ISEAGE Traffic Generator

Final Report

Team 39
Client: Dr. Doug Jacobson/Dr. Julie Rursch
Matt Vanderwerf: Architect
Dustin Ryan-Roepsch: Quality Assurance
Josh Wallin: Requirements
Ethan Williams: Product Manager
sdmay19-39@iastate.edu
http://sdmay19-39.sd.ece.iastate.edu/

Table of Contents

1 Executive Summary

Iowa State University holds a Cyber Defense Competition (CDC) every semester[1]. Participants are split into three teams: (1) the "Blue Team", composed of college students who are attempting to run and secure several services (websites, mail servers, etc), (2) the "Red Team", composed of industry professionals who are trying to penetrate/eliminate these services, and (3) the "Green Team", composed of volunteers who test to make sure the Blue Team's servers are still providing their services.

There is a problem with this setup: the Green Team is only checking services at a fixed schedule. Since the frequency of friendly "Green traffic" reaching the Blue Team's servers is so low, the Blue Team can often assume that any other traffic is malicious. This makes reacting to the Red Team "on-the-fly" easier and mitigative actions like banning IP addresses practical.

Our project resolves this issue by generating a large amount of network traffic targeted at the Blue Team's servers. This traffic is both "Green" and "Red" (benign and malicious, resp.), with the intention of reducing the Blue Team's ability to detect the origin of the traffic.

Additionally, this web traffic generator makes it easier to teach network security tools in a classroom setting. For example, currently, if a professor at Iowa State wants to have students download and install an intrusion detection system (IDS), the students cannot observe its functions usefully (without any traffic in the enclosed ISEAGE classroom environment). By enabling our solution, instructors can generate traffic that triggers these systems, illustrating the functionality of an IDS.

2 Requirements

2.1 High-Level Functional Requirements

R1. The system shall obscure the Red and Green teams' traffic, to limit the effectiveness of basic IP banning.
R2. The system shall produce useful traffic for a classroom setting, which can be used to trigger responses from IDS.
R3. Each type of traffic shall be configurable such that, for example, a task that will perform an SSH attack should be able to be run with different password lists.

[1] http://www.iac.iastate.edu/cyber-defense-competitions/

2.2 Low-Level Functional Requirements

R4. The traffic generator shall accept a list of target IP addresses.
R5. The traffic generator shall be reconfigurable with respect to the attack/traffic types.
R6. The traffic generator shall be reconfigurable without requiring a restart.
R7. The traffic generator shall accept a configuration file.
R8. The traffic generator shall consist of a task producer and a group of task consumers.
R9. The traffic generator shall appropriately rewrite source addresses to obfuscate packet origins.
R10. The traffic generator shall produce both normal (i.e., non-attack) traffic and attack traffic.
R11. The producer node of the traffic generator shall execute on a virtual machine within the ISEAGE network.
R12. The consumer nodes shall execute within Docker containers housed on the ISEAGE network.

2.3 Non-Functional Requirements

R1. The design shall scale to support a full cyber defense competition.
R2. All software libraries employed shall be licensed such that their use is permitted in both a classroom and competition setting.
R3. The design and implementation shall follow all relevant and reasonable standards, as encountered during their elaboration.
R4. The product shall be sufficiently secured such that the client can reasonably assume outside parties will not have access to critical system settings

