Rook: Using Video Games as a Low-Bandwidth Censorship Resistant Communication Platform

Paul Vines
University of Washington
plvines@cs.washington.edu

Tadayoshi Kohno
University of Washington
yoshi@cs.washington.edu

ABSTRACT

Censorship and surveillance is increasing in scale, sophistication, and prevalence across the globe. While most censorship circumvention systems still focus on escaping a given censored region to access Internet content outside of its control, we address a different but equally pressing problem: secure and secret chat within a censored region.

We present Rook as a censorship and surveillance resistant platform for communication using online games as its cover application. The use of online games represents a novel form of cover application that provides several features that make them uniquely well-suited for this purpose. Rook transmits data secretly by embedding it in the network traffic of an online game. To mitigate current attacks based on deep-packet inspection and traffic shape analysis Rook uses the normal traffic used by the game, it does not generate additional packets, does not change the length of existing packets, and ensures altered packets are still valid game packets.

For evaluation, we implement Rook using the online first-person shooter Team Fortress 2. Rook is evaluated against both active and passive attacks demonstrated in recent years including anti-mimicry probes, deep-packet inspection, traffic shape analysis, statistical analyses of packet payloads, and game-specific n-gram analyses.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Censorship circumvention, covert channel, steganography

Keywords

Censorship, covert channel, steganography, game, online, first-person shooter, privacy, anonymity, surveillence

WPES'15, October 12 2015 Denver, CO USA

ACM 978-1-4503-3820-2/15/10$15.00
DOI: http://dx.doi.org/10.1145/2808138.2808141 .

1. INTRODUCTION

There are increasing concerns about the privacy of online communications throughout the world: both surveillance and censorship of the Internet has continued to increase in scale, sophistication, and prevalence [6, 13, 34]. There have been many systems developed for attempting to hide communications on the Internet from censors, we will simply refer to these as circumvention systems. Most of these are designed to route a user's traffic outside of a censored region to access censored content. These systems have many different approaches and goals with respect to circumventing censorship and surveillance, and what kinds of attacks they can withstand. This has led to an arms race situation in which new circumvention systems are invented and improved followed closely by attacks against them similarly being invented and improved.

In this paper we present Rook as a low bandwidth low latency (approx. 30 bits/second) censorship resistant communication platform. While Rook can be used to secretly communicate any kind of data, the structure of its network and limitations of its bandwidth make it best suited for chat applications. Whereas most circumvention systems focus on enabling censored users to access banned external content, Rook focuses on enabling uncensored and unserveilled communication between parties within a censored area. We intend Rook to be used to facilitate secret IRC-like services among users. These services can run the Off-The-Record (OTR) protocol [2] between clients to ensure end-to-end security even from the Rook server.

Rook was partially inspired by recent works on pushing circumvention systems further into the application layer [11, 18], but also represents a new direction for circumvention systems in several ways: first, it utilizes low-latency online games as its cover traffic. Only one other system, Castle [7], published concurrently, has also explored this area of applications. Two other previous systems have explored using computerized boardgames as covert channels [8, 23] as well, we further discuss these three systems in relation to Rook see Section 5. Second, Rook alters the host game traffic in full compliance with the application's protocol and does not artificially add or lengthen any game traffic; this means an adversary must first commit the resources to develop a program to correctly parse the application protocol, and then must also commit computational resources to each suspected Rook connection because a simple deep-packet inspection cannot detect use of Rook. As is the case with all of these circumvention systems, as the censor's resources increase its ability to potentially detect or disrupt Rook also increase;

we outline possible new research directions for attacks below (see Section 5 and 6). However, Rook does represent a significant increase in the protection afforded users given the increase in cost to attackers.

Many different types of applications have been used as cover or host applications for previous circumvention systems. Skype in particular has proven popular because its calls operate on a peer-to-peer connection and are assumed to be a reasonably legitimate activity for a user to be engaged in [11, 21]. A key challenge with using Skype for circumvention is that prolonged usage of Skype may not match many users' normal behavior patterns. Online games provide a similar opportunity but with greater deniability on the part of users. Traditionally, many online games have allowed individuals to host their own private servers to reduce the resource burden on the companies making the games. This is particularly the case for the genre of game Rook is primarily focused on: the First Person Shooter (FPS). The existence of privately-hosted servers creates opportunities for communities to arise and causes many regular players of these games to play almost exclusively on one or a handful of servers. This provides a completely legitimate cover for a Rook user to repeatedly connect to the same game server over and over to communicate with the other Rook users also connecting to these game servers. Furthermore, legitimate players will often play for hours at a time, day after day. This could be significantly more suspicious in the case of another application, such as Skype, repeatedly being used to contact the same IP for hours at a time every day. Finally, like VoIP services, games are a widespread and popular form of network use [27]; we believe a censor would face similar dissent to a decision to block all Internet-gaming as they would to blocking all VoIP. The design of Rook is not specific to a game, and so if a censor attempted to block Rook by blocking a single game, Rook could simply be adapted to another.

Another advantage of games is that they do not imply actual communication between users unless the in-game chat channel is used, which is unencrypted. A VoIP connection inherently implies the two parties are sharing information, while a game client connecting to a server does not imply any user-to-user communication. This creates a plausible deniability that Rook users are connected in any way so long as they are not communicating using the in-game chat. Whereas a VoIP connection implies two IPs are exchanging information, a game connection only implies two IPs happen to be playing the same game, and probably are not even aware of who the other IP is. Rook also allows covert communication without disrupting the normal use of the game; a Rook user plays the game normally while additionally sending covert messages.

Rook was developed with the FPS as the primary type of game to be used: this is because of the prevalence of private servers. FPS games generally feature between 8 and 128 players on a server at a time. Each player controls one avatar inside a 3-dimensional game world in which they attempt to maneuver and kill the other avatars. As an example implementation, Rook uses the FPS Team Fortress 2 [28], based on the Source Engine from Valve Software. The Rook design can also be used for other types of games, although the game must allow private servers or have a peer-to-peer architecture.

