Covert Channels in Multiplayer First Person Shooter Online Games

Sebastian Zander, Grenville Armitage, Philip Branch
Centre for Advanced Internet Architectures (CAIA)
Swinburne University of Technology
Melbourne Australia
{szander, garmitage, pbranch}@swin.edu.au

***Abstract*—Covert channels aim to hide the existence of communication between two or more parties. Such channels typically utilise pre-existing (overt) data transmissions to carry hidden messages. Internet-based covert channels often encode new information into unused (or loosely specified) IP packet header fields, or the time intervals between IP packet arrivals. We propose a novel covert channel embedded within the traffic of multiplayer, first person shooter online games. We encode covert bits as slight, yet continuous, variations of a player's character's movements. Movement information is propagated to all clients attached to a given game server, yet the channel remains covert so long as the variations are visually imperceptible to the human players. A modified version of Quake III Arena is used to demonstrate our concept. We empirically analyse the covert channel's bit rate, and compare the statistical characteristics of unmodified game traffic with those of game traffic carrying covert information.**

***Index Terms*—Security, Covert Channels, Online Games**

I. Introduction

Encryption alone is not sufficient to secure communication. Often the simple fact that communication exists is enough to raise suspicion and take further actions. Covert channels aim to hide the very existence of the communication [1]. Individuals and groups may have various reasons to utilise covert channels, often motivated by the existence of an adversarial relationship between two parties. Examples include government agencies versus criminal or terrorist organisations, hackers or corporate spies versus a company IT department, or dissenting citizens versus their governments. Usually covert channels use means of communication not normally intended to be used, making the job of identifying them quite elusive.

Many network protocol-based covert channels have been proposed, often exhibiting bit error and fading characteristics not-unlike low-power radio channels. These channels range from very simple (such as encoding covert information in unused header bits) to very complex (such as encoding covert information through temperature changes) [1]. Limited work exists on covert channels in networked games, primarily focused on board games [2], [3].

We propose a novel covert channel between players in multiplayer first person shooter (FPS) online games. Modern FPS games typically offer two internal communication channels – text chat and voice. However, text chat is usually logged and filtered at the server and the same can be easily done with any in-game voice communication. Our approach (which we shall refer to as FPSCC) hides the covert communication from server operators and unwitting players. Even players whose clients are endpoints of the covert channel may remain unaware of the covert information flow.

FPSCC hides covert information in minute, additional movements of player characters in the virtual world. Character movements intended by a human player are subject to slight variations that encode covert bits. We choose variations that will have no visible effect on the character's movements as perceived by other human players inside the game. A key advantage of this approach is that character movement is usually not logged at the server.

FPSCC has a number of desirable properties. FPS games are very common and their network traffic is not suspicious (although it may not be present in all circumstances). FPSCC is a broadcast channel – information flows from one covert sender to one or more covert receivers. FPSCC is an indirect channel – covert sender and covert receiver use the game server as an intermediary, rather than directly exchanging IP packets. Detection of the covert sender does not directly expose the identities of the covert receiver(s) (who could be any of the players online at the same time).

Simple covert channels in network protocols may often be easily eliminated, for example by protocol normalisation [1], [4]. FPSCC is more challenging to eliminate, because it is tied to player movement (an intrinsic function inside the games). We discuss various ways the channel could be detected or its capacity could be limited. Nevertheless, our work supports the opinion expressed in various quarters that covert channels are impossible to eliminate entirely, even in highly optimised network protocols [5].

Our proof-of-concept FPSCC implementation is based on the game Quake III Arena (Q3) [6]. We chose Q3 because it uses the common client-server communication architecture and the source code is freely available allowing us to understand the network protocol without resorting to reverse engineering. Implementation of FPSCC has enabled empirical analysis of the covert channel scheme. We report on bit rate, and compare statistical characteristics of unmodified game traffic with those of game traffic carrying covert information.

978-1-4244-2413-9/08/$25.00 ©2008 IEEE

Like many covert channels FPSCC is a noisy, low bit rate channel (we measured transmission rates of up to 7-9 bit/s). We see its main purpose in exchanging SMS-style text messages. On the other hand we find FPSCC is difficult to detect, as it looks very similar to 'normal' use of the protocol. Despite our initial focus, our technique is not limited to FPS games. FPSCC could be applied to any game where player movements of one or more in-game characters are regularly propagated to other players (and their game clients).

The paper is organised as follows. Section II reviews covert channels, FPS games, and previously proposed covert channels in games. Those parts of the Q3 game protocol relevant to FPSCC are described in Section III. Section IV presents a general technique for encoding covert information into player movement, describes how it is specifically achieved in Q3, discusses factors that introduce bit errors into the channel, and describes available countermeasures. Section V overviews the covert channel implementation. Section VI reports typical bit rates of the covert channel and investigates the 'fingerprint' of FPSCC based on a number of test games. The paper concludes in Section VII with a discussion of future work.

