Cyber Network Capture Generator

Final Report

SD-MAY19 Team 5

Client: Argonne National Laboratory

Advisor: Benjamin Blakely

Team Members:

Jacob Perin - *Scribe*

Luke Tang - *Meeting Facilitator*

Collin McElvain - *Chief Architect*

Abdelrahman Baz - *Chief Architect*

Hazem Abdeltawab - *Test Manager*

Bernard Ang - *Report Manager*

List of Figures

List of Definitions

ETG : Electronical Technology Group
GPL : General Public License
RSPAN : Remote Switched Port Analyzer
SPAN : Switched Port Analyzer
CAPEC : Common Attack Pattern Enumeration and Classification

0. Executive Summary

Acknowledgement

We would like to acknowledge our client, Benjamin Blakely for his contribution to our project. Benjamin provided the key insight and expertise that greatly assisted our project. He has dedicated a significant time to meet with us every week as we continue to work out the details of the project.

<u>General Problem Statement</u>
The needs to analyze traffic for hosts, applications, or services is essential in the world of computer security. Traffic is a way of describing how a computer sends information to the internet, and how the computer receives that information back. Traffic analysis is used to detect any malicious or harmful programs that can enter and harms one's computer, like a virus. Thus, preventing any undesired outcomes. This project can be used to easily track malwares and bad traffic running through a network or application.

<u>General Solution</u>
The solution of the problem is to create a program that automatically analyzes traffic data of many types, helping researchers create more innovative ways to combat malwares, and other related softwares. This proposed program will not only serve as a catalyst for researchers to come up with potential solutions, but also provide a simple understanding of Traffic and its effect in computers. The team hopes that we are able to come up with an web interface on the front end, with scripts and a program that will automate VM creation to test multiple scenarios that will be specified by the administrator, resulting in a printout of PCAP/Netflow files to show traffic.

1. Requirements specification

1.1 Functional requirements

1. **Ubiquitous Requirements**
    1.1. The hypervisor software shall be remotely accessible through a web application
    1.2. The web application shall provide secure user authentication prior to access
    1.3. The web application shall allow the user to create network capture from pre-determined combination of client, server, daemon(s), application, and activity
    1.4. The generated network flow shall be stored in a database for later access

2. **Event-driven Requirements**
    2.1. When the user selects client/server combination the hypervisor shall allocate and create two separate virtual machines
    2.2. When the hypervisor has created a virtual machine the configuration management shall establish a connection and load configuration file to virtual machine
    2.3. When configuration management has initialized a virtual machine the application shall load/execute behavioral scripts on the virtual machine
3. **State-driven Requirements**
    3.1. While the virtual machines are active the server shall store network traffic to database

1.2 Non-functional requirements

Non-functional requirements

- Performance
    - Demonstration of a working system.
- Scalability
    - Prototype will handle at least 2 virtual machines on a network.
    - Store scenario data for at least one hour long session worth of traffic flow of the virtualization network.
- Availability
    - Available only to our team of developers and permissioned users during our prototype development.
- Reliability
    - Always properly store compressed PCAP in a reliable manner.
    - Spun up virtual machine scenarios should have a 99% success rate.
- Recoverability
    - No backup data will be required for the prototype development.
- Maintainability
    - Be able to continue development of features and bug fixes of the project beyond the 492 Spring semester at Iowa State University.
- Regulatory
    - The majority of software should be written in Python 3.

    - All software incorporated in our project has been selected because of their licensing and open source status. The GNU General Public License (GNU GPL or GPL) is a widely used free software license, which guarantees end users the freedom to run, study, share and modify the software.The Apache License is a permissive open source software license — so users can release modified versions of the Apache licensed product under any license of their choice. Users can freely use, modify, distribute and sell a software licensed under the Apache License without worrying about the use of software: personal, internal or commercial.
        - KVM
            - KVM kernel module: GPL v2
            - KVM user module: LGPL v2
            - QEMU virtual CPU core library (libqemu.a) and QEMU PC system emulator: LGPL
            - Linux user mode QEMU emulator: GPL
            - BIOS files (bios.bin, vgabios.bin and vgabios-cirrus.bin): LGPL v2 or later
        - Selenium
            - Apache 2.0 License
        - Ansible
            - GNU GPL
        - OpenStack
            - ASLv2
        - PFSense
            - Apache 2.0 License
        - Squid Proxy
            - GNU GPLv2

- Usability
    - All use case functionality will be accessible through a web application.
- Interoperability
    - Accessible through the Iowa State network for the development team.
    - Virtual networks between the virtual machines should be manageable.

- Cost
  - No costs associated with software as everything is open source.
  - Hardware initial cost and maintainability for hosting VMs, data, server, client information.
- Platform compatibility
  - Web application compatible with any machine capable of hosting any popular web browser.
- Security
  - Any password information will be salted and hashed passwords stored separately from the Server.
  - Any execution of potentially malicious software should be isolated to the virtual network, this will be done with a virtual router and firewall using a combination of PFsense loaded with squid to ensure network connectivity to ensure traffic will not leave the environment. In additional any rules for KVM itself may restrict access to the outside network.
- Safety
  - All hardware should be stored and operated in a safe and responsible manner
- Ethicality
  - All work should be original for our development team with credit given to proper sources.
  - No unauthorized copying of software.