2.4 Use Cases

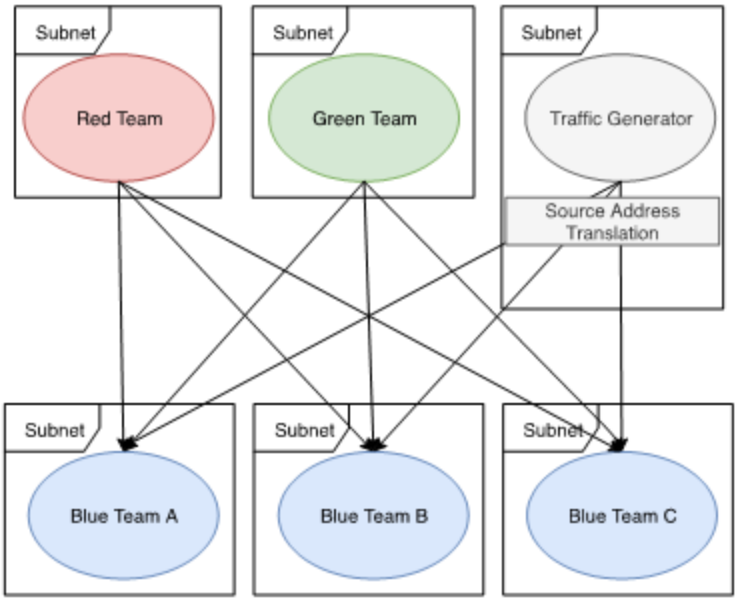
Cyber Defense Competitions (CDCs)- The tool will be used to generate traffic that looks like the Red and Green teams and will be targeted at Blue teams. This will make the competition network more closely resemble the real internet and will help teams learn about the concepts better.

Cyber defense classes- The tool will be used to generate traffic to use in several scenarios in cyber defense classes at Iowa State. First, the tool can be used in classes to demonstrate how different tools work, such as Intrusion Detection Systems (IDSs) which will be triggered by SNORT traffic types. Additionally, the tool can be used to test student projects which is currently being done manually by professors and other students. The

volume of manual traffic is not sufficient to accurately test these projects and our tool makes up for these shortcomings.

3 Design and Development

3.1 Design Diagram



3.2 Design Plan

The architecture of the tool is based on a producer-consumer pattern, in which producers formulate tasks and submit them to a task queue (RabbitMQ) that consumers access when available [7]. This architecture allows simultaneous attacks to be sent to the targets and also allows easier rerouting of responses to the correct part of the system when the ISEAGE infrastructure networking obtains that functionality. Currently ISEAGE cannot reroute packets once the source address has been spoofed. Our clients determined this was outside the scope of our project and will be implemented by a future team.

The producer will formulate attacks based on a configuration file which defines what subnets should be targeted and which types of attacks should be formulated for a given subnet. For example, the types of attacks could be a WGET request, a SNORT packet, or an SSH attempt [2]. These objects will be continuously built by the producer and put in the corresponding target subnet's queue.

After submitting a task object into a given queue, the corresponding consumer will dequeue and execute it. Tasks are discarded after completion, and the consumer will pick

up another task object to execute against the same target. Each of these attacks will be sent from a different IP, the range of which is specified in the configuration file.

Lastly, after the consumer has sent the packet, the IPTable rules on the host image rewrite the source address [9]. This allows us to change the location the packet appears to originate from to better confuse teams in the ISEAGE network.

3.3 Design Analysis

The consumers are each encapsulated in their own docker containers, which allows them to run independently of each other to minimize the amount of time targets will have between successive attacks. Another benefit to this design is that rewriting source addresses from the consumers is easier, as each container has its own virtual network interface that can be used for its own unique set of IPTable rules and thus doesn't have to track the rewrites for all of the targets.

