MMORPG Trading Covert Channel

Joshua Geise
Rochester Institute of Technology
jtg6159@rit.edu

Charissa Miller
Rochester Institute of Technology
clm5067@rit.edu

Laura Weintraub
Rochester Institute of Technology
lhw2553@rit.edu

ABSTRACT

Covert channels are on the rise in order to secretly transfer information for private communication. Video games are particularly of interest for secret communications because of their popularity, large data network, and accessibility at all hours of the day. This paper proposes a covert channel via a common game functionality of MMORPGs: player-to-player trading. A trade is only between two players, therefore, the contents of the trade is private. In-game trading is very common, thus attention is not drawn to the characters due to suspicion. The encoding of a message is performed using tradable items that correspond to various bit values.

This channel is demonstrated in the online game, Old School RuneScape. It achieved a bandwidth high of 1.600 bits/second with an error rate of 0%. Improvements can be made to increase bandwidth by assigning items larger bit values. The approach taken for this implementation can easily be extended to other MMORPGs.

KEYWORDS

Covert Channel, MMORPG, RuneScape

ACM Reference format:
Joshua Geise, Charissa Miller, and Laura Weintraub. 2018. MMORPG Trading Covert Channel.

1 INTRODUCTION

Online games are an attractive means for covert channels as there is a wide range of types and sizes. Channels can easily be created in the peer-to-peer network since many online games allow individuals to host their own private servers. Suspicion would not be drawn to a user even if they were to continuously connect to the same private server, due to the sense of consistent community around games. In addition, games are typically played for a long duration of time and can be reached at all hours of the day, making communications even more common [13]. Covert channel within online games are difficult to detect due to the popularity and large amount of data generated by players.

We demonstrate that a covert channel can be formed using a feature commonly found in massively multiplayer online role-playing games (MMORPGs). Similar to Rook's [13] and Castle's [12] framework, which are described in Section 3, our channel will extend across many other MMORPGs by manipulating the use of the trading functionality. Since trading is a common mechanic within certain MMORPGs, this channel has a stealth advantage over other gaming covert channels that embed information in network packets, for example.

In order to transmit a message via a trade, each item needs to correspond to a bit value. MMORPGs are notable for the large amount of tradable items in each game. As some items are more common than others, it is important to address how this will affect our channel. This is discussed in Section 5, as we consider the options for assigning values to items.

This channel has the potential for a large amount of bandwidth due to the number of items in-game that can be assigned bit values, as well as little to no errors during transmission. These two characteristics provide more utility to people using the channel in comparison to other gaming covert channels. The majority of gaming covert channels have neither the ability to transmit a large amount of data nor significant error bit rates if an increase of speed is attempted.

We utilize Old School RuneScape (OSRS) as our platform for our channel implementation due to its current popularity and large user base. Custom scripts are run on a pre-existing bot client to trade items in a specific order that ultimately relays a message to the receiving player of the trade. The receiving player will then convert the binary message taken from the items back into the original message. To further decrease the potential detection of the channel, the receiving player will always decline the trade, resulting in both players holding the same items they did at the beginning of the trade.

2 BACKGROUND

As trading is a commonly used feature in many MMORPGs, it is important to understand how they operate. First, one player will need to initiate the trade. Typically, this occurs by right-clicking a player and selecting the "trade" or similar option in the drop-down. Once the receiving player accepts the invitation, both players can propose items to trade. Each user will see empty item 'slots' to which they can add items and the screen will update to show the items added by the other player. Lastly, both users can review the offer and choose to accept or deny the trade. Both users must accept during this stage in order for the trade to be successful.

2.1 RuneScape Trading

A trade in RuneScape is done exclusively between two players, which means that no players outside the trade have the ability to view nor to determine the items that are being traded. To initiate a trade, it must be requested from one player to another and, once accepted by the receiving player, will open a new trading window [4]; this window is shown in Figure 1. In this window, each player can send up to 28 items to trade from their inventory, which provides plenty of room for a message to be transmitted using the vast amount of items available in the game. Once the items to be traded have been added, both players can either choose to accept or

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, Washington, DC, USA
© 2018 ACM.

decline the trade [4]. A declined trade will exit the trading window for both players and their inventory items will remain unchanged. An accepted trade will bring the players to a second trading screen that shows a summary of the trade and gives them a second chance to decline the trade. Upon a successful trade, the items indicated for trade will be transferred to the other player.