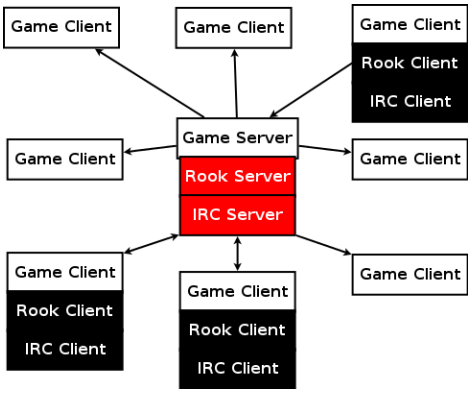

Figure 1: Example formation of a Rook network: one Rook server with multiple Rook clients and normal game clients connected to it.

2. SYSTEM DESIGN

Rook is a system to facilitate secretly passing data between two Rook clients and a Rook server, operating on machines running game clients and a game server, respectively. In this case, secret is defined as the act of sending the data being unobservable from to an outside observer, as well as the contents of the data itself being encrypted. The data channel between a Rook client and Rook server is composed of two one-way channels to create an overall bidirectional data channel. The creation of the channel requires a shared secret key between the Rook client and Rook server.

2.1 Intended Use Model

The high-level overview of Rook is:

- Rook clients run the game client
- Rook servers run the game server
- Clients connect to the server by joining the game server
- The Rook connections provide low-bandwidth real-time communication free from surveillance or censorship
- Clients chat with other clients using the server and Off-The-Record messaging

2.2 Threat Model

For the rest of the paper we assume the following threat model for our monitor adversary, similar to the standard warden and prisoners threat model used in steganography [1]. The monitor is attempting to detect and/or block the exchange of any information besides legitimate game information. The monitor is considered to have the following capabilities:

- All network traffic between all clients and the server are observed
- The normal application traffic is unencrypted
- The monitor can store all data observed over a gaming session and run statistical analyses on it
- The monitor has no internal access to the devices running the game clients and server
- Users can conduct a 1-time secure rendezvous to exchange a shared secret key
- The monitor does not wish to disrupt legitimate gameplay of innocent users

- The monitor can conduct some active probing of game client and servers, but is not willing to disrupt legitimate traffic for extended periods of time
- The monitor seeks evidence that information, aside from normal game data, is being exchanged

2.3 Criteria for Success

Rook is a successful censorship resistant platform if the monitor cannot either:

- Positively identify use of Rook more often than falsely identifying legitimate game traffic as use of Rook
- Successfully disrupt Rook communication without impacting legitimate gameplay.

We assume that even if Rook becomes popular, it will still represent a minority of game traffic. Therefore, the first criterion is subject to this difference, so that even a small false positive rate in a detection scheme can yield larger absolute numbers of false positives than true positives. Additionally, the efficiency and practicality of deploying a given detection scheme should be taken into account. General-purpose Deep-Packet Inspection (DPI) techniques, for example, are easier for an adversary to use on a wide-scale than specialized detectors that require storing and analyzing entire traffic captures.

2.4 System Overview

The essence of Rook's scheme for secretly sending data is to identify portions of game packets that can contain many different values and then infrequently select a packet and replace some of that mutable data with other legitimate game values representing the secret data to be communicated. The other side knows which packets have been selected and reads the secret data from them. There are security risks to this approach, the following lays out how our design addresses these.

2.5 Mutable Fields

Rook relies on finding *mutable fields* within game packets. In theory, we could replace any arbitrary bits in the payload of the packet with the secret data, because the packet will never be sent to the actual game process on the other end. However, many bit values are immutable with respect to the protocol of the particular game being used. If these were changed, then the packet would no longer be a valid game packet and an attacker could trivially detect Rook use by intercepting and attempting to parse these altered packets.

Therefore, the implementation of Rook must correctly implement at least part of the game network protocol in order to be able to parse packets and correctly identify which bits are part of *mutable fields*. These are the only bits which are modified when Rook sends data.

2.6 Symbol Tables

Unfortunately even if the *mutable fields* of a game packet can be correctly determined, not all combinations of these bits are necessarily valid or reasonable to exist in normal game traffic.

For example, imagine a *mutable field* sent from the server to the client representing the velocity vector of another avatar in the game. That value might be encoded as a 16-bit float. However, it might be that the velocity of an avatar is never actually greater than 100.0. If Rook replaced all 16-bits with arbitrary data it could easily be spotted by an adversary parsing packets and looking for velocity vectors with values greater than 100.0.

To prevent this type of attack, Rook does not insert the raw bits of the data it is sending. Instead, Rook keeps a *symbol table* for each *mutable field*. This *symbol table* is constructed by observing normal gameplay at the start of a game connection. For each *mutable field* encountered in the observed data a count of what values it had is kept, and then converted into a frequency. The *symbol table* for that *mutable field* is generated by pruning values whose frequencies are more than two orders of magnitude less than median frequency and then removing less frequent values until the list is a power of two in length; that list is the *symbol table* for that *mutable field*. To our knowledge, using existing data from cover traffic to dynamically generate a symbol table is a unique technique for a circumvention system. We believe similar approaches have potential for improving other circumvention systems based on steganographic approaches.

When Rook sends data by altering a *mutable field*, it converts n-bits of secret data into a symbol using the *symbol table* for that *mutable field*. By using a *symbol table* generated from normal game traffic Rook avoids sending values that are never or very infrequently seen and so prevents an adversary from filtering traffic based on these.

When the altered packet is received by the other Rook user, the process is reversed to translate from symbols in *mutable fields* to secret data bits, using the same *symbol table* in reverse. Because of this, it is required that both sides construct the same *symbol table* at the start of the connection by observing the same packets.

2.7 Scheduling Altered Packets

Rook only alters roughly one-in-ten packets, both to preserve game appearance and help reduce statistical anomalies that might arise in packet payloads.

Since the values inserted by Rook are only legitimate game values, Rook cannot use a flag indicating an altered packet, because any flag value is either too short to not occur by chance or creates too much overhead for the low bandwidth available. Instead, the Rook sender and Rook receiver preschedule which packets will be altered.

