Cyber Security in Internet-Based Multiplayer Gaming

Christopher Dixon and Luay A. Wahsheh*

Abstract—**Internet-based multiplayer gaming has been used extensively over the past few years. The main concern that has arisen from online game distributers is the prevention of cheating. Most online games are organized through the client/server model. This is done in an attempt to prevent cheating, steering away from peer-to-peer applications which would be better for relieving the work load of servers by pushing tasks onto the clients, but in turn opens up Pandora's box for hackers. The quest for better security in online games benefits both the players as well as the game developers. In this research work, we investigate current security issues in online multiplayer gaming as well as potential solutions to these problems. We discuss how cheats are usually centralized in the protocol level of the game design. These cheats will range from fixed-delay cheats to collusion cheats. Also discussed in this research work are four models that attempt to counterattack these exploits and in turn fix the cheating problem.**

Keywords: cyber security, networked computer games, cheat prevention, peer-to-peer, latency.

1 Introduction

With the rising trends in online gaming, comes the problem of presenting a cost effective measure to prevent hackers and cyber security issues arising from users. It has been shown that many cheats in online games are due to poor or non-existent security objectives in the game design [12]. Although security plays a major role in the design of software systems, it is still not considered an explicit part of the development process [8]. Security requirements are usually added to an already existing system. As a result, this leads to numerous problems with the overall security design.

Hacking can be defined as gaining access to a computer by illegally accessing information. A simple way to look at hacking is to think of a burglar breaking into a house [1]. The house is a computer, and the burglar is a hacker. Hacking has become a very serious issue in computer games [13]. Cyber security can be viewed as mechanisms that are designed to enforce secure (proper) behavior on the operation of computers in networks that are connected to the Internet. Secure is defined by a security policy that addresses information confidentiality, integrity, and availability [7]. We consider a system secure if the security policy is being correctly enforced. In current models, the game itself is run off of a client/server (C/S) model, but most updates are distributed on a peer-to-peer network [6]. While C/S is simple, secure, and reliable, it has limited scalability to support a large number of players [10]. As soon as the servers offload some resources onto the clients in order to be more cost effective, the issue of trustworthiness comes into play. Although, putting everything on a peer-to-peer system does not necessarily make it secure [6].

In an online survey [11], 35% of 594 players who responded within a 2-week period admitted to cheating/hacking in an online game. The hacks can be as simple as a speed hack which allows the user to move faster than was originally designed or an aimbot which allows the user to have a heightened sense of precision accuracy not originally designed in the game; or this can lead to more serious problems like stealing online accounts as well as altering online game play environments. Table 1 shows examples of common cheats in the game level, application level, protocol level, and the infrastructure level and possible solutions to fix them [9].

Potential solutions have been proposed to fix certain problems, but most of these issues cannot be resolved. In this paper, we discuss simple hacks in online gaming as well as hacks that can occur in peer-to-peer applications. This paper also discusses solution designs to hacks that occur in the protocol layer of game design which is where most hacks occur.

For the purpose of this research work, let us refer to Bob and Alice. Bob is the trusted player and Alice is the cheater. Cheating in online gaming can be categorized by the layer in which it occurs: game, application, protocol, or network [5]. All of the designed protocols discussed in this paper refer to issues that arise from the protocol level of the game design. This layer is where the passing of packets from an individual to the server as well as other players occurs.

The five protocol level cheats that all these designs

*Department of Computer Science, Norfolk State University, 700 Park Avenue, Norfolk, Virginia 23504, USA, c.s.dixon@spartans.nsu.edu, law@nsu.edu.

Table 1: Game cheats and their possible solutions.

Cheat	C/S	PB/VAC2	AS	NEO/SEA	RACS	P2P RC
Game Level						
Bug	✓	×	✓	✓	✓	✓
RMT/Power Levelling	✓	×	×	×	✓	✓
Application Level						
Information Exposure, Invalid Commands	✓	×	×	×	✓	✓
Bots/reflex enhancers	×	✓	×	×	×	×
Protocol Level						
Suppressed update, Timestamp Fixed delay, Inconsistency	✓	×	✓	✓	✓	✓
Collusion	☒	☒	☒	☒	☒	☒
Spoofing, Replay	✓	×	×	✓	✓	✓
Undo	N/A	×	✓	×	N/A	N/A
Blind Opponent	N/A	×	N/A	N/A	✓	N/A
Infrastructure Level						
Information Exposure	✓	✓	×	×	✓	✓
Proxy/Reflex Enhancers	☒	☒	☒	☒	☒	☒

