Securing Virtual Worlds Against Real Attacks

The challenges of online game development

By Dr. Igor Muttik
Table of Contents

Executive Summary ............................................................................................................. 3
The Growth of Online Gaming ............................................................................................ 3
The Dynamics of Stealing Data .......................................................................................... 5
Client-Server Architecture and Scripting ........................................................................... 7
Three Pillars of Security .................................................................................................... 10
Predictions .......................................................................................................................... 15
Conclusion .......................................................................................................................... 19
Securing Virtual Worlds Against Real Attacks

The challenges of online game development

By Dr. Igor Muttik

Executive Summary

For many people, computing has become synonymous with the pursuit of fun and enjoyment gained from online gaming. The number of online games, especially multi-player online role-playing games (MMOGs), has seen rapid growth in recent years and so have the security and data issues. Online gaming is starting to suffer from real-world problems—theft of identity and virtual assets, extortion, and even terrorist attacks.

MMOGs are supported by virtual online communities, which are sometimes referred to as metaverses or digital worlds, where people compete, fight, buy, sell, trade, study, travel, and do many other things that people do in real life. It is not surprising that online gaming is beginning be plagued by almost all the problems of the real world—theft of identities and virtual assets, extortion, and even virtual terrorist attacks in these venues are becoming more and more common! Metaverses grow their own economies, and virtual currencies are converted into real money and back, so it is only natural that virtual profits, too, get targeted by cybercriminals. If Willie Sutton, the accomplished twentieth century American bank robber, were alive today, he probably would have an avatar and would be writing password-stealing Trojans.

In this paper, we provide a brief overview of what life is like in these virtual worlds, with statistics on malware and how it relates to gaming and the real world. Online computer games are large, intricate programs that require permanent Internet connections, so exploitation of vulnerabilities in an online game—either on a server or a client—could be used to steal user data from both real and virtual environments. Since the beginning of this century, we have seen significant growth in advertising and shopping within games. Here we will discuss MMOG-based spam, phishing, adware, and spyware.

With the accelerated growth of malware in the gaming environment, it’s apparent that certain design decisions made by game developers can make an online game either more secure or more vulnerable to hackers. We’ll discuss issues surrounding plugins and scripts on both server and client sides and explore the security implications of these approaches. And we will touch on how an open-source approach may affect the security posture of online games. Finally, we offer predictions about future trends by analyzing current market and technological shifts.

We expect that major gaming vendors are aware of the security issues related to online gaming; nonetheless, those developers can benefit from reading this paper. The overall audience for this article is much wider and includes emerging gaming software developers, security professionals, and even the general public. This paper will offer interesting perspectives for anyone interested in the subject.

The Growth of Online Gaming

The number of online games and their subscribers is growing at an extraordinarily rapid rate. Especially popular are MMOGs such as World of Warcraft (WoW) and Lineage. There’s even a web site that specializes in counting the number of subscribers to these games: http://www.mmogchart.com. Figure 1 covers the years 1997 to 2006 and shows the total number of subscribers and the distribution among major game vendors. We advise readers to take these numbers with a grain of salt, as many people register multiple times. Nevertheless, the explosive trend is undeniable.

---

1 We will use the single acronym “MMOG” to stand both for Massively Multi-player Online Game and for Massively Multi-player Online Role-Playing Game (also known as MMORPG).
Securing Virtual Worlds Against Real Attacks: White Paper

Data based on counting subscriptions is supported by market analysis. According to one study, the online gaming market grew 288 percent from 2002 to 2005. Worldwide revenues from online gaming exceeded $1.1 billion in 2006, and by 2009, the revenues are expected triple, according to market research firm Parks Associates. The biggest share is currently MMOGs; predictions see no change in that position well beyond 2009.

Big companies are investing in gaming too—both Microsoft and Google, for example, have acquired companies involved in in-game advertising. Intel bought the gaming video engine Havok. (Havok 1.8.3 is used in Second Life.) And Sony has started its own virtual world called Home. There are rumors that Google is developing a virtual world of its own.

People spend a lot of time playing. More than 25 percent of gamers play for more than 30 hours every week. In most games, players collect and produce some sort of virtual commodities. These can be virtual objects (weapons, gear, clothes, property, furniture, music), money, and relationships (you can be a lord of a castle with many subordinates and even get married virtually). Even names of characters are valuable and can be resold at a profit—which is a virtual equivalent of cybersquatting (registering domain names to resell in the future).

Because players spend so much time and effort doing this, they value the database records representing their virtual assets as much as they value real objects. It’s no surprise that people are ready to take shortcuts and pay real money to get advanced virtual objects to avoid boring routine work, commonly called “grinding.” So, naturally, we see secondary markets for virtual commodities—for example, virtual currencies are sold on eBay. Figure 2 shows the rates for “adena” (the currency in Lineage2) for different game servers.

Gamers track the rates of virtual currencies almost as carefully as real money.

Virtual objects are traded in two connected markets—fully virtual and real. The intertwining of real and virtual markets is growing, and there are now real shops in virtual worlds (where you can buy real goods for virtual money). Both of these markets attract criminal elements.

Gaming is extremely popular in the Asian-Pacific countries. Consequently, we should expect statistics from this region to give us a hint about future trends for the rest of the world. According to a study in Taiwan, 37 percent of criminal offenses are related to online gaming. We can see that the level of penetration of virtual offenses into real life is alarmingly high. Many of the players are fairly young, which is reflected in the statistics showing that most offenders belong to the 15-to-20-year-old bracket.

