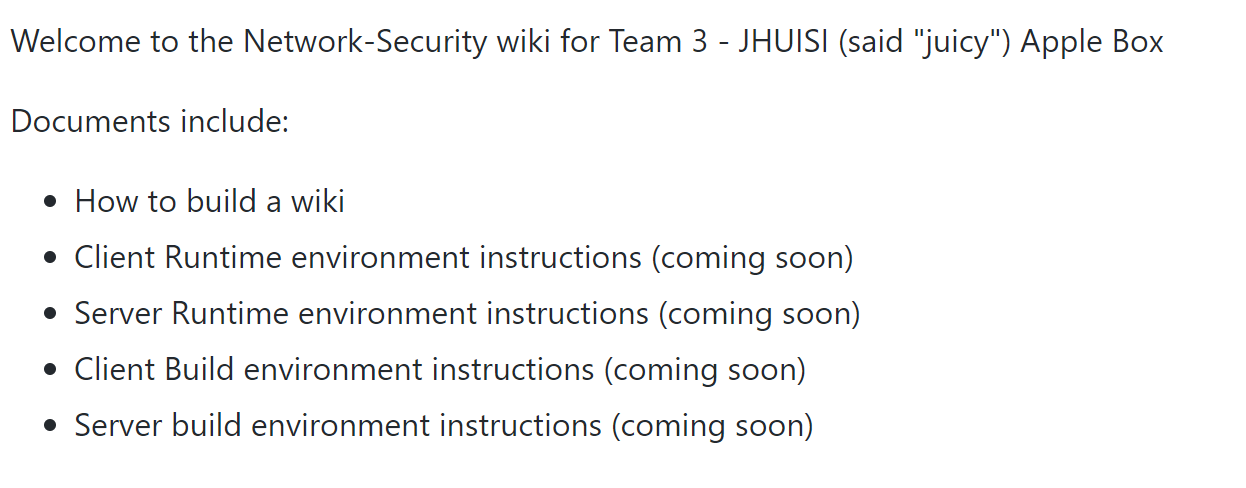


Figure 3. Wiki Markdowns Provided

In addition to capturing “this is what I did, and this is how I did it” instructions, it is also useful to document lessons learned and other team-based lessons. This can be done in a variety of methods. For our team this includes this executive summary. We faced many significant hurdles through this project across a variety of topics - technical, academic, and personal.

Technically this project introduced many new items including cross-compilation, build vs runtime environments, Docker, containerization of applications, multiple Linux based operating systems, and a variety of media server/client options. Each of these items required extensive individual research prior to attempted implementation. Part of learning is failing and we each did a great deal of both – finding new ways to break our containers and VMs we were able to practice extensive (and at times painful) troubleshooting. Each team member learned a lot about each concept which will undoubtedly help us in our future careers.

Academically, this project was introduced in the lead up to mid-terms and was (kindly) extended into the week after spring break. The ability to multi-task is a critical skill of students and professionals alike. This semester provided us ample opportunities to enhance our skills and test our academic understanding of concepts via real-world applications.

Personally, we had to adjust and refine our team dynamics. We went from working as a team in person to remotely coordinating due to COVID-19 which added some obstacles. Additionally, midterms, other classes, an unstable and dynamic COVID-19 environment, and general personal issues complicated the process. Personal stress and other difficulties are important for team members to be aware of and to learn how to work with and thru. DevSecOps is certainly a team sport and therefore has its own challenges.

## Runtime Environment

The runtime environment was modified slightly due to complexities in X11 rendering between containers on the RPI and the controlling SSH host. The following diagram depicts the architecture used for the runtime environment.

