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>

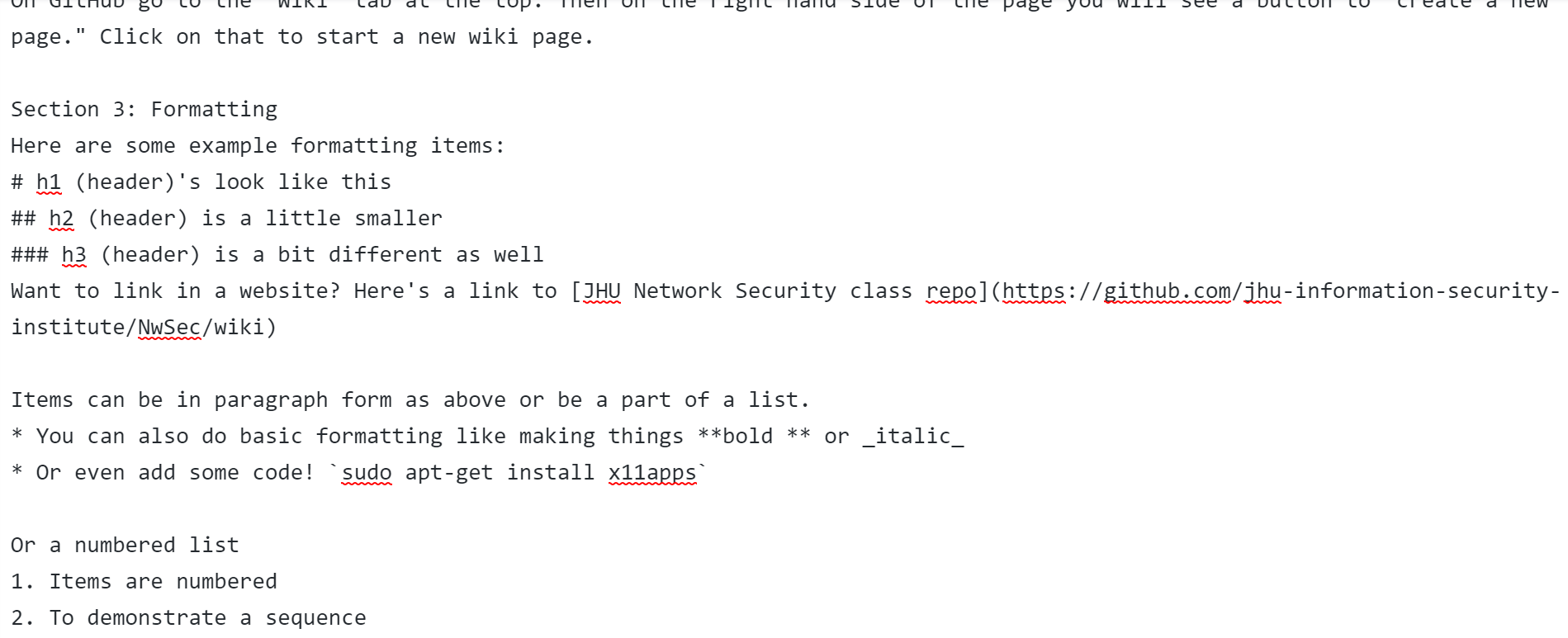


Figure 1. Markdown file (Editor’s view)