This schedule is arranged by synchronizing on an initial flag value. The flag value is derived from the shared secret key to prevent an attacker easily enumerating all Rook servers or passively detecting Rook connection setups. The sender waits for a packet with enough *mutable fields* to store the flag, default of 40-bits in length. The sender then sends the flag in that packet, and initializes its sending schedule using this packet as the start. The receiver scans for flag values in incoming packets and then initializes its receiving schedule starting with the packet with the flag value. Both sender and receiver then have schedules synchronized on the same starting packet.

In our implementation, the server performs this receiver-side scanning for all clients for the first 5 minutes of their connection and then assumes they are not potential Rook clients until they reconnect again.

2.8 Shared Deterministic Cryptographic Random Number Generator (DCRNG)

The schedule for which packets to alter cannot just be a constant regular interval because an adversary could poten-

tially isolate a set of only altered packets and perform statistical comparisons of just altered versus normal packets. To avoid this, the sender and receiver use a shared deterministic cryptographic random number generator (DCRNG) with a seed derived from their shared secret key. This DCRNG is used to produce a set of sequence numbers to use as the shared schedule for which packets to alter.

2.9 Shared Keys

As the previous two sections show, the Rook client and server need to share a secret key from which they derive the expected initial flags and DCRNG seeds. This key exchange must be performed out-of-band once before the first Rook connection is made. The client and server can then use a Diffie-Hellman key exchange to generate a shared key for the next connection.

2.10 Message-Present Bits

Since packets are altered according to a schedule, the receiver assumes a packet is altered even if the sender had no data to send. To avoid forcing the sender to always send data a message-present bit is used. This bit is essentially just prepended to the start of the secret data. If there is data to send, it is set to 1, otherwise it is set to 0. When the receiver extracts data from the altered packet, it checks the message-present bit: if the bit is a 1, it continues and extracts the rest of the secret data; otherwise it stops extracting data since the sender had nothing to send.

2.11 Packet Loss Resilience

Since Rook utilizes the game's UDP packets as its channel, it must provide its own reliability. It leverages the existing sequence-numbering and acking (present in all the game protocols observed) to detect if an altered game packet was dropped. In this case, the Rook receiver will simply not increment its sequence-number-ack value and the original sender will retransmit, as in normal TCP.

The DCRNG is kept synchronized despite dropped packets by generating a fixed number of random numbers is generated per-packet (for use in selecting the *mutable fields*). This way, if a receiver sees a packet was dropped, it can still run the DCRNG the correct number of times as if it had received the packet, and so still be synchronized when the next altered packet arrives.

3. IMPLEMENTATION

The preceding section described the design of the Rook system, which could be applied to any games with certain network characteristics. To study and evaluate the effectiveness of this system we implemented it for the game Team Fortress 2. Because Team Fortress 2 is built on the Source Engine, it shares the same network stack as other popular Source Engine games including Counter Strike: Global Offensive, and Day of Defeat: Source. Therefore, our implementation can also function for these games. All the testing and evaluation was done using Team Fortress 2.

The main effort to implementing Rook for a given game is reconstructing the packet format for the game. Both sides of the Rook connection must be able to parse the packet enough to correctly identify bits that can be altered without causing the packet to provoke errors from the real game packet parser. In practice this means the Rook code needs to be able to successfully parse the most common types of packets and correctly identify any others to be safely ignored. In our case this was a manual effort, however improvements in automated protocol analysis may reduce or remove this burden [30]. Furthermore, increased use of licensed game engines may also reduce the variety of protocols being used for games as a whole, thus increasing the number of games a new implementation provides access to.

4. EVALUATION

4.1 Bandwidth and Usability

Our implementation of Rook for Team Fortress 2 currently operates at 34 bits/second from game client to game server, and 26 bits/second from game server to game client. This is relatively low but still useful for the real-time chat messaging that is the target of this system. As part of our evaluation, we also incorporated an open-source implementation of the Off-The-Record [2] chat protocol to communicate with other Rook clients connected to the same server. The main overhead of the OTR protocol is in the initial key exchange, which can take several minutes in the current Rook implementation. However, after the initial connection is made the secure messages between clients are not significantly slower than unencrypted Rook messages.

Rook use also did not significantly impact the gameplay of the user or trigger any warnings from the built-in Valve Anti-Cheat system. We believe a server of Rook users and non-users could play together without any obvious differences noticeable between the two. However, Rook is not designed to protect against a human attacker individually targeting a specific Rook server. All FPS games studied also allowed password-protected servers, which could prevent an attacker from being able to join and observe a Rook server.

4.2 Censorship Resistance

To evaluate the censorship resistance of Rook we classify the types of attacks we consider into five types:

1. Anti-Mimicry
2. Single-Packet Deep-Packet-Inspection
3. Traffic Shape Analysis
4. Statistical Multipacket Deep-Packet-Inspection
5. Game-Specific n-gram Analysis

Anti-Mimicry.

Anti-mimicry attacks are defined as attempts to probe and identify Rook servers or clients based on comparing their response to probes to the response of a normal game server or client. It has been previously shown [4, 10] that many anti-surveillance and anti-censorship systems are vulnerable to these types of attacks. Rook is not vulnerable to these types of attack because it is not mimicking the game client and server but actually running them on both ends.

Altering packets could be used to create a denial of service attack against Rook. The attacker could replace mutable field values with other values to corrupt the data Rook receives. The attacker would need to do this to all game packets since they do not know which packets are scheduled to be read by Rook. Thus the attacker would still impact legitimate players by randomly changing their commands and server updates. Randomly dropping packets could also be used as a denial of service attack. The attacker would need

to drop few enough to not degrade play experience for legitimate players while also dropping enough to consistently drop a Rook-altered packet by chance and disrupt the messages. Rook could potentially respond to this type of attack by sending redundant messages although this would further lower its bandwidth.

Stateless Deep-Packet-Inspection.