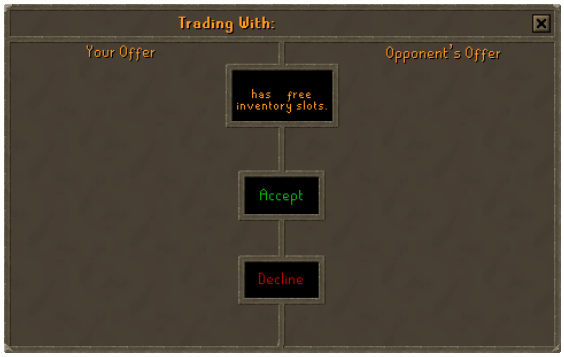

Figure 1: Empty trade screen in OSRS

While there are approximately 14,000 items registered in the game [8], not all of these items are tradable. According to an unofficial source, there are approximately 3,300 tradable items in OSRS [3]. Furthermore, some of these items are uncommon or expensive and should not be used in a covert trading channel. As stated above, each trade supports a maximum of 28 unique items. But, each trading slot can "stack" multiple of the same item to contain 2^{31} - 1 items, thus expanding the range of values that can be sent in a single trade. "Stacking" items is not in the scope of our channel implementation and is further discussed in Section 8.

2.2 World of Warcraft Trading

A trade in WoW is conducted only between two players, and no one outside of these two players can view or determine the items being transferred. To initiate a trade, a trade request is sent from one player to another. If the trade request is accepted, a trade window is opened [1]. An example of the trade window is shown in Figure 2.

In this window, each player is able to trade up to six items at at time. This is done by dragging items over from their inventory into the boxes shown in the picture above. Once a player has dragged over everything that they wish to trade, and are happy with what they are receiving in return, they click the "Trade button". This will cause their side of their trade window to be highlighted in green [1].

The completion of the trade can be requested by either player. If the other player agrees with the current state of the trade, they can also click the "Trade" button to complete the trade. If the other player doesn't agree with the trade, they can add/remove items on their side before accepting the trade or cancel the trade entirely by clicking the "Cancel" button. If the items in the trade window are moved, it will remove the green highlight on the other player's side of the trade window and require them to re-accept the trade as well [1].

There are around 111,021 items that exist within the game, however are a lot of items that are extremely expensive or hard to obtain [11]. We were unable to find a list of tradable items within the game, but we were able to calculate that there are 50,228 items in the "poor", "common", and "uncommon" categories in the game [11]. These categories are the three lowest tiers of item classifications, which means that they are the easiest items to obtain or the cheapest items to buy. This shows that there is a very large pool of items to assign bit values to within this game.

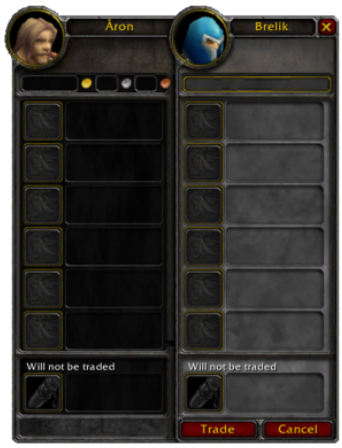

Figure 2: Empty trade screen in WoW

3 RELATED WORK

Video games are an appealing means of covert communication for many reasons, including the potential to easily hide embedded data in common game features. The majority of video games have commonalities with each other, allowing a single covert channel framework to work across multiple games. In-game voice or text chat channels have been used to form covert channels. Transmitting large amounts of data using such channels can be difficult though, as it is uncommon in Real-Time Strategy (further referred to as RTS) games, and long text or audio conversations is rare. This has become outdated as many communication channels are un-encrypted and monitored by gaming companies [12]. Research has been done on covert channels within many different games, but we focus our attention to three specific approaches that are most relevant to our channel: FPSCC, Rook, and Castle.

