Securing Online Games

Reducing the Attack Surface in Massively Multiplayer Online Role-Playing Games

As online games become increasingly complex and popular, malware authors could start targeting these virtual worlds to launch attacks. Two case studies show how an attacker can leverage various features of online games to take over players' computers.



ames is 15 years old and loves playing computer games online with his friends. While playing his favorite game one day, he gets a message from a player he doesn't know, telling him to check out a cool, new in-game item. Curious, he clicks a link in the message—nothing. He clicks the second link in the message—still nothing. Disappointed, James returns to his usual gaming, but unbeknownst to him, an identity thief now controls his computer, which silently grants the thief access to the family's tax returns, emails, and other personal information. James is just one of the millions of vulnerable subscribers to this online game.

Massively multiplayer online role-playing games (MMORPGs) are joining the ranks of software popular enough to be bombarded by attacks. For the past decade, we've witnessed exploit after exploit targeting our favorite Web browsers, email clients, office productivity software, and operating systems.¹ Many of the characteristics that make attacks against these applications possible and profitable have worked their way into online games, too. The large attack surfaces in these games, for example, create ample opportunity for attackers to pinpoint security vulnerabilities, and as the user base increases, so does attacker incentive. Thus, it's important for game developers to recognize this trend and incorporate a strong security focus into their software development life cycles.

So far, most research surrounding online games has been specific to cheating² and theft of virtual goods via external malware and overly permissive game scripts.³ Attackers haven't capitalized on the large MMORPG attack surface as a means for compromising gamers' computers—yet. To our knowledge, this article describes the only

two successful attacks of this nature. These two case studies demonstrate the wide array of attack possibilities that MMORPGs make possible.

Beyond Cheating

Online games and virtual worlds have experienced various nefarious activities at the hands of cheaters, including item duplication, sight through walls, invincibility, and automatic aiming. These cheats are sometimes called *exploits*, confusing them somewhat with attacks that fully compromise host machines. In fact, at worst, cheats alter the gaming experience for those playing, but they're limited to the virtual worlds they affect.

This isn't to say that we should take cheats lightly just because they have limited effects on host machines. After all, some aspects of the game environment are tied to the real world-for example, players can buy and sell in-game currency and items using real-world money.² Duplication in this context becomes a profitable business for criminals and hurts players who have made honest financial investments in these game worlds. Gamer dissatisfaction leads to lower sales and canceled subscriptions, motivating game companies to continually stop cheaters and retain their subscribers. But beyond cheats lurks a more serious threat to both gamers and game companies alike-the risk of security vulnerabilities that could compromise players' home computers. A vulnerability in an MMORPG or other online game that grants an attacker control of another player's computer costs the victim more than STEPHEN BONO, DAN CASELDEN, GABRIEL LANDAU, AND CHARLIE MILLER Independent Security Evaluators just an unsatisfactory gaming experience: it can result in stolen personal, financial, or corporate information or a corrupted, useless computer. For videogame companies, the consequences can be equally harrow-

It's one thing to patch a cheat every month or so—but a 0-day exploit compromising hundreds or thousands of subscriber

computers isn't soon forgotten.

ing. It's one thing to patch a cheat every month or so—once the nuisances disappear, the gaming generally continues—but a 0-day exploit compromising hundreds or thousands of subscriber computers isn't soon forgotten.

Complexity

MMORPGs aren't like typical computer games: thousands of people play them by simultaneously interacting online in a virtual environment. But more than the complexity of servicing all these players at once in real time is the enormous set of features coded into the games.

MMORPGs, like most Internet software, adopt client-server models as their bases. Game clients (the players) connect to online servers (the virtual worlds). The servers constantly update the client software with the sights, sounds, and happenings in proximity to the player's avatar. When an avatar in the game performs an action, such as casting a spell, shouting in a crowded room, or making a slight movement, the action is sent to the server, processed, and forwarded to all relevant game clients so that other players privy to the act can witness it in near real time.