✓ - solvable × - not yet solved
☒ - not solvable N/A - not applicable

discuss are fixed-delay cheats, timestamp cheats, suppressed update cheats, inconsistency cheats, and collusion cheats. A fixed-delay cheat is when Alice adds a delay to her packets being sent to Bob. This results in Alice receiving Bob's packets faster, thus allowing Alice to react faster in relation to Bob's actions. A timestamp cheat is when Alice alters the time in which her action was sent to Bob. Alice would receive Bob's move and then send Bob a move that was time stamped for Bob before the received update was received resulting in a future look at what Bob was going to do and thus gives Alice a way to react. A suppressed update cheat is when Alice does not send any updates at all to Bob, but will still continue to receive Bob's updates. This makes Alice invisible to Bob. Alice will eventually send an update when she realizes that she is about to be kicked from the server. An inconsistency cheat can be very complex. Alice will send correct updates to all other players in a game except for Bob. Bob will think Alice is in a different location, but Bob's team will disagree with him. Alice will eventually send a correct update to Bob to hide her cheat. A collusion cheat happens when players share information about another player. Bob's team mate can share Bob's updates with Alice, thus giving Alice an advantage over Bob.

In all of the research work that is discussed in this paper, the developers tested their protocols on games that were designed by the protocol developers. None of these protocol designs have been tested on popular mainstream online multiplayer games as of their publications.

The remainder of this paper is organized as follows: in Section 2, the new event ordering protocol design is discussed which was designed to combat all five protocol level cheats previously discussed. Section 3 discusses a model that was built around the fair-ordered message exchange protocol and its attempt to prevent users from using the timestamp cheat. Section 4 discusses the secure event agreement protocol design which is built on the new event ordering protocol design and claims to be an improvement on the new event ordering protocol design. Section 5 discusses the efficient and secure event signature protocol. Then, Section 6 concludes the paper.

2 New-Event Ordering Protocol

The goal of the new event ordering protocol is to address the problems associated with the peer-to-peer architecture for massively multiplayer online games. New-event ordering protocol provides low latency event ordering while still preventing common protocol level cheats [5]. In this protocol, the players decide what happens. The majority of players vote on the updates that are received and if the majority of votes say that an update is valid, then the update is valid, even if all the players did not receive the update.

The new event ordering protocol was inspired by bucket synchronization in that it divides time into "rounds" which is used to set a maximum latency in which a player has time to submit his or her updates [5]. The maximum latency can be no larger than three times that of the slowest player. For example, assume that we have a multiplayer game active and all players as well as the servers are located in the United States of America. Sometime later, a player from China connects to the server where the server calculates that his or her round trip time is 500 ms. The maximum latency of that server would now be 1500 ms between rounds. Of course, the lower the round trip time, the better the gaming experience. Each player sends updates to the server which distributes his or her updates among the other players. Any late updates submitted to the server are considered invalid. At the end of the round, the players "vote" on what updates were submitted to them. These "voting sessions" are done automatically with the player having to do nothing. This prevents other players from changing their updates from player to player.

Each update that is sent through the new event ordering protocol must be received before the end of the round. Also, these messages are encrypted. In the following round, the players will send their keys for the previous encrypted update sent to all players so that the update can be decrypted. The updates that are sent to all players have the following format using equation 1 [5]:

Table 2: Player A's table of votes.

Player	Bit-vector
A	1 1 0 1 0
B	0 1 0 1 0
C	1 1 1 0 0
D	1 1 1 1 1
E	*packet lost*
Voting tally	3 4 2 3 1

$$M_A^r = E(S_A(U_A^r)), K_A^{r-1}, S_A(V_A^{r-1}) \quad (1)$$

In this message, $E(S_A(U_A^r))$ is an encrypted signature, $S_A()$, of update U_A^r, K_A^{r-1} is A's key for the update from round $r-1$, and V_A^{r-1} is the bit vector of votes for the messages received during round $r-1$.

Consistency is achieved through the majority voting. The updates that are sent are only validated if the majority of the players received the update. If a vote is not received, it is considered invalid. If the majority of the players vote that the update is invalid, then all players will once again attempt to communicate with the "invalid" player. Table 2 shows an example voting round using the new event ordering protocol [5].

One of the benefits to the "voting" concept is that the game can continue no matter what. Assuming that the majority of the players are returning a "valid" update for all players, then the game will continue to proceed with no problems. Another benefit to "voting" is that it will keep the majority of the players happy. Instead of lagging a server beyond playing, if a player fails to communicate with the other players, then the non-communicating player is kicked from the server.