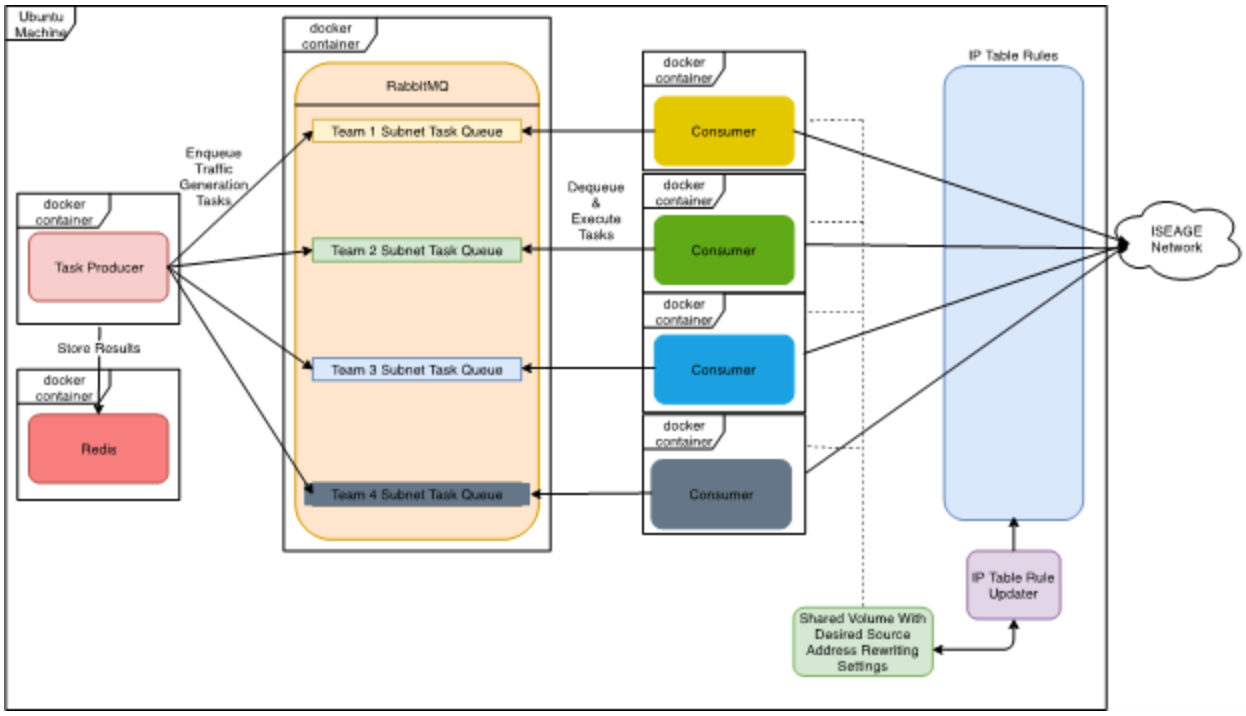
The main weakness of this design is that, if there are too many targets, the number of containers running on the system could strain resources. The client doesn't expect to reach this number of targets and, as such, this limitation is not a primary concern of the project. If this changes, alterations can be made such as migrating each container to a separate machine each with more processing capability. Additionally, it could be modified to make each consumer handle more than one target subnet.

3.4 Constraints

The ISEAGE environment will need to include previously-unseen functionality to support our product such as returning traffic to the tool, which, falling out of the scope of this project, will be completed by a different development team. We will also be limited in complex attack types of SNORT packets because other tools were not readily available in Python and would take a considerable amount of additional work to add in.

4 Implementation

4.1 Implementation Diagram



4.2 Software Used

Docker [1]

- Used for containerization of the project. This allowed us to work on the tool before we had an ISEAGE environment, with confidence that the tool would also work in the ISEAGE environment. Additionally, this makes creating the RabbitMQ and Celery producer consumer model much easier.

RabbitMQ [2]

- This is a message queue which has a queue for each of the target subnets specified in the configuration file. The producer builds tasks based on the configuration file and submits it to the corresponding queue. The consumer assigned to the given target takes from this queue to execute the tasks.

Python [3]

- Used as the main language of the project. The producer formulates Python objects which will include all information needed for a producer to execute it. The consumer is also written in Python and uses a number of libraries in order to

execute the different types of attacks. The configuration file orchestrating all of this will also be is also in Python.

WGET [4]

- One of our attack types which can be given any flags in the configuration file. The attacks are executed through the host machine which will install WGET upon container build.

SNORT [5]

- The SNORT packet signature database is used to generate simple packets that make IDS systems go off. This is another attack type used by consumers and specified in the configuration file.

SSH [6]

- Another attack type which uses lists of commonly used passwords against the target machine and executed using a Python library.

4.3 Rationale for Software Choices

Docker [1]

- We wanted to containerize our application to make the development process easier for us and make it easy to transfer to ISEAGE. We chose Docker specifically because it is the industry standard for this practice.

RabbitMQ [2]

- We chose to use a message queuing system to make the tool more scalable as we can spin up producers and consumers as needed. Once again, this technology is the industry standard for message queueing.