This model might not seem overly complicated, but in striving for a limitless interactive experience, these games are packed with features, letting players perform thousands of actions. When compounded by the numerous side effects each action can have within the online world, the game logic's complexity soars. In fact, MMORPGs can become so complex that even developers can lose control of them, as was seen in the online game World of Warcraft when, due to an unforeseen combination of game features, a developercreated disease ran rampant through cities, afflicting and killing thousands of player avatars.⁴

To stay ahead of the curve (and the competition), game developers must constantly update and add features to their software. Consider Web browsers feature after feature has driven the straightforward concept of text-only Web browsing to the massive, dynamic Web 2.0 multimedia experience we see today. In the same way, competition and consumer demand for rich functionality pushed online roleplaying games from text-based multiuser dungeons in the early to mid-1990s to the massively complex adventures they've become.

For Web browsers, the feature-laden fight for market dominance comes with a cost: a decade or more of frequent security vulnerabilities. Internet users are generally aware of what not to do when it comes to handling email attachments, dealing with spam, disclosing personal information, and clicking on popup windows, but online gamers haven't yet had to learn analogous safety behavior for virtual worlds. In many cases, games provide so much opportunity for possible attacks that it might not matter.

Large Attack Surfaces

Like Web browsers, the attack surface in MMORPGs is extensive. On top of the numerous client-to-serverto-client communications occurring, MMORPGs have begun to incorporate third-party plug-ins, processing capabilities for various movie, sound, and image formats, a reliance on external applications, and the ability for direct P2P communication. Although this wealth of features enhances the gaming experience, it also provides attackers with ample opportunity to exploit the game client.

Client to Server to Client

Most client-to-client interaction during online gameplay is provided through a server middle man. A message sent from one user to another is handled in the sender's client software, the server software, and finally delivered to and processed by the recipient's client software. Despite the ability to include several security checks along the way, attackers can sometimes send malicious content through—essentially, delivering an attack to another player's game client through the server.

Following the tradition of denying bad traffic closest to its source, the most efficient place to prevent an attack of this kind is within the sending game client. However, a determined attacker can modify or emulate the game client to circumvent such restrictions.² In some cases, an attacker might also be able to spoof messages from a game server to the player, bypassing server-side validation entirely. Effective input validation must deny bad traffic at both the sending and receiving ends of game clients and servers.

In general, each additional way by which two clients can communicate with each other increases the probability of vulnerabilities. Because these features are essential for online games, the best mitigation is a strong security-conscious quality assurance component in the development process's testing phase to detect and repair possible vulnerabilities.

Third-Party Plug-ins

Some features aren't coded by game developers—the inclusion of user-created add-ons lets players create features of their own, such as the SpamMeNot addon for World of Warcraft, which blocks in-game solicitations from known spammers (wow.curse.com/ downloads/wow-addons/details/spam-me-not.aspx). MMORPGS also use third-party plug-ins such as Vivox, which allows streaming voice capability between game clients. Plug-ins and add-ons expand the number of potential attack vectors, from installing services and opening listening ports on client machines to downloading and processing content from the Web and other users.

There's no question that many of these additions are useful, often necessary, but as has been demonstrated repeatedly, these extra features can come with security holes that open new windows for attackers. Consider the enormous amount of content on the Web that requires the Adobe Flash Player. A review of this Web browser plug-in's revision history reveals several security vulnerabilities fixed with each update (www. kb.cert.org/vuls/byid?searchview&query=flash). World of Warcraft alone has more than 5,000 available add-ons for download (wow.curse.com/ downloads/wow-addons/default.aspx). With more than 10 million subscribers, the game's popularity speaks for itself (http://eu.blizzard.com/en/press/080122.html). As individual add-ons become widely used, they'll likely become the focus of malware authors intent on taking over subscriber computers.

Mitigations for these sorts of attacks can stem from the same techniques adopted in Web browsers running plug-ins and add-ons in a sandboxed environment in which privileged system commands are restricted. In some cases, a standard scripting language for creating add-ons might be preferable, such as World of Warcraft's use of Lua. Additionally, playerbuilt add-ons can undergo a security vetting process before being "approved."