2. System Design & Development

2.1 Design plan

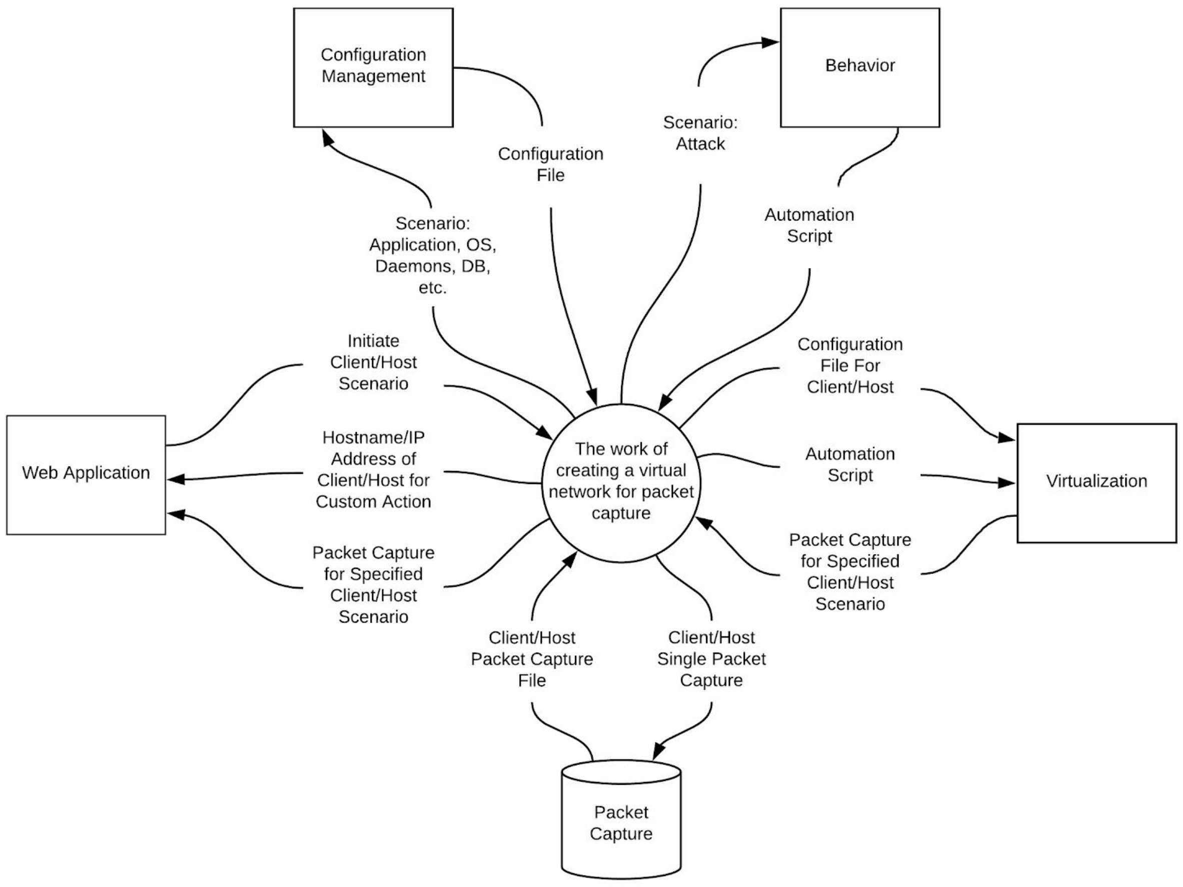

Configuration Management
Behavior
Configuration File
Scenario: Attack
Scenario: Application, OS, Daemons, DB, etc.
Automation Script
Initiate Client/Host Scenario
Configuration File For Client/Host
Web Application
Hostname/IP Address of Client/Host for Custom Action
The work of creating a virtual network for packet capture
Automation Script
Virtualization
Packet Capture for Specified Client/Host Scenario
Packet Capture for Specified Client/Host Scenario
Client/Host Packet Capture File
Client/Host Single Packet Capture
Packet Capture

Figure 1: Context Diagram

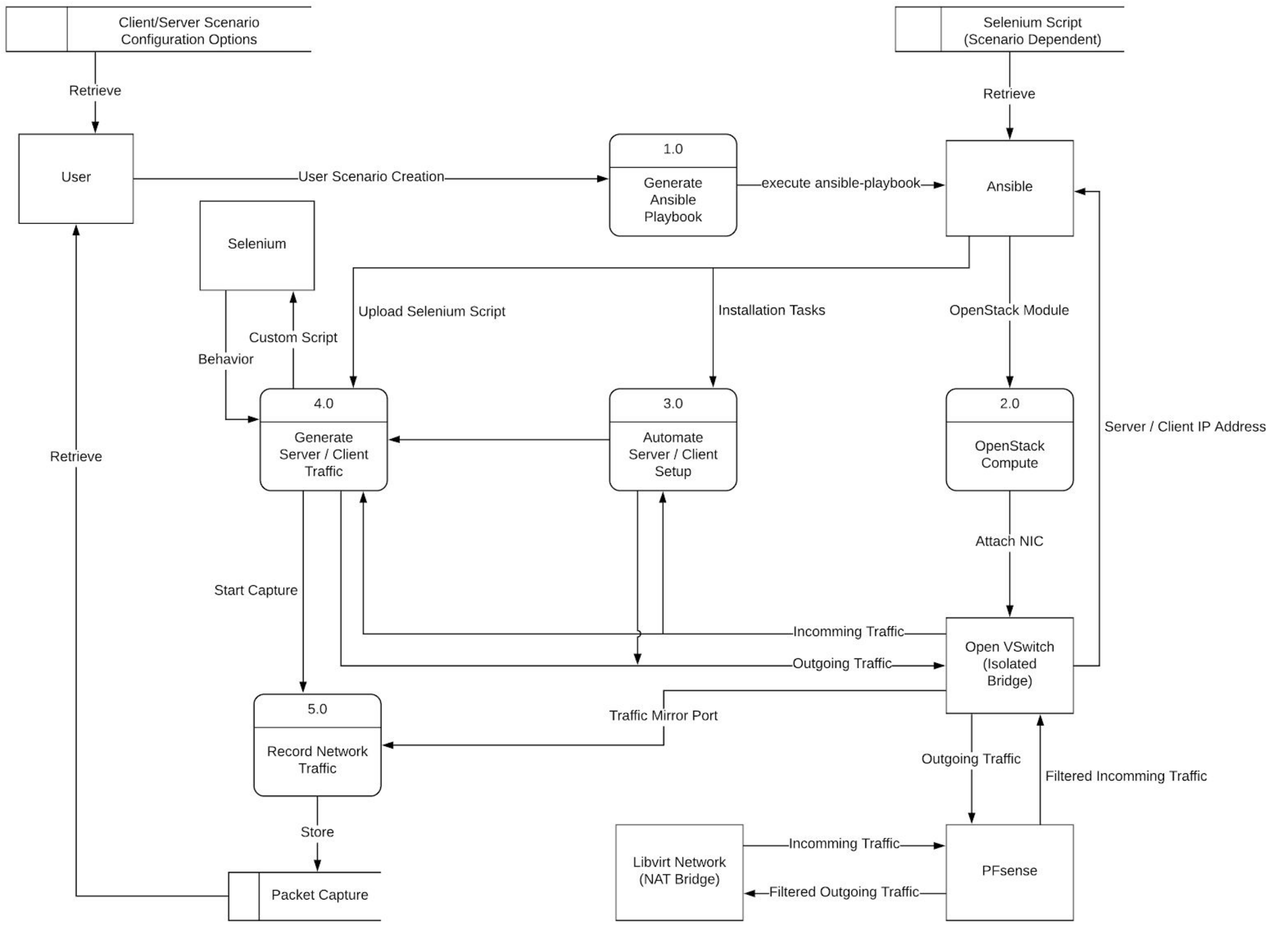


Figure 2: Data Flow Diagram

Application will consist of a simple front facing web application. This application will use a web framework and database that is optimal for backend function.

Project implementation utilizes Django, modern web framework, to allow for python language in backend.

Django backend, python, makes calls to (semi-dynamic) creation factory for Ansible Playbook. Parameters passed in are used to generate a playbook (installation instructions.) Ansible kicks off OpenStack using module (ansible feature.)

OpenStack Compute (in combination with dependent services) will install operating system, reserve resources, and establish veth pair on isolated bridge, Open VSwitch.

Ansible playbook recognizes machine boot up. Establishes SSH connection with the virtual machines in the scenario and installs scenario dependent applications.

Gateway, PFsense Router, analyzes traffic and passes determined non-malicious traffic to libvirt network (non-isolated bridge.)

Ansible determines successful application install, and uploads Selenium script for specific scenario. Ansible starts each Selenium script.

Selenium begin behavior automation once files have been started on each virtual machine for described scenario.

On completion of scenario, PCAP data sniffed via Open VSwitch bridge is stored in MySQL database for later access.

Web application history page displays new entry uploaded to MySQL. Allows for download of PCAP data for specific scenario.

2.2 Design Objectives, System Constraints, Design Trade-offs

Design Objectives:

- In a safe manner, simulate scenarios between a client and server in a virtual envioronment.
- Simple management, repeatability, and collection of scenarios from a remotely accessible application.

System Constraints:

