2007

A security analysis of PPStream

Linan Chang
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Chang, Linan, "A security analysis of PPStream" (2007). *Master's Theses*. 3439.
DOI: <https://doi.org/10.31979/etd.x59f-rdzw>
https://scholarworks.sjsu.edu/etd_theses/3439

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

A SECURITY ANALYSIS OF PPSTREAM

A Thesis

Presented to

The Faculty of the Department of Computer Engineering
San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Linan Chang

August 2007

UMI Number: 1448879

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1448879

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

©2007

Linan Chang

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF
COMPUTER ENGINEERING

Dr. Donald Hung

Dr. Harry Li

Dr. Xiao Su

APPROVED FOR THE UNIVERSITY

ABSTRACT

A SECURITY ANALYSIS OF PPSTREAM

by Linan Chang

This study seeks to analyze PPStream's effectiveness against security violations. To achieve this end, the PPStream protocol was first reverse engineered. Then, security issues, such as privacy, exploits, and denial of service attacks, were tested against PPStream and the results were evaluated. These tests find that PPStream is unable to protect the privacy of its users and its data. Also, the tests show that a PPStream peer can decrease its upload rate, by operating behind a router or by limiting its own upload rates, without negatively affecting its download rate. Finally, PPStream is found to be fairly robust against denial of service attacks that target the channel discovery, peer registration, and peer discovery protocols.

Acknowledgements

I would like to thank my thesis advisor, Dr. Xiao Su, for her guidance and suggestions throughout the course of this process.

I would also like to thank my thesis committee, Dr. Donald Hung, Dr. Harry Li, and Dr. Su, for their time and helpful suggestions for corrections.

Finally, I would like to thanks friends and colleagues that provided me with ideas and information.

Table of Contents

Abbreviations.....	ix
1 Introduction.....	1
2 Tools and Methods.....	3
2.1 Tools.....	3
2.2 Methods.....	3
3 Protocol Overview.....	5
3.1 PPStream operation.....	5
3.2 Channel discovery protocol.....	7
3.3 Peer registration protocol.....	10
3.4 Peer discovery protocol.....	10
3.5 NAT traversal.....	14
3.6 P2P media chunk distribution protocol.....	15
4 Privacy.....	18
4.1 Availability of peer lists.....	18
4.2 Clear text peer and channel lists.....	19
5 Exploits.....	20
5.1 PPStream operating via NAT.....	20
5.2 Single peer download.....	24
5.3 Limited upload rates.....	26
6 Denial of Service.....	28
6.1 Block channel discovery.....	28
6.2 Block peer discovery.....	28
6.3 Block peer registration.....	31
7 Related Work.....	33
8 Conclusion.....	35
References.....	37

List of Figures

3.1	Overview of the PPStream system.....	6
3.2	Channel discovery queries.....	8
3.3	A channel discovery reply.....	9
3.4	Channel information window.....	9
3.5	A ppsurl string.....	10
3.6	PPStream method for obtaining peer list.....	11
3.7	Peer discovery reply containing peer information.....	13
3.8	Another peer discovery reply with peer information.....	13
5.1	Comparing the data downloaded between normal and client-like behavior.....	22
5.2	Comparing the data uploaded between normal and client-like behavior.....	22
5.3	Comparing the time elapsed between normal and client-like behavior.....	23
5.4	A sample graph of PPStream uploading rates.....	27
6.1	Peer information and connection initiation.....	30

List of Tables

5.1	Averages for time and data in the limited-upload exploit trials.....	27
6.1	Average for time and data in the block-peer-discovery attack trials.....	29
6.2	Averages for time and data in block-peer-registration attack trials.....	31

Abbreviations

ASCII	American Standard Code for Information Interchange
IP	Internet Protocol
IPTV	Internet Protocol Television
NAT	Network Address Translation
P2P	Peer-to-Peer
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol
RTSP	Real Time Streaming Protocol
TCP	Transmission Control Protocol
SYN	Synchronize
UDP	User Datagram Protocol
URL	Uniform Resource Locator

1 Introduction

A mesh-pull peer-to-peer streaming system is one that utilizes the collective bandwidth of its users to help provide streaming multimedia content, for those same users. A mesh-pull system is one that does not use a tree for broadcasting the media (Hei, Liang, Liang, Liu, and Ross, 2006). Instead, it allows its users to freely connect and transfer data with one another. A streaming system is one that focuses on providing streaming media rather than a whole file. Instead of transferring a large media file and playing it upon completion, a streaming system will split the media file into chunks. Next, the system will continuously send chunks to a receiver. When the receiver has obtained a fraction of the chunks, it can watch that portion of the media file. Thus, the user can simultaneously begin playback of the media and download future parts of the file. If the next chunks are downloaded by the time playback reaches that portion of the file, then playback will be smooth and continuous.

PPStream is one mesh-pull peer-to-peer streaming system and is popular in Asia (“PPStream,” 2007). PPStream can broadcast programs such as movies, television shows, and sporting events. In spite of PPStream’s popularity, little is known about its exact inner-workings because it uses a proprietary protocol. Also, there appears to be little understanding regarding any potential security issues. Researchers could benefit from knowing the strengths and weaknesses in the protocol design. Thus, this study set out to find answers to these two matters.

The rest of this paper is organized as follows. First, in section 2, the tools and methods used in this study are discussed. Next, section 3 attempts to describe the PPStream protocol in as much detail as possible. Then, sections 4, 5, and 6 analyze privacy issues, exploits, and denial of service attacks, respectively. Section 7 discusses related work and section 8 concludes the paper.

2 Tools and Methods

2.1 Tools

The following tools were used in this study. All tools are freely available for download.

