Examining Security Issues in Dolphin Netplay

Austin Taghavi
Department of Computer Science and Engineering
Texas A&M University
College Station, TX
Email: austintaghavi@gmail.com

Philip Van Ruitenbeek
Department of Computer Science and Engineering
Texas A&M University
College Station, TX
Email: philipvr@gmail.com

***Abstract*—Online gaming is no longer simply a past-time for the youth; it is not only a large industry, but also a career for many. Because of this, online gaming security is becoming more and more important. Hacking online games can amount to a serious offense if it is used to skew the results of games played for money. In this report, we explore the security concerns of Super Smash Bros Melee, as played through Netplay. We explore methods of cheating within the game, performing malicious activity, as well as possible security measures to address these problems.**

I. Introduction

Online gaming is as popular as it has ever been, and only continues to grow. With the advent of eSports, we are seeing competitive video gaming turn into a legitimate industry. Video games are played for fun, but they are also played for sport and, in some cases, as a career. This changes security issues and cheating in online gaming from an annoyance to a serious issue.

One such eSport is the Super Smash Bros franchise, from Nintendo. This is considered by many to be the first eSport, as the competitive Super Smash Bros scene emerged before the game was playable online. While newest installations of Super Smash Bros have online support through Nintendo, older versions do not. In particular, we are concerned with Super Smash Bros Melee, which we will refer to as SSBM throughout this report. The competitive community for SSBM began with groups of people meeting in person and holding tournaments. However, in recent years, ways of playing the game online have emerged. Dolphin is a Gamecube emulator for Windows, which provides a system called NetPlay for playing certain Gamecube games online.

We wish to explore the security issues with NetPlay for Dolphin, in particular how they pertain to SSBM. Our analysis will be threefold: (1) Cheating at SSBM through NetPlay for Dolphin, (2) Malicious activity via the connection provided by NetPlay, and (3) possible solutions for the issues we find.

II. Proposed Technique

To explore the methods of cheating at SSBM via NetPlay, we will first use a network traffic tool such as Wireshark to capture traffic during an SSBM session. This will allow us to analyze the way that data is sent back and forth to establish the game, and enable us to spoof input from one user to cheat.

Next, we will try various known methods of attack using the NetPlay connection, such as buffer overflow. We will note the results and how NetPlay can be used to facilitate malicious behavior.

Finally, we will propose possible solutions to the vulnerabilities that we have found during this project.

III. Related Work

The online PC gaming in the U.S. is a multi-billion dollar industry [1]. When online gamers encounter cheaters, they may feel that the game is ruined and may give up on playing the game [2].

Online gaming is the most successful software industry in Asia and has led to a rapid increase in cyber-criminal activity in Taiwan [3].

Steam, a popular online game platform, reported that 77,000 accounts are hijacked and pillaged each month [4]. It was later found by [5] that malware had been developed to steal Steam user credentials.

[6] uses a time series based user modeling approach to automatically detect compromised accounts in Massively Multi-player Online Role Playing Games. [7] found that idle time distribution is a representative feature of game players.

IV. NetPlay Overview

NetPlay is an online gaming platform that is provided by the Dolphin emulator. It provides a way for players using the emulator to play games against each other via the internet. These games do not necessarily need to have had online play originally implemented.

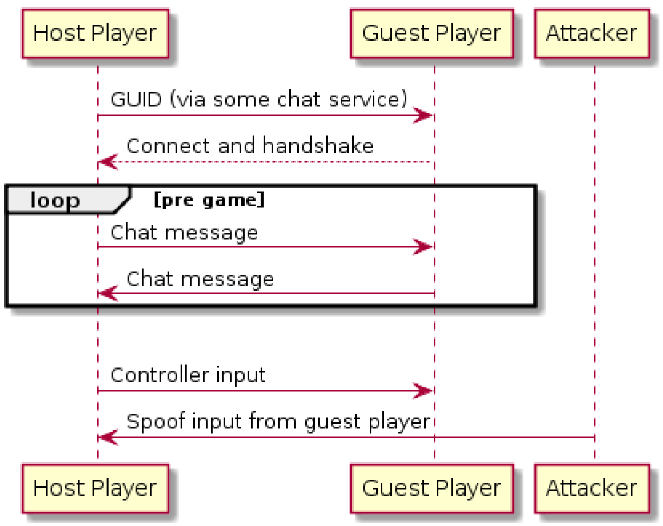
There are two different ways in which users can communicate with one another to establish a game.