Stateless Deep-Packet-Inspection (DPI) is what many censors appear to currently use if they do anything more advanced than IP or port blocking [32]. Stateless DPI is not effective against Rook because all packets altered by Rook adhere to the packet specification for the game and only use previously observed game values in the alterations. Therefore, if a stateless DPI system detected a Rook altered packet as malicious, it would necessarily have to also falsely detect many game packets of legitimate players as malicious.

4.3 Statistics-Based Attacks

While more costly to deploy, and hence less likely, we now consider attempting to detect Rook using multipacket statistical analyses to compare normal game traffic to game traffic altered by the use of Rook.

Data Gathering Methodology.

To evaluate the traffic shape analysis and the statistical DPI attacks discussed below, we created datasets as follows: a TF2 server with 20 bots is run on one machine on the LAN. This machine also runs the Rook server in the datasets using Rook. A second machine on the LAN runs a TF2 client, connects to the server, and runs the Rook client.

There are no universal standards for evaluating circumvention systems, owing in part to the diversity of their methods. We showed above how Rook is secure against past methods of attack on some circumvention systems. However, there is a large space of possible statistical attacks to try. To show the efficacy of our attacks, we also gathered samples a high-bandwidth (HB) configuration of Rook. It functions in the exact same manner as Rook, except it replaces every 1-in-2 packets, while normal Rook replaces approximately every 1-in-10. This allows us to demonstrate a particular statistical attack is capable of detecting systems like Rook, even if it cannot detect Rook being run in its typical configuration.

For datasets using Rook the user connects to the TF2 server, both the Rook server and Rook clients observe 600 packets (approximately 60 seconds of gameplay) in each direction and then create their symbol tables. The Rook client then connects to the Rook server and each side sends pseudo-random data at the maximum data-rate. The packets sent and received after the Rook connection is made are captured using Wireshark for analysis as described below. Each capture is approximately 7,000 client-to-server packets, and 6,000 server-to-client packets (about 5 minutes of gameplay/actual Rook use).

We gathered 61 datasets: 20 using Rook in HB configuration; 20 in the typical configuration; 20 of normal gameplay; and one of normal gameplay that was used as a baseline for investigating the three datasets listed above. The baseline is used to represent a known-normal dataset an attacker could use to try to differentiate normal and Rook traffic by comparing them to the baseline.

During the course of experimental setup, we observed that

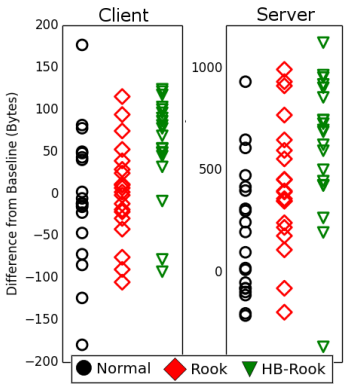

Figure 2: The difference in bandwidth consumed from the baseline traffic. HB-Rook appears to average slightly higher than normal.

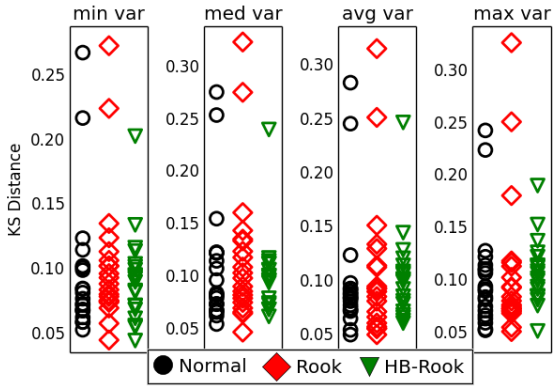

Figure 3: Results of the KS test performed on the distributions of variance across byte positions in client packets of the same size. There are a few outliers in all three categories, but no clearly distinguishing features.

many unexpected factors can affect the values sent in the game packet payloads; these appear to include: operating system, graphics hardware, and game window resolution. To give our attacks the best reasonable environment, we attempted to hold as many variables in the experiment stable as possible. In addition to using the same number of bots in all cases, the game player was also the same and already had experience in the game, the class played was the same, the game level played on was the same, the level was restarted before each sample, and data was continually sent at the maximum bandwidth. These essentially reflect the best possible circumstances an adversary could be expected to capture traffic under, as there should be as little variance as possible between samples and the covert channel's bandwidth use is maximized.

The above efforts to control the environment for more consistent analysis contributed to the dataset size, since the generation could not be automated or parallelized. However, this size appears reasonable for evaluation since most of the samples show strong similarities with only a few outliers.

Traffic Shape Analysis.

Traffic shape analysis is conducting statistical analyses on the size and timing features of the traffic between the client

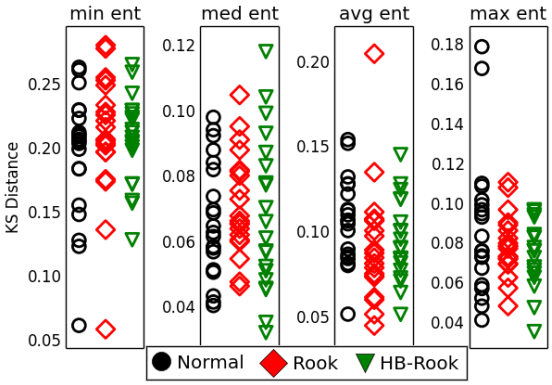

Figure 4: Results of the KS test performed on the distributions of entropy computed across each packet for client packets of the same size. No feature where Rook or HB-Rook are clearly distinguishable from all normal samples.

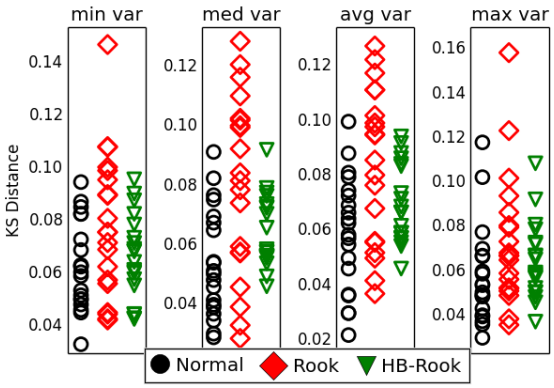

Figure 5: Results of the KS test performed on the distributions of variance across byte positions in server packets of the same size. Slightly lower average distances for the Rook traffic.