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

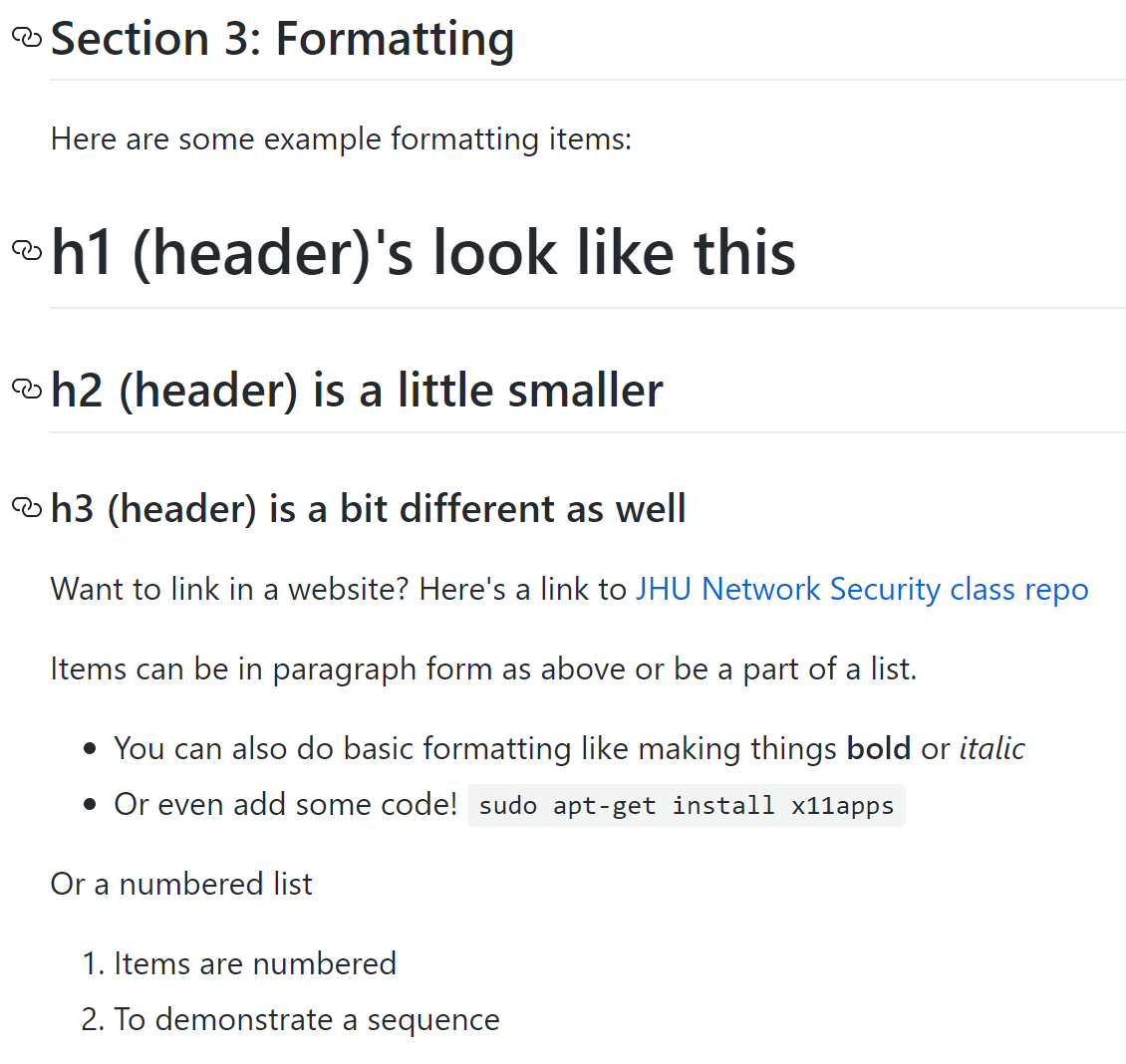


Figure 2. Markdown file (Reader's view)

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.