- Wireshark – A protocol analyzer that captures incoming and outgoing packet traces. This tool was useful when trying to understand the PPStream protocol. Wireshark also provided time and data measurements.
- NetPecker – A personal firewall and bandwidth limiter. In the experiments that required the use of a personal firewall, this was the tool used to filter packets.
- NetLimiter – Another personal firewall and bandwidth limiter. This tool was primarily used to limit bandwidth.
- Comodo Firewall Pro – Another personal firewall.

2.2 Methods

All PPStream experiments were conducted on a home machine. This host connected to the Internet via a NETGEAR RP614v3 router, which connected to the Internet via a cable modem. The router was not configured with any special port settings.

When capturing packets, each capture session was a new PPStream playback session. So, each capture included the playback setup for the playback sessions. The reasoning was that a sequence of these captures would be less biased towards network conditions, compared to taking a sequence of captures during one long playback session.

The latter could be strongly influenced by network variance. The former would at least give each capture an opportunity to setup a new session.

Usually, experiments measured time, data downloaded, and data uploaded. Quantitatively, a larger time value means that PPStream experienced delays due to longer playback setup or more buffering delays. Also, data downloaded was used as a measure of progress. If two sessions downloaded a similar amount of data, it was assumed that both were able to obtain the same amounts of media chunks. Lastly, data uploaded was used as a measure of contribution. In a peer-to-peer network, such as PPStream, individual peer nodes are required to contribute upload bandwidth to the network. So from a network-wide perspective, it is preferable for individual nodes to contribute as much as possible. The more upload bandwidth contributed, the greater the network's capacity to serve data to its users. On the other hand, from a single end user's perspective, it is unfavorable to contribute large amounts of upload bandwidth. If the user's machine uploads too much, other Internet applications could be affected by the limited upload bandwidth available.

3 Protocol Overview

According to “A Measurement Study of a Large-Scale P2P IPTV System” (Hei, Liang, Liang, Liu, and Ross, 2006), “a mesh-pull P2P live streaming software [such as PPStream], running in user computers (peers), typically has two major communication protocols: (i) a peer registration, channel and peer discovery protocols; and (ii) a P2P media chunk distribution protocol (3).” Based on detailed inspections of PPStream packet captures, it can be said that the above is true for PPStream as well. Generally speaking, the peer registration, channel and peer discovery protocols are carried over UDP packets and the P2P media chunk distribution protocol is carried over TCP packets. The following section describes normal PPStream operation. The remaining sections describe the above protocols in more detail.

3.1 PPStream operation

There appear to be at four types of nodes on the PPStream network: channel list disseminators, trackers, peer list disseminators, and media chunk sharers. These types of nodes are shown in Figure 3.1. The channel list disseminators are servers that send channel information, so a peer node can locate the channel’s tracker. The trackers are servers that collect peer node information. The peer list disseminators are servers that distribute peer lists to peer nodes. The media chunk sharers mainly consist of peer nodes, but the origin server of the media chunks can also be contacted for chunk distribution. It is believed that some servers assume multiple roles by acting as multiple types of nodes.

Figure 3.1: Overview of the PPStream system

Each type of node can be contacted using a protocol: peer registration, channel discovery, peer discovery, and chunk distribution. The channel discovery protocol is used to exchange channel information, so a host can locate the channel's tracker. The peer registration protocol is used to exchange registration information, so a peer node can register itself with the channel's tracker. The peer discovery protocol is used to exchange peer information, so a peer node can locate, and later connect with, other peer nodes and share media chunks. The chunk distribution protocol is used to exchange the media chunks among peer nodes.

When the PPStream application is initialized, the peer node first obtains a channel list from a channel list server. These, and the following, steps are depicted in Figure 3.1. The user then browses the list and selects a channel to watch. Once the channel is selected, the peer node registers its information with the corresponding tracker. Next, the peer node queries the peer list servers for lists of peers. Once the peer node has the information for other peers, it begins playback by contacting these peers. During playback, peers share data chunks amongst themselves. When a user decides to end his viewing session, the peer node's connections with other peers are terminated.

3.2 Channel discovery protocol

The channel discovery protocol includes two types of packets: queries, sent by PPStream, and replies, sent by a server. Queries and replies are carried in the data payload of UDP packets. These packets are passed between the following endpoints: localhost:65535 and remoteserver:7201. Usually, only one server is contacted to obtain the channel list. When PPStream is initialized, it sends a series of queries to the channel list server. The server then responds with a series reply packets, one for each channel. After receiving these replies, PPStream has a current list of available channels that the user can choose from. PPStream also occasionally sends queries and receives replies, presumably to refresh its channel information.

Figure 3.2 displays some sample channel discovery queries. Based on observation, the first few query packets are usually 8 bytes each, the remainder 11 bytes each. Over the course of this channel discovery phase, hundreds of queries are sent. It

appears that each query will have one reply. As for replies, Figure 3.3 displays a reply's data payload in ASCII format. It is unclear what all the fields mean. However, the meanings for some fields may be intuitive. For example, the "encode" and "mediatype" fields most likely describe the type of video encoding and the type of media, respectively. Additionally, Figure 3.4 shows the meaning of the "comment" and "ppsurl" fields. This is a screenshot of the "Information" window for a channel. The screenshot indicates that the "comment" field is a notice or message, presumably about the channel. It also indicates that the "ppsurl" is the string used to locate the media for the channel.

One could infer that "ppsurl" stands for PPStream Uniform Resource Locator. Figure 3.5 displays a sample PPStream URL. Based on this figure, the protocol is "ppstream," the domain is "hwh...b5s.pps," and the path is "UWL...IWI." An assumption is that the domain string is an encoded string that contains the channel's tracker information.

