PROJECT ANALYSIS

Revision 2

15 April 2014

Team C

Jamie Lane, Bradley Norman, Daniel Ross

Project Purpose

This application, 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, destination port, 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, and destination port 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.

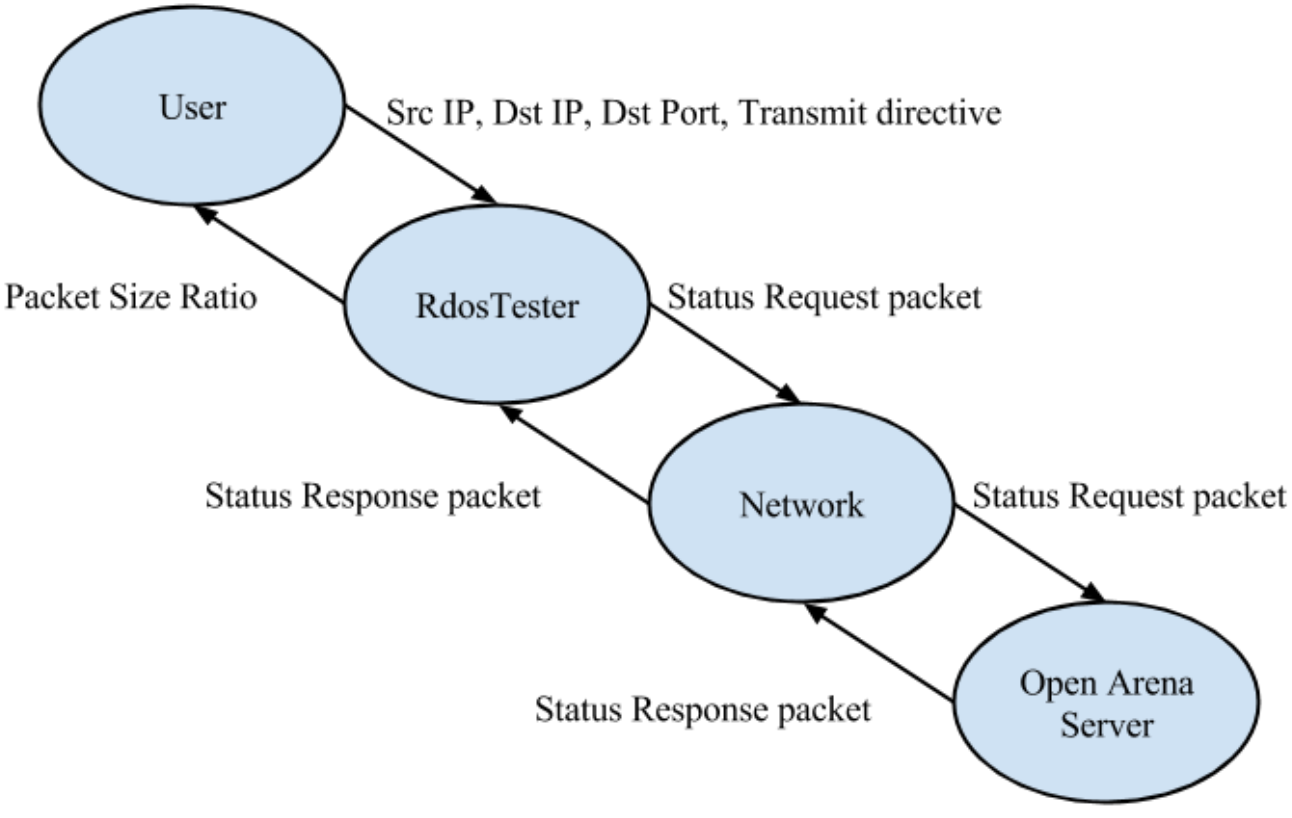

Figure 1. System Context Diagram. Each oval represents a system. Each arrow represents a data pathway.

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 packets, returns packets to RdosTester

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

Analysis – calculates packet size ratio of sent/received 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.