Numerous File Formats

The increasing number of supported file formats transmitted and processed during gameplay is further broadening game attack surfaces. The MMORPG Second Life supports dozens of multimedia formats that support various images, sounds, movies, and other multimedia as well as scripts and markup languages to enhance the gaming experience. However, more supported file formats yield more opportunity for attackers to locate vulnerabilities. Research has shown time and again that parsing small and seemingly innocuous (yet malicious) multimedia files can have drastic consequences (www.securityfocus.com/bid).

Often, the standard media-processing libraries included with the distributed game software parse

and render these files. Although many such libraries are standard—and using time-hardened standard libraries is almost always safer than writing new code from scratch—they, too, are known to be vulnerable. Moreover, it's common for software utilizing these libraries to remain unpatched long after a vulnerability is publicly documented and corrected. Attackers simply find software that uses outdated and vulnerable libraries and exploit them.^{5,6} Game developers must therefore ensure that homegrown multimedia processing engines are rigorously tested for vulnerabilities and that any standard libraries their games use are patched along with the game itself.

Additional Delivery Mechanisms

To be as interactive as possible, some MMORPGs take advantage of external applications, providing in-game actions that trigger out-of-game software execution, such as opening Web browsers and load-ing music players. Although this allows easy access to external information and makes the gaming experience more complete, it also opens numerous delivery mechanisms for attacks.

For more than a decade, email clients have proven an effective delivery mechanism for malicious files by convincing naive users to open them, execute their payload, and compromise the host machine. This exploit has also spread to instant messaging software, filesharing communities, Web sites, and other mediums in which users can be duped into downloading and opening malicious files. It's only a matter of time before MMORPGs are similarly targeted: most gamers haven't yet instituted the same caution associated with safe Web browsing and email reading during gameplay. It might be too much to ask for game developers to exclude features that automatically launch external software, but they should caution users when doing so, take care not to allow external applications to launch without player approval, and make it as difficult as possible for an attacker to dupe players into accepting.

P2P Communication

With the enormous amount of data transmitted between players in online games, it's no wonder that many of them allow direct P2P communication,

Game developers must ensure that homegrown multimedia processing engines

are rigorously tested for vulnerabilities.

often without user consent. In the P2P model, data flows directly from one player's computer to another, bypassing game servers altogether. In fact, game clients often seek P2P connections to reduce the load on central game servers and latency in data transfers. Vivox, for example, supports P2P connections for voice chat between two players—by removing the middle man, these online conversations become more fluid.

Direct P2P communication lets attackers deliver exploit payloads unfiltered and unseen by the server. When P2P communication is automatic and user consent isn't required, as is often the case, simply coming within an attacker's proximity in a virtual world could compromise another player's machine. Because P2P communication bypasses game servers, these attacks can't be blocked, and developers are forced to create emergency patches for the game client software.

To mitigate the additional threats P2P communication poses, game software should require player consent before creating a connection between peers or let players white-list specific peers for automatic connection. Furthermore, all information transmitted between players should be treated as suspect and subjected to stringent validation.

Case Study: Second Life

Released by Linden Labs in 2003, Second Life is an MMORPG that lets users interact, buy and sell land and goods with Linden dollars (which can be converted in-game to and from US dollars), and even create in-game content such as clothes, houses, and cars. Unlike games such as World of Warcraft, Second Life has no official objective—rather, it's a place to explore, make friends, and do things not possible in a player's "first life."

Feature: User-Created Content

To make Second Life as realistic as possible, creative freedom is available to players for designing their own clothing, homes, and vehicles. They can also embed multimedia creations of their own within game objects—for instance, a player can create a unique ambient-noise track that other players will hear while in a particular room or display a movie screen on the side of a wall that plays a favorite movie around the clock. Few limitations are placed on the creations possible within the virtual world.

Of particular interest to this case study is players' ability to create multimedia files and associate them with objects in the virtual world. When a player's avatar encounters an object in the game with an embedded media file, the Second Life viewer displays this content by rendering it with the optional QuickTime library installed on that player's computer.

These multimedia files aren't hosted on game servers but rather on public- or player-controlled servers. This relieves Second Life servers from handling this content's storage and delivery overhead and lets players update content easily and constantly. Instead of a player's game client retrieving all of the sight and sound information from the virtual world, the client is guided to pull the multimedia content directly from player-controlled servers.