3.1 FPSCC

In 2008, Zander et al. proposed a covert channel that embedded information in the traffic of a multiplayer first person shooter online game (FPSCC) called Quake III Arena [14]. Quake III makes use of UDP/IP packets to transfer information between the game host and the players. The covert channel proposed worked by varying the character's movement by a slight and continuous amount and embedding data into the network traffic. The player movements they varied ranged from left to right, up and down, forward and backward, as well as pitch and yaw of the viewing angle. This

allows the covert channel to have a large enough bandwidth due to the high quantity of imperceptible variations of movement without being detected [14]. The implementation of their channel introduces the concept of noise that occurs from the loss of UDP/IP packets in the network or from unexpected player changes, such as a player death. Thus, this channel has a potential for increased error rates.

Player movement is a common and expected game mechanic within Quake III, therefore, the exploitation of minuscule adjustments of this mechanic prevents the detection of the channel [14]. Following this notion, our covert channel will also exploit the use of a standard and frequently used game mechanic for the sake of concealing our message transmissions. Additionally, our channel is implemented such that there is little to no errors in data transmission.

3.2 Rook

Another online gaming covert channel presented, Rook, transmitted data by embedding it into network traffic of an online game without changing the length or invalidating the game packets [13]. Using the game Team Fortress 2 as their channel's host, Rook selected approximately one-in-ten packets and replaced some of the mutable data with other game values in order to convey a message. The fields were chosen for modification based on their importance to the processing of the game's network traffic [13]. In particular, fields that were not necessarily used for specific game protocols were targeted, as the packets still needed to pass as valid traffic. The encoding and decoding of the message embedded into these packets utilized a lookup table that is shared by both the sender and receiver. The lookup table was created to store the normal gameplay values observed for each of the mutable fields.

Similar to Rook, our covert channel utilizes a lookup table accessible by both sender and receiver to convert the message from text to bits, and vice versa. However, Rook's current system design in finding and contacting the game server is not as adaptable towards other games as it could be [13]. Our channel bypasses this issue by utilizing bots to contact the servers and applying the channel to a common game mechanic.

3.3 Castle

Lastly, we explore Castle, a general covert channel mechanism that can be used in a large number of real-time strategy games. Rather than focusing the implementation on a particular game, Hahn et al. chose to create a overarching covert channel framework to work with other RTS games [12]. The channel encoded data by using a custom algorithm involving game-specific information, such as number of player units or building locations. Castle exploited common commands and access to replay logs in order to encode the data into the commands and execute them using an existing desktop automation tool, AutoHotkey [12]. On the receiving end, the process monitors logged the game commands and decoded them to find the hidden data.

While there are limitations to this framework, such as computational resource restraints, Castle is able to successfully transmit secret messages without using modifications that are specific to an RTS game. With the exception of the botting client, our covert channel does not need the use of significant computational resources.

4 THREAT MODEL

The structure of the game allows a covert channel to be easily implemented into existing game mechanics. Many useful mechanics can be used to hide data, such as embedding it within network traffic, player movements, or other player functions. An attacker may utilize these mechanics in a way that is similar to realistic usage, thus making it more difficult to detect. The most feared result of a covert channel within an online video game is detection of secret messaging.

Bots are created for many online games and put on the web both for free and for purchase. OSRS has become a popular game to create bots for and has many available, such as RuneMate [9], EpicBot [7], TRiBot [10], and DreamBot [6]. Many of these existing free botting clients will be caught by the gaming company, which in this case would be Jagex, leading to the accounts being banned from the system. However, more sophisticated bots utilizing custom scripts may be less easily distinguishable from normal gameplay, allowing the covert channel to be more successful. In our implementation we assume that Jagex has the following abilities:

- Monitor for and identify evidence of bot usage on player accounts with Botwatch software [5]
- View logs of player movements and interactions
- View logs of player trading activity including the time of the trade, items traded, and whether it was declined or accepted

Since trading is a popular mechanism in OSRS, a trading covert channel should bypass Jagex's detection of abnormal behavior. Log files containing trading activity would not be suspicious, provided that the bots do not attempt to send an abnormal number of trading requests in a inhuman amount of time. A high number of successive trade declines may also impact this channel. Although this channel will be automated using a bot, the traffic for this channel should blend in as normal trading noise and remain undetected by Jagex.

