COLORADO SCHOOL OF MINES®
@150 | 1874-2024

CSCI 370 Final Report

Team Miniteam

James Crea

Ran Duan

Orion Tanner Magill

Revised December 6th, 2024

CSCI 370 - Fall 2024

Advisor: Kristen Peglow

Client: Nicholas Blair

Table 1: Revision history

Revision	Date	Comments
Rev 1 - Requirements Document	9/1/2024	Sections I through V, and XIII onwards have been filled in, subject to later revision.
Rev – 2	9/15/2024	Section VI has been filled in. Minor revisions were made to the previous sections.
Rev – 3	10/20/2024	Section VII and VIII have been filled in. Section VI was overhauled in response to us switching to a new tool: Raizo62/vwifi.
Rev – 4	11/10/2024	Section VII was updated with test results, and sections IX - XI were added.
Rev - 5	11/21/2024	Minor revisions were made, and an acknowledgement added.
Rev - 6	12/6/2024	Revisions, based on feedback from peer reviews, were made.

Table of Contents

I. Introduction

The grid has become more complicated and connected than ever. This can be seen in the spread of Internet of Things (IoT) devices, prosumer energy generation (solar panels), and the advent of electric cars featuring high-capacity batteries. Consequently, the impact of cyberattacks, as well as cyberattacks themselves, have become more prevalent across the energy space. This is why the National Renewable Energy Lab (NREL), a laboratory operated on behalf of the Department of Energy, has increasingly invested in cybersecurity. In order to prepare for the worst, their cybersecurity team uses several tools for simulating cyberattacks, namely Minimega and Phenix. Minimega is a command line tool for quickly creating a single Virtual Machine (VM), and Phenix is a tool that ties into Minimega to quickly create many VMs. However, there is a problem with these tools; currently, they can only simulate wired ethernet connections to the VMs. Because of this, these tools cannot accurately model many real-life networks, scenarios, and attack vectors. To this end, our client, Nicholas Blair, has set the goal of simulating wireless networks to facilitate higher-fidelity simulations via extending Minimega and Phenix, including Phenix's YAML-based configuration system.

II. Functional Requirements

The functional requirements are:

- Minimega
    - Add the WiFi simulator as an alternative implementation of the aforementioned interface(s)
    - Add the WiFi simulator bootstrapping logic (i.e., the simulator runs as a separate process from Minimega/Phenix and therefore needs to be started up separately)
    - Add/modify the appropriate networking-related commands, *wiring* them up to the aforementioned interface(s)
- Wi-Fi Emulation
    - Simulate a wireless network using MiniNet WiFi or a similar tool (i.e.: Raizo62/vwifi).
    - The user should be able to define virtual machines that communicate over Wi-Fi in the simulation environment
    - The system should allow for the simulation of network conditions, such as dropping packets to simulate a dirty network or degraded service
    - Create integration tests to ensure that the new WiFi simulation logic works correctly with the existing Phenix and MiniMega systems
- Stretch goals
    - Phenix
        - Update the YAML schema to support the wireless network features
        - Update the YAML parsing logic to account for the changes in the schema
        - Update the Minimega command generator to generate the new/modified networking-related commands

III. Non-Functional Requirements

The non-functional requirements are:

- The program modifications must be fully backward-compatible with the upstream source (to ensure our forks are drop-in replacements)
    - The program modifications must not reduce or break any existing functionality
    - The program modifications should not alter the build system/process more than absolutely necessary
    - The program modifications must remain deployable using Docker and Docker compose
- The program modifications must not reduce the quality of the overall codebase
    - The program modifications must follow the established testing framework (i.e.: Go standard test library)
    - The program modifications must follow the established programming style, including standard Go conventions (See section VIII for more details) and the ESLint configuration (for any frontend changes in Phenix, if any)

- The program modifications must be uploaded to the private Git forks
- The private Git forks must be made public when complete, for delivery to the client

IV. Risks

The risks for this project were:

- Low familiarity with the tech stack (802.11, Go, Linux admin, Mininet, Raizo62/vwifi, Open vSwitch, YAML, web development). This threatened to and subsequently did slow down development.
- Mininet-WiFi may not be the right tool for the job. This ended up being the case, necessitating us having to search for alternative tools.
- Open vSwitch is more deeply integrated into Minimega than expected. Open vSwitch supports ethernet (and thereby all connectivity) in Minimega and Phenix. If Open vSwitch cannot be bypassed, we may be forced to refactor very large portions of the codebase. Thankfully, this ended up not being too much of an issue, though it did require several hundred additional lines of code to bypass ethernet-specific functionality.

V. Definition of Done

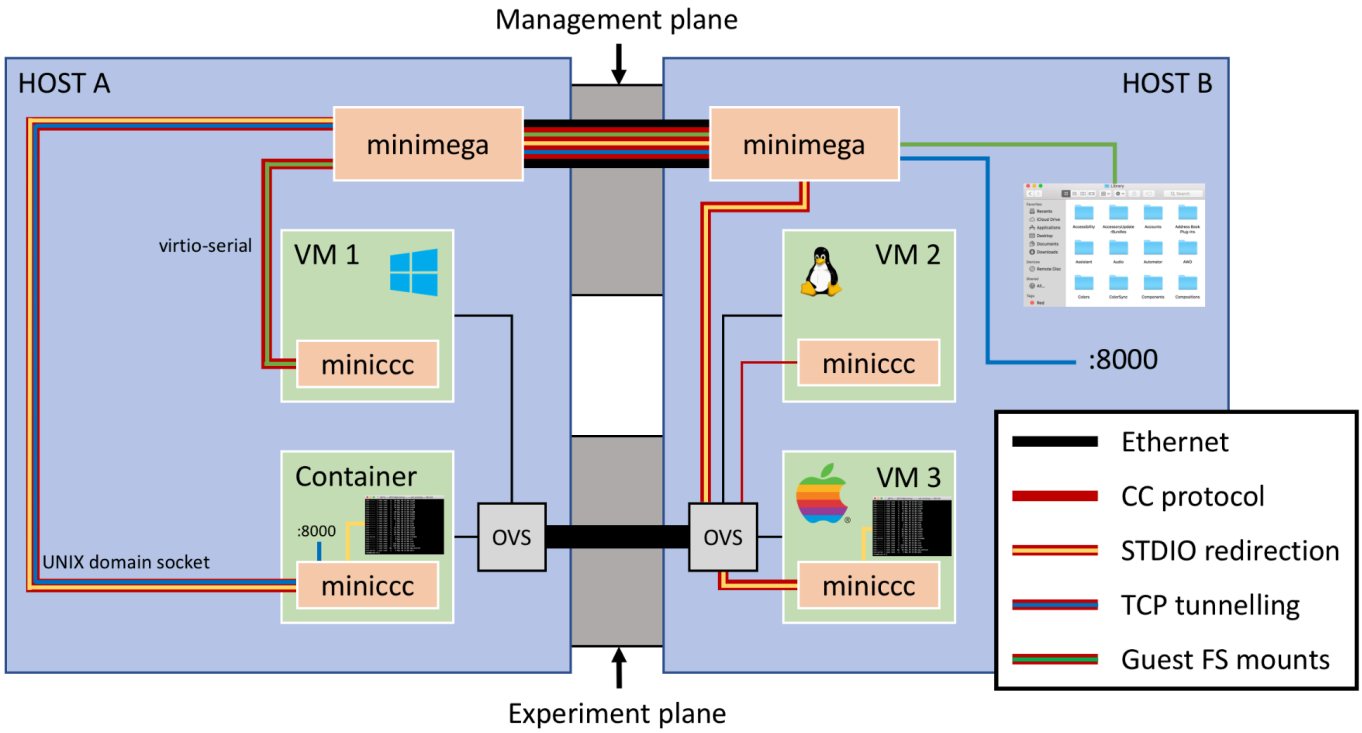
Our definition of done as stipulated by our client is:

- Feature set
    - You can spin up a virtual WPA2 Personal WiFi network with a hardcoded SSID (network name) and password that Linux VMs can discover/connect to the WLAN with a static IPv4 config and can ping each other (ICMP echo request/replies)
- Tests prior to delivery
    - All previously passing automated tests continue to pass
        - Many tests in the upstream repository, without any of our changes, are currently broken at the time of writing
        - Minimega includes a fuzzer named Minifuzzer; while Minifuzzer is useful for finding shallow bugs that can cause crashes in Minimega, Minimega is already crashing without any of our changes in the upstream repositories
    - The above simple scenario (i.e.: WPA2 Personal with hardcoded SSID/password) works
- Delivery
    - Public fork of the minimega and sceptre-phenix repositories that the client has the URL for
        - Build instructions are delivered to the client, both in the development notes and in this report
        - Much of the build process has remained the same as the original tools. (See Setup Instructions in Section VIII for details)
- Stretch Goals
    - Delivery via being merged with their upstream sources
    - Actual WiFi signal attributes, i.e. weakening connection and dropped packets.
    - CGroup Containers

VI. System Architecture

This diagram gives an overview of a typical (i.e., **prior to our changes)** Minimega-based multi-node cluster (**without Phenix**). Minimega works by running a daemon on each node in the cluster, which is, in turn, responsible for orchestrating virtual machines and containers on its respective node. Inside each VM/container is another daemon, miniccc, which is responsible for connecting to the minimega daemon on the host node over TCP or VSock protocol. Miniccc is used to perform certain administrative functions within VMs/containers, including copying files to/from VMs/containers to the host. Open vSwitch (OVS) is also run on each node, providing a highly flexible experiment data plane layer, providing network connectivity for VMs/containers to connect to each other both on the same host node and across different host nodes (i.e., Minimega supports clustering).

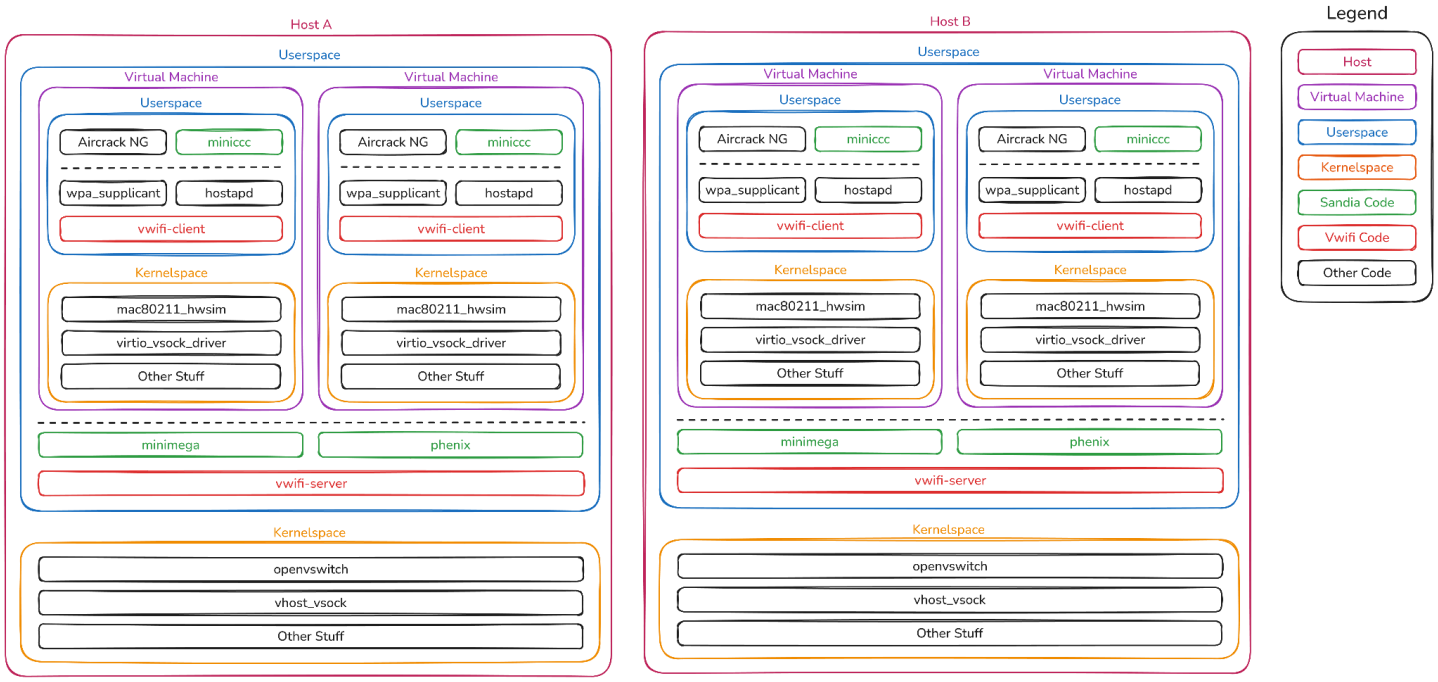
Architecture Overview (Before)



Source: sandia-minimega/minimega

This diagram illustrates the overall architecture of a modified (i.e., **with our changes**) Minimega cluster **with Phenix integration**. The main difference between the architecture before and after our changes is that we add a few additional daemons. vwifi-server is run on the host node (and therefore spawned by the minimega daemon); it is responsible for shuttling around the raw IEEE 802.11x frames between VMs/containers, as well as simulating packet loss/latency. vwifi-client is run inside every VM/container and, together with the mac80211_hwsim kernel module, creates a virtual WiFi adapter in each VM/container that connects to the vwifi-server on the host node over VSock. To facilitate broadcasting virtual WiFi networks, we utilize hostapd, which is a standard Linux tool that turns a WiFi adapter into a software-defined Access Point (AP). We also utilize wpa_supplicant, which is another standard Linux tool that complements hostapd and allows a WiFi adapter to authenticate to a remote AP.

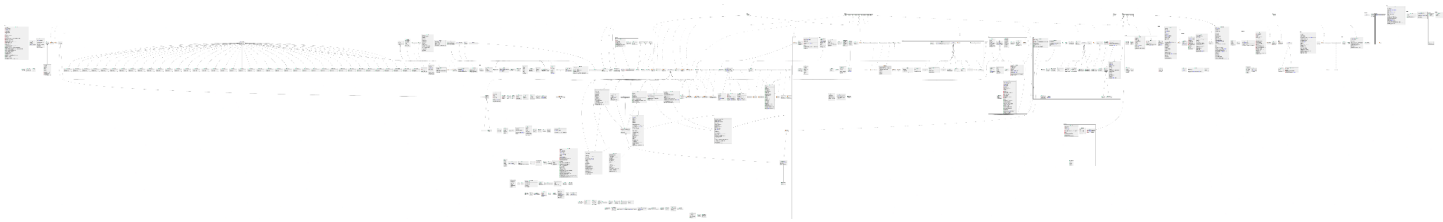
Architecture Overview (After)



Source: author-produced. View a larger version here.

This is a UML diagram of Minimega prior to any changes. We only provide this to **illustrate the complexity of Minimega's code base** - we don't expect the casual reader to be able to understand Minimega's architecture or really anything else from this. For reference, Minimega contains approximately 87,000 lines of code.

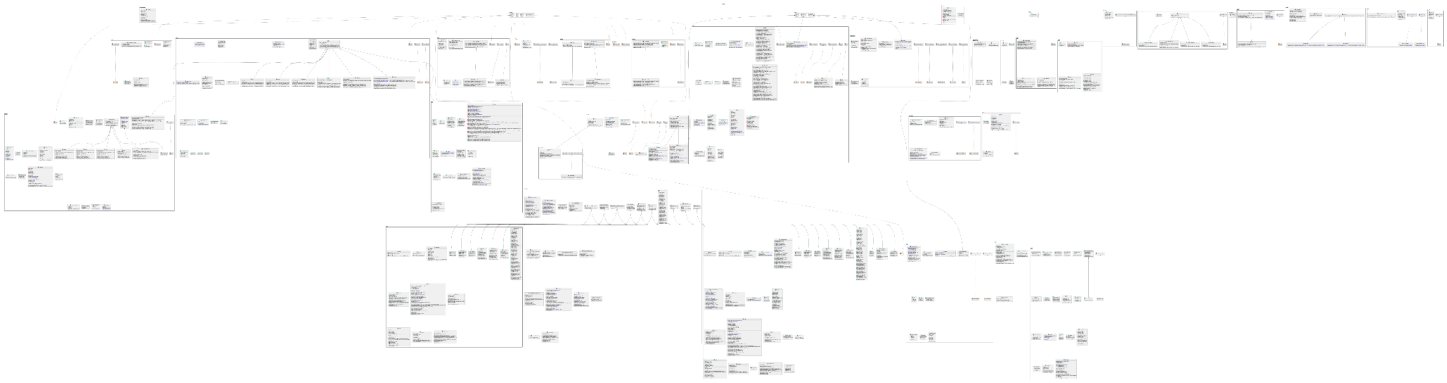
Minimega UML



Source: author-produced. View a larger version here.