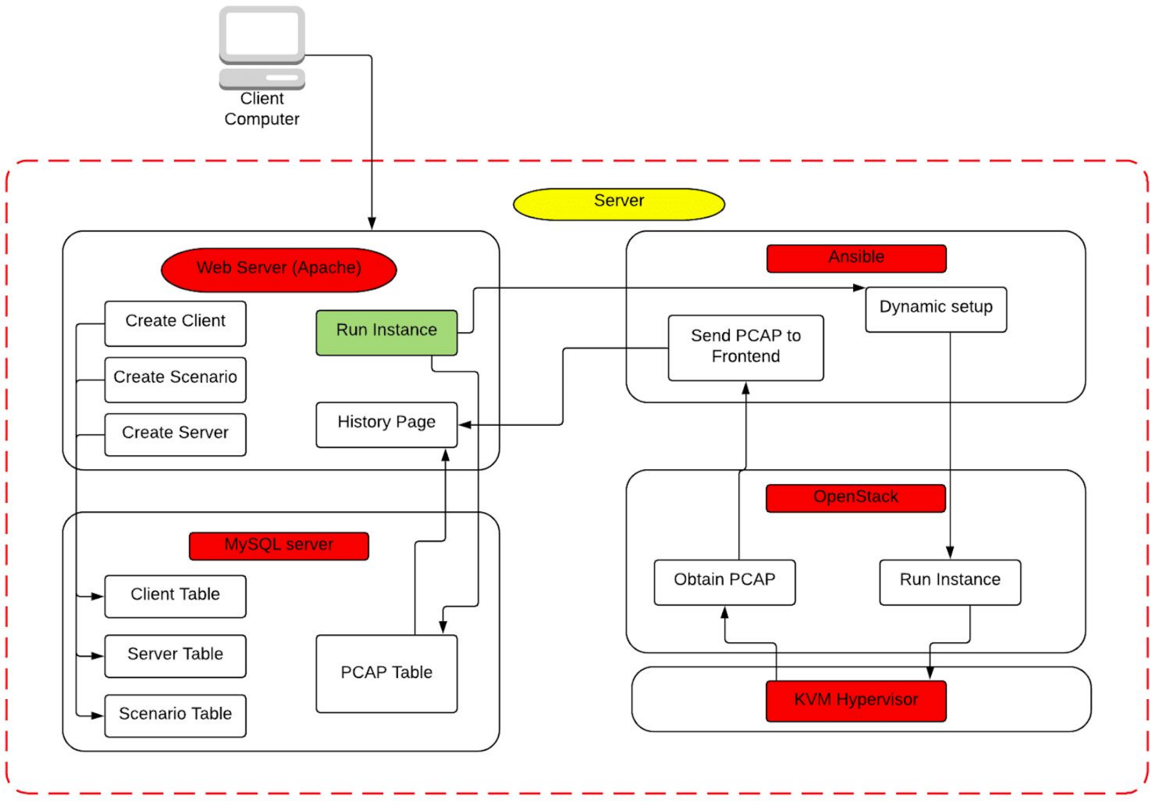
- Open source software (free)
- Single machine

Design Trade-Offs:

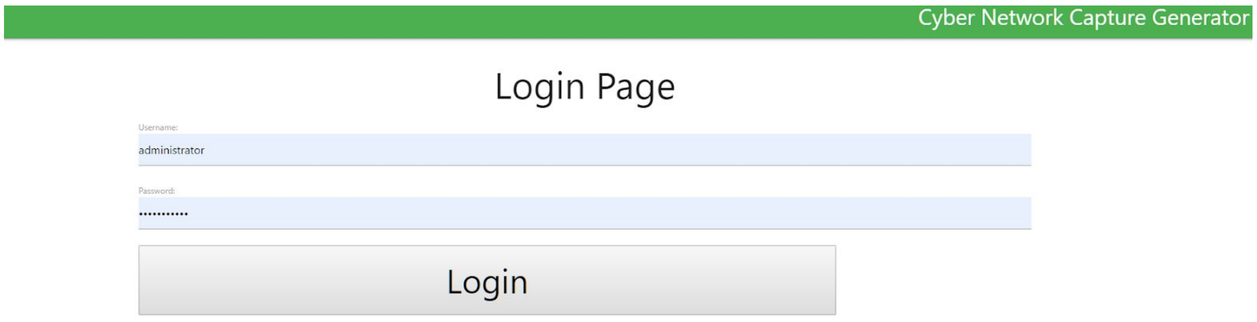
- Simplicity
    - Open source projects add complexity
    - VMware / Ansible Tower / AWS may offer more elegant solutions

2.3 Architectural Diagram, Design Block Diagram -- Modules, Interfaces

ARCHITECTURAL DIAGRAM



- Figure 3: Architectural Diagram


Cyber Network Capture Generator

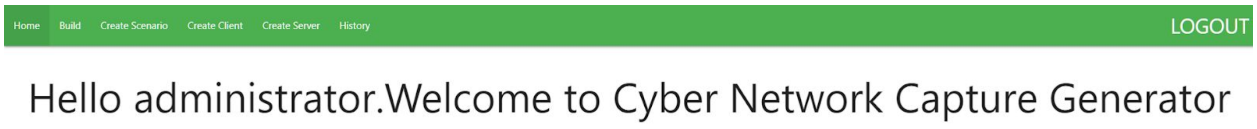
Login Page

Username:
administrator

Password:
....

Figure 4: Login Page

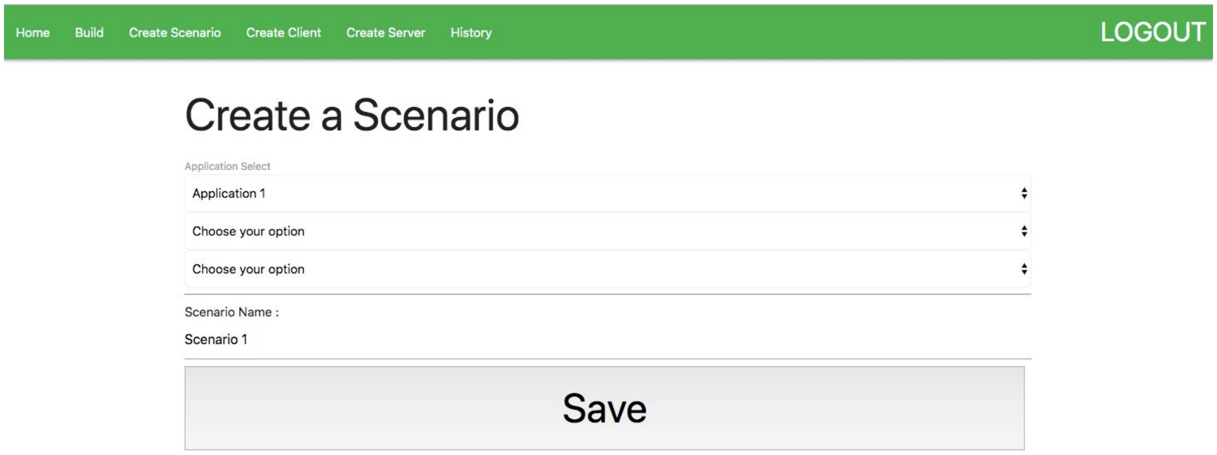
- User key in authentication in the login page



Hello administrator.Welcome to Cyber Network Capture Generator

Figure 5: Home Page

- User is then met with a Homepage. This page and the pages to come can only be accessed if the user is authorized



