Generating Attack Traffic using DETERLab in an Emulation-Simulation Environment

John Mela
Computer Science
Youngstown State University
jnmela@my.ysu.edu

Graduate Mentor: **Wei Yan**
Faculty Mentor: **Dr. Yuan Xue**

July 31, 2011

TRUST Research Experiences for Undergraduates (TRUST-REU)
in Cyber Security and Trustworthy Systems 2011

VANDERBILT
UNIVERSITY

Department of Electrical Engineering and Computer Sciences
College of Engineering
Vanderbilt University

Generating Attack Traffic using DETERLab in an Emulation-Simulation Environment

John Mela

1 Introduction

The inclusion of attack traffic into the Command and Control Wind Tunnel (C2WT) tool suite with network emulation capability was achieved by integration with the back-end of the Security Experimentation Environment (SEER) workbench. "SEER integrates various tools for configuring and executing experiments and provides a user-friendly interface for experimenters to use the tools[2]." We wrap a Java application around an executable used by SEER to create attack traffic in the emulation. The addition of this application allows us to generate attacks at varying intensities. We can produce traffic at types of *flat rate*, *ramp up*, *ramp down* and *ramp pulse*. The existing Java code used to coordinate communication between entities in the simulation and emulation is tied together by Run-Time Infrastructure (RTI) implemented on a Publish/Subscribe system. RTI is an extension of High Level Architecture (HLA) and is responsible for time synchronization between the emulation and simulation environments[4]. Time synchronization is required due to the in-congruent passage of time between these environments. Although the in depth analysis and solution to the time synchronization problem is beyond the scope of this paper, resources exist for further explanation[1]. We extend the C2WT code to communicate attack commands to the emulated network. Extending the existing code allows for the future possibility of simulated attackers. In order to test our implementation, we create a simple application which instructs the emulation to perform four attacks of each type listed above. We monitor traffic using SEER to confirm that the application is doing what it should be.

2. C2WT Integration with DETERLab

The communication between simulated environments over an emulated network is achieved by expanding the functionality of C2WT to communicate through the Cyber-Defense Technology Experimental Research Laboratory (DETERLab) testbed. "The DETERLab cybersecurity testbed[3] is a dedicated network testbed facility customized for cybersecurity research." DETER provides the emulation environment for communication between simulated entities. Each simulated entity is paired with one physical node in DETER. In our experiment, we simulate a Controller and an unmanned aerial vehicle(UAV). The Controller sends commands to the UAV. The UAV in turn, sends images back to the Controller.

Each simulation must send and receive messages to its corresponding DETER node in order to communicate with the federation. The simulations and nodes are disjointed entities which must be bridged together. An Emulation Gateway was introduced to pass messages between each simulation and its respective node. The Emulation Gateway is a DETER node consisting of a Java federate, the RTI, and Matlab simulations. The simulation environment lives entirely within DETER to alleviate long response times of communicating externally. Each federate/simulation subscribes and publishes to an appropriate interaction type with the RTI. The interactions define what kind of information can be sent or received. The Controller federate subscribes to a *Control In* interaction and publishes to a *Control Out* interaction. The UAV federate subscribes to a *Plant In* interaction and publishes to a *Plant Out* interaction. The Emulation Gateway federate subscribes to both *Plant Out* and *Control Out* interactions and publishes to *Plant In* and *Control In* interactions. These interactions allow for plant/controller communication.

Communication over the emulated network is achieved by the use of several Tap Clients and a Tap Server. The Tap Server exists on the Emulation Gateway. Both the Controller and UAV contain a Tap Client which establishes a socket connection to the Tap Server. The Emulation Gateway, UAV, and Controller all contain a Task Buffer which is used to enqueue incoming tasks to be performed. The Controller node contains a *SendCommand* application which sends commands to the *ReceiveCommand* application on the UAV node. Likewise, the UAV node contains a *SendImage* application to send an image to the *ReceiveImage* application on the Controller node. Refer to Fig. 1 for clarification.

3. Attack Generation

3.1 SEER

The SEER workbench contains a packet flooder module which allows the user to manually introduce attack traffic into a running experiment. This can be achieved by configuring attacks through the SEER graphical front-end. This is fine for manually testing specific types of attacks, but is unnecessarily time consuming. We may want to automate attacks at experiment swap-in or develop an attacker simulation. Integrating SEER's attack generation capability with our system is a means to this end. Inspection of the SEER back-end reveals a flooder utility coded in C. Experiments configured to use the SEER workbench are swapped in with a compiled binary of this utility. Further inspection shows that when a packet flooder agent is started on the SEER GUI, this utility is launched on the source node with various command-line parameters specifying the type of attack. All nodes in the experiment simply need implement the SEER workbench to make use of this utility.