(8 bytes)	08 00 00 00 02 00 00 07
(8 bytes)	08 00 00 00 02 00 00 02
(11 bytes)	0b 00 00 00 02 50 02 03 50 02 00
(11 bytes)	0b 00 00 00 02 5d 02 03 5d 02 00
(11 bytes)	0b 00 00 00 02 60 02 03 60 02 00
(11 bytes)	0b 00 00 00 02 64 02 03 64 02 00
(11 bytes)	0b 00 00 00 02 66 02 03 66 02 00

Figure 3.2: Channel discovery queries

```

..a.....6.}..E.a8 .@....p..E.....!....,$....5 ..[bsinfo]..
mf=3..
encode=1..
mediatype=1..
comment=2333HVR2EDHOFV62Z7OCYTSPJZHSADOV4LFMPUV3XD3NPKGDY
XK7XQ7X2DD3LRFTVS6LNOHD2CTLLRGT5HANNPO2YS7SZR5QZHW45KWP
3S4OPNOEE7L5XUWVY3WL3IRHF3CNVSG5Z7QLLMN6VOZMULGM3CY7BS
W
HYDXS4LRO..
rate=GQYDA..
pub=W2X3RUGT5HANM..
puburl=EM..
index=I2DAWPU3TZ6AYRAUV7OPWTT4NUAVZBVF..
server=hide..
ppstream=ppstream://hw2n6enu6ycokoapmt3ge5rab5tw4ze6dahwotmjxeua6zzjr6syqd3
hoqdtuzapm5uo5peeb5turgamaqhwoseybqga6zizlgmyqd3e.pps/I2DAWPU3TZ6AYRA
UV7OPWTT4NUAVZBVF..
genera=27N5FVOT5HANM..
name=27N5FVOX52YK5T64..[genera]..
lvl=122...

```

Figure 3.3: A channel discovery reply

Figure 3.4: Channel information window

```
ppstream://hwhk6et2ne62zrilapmt3ajzjyb5spmytwea  
hwo3tetyma6zzjr6syqd3hjcmaybapm5ejqdamb5s.pps/  
UWL66LGHFVINUMFS264NG7FT5VE5RIWI
```

Figure 3.5: A ppsurl string

3.3 Peer registration protocol

For the peer registration protocol, PPStream sends a single UDP packet with 98 data bytes and receives a reply with 49 data bytes. The 98-byte packet is always the first packet sent in a playback session. It is sent from the PPStream port. The PPStream port is the port the application uses to send and listen for packets. During playback, more of these packets are sent and received. However, it is difficult to surmise a pattern, based on observation. Since the 98-byte packet is always the first packet sent, this would suggest that it is used by a peer node to register with a tracker. This study was unable to determine the exact contents of these packets.

3.4 Peer discovery protocol

When initiating the playback of a channel, PPStream discovers peers by sending queries to a set of peer list servers and receiving reply packets. This cycle repeats three times in this initial phase. So initially, there are four rounds of queries. These four rounds, and the packets exchanged in each round, are described in Figure 3.6. Later on, during playback, the set of servers will be periodically queried. Also, the queries are sent from the PPStream port. The servers always listen for queries on port 8000 and respond

with information about other peers' IP addresses and ports. These values appear in hexadecimal form and the port values appear to be in network byte order.

- Four rounds of peer discovery queries and replies
- In total, around 38 packets sent and 38 received

Figure 3.6: PPStream method for obtaining peer list

There are two types of peer discovery queries. One is sized 43 bytes and the other is sized 98 bytes. The 98-byte queries are not sent until the second round. It is uncertain how the two types differ. These packets are sent from the PPStream port. Figure 3.7 and 3.8 display reply packets to 43-byte queries. The reply in Figure 3.7 is from the first round of queries. Since it contains three addresses of servers that receive the 98-byte

queries, this indicates that the first round is used to obtain information about these servers that are first queried in the second pass.

In the first round of replies, the replies are of varying size. From observation, some replies can be as small as 42 bytes or as large as 315 bytes. In the ensuing rounds, there appear to be two types of replies. One type of reply is 77 bytes. The other type will usually hold a larger constant value. These values can vary between captures. A possible explanation is that some channel list servers will send more data, in each reply, because they are better connected and thus have more peer information. The 77-byte replies are in response to the 98-byte queries and the other replies are in response to the 43-byte queries. From packet analysis, it appears that the 77-byte replies hold no address-port peer information. The other replies do send peer information. It should be noted that regardless of round, the 43-byte replies always return peer information.

The reply in Figure 3.7 also contains two other types of addresses: other peer list servers' and another peer node's. These peer list servers had already been contacted before the arrival of this reply. So, the purpose of their presence in this packet is unclear. They could be used to confirm a set of online and available peer list servers. As for the peer node, the presence of its address indicates confirms that these packets pass peer information. Figure 3.8 also shows a reply that contains information for at least six peers.

```

21 01 43 00 00 fb 7c b7 c3 d1 c2 ca fl 56 d7 0e 35 77 7d aa b8 e7 98 b1 c7 c2 43 bc 38
e1 0a 80 0b 3c 1c a1 75 40 1f00 00 80 3f00 00 ff ff ff ff 3c 00 00 00 00 00 3d ac c4 6d
40 1f00 00 80 3f00 00 ff ff ff ff 3c 00 00 00 00 00 dd ee f9 cb 40 1f00 00 80 3f00 00 ff
ff ff ff 3c 00 00 00 00 00 00 00 00 00 00 00 00 00 80 3f00 00 ff ff ff ff 3c 00 00 00 00 00
3d 81 39 4e 40 1f00 00 80 3f01 00 ff ff ff ff 3c 00 00 00 00 00 3d 98 9d 88 40 1f00 00
80 3f01 00 ff ff ff ff 3c 00 00 00 00 00 3d ac c5 0b 40 1f00 00 80 3f01 00 ff ff ff ff 3c
00 00 00 00 00 d2 26 03 01 40 1f00 00 80 3f01 00 ff ff ff ff 3c 00 00 00 00 00 d2 26 03
03 40 1f00 00 80 3f01 00 ff ff ff ff 3c 00 00 00 00 00 da 10 e1 50 40 1f00 00 80 3f01
00 ff ff ff ff 3c 00 00 00 00 00 db 99 27 86 40 1f00 00 80 3f01 00 ff ff ff ff 3c 00 00 00
00 00 de c3 c0 b9 fa 00 00 ba a3 cd e2 00 00 00 00 00 00 00

