ISEAGE Traffic Generator

Design Document

Team Number 39
Client: Dr. Doug Jacobson
Adviser: Dr. Julie Rursch
Dustin Ryan-Roepsch: QA
Josh Wallin: Requirements
Matt Vanderwerf: Architect
Ethan Williams: PM
sdmay19-39@iastate.edu
sdmay19-39.sd.ece.iastate.edu

Revised: 12/2/18 Version 3

Table of Contents

List of Figures

1 Introduction

1.1 Acknowledgment

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 the entire ISEAGE team, who will be handling the integration of the product after it is completed.

1.2 Problem and Project 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 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 Operational Environment

The product will live in the ISEAGE system as a separate virtual machine which brings up several design considerations. The most difficult of these considerations that we will need to perform source address translation on our outgoing network traffic. By doing this translation, we will be able to make it appear as if the generated traffic is coming from multiple machines. This special routing scheme must also ensure that source address

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

translation does not inhibit two-way communication over multiple packets (i.e., so responses sent by targets can be correctly routed to the product). Additionally, our clients have requested that performance and diagnostic information about the product be communicated to ISEAGE, for the purpose of ensuring correct operation over the course of competitions and lectures.

1.4 Intended Users and Uses

Our project will have two main end user groups: The ISU Cyber Defense Competition (commonly referred to as the CDC), and classes at Iowa State that focus on networking and security.

During a CDC, the Red Team (hackers from industry) must penetrate Blue Team (student participant) systems and capture "flags" (secured information) or perform other malicious acts. In order to better obscure the actions of the Red Team, realistic traffic must exist constantly on the network, as would be the case on the broader Internet. Without this, it becomes trivial for the Blue Team to identify the Red Team, as most of the traffic can be assumed to be the Red Team. Thus, our ISEAGE Traffic Generator is intended to continuously generate realistic internet traffic that would commonly exist on a production network. We will support traffic from a wide array of protocols, including SSH, IMAP, ICMP, HTTP, HTTPS, POP3, IMAP, etc. As these are common traffic types, supporting them will improve the realism of future CDCs.

Additionally, classes at Iowa State need realistic traffic within their lab environments. Not only must benign traffic be generated as listed before, but so must malicious traffic. A good example of this is for use with intrusion detection systems (IDS). Currently, there is no useful way for classes to test IDSs in Iowa State labs, as there is no realistic traffic on the network. By generating both benign and malicious 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 own IDS.

1.5 Assumptions and Limitations

1.5.1 Assumptions

- The packets that we create with fake source addresses, will be rerouted back to us (this is a capability to be added by the ISEAGE developers)
- The scale of the traffic generated will be small enough that a VM in an ISEAGE environment can handle it.

1.5.2 Limitations

- The ability to rewrite the source address of generated traffic and maintain two-way communication, as is needed to ensure traffic appears to originate from multiple sources, will only work in an environment like ISEAGE

1.6 Expected End Product and Deliverables

The tool will be developed over the course of Fall 2018 and Spring 2019; it will be delivered to Dr. Doug Jacobsen for use in ISEAGE during the first week of May 2019. The product will be handed off as a virtual machine encapsulating the Docker image for the product [3]. This image will be fully integrated into the ISEAGE system upon delivery. The source code will also be transferred to enable ISEAGE to extend and change the tool, should their requirements evolve in the future.

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.

The specific deliverables which will be handed over are:

- A virtual machine with the tool set up to be run
- Documentation for users on how to set-up the configuration file
- Documentation for ISEAGE maintainers for further work on the tool

2 Specifications and Analysis

2.1 Requirements

2.1.1 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 Intrusion Detection Systems (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.

2.1.2 Non-Functional Requirements

R13. The design shall scale to support a full cyber defense competition.
R14. All software libraries employed shall be licensed such that their use is permitted in both a classroom and competition setting.
R15. The design and implementation shall follow all relevant and reasonable standards, as encountered during their elaboration (see Standards below).
R16. The product shall be sufficiently secured such that the client can reasonably assume outside parties will not have access to critical system settings
R17. The product will not cause harm or interfere with the operation of computers that are not participating in the cyber defense competition.