II. Background

A. Covert Channels Overview

This section reviews the concept of a covert channel (a detailed treatment can be found in [1]). Covert channels are often illustrated by the *prisoner problem* (Figure 1) introduced by Simmons [7]. Prisoners Alice and Bob must communicate to establish an escape plan, but Wendy (the warden) monitors all their messages. Any signs of suspicious messages will result in Wendy placing Alice and Bob into solitary confinement, preventing an escape. Alice and Bob must exchange innocuous messages (*overt* channel) containing hidden information (*covert* channel) that Wendy will (ideally) not notice.

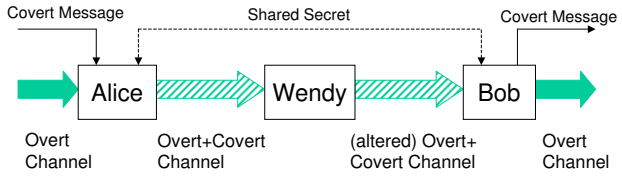

Figure 1. Prisoner problem – communication model for covert channels

Assume Alice and Bob use two networked computers to communicate. Their covert channel will be encoded in subtle variations of the network traffic flowing between their computers during normal, innocuous activity. Alice and Bob must also share a secret – knowledge of how the covert channel is encoded onto the overt channel, and any additional authentication/encryption applied to the covert channel messages.

In practice Alice and Bob may well be networked devices controlled by the same person. For example, a corporate spy might set up Alice and Bob on either sides of the company's network boundary to ex-filtrate restricted information. Wendy represents the network manager who can monitor the passing traffic for covert channels or alter the passing traffic to disrupt or eliminate covert channels.

Alice and Bob are not required to be the sender and receiver of the overt communication. One or both may be compromised network devices along the overt traffic's path (known as *middlemen*). It is sufficient that Alice can observe and manipulate overt traffic passing through, and Bob can observe the overt+covert traffic passing through. (If Bob can manipulate the traffic he may also chose to reverse the covert channel encoding, thus restoring the original overt channel.)

Covert channels suffer from errors, due to interactions between the covert encoding and the overt channel's own intrinsic traffic patterns: *substitutions* (bits decoded incorrectly), *deletions* (bits completely lost) and *insertions* (bits inserted).

B. First Person Shooter (FPS) Online Games

The publishing model for FPS games (such as Quake III Arena, Unreal Tournament and Counter-Strike Source) makes them intriguing for use as covert channels. FPS games are based on the client server architecture, and publishers typically release their server code free – relying on Internet Service Providers (ISPs), dedicated game hosting companies and individual enthusiasts to host FPS servers. FPS game servers typically host from less than 10 to around 30+ players and (particularly for popular games such as Counter-Strike Source) there may be tens of thousands of individually operated game servers active on the Internet at any given time [8], [9]. Alice and Bob thus have a wide variety of game servers through which to establish legitimate-looking overt traffic flows.

C. Covert Channels in Games

Murdoch *et al.* [2] illustrated covert channels for collusion in an online *connect-4* contest (where one human contestant could enter multiple programs as players). Murdoch *et al.* won the contest by deploying two types of colluding players: foxes and chickens. Foxes would play their best against other competitors. Chickens would deliberately lose against foxes and play their best against other competitors. Chickens used a covert channel (based on the redundancy in the moves of the game) to detect a fox.

Hernandez-Castro *et al.* [3] propose a framework for hiding data in games based on game theory. They also describe how covert information should be hidden in game strategies and explore possible countermeasures. They integrated the proposed covert channel into a Go program and empirically analysed the effectiveness of several detection strategies (steganalysis).

So far research focused on board games or their electronic versions (such as Chess or Go) with two players taking alternating turns. Both Murdoch *et al.* and Hernandez *et al.* propose encoding covert information into the moves or strategies played. Murdoch *et al.* also discusses how the timing of moves could be used to encode covert information. While these covert channels could be used for any purpose, collusion is the main objective.

Covert channels developed for turn-based board games cannot readily be applied to real-time FPS games. FPSCC

aims to provide a general-purpose communications channel, not merely co-ordinating collusion amongst players. Human players provide the overt channel (movements, driven by strategy and tactics). FPSCC provides a covert channel by adding imperceptible fluctuations to player movement signals. Players may be unaware of the covert channel's existence, nor aware of the traffic flowing over the covert channel.

III. Quake III Arena Player Movement Protocol

Q3 uses a client-server architecture and relies on UDP/IP packets to carry information between servers and clients. FPSCC utilises the network traffic that occurs regularly during game play, and ignores traffic associated with server-discovery and initial client connection.

Figure 2 illustrates the message flow during game play. *User commands* are sent from client to server once per graphics frame rendered (but no more than once every 10 ms). *Snapshots* are sent from server to client once every 50 ms (by default). Transmissions of user commands and snapshots are not synchronised.