```

Identity of packet sender: 43bc38e1:800a

Peer list servers that receive 98-byte queries:

3c1ca175:1f40

3dacc46d:1f40

ddeef9cb:1f40

Peer list servers that receive 43-byte queries:

3d81394e:1f40

3d989d88:1f40

3dacc50b:1f40

d2260301:1f40

d2260303:1f40

db992786:1f40

Figure 3.7: Peer discovery reply containing peer information

```

bc 00 43 00 00 fd 7c b7 c3 d1 c2 ca fl 56 d7 0e 35 77 7d aa b8 e7 98 b1 c7 c2 39 8f 2e
46 43 bc 38 e1 0a 80 50 19 18 54 b0 9c ed fd 3d be a6 b0 fa 41 dc b0 a8 9b 52 33 dc be
8c 7a bd 20 dd 00 69 57 c0 33 3d ab 30 85 a0 21 8d fc e3 20 c6 1f79 38 41 27 05 23 da
11 a8 46 f8 28 c3 bd 20 87 b4 35 3c 1f72 2e 12 48 7d 6c f4 2d ba 1f da 3b d6 a5 81 25
da 58 b0 e1 61 2e de 59 91 5b 12 25 3c a8 83 8e c5 2d 3c b0 c6 c7 06 44 da 58 a0 26 9e
44 de 55 7e 88 79 cd 3a 28 54 d3 bf25 dc af 78 12 ef 38 7d 21 27 da 78 29 de b9 a6 ab
eb 28 7d 2d 91 c6 af22 dd 07 4a 2d 05 34 02 00 00 00

```

Peers:

3dab3085:21a0

43bc38e1:800a

dcb0a89b:3352

dd006957:33c0

8dfce320:1fc6

de59915b:2512

Figure 3.8: Another peer discovery reply with peer information

3.5 NAT traversal

When a host is behind a router, the Network Address Translation (NAT) will translate outgoing packets' private IP addresses to public IP addresses and the reverse for incoming packets. With NAT, every host behind a router shares a single IP address. Thus, the usage of NAT helps prevent the Internet from running out of IP addresses.

NAT usually only forwards incoming packets that have arrived on ports that have previously sent an outgoing packet. So, a host behind a router can communicate with a remote host with a public address (server), as long as the host initiates the contact. But if two hosts, each behind a router, try to communicate, the NATs will make it impossible. One host would initiate the conversation by sending a packet to a certain port. But since the receiving host has had no prior activity on that port, its NAT filters the incoming packet. Thus, the hosts will never be able to reach each other. This is an issue in peer-to-peer networking, because hosts, many behind routers, need to establish connections with other hosts that are also behind routers.

This NAT issue is solved by a NAT traversal method called hole punching, described in "Peer-to-Peer Communication Across Network Address Translators" (Ford, Srisuresh, and Kegel, 2005). Hole punching is a technique that utilizes a third-party server to help two hosts cross NATs and communicate with each other. This server provides the addresses for each host. Each host, behind a router, first contacts the server to obtain host addresses and to punch a hole in its own NAT. This punches a hole in their NATs because it is an active connection emanating from the host. The NAT would then forward any incoming packets destined for the port which was used to contact the server.

Each host uses their hole to initiate other connections or to listen. So, one host would initiate a connection with the other host by sending a packet through the hole. The receiving host listens for packets coming through its hole. Thus, there is connectivity between the two hosts, across the NATs.

In UDP hole punching, a UDP packet is sent to the server to create the hole. Then, other UDP packets can be sent or received via the hole. TCP hole punching uses a similar technique. The difference is that, at least, 3 different TCP sockets need to be created, one connected to the server, one for listening, and one for connecting to the other host. This is because TCP sockets are usually associated with a specific port. Since TCP does not allow for multiple sockets to reuse an address-port combination, these sockets must have the `SO_REUSEADDR` option set.

Based on observation, it is believed that PPStream uses the UDP peer registration and discovery packets to punch the hole and TCP to connect with other peer nodes. This is because these packets are always the first ones sent in a playback session. In addition, these packets are sent from the PPStream port. This indicates that at least one of these packets is used to punch a hole through the NAT for the PPStream port.

3.6 P2P media chunk distribution protocol

PPStream is believed to use RTSP/RTP, or similar protocols, carried on TCP packets, for the distribution of media chunks. Little is known about the exact information being carried in each packet. What is known is that peer nodes are linked by TCP connections and data is transferred using these links. Peer nodes first establish a TCP

connection between them. After, around eight packets are exchanged between the two peers. Given the timing, these packets probably authenticate the peers or negotiate the basic parameters for the stream. Such functionality is akin to the RTSP requests, SETUP and PLAY. Thus, this points to the possibility that PPStream uses RTSP or a similar protocol. Following these packets, data transfer begins.