2.2 Proposed Design

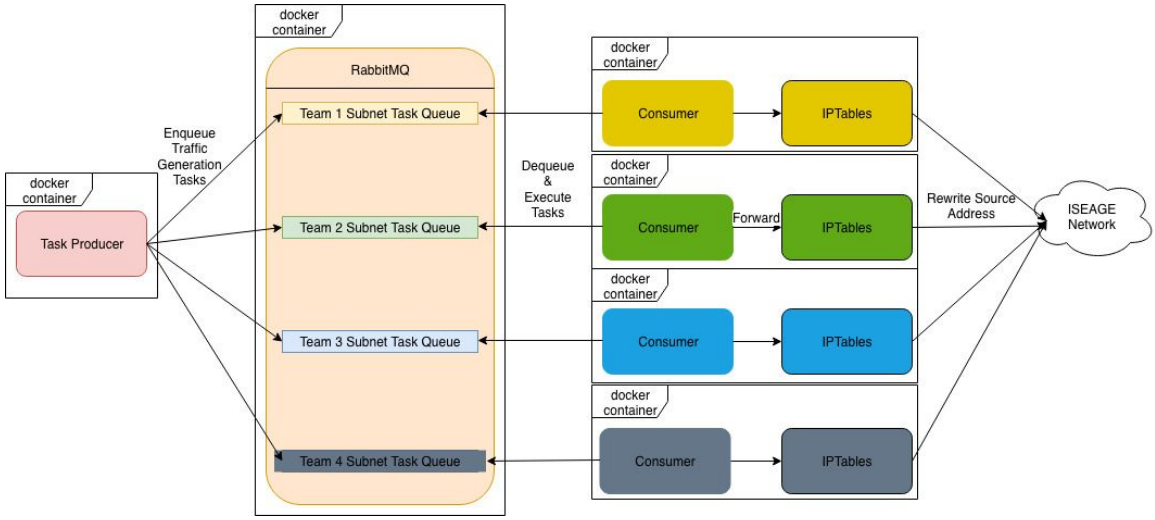

Figure 1- Design Architecture

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 if an attack requires communication between target and attacker.

The producer will formulate attacks based on a configuration file which details which 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 call, 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 Docker container will 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.

2.3 Design Analysis

The architecture of our system has a number of strengths, principally that it will be efficient and perform well under load. The consumers will each be 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 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 does change, alterations can be made such as hosting each container on a separate machine with more processing capability or making each consumer handle more than one target subnet.

3 Testing and Implementation

3.1 Interface Specifications

Our tool will have to be fully integrated into the ISEAGE system. To maintain the modularity of the environment, this will necessitate a simple interface for both conference organizers and classroom instructors. Additionally, this system requires that the targets of generated packets (i.e. "Blue Team") can communicate back-and-forth with the tool.

To allow for easy reconfiguration across use cases, our tool will accept configuration files to reflect the needs of the conference organizer or instructor. This will allow for the selection of a particular group of targets (by IP), as well as some selection of attacks that can be automatically generated using the SNORT database and other resources. This configuration file will be human-readable, and the tool will provide reasonable default values to prevent the need for intense configuration in most cases.

The ability to collect usage statistics is also valuable, especially in a larger environment (such as a CDC). The tool will provide an interface for data collection so that organizers can analyze results and tune the tool as necessary to fit a particular audience.

3.2 Software

RabbitMQ [7]

- This is our task runner which will set up a queue for each of the target subnets specified in the configuration file. The producer will build attack tasks based on the configuration file and submit it to the corresponding queue. The consumer assigned to the given target will take from this queue to execute the tasks.

Python [8]

- Used as the main language of the project. The producer will formulate Python objects which will include all information needed for a producer to execute it. The consumer will also be 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 written in Python.

WGET [5]

- 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 [2]

- The snort packet signature database will be 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 will try a list of commonly used passwords against the target machine and executed using a Python library.

3.3 Functional Testing

Our tool will have one testing suite which will send packets to a test server which will verify that all expected test packets are appropriately formed. This will be accomplished by providing the tool and test server with the same configuration file, allowing the test server to easily verify that all packets received fall within the correct rules. We will also have an IDS, configured to allow us to determine whether our generated traffic is correctly identified as malicious.

3.4 Non-Functional Testing

Testing for performance, security, usability, compatibility

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 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.

