ISEAGE Traffic Generator

PROJECT PLAN

Team 39
Client: Dr. Doug Jacobsen
Adviser: 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/

Revised: 12/2/2018 Version 3.0

Table of Contents

1 Introductory Material **5**

1.1 Acknowledgement 5

1.2 Problem Statement 5

1.3 Operating Environment 5

1.4 Intended Users and Intended Uses 6

1.5 Assumptions and Limitations 6

1.6 Expected End Product and Other Deliverables 7

2 Proposed Approach and Statement of Work **7**

2.1 Objective of the Task 7

2.2 Functional Requirements 7

2.3 Constraints Considerations 8

2.3.1 Non-Functional Requirements 8

2.3.2 Standards 9

2.4 Previous Work And Literature 9

2.5 Proposed Design 10

2.6 Technology Considerations 11

2.7 Safety Considerations 11

2.8 Task Approach 12

2.9 Possible Risks And Risk Management 12

2.10 Project Proposed Milestones and Evaluation Criteria 13

2.11 Project Tracking Procedures 13

2.12 Expected Results and Validation 13

2.13 Test Plan 14

3 Project Timeline, Estimated Resources, and Challenges **15**

3.1 Project Timeline 15

3.2 Feasibility Assessment 15

3.3 Personnel Effort Requirements 16
3.4 Other Resource Requirements 17
3.5 Financial Requirements 17
4 Closure Materials **17**
4.1 Conclusion 17
4.2 References 18

List of Figures

Figure 1- Project architecture

Figure 2- Task approach diagram

Figure 3- Project timeline

List of Tables

Table 1- Personnel Effort Requirements

List of Definitions

CDC: Cyber Defense Competition

A competition held at Iowa State University

IDS: Intrusion Detection System

A tool used to monitor a server and raise alarms when suspicious activity is noticed

ISEAGE: Internet-Scale Event and Attack Generation Environment

The environment that hosts the cyber defense competition

VM: Virtual Machine

A piece of software that emulates a full computer.

SNAT: Source Network Address Translation

Changing the source address of a packet to make it appear as if it's coming from a different location.

1 Introductory Material

1.1 Acknowledgement

We would like to thank Dr. Jacobsen for giving us the problem and helping us design the product so that it fits into the ISEAGE system. We would also like to thank Dr. Julie Rursch for helping us organize the project and determine the scope of each feature. Finally, thank you to ISEAGE who will be handling the integration of the product after it is completed as well as the rerouting of user responses to the product.

1.2 Problem Statement

Iowa State University holds a Cyber Defense Competition (CDC) every semester[1]. This competition is 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 traffic is malicious. This makes reacting to the Red Team "on-the-fly" easier, and doing things 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 will be 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 final product, instructors will be able to generate traffic that triggers these systems, illustrating the function of an IDS.

1.3 Operating Environment

There are two main environments that the web traffic generator will be used in, the CDC (Cyber Defense Competition) and in cyber security classes at Iowa State. The product is purely software, and will run in a vmware VM in both cases. For the CDC, the software will

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

be a VM inside of ISEAGE, while in the classroom it will either be a vm deployed on the students machine, or also in an ISEAGE environment.

1.4 Intended Users and Intended Uses

Our project will have two main intended end user groups: The Cyber Defense Competition and classes at Iowa State that focus on networking and security.

During the Cyber Defense Competition the Red Team (hackers from industry) must penetrate Blue Team (students participating) systems and secure flags or perform other malicious acts. In order to better obscure the actions of the Red Team, realistic traffic must exist constantly on the network such as would be the case in a normal network. Otherwise, it becomes trivial for the Blue Team to identify the Red Team because most of the traffic can be assumed to be the Red Team. Our ISEAGE Traffic Generator is intended to continuously generate realistic internet traffic that would commonly exist on a production network. This means generating traffic from a wide array of protocols such as: SSH, IMAP, ICMP, HTTP, HTTPS, POP3, IMAP, etc. though initially we will do HTTP and SSH and expand upon those if we have time. This will help to simulate a normal production environment network for use during the CDC.

Additionally, classes at Iowa State need realistic traffic within their lab environments for use in class. Not only must good traffic be generated as listed before, so must bad traffic. A good example of this is for use with IDS. Currently there is no great way for classes to test IDSs in Iowa State labs because again, there is no realistic traffic on the network. By generating good traffic and also bad traffic, such as a brute force password fuzzer, there is better data to analyze using the IDS. Without realistic traffic it becomes hard for students to gain a full understanding of how an IDS works because the only traffic they see is the traffic generated from other students setting up their IDS.