Create a Scenario

Application Select

Application 1

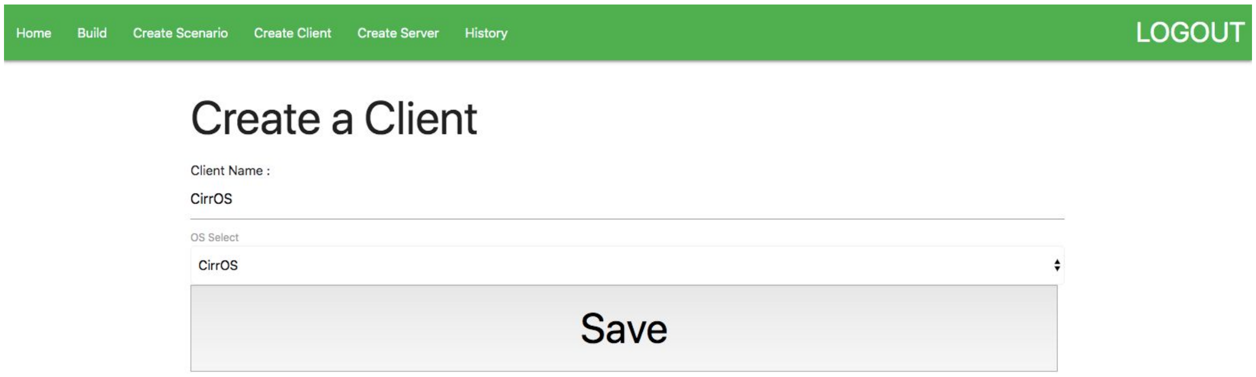
Choose your option

Choose your option

Scenario Name : Scenario 1

Figure 6: Scenario Page

- User is able to create a scenario to run on the Virtual Machine



Create a Client

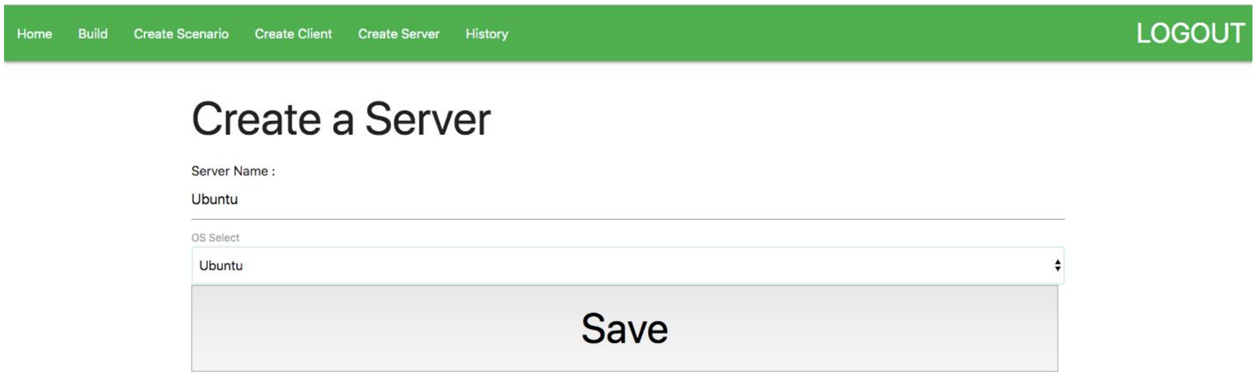
Client Name : CirrOS

OS Select

CirrOS

Figure 7: Client Page

- User is able to create a Client running on the OS preferred.



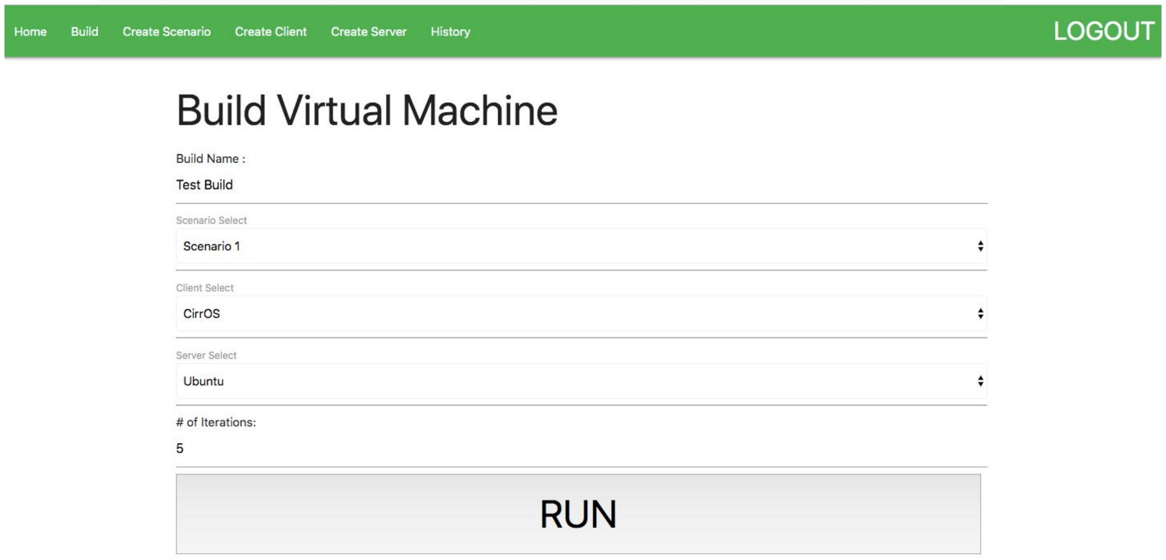
Create a Server

Server Name :
Ubuntu

OS Select
Ubuntu

Figure 8: Server Page

- User is able to create a Server running on the OS preferred.U



Build Virtual Machine

Build Name :
Test Build

Scenario Select
Scenario 1

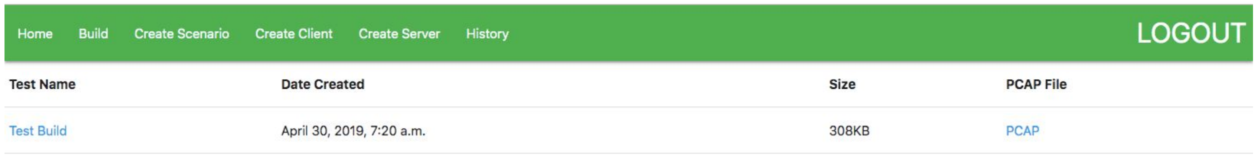
Client Select
CirrOS

Server Select
Ubuntu

# of Iterations:
5

Figure 9: Build Page

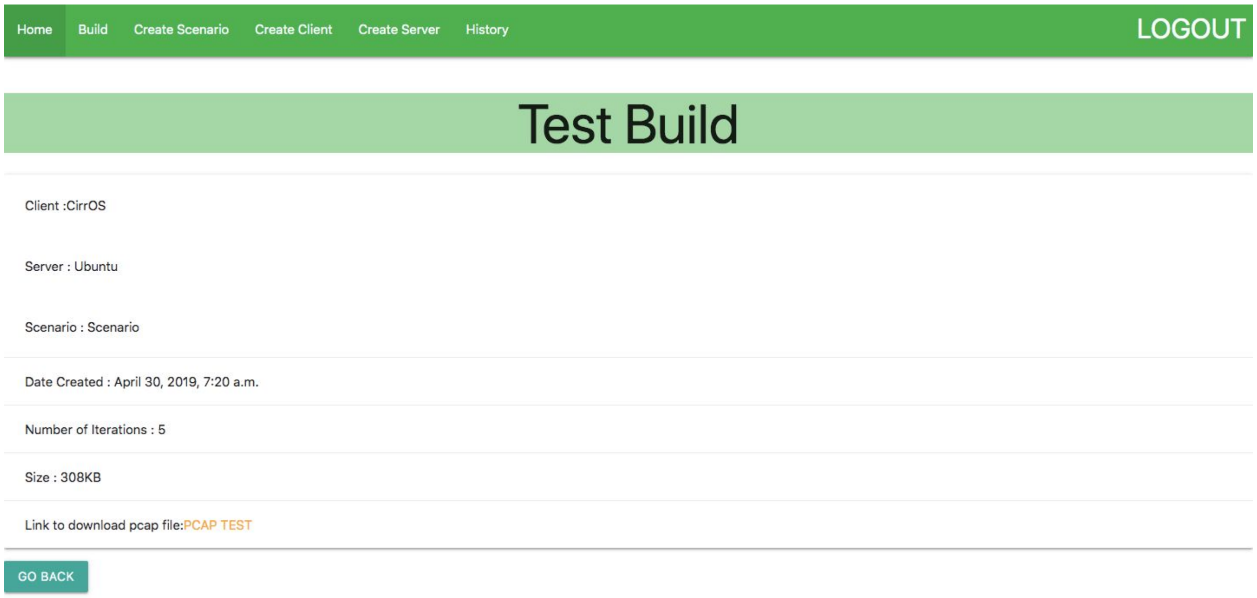
- User then is able to build a virtual machine with the following options.



Test Name	Date Created	Size	PCAP File
Test Build	April 30, 2019, 7:20 a.m.	308KB	PCAP

Figure 10: History Page

- As the system builds the VM, the information is saved on the database and displayed in the History page.



Test Build

Client :CirrOS

Server : Ubuntu

Scenario : Scenario

Date Created : April 30, 2019, 7:20 a.m.

