Editors: David Ahmad, drma@mac.com
Iván Arce, ivan.arce@coresecurity.com

Online Games and Security

Gary McGraw
Cigital

Greg Hoglund
HBGary

Online games have taken the computer world by storm. Gaming has always been (and remains) a prime driver of PC technology, with deep penetration into the consumer market. In the past 10 years, it has grown as quickly as the Internet, and can now be found in tens of millions of homes.

Along with the Internet's phenomenal growth comes plenty of adolescent growing pains, and as *IEEE Security & Privacy* readers know, these pains mostly concern problematic and pervasive computer security issues. Online games—especially massively multiplayer online role-playing games (MMORPGs)—suffer from such security problems directly; examples of MMORPGs include World of Warcraft, Second Life, EverQuest, EVE Online, Star Wars Galaxy, Lineage, and Ultima Online.

In this short introduction to MMORPG security, we focus on bugs involving time and state. We can expect to see more of such bugs as real-world software evolves to become more like game software.

Massively distributed systems

From an attack trends perspective, online games are noteworthy due to their architectures and the resulting security issues. MMORPGs are made of very sophisticated software built around a massively distributed client-server architecture. (By "massive," we mean more than 400,000 fat clients running complex game software connected simultaneously to banks of central servers. Because these games push the limits of software technology, especially when it comes to state and time (not to mention the real-time interaction of hundreds of thousands of users), they're particularly interesting as a case study in software security. In fact, MMORPGs are a harbinger of technical software security issues to come.

Modern software of all kinds (not just game software) is evolving to be massively distributed, with servers interacting with thousands of users at once. The move to Web services and service-oriented architecture (SOA) built with technologies like Ajax and Ruby follows hard on the heels of online games. What we learn in the MMORPG security world today is bound to be widely applicable tomorrow in every other kind of software.

Adding to the urgency of the security problem is the fact that online games are big business. The most popular MMORPG in the world, Blizzard Entertainment's World of Warcraft, has more than nine million users, each of whom pay US$14 a month for the privilege of playing.

Inside MMORPGs' virtual worlds , simple data structures have a value, mostly a reflection of the time gamers spend playing the game. Players also accumulate and trade virtual wealth, such as play money and pretend goods and services. Many virtual game economies have per capita GDPs greater than most small nations.[1] Not surprisingly, large numbers of direct connections exist between the virtual economies of games and the real economy. A well-developed middle market also exists, with the largest company, IGE (www.ige.com) earning more than US$400 million per year by acting as the real-world "middle man" for virtual goods. Virtual economies make excellent money-laundering systems, as well. Sheer economics has led to the emergence of a class of players more interested in wringing virtual wealth out of the game than playing the game itself. Entire industries of sweatshops with hundreds of thousands of workers in China now exist to do just that.[1]

Wherever money is at stake, criminals gather and linger. In the case of MMORPGs, cheaters have real economic incentive to break the games' security and accumulate virtual items or gain experience points for their characters. Many of these items, and even the characters themselves, are then sold off to the highest bidder.

Sophisticated hackers have worked the fertile fields of MMORPGs for years, some of them making a living directly from gaming (or cheating at gaming).

Time and state bugs in games

Bugs and flaws in software account for a majority of computer security risks. (Other work provides discussions about the difference between bugs—found at the implementation level in software—and flaws—found

Race conditions 101

Let's say that Alice and Bob work at the same company. Over email, they decide to meet for lunch, agreeing to meet in the lobby at noon. However, they don't agree on whether they meant the lobby of their office or the building lobby several floors below. At 12:15, Alice is standing in the company lobby by the elevators, waiting for Bob. It then occurs to her that Bob might be waiting for her in the building lobby, on the first floor. Her strategy for finding Bob is to take the elevators down to the first floor to see if Bob is there.

If Bob's there, all is well. If he isn't, can Alice conclude that Bob is late or has stood her up? No. Bob could have been sitting in the lobby, waiting for Alice. At some point, it could have occurred to him that Alice might be waiting upstairs, at which point he would have taken an elevator to check. If Alice and Bob were both on different elevators at the same time, they'd pass each other during the ride.

When Bob and Alice each assume that the other one is in the other place and both take the elevator, they've been bitten by a race condition. A race condition occurs when an assumption needs to hold true for a period of time, but actually might not; whether it does or doesn't is a matter of exact timing. Every race condition has a window of vulnerability—that is, a period of time during which violating the assumption will lead to incorrect behavior. In Alice and Bob's case, the window of vulnerability is approximately twice the length of an elevator ride. Alice can step on the elevator up until the point at which Bob's elevator is about to arrive and still miss him. Bob can step on the elevator up until the point that Alice's elevator is about to arrive. We could imagine the door to Alice's elevator opening just as Bob's door shuts. When the assumption is broken, leading to unexpected behavior, then the race condition has been exploited.

at the architectural level.[2]) This is true for banking applications, and it's also true for online games. So many security-related bugs exist in software, and they're so pervasive, that software tool vendors have created special tools just to look for them. Complicating the situation somewhat, scientists have published reams of papers on taxonomies for bugs, all of which disagree with each other.

