Network Working Group S. Futch, Y. Fan, Y. Jou, M. Zhang, R. H. Luo

Playground Request for Comments: 1 JHU Network Security Fall 2017

Category: Standards Track October 2017

Playground Request For Comments

Abstract

The PEEPPacket (Players Enjoy Eavesdropping Protocol) Packet is a network transmission standard that is designed to give Johns Hopkins University Network Security students a standard in which to develop network protocols in the PLAYGROUND (Poly-Localized Algorithmic Yields Generated Remotely

On Unused Network Devices) environment. This PRFC specifies the format standard for PEEPPackets.

Status of this Memo

This document specifies a Playgrounds standards track protocol for

the Playground Experiment, and requests discussion and suggestions

for improvements. Distribution of this memo is unlimited.

Copyright Notice

Copyright (c) 2017 SETH JAMES NIELSON. All rights reserved.

This document is used solely for educational purposes. Any citations

within this document are also for educational purposes and also for

technical reference. If any of these cited elements are protected

under copyright, their inclusion here falls under Fair Use

provisions.

Futch,et al. Standards Track [Page 1]

PRFC 1 PRFC's October 2017

Table of Contents

1. Introduction . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.1. Purpose and Technical Background . . . . . . . . . . . . 2

2. Packet Description. . . . . . . . . . . . . . . . . . . . . . . 3

3. Session Establishment. . . . . . . . . . . . . . . . . . . . . 4

4. Session Termination . . . . . . . . . . . . . . . . . . . . . 6

5. Data Transmission . . . . . . . . . . . . . . . . . . . . . . 6

6. Error Handling . . . . . . . . . . . . . . . . . . . . . . . 8

7. Computational Algorithms . . . . . . . . . . . . . . . . . . . 9

8. State Machines. . . . . . . . . . . . . . . . . . . . . . .. . 10

Intellectual Property and Copyright Statements . . . . . . . . . . . 13

References

Futch, et al Standards Track [Page 2]

PRFC 1 PRFC's October 2017

1. Introduction

The PEEPPacket is intended for use on the PLAYGROUND network as a reliable protocol between hosts and servers in packet-switched computer networks, utilizing the basis of Transmission Control Protocols (TCP)as the underlying methodology for network communication.

This document describes the main functions and specifications performed by the PEEPPacket, as well as its interface to users and connecting applications.

1.1. Purpose and Technical Background

The purpose of the PEEPPacket is to encapsulate data within a TCP-like packet for transmission over a specific computer network, PLAYGROUND. It is intended to simulate some (not all) of TCP transmission, specifically retransmission, reliability, flow control, basic data transfer, and precedence and security. It does not have exactly the same features as a full TCP packet, and is best considered akin to “TCP-Lite” transmission protocols.

Per FRC 1, “The PLAYGROUND experiment, developed by Seth James Nielson, is a

pedagogical framework for instructing principles of Internet Security. The primary goal of the PLAYGROUND framework is to provide a network system that allows students and researchers to develop security mechanisms as well as attacks against them. The PLAYGROUND system is designed to provide a complex enough system to enable kinds of attacks and defenses one finds in the real Internet but with a much reduced complexity.” The PEEPPacket is the only approved data transmission protocol for the Network Security Class-Fall 2017.

Futch, et al Standards Track [Page 3]

PRFC 1 PRFC's October 2017

2. Packet Description (information summarized from techrepublic.com)

The PEEPPacket is a connection-oriented protocol that must provide acknowledged, error-corrected, and flow-controlled transmission to applications and users. It is designed to move data in a continuous stream of serialized bytes. Sequence numbers identify packets of bytes within the data stream.

The packet includes a Source and Destination address/port field to identify the endpoints of the connection, a sequence number to identify the specific packet, checksum to ensure packet validity, acknowledgement field for assurance of transmission and retransmission, and data field for the transmission of information. Additionally, it includes the following protocol types (which will be explained further later in this PRFC):

PEEP Protocol Types

-------------------

SYN - TYPE 0 – Synchronizes the sequence numbers for connection

SYN-ACK - TYPE 1 – acknowledges the SYN packet ONLY

ACK - TYPE 2 – acknowledgement packet (for packets other than SYN)

RIP - TYPE 3 – Rest In Peace. Sends signal that data is complete

RIP-ACK - TYPE 4 – acknowledgement of RIP packet, closes connection

DATA - TYPE 5 – Data packet. Has information inside.