1.5 Assumptions and Limitations

Assumptions

- The packets that we create with fake source addresses, will be rerouted back to us (our software does not do anything to make that happen, that needs to be handled by the environment)
- The scale of the traffic needed to generate will be small enough that a VM in an ISEAGE environment can handle it
- All exploits needed for the tool will be contained in the SNORT database [2]

Limitations

- The ability to rewrite the source address of the generated traffic, to make the traffic appear like it came from several computers, will only work in an environment like ISEAGE

- Using more complex attacks that require communication with the target addresses which would increase sophistication but will likely require writing exploits used in tools manually for Python compatibility [14]
- The tool will only be able to support competitions of a certain size based on the VM it's hosted in. Events larger than a conceivable CDC is not be expected to be performant when running attacks against that many targets and in fact might even break completely

1.6 Expected End Product and Other Deliverables

The tool will be developed over the term of the next 2 academic semesters and will be delivered to Doug Jacobsen for use in ISEAGE in the first week of May in 2019. The product will be handed off as a virtual machine encapsulating the Docker image for the product [4]. The source code will also be transferred to enable ISEAGE to extend and change the tool should their requirements change in the future.

The product will be capable of running exploits through HTTP and SSH, though more may be added as the tool is built out or our clients identify more needs in the future. These exploits will be configurable through an easy-to-understand configuration format allowing the organizers to use the tool with little technical knowledge required.

The second item that will be transferred to ISEAGE in early May is extensive documentation in two parts: Integration/use and design. The integration/use documentation will detail how to integrate the product into existing environments as well as how to run the program to generate traffic for the desired targets correctly. The design documentation will detail the design and implementation of the product's source code. This is intended to give ISEAGE the ability to extend the product should their requirements change in the future.

2 Proposed Approach and Statement of Work

2.1 Objective of the Task

To produce a configurable piece of software that can be used during the CDC and cyber security classrooms to obfuscate the red and green team's location on the network, and provide cyber security students with interesting data to play with while learning about IDS' (intrusion detection systems).

2.2 Functional Requirements

2.1.1.1 High-Level 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.

2.1.1.2 Low-Level 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 Constraints Considerations

The ISEAGE environment will need to include previously-unseen functionality to support our product, which, falling out of the scope of this project, will be completed by its development team. We will also be limited for simple attacks to the SNORT database. We assume at this point that this will be sufficient, though it may require us to create our own exploits. This will not interfere with the planned architecture of the tool.

2.3.1 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 (see Standards below).
R4. The product shall be sufficiently secured such that the client can reasonably assume outside parties will not have access to critical system settings
R5. The product will not cause harm or interfere with the operation of computers that are not participating in the cyber defense competition.

2.3.2 Standards

In order to maintain quality in accordance with widely-accepted best practices, we will focus on conformance with the following IEEE standards:

S1. IEEE 26512-2017 (Requirements for acquirers and suppliers of information for users) [9]
    We will observe this standard in order to ensure that we are ethically sourcing and utilizing user information within our product.
S2. IEEE 1028-2008 (IEEE Standard for Software Reviews and Audits) [7]
    We will observe this standard in order to ensure quality and reduce the risk of introducing software defects.
S3. IEEE 12207-2017 (Systems and software engineering -- Software life cycle processes) [10]
    We will observe this standard to ensure that our evolving software system continues to meet our functional and non-functional requirements over time.
S4. IEEE 1012-2016 (IEEE Standard for System, Software, and Hardware Verification and Validation) [8]
    We will observe this standard in order to demonstrate, through means both formal and informal, the validity and security of our final product.

2.4 Previous Work And Literature

TRex

TRex is Cisco's traffic generator used for benchmarking and stress testing several different parts of a network stack including: DPI, NAT, Firewall, IPS, load balancers, and network caches [3]. TRex is built on Linux Foundations' Data Plane Development Kit or DPDK. DPDK is an important library because it allows TRex to utilize libraries to accelerate packet manipulation. Without the support of this acceleration technology, TRex would not be able to scale their request load as high as 200-400GB per second. This quantity of traffic is only necessary to test enterprise applications and is far from the scope of our project. TRex is able to test both stateful and stateless protocols. TRex is also able to modify fields of the source packets but only for stateless protocols.