Further, Jagex uses a bot detection software called Botwatch [5] that looks for inhuman behavior, such as chatting in game while the bot is actively using the computer mouse. While the particular functionality of this software is unknown to the public, it is known that it is an automated program that can directly ban accounts [5]. Our covert channel will avoid detection by Botwatch by using a high-quality automated botting client that uses normal human behavior, such as timely use of mouse clicking and an imperfect mouse movement path.

5 OUR APPROACH

In order to establish a trading covert channel within an MMORPG, a functional automating system must be chosen. Many botting clients are free to use and have been designed to automate player movements and actions that are undetectable by the wardens. The process of trading should be automated for efficiency and speed, as it would not be easy for users to encode their message into the associated bit values on their own.

As the majority of covert methods involve encoding data for transmission, it is important to understand how encoding works. Encoding is the process of creating a message that can be properly transmitted from the sender to receiver. In the case of a covert channel, encoding can mean converting the message into a different

base such as binary, unreadable symbols, or another medium that will remain hidden from analysis and understood by the receiver. The receiver would need to have the means to convert this encoded message back into its original form.

Messages are delivered between players by exchanging game items within a trade. There must be care in picking which items should be assigned values, as trading expensive or difficult to obtain items may trigger warnings for suspicious activity. The most reasonable way to avoid detection is to use commonly used items in the automated trades. Also, a one-sided trade, where one user proposes many more items or more expensive items than the other player, might also surface malicious intent. Lastly, the amount of items that can be held by a user at one time needs to be taken into consideration as well. This is due to the inability to trade more items than the user currently holds. In some cases, users can visit a bank to take-out the necessary items to successfully hold the items needed to transmit the message. But, in other cases, this is not possible in the middle of transmission.

Items are assigned a unique value and stored in a lookup table that is accessible to both players. This lookup table will define the binary values for each item that is selected to be used in the automated trades. The number of bits associated with each item will depend on the quantity chosen to be used in the lookup table. If we strive for each item to be associated with a 4 bit values, we would need 16 items. The generalized formula for the number of items required is 2^n, where n is the number of bits per item.
The process to transmit a message using the trading functionality of an MMORPG is as followed.

Sender:

(1) Translate their plaintext message into binary.
(2) Convert the binary string into a list of items based on their corresponding game items in the lookup table.
(3) Gather necessary items to transmit the list of items.
(4) Initiate a trade with the receiver and wait for receiver to accept.
(5) Add items in the correct order of the list to convey the proper message.

Receiver:

(1) Accept the invitation to trade with the sender.
(2) View the items offered in the trade by the sender in same order they were added in.
(3) Use an identical lookup table to translate the traded items back into their binary values to decode the message.

The players can initiate multiple trades if needed, based on the size of the message, until the EOF item is received. Items in a trade are received in the same order they are given, allowing the receiving player to accurately convert the values of the items back into the binary of the message.Since there are a large number of items within the game, we expect to see a large bandwidth for this channel and little to no errors during transmission. We also expect that this channel will be difficult to detect due to the commonality of trading. Additionally, the desired message to transmit will not be embedded into network traffic, allowing this channel to be stealthy.

6 IMPLEMENTATION

This covert channel will demonstrate secret message transmission through the use of trading in Old School RuneScape. We utilize a bot software called DreamBot [6] with scripting functionality to designate the behavior of the two communicating players. In our contribution, we determine the necessary game items to trade and their corresponding binary values, which will be stored in a pre-determined lookup table shared between the two players; this is discussed in Section 6.2. We then design a custom Java script to work with DreamBot, utilizing DreamBot's existing Trading library [2], to control the two players' actions and translate the message into items that can be traded. These items will then be used to transfer messages between the two players in a way that is undetectable by the game and other players, discussed in Section 6.3. Finally, we provide an evaluation of our channel's bandwidth and error rate in Section 7.

The implementation of our channel can be found on GitHub at: https://github.com/joshuageise/GamingTradingCC.