Python [3]

- We used Python because it's easy to do a lot with little code and many of the packages we used for traffic generation were available.

WGET, SNORT, SSH [4]

- We chose these technologies because it was a requirement of our client.

4.4 Best Practices

Configuration as Code / Containerization: By using Docker to describe the systems configuration and environment as code, we were able to test the framework on our local machines outside of the ISEAGE environment. This setup ensures repeatability of deployment, as the environment our code runs on will be exactly the same every time.

Producer-Consumer Architecture: The producer-consumer design pattern allows for high scalability and a strong separation of concerns.

Composition over inheritance: The reusable code in the system is modeled with a "has a" relationship instead of an "is a" relationship. For example, the Snort and SSH tasks both "have a" IPRewriteConfiguration, instead of being derived from IPRewritable. This avoids the "diamond problem", and also avoids long hierarchical inheritance chains.

Python type hints: Python provides a severely underutilized type hint system. We use this system to specify the argument and return types of our function, which decreases developer cognitive load and makes the code more readable.

4.5 Standards

PEP 8: A common Python standard, we used PEP 8 to ensure that our code was easily understandable and readable. Moving forward, well-documented and maintained code will be essential for future Senior Design teams that look to continue our work on this system.

5 Testing, Validation, and Evaluation

5.1 Test Plan

We have chosen to use Docker-Compose to test our tool. We created an additional yaml configuration, docker-compose-test.yaml, which spins up all necessary dependencies for testing and configures the environment appropriately.

5.2 Test Cases, Verification, Validation, & Evaluation

- Apache
    - Apache hosts a set of basic websites. The tool is configured to run HTTP tests against this container for the appropriate test pages.
    - Success: The tool must return 200 OK responses for each webpage request.
- Ubuntu
    - This is a base Ubuntu Linux container. The tool is configured to run SSH tests against this container from a predefined list of test username-password combinations.
    - Success: The tool must be able to successfully establish a secure shell to Ubuntu container for the appropriate test accounts.
- SNORT
    - This is a community-made Docker container running SNORT. The tool is configured to generate malicious looking packets and send them to this container.

    - Success: The container should sound the appropriate alerts for the malicious-looking traffic the tool is generating.

The tests are run manually by building and spinning up the docker-compose-test.yaml. The output of the tasks can then be viewed from the console. When building the container images environmental variables such as $HTTP_PROXY are inherited from the environment the image is built on. This means the tests can be run both locally and within ISEAGE with full networking capabilities without any changes to code. For our project only integration testing is necessary. Since our project utilizes third-party libraries for each protocol there is no need for unit testing. Unit testing would result in simply testing the library we are using which is outside the scope of our project. We also do not have any user-level testing to perform or interfaces to test. By performing integration testing we can verify that our tool works within the ISEAGE environment and that the source addresses of packets are successfully rewritten and spoofed. Our test cases include tests for each protocol as they would be used in the Cyber Defense Competition. In the future, when a full Cyber Defense Competition setup (including at least 20 team subnets) is available, scalability testing can be performed to ensure that the tool is capable of generating enough traffic to be beneficial. This would be verifying that enough traffic can be generated per team to obscure green and red team traffic.

6 Project and Risk Management

6.1 Roles and Responsibilities

Ethan- Product Manager

- Setup ESXI and ISEAGE, ended up using a prebuilt solution
- Generate WGET traffic coming from tool
- Setup scheduling infrastructure
- Worked on configuration setup
- Researched adding metasploit attacks for more complex traffic

Dustin - Quality Assurance

- Setup ESXI / attempt to install ISEAGE - eventually scrapped for a solution already built
- Setup common configuration layout
- Generate packets based off of snort rules
- Setup multiple virtual network interfaces for each docker container

- Integrate automatic IPtable rule configuration based off of each “task” configuration