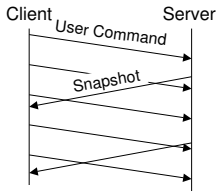

Figure 2. Messages exchanged between Q3 client and server

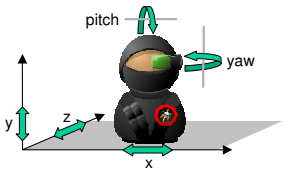

Figure 3. Player character movement

Figure 3 illustrates the player movements that may be indicated in each user command. Movement occurs along three axes (left/right, forward/backward, up/down) and change of view angle may be requested along two axes (yaw and pitch). The player cannot modify roll. A user command also indicates mouse button and keyboard button state (other than movement related) as well as the selected weapon. To compensate for packet loss, each UDP/IP packet sent by the client actually contains the current and previous user commands. Client messages also contain 'reliable' commands, such as the disconnect command.

Each client receives client-specific game world state updates in snapshots. A snapshot contains the server's authoritative belief about the state of the client's player (position, view angles and player-specific events) as well as the state of all other entities potentially visible to the client's player (positions, view angles and events). Entities can be other human player characters, computer-controlled characters (bots) or objects. Entity state updates are not sent for entities that the client's player cannot see; however, note that not all potentially visible entities are actually visible on the player's screen. This reduces network traffic and mitigates a source of potential client-side cheating (chapter 7, [9]). Server messages also contain 'reliable' commands (such as printing in-game messages on the client screen).

Q3 uses sequence numbers in both directions to detect loss of UDP/IP packets. If loss occurs, 'reliable' commands (that affect the overall game state) are retransmitted. Lost user commands and entity state updates are never retransmitted as they are continuously updated anyway. User commands sent by the client are timestamped, as are player and entity state updates sent by the server. Consequently every update of player state sent by the server can be unambiguously linked to a corresponding (previous) user command sent by a client.

All user commands and state updates are delta encoded to reduce packet size (so a data field is only sent if it has changed) and all messages are compressed using adaptive Huffman encoding [10]. Despite differences in specific details, most FPS game protocols utilise a similar overall design.

Figure 4 illustrates the relationship between player position information sent to the server, and the same information received by other clients. Let us define x_i to be client 1's player input for their character's position along an axis (or the view angle along one axis) in user command i and define y_j to be the position or view angle of client 1's character sent by the server to both clients in snapshot j. As user commands usually arrive more frequently than snapshots are emitted, each y_j is computed based on the most recently received x_i. Client 2 renders client 1's player on screen based on y_j until it receives y_{j+1} (a period indicated by the boxes).

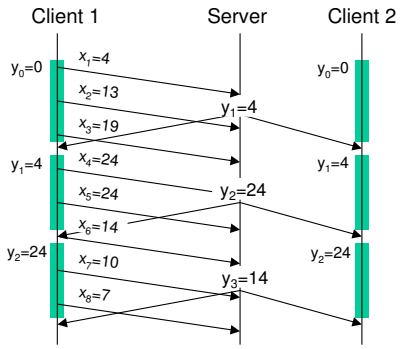

Figure 4. Example user input values and server snapshot values

IV. FPS Covert Channel (FPSCC)

FPSCC creates a covert channel between two Q3 game clients, where one client acts as Alice (sender) and the other acts as Bob (receiver). FPSCC aims to avoid detection by either the players controlling the game clients, or by an independent observer (Wendy). Referring back to Figure 4, FPSCC encodes covert information by modulating x_i from client 1 (Alice) with visually imperceptible fluctuations in character position or view angles. Client 2 (Bob) decodes the covert information from y_j updates arriving in successive snapshots. FPSCC is not limited to unidirectional communication. Bob could send covert information to Alice at the same time.

A. Encoding and Decoding

We leverage the fact that Q3 encodes more detail in x_i and y_j than can normally be resolved (visually) by the human

player. FPSCC modulates player view angle updates for pitch and yaw, as player view angles are (with small exceptions discussed later) almost entirely dictated by player input. (We chose not to use position information as a character's position may be perturbed by various 'forces' acting on the player's character inside the game's virtual world, making it very hard to predict y_j from x_i.)

Because x_i conveys relative view angles we use *changes* in view angle to encode covert information. To minimise detection, FPSCC only encodes covert information when the player is adjusting their character's view. If the player stops moving their view, the covert channel pauses. (The covert channel is effectively masked if FPSCC-induced changes are small compared to the player's own input). Our FPS game-play experience suggests that such a covert channel is unlikely to be noticed by average players in the heat of battle, and would not affect their ability to play.

The details of FPSCC encoding are as follows. Let the change in user input be $\Delta_i = x_i - x_{i-1}$. Alice encodes N bits of covert information with an integer value of b ($0 \leq b \leq 2^N - 1$) into each angle change so that

$$b = |\tilde{y}_j - \tilde{y}_{j-1}| \ mod\, 2^N, \tag{1}$$

where $\tilde{y}_j$ and $\tilde{y}_{j-1}$ are the angle values manipulated by Alice. However, Alice can only indirectly modify y_j by modifying the user input x_i.

As noted previously, the game server computes y_j from the most recently arrived x_i. Q3's asynchronous message transmissions, unpredictable client message rate and variable network delay make it impossible for Alice to predict which x_i will be used by the game server to compute y_j . Therefore, Alice has to encode the same covert bits in all x_i sent between the arrival of y_{j-1} and y_j.