1) Direct Connection
2) Traversal Server

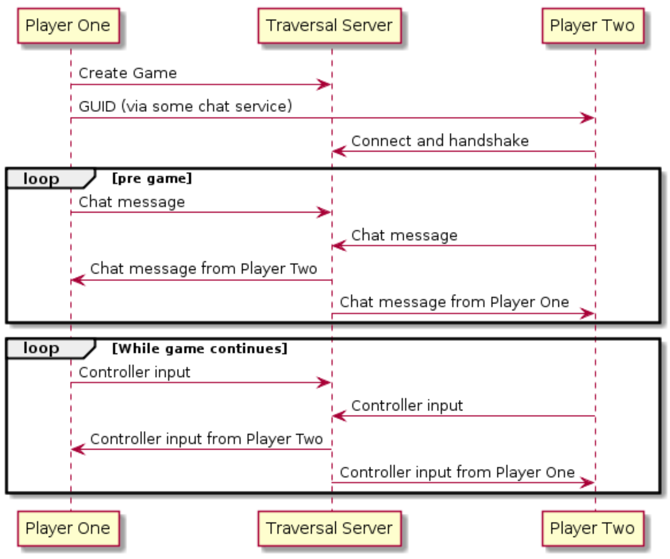
For both of these connection types, the flow of communication is the same. It consists of the following:

1) One user ”hosts” the game, and communicates a GUID for the game to the other player
2) The other user(s) ”connects” to the session.
3) The host decides when to start the game.
4) Both (or all) sides communicate their controller input for the game to take place.

An example sequence diagram for a direct connection is provided below:



And the equivalent diagram for the traversal server:



V. Source Code Analysis

The source code for the Dolphin emulator is available online. Because of this, we have the opportunity to view the source code to even further interpret the security vulnerabilities.

VI. Traffic Monitoring

Our first task was to monitor traffic during a game of SSBM via NetPlay. The goal of this is to capture and observe network traffic to analyze security vulnerabilities. From there we will design and explore possible exploits. Our experiment here is simple: Play a game of SSBM between two hosts via NetPlay, and monitor the network traffic during the game.

For monitoring, we used the tool ”Wireshark” to capture network traffic. During our capture, we used the following filter:

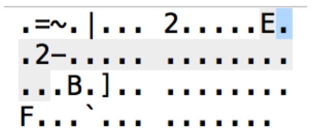
$(ip.src == IP_1 \&\& ip.dst == IP_2) || (ip.src == IP_2 \&\& ip.dst == IP_1) \&\& udp)$

This filter allowed us to only look at traffic that is to and from the two IP addresses that we used, and only UDP traffic of that. Initially the traffic was numerous and overwhelming. We were able to do shorter captures of different stages of the game to narrow down the traffic and more closely associate it with different actions in the game. For this, we broke the game down into two stages:

1) Initial handshake and pre-game
2) During game communication

Some of the initial handshake data is somewhat indecipherable to us. However, the pre-game client allows users to ”chat” with eachother, and this is easily decipherable. An example of such a packet is provided below:

For the actual game communication, it is our analysis that the games send ”controller input” back and forth to establish the game. An example of such a packet is provided below:



```
.=~.|... 2.....E.
.2-..... ........
...B.].. ........
F...`... .......
```

Each of these UDP packets contains data about a single controller input.

VII. Experiment 1: Third Party Intervention

In this first experiment, we explore a fairly simple exploit of the SSBM game via NetPlay. Our goal, in this experiment, is to interfere with the flow of a game via a third party host. To achieve this, we use our analysis of the monitored traffic from other games to be able to construct packets that we think will interfere with a game being played by two other hosts.