Number of Iterations : 5

Size : 308KB

Link to download pcap file:PCAP TEST

Figure 11: Detail Page

- By clicking on a test, more details will be shown of the build

2.4 Description of Constraints and Interfaces

Interface Specifications

Our project considerations will be almost entirely software based. Dealing with virtualization, web application, and a database; the only hardware interaction will be hosting the software. Because of this we will have four main interface specifications:

1. Web application UI.
    a. This is the interactivity between the user client and the rest of the system through the centralized web application.
    b. This interface will require ease of access in a user friendly fashion.
    c. This interface should only be used by authorized and authenticated users
    d. User will be able to provide input for client, server, daemon, application, and activity based on predetermined lists.
2. Framework interface with OpenStack.
    a. OpenStack instantiates virtual machines through nova service.
3. Framework interface with Ansible.
    a. OpenStack service access will be automated through the use of dependent Ansible module
    b. Client, server, daemon, application, and activity input will be configured and installed through simple ansible playbook tasks

4. Framework interface with Database.
    a. Handle the storage and retrieval of PCAP data gathered from the virtual environment

Constraints

Our project consists of two major constraints. The first is the use of open source software. This constraint caused our team some struggle, as open source softwares' do not always integrate well with each other. So this constraint took a considerable amount of time to figure out, until we finally found the set up that worked the best for our project (Ansible, KVM, OpenStack, Django).

The other major constraint was the use of only one machine. This constraint was easy to over come as much of the front end development was done on personal computers and then uploaded to GIT. This led to easy development for our two teams.

3. Implementation

3.1 Implementation Diagram, Technologies, Software Used.

Implementation Diagram

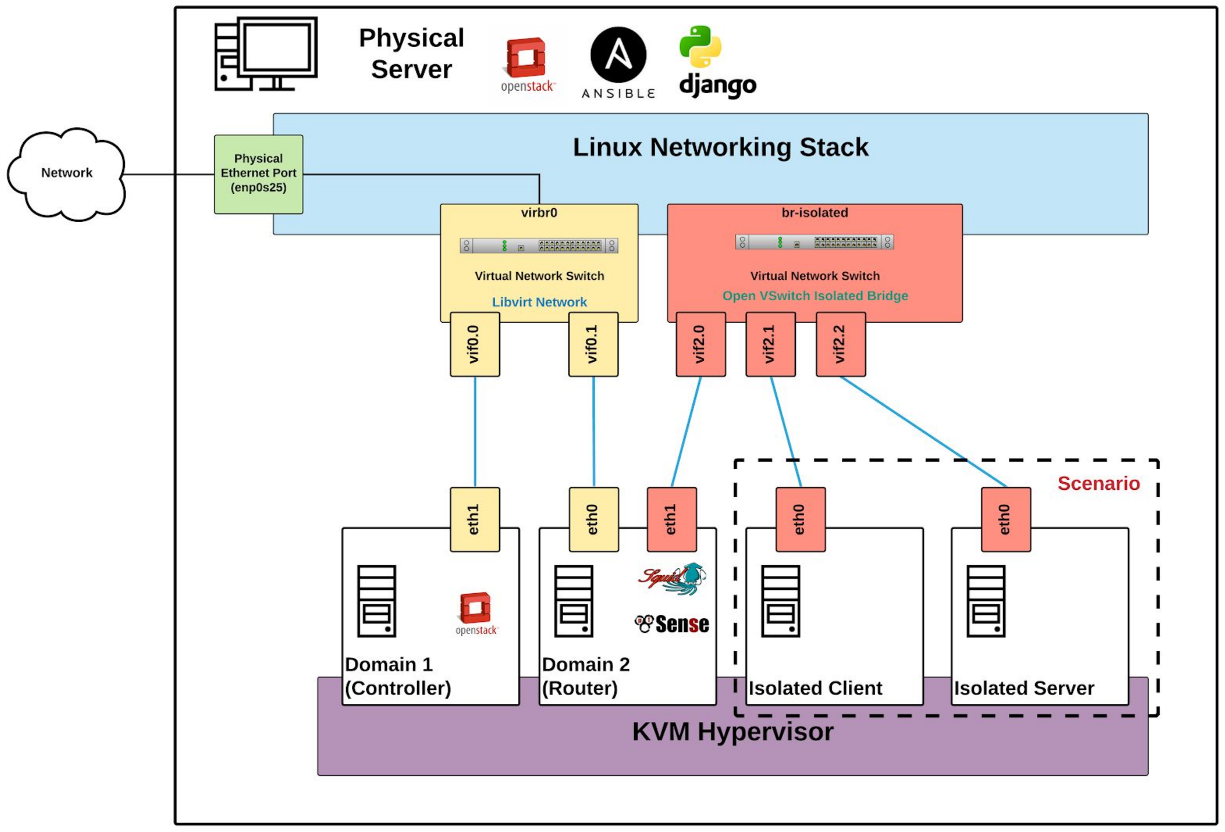


Figure 12: Implementation Diagram

Technology and Software

Hardware

The team will be using 2 desktops provided by the Electrical and Computer Engineering Department's Electronics and Technology group to be used in the department's lab as our "server" that will be running the virtual machines as well as databases. These desktops will help us to run multiple VMs without using our own personal computers. We will also be able to keep them running for a very long time.

Software

- Ansible
  - Software provisioning, configuration management, and application deployment tool. This is responsible for the initial kick off of the virtual environment after receiving the request from the web application.
- MySQL
  - Relational database management system used for storage of user credentials and captured virtual environment network traffic.
- Openstack
  - Software platform deployed as Infrastructure-as-a-service that has been utilized in this project for the deployment of our network infrastructure and initial virtual machines.
- PFsense paired with squid
  - A firewall/router software distribution that is incorporated into our project to contain and monitor network traffic while still allowing internet access to the virtual machines.
  - Squid is a caching is forwarding HTTP web proxy responsible for helping aid in the security of the system by filtering traffic.
- Python 3
  - High-level, interpreted, general-purpose programming language used for scenario scripting and use in the web framework.
- Django
  - Based on Python this web framework serves as a easy way to maintain routes to the database, web application, and virtual environment.
- Selenium
  - Selenium has been selected for its ability to execute javascript and emulate human testing scenarios when paired with a firefox or chrome driver. This is scripted in Python and is necessary for generating randomized traffic at a human pace that the system can record.
- KVM Hypervisor
  - Kernel-based Virtual Machine (KVM) is a virtualization module that allows the kernel to function as a hypervisor. We use this to expose the /dev/kvm interface

where the host feeds simulated input and output, maps the videos display back to the host, and sets up the image and address space for the guest VMs.

- Github
    - Github is widely used for maintaining source code and keeping track of work done.It's space is provided by the ETG department from Iowa State University

3.2 Rationale for Technology/Software Choices

- Ansible
    - Ansible was ultimately chosen due to its OpenStack Support
    - Very active community (compared to Chef)
    - Simplicity of playbook instantiation
- MySQL
    - Well understood technology by most team members
- Openstack
    - OpenStack is the best open source choice for virtual machine creation
    - This has many benefits over containers -- in terms of the networking aspect in project.
    - HIGH customizability for project needs
- PFsense paired with squid
    - PFsense paired with squid became the solution after several failed iterations of implementing solutions such as: OpenWRT, ISERINK, and Streisand because of incompatibility issues or lack of relevant documentation. Its ease of use and notable reputation in the information technology security fields have proven true as the router and firewall solution for our virtual network.
- Python 3
    - Python was chosen over our other option Ruby early on in the research phase because of the teams past experience with Python and its extreme flexibility in fulfilling deliverables. It has the added benefit of coming preloaded in Ubuntu operating systems that are used for the virtual guests.
- Django
    - Simple Python tie ins to backend logic
- Selenium
    - Selenium was chosen for its behavioral capabilities in most closely automatically reenacting human web browsing traffic. Its original design is for use in testing browsers as a human would use them with the flexibility of using different driver binaries such as Firefox or Chrome drivers.
- KVM Hypervisor
    - KVM is used most actively in the hypervisor community
    - Supports about any operating system (compared to Xen)
    - Default with Ubuntu
- Github

    - Github has become an industry standard for group tech development for its ability to backup revisions of code and allow concurrent development of the same project.