When Alice detects an angle change ($\Delta_i \neq 0$), she starts encoding the next covert bits to be send b_n in the current and all following user commands. Each time a snapshot is received from the server, Alice checks whether the angle value has changed ($\tilde{y}_j \neq \tilde{y}_{j-1}$). If not, Alice continues sending b_n. Otherwise Alice assumes b_n has been successfully transmitted and updates the previous angle value $\tilde{y}_{j-1}$. The next user angle change will cause Alice to start sending bits b_{n+1} and so on.

Covert bits are encoded by modifying user inputs as follows. We define the user input modified by the covert sender to

$$\tilde{x}_i = x_i + \delta_i. \tag{2}$$

If $\Delta_i \neq 0$ Alice encodes b by selecting δ_i such that

$$b = |\tilde{x}_i - \tilde{y}_{j-1}| \ mod\, 2^N. \tag{3}$$

From equation 2 and equation 3 follows:

$$\delta_i = \begin{cases} b - (x_i - \tilde{y}_{j-1}) \ mod\, 2^N & x_i - \tilde{y}_{j-1} \geq 0 \\ -b - (x_i - \tilde{y}_{j-1}) \ mod\, 2^N & x_i - \tilde{y}_{j-1} < 0 \end{cases}. \tag{4}$$

However, Alice must avoid completely negating the angle change. If $(x_i - \tilde{y}_{j-1}) + \delta_i = 0$ she needs to modify δ_i:

$$\delta_i = \begin{cases} N + \delta_i & x_i - \tilde{y}_{j-1} \geq 0 \\ -N + \delta_i & x_i - \tilde{y}_{j-1} < 0 \end{cases}. \tag{5}$$

The next snapshot value $\tilde{y}_j$ will be based on one of the $\tilde{x}_i$ arrived at the server between snapshot $j-1$ and j, and possible noise on the channel n_j (see next sub section). Bob decodes the covert bit(s) similar to equation 3:

$$\hat{b} = |\tilde{y}_j - \tilde{y}_{j-1} + n_j| \ mod\, 2^N. \tag{6}$$

Figure 5 illustrates the encoding of covert bits in pitch, with $y_0 = 0$ and the same user input as in Figure 4. One bit of covert information is encoded per pitch change, meaning an even change signals a 0 bit and an odd change signals a 1 bit. The angles modified by Alice are shown in bold. The boxes indicate the time periods in which a covert bit is transmitted.

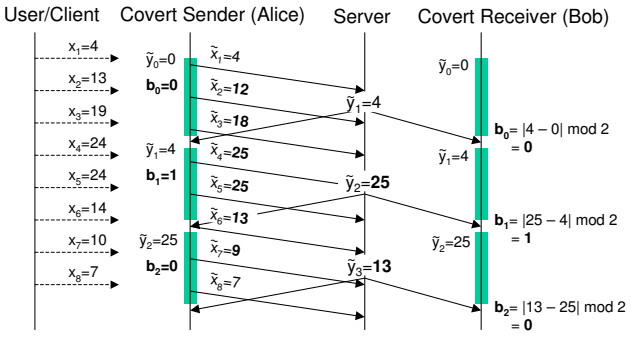

Figure 5. Example of covert channel encoding

B. Impact of Round Trip Time

Figure 5 assumes that the round trip time (RTT) between client and server, plus the time between two client messages, is smaller than the time between two server updates. If not, Alice never knows the actual value of $\tilde{y}_{j-1}$ and hence cannot compute the correct δ_i. Given that server updates usually occur every 50 ms, and client messages often every 10 ms, the covert channel is limited to situations where RTT$\leq$40 ms.

FPSCC can function over larger RTTs at the cost of reducing the covert bit rate. If u is the time between game server updates (50 ms), we now send b_n in m server updates intervals, where $m \geq 2$ and $m \cdot u \geq$RTT. The timestamps in server messages can be used to control which updates contain covert bits.

C. Number of Encodable Bits

Our simplest case sends one bit per angle change ($N = 1$). We might also select N based on the user input: the larger the user's change, the larger Alice's modification can be without being noticed. However, we also assume there will be an upper bound δ_{max}, which is the maximum modification Alice can make before the channel security is compromised (e.g. the channel affects game play or creates visible artefacts).

We define L to be the limit of how an angle can be modified:

$$\frac{max\left(\delta_i\left(b, N_i\right)\right)}{max\left(\delta_i\left(b, N_i\right)\right) + \Delta_i} \leq L. \tag{7}$$

Then the number of encoded bits N_i is chosen to:

$$N_i \leq min\left(\lfloor log_2\left(\delta_{max}+1\right)\rfloor,\ \left\lfloor log_2\left(\frac{1+L\cdot(\Delta_i-1)}{1-L}\right)\right\rfloor\right). \quad (8)$$

Figure 6 illustrates the number of bits encodable N over the user change Δ_i for different example values of δ_{max} and L. (For different δ_{max} the lines have been slightly offset for improved visibility.)