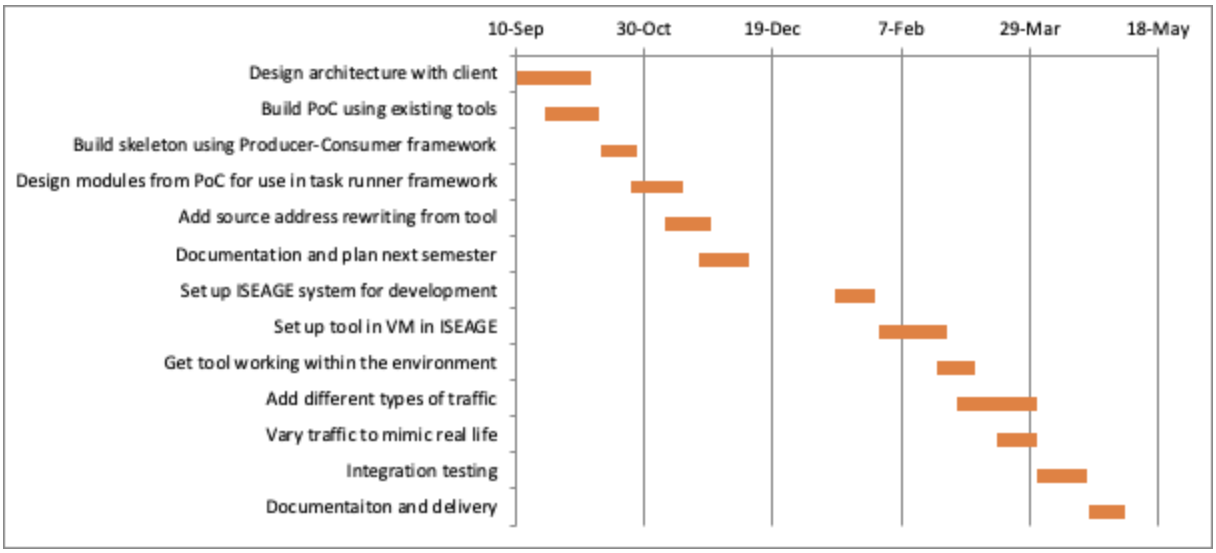
Josh - Requirements

- Demonstrate initial iptables SNAT off-network
- Develop reconfigurable source-address rewriting within ISEAGE alongside Dustin
- Write SSH dictionary attack task
- Setup virtual target machines from given images on network

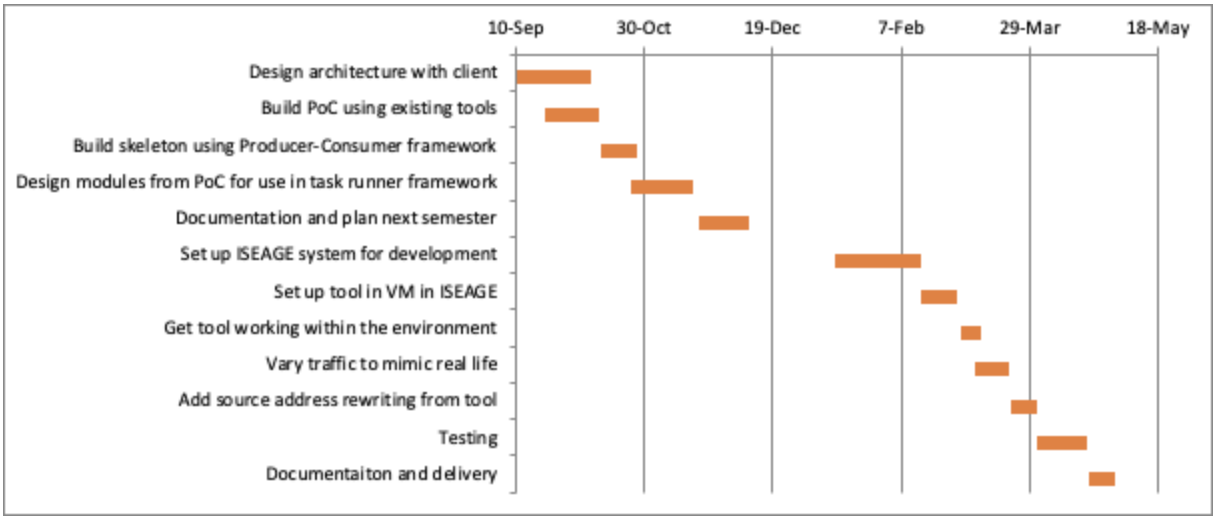
Matt- Architect

- Setup ESXI and ISEAGE, ultimately used a prebuilt solution configured mostly similarly
- Designed initial application architecture
- Design initial deployment architecture with Docker-Compose / Docker
- Created testing configuration with Docker-Compose
- Helped with joint research tasks