6.1 Platform

RuneScape is a click-based fantasy massively multiplayer online role-playing game (MMORPG) game that was released in January of 2001. There are currently two versions of the game, Old School RuneScape and RuneScape 2. RuneScape 2 is a new version of the game that has updated graphics and new mechanics, while Old School RuneScape is available for players that prefer the classic feel of the game before the update. The game allows players to complete quests, engage in combat, improve their skills, increase their wealth, and trade each other; the latter is the most important game mechanic for this covert channel.

For the purposes of creating this covert channel, we use Old School RuneScape due to the popularity of trading, as well as increased quality and efficiency of bots created by online users. A high quality bot is important in order to prevent its detection within the game and to support custom made scripts on which the covert channel heavily relies. Additionally, since this channel focuses on trading, it is important to note that although there are many tradable items in OSRS, some of them are uncommon or expensive and are not used in the implementation of our channel.

OSRS Item	Binary Value
Bones	0
Egg	1
Pot	EOF (non-binary)

Table 1: Lookup table used for message encoding with 1 bit value per item and the EOF item

6.2 Lookup Table

The lookup table is implemented as a `Hashtable` due to the small number of items being used for the purposes of the preliminary covert channel implementation. As a simple workaround for a backwards lookup in the `Hashtable`, we implement a second reversed `Hashtable`.

An example lookup table, where each item only corresponds to one bit, is shown in Table 1. This is used for our first version of this trading covert channel, due to simplicity. This type of table will continue to expand when more items are used to convey higher bit values. Logistically speaking, if there are 27 trading spots available, 28-1 due to the EOF item, then there can be $27n$ message bits transmitted in one trade, where n is the number of bits per item value. Therefore, in our current implementation with 1 bit per item, we can only transmit 27 message bits. If we were able to use 8 bits, or 1 byte, per item, we would be able to transmit *27*8=216* message bits.

The lookup table is used in the PREP state and the RECEIVE state. These states are defined and explained in Section 6.3. In order to convert the ascii message into a transmittable form in the PREP state, the lookup table is needed. Each item must correspond to a certain bit value for the binary string to change into a list of tradable items. Once the sending player is finished offering items, the receiving player will gather a list of all items that were seen in the sending player's side of the trade screen in the RECEIVE state. Lastly, the same lookup table is used to convert the list back into its binary form.

The lookup table can be transformed into a more efficient structure in the future, as we will discuss in Section 8.

6.3 Data Transmission

There are three possible states that a player in our script could reside in: PREP, TRADE, or RECEIVE. This is represented as an `enumeration` in Java. To determine which state a player should be in, there is a method called `getState()` in the script.

The `getState()` method first collects the name of the player running the script. If that player is designated as a sender in the script and the `ArrayList` of items is empty, then the player is set to the PREP state. If the player is designated as a sender and the list of items is filled, the player will be set to the TRADE state. Lastly, if neither of these conditions are true, the player will be set to the RECEIVE state.

6.3.1 PREP. This state encompasses the conversion of the ascii message to a list of tradable items. Once this process has occurred, and the `ArrayList` of items has been filled, the player will change status to the TRADE state.

First, the desired message to be sent is converted to its binary representation using ascii values. The script then loops through each bit of the binary string to find the corresponding item from the lookup table, explained in Section 6.2. This generates an `ArrayList` that lists the items to be traded in a specific order so that the message is transmitted properly. The end of file (EOF) item is also added to this list of items at the end of the array.

To demonstrate, we use the example string *"hi"* as our message and the lookup table shown in Table 1. This message is first converted into a binary string with the value of *"0110100001101001"*. The script then loops through this binary string to determine the corresponding items per bit. Each item is added into the `ArrayList` in the same order it was found, resulting in a list containing: "[Bones, Egg, Egg, Bones, Egg, Bones, Bones, Bones, Bones, Egg, Egg, Bones, Egg, Bones, Bones, Egg]". After all bits of the message have been traversed, the EOF item, *"Pot"*, is added to the end of the list.

6.3.2 TRADE. To begin the transmission, a new trade is initiated by the sender to the receiver based on their character's name using the function `tradeWithPlayer()` [2]. The sender must then wait for the receiving client to accept the trade. Once the connection is started on both ends, the trade will officially begin.