Our project will be quite similar to TRex however we will only focus on stateless protocols. It also differs from TRex because it does not have the same level of functional requirements. We only need to generate enough traffic to sufficiently test the CDC environment machines and obscure the traffic of the red team. This means that our traffic will likely be under 1 GB a second. For that reason, we will not need to build our technology on top of TRex or DPDK. Instead we will be designing our own producer-consumer architecture to generate test traffic.

2.5 Proposed Design

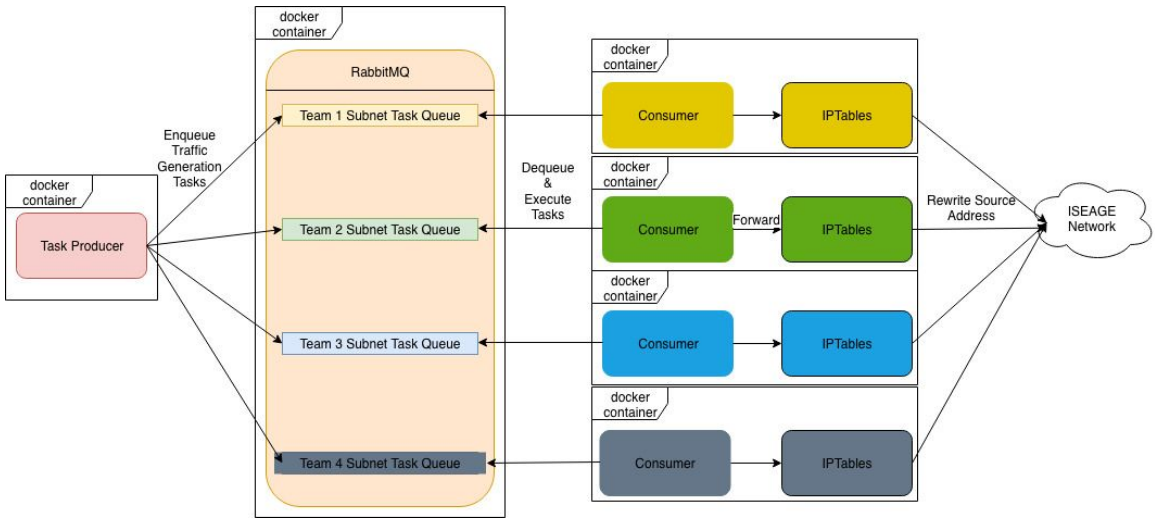

Figure 1

The tool will operate by generating a random attack in the task producer which conforms to the rules set out in our configuration file. This task will be submitted to a consumer, which takes the attack and executes it using a variety of tools against the target. The consumer will also be in charge of masking the source address through iptables which will make it hard for the targets to discern that the traffic is coming from a single tool. This will repeat until the tool is ended and will give each of the targets a constant load of traffic that will mimic that of a real web server.

The project will be set up as a producer-consumer task queue, where a task is an arbitrary attack to be carried out over a specific source address. The correspondence of attacks to addresses will be determined by a configuration file which will be used by a factory to build out task queues and subsequently the tasks themselves. A separate Docker container will be created to consume tasks for a given address. The consumer will also be responsible for rewriting the source address on the fly so that the target of the attacks will not constantly be getting the same source address from the attacks. Any communication back from the target address will be relayed to the producer via ISEAGE.

The traffic generator will be configured by a configuration file which will communicate to the program which targets should receive traffic identified by their address, what kind of traffic the target will receive (i.e. different types of attacks), and additional configurable flags for additional functionality. This file will be used by the system to create a separate task queue for each target. This task queue will be populated by an attack producer which will place the action in the appropriate queue for the target. From there, the consumer for each target will execute actions from the task queue generating the traffic. The tool will also be processed by ip table rules before sending which will reassign the source address

so that the attacks are not easily identifiable by the users who simply monitor the source addresses making requests. Return communication will be handled by the ISEAGE system.

We considered several different variations of parts of our design, primarily how many consumers should be created. The main concern is that too many consumers could be created and severely strain system resources. Our client has determined that this shouldn't be an issue in the tool's intended use so we decided to choose a consumer for each target so that traffic can be generated for each target with uniform frequency.

2.6 Technology Considerations

Docker was chosen due to its strength in scalability. Docker containers can be easily spun up and moved around due to the high encapsulation of the software within the container.