6.2 Project Schedule (Proposed)



6.3 Project Schedule (Actual)



6.4 Anticipated Risks

One risk we considered was the possibility that a malicious user may use our tool to effectively DDOS a service, given that we essentially created a scalable botnet. In general, ISPs use this restriction to prevent traffic from exiting the network with a source address that doesn't originate within the ISP itself; in general, such "egress filtering" would prevent this attack, but we still had to consider the possibility of similar malicious activities.

6.5 Actual Risks and Mitigation Techniques

Since the tool is potentially dangerous as an out-of-the-box botnet, our primary mitigation technique was to limit the exposure of the tool outside the university. This was accomplished by using a private repository to develop the tool with and only testing the tool in the ISEAGE environment and locally. The project is being handed off to ISEAGE exclusively and will be kept within the project whose source code is already protected from the public.

6.6 Lessons Learned

One lesson learned by our team was to never underestimate the amount of time it may take to integrate a tool with its environment. We knew this would likely take a substantial amount of time given the unique nature of our environment. This was part of the reason we chose to containerize or tool inside Docker. However, we did not anticipate how long it would take to get Docker to work inside our environment (ISEAGE). This was primarily a lesson in logistics as our main source of delay was communicating our ISEAGE networking issues to our clients as quickly as they arose. Initially there were issues in

proxy configurations which made it impossible to communicate with the internet from within ISEAGE. Following this, once Docker was able to be installed, we had issues with DNS resolution which made it impossible to pull containers in Docker. Once we communicated these issues to our clients they were able to resolve the issues in the infrastructure.

7 Conclusions

Our project is being transferred to ISEAGE as a virtual machine which can be loaded directly onto the machines connected to the competition network. This VM provides the functionality developed over the course of the project, including (1) configurable source and destination IP addresses, (2) WGET, SNORT, and SSH traffic types, and (3) a Dockerized build for scalability and easy testing.

Looking forward, this project provides multiple opportunities for work by senior design teams in future semesters. A major extension to our work requires structural changes to the ISEAGE infrastructure: simulated two-way traffic necessitates that packets directed towards fake source addresses can be properly rerouted to our machine. This will enable a new variety of simulated attacks and network activity, increasing the realism of our system. As well, another team may work in conjunction with ISEAGE developers to establish a full CI/CD pipeline, allowing automatic deployments to the CDC networks. Finally, there are a multitude of smaller changes that may be made, including the addition of new traffic types and support for alternate scheduling algorithms, all of which will enhance the verisimilitude of the environment.

8 References

[1] *Ask Solem & contributors.* Celery 4.2.0 Documentation, 2018, docs.celeryproject.org/en/latest/. Accessed 2 Dec. 2018.

[2] *Cisco.* Snort Rule Doc Search, 2018, www.snort.org/docs. Accessed 2 Dec. 2018.

[3] *Cisco.* TRex Documentation, 2018, trex-tgn.cisco.com/trex/doc/index.html. Accessed 2 Dec. 2018.

[4] *Docker.* Docker Documentation, 2018, docs.Docker.com/. Accessed 2 Dec. 2018.