Upon connection, the first trade screen will appear on both the sending and receiving clients. The sender will add each item from the items `ArrayList` in the order they appear, using the `addItem()` method [2]. Once all of the list has been traversed, the trade window should have all of the message (with the EOF item) or the first 28 items worth of the message. Each time an item is added to the trade by a player, it will appear in this window for both to see.

6.3.3 RECEIVE. The receiving player will also create a new trade with the sender for a connection to be established. The receiving player observes the trade window for the items the sender adds, using the trade function `getTheirItems()` [2]. This function will return a `List` of items that have been added by the other player to the current trading screen. Once the EOF item is identified or the list is 28 items in length, the receiving player will terminate the trade by declining using the `declineTrade()` [2] function. This means that the second trade screen is never reached and items are never transferred to the receiver. Lastly, the receiver will convert the items back to binary form using the lookup table once again, discussed in Section 6.2.

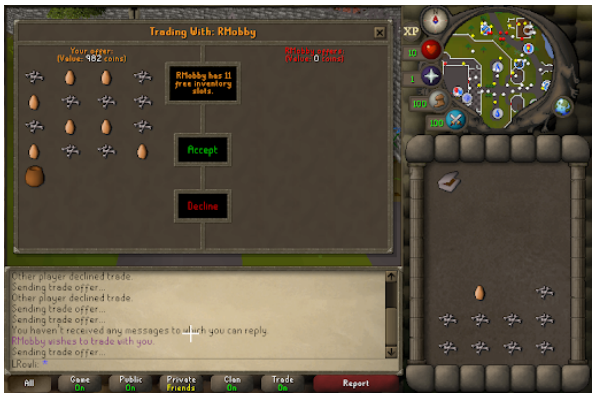

Figure 3: Trade screen with example message "hi"

7 RESULTS & EVALUATION

Upon creating a minimal viable product using DreamBot scripts, we learned that most of our initial predictions regarding automating the trades were correct. The current version of our covert channel supports a message up to 27 bits, since the trading window only allows 28 items and we need to allocate one spot for the EOF item. We discuss ideas on how to transmit longer messages and create a more efficient system in Section 8.

Testing of this covert channel with various message length showed an error rate of 0%. This is a tremendous accomplishment that most other covert channels that we have studied have been unable to achieve. Because there is no errors in the transmission of messages, we are able to suspect that larger messages, if done

properly, could also attain a zero error rate. The highest bandwidth achieved by this channel is 1.6 bits per second. This bandwidth might not appear as impressive, but the ability to trade items perfectly shows the potential of this covert channel. Once more tradable items are incorporated, the bandwidth will increase due to the smaller amount of items needed to transmit a message. This is because items will correspond to larger bit values rather than the single bit values of 0 or 1. Table 2 shows a comparison between the total time it takes to transmit the message, the bandwidth occurred, and error rate achieved. To generate this table, we tested a message of length 8, 16, and 24 bits.

Message Bit Length	Total Time	Bandwidth	Error Rate
8	12.80s	1.600 b/s	0
16	20.22s	1.264 b/s	0
24	29.15s	1.214 b/s	0

Table 2: Bandwidth and error rate results of the covert channel at varying message lengths

To expand on the results shown in this table, we can discuss the predicted bandwidth once the functionality for a larger lookup table is added. We are able to translate the total time our channel took to transmit a message of 24 bits, 29.15s, to transmitting a message of 24 items long. We use the following formula to determine the estimated bandwidths for the If each item were to represent 4 bits, then the estimated bandwidth would be 4.858 b/s. Further, if each item represented 8 bits, or 1 byte, then the estimated bandwidth would be 9.717b/s.

$$Bandwidth = \frac{Total\ Time}{Message\ Bit\ Length}\ x\ Number\ of\ Bits\ per\ Item$$

8 FUTURE WORK

This covert channel has the potential to be more efficient and transmit larger messages. Currently, only two tradable game items are being used to represent the single bit values of 0 and 1. To move forward with and improve this channel, more items should be used to represent larger bit values, thus increasing its efficiency and bandwidth. This would entail using a larger number of unique tradable game items and utilizing a banking system of some sort, as players would not be able to hold all of the items needed for the trade. This would expand the channel's lookup table to include the same EOF item with 2^n items that could be used to encode the desired message, where n is the number of bits per item value. The channel would then be able to transmit covert messages between players that are much larger, and eventually, even transmit a text file.