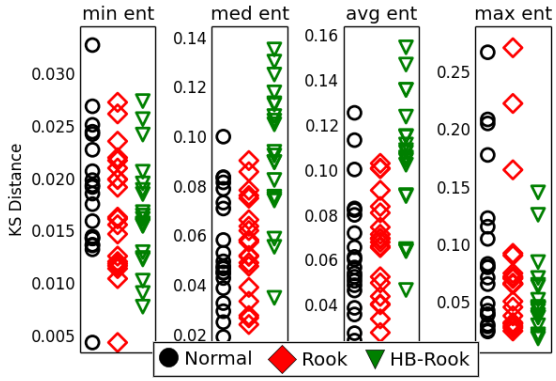

Figure 6: Results of the KS test performed on the distributions of entropy computed across each packet for server packets of the same size.

and server. Some previous covert channels have used timing changes to send secret information, and some have simply injected extra bytes into application packets, e.g., [5, 17]. These approaches have the potential to be detected by comparing statistics between known normal traffic and suspicious traffic, e.g., average size of packets or median timing between packets [12].

Rook should not be vulnerable to these approaches because it does not alter the timing or length of game packets to embed its information, it only alters individual data-fields within the packet. Furthermore, using Rook does not cause any additional packets to be sent, or packets to be changed in size, by the game server or client, so the traffic shape should be unaffected.

To evaluate this attack we extracted the overall bandwidth and spacings of packets and compared normal traffic to Rook use using the 2-sample Kolmogorov-Smirnov (KS-2S) test. We chose this test for its simplicity and use in previous system evaluations [9, 11, 12, 15]. The results show (see Fig. 2) are that both typical Rook and HB-Rook use is difficult to distinguish from normal game traffic. We are not certain of the cause of the slightly higher average bandwidth for HB-Rook: it may be due to Rook causing more packets with significant updates to be sent, reducing the effectiveness of the delta compression the game uses.

Statistical Multipacket Deep-Packet-Inspection.

Statistical multipacket DPI is the monitor taking a packet capture of all traffic between the client and server and running statistical tests across the payloads to compare these results to those obtained from doing the same process to traffic from a known normal game. There are many possible tests one could do, and no standard for using this kind of approach to evaluate a channel of Rook's kind. As a starting point, we adapt methods used in the related area of covert timing channels to test Rook's detectability. In these systems the timing of packets is the information channel and therefore statistical tests are run on the distributions of packet timings (similarly to the traffic shape analysis above) [9]. The analogous channel in Rook is the packet payloads, so we run the same tests on statistics computed over the payloads to detect anomalies.

Since only previously observed values are used by Rook, the only potentially detectable difference between Rook and normal payloads is differences in the distributions of these values. Therefore, we measure the variance and entropy across each data-field in Rook versus normal traffic. We measure the variance by first grouping packets by size and then computing the variance across all these packets at the same byte position. This process is repeated for each byte position, and for each size of packet captured. Entropy was measured by computing the entropy of each packet for all packets of the same size. The minimum, median, average, and maximum of each of these two statistics for each packet size was extracted to form eight distributions to test. The distributions of all of these results were compared to the same distributions derived from the baseline game traffic, and compared using the KS-2S test.

The resulting graphs can be seen in Figs. 3, 4, 5, and 6. These are each the stated statistic gathered from the sample traffic compared to the same statistic gathered from a baseline normal traffic sample using the KS-2S test. These show relatively little difference between the three categories. In Fig. 5 there is a slightly lower average distance for the normal Rook traffic; this is somewhat odd since the lower KS distance means the Rook samples are more similar to the baseline normal sample than the other normal samples are.

Figure 7: Sample of counts of distinct unigrams for one client field and one server field.

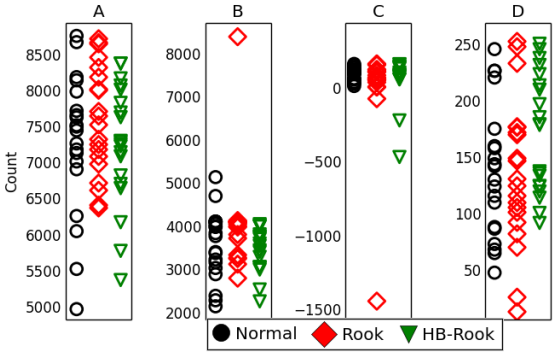

Figure 8: Adjusted counts of distinct trigrams from client traffic. A few outliers for both HB-Rook and Rook.

Game-Specific n-gram Analysis.

The final analysis we conducted was a game-specific n-gram analysis: this is an analysis of the values observed in the mutable fields of our implementation of Rook for Team Fortress 2. Using the same packet parsing module, we constructed lists of unigrams, bigrams, and trigrams for each mutable field for both the client and server. This analysis is similar to the analysis of variable-bit-rate encoding in VoIP used to distinguish languages spoken without decrypting the data-stream [33]. It is important to note that for any of the following attacks, the censor must have a game-specific implementation of data gathering and analysis tools, not general-purpose statistics like those used above. This would require an adversary to build a parsing engine and potentially manually generate a list of the useful features for detecting a specific game. Further, for the actual use the adversary must do a full capture of the traffic and, if blocking is desired, parse and analyze it in relative real-time.

In the following section we show and discuss the results of analyzing solely the unigram and trigram data. Analysis of the bigram data showed the same results as trigram data and is excluded for brevity. The labels A, B, C, and D. on the graphs are references to different mutable fields Rook uses. They represent actual game information fields, but are relabeled for simplicity.