Buffer Overflow: Full Compromise

When a vulnerability exists in the QuickTime library, it's possible to compromise the host machine of any player whose avatar approaches an in-game object embedded with malicious multimedia content. This happens as follows: first, an attacker creates a virtual object somewhere on his or her property in the online world and then associates a URL with the virtual object, indicating that a multimedia file is to be presented when this object is encountered. The URL itself points to a malicious media file with an embedded attack payload located on an attacker-controlled server. When a vulnerable player's avatar encounters this object in the virtual world, the malicious payload is automatically downloaded, processed by the underlying QuickTime library, and the host machine is completely compromised.

This attack requires a vulnerable QuickTime library to reside on the victim's machine; the library isn't installed with the Second Life software, but it's recommended and necessary to experience many of the virtual world's enhanced aspects.

Charlie Miller and Dino Dai Zovi developed an exploit with some specific properties: by the nature of the QuickTime vulnerability, the payload downloaded additional malicious code and injected it into the game client's running process.⁷ This code checked the amount of Linden dollars available to the victim and immediately transferred 12 of them to the attacker's account. The player's avatar was then forced to shout, "I've been hacked!" Finally, the attack stopped, and control of the game returned to its normal running state as if nothing had happened. At the time Miller and Dai Zovi demonstrated this attack, QuickTime was known to be vulnerable and had remained unpatched for several weeks (www.securityfocus.com/ bid/26560).

This attack demonstrates how an attacker can abuse the extensive feature list and freedoms granted to players and compromise their machines. By allowing players to create custom content of various formats, Second Life and other MMORPGs establish new attack vectors for malicious players to inject exploit payloads that either target game software itself or third-party libraries used for data processing. Moreover, providing players with the ability to host their own content grants attackers the same privilege, letting them bypass Second Life servers, communicate directly with their victims, and deliver unfiltered malicious content. Furthermore, attackers have the freedom to serve malicious content based on timing, the source IP address, or any other criteria of their choice.

After Miller and Dai Zovi demonstrated this attack, Linden Labs took steps to mitigate future attacks of this kind by requiring the latest version of Quick-Time to ensure older, unpatched versions couldn't be exploited. Additionally, players can now disable multimedia content (although it's unlikely they'll do so).

Case Study: Anarchy Online and Age of Conan

Anarchy Online and Age of Conan are MMORPGs created by Funcom. Age of Conan is set in a virtual environment modeled after the fictional universe of Robert E. Howard's *Conan the Barbarian*, and Anarchy Online, Age of Conan's predecessor, is set in a futuristic science fiction environment. In both games, players explore the online world and advance characters through a series of challenges and encounters. During their advancement, players interact collaboratively or competitively.

Gameplay is similar in both games. In-game communication and messaging are essentially the same, and syntax is identical, as are the methods for executing scripted commands. In fact, based on the similarities between the games and the existence of identical vulnerabilities, it wouldn't be surprising if Funcom reused the bulk of the code in both games.

Due to the games' similarities, their clients are susceptible to the same attacks, although the results vary slightly by vulnerability. In both games, attackers can read files from anywhere on the victim's machine and crash others' game clients. However, Anarchy Online is readily exploitable through its stack-overflow vulnerability, but Age of Conan is not.

Feature: In-Game Communication

In the game world, players communicate through various means, the most common of which is a textbased messaging system similar to that of chat rooms and instant messaging programs. For example, players can communicate privately by issuing the command

/tel1 [player_name] [message]

through the game's command-line interface or send a message to all nearby individuals by issuing the command

/say [message]

The recipient sees the sender's name and message displayed in the game's chat window; colors differentiate messages between channels, as well as whether they're public or private.

It's also possible to format messages with basic HTML to send hyperlinks, formatted text, and images; such hy-

perlinks can contain in-game scripting commands. For instance, a player could send a message containing the following HTML and in-game scripting language:

which appears to Alice in a separate message window as a hyperlink to a Web site. However, if the player clicks on the underlined text, http://securityevaluators.com, he or she unwittingly executes a script called fun_script.