This is a UML diagram of Phenix prior to any changes. Again, we only provide this to **illustrate the complexity of Phenix's code base** - we don't expect the casual reader to be able to understand Phenix's architecture or really anything else from this. For reference, Phenix contains approximately 176,000 lines of code.

Phenix UML

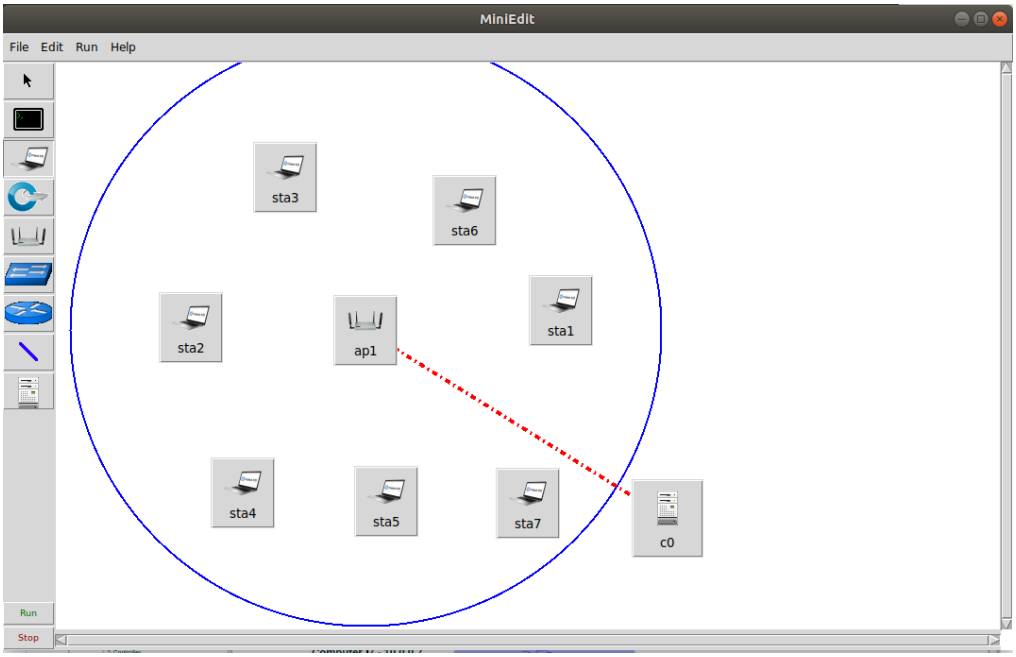


Source: author-produced. View a larger version here.

VII. Technical Design

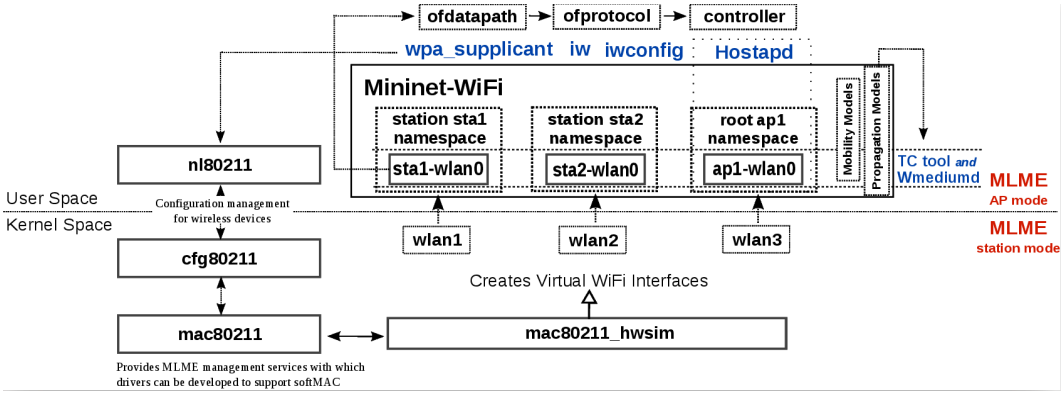
WiFi Simulation

One of the most challenging aspects of this project was determining how to actually simulate WiFi. Our client suggested looking into Mininet-WiFi at first, which had loads of promising features, including a GUI to develop the virtual topology:



Source: Mininet-WiFi project

Unfortunately, however, Mininet-WiFi was never intended for use with VMs and extensively relies on Linux namespaces to isolate the traffic for different WiFi stations. You can see this in the Mininet-WiFi architectural diagram:



Source: Mininet-WiFi project

After extensive research, modifications, and testing, we were unable to pass through the virtual WiFi adapters into the VMs, while still maintaining connectivity. Following this, we turned to a project named vwifi from sysprog21 which provided a similar functionality but was intended to work with QEMU VMs. Unfortunately, after further testing, we were unable to get it to work, running into a documented problem with it. Nevertheless, we persevered, landing on another project of the same name - vwifi from Raizo62 (which appears to have inspired the sysprog21 implementation). Thankfully, this implementation worked without any major issues and is what our final deliverable is based on. Note that all references to vwifi in this project are for the Raizo62 implementation, not the sysprog21 one.

YAML Schema

Our client wanted to extend Phenix's YAML configuration interface to also support the WiFi simulations but to do so in such a way that the WiFi configuration would be relatively simple and concise. Because hostapd and wpa_supplicant support an enormous number of features, this resulted in some challenges with mapping a relatively simple configuration interface onto hostapd/wpa_supplicant's incredibly complex configuration interfaces. Ultimately, we took inspiration from Netplan's WiFi configuration interface and arrived at something that allows reasonably concise descriptions of common WiFi configurations. For example, to create an open network (such as something you might connect to at a coffee shop), you can just use:

open.yml

```yaml
# Rest of VM network config omitted
type: wifi
wifi:
  mode: ap
  ssid: Phenix Test WLAN
  auth:
    mode: none
```

Source: author-produced with Ray.so

Or to create a typical WPA2 Personal network (such as the kind you are likely using at home), you can just use:

wpa2-personal.yml

```yaml
# Rest of VM network config omitted
type: wifi
wifi:
  mode: ap
  ssid: Phenix Test WLAN
  auth:
    mode: wpa2-personal
    password: "bonjour no wait wifi"
```

Source: author-produced with Ray.so

Or to create a WPA2 Enterprise Network (similar to Eduroam), you can just use use:

wpa2-enterprise.yml

```yaml
# Rest of VM network config omitted
type: wifi
wifi:
  mode: ap
  ssid: Phenix Test WLAN
  auth:
    mode: wpa2-enterprise
    method: tls
    ca_certificate: /phenix/test-assets/tls/ca.crt
    certificate: /phenix/test-assets/tls/router.crt
    key: /phenix/test-assets/tls/router.key
```

Source: author-produced with Ray.so

Note that this requires the appropriate cryptographic materials to be created prior to use, but this can be accomplished with OpenSSL in just a few minutes.

Miscellaneous Issues

Some of the other miscellaneous issues we ran into are:

- Phenix and to a lesser extent Minimega are not documented/maintained particularly well.
- Environment setup, particularly for Phenix, was quite challenging as a result of the lack of documentation.
- The time between making a change and testing if the change was correct was considerably slowed by the time it took to rebuild Phenix using Docker Compose. Eventually, we succeeded in adding VSCode debug configurations, massively reducing this latency and improving development velocity.

Nevertheless, we were able to overcome all of these challenges.

Prerequisites

All project testing has been completed on Ubuntu 24.04 LTS, running on bare metal (i.e., not in a VM); there may be issues if another distribution or a VM is used. There is no Windows or macOS support for Minimega/Phenix at this time. Please refer to Appendix B - Setup Instructions for step-by-step instructions on how to set up Minimega and Phenix for testing.

Test Cases

Unit/Integration Tests

For each of the below cases, run the corresponding action in the Minimega repository root.