3.3 Applicable Standards and Best Practices

PEP8

- Code styling standard for all python code

CAPEC

- active catalog of attack vectors
- Tasks executed in our application for any given attack scenario should correlate directly to the matching vector described in the cataloged attack scenario
- Malicious applications should be limited to the intended virtual environment

IEEE Standard for Local and metropolitan area networks - Bridges and Bridged Networks

- Background: Network developed must contain malicious traffic to specific host and client. Accomplished by bridging the traffic generated by each host through a proxy.
- Section 7 as outlined in this document describes best practices when bridging a network of VLANs. Incorporating this standard will prevent network leaks that could result in consequences for victim users outside network.

4. Testing, Validation, and Evaluation

4.1 Test plan - testing method

Manual:

Frontend:

- Displayed MySQL Data to front end
    - User
    - Scenario
    - History
- Scenario creation interface
    - Creates client/server on backend
    - Updates user through creation process
    - Sends new scenario data to database
- History interface
    - Loads data from MySQL Database

Backend:

- OpenStack creates virtual machine on KVM Hypervisor
- OpenStack attaches virtual machine to isolated-bridge, and can access internet through PFsense gateway
- PCAP is captured, filtered by protocol, stored, and accessible through application.

System Testing:

- Python code is PEP-8 complaint.

Acceptance Testing:

- Server has defined host, application, and service installed.
- Client has defined host, application, and service installed.
- PCAP is captured, filtered by protocol, stored, and accessible through application.
- Application should be accessible through a web interface.

Nonfunctional testing

Performance Testing:

- Web application should be able to remotely access the server in timely manner.

Scalability Testing:

- Web application should be able to initialize two virtual machine combinations for packet capture as well as the stack virtual machine and PFsense virtual machine.
- Web application should be restricted from surpassing maximum capacity of server.

Security Testing:

- Web application is locked to specific accounts.
- All network traffic is routed through PFsense and Squid proxy.

Usability Testing:

- Web application input fields for creation create virtual machine.
- Captured network traffic is accessible through web application.

Compatibility Testing:

- Server should be runnable on multiple box configurations.

Maintainability Testing:

- Server setup should be documented and replicable.
- Design decision, models, and interfaces should be documented for maintaining the application after passing on the software

4.2 Interface testing

Frontend

- User selects scenario, server, client and number of iterations and builds the VM.
- User selects applications and updates the MySQL scenario database.
- User selects operating system for server and client and updates the MySQL database.

Backend

- PCAP capture on HTTP client/server requests between the two virtual machines.
- PCAP capture on nmap port scan from one virtual machine to another.
- PCAP capture on SSH requests between two virtual machines.
- PCAP capture on Selenium web crawler impersonating human traffic.

4.3 System integration testing

- OpenStack and Ansible communicate and can instantiate virtual machine through module
- Open VSwitch and OpenStack allow peer-to-peer traffic communication
- Open VSwitch forwards traffic through PFsense
- Libvirt forwards traffic through PFsense

5. Project and Risk Management

5.1 Task Decomposition & Roles and Responsibilities

Initial Task Approach

1. Configure and launch standalone virtual machine using KVM.
2. Configure Ansible to be able to ensure compatibility with the project.
3. Capture PCAP traffic in the virtual machine.
4. Configure and launch specific virtual network with 2 virtual machines.
5. Capture PCAP traffic in the virtual network.
6. Develop a server to handle basic requests and integration with a database.
7. Establish database/preliminary storage.
8. Establish front end web application.
9. Store preliminary PCAP data in database/preliminary storage through server.
10. Develop Ansible to handle required requests for initial kickoff of the virtual system.
11. Develop Database (or preliminary storage):
    a. Establish tables
    b. Establish dependencies
    c. Establish meaningful requests within the scope of our project
12. Develop Server:
    a. Create and Handle KVM VMs
    b. Create and Handle Django
    c. Handle data to and from database

    d. Accept and respond to requests from the front end web application
    e. Safe and secure
    f. Handle compressed PCAP files

13. Develop Web Application:
    a. Establish interactivity that suits all required use cases
    b. Make user friendly and appealing
14. Develop Automation Scripts:
    a. Selenium to generate proper traffic
    b. Client and server to generate proper traffic
    c. SSH test to generate proper traffic
    d. Nmap port scan to generate proper traffic
15. Work on stretch goals:
    a. Multiple operating systems
    b. Develop users functionality
    c. Cloud interactivity

Roles and Responsibilities

Jacob Perin - Scribe

- System Architecture Design
    - Wide-Scale integration research
        - Operating system -> application -> support / integration
    - OpenStack configuration
        - Custom service configuration (neutron, glance, etc.)
- Integration Testing
    - Open VSwitch
        - Highly customizable bridge
        - Separate OpenStack logic for integration with PFSense router
    - PFsense
        - Forwarding between Open VSwitch and Libvirt network through PFsense
        - Initial box setup
    - Ansible
        - Initial script creation
        - Integration into OpenStack
        - VM Creation feasibility
    - OpenStack
        - Custom security (aka purge it)
        - Custom setup with Compute (local machine) and Controller (VM)
- Management
    - Initial research into team member parts

    - Delegation and initial integration of team member parts

Luke Tang - Meeting Facilitator

- Initial research on Xen, KVM, capturing PCAP data, Selenium, and early software dependencies.
- Initial research and testing on network solutions focused around proxy/router/gateway solutions and possible integration with ISERink. This lead to further investigation into Squid, Streisand, SecurityOnion, PFsesnse, OpenWRT, and ESXi image conversions.
- Expanded into possible additional network features such as Wireshark, PeStudio, RegShot, TotalCommander, ProcessExploerer, ProcessMonitor, Fakenet, ApadeDNS, Hexinator, Resource Hacker for further insight on PCAP data.
- Responsible for understanding secondary consequences for vulnerabilities in the hypervisor to prevent write executions on shared folders and malware execution in a virtual environment.
- Manual creation of the virtual system to create and test required behaviors and capturing and saving their generated network traffic for the following scenarios:
    - SSH from one virtual guest to another.
    - Client/Server interaction supplying dummy HTML file.
    - Nmap port scan from one virtual guest on another .
    - Selenium web crawler emulating human traffic to the internet on a guest virtual machine.
- Proper implementation of the PFsense router/firewall loaded with squid into the virtual environment.
- Aid in the research and deployment of the initial virtual network capable of being recreated by Openstack with the full functionality required for the project.

Collin McElvain - Chief Architect

- Initial research on Ansible, Chef, Puppet, Xen, and KVM
- Experimented with small bits of Chef, until JAke found an easier route through Ansible and KVM.
- Helped with frontend decisions, mostly UI.
- Assisted Ambaz in the static IP setup for the frontend on our Apache server
- Began integration of front and backend through dynamic Python scripts. These scripts would access data through Django and create dynamic Ansible “playbook”.
- Worked with Bernard on frontend integration with scripts.
- Worked with Lucas to bring his scripts to the frontend, thus completing our 4 main scenarios the client wanted.

Abdelrahman Baz - Chief Architect

- Initial research on Netflow, Chef, Puppet, and Django
- Experimented with Chef initially but then moved to the frontend. So, experimented with Django by making simple web apps to get familiar with the Django modules
- Created a simple design for our web interface using Django, but then we switched to a different design.
- Responsible for running our web tool on Apache instead of Django’s default server which is meant for development and testing only.