Count of Distinct Trigrams The first analysis we performed was a comparison of the number of distinct trigrams in each sample. An n-gram is distinct if Rook only uses values for mutable fields that have been observed in normal

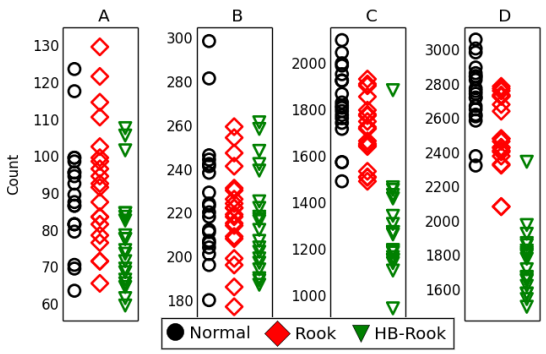

Figure 9: Adjusted counts of distinct trigrams from server traffic. HB-Rook is clearly distinguished over several fields from both normal and Rook samples.

game data, so the number of distinct unigrams is never increased by running Rook. The count of unigrams in both normal and Rook samples can vary, which leads to correspondingly amplified variation in trigram counts (see Fig. 7). To help correct for this, we computed an approximation of the function of unigrams-to-trigrams and used this function to baseline each trigram count sample before analyzing. The results in Figs. 8, 9 show that there are a few outliers, but most of the normal samples and typical Rook samples are very similar. The HB-Rook samples, however, are clearly distinguishable in the server trigram counts. We believe this is because the server will be replacing so many mutable fields that would normally contain new values with previously seen values stored in its symbol table that the total number of distinct trigrams is significantly reduced. A detector could potentially be developed to detect the normal Rook traffic in the same way, but it would either have a very high false-negative or false-positive rate. Additionally, we provide mitigating solutions to this attack (if it proved practical) in Section 6.

Frequency Distribution We also performed a comparison of the frequency distribution of trigrams. If Rook causes unlikely values to occur more frequently then there could be a shift to a more uniform frequency distribution versus the normal traffic. Figs. 10 and 11 show the results of a KS-2S test on frequency distribution across different mutable fields. For most fields the KS-distances are intermixed, showing poor distinguishability. In a few cases Rook or HB-Rook appear distinguishable by a smaller KS-distance than the normal captures, indicating they are more similar to the baseline-normal traffic. We are not able to explain this difference since the normal samples and the baseline sample were created in the exact same way. Given the small differences in KS-distance this could just be a statistical anomaly arising from our sample sizes.

Single-Frame Anomalies In the case of client-commands, the client could send repeats of the same command as a result of normal gameplay (for example, holding the forward button to continue moving forward). If such a repeating field was a mutable field in our Rook implementation then Rook could inject a different value into the field for a single-frame in the middle of a run of repeating values. This would create what we call a single-frame anomaly, where the first and third values of a trigram are the same value, but the second is different. Our analysis shows no consistent difference between normal gameplay and Rook

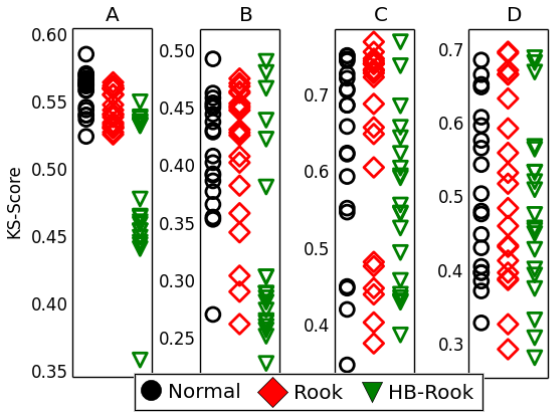

Figure 10: Results of the KS test performed on the frequency distributions of client trigrams. One field where HB-Rook is fairly distinguishable but due to being more similar to the baseline sample than normal or Rook samples.

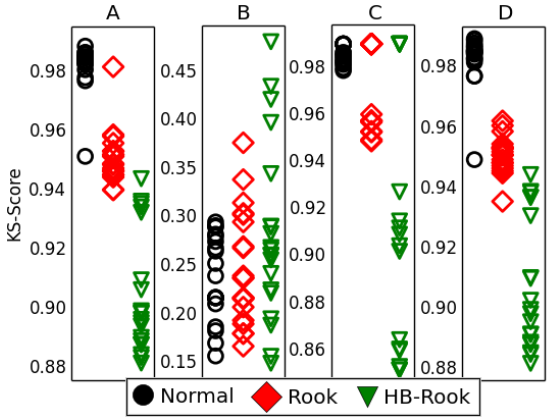

Figure 11: Results of the KS test performed on the frequency distributions of server trigrams. Two fields where Rook is distinguishable as being more similar to the baseline than either normal or HB-Rook samples.

use (see Fig. 12). We do not show this analysis for the server's packets because there are no repeating sequences.

In summary, we evaluated the security of Rook against five different major attack vectors. It is robust against standard anti-mimicry, DPI, traffic shape analysis, and generalized statistical attacks. Further, it is resilient against game-specific statistical attacks but may be vulnerable to a focused adversary devoting resources to detecting use of Rook on a specific game leveraging full packet captures. We believe this represents an improvement in the state-of-the-art in this field and still meets a high level of security since the resources to mount an attack are at the upper bound of our threat model, requiring multipacket statistical analysis using game-specific knowledge.

5. RELATED WORK

There have been many different approaches taken to enabling censorship circumvention or surveillance avoidance in the past. These have ranged from manipulating traffic shape attributes such as packet timing and size. Recently

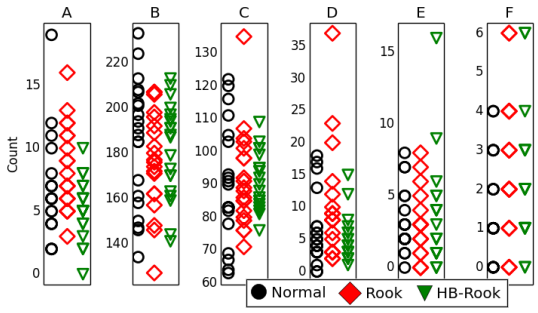

Figure 12: Count of Single-Frame Anomalies in client samples. Showing a few outliers from Rook and HB-Rook but overall not very distinguishable from normal samples.

there have been many systems developed which try to slip past monitors by disguising themselves as normal traffic [11, 18, 24, 31, 35].