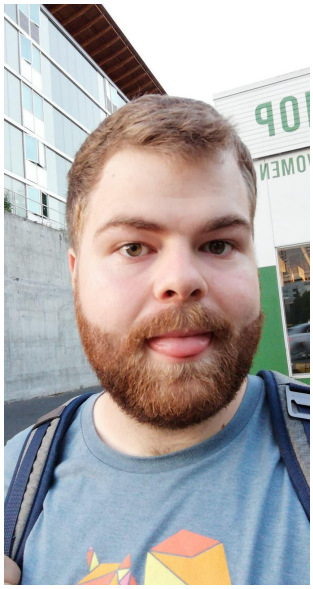
[5] *Docker.* Docker-Compose Documentation, 2018, docs.Docker.com/compose/. Accessed 2 Dec. 2018.

[6] *GNU Project - Free Software Foundation.* GNU Wget 1.20 Manual, 2018, www.gnu.org/software/wget/manual/wget.html. Accessed 2 Dec. 2018.

[7] IEEE Standard for Software Reviews and Audits," in *IEEE Std 1028-2008*, pp.1-52, 15 Aug.
2008
[8] IEEE Standard for System, Software, and Hardware Verification and Validation," in *IEEE*
Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1-2017) , pp.1-260, 29 Sept. 2017
[9] ISO/IEC/IEEE International Standard - Systems and software engineering - Requirements for acquirers and suppliers of information for users," in *ISO/IEC/IEEE 26512:2017(E)* , pp.1-47, 1 Nov. 2017
[10] ISO/IEC/IEEE International Standard - Systems and software engineering -- Software life cycle processes," in *ISO/IEC/IEEE 12207:2017(E) First edition 2017-11* , pp.1-157, 15 Nov. 2017
[11] *Metasploit*. Metasploit Documentation, 2018,
https://metasploit.help.rapid7.com/docs.
Accessed 2 Dec. 2018.
[12] *OpenBSD*. OpenSSH Manual, 2017, www.openssh.com/manual.html. Accessed 2 Dec. 2018.
[13] *Pivotal.* RabbitMQ Documentation, 2007 - Present, www.rabbitmq.com/documentation.html. Accessed 2 Dec. 2018.
[14] *Python Software Foundation.* Python 3.7.1 Documentation, 2018, docs.python.org/3/index.html. Accessed 2 Dec. 2018.
[15]*Welte, Harald, and Pablo Neira Ayuso.* Documentation about the netfilter/iptables project, 2014, www.netfilter.org/documentation/index.html. Accessed 2 Dec. 2018.

9 Team Information

Dustin Ryan-Roepsch - Quality Assurance



I'm a Senior in Computer Engineering and minor in Math. I've been a teaching assistant for CprE 185 since 2015, and in my time at ISU I have completed 5 internships / co-ops, including Google and Microsoft. While an intern, I've worked on Microsoft Outlook, Microsoft Visual Studio, and Google Cloud Platform. I will be joining Google in August 2019 to work on the Google Cloud Platform team in Seattle, Washington. I play a lot of boardgames, collect Rubik's cubes, and enjoy rhythm games like beatmania and Guitar Hero.

Matthew Vanderwerf - Architect

I'm a senior in Software Engineering with a minor in Cyber Security. While at Iowa State I have been a teaching assistant for three classes: CprE 185, SE 339, and ComS 362. I have also completed three internships at: the National Center for Supercomputing Applications, UnitedHealth Group, and Spreetail. My hobbies include playing water polo at Iowa State and having fun with friends.

Josh Wallin - Requirements

I'm a Senior in Computer Engineering and Spanish, focusing on research in techniques for formally verifying software systems. My research professors include Dr. Kristin Rozier (Dept. of Aerospace Engineering, ISU) and Dr. Robyn Lutz (Dept. of Computer Science, ISU). I previously interned at NASA Marshall Space Flight Center (ES51 - Software Systems Engineering) and Rockwell Collins (Formal Methods Team - Trusted Systems Group, ATC). My hobbies include spending time with friends, travelling, and eating as much Mexican food as humanly possible.

Ethan Williams - Product Manager

I'm a senior in Computer Engineering focusing in backend software development. Throughout my time at college I've worked as a tutor, TA for CprE 185 and 186, software intern at Nexteer Motors, and two-time software intern at Spreetail where I'll work full time after graduation. My hobbies include attempting to run long distances, traveling whenever I can, and spending time doing interesting things with friends.