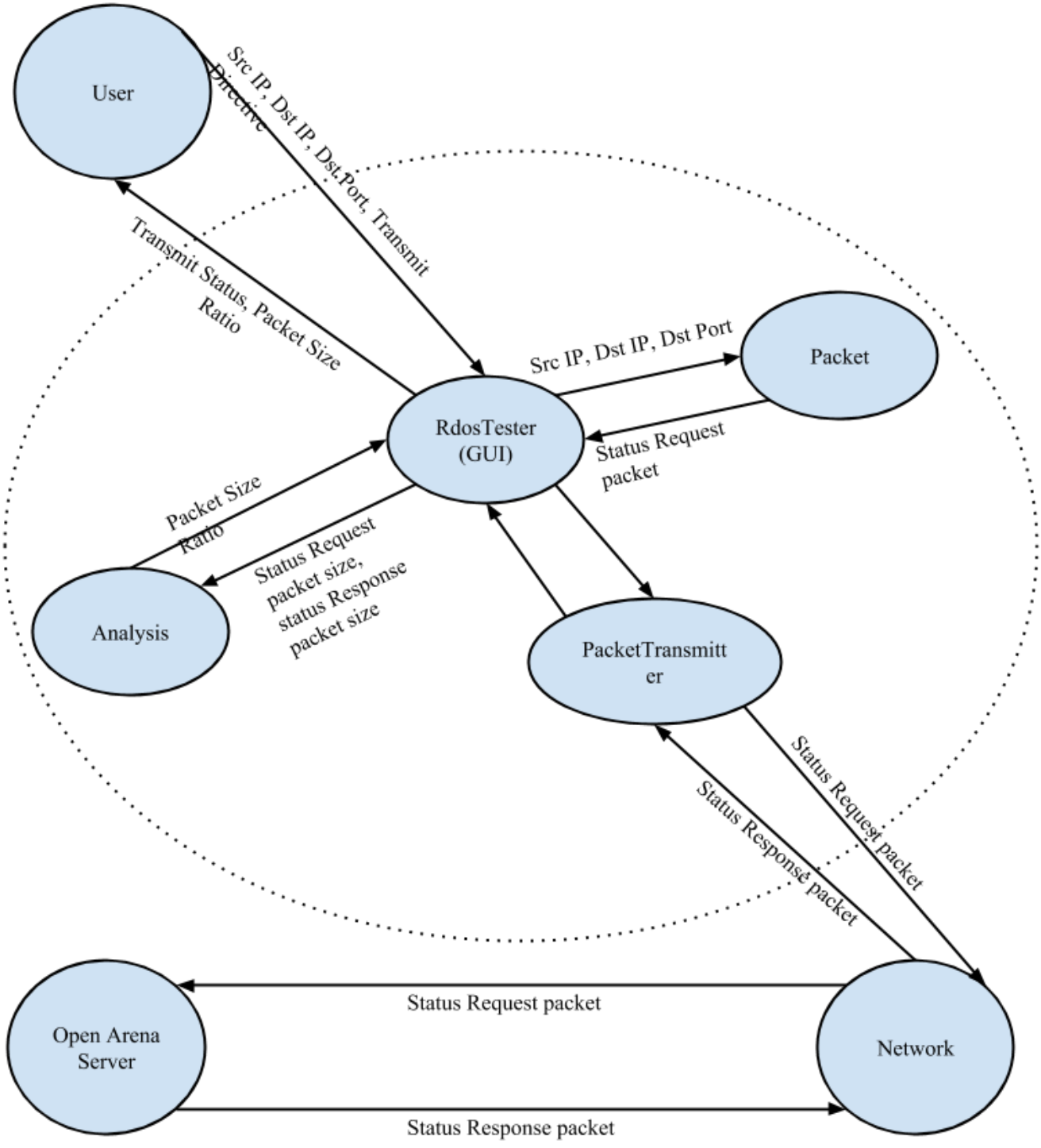

Figure 2. Subsystem Diagram. Each oval represents a system. Each arrow represents a data pathway.

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	Packet
9	Packet
10	Packet
11	Packet
12	PacketTransmitter

13	PacketTransmitter
14	Packet
15	Analysis
16	RdosTester
17	nonfunctional requirement
18	nonfunctional requirement
19	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. Identify possible enhancements (new features) to your design; this is a way to get future work.

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.