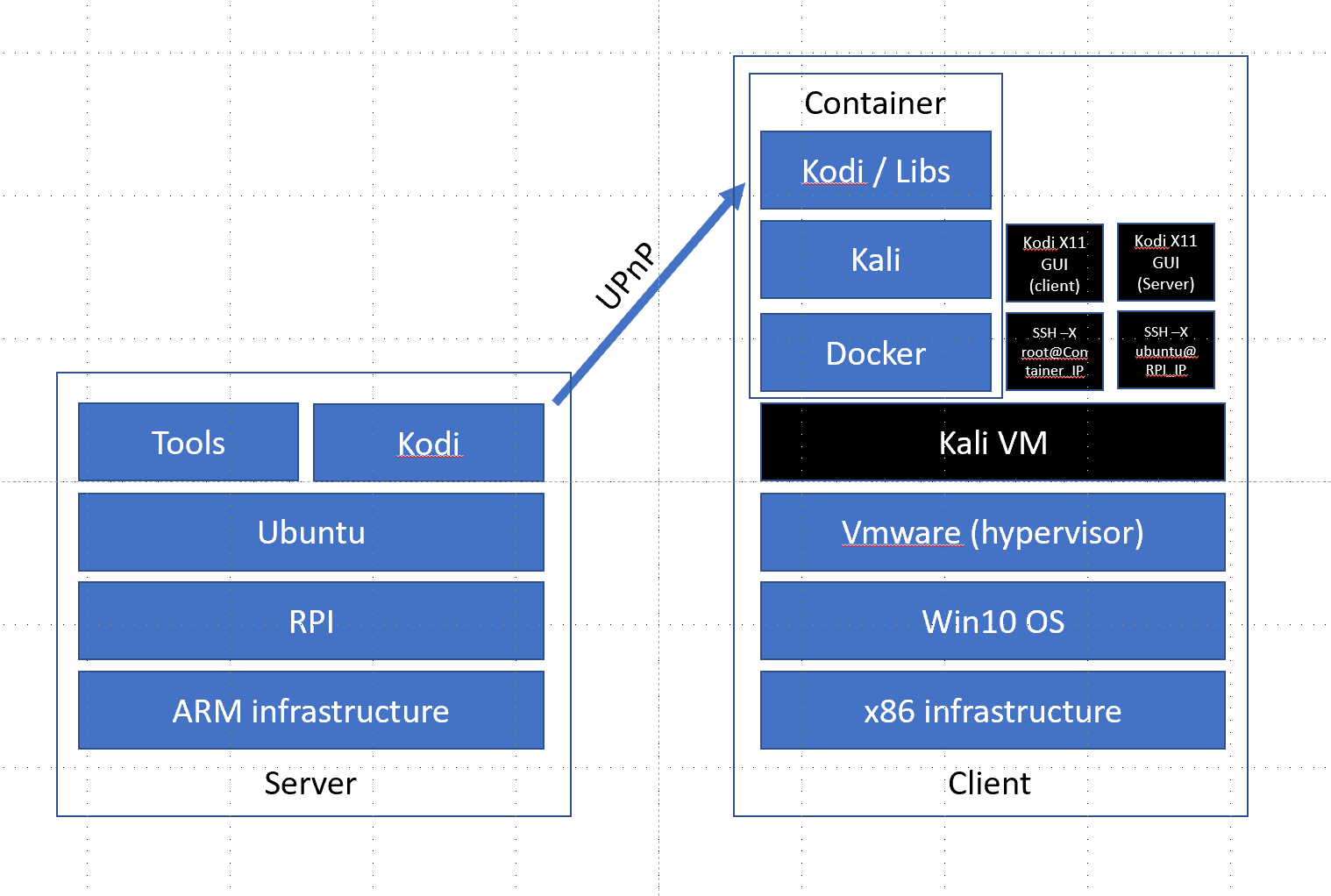


Figure 4. Runtime Architecture Diagram

Server. The server architecture is implemented on a Raspberry Pi (RPI). The RPI ran Ubuntu server v18.04 operating system. The RPI uses an ARM architecture. Kodi and required dependencies and useful tools were installed directly onto the Ubuntu server operating system. A detailed explanation of the steps is located on the Wiki.

(INSERT PICTURE OF WIKI HERE)

Key components of the server side include Kodi media control software, SSH server software, and X11 dependencies. SSH server and X11 dependencies were configured via script for ease of use and replication. X11 is a “communications protocol” that “defines protocol and graphics primitives.”[4] X11 is used to send the Kodi software’s graphical user interface (GUI) thru the SSH connection and use the host’s resources to display the GUI. This allows the container to run the Kodi software but use the host’s resources to display the Kodi GUI to allow the user to configure and use it. SSH is required to transport the information from the server to the host to facilitate the X11 link.

Kodi is an open source home theater software. It is designed “to run on a computer/supported-device that is directly connected to the TV by a video cable.” [5] However, “Kodi has some support for [streaming Kodi from my PC to my TV using DLNA/UPnP].” [5] This process was entirely more complicated that it should be. The most comprehensive guide is provided by Kodi. [6] Key to this process is “scrapping” provided by the default “information provider” (The Movie Database). Videos in your repository must be “scrapped” by the information provider to be added to your library in ways that allow sharing via UPnP. If a video is not scrapped, the folder structure will be visible on the client, but the actual media is missing. There is no easy way to see if a file was properly scrapped. Perhaps this would be a useful addition to Kodi?