- Made Apache the main server of the project and connected it to work with MySQL (instead of SQLite our testing database) and Django
- Binded a static IP (given to us from ISU) to Apache server to be able to access the web tool from other machines connected to ISU network
- Ran tests on the server to make sure it worked smoothly and fixed permissions issues related to the server-database connection

Hazem Abdeltawab - Test Manager

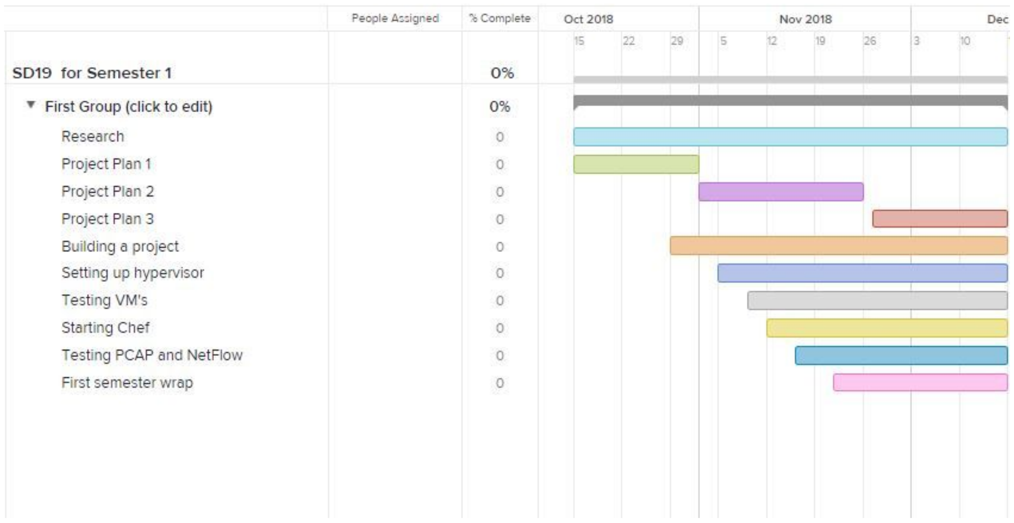
- Initial research on Python, Django, MySQL, HTML, CSS.
- Experimented with different Web building Frameworks such as Flask, Pyramid and Django to decide which Framework best suits our project.
- Worked with different Databases such as Postgresql, Oracle, SQLite, and MySQL to see which best suits Django Framework
- Responsible for replacing built-in database (SQLite) inside Django with more competent Database (MySQL).
- Designed first prototype for web interface with coding language, HTML and styling language, CSS.
- Generated tests for mock database to ensure compatibility before adding backend connections to the project.
- Added styling PEP 8 to the project to ensure compatibility with client's requirements and needs.
- Final testing to ensure connectability between frontend and backend.

Bernard Ang - Report Manager

- Research on the Django Framework, Python and how to incorporate them with our project
- Ran multiple test runs to make sure that the Django Framework will work well with our project
- Integrated mySQL together with our project as the main database
- Main designer of the currently used web interface
- Worked with Collin on the integration between the frontend and backend
- Made sure all reports that are needed for the class are done in a timely manner

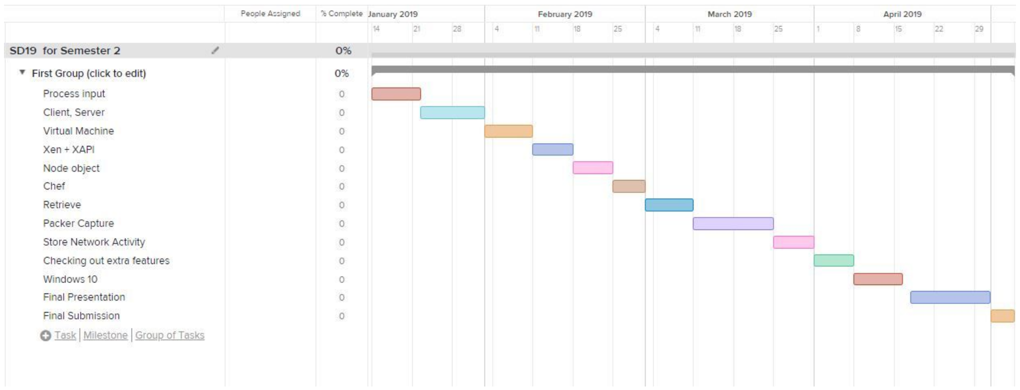
5.2 Project Schedule - Gantt Chart (proposed vs. actual)

Proposed:



	People Assigned	% Complete
SD19 for Semester 1		0%
▼ First Group (click to edit)		0%
Research		0
Project Plan 1		0
Project Plan 2		0
Project Plan 3		0
Building a project		0
Setting up hypervisor		0
Testing VM's		0
Starting Chef		0
Testing PCAP and NetFlow		0
First semester wrap		0

Figure 13: Proposed Gantt Chart (semester 1)

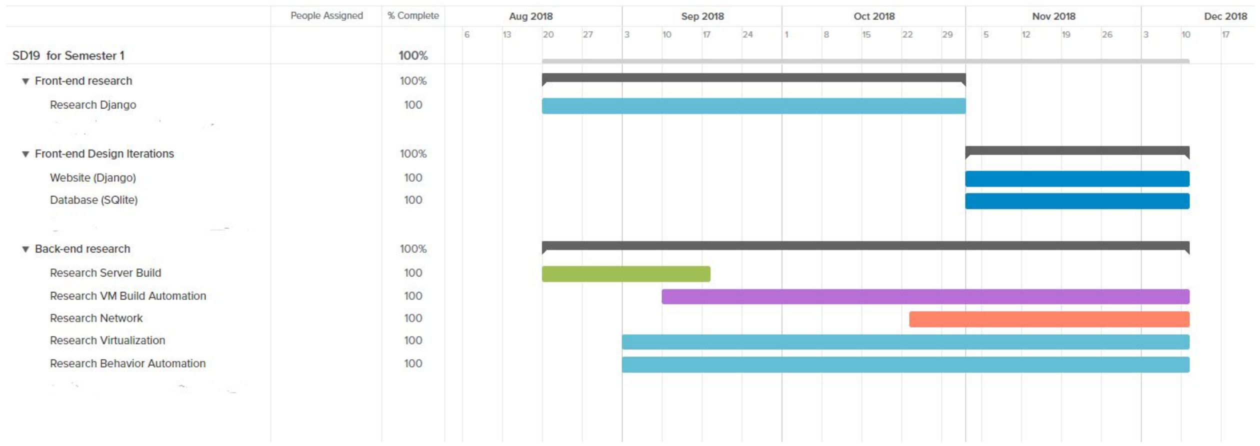


	People Assigned	% Complete
SD19 for Semester 2		0%
▼ First Group (click to edit)		0%
Process input		0
Client, Server		0
Virtual Machine		0
Xen + XAPI		0
Node object		0
Chef		0
Retrieve		0
Packer Capture		0
Store Network Activity		0
Checking out extra features		0
Windows 10		0
Final Presentation		0
Final Submission		0

Figure 14: Proposed Gantt Chart (semester 2)

Actual:

Semester 1:



	People Assigned	% Complete
SD19 for Semester 1		100%
▼ Front-end research		100%
Research Django		100
▼ Front-end Design Iterations		100%
Website (Django)		100
Database (SQlite)		100
▼ Back-end research		100%
Research Server Build		100
Research VM Build Automation		100
Research Network		100
Research Virtualization		100
Research Behavior Automation		100

Figure 15: Actual Gantt Chart (semester 1)

Semester 2:

Front-end:

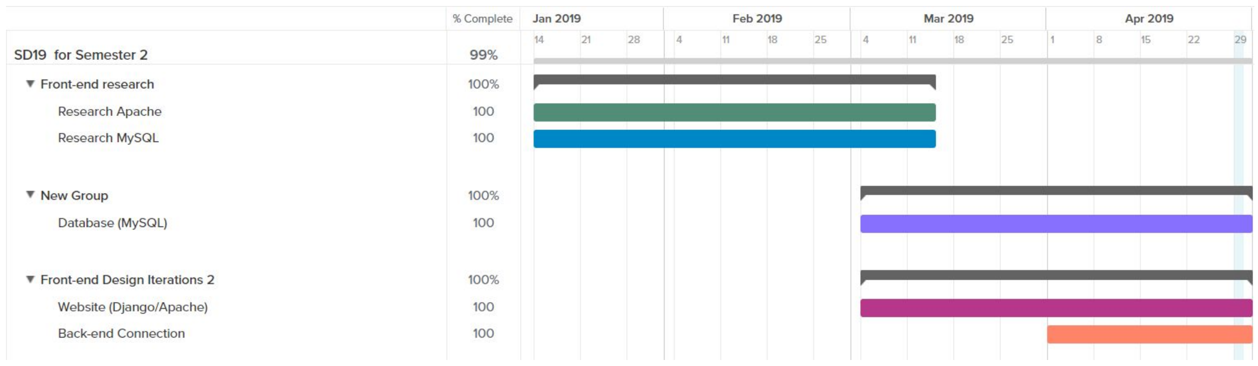


Figure 16: Actual Gantt Chart (semester 2 - frontend)

Back-end:

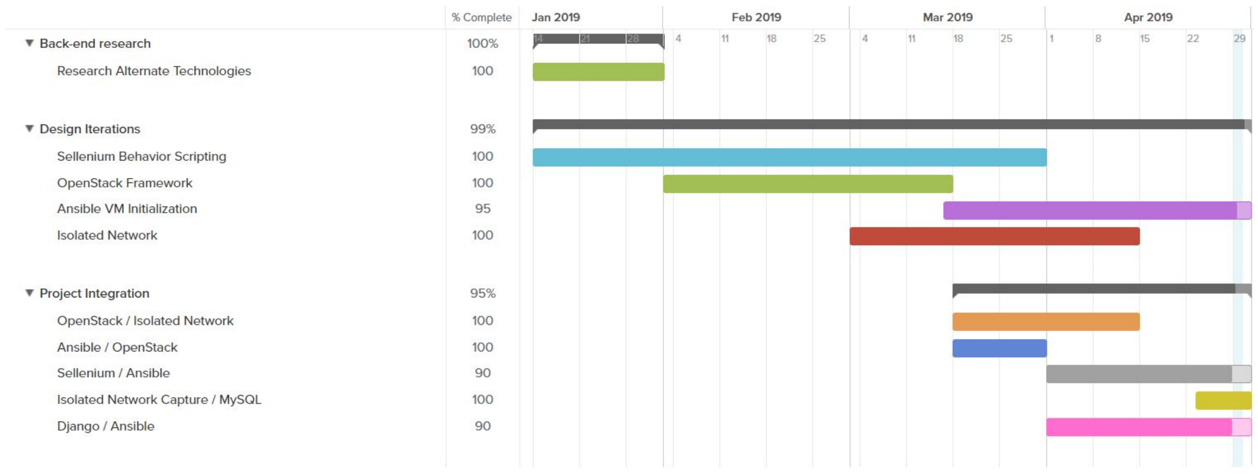


Figure 17: Actual Gantt Chart (semester 2 - backend)

5.3 Risks and Mitigation: Potential (anticipated) vs. Actual (happened) and how they were mitigated

Initial Risk Assessment:

- End product goal is ambiguous. Although individual parts (initiate hosts, generate behavior on hosts, etc.) are defined, the approach itself is unclear.
    - Ex: The “Attack Scenario” is a series of actions. The extent of this is unknown. Creating the behavior of scenario itself, such as: generating specific web traffic

    traffic, attack execution, etc. is going to be heavily scenario dependent and possibly unique for each scenario. How to approach this is in a automated system is not clear.

- Project has no clear completion criteria. If we are able to initialize a set of hosts for an attack scenario this does not define “completion.” Notion has been made toward several network protocols and attack vectors, however, this has been and will be subject to change as project progresses.
- Project is at risk to not be completed in allotted time frame, for such reasons:
    - Students designing the “network capture generator” have no experience in system design. This has, and will lead to failed design decisions.
    - Python, although ideal for back end glue is not familiar to any of the students. Learning and design patterns will be imperfect, and subject to frequent change.
    - Students designing the front end have little to no experience with web frameworks or best practices. This will lead to slow development, and will most likely need to be changed frequently.
- Scope of work is fluid. The actual extent of “attack vector automation” is unknown.
- Using multiple open source projects such as xen, openstack, and chef is possible. However, this is a lot. Especially for a student who is new to concepts such as “virtualization”. This risk, over time propagates. What seems manageable becomes less manageable a week later when student learns new concepts.
- Scripting a single attack is manageable. However, creating an environment that is built to run different attacks remotely on hosts is difficult. How the server stores attacks, accesses, and executes attacks on virtual hosts will define the manageability of the system itself.

Actual risk realization:

Our initial risk assessment were swiftly and repetitively realized throughout the continued development of this project. The ambiguous goals that were not well defined in the beginning of research and development left the project needing to be adaptable to incorporate a large variety of scenario subjects. These increased constraints demanded very flexible and open ended development that was not easy to test and sometimes left the team demoralized when hard-work was hard to bring to evidence with demonstrated working systems.

Further the lack of knowledge in virtual environments and specifically virtual networking put a huge strain on time resources for the team. Even after creating functioning networks we would have problems automating the creation process of these networks to function in the same manner that they did when they were set up manually.

The open source technology proved to be effective when the team finally found working solutions between different softwares. However the constant iterative process of creating failed states was time consuming and frustrating.

An unforeseen issue was a phase of hardware failure where the lack of memory on the physical machine caused the machine to lock up and require a manual power cycle. This impeded progress from the loss of work, as well as limit the ability to work remotely on the machine.

Risk mitigation:

To solve our first issue we redefined the project deliverables with our client to meet much more detailed and realistic goals for what the team was capable of. This included completely removing the immediate need for malicious attack vectors to be executed and tested, but setting up an environment that would allow for the safe execution of such a scenario. We further nailed down our testing operating system to Ubuntu for the virtual guests. We then defined four vastly different but important scenarios: client/server, SSH, nmap port scan, and a web crawler where the first three would be communication between the two virtual guests and all scenarios would generate observable network traffic.

Overcoming the frustrations of incompatibility and lack of expertise was an extensive process of trial and error with many hours dedicated to the research and development of new systems that could be tested to meet specifications. Many of these technologies worked together or served their purpose individually, but lacked documentation or related materials that were required in setting up the automatic process.

The hardware failures were a frustrating and time consuming setback, but this was eventually remedied by upgrading the memory the host machines to meet the demand of running its own operating system as well as four or more virtual domains with their own processes.

5.4 Lessons learned

This project showed many challenges throughout the year. A lot of these challenges came from inexperience with the tools we were using as well as changes in scope. The biggest lesson learned throughout this project was the need for communication. Our project started off by splitting into groups in order to work as fast as possible on the multiple problems ahead. However, the two teams that were formed lacked in communication throughout a period of time. This led to misunderstandings that then led to work that would not fulfill the other teams requirements.

Once communication was established as our biggest issue, we began to get into a better groove and workflow. This led to eventual completion of our project. Communication is key in a long-term project with multiple tasks. Without communication, our team was not very efficient in producing work that would integrate well. With communication everyone knew exactly what was needed to be done and our workflow was much better.

6. Conclusions

6.1 Closing remarks for the project

This project required several iterations of functional testing before arriving with a working prototype. Free and open-source constraints on all technologies left us limited with a time-consuming trial and error process to eliminate incompatibile or outdated technologies. Our team was forced to abandon Chef, Apache, Xenserver, Vagrant and many other options after extensive research and development. In the end our entire team learned in depth about networking solutions, web applications, and virtual environments.

6.2 Future work (potential directions)

This project can be expanded upon by adding support for more operating systems, number of virtual machines, as well as added scenarios. Further, the environment has the capability to contain potentially malicious softwares while maintaining the ability to capture and monitor network traffic for further investigation.