Timing Steganography.

There have been several systems based on using packet timings to covertly communicate [5, 9]. These systems are potentially harder to detect than Rook because the packet timings they are modifying are already impacted by what other processes are running on the machine, which is outside of the attack scope. However, an active adversary can potentially block the channel by adjusting the timing of packets slightly to destroy the information, or drastically reduce the bandwidth, without necessarily impacting the legitimate application use. Despite these drawbacks, a timing-based covert channel like CoCo could actually work excellently alongside Rook. The bandwidth gain would be modest, about 5-10 bits/second, but the two systems operate orthogonally to one another and so the risk of detection would only be whichever system is the most detectable.

Header Value Steganography.

Many fields in standard network protocol headers (TCP, UDP, RTP, etc.) have been found to be useful for steganographic purposes [16, 20, 22]. These include the least-significant-bits of the timestamp, padding values, initial sequence numbers, and various flags. These covert channels have an advantage in that they are ubiquitous to all applications with any standard type of network traffic and so can easily bypass any application filtering an adversary could put in place. However, attacks against the covertness of several of these channels have been shown [25, 26].

Furthermore, these schemes are based on the application ignoring whether these fields are set or not, so an adversary can simply normalize them to deny the covert channel. Rook modifies data that is used by the application, so this type of attack degrades legitimate use. As we argue in Section 2, adversaries are disincentivized from impacting the activities of normal users.

Application Protocol Mimicry.

There have been several recent systems based on transforming the appearance of traffic to evade censorship [21, 29, 31]. These types of approaches generally suffer from weakness to active probing: if either end is not actually running the application the altered traffic would be pro-

duced by, probes from adversaries will be ignored. As shown by Houmansadr et al., it is very difficult to try to accurately mimic how a real application will respond to an arbitrary probe, particularly those which would cause errors [10]. Since Rook actually runs the game client and server, an active adversary's probes will be responded to in the same manner as a normal game client and server. Depending on the application being mimicked, an adversary could also attack these systems by parsing the packets to detect any nonconformity to the protocol specification or usual behavior.

Application Subversion.

Since these attacks on mimicry-based systems have been published, there have been several new systems proposed which hide data at the application-layer, rather than inserting it at the transport layer [7, 8, 11, 18, 23, 24, 35]. These approaches are the most similar to Rook. Two previous systems have used steganography to communicate in computerized boardgames [8, 23]. Castle [7] (coincidentally named) also uses online games, but of a different type, to hide its data. It also uses automated in-game commands to generate specific game actions the other game client interprets as secret data. This has the advantage of being higher bandwidth than Rook, but also have detectability trade-offs associated with it because it potentially requires the game's default transportation security to remain intact for Castle to avoid detection.

Unlike Rook, all of these approaches rely on an encrypted channel between the ends of the application to be unbroken. Current events show that some censors will force man-in-the-middle attacks or subverted versions of programs to be used to allow breaking this encryption [3, 14]. Rook does not require its applications traffic to be encrypted to remain secure.

6. DISCUSSION AND FUTURE WORK

Rook is a new approach to an established problem of censorship resistant communication. We argue that online games provide an excellent form of cover for secret communication and have enough mass appeal that a censor would be reluctant to outright block their traffic. The evaluation of our implementation of Rook for Team Fortress 2 shows it to be robust to all currently known forms of network interference and censorship. Further, the evaluation shows Rook is resistant to potential new forms of censorship, such as deep-packet statistical analyses and application-specific analyses, that may become common tools for censors in the future.

Rook also aims itself at somewhat under-represented facets of censorship resistance: establishing safe communication entirely within a censor's region of control, and emphasizing keeping plausible deniability for its users. The current implementation of Rook is functional in exchanging message undetected; however, we believe it could be expanded upon in the following ways:

The first and most obvious extensions from a utilitarian perspective is the implementation of Rook for more games. We developed the Rook code in a modular fashion so that new modules for interpreting different game packet protocols can be easily added. In addition to many other First-Person Shooter games, preliminary research shows Rook could be applied to many other types of online games such as: Real-Time Strategy (RTS) games like Starcraft and Starcraft 2; Multiplayer Online Battle Arenas (MOBA) like Dota 2 and League of Legends; and even Massively-Multiplayer Online (MMO) games like World of Warcraft. For some of these unsanctioned private servers would have to be used, but these already have significant followings among normal players as well [19].

From a system design perspective, a secure bootstrapping mechanism for finding and contacting Rook servers would be a significant addition to the usability of the system, and could potentially also be conducted over the same online game, but perhaps with a higher-latency mechanism. However, this is a major challenge in circumvention systems in general that has not yet been solved.

Another improvement that could be made to Rook is making the symbol tables dynamically self-adjusting to attempt to better preserve the traffic statistics. Essentially each side would monitor how the packets it altered have impacted a set of statistics and then modify their symbol tables to try to minimize the statistical deviance created by hiding data. This would inevitably reduce bandwidth to some extent; however, if statistical attacks are found to be a non-negligible threat to Rook, the tradeoff would be worthwhile to defeat such attacks. In general, we believe Rook can be advanced to mitigate higher-order statistical attacks by implementing these same statistical models to adjust the way it alters packets and thus remain undetectable.

7. CONCLUSION

In this paper we presented Rook, a system designed to provide low bandwidth low latency censorship resistant communication using the network traffic of online games. Rook, along with the concurrently-researched system Castle [7], represent the first censorship circumvention systems to use online games as a cover for secret communication. Rook and Castle take different approaches, and seek to accomplish different goals, and hence represent complementary investigations into this new cover medium. Beyond its novelty, Rook also represents a useful addition to the space of existing circumvention techniques and systems. Unlike many other systems, Rook focuses on providing secret communication within a censor's region of control, and presents greater secrecy and deniability than previous systems by virtue both of how its communication is hidden and by it being hidden in online game traffic instead of other applications that would show more differences between legitimate users and censorship circumventing users.