According to the creators of the new event ordering protocol, all five forms of the protocol level cheats were fixed. Fixed-delay cheats were addressed by the bounds placed on the time limit in which updates could be received. If an update was received late, then the update was considered invalid and did not count towards the final vote. The timestamp cheat was also addressed in the same manner. Since the rounds had a maximum length, no update could be received late since it would be considered invalid. The suppressed update cheat was fixed by the new event ordering protocol's ability to not send updates to players that are not in turn sending updates. The user will not receive updates pertaining to the actions that other players are performing and will eventually be kicked from the server if the activity continues to happen. The inconsistency cheat is addressed with the use of digital signatures and state comparisons. During gameplay, players will occasionally check the state of all players. If two or more players discover an inconsistency, the protocol can trace the packets and can be used as evidence when attempting to ban a player. Once the proof is obtained, the procedure of banning a player is inevitable. If two players receive different information from a player, that information can be used against the cheating player and can be used as evidence to ban that player from the server. Finally, the new event ordering protocol claims to address collusion in three ways [5]. First, the new event ordering protocol can adjust the majority value sufficiently high to prevent collusion. Second, the authenticating directory (AD) service prevents players from logging in multiple times and gaining a majority. Third, the communication component of the protocol design can randomly select witnesses for a new event ordering protocol group.

Overall, the new event ordering protocol design has been a stepping stone for other developers in the gaming community, generating many "offspring", two of which will be discussed in this research work. Looking at network latencies and dividing them into rounds is a smart way to organize the time frame in which updates from players are to be received. The future work of the new event ordering protocol developers is to design a protocol for group management, event propagation, storage, and computation.

3 Cheat Controlled Protocol

The cheat controlled protocol design was created to address the timestamp cheat which refers to a situation where a player will alter the time that his or her update would have been sent to the other players, thus giving the cheating player a future look at what is going to happen and allowing him or her to react in the way that best fits his or her needs. The cheat controlled protocol uses two main equations to calculate whether a delayed message is due to network congestion or if the delayed message is an attempt to cheat.

Equation 2 measures the proposed arrival time of an update from a player [3].

$$PAT_{jk}^i = U_i + \delta_{jk}^i + ERTT_j + ES_j^i + RTTT_j + ST_j^i \quad (2)$$

In equation 2, PAT stands for the proposed arrival time of the update. U_i stands for the server time at time i. δ_{jk}^i stands for the reaction time of a player. $ERTT_j$ stands for the estimated round trip time. ES_j^i stands for the estimated update message processing time. $RTTT_j$ stands for the round trip time tolerance. ST_j^i stands for the update message processing time tolerance.

The information that represents the proposed arrival time of the update is used in combination with equation 3, which calculates the actual arrival time for the update [3]:

$$AAT^i_{jk} = U_i + A\delta^i_{jk} + ARTT_j + AS^i_j \quad (3)$$

In equation 3, AAT stands for the actual arrival time for the update to the server. U_i once again stands for the server time. $A\delta^i_{jk}$ stands for the actual reaction time that the player submitted. $ARTT_j$ stands for the actual round trip time. AS^i_j stands for the actual update message processing time. This equation along with equation 2 help the cheat controlled protocol design to decide whether an update is "valid" or "invalid".

In order to determine whether or not a player is cheating, the cheat controlled protocol compares the proposed arrival time and the actual arrival time. If the actual arrival time is less than or equal to the proposed arrival time, then the player is assumed to not be cheating. If the actual arrival time is greater than the proposed arrival time, then there can be a few reasons to why there is a delay. One reason is that there can be network congestion between the server and the player [3]. An easy way to tell if a player is attempting to cheat is done by the server. If the server does not see any network congestion occurring, then the delayed message is an attempted cheat by a player.

The cheat controlled protocol is a very good design, but is very primitive when compared to other protocols designed to prevent cheating in online games. Only looking at one cheat, the timestamp cheat, helps the cheat controlled protocol focus on the main problem but detours from the fact that developers are trying to prevent all five protocol level cheats that are discussed in this paper. Another issue is that the cheat controlled protocol is useful only if all players are playing on a steady connection. If a player has any network congestion that may delay the passing of that player's update to the server, the cheat controlled protocol may see that player as a cheater and kick that player from the game server. Overall, this protocol would be very effective, if we were only considering the timestamp cheat; but since we are trying to remedy five of the major protocol level cheats, the cheat controlled protocol seems quite weak when trying to fully protect a game server.