Additionally, the implementation does not currently cover longer messages that would need more than 27 items in the trade to convey. Here, we see two potential future routes. The first would be to decline the trade and initiate a new one continuously until the whole message was transmitted. Alternatively, the sender may be able to clear the trade screen instead of closing and reopening a trade with the receiver. The latter would decrease the amount of trades that may be logged between the two players.

Another option to increase efficiency would be to stack items in the trading window. Although not all items are stackable, the items that are may be able to represent an even larger set of bit values in the lookup table. This would also assist in transmitting longer messages via trading.

These options are not the only directions in which this channel could expand. This project has proven that covert communications in RuneScape is possible through the use of player-to-player trading. Since many other massively multiplayer online role-playing games also take advantage of trading between users, this research could be applied and tested on these platforms as well. In order to extend this channel across other gaming platforms, research must be taken to find the amount of trading slots possible. Then, the amount of unique items and how many bits each should hold can be discussed for longer messages to be transmitted.

9 CONCLUSIONS

RuneScape proved to be a worthy platform for such secret communications through the use of the popular player-to-player trading feature. Under the assumption that the developing company, Jagex, can access logs containing trading information between players, the automated trades executed by this covert channel do not seem detectable. Instead, they appear as normal traffic that would occur if two players initiated a trade and then one decided to decline after some items were proposed.

The covert channel we propose currently achieves a range of 1.214 to 1.600 bits per second with a non-existent error rate. We tested messages up to 24 bits in length, but there were 0 errors, proving that the intended message was always received correctly. With the additions proposed in Section 8, the efficiency and bandwidth of this channel can improve tremendously.

REFERENCES

[1] 2012. WoW Trading. http://wowwiki.wikia.com/wiki/Trade. (2012).
[2] 2017. Class Trade. DreamBot Libraries: https://dreambot.org/javadocs/org/dreambot/api/methods/trade/Trade.html. (2017).
[3] 2017. Tradable Items. https://pastebin.com/LyZyZhpf. (2017).
[4] 2017. Trading. The RuneScape Wiki, https://runescape.fandom.com/wiki/Trading. (2017).
[5] 2018. Botwatch. https://runescape.fandom.com/wiki/Botwatch. (2018).
[6] 2018. DreamBot - #1 OSRS Bot for Runescape. https://www.dreambot.org. (2018).
[7] 2018. EpicBot - Free RuneScape Bot, Old School 07 & RS3 Bot, Runescape 3, OS Bot. https://www.epicbot.com. (2018).
[8] 2018. Item Database. RuneHQ, https://www.runehq.com/item. (2018).
[9] 2018. RuneMate — Free RuneScape 3 & Old School 07 Bot Client. https://www.runemate.com. (2018).
[10] 2018. TriBot: RuneScape Bot. Trilez Software Inc. [CA], https://www.tribot.org. (2018).
[11] 2018. WoW Items. https://www.wowhead.com/items. (2018).
[12] B. Hahn, R. Nithyanand, P. Gill, and R. Johnson. 2016. Games without Frontiers: Investigating Video Games as a Covert Channel. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 63–77. DOI: http://dx.doi.org/10.1109/EuroSP.2016.17
[13] Paul Vines and Tadayoshi Kohno. 2015. Rook: Using Video Games As a Low-Bandwidth Censorship Resistant Communication Platform. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society (WPES '15)*. ACM, New York, NY, USA, 75–84. DOI: http://dx.doi.org/10.1145/2808138.2808141
[14] S. Zander, G. Armitage, and P. Branch. 2008. Covert channels in multiplayer first person shooter online games. In *2008 33rd IEEE Conference on Local Computer Networks (LCN)*. 215–222. DOI: http://dx.doi.org/10.1109/LCN.2008.4664172