Feature: Automated Scripts

Players can write custom script files to automate tasks that consist of a sequence of commands or to quickly "speak" long messages without typing them out fully—for example, a single script might contain the commands for greeting another player, such as saying hello, waving, and bowing.

Scripts are located in the game directory under a folder called "scripts." Each script is nothing more than a text file consisting of several commands, with each command represented by a single line. Those lines that don't begin with a "/" character aren't interpreted as commands, so the game client "speaks" them into the game world as text. A player can execute a script like a normal command by typing a slash followed by the script's name in the game's command line:

/[script_name]

Through a bug in both games' code, users can specify scripts located anywhere on the local machine, as opposed to strictly those within the "scripts" folder. This type of bug is commonly called a *directory traversal vulnerability* because it allows a command path to traverse the file system, leaving the original directory and accessing a file elsewhere in the system. This is accomplished by prepending any number of "up one directory" instructions (`..') when specifying the desired file's relative path to the "scripts" folder. For example, an attacker could issue the command

```
/../../../Users/<user>/Desktop/
fun_script
```

to execute a script called "fun_script" found on the user's desktop.

Directory Traversal Attack Example

By combining the directory traversal bug with the ability to mask local scripts as Web site URLs, not

only does an attacker have a potentially interesting cheat that can fool another player into performing ingame actions, but worse, the attacker can read confidential information directly off the player's computer.

All information transmitted between players should be treated as suspect and subjected to stringent validation.

Consider fooling a player into activating the following script:

/../../../Users/<user>/AppData/ Roaming/Intuit/Quicken/CONNLOG.TXT

This script causes the victim player to "speak" inside the game world lines of text from the specified Quicken log file or other files containing information about the user's bank accounts, the location of Quicken files and backups, and even exported report information with hard financial data.

The severity of a directory traversal vulnerability is apparent in the information leaked and how it can be used. Attackers can leverage these bugs to obtain specific information for subsequent attacks that take full control of the target computer, such as local IP addresses, passwords, or process information.

Buffer Overflow: Crash Example

A second vulnerability in both games is found in their script-parsing engines. If a script is loaded with a single line that's longer than 1,024 bytes, a stack buffer overflows, the executable in memory becomes corrupted, and the game client crashes. An attacker can easily exploit this vulnerability by issuing a message that executes the script

/../AgeOfConan.exe

Here, the script-parsing engine attempts to parse the 18-Mbyte file, and the game client promptly crashes. This alone presents an interesting cheat whereby players can fool each other into crashing their own game clients and temporarily exiting the virtual world. Additionally, because this is a stack-buffer overflow, it provides an opportunity for possibly overwriting the game client's stack with executable code and taking full control of the victim's computer. This is difficult in Age of Conan because it's compiled with stackprotection measures to prevent these specific attacks, but the same can't be said about Anarchy Online.

Feature: Launch External Browser

As discussed previously, messages communicated be-

tween players in the game world can contain HTML formatted links to external URLs. Until now, we've only talked about masking local scripts as links to Web sites. If used legitimately, clicking a link within the game launches an external Web browser and directs it to the specified URL accordingly. For instance, Alice might want Bob to visit her avatar's guild Web site, so she passes along the link:

However, this feature is the first step in an attack. In addition to displaying Alice's guild Web site, simply visiting the site silently puts a cookie on Bob's computer. This cookie contains the exploit payload.

Buffer Overflow: Full Compromise

Once the payload is downloaded onto the victim's computer, the attacker dupes the victim into clicking a second link, at which time the script is then loaded by the script-parsing engine, overflows the stack buffer, overwrites the executable in memory, and executes the exploit code. Alice needs only to convince Bob to click on the second link:

This second link again appears exactly like a link to a typical Web site, http://AliceRulezBetter.com. Once clicked, however, the payload is loaded and executed with the same permissions as the game software, and the victim's computer has been compromised.