In the runtime environment, the client Kodi connected to the UPnP server (Kodi on RPI Ubuntu). The client was able to browse movies hosted by the server and begin streaming. The following images show browsing for videos and then video playback. The top Kodi window is the Client Kodi GUI and the bottom is the Server Kodi GUI.

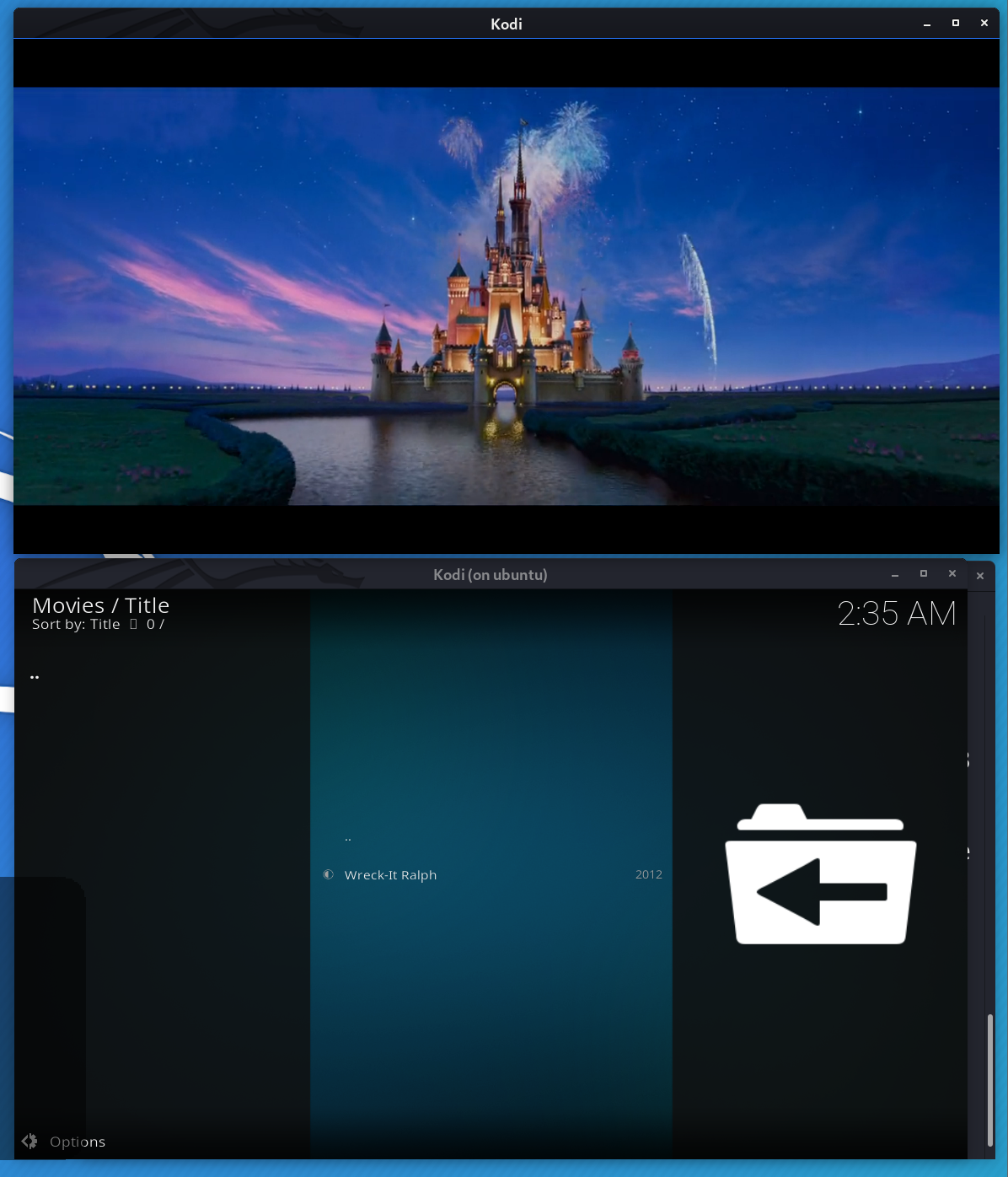
Machine generated alternative text:
Kodi 
Movies / Kodi (ubuntu) 
Sort by: Title D 1/1 
Wreck-It Ralph 
The story ofa regular guy just 
looking for a Little wreck-ognjtjon. 
Wreck-It Ralph is the 9-foot-tall, 
643-pound villain of an arcade video 
game named Fix-It Felix Jr., in which 
the game's titular hero fixes 
buildings that Ralph destroys. 
Wanting to prove he can be a good 
guy and not just a villain, Ralph 
escapes his game and lands in 
Hero's Duty, a first-person shooter 
where he helps the game's hero 
battle against alien invaders. He 
later enters Sugar Rush, a kart 
racing game set on tracks made of 
candies, cookies and other sweets. 
There, Ralph meets Vanellope von 
Options 
Movies / Title 
Sort by: Title 0/ 
Options 
Wreck-It Ralph 
Kodi (on ubuntu) 
C Wreck-It Ralph 
201 2 
2012 
1 35 AM 
101 min 
2 34 A M 