4 Secure Event Agreement Protocol

The secure event agreement protocol was designed after the new event ordering protocol (which was discussed earlier). The reason for the creation of the secure event agreement protocol was because the developers who designed the secure event agreement protocol researched the new event ordering protocol and claimed that it only prevented three out of the five types of protocol level cheats that it claimed to remedy. The developers of the secure event agreement protocol claim that their new protocol design addresses all five protocol level cheats and also that the performance of the secure event agreement protocol is at worst equal to that of the new event ordering protocol and in some cases is better [4].

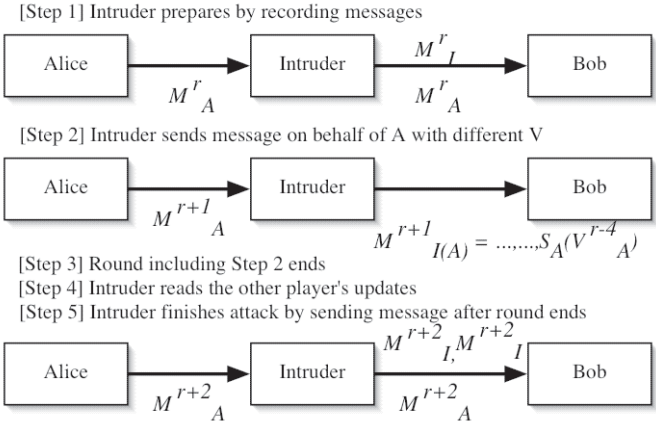

Figure 1: An illustration of the voting attack.

As a refresher, the new event ordering protocol limited rounds to three times that of the slowest player's round trip time. The new event ordering protocol also compensated for loss of updates by using a voting system in which if the majority of the players could "validate" the update, then the update was considered to be a "valid" update even if not all of the players received the update. The developers of the secure event agreement protocol claim that there are three problems with the design of the new event ordering protocol [4]. First, it was discovered that an attacker can replay older updates for another player. Second, an attacker can fabricate messages from previously seen "validated" updates. Third, an attacker can send different updates to different players. The developers of the secure event agreement protocol show how people can exploit these design flaws and give a proposed design that will effectively counter all protocol level cheats. Figure 1 shows how an intruder can initiate a voting attack in a system that does not implement the secure event agreement protocol [4]. In this case, both Alice and Bob are honest players with an intruder intercepting their messages.

First, an attacker can resend an update to another player. We will once again refer to Alice as being the player who is cheating and Bob being the honest player. To resend an update to another player, a player would need to have seen the previous round update and the current round update so that he or she has the key to decrypt the message. So Alice would have the previous round message as well as the message of the current round to obtain the decryption key for the message. Once Alice has decrypted the message, she can either re-encrypt it with a new key or resend it as it is. This is still considered to be an inconsistency cheat.

Second, an attacker can fabricate messages from

previously seen "validated" updates. The voting for the new event ordering protocol was done with zeros and ones, *zero* meaning that the update was not received before the end of the round and *one* meaning that the update was received before the end of the round. So Alice could exploit this by gathering the votes from many other players, and creating false messages that would appear to come from the other players. Doing this gives Alice the ability to send her update after the round has come to an end and the other players will consider that update valid. This allows Alice to get a future look at the other players' moves and allows her to react in an appropriate manner. Fabricating these messages will make the voting scheme used by the new event ordering protocol obsolete. This is still considered to be a timestamp cheat.

Third, an attacker can send different updates to different players. The problem that has arisen with this is that the new event ordering protocol did not provide an assurance policy that all players had received the same updates. Since the updates are not tied to any specific round, it is possible for Alice to resend signed updates to different players which would convince them, for example, that Bob was cheating. This would give Alice the upper hand. If Alice can distract the rest of the players from her cheating by making it seem that Bob is the dishonest player, Alice will have the advantage of doing other cheats and having Bob take the blame. This is still considered to be an inconsistency cheat.