We chose to use a queuing system to enable us to horizontally scale our application. By using a queuing system we can easily change the number of producer or consumer nodes without any effect on the software itself. RabbitMQ was chosen because it's a highly scalable and available queueing system that supports multiple queues [13]. It also is supported by our Python library we are using for asynchronous tasks. We wanted the multiple queue support that RabbitMQ provides because it allows us to create a separate queue for each team we are generating traffic against.

One goal we had that isn't very feasible was using Metasploit to run complex attacks against targets [11]. The framework is written in Ruby so we needed to find a Python library to interface with it. The libraries found were not well maintained and relatively bulky; each consumer would have to install the full framework. The company developing metasploit is working on making their own Python wrapper, but until then we determined that getting this tool to work would not provide enough benefit to the project.

2.7 Safety Considerations

First and foremost we need to make sure that the attacks against the target addresses are safe and will not cause real damage to the servers or user laptops. Additionally, we need to make sure that there is a good way to limit the target addresses that can be put into the tool so that it's not used to attack real servers. This detail will be handled by the ISEAGE environment but will likely require collaboration from us to ensure it's contained.

2.8 Task Approach

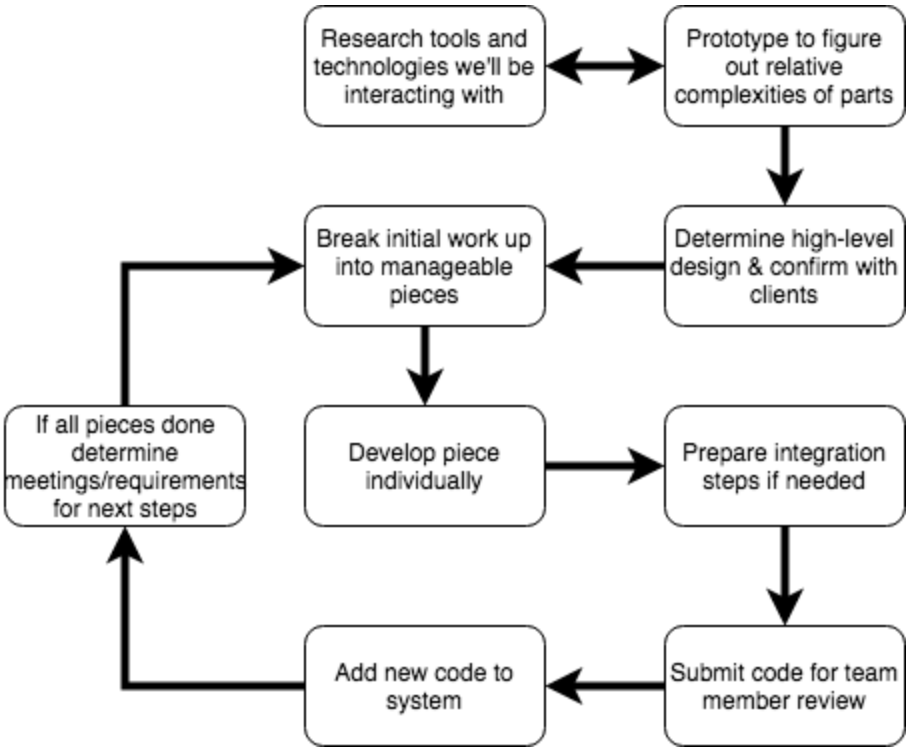

Figure 2

When building this software, we will follow a design approach very similar to agile design. We start by doing basic research on the tools and technologies we will be using, and move on to design a prototype to prove feasibility of our choices. Then we create a high level design and confirm it with our clients.

After this initial work, we enter a loop of continuously dividing work, making progress individually, and combining our individual efforts back together and testing, only to repeat again ad infinium (until the project is complete).

2.9 Possible Risks And Risk Management

We are essentially making a scalable botnet. Most ISP's do "egress filtering" which means that traffic that has a source address that different from the ISP's net is automatically blocked, but if there is a situation in which people are able to bypass this restriction, then they would be able to DDOS a service very effectively, as it would be hard to ban the traffic since you can't just ban a range of IP addresses.

2.10 Project Proposed Milestones and Evaluation Criteria

Limited Support for Simple Protocols

Our first milestone is a very simple prototype, that will test c our configuration file format, and be able to support http traffic through wget, and SSH traffic.