Figure 5. Video Selection Client Streaming Video

## Build Environment

In this task we researched and developed a network architecture to support our novel, online multi-player gaming company. The company must support local users, users, a remote office, and external customers. We implemented a multi-tiered architecture to provide a defense-in-depth approach. We created several network segments including a demilitarized zone (DMZ), untrusted zone, and the trusted internal network.

The DMZ provides a way to allow external untrusted users access to company resources without providing them access to the internal private network. Segmenting the network based on trust provides additional layers of security. We placed the web server, a file transfer server, and DNS server in the DMZ. Additionally, we built a honeypot and placed it in the DMZ. The honeypot allows us to lure potential attackers away from critical resources and provides a sandbox to examine their tactics, techniques, and procedures (TTPs). Using the information gathered from honeypot analysis we can build and refine network security mechanisms and parameters to prevent, halt, and deter similar attacks to our valued resources.

The untrusted zone includes the interface to the external network, namely, the internet. To monitor traffic, we placed a stateful inspection firewall as well as application-level gateway (proxy firewall). The proxy firewall allows us to optimize and secure internet requests originating from within the network. The stateful firewall conducts packet inspection and maintains the state of connections to help deter attacks.

We further separated the internal network and into user access locations and production environments hosting critical infrastructure servers. By segmenting the users away from production servers, we help contain network traffic based on business requirements. We segmented the network into the network core, distribution, and access layers. The core’s main responsibility is routing between the network segments, but it also plays a vital security role by passing data to the intrusion detection system and firewall. The access layer is characterized by user access switches providing increased port density to support user connections across the office spaces.

## Network Analysis of Runtime Environment

In this task we analyzed the network traffic from the runtime environment. Since the session between the client and server was initiated via a secure socket shell (SSH) connection all of the data transmitted is encrypted. Below is an excerpt showing network traffic captured during the video playback. Communication is from the server (IP: 192.168.1.186 – RPI) to the client. The docker container’s IP address is not the packet’s destination of the IP packet because it is routed to the host VM which then routes it to the appropriate docker container (on a private network). Note the destination port is port 22 which is used for SSH. SSH communicates via the TCP protocol. The figure demonstrates server and client exchanging packets.