FIGURE 1

The maximum data size for a PEEPPacket is currently 1024 bytes. This is also known as the Maximum Data Chunk.

Futch, et al Standards Track [Page 4]

PRFC 1 PRFC's October 2017

3. Session Establishment

The “three-way handshake” procedure is used to establish a connection. This procedure is typically, but not always, initiated by a PEEP client and responded to by a PEEP server. The procedure also works if multiple PEEP clients simultaneously initiate the procedure. When multiple attempts occur, “SYN” packets from different PEEP clients can be differentiated from one another by the randomly generated sequence numbers.

An example of connection initiation follows. The three-way handshake reduces the possibility of false connections. It is the implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure 2 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a PEEP packet from PEEP client to PEEP server, or arrival of a PEEP packet at PEEP server from PEEP client. Left arrows (<--) indicate the reverse. PEEP states represent the state AFTER the departure or arrival of the packet (whose contents are shown in the center of each line). In the figure, PEEP contents are shown in abbreviated form, with type, sequence number, acknowledgement number, and data field. The checksum field is left out in the interest of clarity.

**PEEP Client**   **PEEP Sever**

1. SYN SYN-ACK

2. SYN-ACK --> <TYPE=0><SEQ=123> --> SYN-ACK

3. SYN-ACK <-- <TYPE=1><SEQ=456><ACK=124> <-- SYN

4. Transmission --> <TYPE=2><ACK=457> --> Transmission

5. Transmission --> <TYPE=5><SEQ=124><DATA> --> Transmission

Basic 3-Way Handshake for Connection Synchronization

Figure 2.

In line 1 of figure 2, PEEP Client prepares checksum for header fields.

In line 2, PEEP client begins by sending a SYN packet to PEEP Server indicating that it will use sequence numbers starting with sequence number 123, and then transitions to SYN-ACK state where it waits for SYN-ACK packet from PEEP Server. After receiving the packet, PEEP Server computes checksum for header fields and verifies that it is correct.

In line 3, PEEP Server sends a SYN-ACK packet to acknowledge the SYN packet it received from PEEP Client, meanwhile indicating it will use sequence numbers starting with sequence number 456, and then transitions to SYN state where it awaits ACK packet from PEEP Client. Note that the acknowledgement field indicates PEEP Server is now expecting to hear sequence 124, acknowledging the SYN packet which occupied sequence 123. After receiving the packet, PEEP Client verifies checksum for header fields.

At line 4, PEEP Client responds with ACK packet containing an acknowledgement number for PEEP Server’s synchronization, and then transitions to Transmission state. Upon receiving the packet and on correct checksum calculation, PEEP server transitions to Transmission state.

Finally, in line 5, PEEP Client sends some data with a DATA packet. Note that the sequence number of the packet in line 5 is the same as the acknowledgement number of the packet in line 4 because the ACK packet does not occupy sequence number space. This ensures that the protocol does not enter a state of continually acknowledging prior ACK packets, creating network congestion.

Futch, et al Standards Track [Page 6]

PRFC 1 PRFC's October 2017

4. Session termination

Ideally, a session is terminated when both of the protocols have no more data to send; However, in reality, it is more likely for one side of the hosts to be ready to terminate while the other still remain sending. In PEEP, when one of the hosts finishes sending data, it sends a RIP (type 3) packet to the other machine. When one machine receives a RIP packet, it must respond a RIP-ACK packet (type 4) for acknowledgement, and finish sending all of the remaining packets it has buffered to send followed by a RIP packet. At this point, both sides of the communication finish sending data, and the connection can be closed.