Test Name	Command	Expected Result	Actual Result
Minimega Builds	`./scripts/build.bash`	Minimega builds without errors	Minimega builds without errors
Minimega's Automated Tests Pass	`./scripts/test.bash`	Automated tests pass without errors	Automated tests pass without errors
Minimega Starts	`sudo ./bin/minimega --filepath /phenix/images --level debug`	Minimega starts without errors	Minimega starts without errors

Minifuzzer Passes	`sudo ./bin/minimega --base /tmp/minimega --level debug --nostdin`
	`as root`
	`# Once the above is running, then run`
	`sudo ./bin/minifuzzer --level debug`
		Minifuzzer runs with the same errors as the upstream version of Minimega (As noted above, Minifuzzer errors out on the original code; this tests to make sure that we aren't introducing more errors)	Minifuzzer runs with the same errors as the upstream version of Minimega

End-To-End Tests

For each of the below cases, upload the corresponding topology file you downloaded earlier (from this folder), per the setup instructions.

The test cases are:

- Ethernet
  - Topology file: `ethernet.yml`
  - Expected result: Each VM in the experiment is able to access and connect to the ethernet network. This is to ensure we didn't break any pre-existing functionality.
  - Actual result: Success

- WiFi Without Authentication
  - Topology file: `wifi-open.yml`
  - Expected result: Each VM in the experiment is able to access and connect to the WiFi Network.
  - Actual result: Success
- WiFi With WEP Authentication
  - Topology file: `wifi-wep.yml`
  - Expected result: Each VM in the experiment is able to access and connect to the WiFi Network, given that they have the correct password.
  - Actual result: Success
- WiFi With WPA2 Personal Authentication
  - Topology file: `wifi-wpa2-personal.yml`
  - Expected result: Each VM in the experiment is able to access and connect to the WiFi Network, given that they have the correct password.
  - Actual result: Success
- WiFi With WPA2 Enterprise Authentication
  - Topology file: `wifi-wpa2-enterprise.yml`
  - Expected result: Each VM in the experiment is able to access and connect to the WiFi Network, given that they have the correct TLS certificate and key.
  - Actual result: Success

```
# Initialize the directory structure
mkdir -p tls

# Initialize the certificate authority
openssl req -x509 -new -nodes -sha512 -days 365 -newkey rsa:4096
-keyout tls/ca.key -out tls/ca.crt -subj "/CN=Phenix Test CA"

# Initialize the router's key and certificate
openssl req -new -nodes -sha512 -newkey rsa:4096 -keyout
tls/router.key -out tls/router.csr -subj "/CN=Phenix Test WLAN"
openssl x509 -req -sha512 -days 365 -in tls/router.csr -CA tls/ca.crt
-CAkey tls/ca.key -CAcreateserial -out tls/router.crt

# Initialize alice's key and certificate
openssl req -new -nodes -sha512 -newkey rsa:4096 -keyout
tls/alice.key -out tls/alice.csr -subj "/CN=alice"
openssl x509 -req -sha512 -days 365 -in tls/alice.csr -CA tls/ca.crt
-CAkey tls/ca.key -CAcreateserial -out tls/alice.crt

# Initialize bob's key and certificate
openssl req -new -nodes -sha512 -newkey rsa:4096 -keyout tls/bob.key
-out tls/bob.csr -subj "/CN=bob"
openssl x509 -req -sha512 -days 365 -in tls/bob.csr -CA tls/ca.crt
-CAkey tls/ca.key -CAcreateserial -out tls/bob.crt

# Eve is an attacker and does not have a certificate
```

The test procedure is to access each of the 4 VMs (via Phenix's UI) for a given test case topology and run:

- Router (`10.0.0.1`) commands:

```
ping 10.0.0.5 -c 1  # Should work (Because Alice is on the same network)
ping 10.0.0.10 -c 1 # Should work (Because Bob is on the same network)
ping 10.0.0.15 -c 1 # Should fail (Because Eve is not on the same network)
```

- Alice (`10.0.0.5`) commands:

```
ping 10.0.0.1 -c 1  # Should work (Because router is on the same network)
```

```
ping 10.0.0.10 -c 1 # Should work (Because Bob is on the same network)
ping 10.0.0.15 -c 1 # Should fail (Because Eve is not on the same network)
```

- Bob (10.0.0.10) commands:

```
ping 10.0.0.1 -c 1  # Should work (Because router is on the same network)
ping 10.0.0.5 -c 1  # Should work (Because Alice is on the same network)
ping 10.0.0.10 -c 1 # Should fail (Because Eve is not on the same network)
```

- Eve (10.0.0.15) commands:

```
ping 10.0.0.1 -c 1  # Should fail (Because Eve is not on the same network)
ping 10.0.0.5 -c 1  # Should fail (Because Eve is not on the same network)
ping 10.0.0.10 -c 1 # Should fail (Because Eve is not on the same network)
```

Code Reviews

A code review was performed by a team member to ensure that any added code conforms to the Go style, as well as general software engineering concepts.

There are 5 Go style principles that must be followed:

1. Clarity:
    a. The purpose of any code must be clear to any readers.
    b. It must be clear what the code is doing.
2. Simplicity: The code must not be needlessly complicated.
    a. It should be easy for any reader to understand, regardless of experience with our codebase.
    b. Use the most standard tool for any given purpose. Use, in order:
        i. A core language construct (e.g.: channels, loops, and structs in Go)
        ii. A tool within a standard library
        iii. A library already included as a dependency to the project
        iv. A library not already included as a dependency to the project
3. Concision: The code has a high “signal-to-noise ratio”
    a. Avoid:
        i. Repetition
        ii. Unclear names
4. Maintainability: It will be easy for a future programmer to modify.
    a. Clarify any assumptions made.
    b. Use abstractions that make the problem clearer.
    c. Avoid unnecessary interdependence.
    d. Use a test suite to ensure future iterations continue to behave correctly.
5. Consistency

Furthermore, adherence to code style was ensured by the use of the gofmt tool - the standard formatting tool for Go.

IX. Project Ethical Considerations

The primary ethical concerns for this project were:

- Security and Privacy Concerns
    - Since the project involves simulating network attacks, there is an ethical duty to ensure that all simulations are conducted responsibly. It’s important to limit experimentation to closed, controlled environments and avoid exposure to production systems or networks, which could compromise real user data or infrastructure.
- Data Protection and Consent
    - If the project were ever expanded to simulate real-world scenarios involving personal data, obtaining consent and ensuring data privacy would be essential. Currently, using anonymized or synthetic data in simulations avoids potential privacy breaches.
- Dual-use Technology
    - The tools developed could be misused by individuals with malicious intent. It is vital to incorporate ethical guidelines and access control mechanisms in the final deployment to ensure that only authorized users with a legitimate purpose can access and use these tools.
- Transparency and Accountability
    - Communicating clearly about the project's goals and limitations helps mitigate concerns about misuse. Including transparency measures and logging activities within the tools could increase accountability.
- Bias and Fairness in Simulation
    - Ensuring that the simulation is unbiased and reflects realistic network conditions is important to avoid misrepresentations that could influence decision-making or security policies.

X. Project Completion Status

The requirements, as laid out in previous sections, have been completed and tested. Furthermore, several stretch goals have also been completed. The original MVP was to have basic ping functionality work on a WPA2 Personal network simulated within Minimega. We achieved that goal, and also integrated our product with Phenix, which allows our product to be used in exactly the same way as the original tools. And, Raizo62/vwifi is capable of simulating virtual Wi-Fi characteristics, such as packet dropping due to distance or an otherwise poor connection. Moreover, the product also supports many more types of networks than the original we promised; it supports open (no password), WEP, WPA2-Personal, and WPA2-Enterprise networks. All of these features were tested through the methodology outlined above, and they passed. Each individual type of network, including the original ethernet network, was successful in simulating the topologies as outlined above, and the results were as expected. Besides that, the original automated tests were included, and it passed the same tests as the original product. Overall, the product delivered is more useful than the MVP, and was tested thoroughly.

XI. Future Work

The future work for this project is:

- Windows Support: Raizo62/VWifi is inextricably linked with Linux. Supporting Windows (or any OSes besides those supported by VWifi) would require finding different tools, or creating new ones. This would likely entail completing most of this project's work all over again.
- Various other connection standards: Implementing these would require finding suitable tools and/or writing new virtual device drivers.
    - 5G
    - Bluetooth
    - etc.
- Merging with upstream sources. Considering the state of current pull requests, this would likely take months.
- CGroup Containers. This may be relatively simple to implement, just requiring that we run the pertinent scripts to install VWifi into the containers.
- Netflow Capture for WiFi Simulation
    - Netflow is a feature included in OVS. To capture WiFi in the same way would require implementing netflow (or similar) in VWifi, and updating Minimega's netflow creator function (in netflow.go) to call this new feature from vwifi. This would probably be enough work to be its own field-session project.