3.5 Process

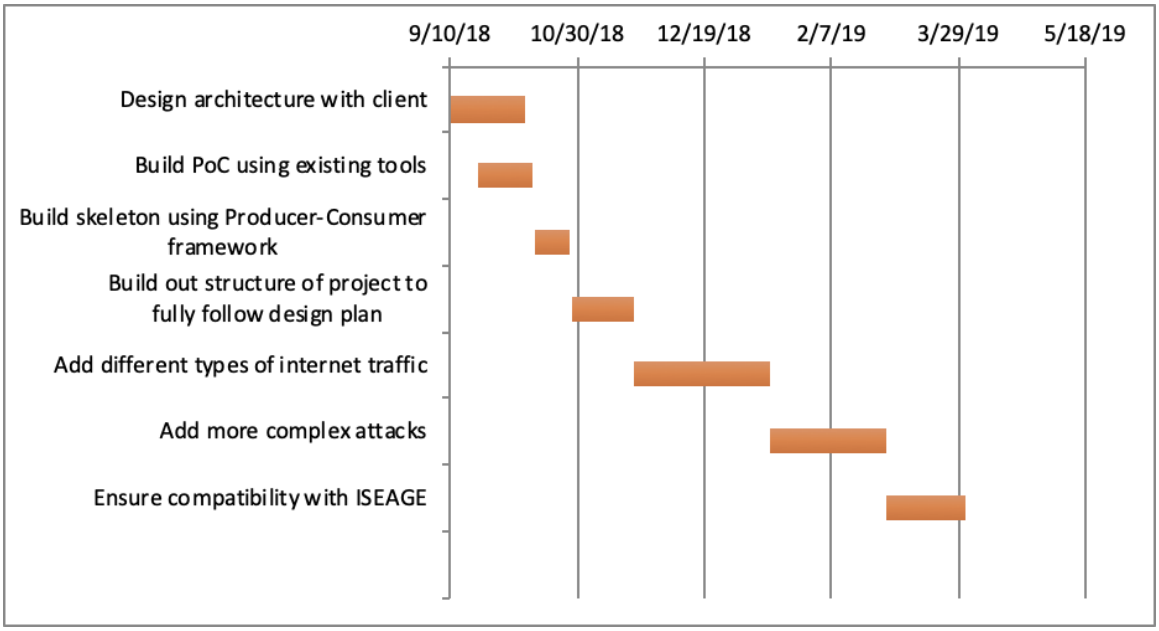

Figure 2- Timeline of design process

Building the tool will come in several stages, the first of which has been completed; the initial build-out of the architecture. Right now it's just one producer and one consumer which conduct sample attacks of each type the tool will implement.

The next phase of the project will be to set up consumers and task queues based on the number of targets and get that set up. At this point, we will be able to start testing how well the tool is keeping up with sending attacks to all the consumers. This will require machines to attack which are being provided by our client and advisor early next semester. The end of this phase will be marked by successful source address rewriting from each consumer to mask the tool's attacks.

The next phase will be determined in part by our progress throughout the previous phase and the results of our working with the ISEAGE team. The tool may get more complex attacks but this will likely require the rerouting of responses back to the tool done by the ISEAGE environment. The bulk of this phase will be getting the tool working within the ISEAGE system and compiling final documentation to hand off at the end of the summer so the tool can be used and built out more.

3.6 Results

An important prototype our team completed was our test of source address rewriting using IPTables. This prototype consisted of three docker containers running on the same machine each with different IPTables rules for source address rewriting. During this test, one container would attempt to establish an SSH connection to another container with IPTables rewriting the source address before that packet left the host machine. Then by checking the logs of the second docker container we are able to validate that the source address was rewritten and the packet reached the proper host machine and docker container. IPTable rules on the second docker container we created to rewrite the destination address in the response packets to the IP address of the third docker container. Once these packets have reached the third docker container, IPTables rules that have been set there rewrite the destination address another time to the correct IP address of the first docker container. This proves that we can successfully implement two-way communication without the target (in this example the second docker container) ever knowing the real IP address of the machine the traffic originated from (the first docker container). Later when we have access to the ISEAGE network, it will no longer be necessary to have IPTables on the target machine (in the previous example the second docker container) which means they will no longer have any idea the packets are spoofing their address. This is possible because the ISEAGE environment will be running custom protocols to reroute the packets back to the consumers through the use of VLAN tagging. This was definitely our largest research issue throughout the semester as most of our research revolved around finding a solution to this before we could proceed further with our project. Now that we know we have a technique that works for this specific functional requirement we may focus on others.

Our second main prototype is our current demo architecture. Our current architecture utilizes docker-compose to spin up a producer container, two consumer containers, and a container running RabbitMQ with two queues (one for each consumer). In this prototype, our producer reads a configuration file and enqueues all of the specified tasks into each of the target queues. The two consumers are configured to consume from their respective queues. They begin dequeueing tasks and executing them as soon as they enter a queue. Once a consumer finishes a task it is marked as completed in the RabbitMQ queue. At this point, the consumer dequeues the next task and the process repeats. The purpose of this prototype was to validate our current architecture to ensure that it can meet the requirements mentioned previously in this document.

Additionally, we have completed some smaller demos to test “one-off” Snort packets and Wget. These are two example traffic tasks that will commonly be enqueued. The “one-off” Snort packets are designed to resemble malicious packets commonly detected by the Snort intrusion detection system. The purpose of testing these packets was to ensure we could also generate malicious traffic. Otherwise, if only non-malicious traffic originates from certain IP addresses, the blue team will know that these addresses can be ignored because they could not be red team machines. The Wget tool will be utilized by the consumers to facilitate HTTP requests to the target networks. During the CDC, teams will be expected to ensure their web servers are constantly running. These tasks will ensure that each target network receives sufficient network traffic to obscure the actions of the red and green teams.

4 Closing Material

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

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

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

[6] *OpenBSD*. OpenSSH Manual, 2017, www.openssh.com/manual.html. Accessed 2 Dec. 2018.

[7] *Pivotal.* RabbitMQ Documentation, 2007 - Present, www.rabbitmq.com/documentation.html. Accessed 2 Dec. 2018.

[8] *Python Software Foundation.* Python 3.7.1 Documentation, 2018, docs.python.org/3/index.html. Accessed 2 Dec. 2018.

[9] *Welte, Harald, and Pablo Neira Ayuso.* Documentation about the netfilter/iptables project, 2014, www.netfilter.org/documentation/index.html. Accessed 2 Dec. 2018.