5. Data transmission

Data transmission in PEEP is based on a fundamental notion in the design: a sequence number. In a transmission environment with errors (such as TCP or PLAYGROUND), it is expected that packets will end up spoiled, out of order, duplicated or even missing. To deal with these problems, every packet sent over a PEEP connection needs a number to identify the packet. The first packet within each data transmission is the lowest numbered, and the following packets are numbered consecutively. Since every packet is sequenced, each of them can be recognized then acknowledged when receiving them. This mechanism allows PEEP to ensure delivery of every packet by retransmission: the sending system must wait a specified time for an ACK. if no acknowledgement occurs, it will resend the packets.

To make error handling easier, senders and receivers must transmit data in a specific manner. Once the receiver gets an undamaged packet with an expected sequence number, is required to send an acknowledgement telling the sender all packets up to that acknowledge number have been successfully received. Note that the correctness of a packet is validated by a checksum, which is calculated with a standard checksum algorithm (IE Adler32). The intent of the checksum is to verify the integrity of the packet since the probability of finding a valid checksum in a damaged packet is negligible. If the sender does not get an acknowledgement from the receiver, it resends the packet after a timeout. (Note that there are several ways to implement acknowledgment, thus there are multiple techniques of receiving an acknowledge number. To avoid ambiguity, we define an acknowledgment of sequence number “X” indicates all duplicate detection in the presence of retransmission. The receiver can just ignore any packet that has a smaller sequence number, after comparing to the latest acknowledge number it sent to the sender.

When considering the possibility that a connection is not available, placing a logical constraint on sending and packets instead of sending all packets at once avoids redundant or wasted data transmission. In PEEP, this constraint is called a window, and the window size is fixed to 5 packets. This means the sender has to keep track of the latest acknowledgment “Y” he got from the receiver and stop sending packets when the sequence number of the packet is greater than or equal to “Y+5”. On the other end of the transmission, The receiver is under the same restraint. The receiver should also keep track of packets it received from the sender, and ignore packets whose sequence number exceed “Y+5”.

Futch, et al Standards Track [Page 8]

PRFC 1 PRFC's October 2017

6. Error Handling

As mentioned above, the PEEP packet is distinguished from each other by a unique identifier: a sequence number. From the receiver’s perspective, the packet must in the order according to the sequence number to guarantee the integrity of the data transmission.

During the data transmission, an unexpected network situation may cause the packets to not be delivered in the correct order, or maybe even cause packet loss during the transmission. Therefore, the PEEP protocol needs to provide an error handling mechanism to avoid the occurrence of falsified transmission.

The error handling mechanism must provide the following three functions:

1. The error handling mechanism needs to sort the packets into the correct order when receiving the packets. The error handling will notify the protocol when the packets are out of order and assist in sorting the incoming packets into the correct order.
2. The error handling mechanism also needs to identify packet loss. Once the transport finish sending data, it will need to confirm the acknowledgement from the receiving side. In this case, the transport will have a timer for receiving acknowledgement, if the time reaches the time limit, the error handling will notify the transport that the corresponding data packet is lost, and the transport will send the data packet again.
3. Once the error handling encountered duplicate packets, the error handling will drop the packets at once. The error handling keeps a record of the incoming packets inside the window, so packets not in the window will be dropped immediately; the packets within the window will be checked against the current received packet to see if the packets are duplicated.

The error handling mechanism must work closely with the transport, the transport has a sliding window mechanism for data transmission, and the error handling will notify the transport when to shift the window and when to stop. So the transport will send the packets in the window’s range, but the acknowledgement will be received by error handling on the receiving side, and the error handling will determine whether the data is intact, and then send instructions to the transport.

Futch, et al Standards Track [Page 9]

PRFC 1 PRFC's October 2017

7. Computational algorithms

During the data transmission, the data packets may not arrive at the receiving side in the correct order, so the PEEP protocol must have a mechanism to reorder the data into the correct sequence. To accomplish this, in each one of the PEEP packet there is a unique sequence number to maintain order. The sequence numbers are in ascending order and the increment for each data transmission (packet) will be +1. Therefore, the protocol needs computational algorithms to keep track these sequence numbers to make sure they will be processed in the correct order.