The data transfer is likely to be carried out by using RTP. By using Wireshark and judging from the packets' data payload size, there are at least two types of packets. The first type has a set payload size, usually over 1000 bytes each. These packets are most likely carrying the media chunks. The second type of packet typically has a much smaller payload, usually less than around 100 bytes. These are probably RTCP, or RTCP-like, packets, carrying some control information for the TCP connection. From observation, these TCP connections are bidirectional. Both hosts can send or receive media chunks.

When setting up TCP connections, it was observed that PPStream initiates some connections, but mostly accepts incoming TCP connection requests. PPStream uses the PPStream port to initiate and listen for connections. After the peer node obtains a peer list, it begins setting up connections with those peers. Typically, a host spends the first ten seconds sending TCP SYN packets, trying to initiate TCP connections with its peers. This could be seen as a method of actively joining a channel. After the first ten seconds or so, the node becomes more passive and primarily listens for incoming connection requests. Thus, a PPStream node can initiate connections like a client and listen for connection requests like a server.

Usually, TCP is the transport layer protocol used to carry media chunks.

However, a few channels were observed to use UDP to carry media chunks. In these scenarios, it appears that the PPStream protocols are still followed. The only difference is that UDP connections are made with peer nodes, rather than TCP connections. This study was unable to draw any further conclusions regarding these exceptional cases. One theory is that the PPStream system is a conglomerate of different servers (that serve the original copies of media files), each with slightly different implementations. Some of these servers may choose to use UDP for distributing media chunks.

4 Privacy

Generally speaking, people do not like being watched by strangers. The same holds true in the Internet realm. To achieve privacy, IP addresses and user information must be difficult to obtain. This section discusses two PPStream privacy issues: availability of peer lists and clear text peer and channel lists.

4.1 Availability of peer lists

This study found that peer information can be obtained by anyone, without being in the network. First, Wireshark was used to catch a peer discovery packet exchange between the peer node and a peer list server. Then a simple program was used to send the same query, from a separate host outside the network, to the peer list server. This program simply created a UDP socket, sent a packet, and received a packet. The sent packet's data payload was identical to the data payload of the captured query. After sending the packet, a reply identical to the one in the capture was received from the server. Since these replies contain peer addresses, this indicates that anyone can query these servers for peer information. Thus, a curious third-party could theoretically query the servers and obtain a list of all the hosts watching a channel. Although, if a peer node is behind a router with many users, then it could retain its privacy. This is because the third-party would have the router's public address. The third-party would only know that a host behind the router was watching a certain channel.

4.2. Clear text peer and channel lists

Another privacy issue is that the peer list replies appear to be sent in clear text. This is evidenced by the discovery of peer addresses in the data payloads. In addition, the channel discovery replies also appear to be sent in clear text. Though not proven in this study, one could assume that media chunks are also sent in clear text, since the other protocols appear to forgo any encryption. This is an important issue because everything PPStream sends and receives, down to the last detail, could be read by anyone who captures them. Additionally, sending data in clear text makes it difficult to ensure data confidentiality and integrity, also security issues. Nevertheless, data privacy, and security, is practically achieved due to the fact that PPStream packet formats are unknown. While it may be possible to interpret the purpose of the packets, it is difficult to assess the exact data being sent. Thus, this proprietary nature of PPStream acts as the first, and possibly only, defense against violations of privacy.

5 Exploits

An exploit is a series of actions that allow a user to unfairly obtain benefits, without having to give more. For example, a peer node in a peer-to-peer system is expected to act as a client and server. The node should download from and upload to its peers. If everyone does their share of uploading, then everyone will be able to enjoy the collective bandwidth. But when a host acts solely as a client and does not share, it essentially becomes a client. This would be an exploit because it is unfairly leeching bandwidth without contributing its own.

This section discusses three potential exploits: running PPStream via NAT, downloading from a single peer, and limiting upload rates.

5.1 PPStream operating via NAT

It was observed that when PPStream operated behind a NAT, no incoming TCP connection requests were received. This means other peers could not create connections with this node. If the peer node could not connect with other peers, then it may download less and playback could suffer. Yet, PPStream was still able to function. Upon further inspection, it was found that under such conditions, PPStream will continuously send TCP connection requests to its peers. Also, since the peer node was behind a router, it could only be reached if it wanted to be reached. If it chooses not to contact a peer, then there is no way that that peer can contact the former. This was deemed a potential exploit because a peer node could “hide” from other peers, yet still be able to playback media.

In this paper, the behavior for when the peer node is behind a router will be referred to as “client-like.” It is client-like because it does not listen for or accept TCP connection requests. Though, it should be pointed out that once a connection is established, it is believed that PPStream can still serve data over the connection. Also, the behavior for when a peer node is not behind a NAT will be called “normal.”

So, the goal of this section is to determine if this exploit, running PPStream behind a router, works. To test for any benefits of operating behind a NAT, Wireshark was used to conduct a simple study. Ten captures were taken, each capturing playback startup and five minutes worth of streaming data. Each capture was taken one after the other with a short delay between each. The first five captures record normal behavior. The next five captures record client-like behavior.

Afterwards, data was extracted from the captures and plotted in graphs, shown in Figures 5.1, 5.2, and 5.3. Figure 5.1 displays the total number of downloaded bytes and Figure 5.2 displays the total number of uploaded bytes. In these figures (5.1, 5.2), the first five captures (1-5) are for normal behavior and the next five captures (6-10) are for client-like behavior. By visual inspection, Figure 5.1 shows that when PPStream behaves client-like, it downloads about an equal amount of data as it does with normal behavior. As for uploading, Figure 5.2 shows that when PPStream behaves client-like, it uploads about half the data than when it behaves normally.