The most interesting and tricky security bugs involve time and state problems tangled in complex trust models. These problems arise because complex system state must be shared among many distributed processors with different levels of trustworthiness. This category of bugs is particularly relevant because it's also a harbinger of what to expect in the future as SOA catches on. Timing and synchronization problems are already a major issue in real-world software—in fact, security practitioners have discussed them for more than a decade. But as massively distributed systems become more common and trust models get more complex, problems with synchronizing and tracking state will likewise become more common. Thanks to online games' overall design, they're rife with time and state problems. Add to this the notion that servers shouldn't implicitly trust the game client itself, and you have a formula for disaster.

In an online game like World of Warcraft, the biggest architectural challenge is sharing state information about the game with hundreds of thousands of client programs all at once. When so many thousands of client processes interweave on a common server, over the network, in real time, state confusion attacks pop up like mushrooms after a rain. In fact, race conditions (which we'll examine in more detail later) and other problems with state are the primary source of bugs in online games. They're exacerbated by laggy network connections (which tend to warp time in interesting ways … sort of like black holes). By their very design, large online games require vast amounts of data storage distributed across many servers. Some servers store accounting information for billing, others store player statistics and inventory, and still others store the current state of an online world. However, the most serious problems crop up when state information is "cracked off" and shipped to untrusted game clients. Most game designers have chosen to ignore the "insider threat" of a game client gone bad. As a result, some state manipulation attacks can be as simple as directly hacking the parameters controlled by the game client software. (For more on insider threats like this see www.darkreading.com/document.asp?doc_id=131477&WT.svl=column1_1.).

Technically, most MMORPG's don't really involve single monolithic online worlds, but rather have many duplicate "shards" of what only seems to be a "world." Copies of the online world that World of Warcraft uses, for example, tend to limit the number of users to 50,000 players per server. EVE Online (www.eve-online.com) is a single online world, but that virtual world is distributed across such a large universe (think solar systems) that no one server ever gets overloaded.

The problem with multiple world shards is boundaries. Race conditions are found on the borders between software states—such as the state of being logged in and the state of being logged out. If everything happens atomically—that is, if you go from being logged in to being logged out in one fell swoop without gazillions of steps, things can go all right. But if multiple steps are involved, and they aren't protected by semaphores in "critical sections," trouble can crop up.

The sidebar "Race conditions

101" gives a simple example of a race condition that can happen in the real world. Who knows, maybe this has even happened to you?

Bugs involving time and state … are the kind we can expect to see more of as real-world software evolves to become more like game software.

Telehacking: A simple state manipulation attack

Virtual worlds in online games seem real, but in fact, they're only as solid as the models on which they're built. Characters can travel through the game in many different ways: flying, swimming, teleporting, running, and walking are just some standard options. Many games even have magic spells to enhance all your travel needs.

The state machine that manages travel is usually held in the client software, along with almost all of the 3D object interaction. By altering the client, you can alter how you travel. For this reason, many common game exploits modify the way the client program handles travel.

One simple and elegant example of state manipulation involves finding and resetting the player character coordinates in an online game. As in many MMORPGs, your character's coordinates in World of Warcraft are part of the state information controlled by the game client. Instead of walking around the world (or moving normally), a cheating gamer can teleport by directly manipulating the location parameters in memory.

If the character location coordinates set in this manner aren't very far from the current location, this kind of action can even appear to be normal movement. Telehacking uses the direct player character location-setting method, but it usually refers to teleporting over vast regions of the game map, as opposed to small teleports to correct for position within a few virtual yards. (You can find the code for telehacking in World of Warcraft in our book *Exploiting Online Games*; www.exploitingonlinegames.com.) Overwriting a single byte in World of Warcraft's client code can enable a character to climb mountains—or even climb straight up walls. Of course, gamers use this hack to get into places they're not supposed to get into.

Using bugs to confuse state boundaries

Telehacking in World of Warcraft is a very simple example of state manipulation, but time- and state-related problems can be much more complex. Plenty of software-related state boundaries are rife with timing problems. Ultimately, the biggest underlying issue is one of trust: if the client software is trusted to manipulate state properly and it is, by definition, under a potential attacker's control, security problems are the result.

One of the most obvious software-related boundaries involves databases. Transactions that involve multiple databases are often susceptible to race conditions. Because virtual worlds are distributed across many servers, doing things like switching from one virtual dungeon to another or flying from one virtual continent to another often causes a player to be handed off from one server to another.