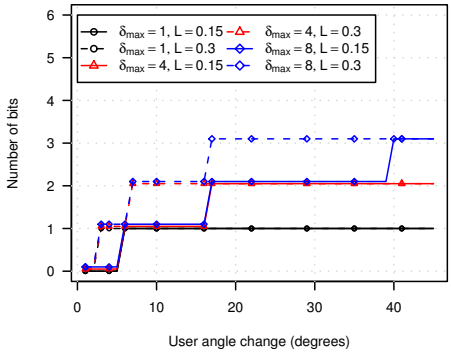

Figure 6. Number of bits encodable vs user change for different δ_{max}, L

FPSCC supports encoding covert bits in both pitch and yaw simultaneously. Alice encodes the bits in the order the user changes the angles between snapshots, for example if yaw changes before pitch, b_n will be encoded in yaw and b_{n+1} will be encoded in pitch. Bob cannot know which angle changed first and will always decode the bits in fixed order, the first bit from pitch b_n and the second bit from yaw b_{n+1}. To prevent FPSCC from swapping bits, Alice needs to swap both bits as soon as she becomes aware that yaw has changed before pitch.

D. Broadcast Versus Unicast Covert Channel

As previously noted, Bob only receives snapshots containing $\tilde{y}_j$ updates relating to Alice, if Alice's in-game character is potentially visible to Bob inside the game environment. Consequently, there are two transmission modes open to Alice – *unicast* and *broadcast*. If Alice knows the in-game identity of Bob's client then Alice can chose to only send covert data when Bob's character is known to be in range (determined from the server snapshots). This is unicast mode, and the covert channel may pause from time to time. However, if Alice has no idea on which in-game client Bob resides, Alice simply transmits covert data all the time. This is broadcast mode, where the covert channel may experience significant periods of lost bits.

Regardless of Alice's transmission mode, Bob is not required to know in advance on which game client Alice resides. Bob can simply attempt to decode covert information from $\tilde{y}_j$ updates relating to every player. Any stream of $\tilde{y}_j$ updates that generates meaningful covert information can be presumed to come from Alice. ('Meaningful' could mean the existence of pre-defined bit sequences as used in [2] or data structures previously agreed upon by Alice and Bob.) However, if Alice's client's identity (e.g. player name) is known, Bob can focus on decoding $\tilde{y}_j$ updates relating to that specific player.

FPSCC allows multiple instances of Bob, each associated with a different game client. Each Bob decodes part (or all) of the covert channel's data stream as their associated players cross paths with Alice inside the game's virtual world.

E. Noise

In broadcast mode there are deletions (bits lost when Bob is out of range of Alice), but no insertions (as Bob does not decode bits when Alice is not visible). In unicast mode both deletions (if in a snapshot Bob is visible to Alice, but Alice is not visible to Bob) and insertions (if Alice is visible to Bob, but Bob is not visible to Alice) can occur (potential visibility is asymmetric in Q3).

In Q3 view angles of player characters mostly depend on user input only. However, there can be 'unexpected' view angle changes, such as players respawning after having died, players teleporting or players standing on moving objects. Alice takes these server-side changes into account as soon as she becomes aware of them from the player state updates. However, some bit errors may occur, with frequency dependent on the game play and the server settings (for example, moving objects or teleporters are not present in all Q3 maps).

The Q3 server clamps pitch to angles between approximately -90 and 90 degrees. Since the server indicates in each snapshot if an angle has been clamped, Alice and Bob only need stop encoding and decoding until the angle is no longer clamped to avoid bit errors.

So far we assumed the server sends a snapshot j out to Alice and Bob at the same time. However, if Alice or Bob have slow Internet connections the server will throttle the rate of snapshots during times of high activity. Then a snapshot j may be skipped for either Alice or Bob, but not the other, causing bit deletions, insertions or substitutions. We assume that these days most players have broadband connections (e.g. at least 128 kbps DSL or Cable) and the server will not throttle snapshots even during times of high activity.

Another source of noise is the loss of UDP/IP packets in the network. User commands are highly redundant and only the loss of all of Alice's commands between two snapshots affects the channel. In the worst case such a burst loss can cause substitutions, but not deletions or insertions. On the other hand loss of snapshots can cause deletions, insertions or substitutions if a snapshot j is lost for either Alice or Bob but not the other.

F. Countermeasures

In general there are three classes of countermeasures: eliminating the covert channel, limiting the covert channel capacity and deterring potential users by demonstrating easy detection of the covert channel [1].

FPSCC cannot be *eliminated* because player movement is intrinsic to the game. Additional noise could be introduced to *limit* the covert channel capacity (such as the game server

introducing minute, random fluctuations in every player's view angles – although such noise must also have no visible consequences for players). Alice could counter by increase the amplitude of the covert encoding, but if the view angle fluctuations become visible the channel is no longer covert.