3.2 Flooder Utility

The flooder is a unix utility which accepts command line parameters. Based on the parameters, it sends traffic on an interface using libnet. The various parameters are listed below and described.

Parameter:	**Default:**
attack target	none
destination hostmask	255.255.255.255
IP protocol	17(UDP)
TCP flag	SYN
ICMP type	Min: 8 Max: 8
ICMP code	Min: 0 Max: 0
payload length	Min: 0 Max: 0
source port	Min: 0 Max: 65535
destination port	Min: 0 Max: 65535
attack rate	flatrate

-flat rate	
-ramp up	
-ramp down	
-pulse	
-ramp pulse	
high rate	1(pps)
low rate	1
high time	0(ms)
low time	0
rise time	0(ms)
fall time	0
rise shape	1.0(linear)
fall shape	1.0

The *attack target*, along with a *destination hostmask* can be configured to target one specific host, or broadened to allow randomization of traffic disbursement between several hosts. The *IP protocol* may be specified or left to a default value of UDP. Selection of the TCP protocol uses a *TCP flag* of SYN unless otherwise set. If ICMP is employed, a range may be specified for both *ICMP type* and *ICMP code*. The *payload length*, *source port*, and *destination port* may also be assigned in ranges. By default, the payload size is 0. Both source and destination ports will default to cover the entire port range(0 - 65535).

The final specifications involve the *attack rate*. A *low rate* and *high rate* may be set. The default rate is one packet per second. *Low time* and *high time* dictate how long the flooder should perform at its respective packet rates. *Rise time* and *fall time* set the amount of time the flooder sweeps between its peak and trough in milliseconds. The *rise shape* and *fall shape* default to a linear change in rate, but may be specified to a more aggressive or passive change in rate. These parameters are tied together by choosing a type of attack. Without designation, the flooder will produce a *flat rate* of packets at the high rate value. A *ramp up* attack uses the low rate, high rate, rise time, and rise shape values to increase the packet rate from the low to high value. *Ramp down* does the opposite, using the low rate, high rate, fall time, and fall shape. The *pulse* attack switches between the low and high rates using the low time and high time. *Ramp pulse* uses low rate, low time, rise time, rise shape, high rate, high time, fall time, and fall shape to form its attack.

3.3 Attack Node

Attack traffic is generated over the emulated network. We achieve this by developing an attacker which sits on a DETER node. The Attack Node is modeled from the UAV and Controller nodes. It contains a Tap Client and Task Buffer. Since the Attack Node does not communicate with other nodes, it does not contain the image or command applications. It does contain a Flooder Application. The Flooder Application wraps around the SEER flooder utility. It receives tasks originating from a federate on the RTI.

The Attack Node establishes a socket connection with the Emulation Gateway as do both Controller and UAV nodes. It is extended from the the same base code used for both Controller and UAV. This being the case, the Attack Node must receive instruction from the federation. We developed a simple attack federate to issue tasks to the Attack Node. The delivery of the tasks is facilitated by an *attack out* interaction. This new interaction defines two values: target host and command line parameters. The task delivers the attack target(s) and command line parameters needed to execute the attack. The Flooder Application passes the target(s) and command line parameters to the operating system shell which launches the flooder utility.

4 Implementation

The integration of network emulation with C2WT was achieved using Java. We implemented attack traffic in the emulation with the addition of the following classes to the network emulation package:

```
c2w.NetworkEmuTest1.AttackHostWrapper
c2w.NetworkEmuTest1.FlooderApp
c2w.NetworkEmuTest1.test.AttackHostTestBase
c2w.NetworkEmuTest1.test.AttackHostTest
```

The *attack out* interaction was implemented with the following class:

```
c2w.NCS.AttackOut
```

The following classes were modified:

```
c2w.NetworkEmuTest1.EmuGatewayBase
c2w.NetworkEmuTest1.EmuGateway
```

5 Summary

Acknowledgements

First and foremost, thanks to Graciela Perera for all her support and encouragement. Wei Yan's advice and instruction in understanding the nuts and bolts of the code structure was indispensable. Thanks to Yuan Xue for including me in her work and giving me the opportunity to learn at Vanderbilt University.

References

[1] J. G. B. W. W. J. S. G. K. Gyorgy Balogh Himanshu Neema, Graham Hemingway. Rapid synthesis of hla-based heterogeneous simulation: A model-based integration approach. Technical report, Institute for Software Integrated Systems Vanderbilt University, 2008.