This version of the software will be able to specify a list of "targets" and the protocol / flags to use on that target.

Note that source address rewriting is not considered in this milestone.

Source Address spoofing

The next milestone is support for source address spoofing. This is a large hurdle to get over as it's a very esoteric thing to do, with little existing support materials found online. For example, a lot of tools allow for rewriting http traffic, but we want to rewrite tcp traffic as a whole.

Wide support for various protocols.

After a limited set of protocols is implemented, the next step is just to expand the types of traffic the generator can produce as much as possible before the end of the semester.

2.11 Project Tracking Procedures

Our team has created a Trello board for our project, and are following an agile process. Every task that needs to be completed for the project will have a ticket on trello, and every ticket will be assigned to a team member. Once a ticket is completed, it will be moved to the done section of the board. During our weekly team meetings, we will triage the remaining tickets, and start over again.

Larger design considerations will be taken into account by assigning tags to the tickets based on which part of the system is being worked on. This will allow us to easily prioritize tickets based on what we decide is the most crucial part of the system to be working on in a given sprint.

2.12 Expected Results and Validation

Upon project completion, our team will have a tool to test at both CDC competitions and within cybersecurity classrooms. Sufficient documentation will also be created to ensure the possibility of future work and extensions, should the clients desire it.

Validation will consist of basic testing within both a mock ISEAGE environment, as well as in actual competition settings. By monitoring traffic for a series of use cases, contained within configuration files for the final tool, we will confirm the conformance of our tool's

output to its specification. This will then be reviewed by the client to ensure traceability from outcomes to high level requirements and client requests.

2.13 Test Plan

Validity:

1. Develop a sequence of configuration files increasing in complexity from basic, normal traffic (i.e. non-attack) to full, expected load in a competition/classroom environment (i.e. mixed attack/non-attack traffic)
2. Generate traffic for each configuration file within a realistic testing environment (ISERINK)
3. Monitor generated traffic from both a network (high) level and a user (low) level to ensure that results conform to specifications
4. Capture appropriate statistics for tool output and provide to client for review and feedback

Performance:

- We will conduct scalability testing by scaling out our producer and consumer nodes through the use of Docker. The purpose of this will be to test how the queuing system continues to operate under high load and will be verified if the host running the tool doesn't see any significant slowdown caused by lack of resources.

Security:

- We will conduct security scanning on our deployed VMs to ensure no ports are open unnecessarily. We will use a third-party tool for this which we will choose once the project is put together. It is important that our system remains secure especially in CDC environment.

Usability:

- We will conduct endurance testing to ensure our product will remain live throughout the duration of a CDC. This will be done by running the tool over several days pointed at a test server and verify that the tool's attack consistency (one attack every few minutes) doesn't change over the period.

Compatibility:

- We will conduct compatibility testing to ensure our product works in the ISEAGE environment. This is necessary to ensure our product will still work despite the special networking requirements. The ISEAGE network will need to be modified such that our consumer nodes will still receive packet responses despite rewriting

our own sending addresses. This test will pass based on the ISEAGE's teams expectations of how the tool should fit into the overall system.