Figure 5.1: Comparing the data downloaded between normal and client-like behavior

Figure 5.2: Comparing the data uploaded between normal and client-like behavior

Figure 5.3: Comparing the time elapsed between normal and client-like behavior

Figure 5.3 shows the total amount of time needed to capture five minutes worth of streaming media. In this figure, each set has three measurements in the 320 to 340 second range and one measurement in the 340 to 350 second range. However, the difference between the third (normal) and first (client-like) points is large, on the order of one minute. Though this difference could be attributed to changing network conditions, it is difficult to draw a conclusive pattern based on visual inspection. Nevertheless, each set does have four approximately equal measurements. To quantify this, under normal behavior and on average, 330.7 seconds were required to stream five minutes of content and 340.4 seconds were required under client-like behavior. Thus, on the average, it appears that operating behind a router will generally require more time to stream the same amount of data. However, this added delay does not appear to be significant. This is because 10 seconds of extra time seems small, compared to the total session time, which is usually over 300 seconds.

The above results suggest that running PPStream behind a NAT is an exploit. Compared to normal behavior, a peer node with client-like behavior will upload significantly less, half in this scenario, data to other peers. At the same time, PPStream does not appear to suffer any punishment by sharing less data. The host still downloads the same level of data and a significant delay is not experienced.

5.2 Single peer download

This potential exploit attempted to download solely from one peer. If a peer node is able to directly connect with another peer and initiate a playback session, then it would be a break from the peer-to-peer paradigm. These nodes would have a client-server relationship. Theoretically, such an exploit would be more unfair than the first exploit. In the first exploit, the peer node may have acted more like a client, but still served data to its peers. With this exploit, the peer node would be a pure client. It would not serve any data because the data it can upload all originated from its only peer.

Since the PPStream protocol and implementation are not clearly understood, it was infeasible to create a modified PPStream client. Instead, the goal was to create conditions so that the peer node would seek to download from one host. The conditions were created by using a personal firewall to restrict TCP connections. The peer node was only allowed to contact one peer node, using TCP. All UDP connections were still permitted. To obtain this peer node, Wireshark was used to capture a playback session. Then, the ten peer nodes that uploaded the most data were selected. It was assumed that these nodes would be more likely to be or act as a server. Next, there were ten trials to

determine if PPStream could download directly from any of these ten nodes. In each trial, PPStream was restricted to one peer node for TCP conversations. While conducting this experiment, it was discovered that, sometimes, none of the ten trials were able to connect to the target peer node. If this was the case, then the process was repeated with another channel. Also, these trials were conducted using PPStream behind a router. This was done mainly for convenience.

It was found to be possible to communicate, using TCP, with only one peer node. For one channel, three different nodes were contacted for a client-server connection. From packet analysis, it was noticed that only in one trial did the server host appear in peer discovery messages. This finding would point to the possibility of auxiliary peer discovery methods. One possible method is via the channel discovery packets. The ppsurl strings are believed to contain the addresses to channel trackers and trackers may also serve as media chunk distributors. So, PPStream could reference these tracker addresses when initiating connections. Another possible method is for PPStream to contain a cache consisting of recently-used or well-known hosts.

Regarding the benefits of this exploit, it does not appear to provide any. Practically, the playback setup required too much time. None of the three trials were able to actually begin playback. The shortest trial waited for two and a half minutes before it was manually abandoned. The longest trial waited for over five minutes. These findings suggest that the serving peer does not make any special accommodations. It does not dedicate bandwidth for the peer node. Therefore, this exploit does not work.

5.3 Limited upload rates

This exploit aimed to limit the upload bandwidth for a peer node. Ideally, a peer node could limit its upload rate, but still enjoy the same download rate. To conduct this experiment, five three minute captures were first taken. Using the total data uploaded and time elapsed for each capture, an average upload rate of 28.2 KBps was calculated. Using a graph similar to Figure 5.4 and the calculated average upload rate as reference, three upload rate cap values were chosen: 50, 25, and 1 KBps. The graph was used to envision the general amount of uploading that could be cut off by setting a limit.

Next, three trials were conducted, one for each upload rate cap. The goal was to see if limiting the upload rate benefited any trial. Each trial consisted of ten three minute captures. In sequence, the odd captures served as control data. These captures were taken without any upload limit. The even captures were taken with limited upload rates. The capping of the upload rate was performed using NetLimiter. The captures were also taken using PPStream behind a router.

Table 5.1 displays the results of these trials. Each chart includes the averages for elapsed time, data uploaded, and data downloaded over the control and limited upload captures. On average, each trial downloaded about 9 MB of data. In the 1 KBps trials, the same amount of data was downloaded and 5.12 MB less was uploaded, but 44.7 more seconds were required. Due to the added delay, the 1 KBps upload limit cannot be considered an exploit. However, the 50 and 25 KBps upload limits could be considered exploits. In these trials, the same amount of data was downloaded in, around, the same amount of time. Additionally, the 50 KBps trials uploaded 0.59 MB less data and the 25

KBps trials uploaded 2.72 MB less data. The 50 KBps upload limit provided a slight benefit, while the 25 KBps limit provided a significant benefit. These results suggest that limiting upload rates can be used as an exploit.

