PROJECT ANALYSIS

Revision 1.4

08 May 2014

Team C

Jamie Lane, Bradley Norman, Daniel Ross

Revision History

Date	Revision	Description
04/06/2014	1.0	Initial document
04/08/2014	1.1	Reworded section headers
04/15/2014	1.2	Changed subsystem and context diagrams. Re-named subsystems to match class names.
05/07/2014	1.3	Changed sections B, D, and Table 1 to include the requirements for the MAC address and network interface. Change paragraph 1 in section E to received/sent instead of sent/received.
05/08/2014	1.4	Added sections J, K, L, alternate approaches, strengths, weaknesses.

Project Purpose

RdosTester, will generate the type of traffic but not the volume of traffic necessary to initiate a reflected denial-of-service attack (RDoS). It is not meant to be an actual hacking tool, but a proof of concept. Without a proof-of-concept exploit application available, many developers will not patch security vulnerabilities. The vulnerability that this application exploits was patched in 2010.

Project Analysis

A. RdosTester interfaces with the following systems:

User, IP Network (LAN, WAN), Open Arena Server

B. RdosTester receives input from the following systems:

User → source IP address, destination IP address, gateway MAC address, destination port, network interface, transmit directive

Open Arena Server → status response packet

C. RdosTester sends data to the following systems:

Status request packet → Open Arena Server

Packet size ratio of received/sent packets → User

D. A description of the data processing operations required by RdosTester.

This application receives source IP address, destination IP address, gateway MAC address, destination port, and network interface input from the user via a GUI. Then it constructs UDP packets and IP packet headers; combines IP packet headers and UDP packet payloads; calculates complete packet size; transmits packets to the Open Arena server at the destination address upon user initiation; receives packets from the Open Arena server; calculates size of received packets; and calculates the ratio of the received/sent packet sizes to be output to the GUI. Figure 1 contains a System Context Diagram which depicts data flow in and out of the system under analysis.