FPSCC may be *detected* in a number of ways. A human Wendy could join a game or replay a recorded game and look for abnormal player movements. Wendy can also analyse packet length and view angle statistics and compare them to 'normal' distributions in order to detect the channel. However, even if Wendy detects the channel and identifies Alice, Bob is still partly protected. If Alice is in broadcast mode, Wendy cannot know the client(s) acting as Bob(s). In unicast mode, Alice can interfere with Wendy identifying Bob by sending 'garbage' (when Bob is not in range) with similar characteristics to FPSCC.

Slight angle movements of another player character are basically invisible to human players. The game server already introduces angle errors of up to one degree by only distributing the integer value of the angle to other players (see Section V). If Alice is a middleman, there is a greater chance that the player whose character's movements are being modulated might notice abnormalities, as the server transmits view angles to the player as full floating point numbers. However, Alice could 'clean' the view angles being sent back to the client from the server.

FPSCC has no need to introduce noticeable additional latency between client and server. Our un-optimised, passive middleman implementation introduced roughly 1 ms additional latency. Given that in-game client-side reporting of 'ping' often fluctuates by ± 10 ms, FPSCC is unlikely to be discovered by players paying close attention to their 'ping'.

V. Implementation

Our prototype of FPSCC is implemented using our Covert Channels Evaluation Framework (CCHEF) [11]. CCHEF is a publicly available software framework under Linux for developing and deploying covert channels over IP networks.

A. Quake III Arena through CCHEF

Figure 7 shows CCHEF transmitting covert information over a network from Alice to Bob. CCHEF allowed Alice and Bob to be implemented as middlemen (instantiated as transparent proxies between Q3 client and server). Current cheat detection systems cannot detect proxy middlemen (see next sub section) and the proxy does not use extra CPU time on either game clients or game server. Finally, during development CCHEF allowed us to test the covert receiver using previously recorded tcpdump trace files as input.

The Q3 network protocol is encrypted. Our implementation of Alice accesses traffic in both directions to derive and update the Q3 encryption keys, decrypt user command packets, perform the covert channel modulation, and re-encrypt the modified user command packets. Bob passively decrypts the packet stream in both directions and decodes the covert channel from the view angle changes of entity state updates.

Our implementation properly handles the various encodings of view angles in different Q3 messages. In user commands angles are encoded as signed 16-bit integer ($\frac{\alpha}{360} \cdot 65535$), angles in player state are encoded as floating-point numbers and angles in entity state (send to other players) are encoded as the integer part of the floating-point number. The integer floating point conversion introduces rounding errors and Alice has to ensure that these do not cause bit errors.

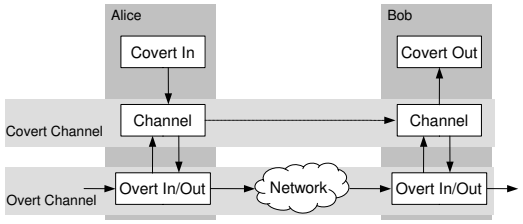

Figure 7. CCHEF instantiating Alice and Bob across a network path

B. Deployment Considerations for Alice and Bob

Cheat protection built into recent multiplayer FPS games makes deployment of covert channels challenging. Client software and data files are checked for integrity and the memory of the client machine is searched for signatures of known cheats. Games typically encrypt their network protocol to protect against proxy-based cheats.

It is relatively straightforward for a player to knowingly act as Bob. FPS clients usually allow recording of demo files, which contain (un-encrypted) all the entity state updates seen during the recorded game sequence. Bob (as player) simply records a 'demo' and decodes the covert data after the game.

Alice needs to modify the game protocol during the game. We have implemented FPSCC using a proxy-based approach, because Q3 is open source and hence the encryption algorithm is known. Proxies cannot be detected by current anti-cheat software integrated in games. However, even if the encryption algorithm is not publicly known it may be possible to crack it as described in [12].

Alice can be deployed on the client as client-side modification (mod). However, many public servers do not accept modified clients. Alice can still be deployed on the client like other client-side cheating tools. These tools work without requiring a modification of the game client and can only be detected if the anti-cheat software knows their signature. Even if Alice's signature became known, it could be easily modified to evade the signature detection. Currently, no method exists for reliably preventing unknown client-side cheats.

VI. Evaluation

We carried out a total of 9 games on the standard Q3 map *q3dm1*. Each game lasted about 10 minutes. Table I summarises the games. If the covert channel was present, Alice sent in broadcast mode and one bit of covert data was encoded per view angle change, encoding in pitch and yaw simultaneously. The covert data sent was random with an equal probability of 0 and 1 bits (uniform random) to avoid bias.

Players other than the covert sender did not know which games had covert channels. Games were played in 'random' order (not as listed in Table I) and players were asked not to drastically change their playing style between games.

Table I
TEST GAMES SUMMARY

Number of players	Games with FPSCC	Games w/o FPSCC
2	1, 2	3
3	4, 5	6
4	7, 8	9

A. Bit Rate and Errors

Table II shows the bit rates and error percentages of the covert channel. Each row is an average of two games. Column one lists the game numbers and column two denotes the client number (possible Bobs). The third column shows Alice's covert transmission rate. Column four shows percentage of bit deletions, column five percentage of bit insertions, and column six percentage of bit substitutions. Column three, four and five show percentages for broadcast (b) and unicast (u) mode (the latter derived after the fact from visibility data).

