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: 10/11/18 Version 1

Table of Contents

Figure 1- Project Architecture

1 Introduction

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 and Project Statement

The traffic generator will solve problems that are encountered at the cyber security competitions and classes at Iowa State. In competitions, the traffic going to the competitors is limited and done manually, making it easy for competitors to detect attacks by the red team. Similarly, in cyber security classes, techniques for guarding against attacks are never demoed because there is no mechanism in place to simulate an attack. In addition to attacks on the targets, the tool also needs to generate normal-looking traffic to give a consistent, normal load to the target machine.

The traffic generator will be configured by a JSON 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 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 go through a proxy 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.

1.3 Operational Environment

The product will live in the ISEAGE system as a separate virtual machine which brings up several design considerations. First, the product must be able to communicate reliably with the ISEAGE system which requires collaboration with the maintainers of it. We should make sure that all part of the VM from exposing ports to exposing endpoints is based on their specifications. Additionally, we need to communicate extra information our system keeps track of which ISEAGE also must be aware of. For example, when sending attacks the addresses will be spoofed. In order to have more complex attacks, there must be communication. Responses are handled by the ISEAGE system, but knowledge of our

product's mappins must be made available so that the response is directed to the correct queue.

1.4 Intended Users and Uses

Our project will have two main intended end user groups: The Cyber Defense Competition (commonly referred to as the CDC), 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. 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 intrusion detection systems (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.

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.

1.6 Expected End Product and Deliverables

The tool will be developed over the term of the next 2 academic semesters and will be delivered to Dr. 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. This image will be be able to 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 change 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.

2. Specifications and Analysis

2.1 Proposed Design

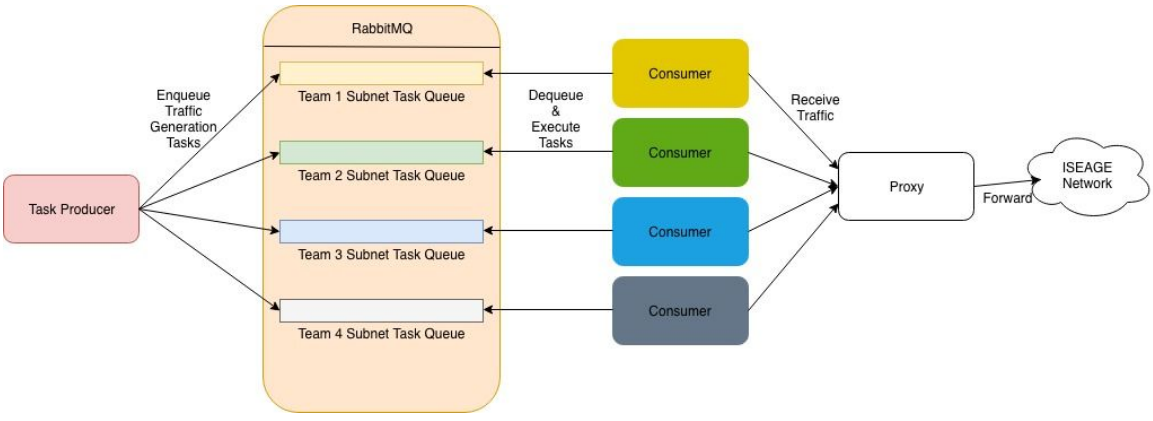

Figure 1- Design Architecture

The task producer reads a configuration file, and sets up RabbitMQ queues for each target subnet. A “task” is an abstract piece of network communication, for example their could be a wget task or a snort task. The task producer will continuously create task objects and add them to the queue for each subnet according to the configuration file. Each queue will have a consumer which dequeues and executes tasks. In order to rewrite source addresses, each consumer sends all network traffic to a virtual sNAT, which will rewrite all of the packets.

At any given moment, each consumer will have all of its traffic’s source address changed to a particular address, but this address is updatable during runtime as part of a task.

So far we have tested the RabitMQ system, creating a few sample tasks.

Lifetime of a task

Consider a task that will send a get request for an index.html file.

The task producer will create a task for the subnet that the index.html lives on, and fill out all of the configuration settings that tasks needs to run a get request. It will also give the task a source address it should used, based on our configuration file.

Eventually, the task is dequeued by a consumer. First, the consumer reads the source address from the task, and sends an update packet to the proxy to reconfigure the proxies consumer mac to source address pairing. It then proceeds to execute the task, while the proxy properly updates the source address. Once the task is completed, a new task will have been created by the producer already, and the process starts over again.

2.2 Design Analysis

So far, the product has a working docker configuration which start a task runner service and sets up a message queue intended to be used with the task runner in the future to send out attacks. Currently, all team members are working on building proof-of-concepts on each of the main subsections of requirements which live as tasks.

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

- This is our task runner, it coordinates the producer consumer model for us

Python

- All of our tasks will be python objects implementing a task interface
- Our configuration will also by a python script

Metasploit

- Metasploit provides a lot of useful traffic generation objects that will be used in tasks

WGET

- used for simple get requests

SNORT

- The snort packet signature database will be used to generate simple packets that make IDS systems go off

3.3 Functional Testing

We will setup a testing ISEAGE environment and test whether our intentionally malicious packets set off simple IDS systems.

We will also install IDS systems on a personal environment with the tool, to make sure the

Student / Classroom use case is affective.

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.

Security:

- We will conduct security scanning on our deployed VMs to ensure no ports are open unnecessarily. It is important that our system remains secure especially in CDC environments.

Usability:

- We will conduct endurance testing to ensure our product will remain live throughout the duration of a CDC.

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.

3.5 Process

3.6 Results

So far we have tested our initial architecture by trying to implement tasks for SSH and WGET.

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 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

4.3 Appendices