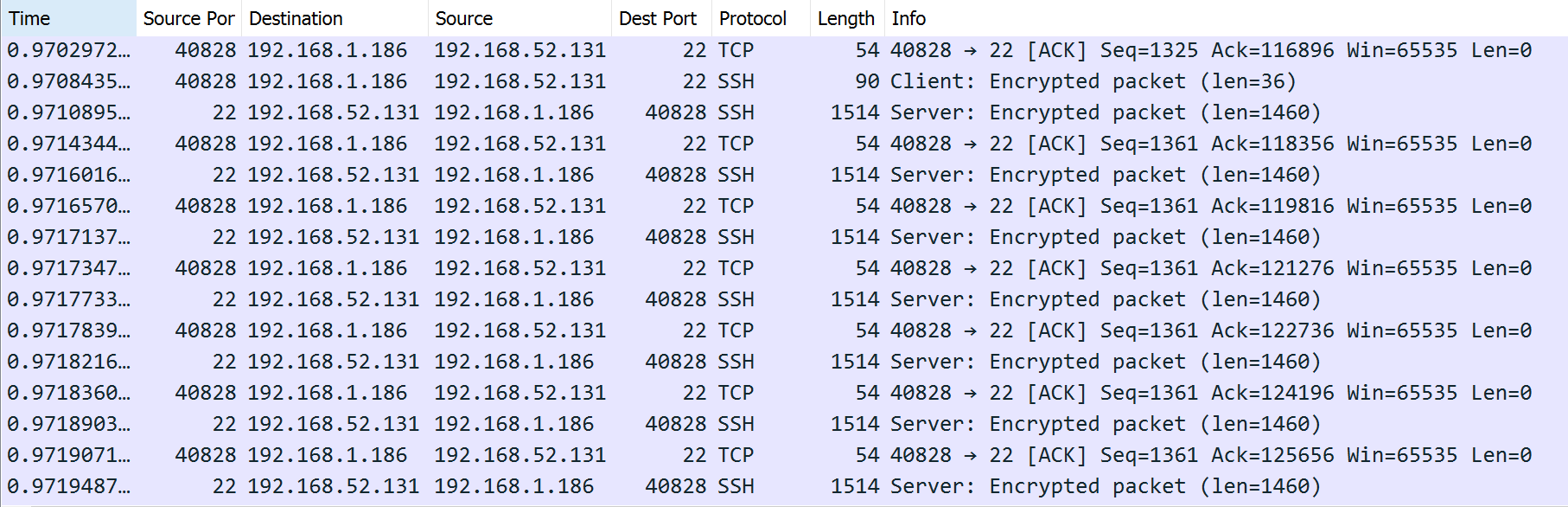


Figure 6. Runtime Network Traffic Analysis

The following figure shows the IP configuration of the Kali VM which is running the Kali (client) Docker container. Note the private network IP address of the Docker interface (172.17.0.1). This 172.17.0.0/16 network only communicates between the host VM and the containers. This is due to how the container was configured – using the –network host command when the container was “run.” This shares the host’s layer three and four configuration between the host and container. The Docker software manages the network routing between the host and the containers. The ethernet connection (etho0 interface) is connected to the 192.168.1.0/24 network which communicates across the private home network (connects the VM and the RPI).

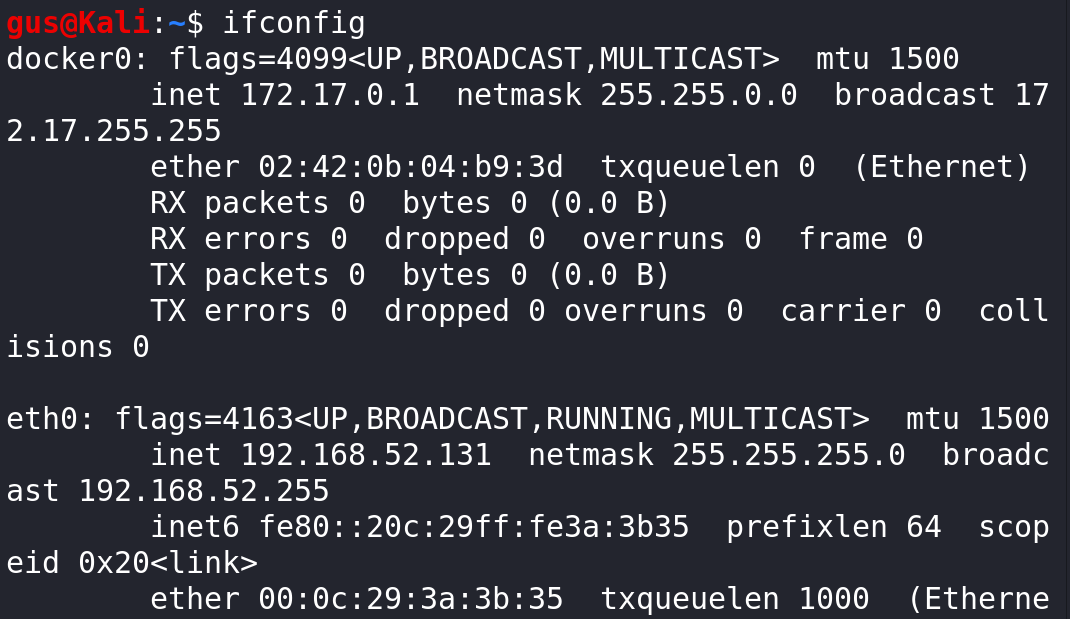


Figure 7. IP Configuration Kali VM

The following image is a depiction of the network setup for the runtime environment used in this project.