Table II
SENDER BIT RATES AND CHANNEL ERRORS

Game	Client	Send (b/u) [bit/s]	Deletions (b/u) [%]	Insertions (b/u) [%]	Subst. [%]
1, 2	1	14.6 / 8.0	45.9 / 3.6	0.0 / 2.5	0.0
4, 5	1	18.0 / 10.4	42.9 / 3.0	0.0 / 1.5	0.0
4, 5	2	18.0 / 9.8	46.7 / 3.5	0.0 / 1.5	0.0
7, 8	1	17.8 / 8.2	54.9 / 5.5	0.0 / 2.7	0.0
7, 8	2	17.8 / 9.6	45.8 / 3.1	0.0 / 1.0	0.0
7, 8	3	17.8 / 9.9	45.2 / 3.6	0.0 / 1.5	0.0

Alice was able to send 14-18 bit/s in broadcast mode. As expected, the percentage of deletions is high (43-55%) and the number of insertions is zero. In unicast mode Alice sent 8-10 bit/s and the channel has deletions (3-5.5%) and insertions (1-2.7%). At best our FPSCC implementation encoding only one bit per angle change could achieve transmission rates up to 7-9 bit/s (assuming good bit synchronisation).

It is important to note that deletions and insertions caused by FPSCC are not uniformly distributed, but occur in bursts. This means often there are longer 'transmission periods' followed by longer 'outages'. Therefore, the channel is usable even with the very high deletion rates in broadcast mode. Figure 8 shows typical cumulative distributions (CDFs) of the length of transmission periods (number of bits between any deletions/insertions) with client 1 as Bob (games 1, 4 and 8). Even for the 'worst' game still 40% of transmission periods were at least 50 bits or longer.

Differences in the percentages of deletions/insertions between different receivers are fairly small. No bit substitutions occurred since our FPSCC implementation handles pitch clamping (and also players pitch varied only between -20 and 60 degrees) and player deaths/respawns, the map has no teleporters/movers and there was no packet loss in our testbed.

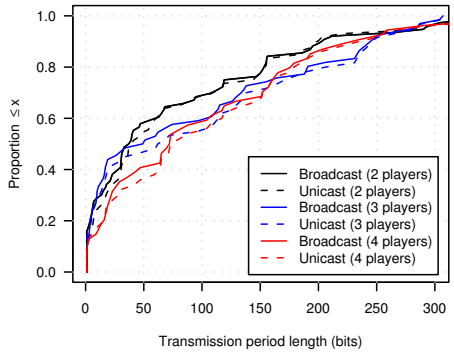

Figure 8. Length of transmission periods for broadcast and unicast mode

This means that if the bit synchronisation problem caused by Q3's asymmetric visibility can be solved, the remaining bit error rate should be fairly low.

B. Fingerprint of FPSCC

The human players did not notice anything strange in the movements of the character being used by Alice. The player being used by Alice likewise did not usually notice any visual abnormalities. However, in our case Alice's player was constantly jumping and moving around. If Alice's player spent a lot of time performing tasks requiring delicate movements (such as sniping at a distant target) discovery of the covert channel is increasingly likely.

Wendy would not have much luck detecting FPSCC by examining packet length distributions. Q3 is known to exhibit wide variation in packet sizes versus the number of players and the map played [13]. Figure 9 compares packet length CDFs for normal traffic and the covert channel (two 2-player games, one 3-player game). It is clear that the existence of the covert channel results in packet size distributions essentially indistinguishable from what is considered 'normal'.

Wendy could instead examine the distribution of angle changes. If Wendy controls the server, a modified server could be installed to log all player movements for later analysis. Alternatively, Wendy could join as a regular player and log all received player movement updates (although covert data would only be received when Wendy is visible to Alice's player). In any case, Figure 10 shows that Wendy would see very little difference between the CDFs of angle values for a normal player and a player acting as covert sender (two games for player 1 and one game for players 2, 3 and 4).

VII. CONCLUSIONS AND FUTURE WORK

We propose and prototype a novel covert channel in first person shooter online game traffic (FPSCC). FPSCC encodes covert bits as imperceptible variations of a player's character's movements. FPSCC is not limited to FPS games, but could be applied to any game where player movements of one or more in-game characters are regularly propagated to other players

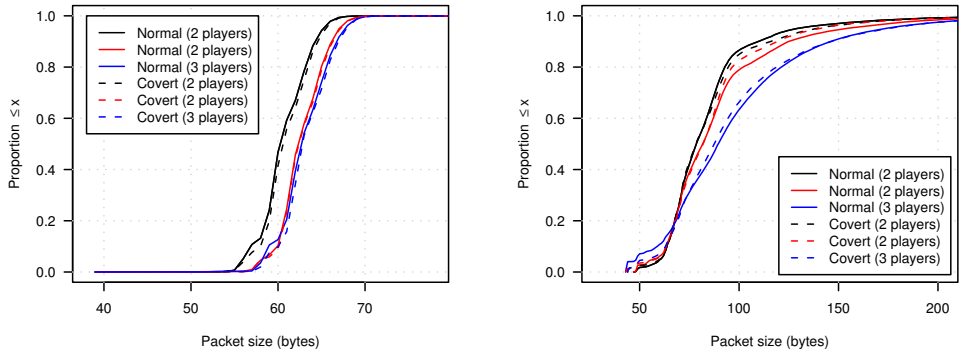

Figure 9. Packet length distributions (including IP/UDP headers) for client to server (left) and server to client (right) traffic