[2] C. K. A. H. Stephen Schwab, Brett Wilson. Seer: A security experimentation environment for deter. Technical report, SPARTA, Inc. El Segundo, CA.

[3] T. F. J. M. S. S. K. S. Terry Benzel, Bob Braden and J. Wroclawski. Current developments in deter cybersecurity testbed technology. Technical report, USC Information Sciences Institute, Sparta, Inc., and MIT CSAIL, 2004.

[4] Y. Xue. A model-based integration of network emulation with hla-based heterogeneous simulation environments. Technical report, Vanderbilt University, 2010.

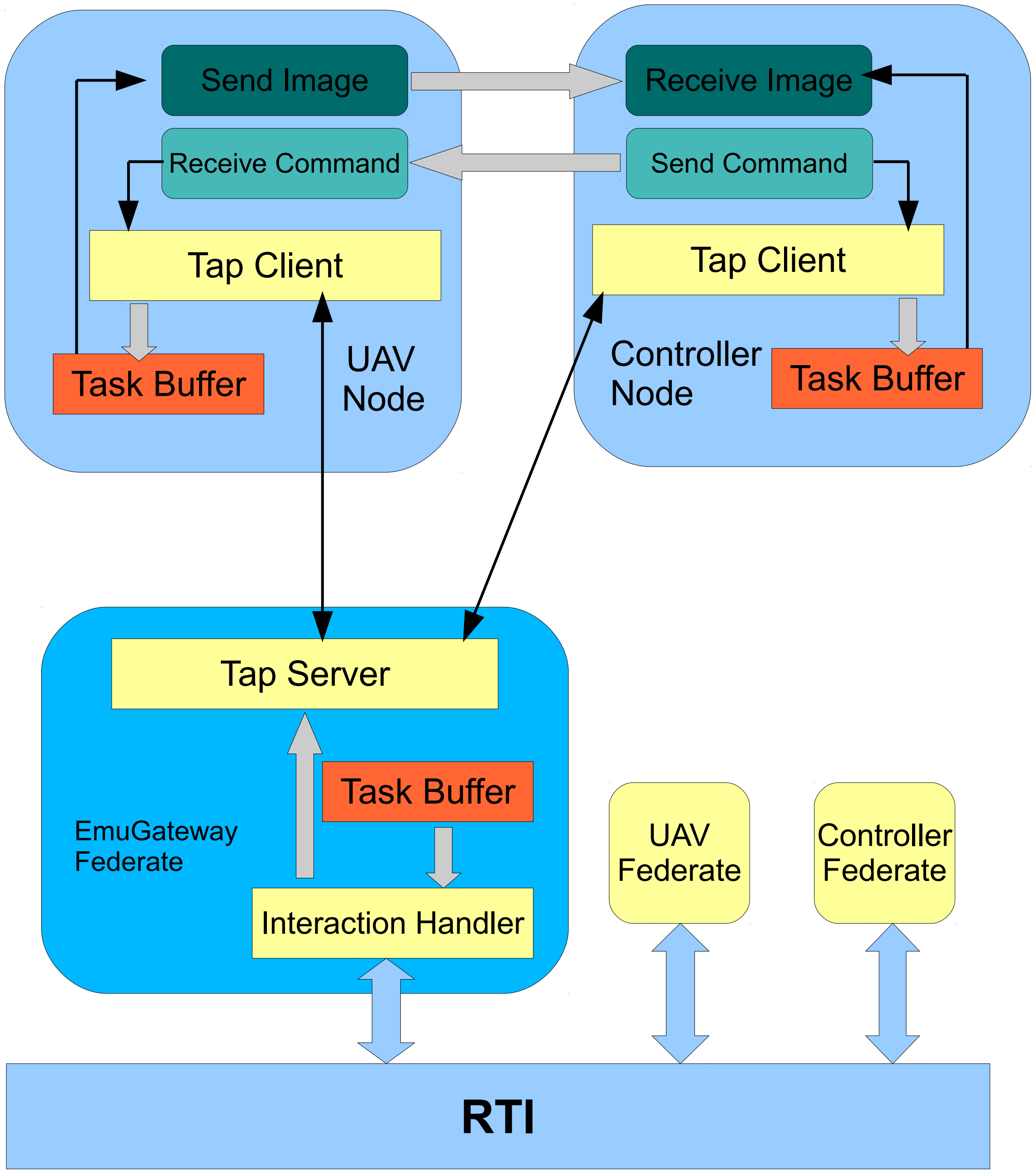

Figure 1. Emulation Gateway with DETER Nodes.