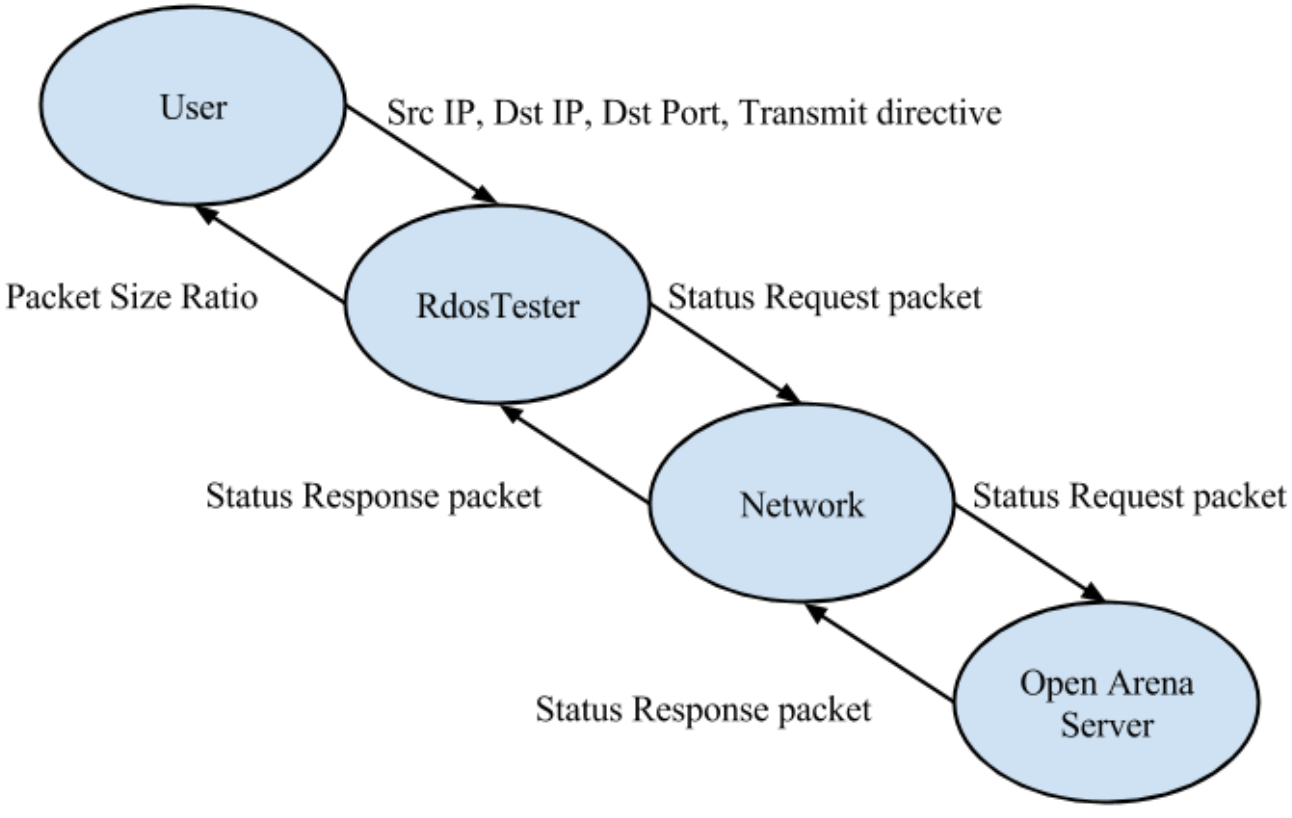

Figure 1. System Context Diagram. Each oval represents a system. Each arrow represents a data pathway.

E. The following subsystems implement the data processing functions required by RdosTester:

RdosTester (GUI) – gathers input from the user, interfaces with the Packet, PacketTransmitter, and Analysis subsystems, displays status and errors for the user

Packet – receives input from RdosTester, builds packet, returns packet to RdosTester

PacketTransmitter – interfaces with Host OS to allow packet transmission and receipt

Analysis – calculates packet size ratio of received/sent packets

Figure 2, Subsystem Diagram, illustrates the data flow from subsystem to subsystem. Additionally, it illustrates the transfer of data into and out of the system under analysis.