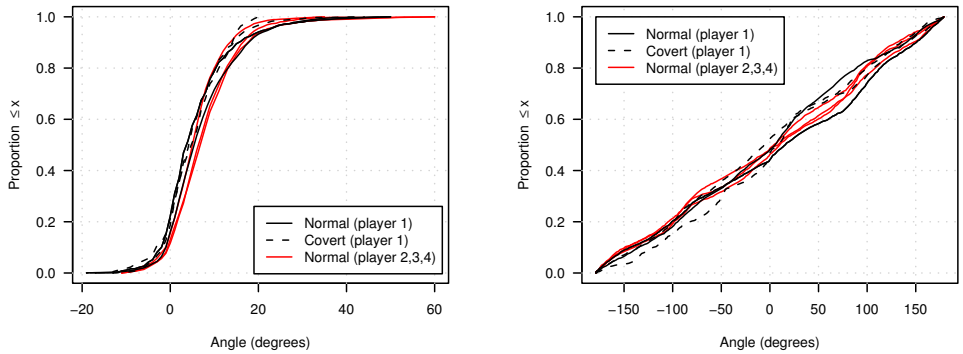

Figure 10. Angle distributions for pitch (left) and yaw (right)

(and their game clients). We analyse FPSCC using a proof-of-concept implementation based on Quake III Arena (Q3). Our tests reveal that FPSCC over Q3 provides bit rates of up to 7-9 bits per second. This low bit rate is sufficient for propagating short text messages. The main advantage of FPSCC over other network channels is that it cannot be fully eliminated and detection is non-trivial.

A number of items remain as future work. We intend to explore mechanisms for reliably transporting messages on top of FPSCC, leveraging existing techniques for framing, loss detection, forward error correction, etc, over noisy wireless channels. We also plan to explore mechanisms for effectively detecting FPSCC. In principle FPSCC allows transmission of multiple covert bits per angle change, but this is not yet implemented in our prototype. While our current results are very encouraging, more trials (with more human players, different maps and game server settings) are planned to enhance our understanding of FPSCC's limitations and performance.

Acknowledgements

We thank Lucas Parry, Lawrence Stewart and Kewin Stoeckigt for being tough opponents in the Q3 tests.

References

[1] S. Zander, G. Armitage, P. Branch, "A Survey of Covert Channels and Countermeasures in Computer Network Protocols," *IEEE Communications Surveys and Tutorials*, vol. 9, pp. 44–57, October 2007.

[2] S. J. Murdoch, P. Zielinski, "Covert Channels for Collusion in Online Computer Games," in *Proceedings of 6th Information Hiding Workshop*, no. 3200 in Lecture Notes in Computer Science, Springer, May 2004.

[3] J. C. Hernandez-Castro, I. Blasco-Lopez, J. M. Estevez-Tapiador, A. Ribagorda-Garnacho, "Steganography in Games: A General Methodology and its Application to the Game of Go," *Computers & Security*, vol. 25, pp. 64–71, February 2006.

[4] M. Handley, C. Kreibich, V. Paxson, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," in *Proceedings of 10th USENIX Security Symposium*, August 2001.

[5] I. S. Moskowitz, M. H. Kang, "Covert Channels - Here to Stay?," in *Proceedings of 9th Annual Conference on Computer Assurance*, pp. 235–244, 1994.

[6] Id Software, "Quake III Arena." http://www.idsoftware.com.

[7] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in *Proceedings of Advances in Cryptology (CRYPTO)*, pp. 51–67, 1983.

[8] G. Armitage, "Optimising Online FPS Game Server Discovery through Clustering Servers by Origin Autonomous System," in *Proceedings of ACM NOSSDAV 2008*, May 2008.

[9] G. Armitage, M. Claypool, P. Branch, *Networking and Online Games - Understanding and Engineering Multiplayer Internet Games*. John Wiley & Sons, April 2006.

[10] K. Sayood, *Introduction to Data Compression*. Morgan Kaufmann, 2000.

[11] S. Zander, "CCHEF - Covert Channels Evaluation Framework," 2007. http://caia.swin.edu.au/cv/szander/cc/cchef/.

[12] M. Bond, "Chewing on the 0xDEADBEEF," August 2007. http://www.cl.cam.ac.uk/~mkb23/research/ChewingOnDeadBeef-Redact.pdf.

[13] T. Lang, P. Branch, G. Armitage, "A Synthetic Traffic Model for Quake 3," in *Proceedings of ACM SIGCHI ACE*, June 2004.