XII. Lessons Learned

Our project was a unique situation within the context of the class; we were not working on a brand-new project or one that was a continuation from previous field sessions. Instead, we were working with and modifying existing tools to make them better suit our client's needs. There are several ramifications of this that led to challenges, which we could better approach with what we know now:

- Code quality:
    - Code quality is hard (i.e. practically impossible) to maintain across our repository, considering that we were working with 100s of thousands of lines of code from other developers. All we could do was ensure that the code that we added was sufficiently documented, tested, and reviewed.
- Documentation
    - As an extension of code quality, the tools that we were using were poorly documented, at least publicly. This led to challenges when we were trying to use the tools as originally constructed, much less our modified version. This emphasizes the importance of proper documentation, which we feel we achieved through our documentation of our development process, and build instructions.
- It is easy to get stuck looking at tools that will not suit our needs.

- We spent weeks looking at Mininet-WiFi, then sysprog21/vwifi, before finally landing on Raizo62/vwifi for our final tool. Had we looked more broadly from the beginning, we likely would have had more time for actual development.

XIII. Acknowledgments

We would like to thank our client Nicholas Blair for this learning opportunity, our advisor Kristen Peglow for their help with this project, and CSM Bodeau for their feedback on this report.

XIV. Team Profile

James Crea

James is a senior in computer science at the Colorado School of Mines. James has experience with Go and Linux. James led overall development efforts and coordinated with our client.

Ran Duan

Ran Duan is a senior in computer science at the Colorado School of Mines. Ran led testing and quality assurance.

Orion Tanner Magill

Orion Tanner Magill is a senior at Colorado School of Mines earning his Computer Science degree. Orion has experience with general networking tasks through an IT internship and a CompTIA Network+ Certificate. Orion led the guest image modifications and coordinated with our advisor.

References

[1] “mac80211_hwsim - software simulator of 802.11 radio(s) for mac80211 — The Linux Kernel documentation.” Available: https://www.kernel.org/doc/html/latest/networking/mac80211_hwsim/mac80211_hwsim.html. [Accessed: Sep. 15, 2024]

[2] “cgroups,” Wikipedia. May 25, 2024. Available: https://en.wikipedia.org/w/index.php?title=Cgroups&oldid=1225572362. [Accessed: Sep. 15, 2024]

[3] “Kernel-based Virtual Machine,” Wikipedia. Sep. 04, 2024. Available: https://en.wikipedia.org/w/index.php?title=Kernel-based_Virtual_Machine&oldid=1243991148. [Accessed: Sep. 15, 2024]

[4] “What is minimega?,” minimega. Available: https://www.sandia.gov/minimega/. [Accessed: Sep. 15, 2024]

[5] “Mininet-WiFi,” Mininet-WiFi. Available: https://mininet-wifi.github.io/. [Accessed: Sep. 15, 2024]

[6] “Introduction — Read the Docs Sphinx Theme 0.5.1 documentation.” Available: https://mn-wifi.readthedocs.io/en/latest/index.html. [Accessed: Sep. 15, 2024]

[7] “Open vSwitch,” Wikipedia. Aug. 15, 2024. Available: https://en.wikipedia.org/w/index.php?title=Open_vSwitch&oldid=1240404425. [Accessed: Sep. 15, 2024]

[8] “phenix documentation.” Available: https://phenix.sceptre.dev/latest/. [Accessed: Sep. 15, 2024]

[9] “Go Style Guide,” *styleguide*. Available: https://google.github.io/styleguide/go/guide. [Accessed: Oct. 17, 2024]

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
80211_hwsim	*A Linux kernel module that allows emulating IEEE802.11x compliant network adapters within the kernel. Note however that 80211_hwsim cannot be passed into a guest VM [1].*
Access Point (AP)	*A WiFi broadcasting station.*
cgroup (Control Group)	*A logical collection of processes within an operating system that can provide limited isolation between processes within the group and their counterparts outside of it. cgroups are commonly used by container run times (i.e.: Minimega or Docker) [2].*
Container	*A partial instantiation of an operating system that shares the kernel with the host. Note that while Docker is used for development, Minimega runs cgroup-based containers directly (i.e.: without Docker).*
Docker	*A container runtime for Linux which also uses cgroups.*
Go (Golang)	*A programming language developed at Google. Minimega and Phenix are both written primarily in Go.*
Hostapd	*A common Linux tool that allows a Linux machine to act as an IEEE802.11x access point for other devices to connect to. Complements wpa_supplicant.*
Hypervisor	*The software that allows a host operating system to run one or more guest operating systems (VMs) inside of it.*
IEEE802.11x	*The IEEE specification that defines WiFi.*
KVM (Kernel-based Virtual Machine)	*Type 1 hypervisor built into the Linux kernel [3].*
Minimega	*A command line tool for quickly creating VMs and/or containers [4].*
Mininet-WiFi	*A WiFi network simulator built on top of the 80211_hwsim kernel module (fork of the Mininet project). Note that despite their similar names, Mininet and Minimega have no relation [5], [6]. Further note that, while we started this project with Mininet-WiFi, we moved to Vwifi as previously mentioned.*
Open vSwitch	*A virtual layer 3 switch that Minimega uses for connecting VMs and containers [7].*
Phenix	*A tool that uses Minimega to quickly create several VMs and/or containers [8].*
QEMU (Quick EMUlator)	*In the context of Minimega, QEMU acts as a frontend for KVM.*
VM (Virtual Machine)	*A full instantiation of an operating system relying on virtualized or paravirtualized hardware provided by the host.*
Vwifi	*A WiFi network simulator built on top of the 80211_hwsim kernel module. Note that, while we started this project with Mininet-WiFi, we moved to Vwifi as previously mentioned.*
wpa_supplicant	*A common Linux tool that allows a Linux machine to authenticate to an existing IEEE802.11x wireless network. Complements hostapd.*

YAML (YAML Ain't Markup Language)	*A configuration language used by Phenix (among many other tools).*

Appendix B - Setup Instructions

Tools

You'll need to install all the tools below:

- Docker (Latest stable release) + Docker Compose (Latest stable release)
- Hashicorp Packer (Latest stable release)
    - Make sure to also install the qemu plugin:
      `packer plugins install github.com/hashicorp/qemu`
- Go (Latest stable release)
- Miscellaneous apt packages:
    - `g++`
    - `git`
    - `libnl-3-dev`
    - `libnl-genl-3-dev`
    - `libpcap-dev`
    - `make`
    - `openvswitch-common`
    - `openvswitch-switch`
    - `qemu-system`

Minimega

To build the Minimega Docker image, follow the below steps:

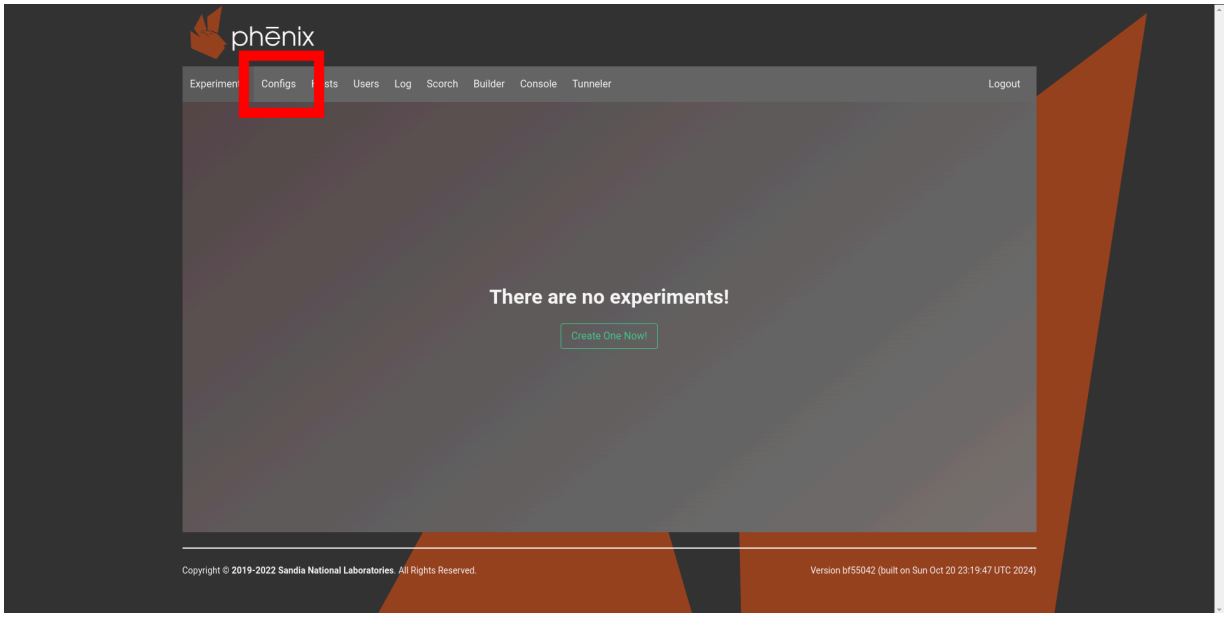
1. Clone the repository:
   ```
   git clone https://github.com/Wakeful-Cloud/minimega.git
   cd minimega
   ```
2. Build the container:
   ```
   docker build -t minimega:vwifi --file docker/Dockerfile .
   cd ..
   ```

Phenix

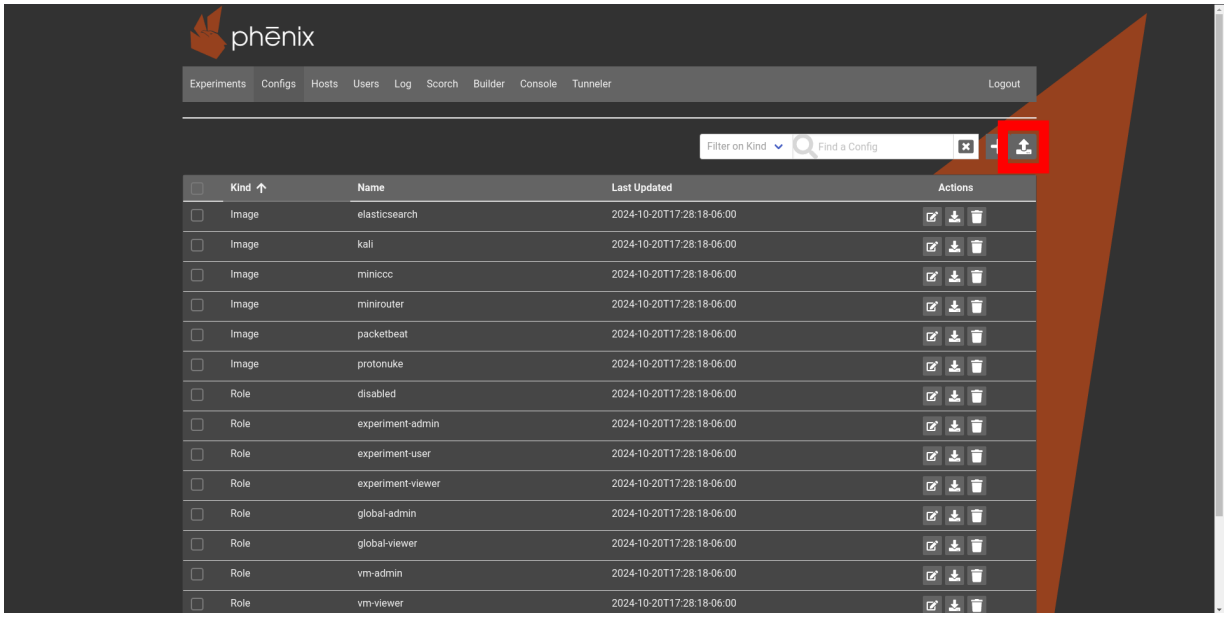
To build the Phenix Docker image and start Minimega/Phenix, follow the below steps. Note that you only need to run these for the end-to-end tests.

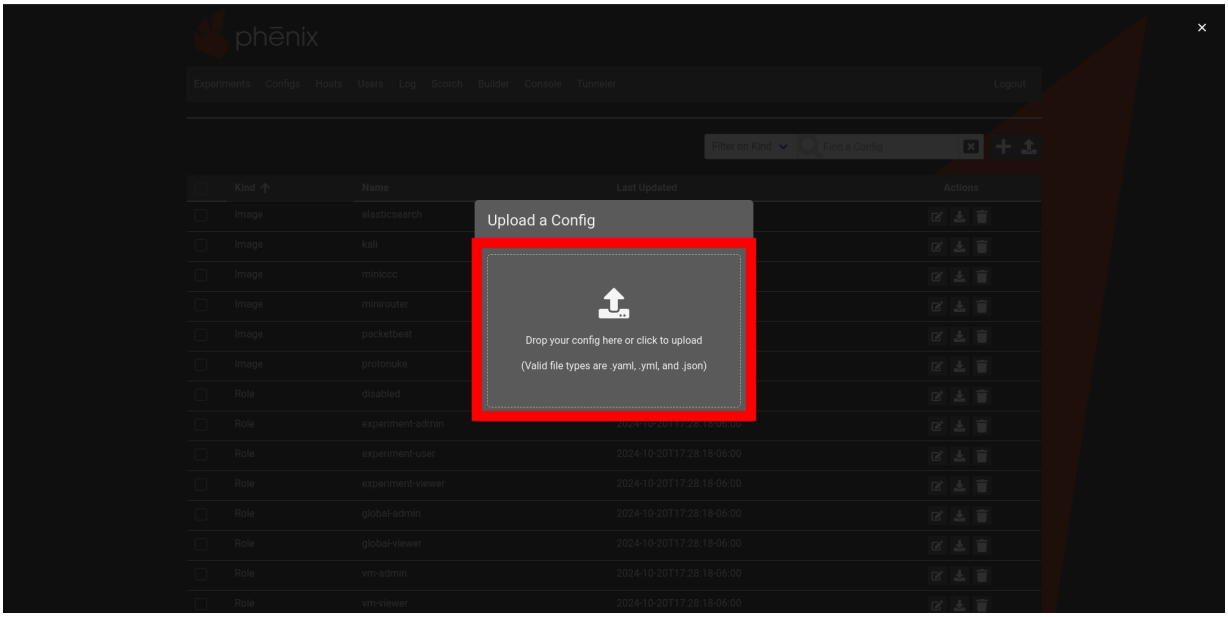
1. Clone the repository:
   ```
   git clone https://github.com/Wakeful-Cloud/sceptre-phenix.git
   cd sceptre-phenix
   ```
2. Start the Docker compose (which will automatically build the Phenix Docker image):
   ```
   cd docker
   docker compose up -d --build phenix
   cd ..
   ```
3. Build the VyOS image:
   ```
   cd hack/vyos
   docker cp minimega:/opt/minimega/bin/miniccc ./http/miniccc
   ./packer-build.sh
   ```
4. Copy the VyOS image:
   ```
   mkdir -p /phenix/images/test
   cp ./artifacts/vyos.qc2 /phenix/images/test/vyos.qc2
   cd ../..
   ```

5. Build and copy the Kali and miniccc (Ubuntu) images:
```
cd docker
docker compose exec -it phenix /bin/bash
phenix image build kali --very-verbose
phenix image build miniccc --very-verbose
cp ./kali.qc2 /phenix/images/test/kali.qc2
cp ./miniccc.qc2 /phenix/images/test/miniccc.qc2
```
6. Download the end-to-end test assets from this folder
    a. Keep the topology files (`.yml`) handy - you'll need these for step 7
    b. Move the `tls` folder to `/phenix/test-assets`:
```
mkdir -p /phenix/test-assets
mv ./tls /phenix/test-assets
```
7. Deploy the experiment
    a. Open http://localhost:3000 in your browser (You should be greeted by the Phenix UI)
    b. Go to Configs page:

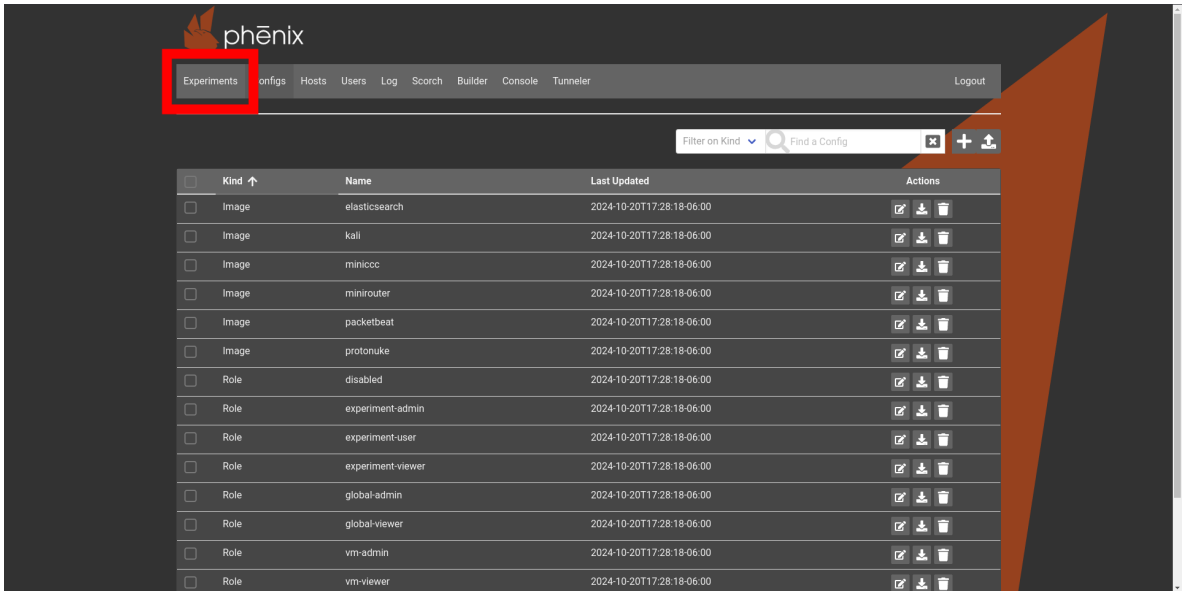


c. Upload the appropriate topology for the test case

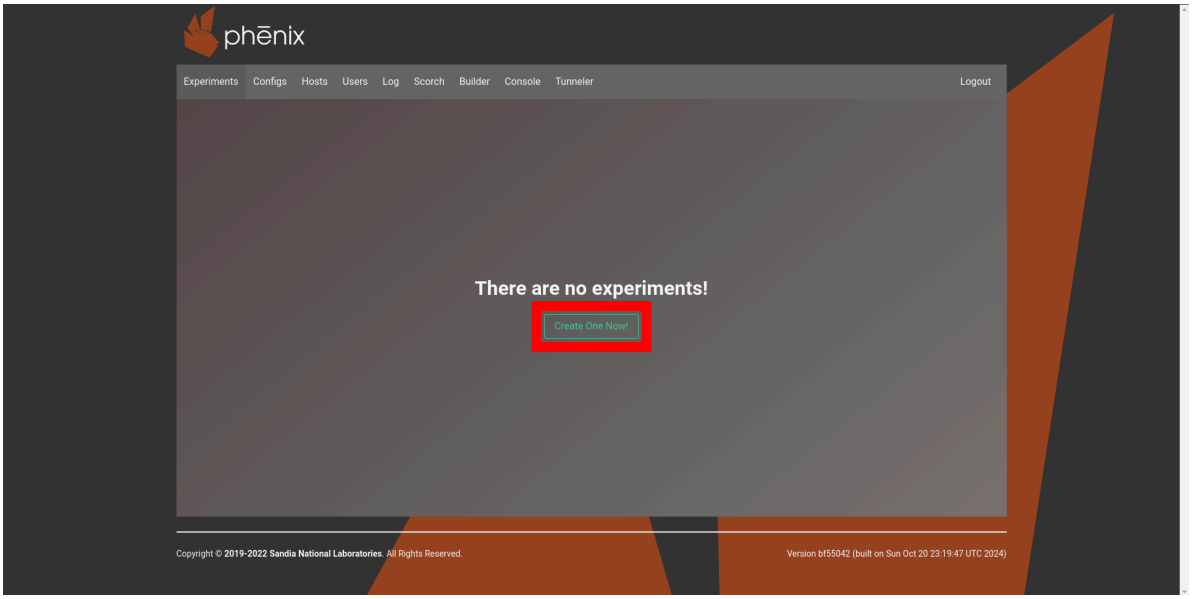


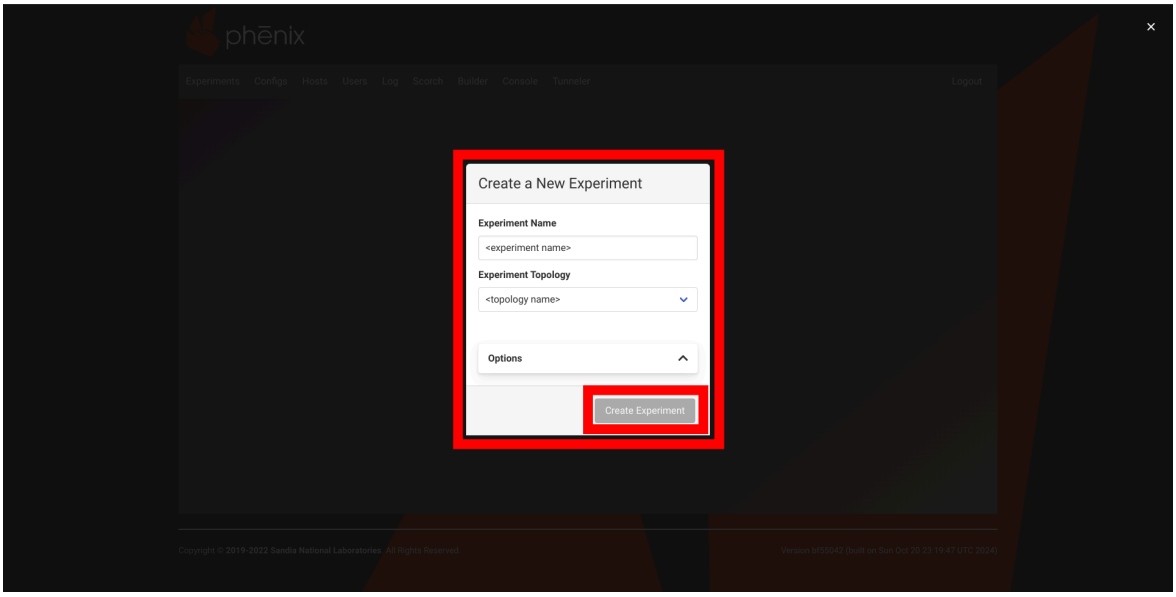


d. Go to the Experiments page

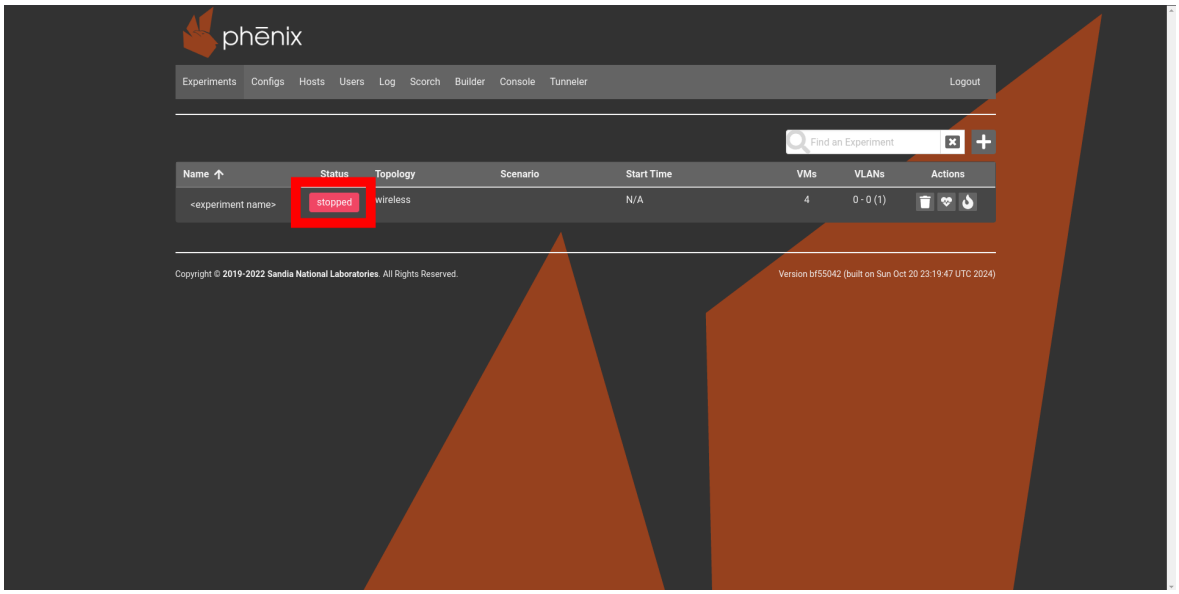


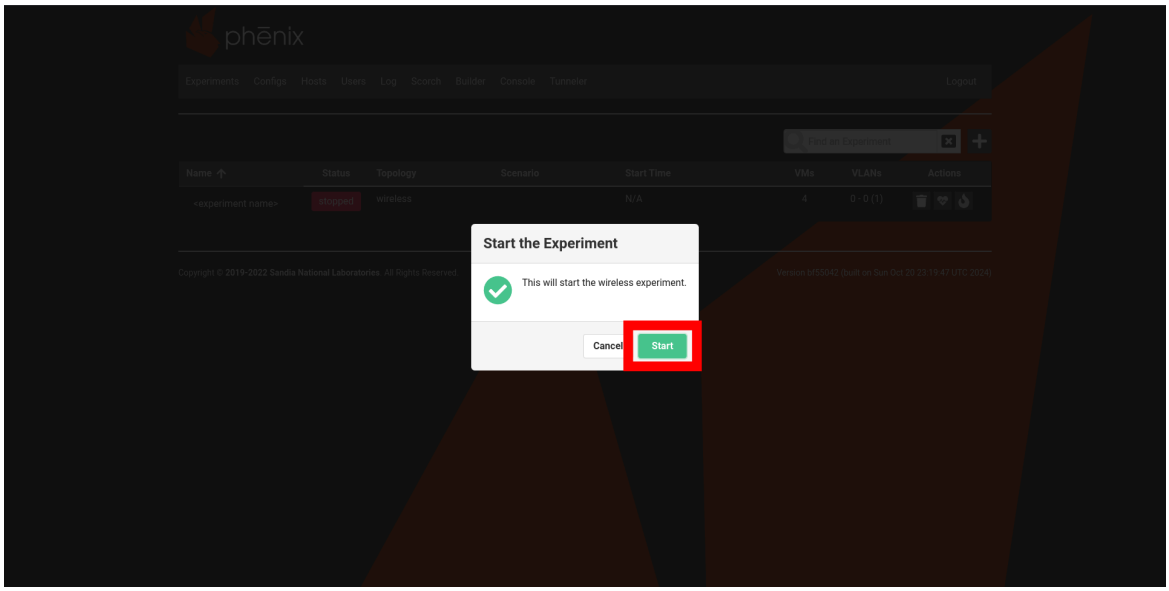
e. Create an experiment with the same name as experiment name from above (`test`), using the uploaded topology



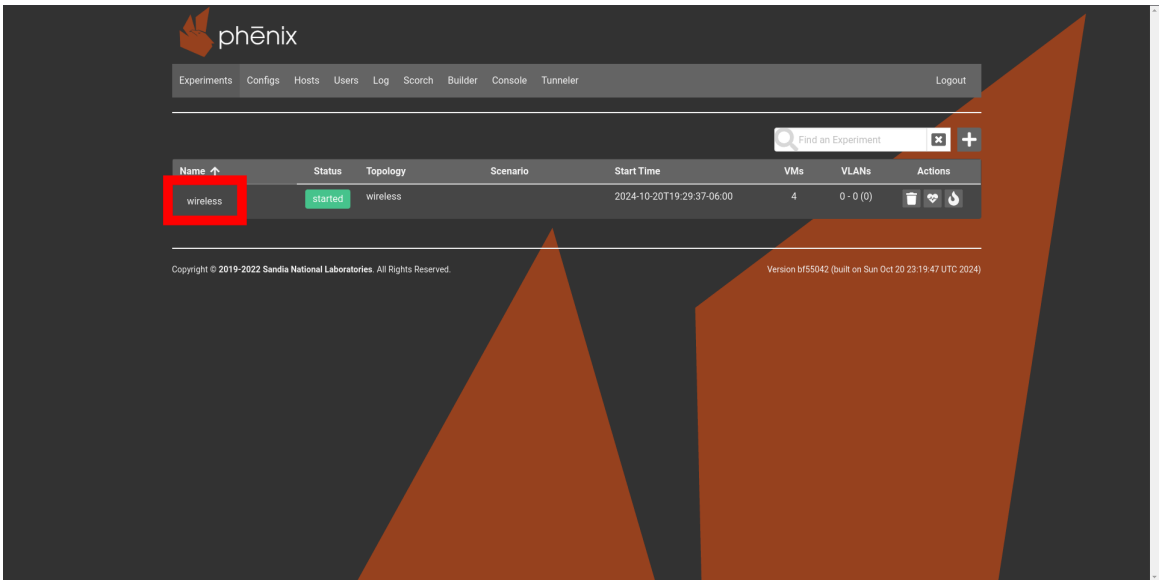


f. Start the experiment

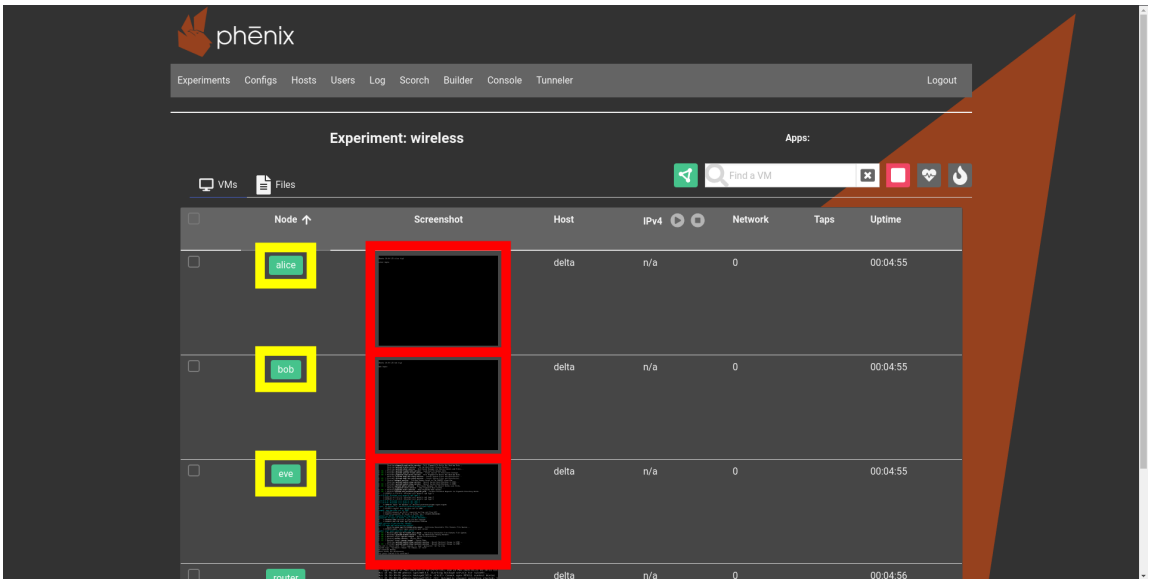




8. Access the VMs
    a. In the Phenix UI, click on the running experiment



    b. Click on the appropriate VM (i.e., router, alice, bob, or eve) to open a remote desktop connection in your browser



c. Log in using the appropriate credentials
   - For Vyatta/VyOS VMs
     - Username: `vyos`
     - Password: `vyos`
   - For all other VMs
     - Username: `root`
     - Password: none (Note: you may need to tab/click off of the username field after typing in the username for the login button to be enabled in some GUI VMs, such as Kali)

d. Access a terminal
   - For GUI VMs, press `CTRL + ALT + T` or search for the terminal in the launcher
   - For CLI VMs, you should already be in a terminal session

e. Run the appropriate commands for the test