The first thing that needs to be addressed is how to keep track of the sequence number during the transmission. Currently the protocols have sequence numbers put into the acknowledgement(ACK) packet as an ACK number. In this way, the protocol can tell whether the PEEP packet has been delivered to the receiving side. To assist in this process, the protocol will keep a record of the sequence numbers the transport has sent, and on the receiving side, the error handling will compute the ACKs received to ensure the packets are complete and delivered in the right order (for example of SEQ and ACK numbers, see figure #2).

In addition, a timer is needed by the protocol to determine if any packets have been lost in the data transmission. If the data is damaged or lost, it is likely that the data packet will not be delivered to the receiver. So the protocol must utilize a mechanism to determine whether the PEEP packet is lost. After the transport sends the packet to the receiver, the transmission will not be considered complete without the acknowledgement from the receiver. During the interval, the receiver is waiting for the ACKs to arrive; there can be two conditions, either the packet is lost, or the ACK is still on its way. Since the transport cannot wait for the ACK forever, it is necessary to have a timer to help it identify the situation. Once the elapsed time exceeds the specified time limit, the transport will assume the corresponding packet is lost, and resend the packet to the receiver with the same data and same sequence number.

Moreover, once the receiver receives the packets, it will utilize an algorithm to put all the packets in the correct order. At this time, the algorithm will sort the packet according to the sequence number, the protocol can then retrieve the data in the correct order.

Futch, et al Standards Track [Page 10]

PRFC 1 PRFC's October 2017

8. State machine

A state machine is defined by a list of states, with its initial state specified, and the condition for each transition. A state is a description of the status of a system that is waiting to execute a transition. A transition is a set of actions to be executed when a condition is fulfilled or when an event takes place. A state machine is a mathematical model for computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The state machine can change from one state to another in response to some external inputs; the change from one state to another is called a transition.

In this protocol, we use state machine to illustrate the actions between client and server more concretely.

On the client side, there are six different states: State 0 stands for the initial SYN state, state 1 stands for the SYN-ACK state. State 2 stands for the transmission state. State 3 stands for the write closed state. State 4 stands for the read closed state and state 5 stands for the closed state.

State 0 is the very beginning state of the client. When the client is in state 0, the client will prepare checksum for header fields firstly. Then client sends SYN packet to server and transform to state 1.

When the client is in state 1, it means that the client is waiting for the SYN-ACK packet. In this case, if the client does not receive the SYN-ACK packet after a timeout, it will terminate the program and the state of the client will transform to state 0. When the client does receive the SYN-ACK packet, the client will compute the checksum for header fields. If the checksum is correct, the client will sends ACK packet to server and then transform to state 2. If the checksum is not correct, on the other hand, the client will ignore and discard this packet.

When the client is in state 2, it means that the client is prepared well and ready to send data packets with the data protocol as defined.

Futch, et al Standards Track [Page 11]

PRFC 1 PRFC's October 2017

On the server side, there are also six different states: State 0 stands for the SYN-ACK state, state 1 stands for the SYN state. State 2 stands for the transmission state. State 3 stands for the write closed state. State 4 stands for the read closed state and state 5 stands for the closed state.

Similarly, state 0 is the initial state of the server. In this state, the server is waiting for the SYN packet from the client side and when the server receives the SYN packet, the server will computes checksum and check if it is correct or not. If the checksum is correct, the server will transmit the SYN-ACK packet and then transform itself to state 1. However, if the checksum is not correct as expected or the server did not even receive the SYN packet, then the server will drop the packet and wait until timeout.

When the server is in state 1, it means that the server is waiting for the ACK packet from the client side. When the server receives the ACK packet, the server will calculate the checksum. If the checksum is correct, then the server will transform to the state 2. On the contrary, if the checksum is not correct or the server did not even receive the ACK packet from the client side, then the server will terminate handshake and clear buffer and then the state of the server will transform to state 0.

When the server is in state 2, it means that the server is ready to send data packets to the client.