Figure 5.4: A sample graph of PPStream uploading rates

Table 5.1: Averages for time and data in the limited-upload exploit trials

	No Upload Limit	50 KBps Upload Limit
Elapsed Time (seconds)	160.20	159.96
Data Uploaded (MB)	3.84	3.25
Data Downloaded (MB)	9.36	9.84

	No Upload Limit	25 KBps Upload Limit
Elapsed Time (seconds)	152.99	151.26
Data Uploaded (MB)	5.55	2.83
Data Downloaded (MB)	9.18	9.02

	No Upload Limit	1 KBps Upload Limit
Elapsed Time (seconds)	151.10	195.80
Data Uploaded (MB)	5.90	0.78
Data Downloaded (MB)	9.15	9.97

6 Denial of Service

A denial of service attack aims to deny service to the end-user. In the context of this paper, a denial of service on a PPStream end node would deny a user the ability to watch a program. In this section, the three attacks targeted the channel discovery, peer discovery, and peer registration protocols.

6.1 Block channel discovery

This denial of service attack targeted the channel discovery protocol. From protocol analysis, it was noticed that channel information requests always emanated from port 65535. So, a personal firewall was configured to filter out such request packets. When initializing PPStream, it was observed that the peer node was unable to acquire channel information. Thus, by filtering out channel information packets, peer nodes would be denied access to current channel information. However, it should be noted that users are not denied access to the channels. Rather, the channel list will be unavailable, so users have no list to select from. But as long as the user has the ppsurl for the channel, playback can be setup and begin.

6.2 Block peer discovery

This denial of service attack targeted the peer discovery protocol. To do this, the attack blocked packets destined for port 8000, thus shutting down the peer discovery mechanism. A personal firewall was used to filter out the peer discovery packets. Ten

three minute captures were taken. In sequence, the odd captures served as control data. These captures were taken without any packet filtering. The even captures were taken with the packet filtering. If successful, the attack could deny peer information to the peer node. Then, the peer node would have an insufficient number of peers to download from, shutting down playback.

In reality, it was found that PPStream playback was largely unaffected. Table 6.1 shows the averages over each type of trial. According to figure, the averages for the elapsed time, data uploaded, and data downloaded are approximately equal. There exists no significant discrepancy whether peer discovery packets are filtered or not.

Table 6.1: Average for time and data in the block-peer-discovery attack trials

	No Packet Filtering	Peer Discovery Packets Filtered
Elapsed Time (seconds)	211.01	209.18
Data Uploaded (MB)	3.12	3.55
Data Downloaded (MB)	11.86	12.44

Hexidecimal	Decimal
18 53 0d 8f/ 20 11	24.83.13.143:8209

Packet 1

Time	Source Address	Source Port
69.052036	58.59.214.135	13869
Data Payload		
00 00 00 00 01 00 00 00 08 35 00 00 00 09 08 00 00 00 00 00 00 00 00 00 7d 5d d3 17 70 23 3a 3b 94 64 4f 39 3d 1d d6 51 7f 3a db 85 b0 8a c6 28 18 53 0d 8f 11 20 7d 7f 9d 95 ea 28 de a8 cf 42 ca 1f		

Packet 2

Time	Destination Address	Destination Port	Info
72.997759	24.83.13.143	8209	2626 > 8209 [SYN] Seq=0 Len=0 MSS=1460

Figure 6.1: Peer information and connection initiation

The reason for this is because it was found that peer information can be carried in TCP packets. Figure 6.1 shows two packets. The first packet is shown to contain peer information. The next packet is a TCP SYN packet, used to initiate a connection with the new peer node. So, the PPStream host obtained the first packet and became aware of the new peer 69 seconds into the playback session. Then three seconds later, it initiated a TCP connection with this newly-discovered peer. This proves that TCP packets can contain peer information and the information is used to create new connections. This finding could indicate that the peer discovery protocol has a TCP component. Or, it could mean that peer information is transferred with the media chunks. Either way, this attack failed because PPStream has a robust peer discovery mechanism that utilizes both UDP and TCP transport.

6.3 Block peer registration

The third attack involved blocking all UDP packets, mainly targeting the peer registration protocol. It was difficult to predict the specific destination ports used for these packets. So, the plan was to filter out all UDP packets and compare the resulting behavior, relative to the behavior in Section 6.2. After the channel lists were obtained, the same personal firewall was used to filter out all UDP packets. Ten three minute captures were taken. In sequence, the odd captures served as control data. These captures were taken without any packet filtering. The even captures were taken with the packet filtering. Based on this paper's understanding of PPStream, blocking all UDP packets could prevent the peer node from joining the network. If the peer node cannot register, then it would be denied access to the channel.

However, this attack was also found to be ineffective because playback still occurred. Table 6.2 shows that whether packets were filtered or not, approximately the same amount of data was downloaded in the same amount of time. The only difference is that 1.45 MB more data was uploaded when UDP packets were filtered. However, this is not a major issue since the important finding is that the attack failed to deny playback.

Table 6.2: Averages for time and data in block-peer-registration attack trials

	No Packet Filtering	All UDP Packets Filtered
Elapsed Time (seconds)	218.88	214.41
Data Uploaded (MB)	6.47	7.92
Data Downloaded (MB)	12.46	12.18

It is difficult to explain why the attack failed. When UDP packets are filtered out, the only difference, compared to no filtering, is that the UDP packets are missing. PPStream appears to simply skip sending the peer registration and peer discovery packets. Instead, it just begins connecting to other peers and transferring data. A possible explanation for this behavior is that the peer registration packets are not required. If a peer node does not register with a tracker, the peer would have to manually convey this information to each peer it comes in contact with. This overhead could account for the extra 1.45 MB of data uploaded. It should be noted that this explanation is just speculation.