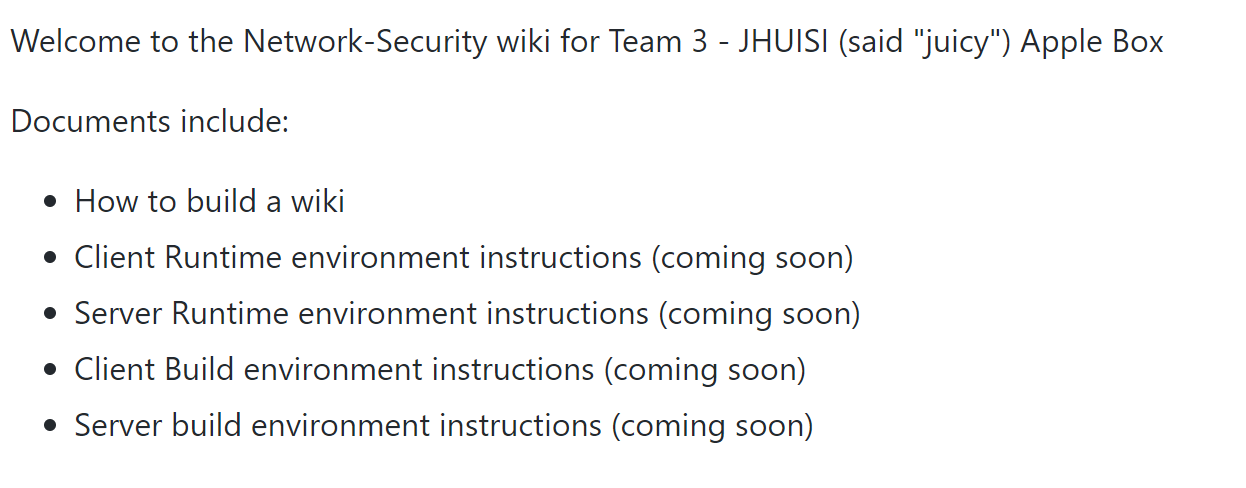


Figure 3. Wiki Markdowns Provided

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

We organized our Company Employee requirements into two main sections – common to all and privileged users. We organized the document in this fashion to ensure privileged users adhere to the same basic principles as normal users and to highlight specific advanced requirements. Examples of common security requirements include basic employee conduct, remote access requirements, user responsibilities, and prohibited user actions. Additional administrative statements highlight the increased importance of account protection, authorized and unauthorized privilege user actions, and adherence to other company policies.

Our customer section focused on requirements for the online multi-player game customers. Since the game is hosted online the organization needs to establish rules for accounts as well as proper behavior. Misuse of company resources is a distinct possibility. Through security mechanisms and legal verbiage, we can mitigate the company’s liability. Examples of customer requirement statements include: the requirement to gain parental consent for minor users, improper customer behavior and actions, and basic customer responsibilities.

## 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 described the overall network architecture as well as the functional components. We included over 30 nodes spread across 25 different node types. Nodes are vender and operating system agnostic however in actual implementation a variety of vendors, specifically for network security devices, is recommended to ensure a single vendor vulnerability does not dismantle the entire architecture. Node categories include external customer computers, employee computers, network devices (wireless access points, routers, switches, etc.), network security devices (intrusion detection systems, firewalls), servers (email, web, DNS, game, VPN), security information and event manager (SIEM), as well as logging, archive and backup servers. Individual machines also include personal firewalls and host intrusion prevention software. Our architecture also supports centrally managed software updates with multiple distribution points to add with package delivery and load balancing.

We also considered threats to the eight information security tenets as well as disaster recovery plans and procedures. As a novel, online gaming company we do not have the financial capital to invest in a hot site but do maintain a contract with vendors for hardware replacement and troubleshooting. Additionally, we used load balancers to distribute network traffic between multiple instances of production servers to remove single points of failure. The network also provides redundancy through multiple network paths (dynamically managed by routing protocols), fiber channel techniques, as well as virtual gateways. This redundancy helps build resiliency in the network.

Protecting against insiders is more difficult. The company was built using the standard three-layer trust model: untrusted external networks, DMZ, and trusted internal network. A more secure approach, but much more intensive, is to use a zero-trust model inside the network as well. This requires machines and servers to conduct mutual authentication as part of each data transfer. In lieu of zero-trust, we have put in place a user monitoring program which logs all actions on key resources and provides an audit trail. Additionally, our company uses mandatory vacation methods for its privileged employees. Finally, our company participates in user awareness training which helps make users more aware of external threats as well as internal ones. Combining

# Conclusions

In this project we followed the steps of threat modeling and threat mitigation as part of a more comprehensive risk management process. We then considered a fictional online game network and designed its supporting network architecture and network security policy requirement statements. A network diagram detailing the architecture and data flow is critical to a risk assessment and overall network security strategy. As part of our network architecture we provided an overall description as well as functional descriptions for each node. We considered multiple threats and designed our network with built in resiliency and redundancy with a variety of baked in security mechanisms.

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/

**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.