3 Project Timeline, Estimated Resources, and Challenges

3.1 Project Timeline

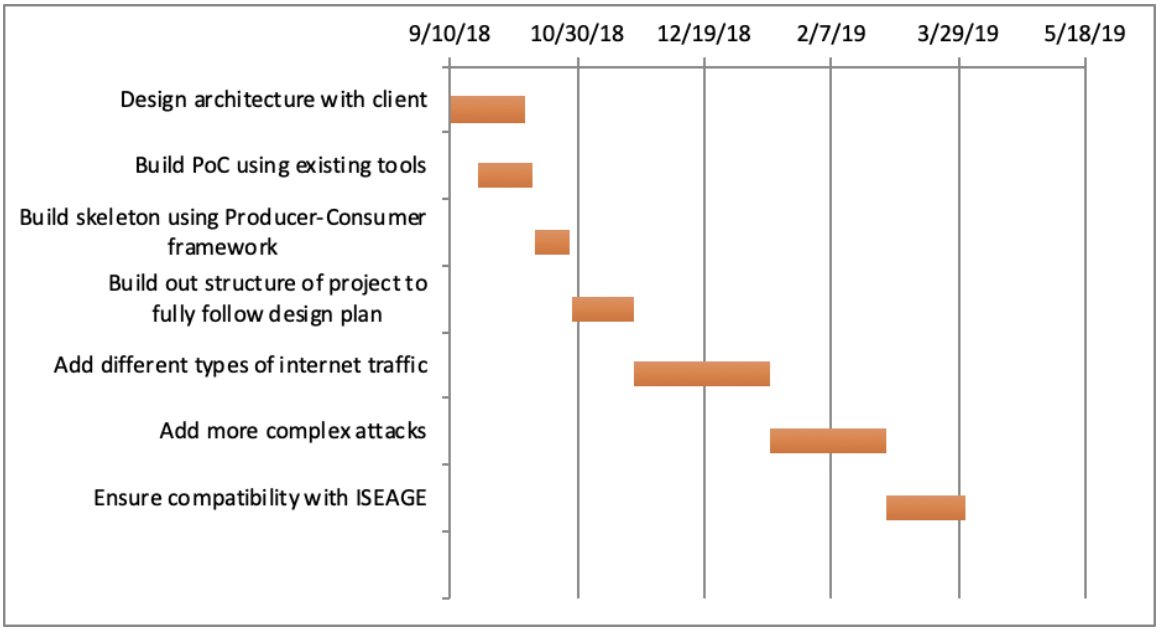

Figure 3

We plan to build out the architecture during this semester and make sure the architecture will be able to accomodate all goals and flex goals that we may add as the project becomes more concrete or additional requirements come up. The second semester will consist of building out the more detailed technical aspects relating to specific types of attacks. These points span large amounts of time on our chart because we will be breaking those up into subtasks which will be partly determined by the ending point of the first semester.

3.2 Feasibility Assessment

Through several meetings with our client and advisor, we have narrowed the scope sufficiently to have a good grasp of the tasks needed to complete this tool in time. Our research this semester has focused on getting each main piece of functionality built out as proof of concept which we have achieved, allowing us to have faith that the technology chosen works. Finally, each member of the team gets along well with one another and all of have professional development experience through multiple internships. We have confidence the team will be able to follow this timeline and accomplish goals in the specified timeframe.

Challenges in finishing the tool will likely come in adding more complex attacks and integrating the tool into the ISEAGE environment. Our initial attempt at implementing complex attacks through metasploit were not successful and further research and consultation with our client will be required. In the integration of the tool, communication and implementation details between our team and ISEAGE will likely prove time consuming. The team has a good understanding of everyone's availability and we hope to use this to coordinate with the other team in a timely and clear manner.

3.3 Personnel Effort Requirements

Task	Description	Developers required	Estimated hours/ developer	Total task time (hours)
Develop configuration file	Determine structure of configuration file so that all the required data can be represented as simply as possible for use by ISEAGE	4	2	8
Establish tools	Create example applications proving that main requirements of the tool can be accomplished with the use of the tools	4	4	16
Build producer	Build out the producer to create the request action that will be executed by the correct consumer for the target. This step includes setting up RabbitMQ	2	10	20
Build consumer	Build out the consumer to execute any request that comes to it from it's specified RabbitMQ queue without source masking	2	10	20
Add HTTP tasks	Add the ability for the producer to build HTTP tasks for consumers to execute	2	5	10
Write scheduling algorithm	The tasks will need to be sent out to the targets at a given rate which will simulate real traffic	3	4	12
Add complex attacks	Find tool or method to execute attacks that require more communication between target and attacker	4	8	32

Integrate tool into ISEAGE	Work with the ISEAGE maintainers and make sure they understand how to build upon it and run it for a competition	6	10	60

Table 1

3.4 Other Resource Requirements

We do not intend to use any other resources.

3.5 Financial Requirements

We do not expect to have any financial costs for this project.

4 Closure Materials

4.1 Conclusion

Before building this product, the CDC as well as cyber security classes at Iowa State lacked realistic traffic to go to competitors or students. This results in easy identification of the attacking team from the team generating normal traffic in the cyber defense competitions. In classes, this results in techniques and tools which are shown to detect or thwart attacks never being demoed or shown working in the real world.

The traffic generator which will be integrated over summer 2019 will make it easy for ISEAGE to set up traffic and attacks going to specified targets through a JSON configuration file. This tool will provide both simple and complex attacks to automatically test the competitors in the CDC and to demonstrate tools and techniques in action for students taking cyber security courses. Additionally, both sets of future users will also get normal traffic to both give noise to the attack requests as well as ensure that their services run under adequately realistic traffic.

4.2 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.