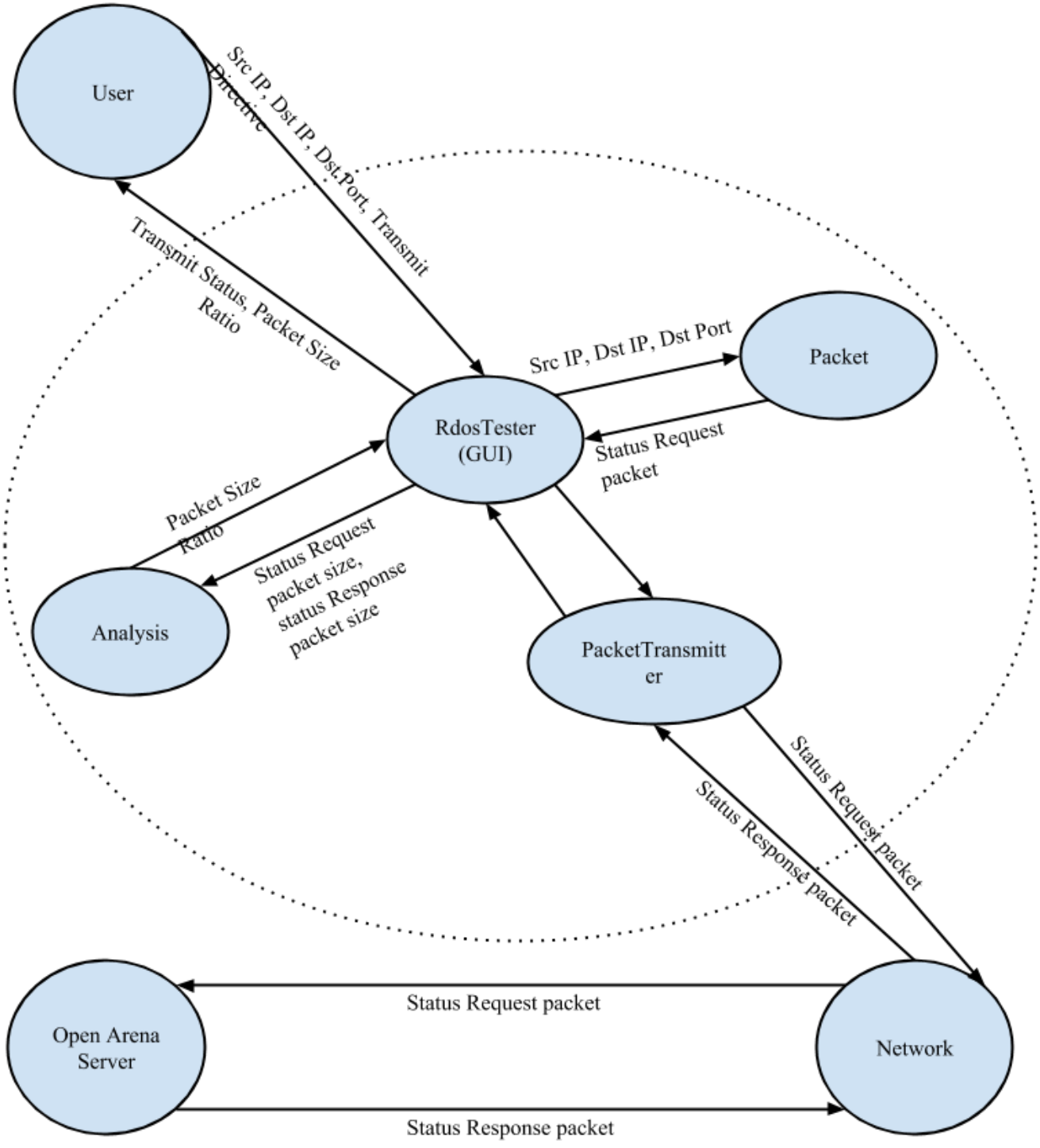

Figure 2. Subsystem Diagram. Each oval represents a system. Each arrow represents a data pathway.

F. The following describes the data interface between the RdosTester subsystems:

User → RdosTester (GUI)
RdosTester → Packet
RdosTester → PacketTransmitter
RdosTester → Analysis
RdosTester → User
Packet → RdosTester
PacketTransmitter → RdosTester
Analysis → RdosTester

G. Table 1, Requirements Matchup, illustrates the correlation between each requirement and its implementing subsystem.

Table 1. Requirements Matchup.

Requirement	Description
1	nonfunctional requirement
2	nonfunctional requirement
3	RdosTester
4	nonfunctional requirement
5	RdosTester
6	RdosTester
7	RdosTester
8	RdosTester
9	RdosTester
10	Packet
11	Packet

12	Packet
13	Packet
14	PacketTransmitter
15	PacketTransmitter
16	Packet
17	Analysis
18	RdosTester
19	nonfunctional requirement
20	nonfunctional requirement
21	nonfunctional requirement

H. Risks and mitigating efforts.

A risk associated with testing or using RdosTester is the possibility of causing a Denial-Of-Service "attack" on the host machine when using the host IP address as the source IP address. This risk will be mitigated by the application sending a single packet per user click. It is highly unlikely that a user would inadvertently generate enough clicks to compromise their own system or network.

An additional testing risk would be the possibility of impacting the performance of an Open Arena server without the administrator's permission. This risk will be mitigated by only testing with a server hosted by a member of the development team.

The application will be designed to provide proof of a vulnerability while minimizing the risk of it becoming an actual malware or hacking tool. One mitigating factor is that the patches for the vulnerability the application exploits became available in 2010. The remaining mitigating techniques will be applied during the application design phase. It will not be designed to be operated remotely. Its design will not include operation by a timer. The interface will not have hidden functionality. The code will not include deliberately opaque or obfuscated functionality.

I. Possible enhancements.

Add the ability to add multiple destination IP addresses to demonstrate a Distributed Denial-of-Service attack.

Add the ability to exploit additional Open Arena server vulnerabilities.

Add the ability for the user to specify a custom UDP payload. With small modifications, the application could be used to demonstrate similar vulnerabilities in other servers.

J. Alternate Approaches.

Java has been chosen as the language for developing this application, as the team members are more experienced with Java than any other programming language. Coding RdosTester in C or C++ would decrease code complexity.

1. Most C and C++ software development kits include libraries that allow raw network access.
2. The few libraries available for Java are wrappers around C libraries.

Microsoft Windows has been chosen as the development and execution platform for RdosTester, as the developers have Windows machines readily available. Using Linux or Unix as the execution platform could decrease complexity of the code.

1. Linux and Unix operating systems have support for raw network access.
2. Windows requires a 3rd party driver to allow raw network access.

Parallelizing RdosTester's packet creation and transmission functions would allow for increased traffic generation. This approach was not taken, as RdosTester's purpose is to send and receive one packet at a time. It is intended to demonstrate an exploit, but not impact another hosts operation through its demonstration.

K. Strengths.

RdosTester's GUI is simple. It requires the least possible user-interaction for its function.

RdosTester's GUI is fault-tolerant. Invalid user input will not crash the application. They generate error messages for display to the user that explain the invalid inputs in plain language.

L. Weaknesses.

RdosTester requires WinPcap and jNetPcap to function. Due to the lack of support for raw networking in Java and Windows, RdosTester has more prerequisite installs than some users might be willing satisfy.

RdosTester is complex. Its functions could be accomplished more simply using procedural code in a language with built-in raw networking support.