The way that the secure event agreement protocol addresses these flaws in the new event ordering protocol is by adding an extra encryption algorithm and applying it to the update [4]. The encryption algorithm uses a hashing function to commit the player to the value of the message. The equation, labeled as equation 4, is as follows:

$$Commit_A^r = H(U_A^r, n^r, SessID, ID_A) \qquad (4)$$

In equation 4, the value U_A^r stands for the update of player A at round r. n^r stands for a random value that is specific to round r. The value of n cannot have been previously used and cannot be used again after the current round is over. The session ID, $SessID$, is included in the hashing function to prevent the message from being used again in the future. The result that is given from equation 4 is then plugged back into an altered form of equation 1 to give the new resulting message that is to be sent to the players [6]. Equation 5 is just an altered form of equation 2 that switches the encryption in the message with the new $Commit_A^r$ equation and provides more security on the message that is being sent to all the other players. With the addition of equations 4 and 5 to the basis that was the design of the new event ordering protocol, the secure event agreement protocol successfully addresses all five protocol level cheats that have been discussed in this paper.

$$M_A^r = S_A(Commit_A^r, U_A^{r-1}, Vh_A^{r-1}, n^{r-1}, r) \qquad (5)$$

The secure event agreement protocol addresses the fixed delay cheat by making all players submit their moves before the end of the round. At the conclusion of the round, all the players will vote to determine whether the updates are "valid" or "invalid" and upon completion of all the voting for the round, the moves of all the players for the round will be revealed. The secure event agreement protocol addresses the timestamp cheat by limiting the rounds to a fixed time. If a player's update has not been sent out by the end of the round, then it will not be accepted and will be considered to be an "invalid" update. The secure event agreement protocol addresses the suppressed update cheat with the inclusion of equation 4 in its message creation design.

Since the commit feature is included in the message creation, it is now possible for players who did not receive the update to know the update to which the player has committed. This is made possible by the inclusion of the players ID in equation 4. The player's ID acts as a signature which authenticates the update making it very difficult to fabricate an update. The secure event agreement protocol addresses the inconsistency cheat by the inclusion of equation 4 in the message creation design. The value of $Commit_A^r$ should be equal for all messages that are sent out during a round. If the majority of the players receive a message with the same value of $Commit_A^r$, then the update will be considered "valid" and accepted by all players. Finally, the secure event agreement protocol claims to address the collusion cheat by still allowing players to share information. With the inclusion of equation 4, the only information that can be shared among players is that of the decrypted updates.

When compared to the other anti-cheating protocols that have been discussed in this paper, the secure event agreement protocol seems to be the best candidate to get the job done. With the inclusion of equation 4 to its design, it now successfully protects from the five protocol level cheats in which it was designed to do. Also, the secure event agreement protocol does an excellent job of building on the design of the new event ordering protocol and also by documenting how the new event ordering protocol does not effectively protect against all five protocol level cheats discussed in this paper. The future work for the developers of the secure event agreement protocol was to design group selection and round negotiation protocols that will work in conjunction with the secure event agreement protocol [4].

Table 3: Notations.

$Player_i$	each player in the game, where $i \in \{1..m\}$
$H(x)$	hash operator with input message x
OSK_i^j	$player_i$'s j^{th} one-time signature key
$S_{sk}(x)$	message signing x by secret key sk
$x\|y$	concatenation of the message x and y
δ_i^j	signature signed by $player_i$'s j^{th} OSK
Δ	signature signed by secret key sk

5 Efficient and Secure Event Signature Protocol

The efficient and secure event signature protocol was designed to effectively sign multiple updates at one time. It was also designed to enforce unforgeability and verifiability when transferring updates from player to player. Unforgeability and verifiability are requirements for digital signatures, assuring the user that the signature that is shown on the updates that are received is from the player [2]. The efficient and secure event signature protocol was designed on both the new event ordering protocol and the secure event agreement protocol. The efficient and secure event signature protocol's main goal is to achieve scalability and fairness for peer-to-peer based massive multiplayer online games.

The only thing that the developers of the efficient and secure event signature protocol claim is wrong with the new event ordering protocol and the secure event agreement protocol is that both of these protocols are not practical due to the excessive use of cryptographic signatures [2]. The efficient and secure event signature protocol computes the digest of the message by using a hash function before signing the message. The efficient and secure event signature protocol is divided into four phases: the initialization phase, the signing phase, the verification phase, and the re-initialization phase. Table 3 breaks down the variable notations that will be used in these four phases [2].

The initialization phase is where everything starts. During the initialization phase, the players will compute their one time signature keys. The one time signature keys are generated with the following formula $OSK_i^j = H(OSK_i^{j-1})$ [2]. In the end of the initialization phase, the players sign the one time signature keys with the player's secret key. Once the keys are generated, they are stored on the hard drive of the user's computer. During the signing phase, the messages are then signed and sent to the other players. As an example, the first update to be sent by $player_i$ would be $H(OSK_i^1|M_i^1)$ which is a hashing of the player's one time signature key and the message for that round [2]. In the second round, $player_i$ will send the message for round two as well as the key that will be used to decrypt the message from the previous round. The third phase of the efficient and secure event signature protocol is the verification phase. It verifies whether or not the update that was received came from the player claiming to have sent it. This is done by using the secret key that is sent in the round after the encrypted update was received. The final phase is the re-initialization phase. Once a player has exhausted all of his or her one time signature keys, the player must generate new one time signature keys. This is the main purpose of this phase, although it does not occur as often as the other three phases.