The specific exploit developed and tested in our lab had the following properties: because it used a valid cookie, and the Anarchy Online script-parsing engine behaved differently depending on bytes within the script, the attack payload was restricted from containing most byte values, thus limiting the processor instructions available. To combat this, the attack payload consisted of a very small decoder built using only the available instructions and a more complex portion, encoded such that valid characters replaced invalid instructions. The decoder would first execute and decode the full payload onto the stack, which was subsequently executed. Malware authors typically use this technique when certain byte values are unusable.^{1,8} Once executing, the exploit downloads and runs additional executables from the attacker's Web server, steals the victim's Anarchy Online account information, uploads it to the attacker's FTP server, and forces the victim's avatar in the game world to don a bikini and start to dance.

The purpose of these specific actions in this proofof-concept exploit payload is to demonstrate the realistic threat these attacks pose. Malware authors can carefully craft them to install spyware and other monitoring software, report back with confidential information, and even directly affect activities within the virtual worlds that players enjoy.

After we demonstrated this attack, Funcom took the steps necessary to patch both Anarchy Online and Age of Conan so that attackers could no longer exploit these vulnerabilities. To prevent deceptive hyperlinking, developers can block or flag links that point to somewhere other than their descriptive text—for example, a link to http://www.example.com should have exactly the text http://www.example.com; many email spam filters use a similar heuristic to detect and flag phishing attempts. Alternatively, games can prompt users to confirm the action that links are to perform.

C heating in MMORPGs has a significant impact on the gaming community, but the impact of attacks that fully compromise players' computers is much greater. Online game developers should identify such problem areas with security-conscious risk assessments and allocate their resources accordingly. In future work, we hope to explore risk in these unique situations and ultimately help online game developers manage security for their games throughout their products' life cycles. □

References

- G. Hoglund and G. McGraw, *Exploiting Software: How* to Break Code, Addison-Wesley Professional, 2004.
- G. Hoglund and G. McGraw, *Exploiting Online Games*, Addison-Wesley Professional, 2008.
- I. Muttick, Securing Virtual Worlds against Real Attacks, McAfee, 2008.
- J. Reimer, "Virtual Plague Spreading like Wildfire in World of Warcraft," Ars Technica, 21 Sept. 2005; http:// arstechnica.com/news.ars/post/20050921-5337.html.
- R. McMillan, "Mac Hack Contest Bug Had Been Public for a Year," *PC World*, 21 Apr. 2008; www.pc world.com/businesscenter/article/144921/mac_hack _contest_bug_had_been_public_for_a_year.html.
- M. Daniel, J. Honoroff, and C. Miller, "Exploiting Android," Independent Security Evaluators, 25 Oct. 2008; http://securityevaluators.com/android/.
- C. Miller, "Virtual Worlds, Real Exploits," Network Security Newsletter, Apr. 2008.

8. "Writing ia32 Alphanumeric Shellcodes," *Phrack*, vol. 0x0b, no. 0x39, Phile #0x0f of 0x12.

Stephen Bono is a principal security analyst and partner at Independent Security Evaluators. His research interests include applied cryptography, secure software development, and RFID security. Bono has an MS in computer science and security informatics from the Johns Hopkins University. For his work in exploiting vulnerabilities in RFID payment systems and car immobilizers, he received the 2007 Award for Outstanding Research in Privacy Enhancing Technologies. Contact him at sbono@securityevaluators.com.

Dan Caselden is an associate security analyst at Independent Security Evaluators. His interests include computer security, game design, human-computer interaction, and technology education. Caselden has an MS in security informatics from the Johns Hopkins University. Contact him at dcaselden@ securityevaluators.com.

Gabriel Landau is a security analyst at Independent Security Evaluators. His main areas of expertise are applied cryptography and secure software development. Landau has a BS in computer science from the Johns Hopkins University. He was the 2007 recipient of the Michael J. Muuss undergraduate research award. Contact him at gabe@securityevaluators.com.

Charlie Miller is a principal security analyst at Independent Security Evaluators. He was first to demonstrate public exploits against Apple's iPhone and Google's G1 phone running the Android operating system. Miller won the CanSecWest Pwn2Own competition in 2008 and was hailed by Popular Mechanics as one of the top 10 computer hackers of 2008. He is a CISSP and has a PhD from the University of Notre Dame. Contact him at cmiller@securityevaluators.com.