When the client and the server are both in state 2, they are ready to transmit data packets to each other, thus they won’t change their state until one protocol finishes the process of sending data packets. To illustrate this more concretely, we will take an example of A and B, where A is a PEEP Protocol and B is the another PEEP Protocol. A can be either a client or a server, and B will be the other. When A finishes sending DATA packets and will not send DATA packets anymore. A will send a RIP packet to B and then the state of A will transform to state 3, which is the write closed state. When B, on the other side, receive the RIP packet, B will send a RIP-ACK packet back to A and the state of B will transform to the state 4, which is the read closed state. When A receive the RIP-ACK packet from B, it will know that B has received the RIP packet successfully. On the contrary, if RIP packet or RIP-ACK packet is lost during the transmission, A will resend a RIP packet to B until A receives a RIP-ACK packet from B. Now, the state of A and B are 3 and 4, respectively. After B finishes sending DATA packets and will not send data packets anymore, B sends a RIP packet to A. In this case, if this RIP packet is lost, A will not receive this RIP packet and will not send a RIP-ACK packet back to B. So after a specified time, B will resend the RIP packet to A. Similar to what B did when receiving a RIP packet, A sends a RIP-ACK packet back to B and the state of A will transform from the state 3 to the state 5, which is the closed state. When B receive the RIP-ACK packet from A, the state of B will transform to the state 5. However, if this RIP-ACK packet is lost and B does not receive this RIP-ACK packet, B will keep resending the RIP packet until B receive the RIP-ACK packet from A. But since A is in the state 5 and A can not send any packet out anymore, B will never receive the RIP-ACK packet. In this case, B will be closed by the system after the specified time expires.

Futch, et al Standards Track [Page 13]

PRFC 1 PRFC's October 2017

Copyright (C) Seth James Nielson (2017). All Rights Reserved.

This document and translations of it may be copied and furnished to

others, and derivative works that comment on or otherwise explain it

or assist in its implementation may be prepared, copied, published

and distributed, in whole or in part, without restriction of any

kind, provided that the above copyright notice and this paragraph are

included on all such copies and derivative works. However, this

document itself may not be modified in any way, such as by removing

the copyright notice or references to the PLAYGROUND experiment or

other associated organizations, except by explicit permission.

The limited permissions granted above are perpetual and will not be

revoked by SETH JAMES NIELSON or his successors or assigns.

This document and the information contained herein is provided on an

"AS IS" basis and SETH JAMES NIELSON and associated academic

institutions DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING

BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF

MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Futch, et al. Standards Track [Page 14]

RFC 1 PRFC's October 2017

"Intellectual Property":

The PLAYGROUND framework and student submissions for PLAYGROUND are

intended to be free of any intellectual property conflicts. All

submitting parties, especially students, are not allowed to submit

source code, specifications, or other components that are known to be

protected by copyright, patents, or other IP mechanisms. Students

submitting creative content grant shared rights and ownership of

their content to the PLAYGROUND project. Specifically, any

submission can be used in future educational settings including

subsequent PLAYGROUND experiments.

Any interested party is invited to bring to the experiment director's

attention any copyrights, patents or patent applications, or other

proprietary rights which may cover technology that may be currently

in use without permission.

It should be noted that these statements have not been reviewed by

legal counsel. Moreover, the academic institutions at which

PLAYGROUND experiments take place may have additional or even

conflicting copyright and intellectual property rules. Accordingly,

concerned parties may need to contact their own lawyer and/or

resources at their institution to understand the rights and

limitations associated with code donated to a PLAYGROUND experiment."

As with the initial copyright notice, the Copyright assignment to

Seth James Nielson can be augmented the the names of the authors.

References:

RFC 793

<https://tools.ietf.org/html/rfc793>

PRFC 1 - Nielson <https://piazza-resources.s3.amazonaws.com/j71giz2yhz172d/j89dtd926mb3pi/p_rfc_1.txt?AWSAccessKeyId=AKIAIEDNRLJ4AZKBW6HA&Expires=1508049660&Signature=Lh9NvfIHBak%2FWjKMOXPlwqb5xPE%3D>

<http://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/>