The efficient and secure event signature protocol succeeds in doing what it was designed to do. The efficient and secure event signature protocol was built on the foundations of the new event ordering protocol and the secure event agreement protocol and wanted to improve on the unforgeability and verifiability. It succeeds in doing this by creating a hash chain when encrypting the messages that are sent from player to player. Also, since it was designed on the basis of two protocol level designs, it has been proven to cover the five protocol level cheats that were addressed in this paper.

6 Conclusion

This research work has shown the designs of four protocols that were designed to counter attack against protocol level cheating. The new event ordering protocol, cheat controlled protocol, secure event agreement protocol, and efficient and secure event signature protocol all have their own merits in attempting to stop protocol level cheating. We believe that the best of these four protocols is the efficient and secure event signature protocol. Building over the design flaws that were documented with the new event ordering protocol and the basis of the secure event agreement protocol when attempting to compare it to the cheat controlled protocol, which was designed to stop timestamp cheating, the efficient and secure event signature protocol overshadows what was done with the other three protocols discussed. Also, the future designs (on which they are being worked) for the secure event agreement protocol seem very interesting and we believe that it can become one of the major foundations for the prevention of protocol level cheating as well as in the continuing development of cyber security in online multiplayer gaming.

References

[1] Cebula, S. L., Wahsheh, L. A., "Computer Ethical Hacking: An Education Perspective," *13th International Conference on Computers and Advanced Technology in Education*, in press (accepted for publication on June 8, 2010).

[2] Chan, M.-C., Hu, S.-Y., Jiang, J.-R., "An Efficient and Secure Event Signature (EASES) Protocol for

Peer-to-peer Massively Multiplayer Online Games," *Computer Networks*, V52, N9, pp. 1838–1845, 2008.

[3] Chen, B. D., Maheswaran, M., "A Cheat Controlled Protocol for Centralized Online Multiplayer Games," *3rd ACM SIGCOMM Workshop on Network and System Support for Games*, pp. 139–143, 2004.

[4] Corman, A. B., Douglas, S., Schachte, P., Teague, V., "A Secure Event Agreement (SEA) Protocol for Peer-to-Peer Games," *1st International Conference on Availability, Reliability and Security*, pp. 34–41, 2006.

[5] GauthierDickey, C., Zappala, D., Lo, V., Marr, J., "Low Latency and Cheat-Proof Event Ordering for Peer-to-Peer Games," *14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 134–139, 2004.

[6] Kabus, P., Buchmann, A. P., "Design of a Cheat-Resistant P2P Online Gaming System," *2nd International Conference on Digital Interactive Media in Entertainment and Arts*, pp. 113–120, 2007.

[7] Wahsheh, L. A., "An Analysis of Security Policy Specification Techniques," *9th International Conference on Security and Management*, in press (accepted for publication on April 12, 2010).

[8] Wahsheh, L. A., Alves-Foss, J., "Policy-Based Security for Wireless Components in High Assurance Computer Systems," *Journal of Computer Science*, V3, N9, pp. 726–735, 2007.

[9] Webb, S. D., Soh, S., "A Survey on Network Game Cheats and P2P Solutions," *Australian Journal of Intelligent Information Processing Systems*, V9, N4, pp. 34–43, 2008.

[10] Webb, S. D., Soh, S., "Cheating in Networked Computer Games: A Review," *2nd International Conference on Digital Interactive Media in Entertainment and Arts*, pp. 105–112, 2007.

[11] Yan, J., Choi, H.-J., "Security Issues in Online Gaming," *International Journal for the Application of Technology in Information Environments*, V20, N2, pp. 125–133, 2002.

[12] Yan, J., Randell, B., "A Systematic Classification of Cheating in Online Games," *4th ACM SIGCOMM Workshop on Network and System Support for Games*, pp. 1–9, 2005.

[13] Yan, J., Randell, B., "Security in Computer Games: From Pong to Online Poker," Technical Report Series CS-TR-889, School of Computing Science, University of Newcastle upon Tyne, United Kingdom, 2005.