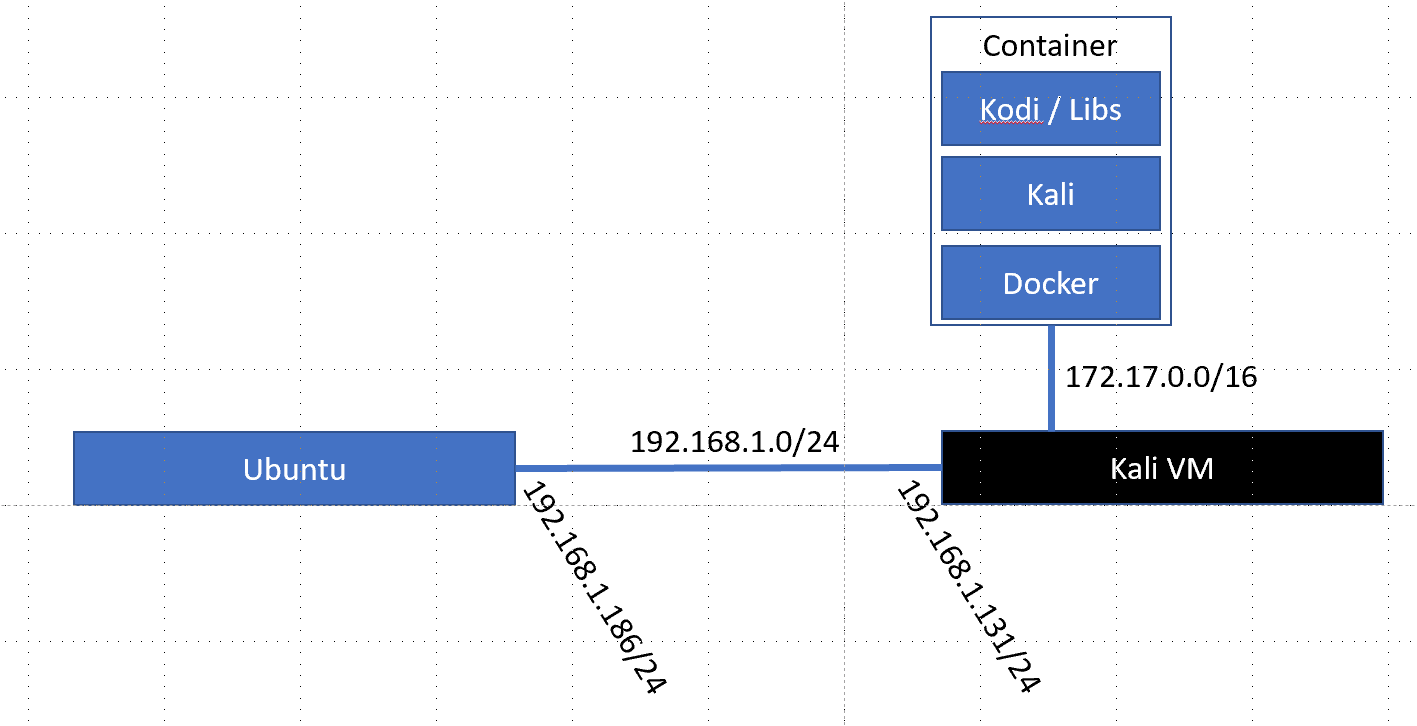


Figure 8. Network Diagram

# Conclusions

This project introduced a host of new concepts and challenges to the team. Several of the original requirements were relaxed due to complexities and difficulties. The team members learned a significant amount about Docker, Kodi, and troubleshooting the associated components. The team also learned a fair amount about team dynamics and operating under stressful conditions.

References

[1] https://apiumhub.com/tech-blog-barcelona/top-benefits-using-docker/

[2] https://www.infoworld.com/article/3310941/why-you-should-use-docker-and-containers.html

[3] https://docs.docker.com/

[4] https://unix.stackexchange.com/questions/276168/what-is-x11-exactly

[5] https://kodi.wiki/view/All\_platforms\_FAQ

[6] https://kodi.wiki/view/Adding\_video\_sources

**Yaamini Barathi Mohan,** I am a first-year Master of Science in Security Informatics Student at Johns Hopkins University.

**Hyoeun Choi,** I am first-year Master of Science in Security Informatics Student at Johns Hopkins University.

**Annamarie Casimes,** I am first-year Master of Science in Security Informatics Student at Johns Hopkins University.