This kind of switch is a normal event in a game, and game company quality assurance (QA) has certainly tested it with a defined test plan—not to mention that plenty of players have actually done the switch many times. But here's what happens in many QA shops. The test plan says something like, "inventory is supposed to remain constant when a player does activity47 at portal68." Then a tester logs in, goes to portal68, performs activity47 and checks to see if everything is fine (such as, say, player inventory). This is what's known to software testers as a functional test. The problem is that this test is both boring and conventional!

You see, attackers don't often do what they're supposed to do; instead, they focus on trying things that programmers never anticipated. They do the unexpected, sometimes with insane results.

So here's an idea. Instead of gracefully walking through portal68, make sure you log out of the game while you're doing it. Pull the Ethernet cord out of the wall. Kill the game client with the task manager. When you're done, log back in and find out if anything juicy happened. Did you make it to the new continent or are you on the original side of portal68? What's the state of your character?

Let's step through some possibilities, again thinking with our black hats on. Let's say you end up on the original side of the portal. What would happen if you gave some money to a player friend of yours just seconds before you killed the process, and that friend continued through the portal like normal? When you log back in this time, back on the original side of portal68, check your wallet. Was the money taken from your wallet, or did it reset itself along with your location? If it did reset (back to the original pre-give-some-away amount, does your friend also have some money on his side of the portal? If your money has doubled, you've

found a duping bug—one of the most coveted bugs of all time in online gaming. Nothing like copying inventory for free!

In World of Warcraft, several bugs like this exist around entrances to "instance dungeons." That's because *instances* are just like continents or any other location in World of Warcraft—they're handled on specific servers, and as players join an instance, they're in reality a glob of data being transferred from one "back office" server to another.

In general, a single instance server is responsible for serving all particular instances of a given dungeon. For example, all deadmines instances run on the same deadmines server. However, because this is a very popular quest in the game, the server tends to become overloaded and laggy—an ideal condition for trying to crowbar a race condition out of a game.

As it turns out, botnets are very effective ways to induce lag on the net. Thus, botnets pose a serious security risk for online games. By using a botnet to cause a given game server to lag, an attacker can set things up for a more efficient race condition exploit.

As an example of how state confusion attacks and broken trust models relate to future software systems, consider security issues surrounding Google's Desktop Search program. In February 2007, the software security firm Watchfire (since acquired by IBM) announced an attack methodology against Google Desktop that provides evidence of the trusted state problem in modern software architecture (http://download.watchfire.com/whitepapers/Overtaking-Google-Desktop.pdf).

The attack that Watchfire describes works by misusing the trust model set up between Google's Web site and Google Desktop. Just as in online games, no malicious hacking or binary payload injection is required. The problem results from an architectural misunderstanding in what should trust what in the model.

These are the kinds of security problems that result from modern software architectures when trust model boundaries are confused. As long as software designers don't explicitly consider the case of client software misbehavior in their designs, we're in for a whole lot of hurt, both in the virtual worlds of gaming and in the real world. ▢

Acknowledgments

Some material in this article is used by permission from Exploiting Online Games *(Addison-Wesley, 2007) and* Building Secure Software *(Addison-Wesley, 2001).*

References

1. G. Hoglund and G. McGraw, *Exploiting Online Games*, Addison-Wesley, 2007, pp. 67–73.
2. G. McGraw, *Software Security*, Addison-Wesley, 2006, pp. 18–19.

Gary McGraw *is Cigital's chief technology officer. His real-world experience is grounded in years of consulting with major corporation and software producers. McGraw is the author of* Exploiting Online Games *(Addison-Wesley, 2007),* Software Security: Building Security In *(Addison-Wesley, 2006),* Exploiting Software *(Addison-Wesley, 2004),* Building Secure Software *(Addison-Wesley, 2001), and five other books. McGraw has a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. He is a member of the IEEE Computer Society Board of Governors. Contact him at gem@cigital.com.*

Greg Hoglund *is the founder of HBGary and runs rootkit.com. He is coauthor of* Exploiting Online Games *(Addison-Wesley, 2007),* Rootkits: Subverting the Windows Kernel *(Addison-Wesley, 2005) and* Exploiting Software *(Addison-Wesley, 2004).*

Interested in writing for this department? Please contact editors David Ahmad (drma@mac.com) and Iván Arce (ivan.arce@coresecurity.com)

$29

New Lower Subscription Price!

IEEE SECURITY&PRIVACY

Subscribe to our magazine today for only $29— our lowest price ever!

You'll receive 6 issues of today's leading-edge, peer-reviewed software development information.

Ask us how you can get this great deal on *IEEE Security & Privacy* magazine!

S&P is the premier magazine for security professionals. Every issue is packed with tutorials, best practices, an expert commentary on:

- attack trends
- cybercrime
- security policies
- mobile and wireless issues
- digital rights management
- and much more.

Visit us at www.computer.org/services/nonmem/spbnr

IEEE computer society 60th anniversary