Our implementation of Rook shows that it lives up to its goal of providing bandwidth high enough for chat, including over the OTR protocol, while remaining undetectable using any mass attack methods currently known to be employed by censorship regimes. Furthermore, Rook presents a fundamentally greater challenge to detect than what most current circumvention systems present. An adversary would have to commit resources both to develop an attack against a particular implementation of Rook, and allocate computational resources to each game connection they find suspicious in order to defeat Rook. Even under targeted attack, we argue novel attack techniques would need to be developed to detect Rook communications.

We intend to release our code along with developer documentation.

8. ACKNOWLEDGEMENTS

We would like to thank Rob Johnson for a helpful exchange of ideas on use of games in censorship circumvention. This work was supported in part by NSF Award CNS-0846065 and by the Short-Dooley Endowed Career Development Professorship.

9. REFERENCES

[1] R. J. Anderson and F. A. Petitcolas. On the limits of steganography. *Selected Areas in Communications, IEEE Journal on*, 16(4):474–481, 1998.

[2] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM, 2004.

[3] J. Crandall, M. Crete-Nishihata, J. Knockel, S. McKune, A. Senft, D. Tseng, and G. Wiseman. Chat program censorship and surveillance in china: Tracking tom-skype and sina uc. *First Monday*, 18(7), 2013.

[4] J. Geddes, M. Schuchard, and N. Hopper. Cover your acks: pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 361–372. ACM, 2013.

[5] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts. Covert messaging through tcp timestamps. In *Privacy Enhancing Technologies*, pages 194–208. Springer, 2003.

[6] G. Greenwald, J. Ball, and D. Rushe. Nsa prism program taps in to user data of apple, google and others. *The Guardian*, June 2013.

[7] B. Hahn, R. Nithyanand, P. Gill, and R. Johnson. Games without frontiers: Investigating video games as a covert channel. *arXiv preprint arXiv:1503.05904*, 2015.

[8] J. C. Hernandez-Castro, I. Blasco-Lopez, J. M. Estevez-Tapiador, and A. Ribagorda-Garnacho. Steganography in games: A general methodology and its application to the game of go. *computers & security*, 25(1):64–71, 2006.

[9] A. Houmansadr and N. Borisov. Coco: coding-based covert timing channels for network flows. In *Information Hiding*, pages 314–328. Springer, 2011.

[10] A. Houmansadr, C. Brubaker, and V. Shmatikov. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 65–79. IEEE, 2013.

[11] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer. I want my voice to be heard: Ip over voice-over-ip for unobservable censorship circumvention. In *The 20th Annual Network and Distributed System Security Symposium (NDSS)*, 2013.

[12] S. Khattak, L. Simon, and S. J. Murdoch. Systemization of pluggable transports for censorship resistance. *arXiv preprint arXiv:1412.7448*, 2014.

[13] J. Killock. Sleepwalking into censorship. *Open Rights Group*, July 2013.

[14] J. Knockel, J. R. Crandall, and J. Saia. Three researchers, five conjectures: An empirical analysis of tom-skype censorship and surveillance. In *FOCI'11: USENIX Workshop on Free and Open Communications on the Internet*, 2011.

[15] S. Li, M. Schliep, and N. Hopper. Facet: Streaming over videoconferencing for censorship circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 163–172. ACM, 2014.

[16] D. Llamas, C. Allison, and A. Miller. Covert channels in internet protocols: A survey. In *Proceedings of the 6th Annual Postgraduate Symposium about the Convergence of Telecommunications, Networking and Broadcasting, PGNET*, volume 2005, 2005.

[17] N. B. Lucena, J. Pease, P. Yadollahpour, and S. J. Chapin. Syntax and semantics-preserving application-layer protocol steganography. In *Information Hiding*, pages 164–179. Springer, 2005.

[18] J. Lv, T. Zhang, Z. Li, and X. Cheng. Pacom: Parasitic anonymous communication in the bittorrent network. *Computer Networks*, 2014.

[19] Marie. World of warcraft (wow) private servers guide.

[20] W. Mazurczyk and K. Szczypiorski. Steganography of voip streams. In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 1001–1018. Springer, 2008.

[21] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108. ACM, 2012.

[22] S. J. Murdoch and S. Lewis. Embedding covert channels into tcp/ip. In *Information Hiding*, pages 247–261. Springer, 2005.

[23] S. J. Murdoch and P. Zieliński. Covert channels for collusion in online computer games. In *Information Hiding*, pages 355–369. Springer, 2005.

[24] B. Ragnarsson and P. Westein. Using git to circumvent censorship of access to the tor network. 2013.

[25] T. Sohn, J. Moon, S. Lee, D. H. Lee, and J. Lim. Covert channel detection in the icmp payload using support vector machine. In *Computer and Information Sciences-ISCIS 2003*, pages 828–835. Springer, 2003.

[26] T. Sohn, J. Seo, and J. Moon. A study on the covert channel detection of tcp/ip header using support vector machine. In *Information and Communications Security*, pages 313–324. Springer, 2003.

[27] Valve. Steam and game stats.

[28] Valve. Team fortress 2.

[29] Q. Wang, X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov. Censorspoofer: asymmetric communication using ip spoofing for censorship-resistant web browsing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 121–132. ACM, 2012.

[30] Y. Wang, X. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo. A semantics aware approach to automated reverse engineering unknown protocols. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–10. IEEE, 2012.

[31] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 109–120. ACM, 2012.

[32] P. Winter and S. Lindskog. How china is blocking tor. *arXiv preprint arXiv:1204.0447*, 2012.

[33] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob? In *USENIX Security*, volume 3, page 3, 2007.

[34] X. Xu, Z. M. Mao, and J. A. Halderman. Internet censorship in china: Where does the filtering occur? In *Passive and Active Measurement*, pages 133–142. Springer, 2011.

[35] W. Zhou, A. Houmansadr, M. Caesar, and N. Borisov. Sweet: Serving the web by exploiting email tunnels. *HotPETS. Springer*, 2013.