7 Related Work

Many studies on peer-to-peer streaming topics have been carried out. Among them, two were used to guide this study: “A Measurement Study of a Large-Scale P2P IPTV System” (Hei, Liang, Liang, Liu, and Ross, 2006) and “Measurement of Commercial Peer-To-Peer Live Video Streaming” (Ali, Mathur, and Zhang, 2006).

The first paper, “A Measurement Study of a Large-Scale P2P IPTV System” (Hei, Liang, Liang, Liu, and Ross, 2006), describes a measurement study of PPLive, a mesh-pull peer-to-peer streaming system similar to PPStream. Although this paper focused on using measurements to discover network characteristics, it provided an overview of mesh-pull peer-to-peer streaming systems. Since, many of the protocol observations found for PPStream were similar to those in the paper, the overview in the paper confirmed and helped give order to the observations. For example, protocol analysis was able to identify the existence of the peer registration protocol. However, there was no evidence to indicate that the purpose of those packets were for peer registration. This work (Hei, Liang, Liang, Liu, and Ross, 2006) was able to help realize the purpose for the observed behavior.

The second paper, “Measurement of Commercial Peer-To-Peer Live Video Streaming” (Ali, Mathur, and Zhang, 2006), is another measurement study that involved PPLive and SOPCast, both peer-to-peer streaming systems. This study devoted a section to reverse engineering the protocols. This section included observations similar to those seen in PPStream. As is with the previous study ((Hei, Liang, Liang, Liu, and Ross,

2006), this study (Ali, Mathur, and Zhang, 2006) thus helped the PPStream protocol observations take shape into a better-defined description.

The difference from (Hei, Liang, Liang, Liu, and Ross, 2006) and (Ali, Mathur, and Zhang, 2006) is that this study (“A Security Analysis of PPStream”) focuses on PPStream, rather than PPLive or SOPCast. The general protocol framework appears to be similar to those described in the two studies. But this study provides a more detailed view into the specific protocols used by PPStream. This study also deals with privacy, exploit, and denial of service issues specific to PPStream.

8 Conclusion

This study is believed to be one of the first that deals with PPStream. In this study, the PPStream protocol was first reverse engineered and described in some detail. PPStream was found to have channel discovery, peer registration, peer discovery, and media chunk distribution protocols. In addition, the NAT traversal technique was observed to use UDP packets to punch a hole for TCP packets.

In order to evaluate PPStream security, privacy issues, exploits, and denial of service attacks were tested and discussed. One privacy issue is that peer lists can be obtained by a curious third-party, without being on the network. Also, it is believed that data is usually sent in clear text.

As for exploits, three were tested: PPStream used behind a NAT, download from a single host, and limited upload rate. The PPStream used behind a NAT was found to be a usable exploit. The same amount of data could be downloaded and the uploaded data could be significantly reduced. The limited upload exploit was also found to be usable. Relative to other measurements taken, it was shown that this exploit could also cut down on the amount of data uploaded. On the other hand, the download from a single host exploit was found to be ineffective. Playback was unable to actually start.

The denial of service attack on the channel discovery protocol was successful. However, there are still ways for PPStream to locate and connect with channels. The other two attacks, on peer discovery and registration, were unsuccessful. It was found that PPStream was able to obtain peer lists via alternate methods. Also, blocking peer

registration may possibly incur an extra upload overhead for session information, but service is not denied.

Since PPStream uses a proprietary protocol, most of the results are based on a best estimation of the protocol. Further work into the protocol details and its behavior would help broaden the understanding of mesh-pull peer-to-peer streaming systems, their strengths, and vulnerabilities.

References

- Ali, S., Mathur A., and Zhang H. (2006, August). Measurement of Commercial Peer-To-Peer Live Video Streaming. *Workshop in Recent Advances in Peer-to-Peer Streaming*. Retrieved 28 May 2007, from <http://www.cs.cmu.edu/~hzhang/papers/PPStreamingMeasurementWRAIPS06.pdf>
- Ford B., Srisuresh P., and Kegel D. (2005, September 17). Peer-to-Peer Communication Across Network Address Translators. Retrieved 20 April 2007, from <http://www.brynosaurus.com/pub/net/p2pnat/>
- Hei X., Liang C., Liang J., Liu Y., and Ross K. (2006, November). A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*. Retrieved 28 May 2007, from <http://cis.poly.edu/~ross/papers/P2PliveStreamingMeasurement.pdf>
- Hei X., Liu Y., and Ross K. (n.d.). Inferring Network-Wide Quality in P2P Live Streaming Systems. Retrieved 25 May 2007, from <http://eeweb.poly.edu/faculty/yongliu/docs/index-buffermap.pdf>
- IPTV PPStream Info, Download PPStream, PPStream Guide, PPStream Channel*. (n.d.). Retrieved 12 June 2007, from <http://www.pprecorder.com/iptv-ppstream.htm>
- Liogkas N., Nelson R., Kohler E., and Zhang L. (2006, February). Exploiting BitTorrent For Fun (But Not Profit). *The 5th International Workshop on Peer-to-Peer Systems (IPTPS)*. Retrieved 19 January 2007, from <http://www.cs.ucla.edu/~lixia/papers/06IPTPS.pdf>
- PPStream*. (2007, June 7). Retrived 15 June 2007, from Wikipedia, the Free Encyclopedia: <http://en.wikipedia.org/wiki/PPStream>
- PPStream*. (n.d.). Retrived 9 June 2007, from <http://www.ppstream.com>
- Tu Y., Sun J., Hefeeda M., and Prabhakar S. (2005, November). An Analytical Study of Peer-to-Peer Media Streaming Systems. *ACM Transactions on Multimedia Computing, Communications and Applications*. Vol. 1, No. 4, pp. 354-376.