A. *Packet Spoofing*

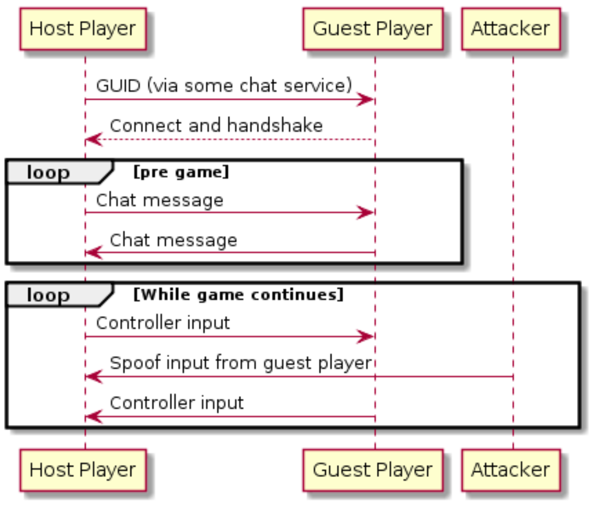
To achieve this goal, we will need to be able to construct and send out packets programatically. Particularly, we will need to construct UDP packets and be able to ”spoof” the sender address of such packets. By this, we mean that we need to be able to indicate that a packet is coming from another host. We have chosen to implement a simple packet spoofer in Python, and use this to perform the attack. We use the **scapy** Python library. A small sample code snippet is provided below:

```
spoofed_packet = IP(src=src_IP, dst=dst_IP) /
UDP(sport=src_port, dport=dst_port) /
payload
send(spoofed_packet)
```

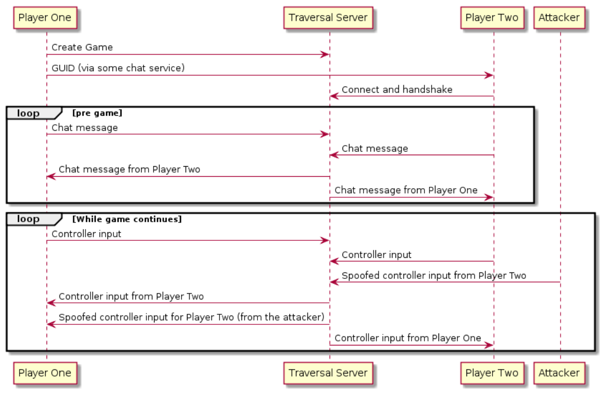
The **scapy** library makes the construction and sending of spoofed packets very simple.

B. Attack Overview

For this experiment, we will first begin a game of SSBM via NetPlay between two hosts. We will then use a third party computer to spoof packets to interfere with the game. The third party will spoof packets ”pretending” to be one of the hosts (the victim, in this case), and sending facetious and malicious data. An example sequence diagram for a direct connection is provided below:



And the equivalent diagram for a traversal server is:


Player One
Traversal Server
Player Two
Attacker
Create Game
GUID (via some chat service)
Connect and handshake
loop
[pre game]
Chat message
Chat message
Chat message from Player Two
Chat message from Player One
loop
[While game continues]
Controller input
Controller input
Spoofed controller input from Player Two
Controller input from Player Two
Spoofed controller input for Player Two (from the attacker)
Controller input from Player One
Player One
Traversal Server
Player Two
Attacker

This experiment is performed on the two types of game connection, namely:

1) Direct Connection
2) Traversal Server

The different between the two is that for the direct connection, we will be spoofing packets to the other host. In the case of the traversal server, we will be spoofing messages to the traversal server.

References

[1] D. Takahashi, “U.S. games industry forecast to grow 30 percent to $19.6b by 2019,” Jun 2015. [Online]. Available: http://venturebeat.com/2015/06/02/u-s-games-industry-forecast-to-grow-30-to-19-6b-by-2019/

[2] J. J. Yan and H. Choi, “Security issues in online games,” *The Electronic Library*, vol. 20, no. 2, pp. 125–133, 2002. [Online]. Available: http://dx.doi.org/10.1108/02640470210424455

[3] Y.-C. Chen, P. Chen, R. Song, and L. Korba, “Online gaming crime and security issue-cases and countermeasures from taiwan,” *Proceedings of the 2nd Annual Conference on Privacy, Security and Trust*, 2004.

[4] “Security and trading,” Dec 2015. [Online]. Available: http://store.steampowered.com/news/19618/

[5] S. Pontiroli, “All your creds are belong to us,” Mar 2016. [Online]. Available: https://securelist.com/blog/research/74137/all-your-creds-are-belong-to-us/

[6] J. Oh, Z. H. Borbora, and J. Srivastava, “Automatic detection of compromised accounts in mmorpgs,” in *Social Informatics (SocialInformatics), 2012 International Conference on*, Dec 2012, pp. 222–227.

[7] K.-T. Chen and L.-W. Hong, “User identification based on game-play activity patterns,” in *NetGames‘07: Proceedings of the 6th ACM SIGCOMM workshop on Network and System Support for Games*. ACM, 2007, pp. 7–12.