Many banks have already announced their plans to open virtual branches—a move that would eventually combine all the known risks of Internet banking with the risks of virtual identity and data theft. But first, let’s look at the statistics.

2 http://www.researchandmarkets.com/reports/339096/on_line_and_mobile_gaming_in_the_united_states.htm
4 http://www.eurogamer.net/article.php?article_id=83422
5 http://news.bbc.co.uk/2/hi/technology/6429039.stm
10 http://dev.hil.unb.ca/TTexts/PS3/pdffichen.pdf
and motivation of traditional data theft, and we will be able to better understand whether virtual assets are at risk.

The Dynamics of Stealing Data

Growth of data-stealing Trojans

We have observed rapidly growing data theft since about the year 2000. (See Figure 3.) In a typical attack, the data-stealing programs (or “stealers”) see and record users’ authentication data (IDs and passwords) along with the IP addresses or the names of the servers they use and transmit this information to the attackers. This is usually done by using a keyboard logger to record all keystrokes. In more sophisticated attacks, the web forms are captured, as are mouse movements and even screenshots. All this information is then transferred to the attackers. (The first stealers used anonymous email accounts, such as Hotmail, to receive stolen data but contemporary ones hide behind outgoing traffic—inside web requests, for example—a much smarter approach.) Sometimes today’s stealers transmit additional information, such as the amount of money in a banking account, or—for a gamer—the level of a player and the contents of their inventory. Later, the attacker can log into the compromised account and retrieve anything of value.

Typically, when a gaming account is compromised, attackers will convert the objects they steal into virtual currency—and then convert the virtual currency into real money. Alternatively, if their motivation was more for fun than finance, they might use the virtual equipment and money for their own game play. (This alternative scenario would apply only to gamers who play on the same server, so such attacks are more likely to be targeted. The attackers might deploy Trojans via in-game messaging or in a dedicated forum.) Most of these malicious programs are distributed using social engineering tricks that lure people into running them.

The consensus of security experts is that financial motivation is the primary force behind the alarming growth of data theft. The distribution of targeted password-stealing applications active in 2007 is provided in Figure 4.

![Figure 4. Gaming Trojans and Trojans targeting online banking are about equally common.](http://www.avertlabs.com/research/blog/index.php/2007/07/16/password-stealers-targeting-games-are-growing-more-than-ever/)

Trojan programs that steal banking data are the most common type of password-stealing malware these days. After that, stealers that target online games come in second; their overall quantity in 2007 almost exceeded the number of banking Trojans.12

We must also stress that a raft of general-purpose malware—backdoors, password stealers, keyloggers, and form grabbers—can be successfully used to steal information related to online gaming. By counting only malware samples that explicitly refer to online games, we understate the numbers of Trojans that plague gamers. Online gamers must come to understand the value their accounts and data represent to hackers—real-world money.

Phishing

Phishing is also a form of data theft, even though normally there is no malicious code involved (other than the spam programs that distribute the phishing emails). It should come as no surprise that phishing has been used to relieve gullible players of their virtual assets. Indeed, there are some examples of this.13

---

At the time of writing this paper, perhaps the most intense spamming runs that were related to W32/Nuwar (also known as Stormworm), used the gaming theme. The bad guys created a web page offering “free” games. Links to it were widely spammed, but clicking anywhere on this web page led visitors to malware in the form of ArcadeWorld.exe.14

Parasitic and polymorphic viruses targeting MMOGs
Trojan programs require a social-engineering trick or lure to work. Otherwise, they would never get installed. To overcome this serious limitation, the evildoers attempt to use replicating code (a virus) to deliver the data-stealing payload. Of course, when a virus spreads, it can propagate to the systems that are used for gaming. The virus can detect this and activate its data-stealing payload.

Employing parasitic techniques also makes removing malware more complicated. There might be a slight delay (from only a few minutes to several hours, depending on the complexity of the virus) in the release of an anti-virus update that cleans the malware, so the extra time is a bonus for virus writers.

Let’s look at four notable virus families that have payloads related to online gaming. (There are many others.) Figure 5 shows the number of variants in these four virus families over time:

- W32/HLLP.Philis: A prepping virus. Appearing in early 2004, it was written in Delphi and downloaded malware that stole login details for Lineage and Legend of Mir.15
- W32/Detnat: A polymorphic virus16
- W32/Bacalid: Another polymorphic virus17
- W32/Fujacks: In 2006, we saw a wave of W32/Fujacks that targeted Lineage, Legend of Mir,18 and the popular Chinese game Zhengtu.19 Members of the W32/Fujacks family have significant code similarities with W32/HLLP.Philis, although we separate them due to the modifications in the replication mechanisms. Nonetheless, both families could, in principle, be merged for the purpose of counting.

We can make the following conclusions from Figure 5:

- The years 2004 and 2005 were slow on the parasitic front, but in 2006 several infectors appeared
- Two attempts at creating polymorphic viruses in 2006 probably produced too low a return on investment. (It takes a great deal of effort to implement a polymorphic engine.)
- Perhaps the bad guys realized that a more successful strategy is to release lots of new nonpolymorphic viruses
- New, more advanced families replaced old ones (new propagation methods in Fujacks versus Philis)
- There is a significant decline in 2007 that started around May, almost certainly due to the arrest in China of Li Jun, the author of W32/Fujacks. In September 2007, he was convicted and was sentenced to four years in prison.

Parasitic viruses written to steal gaming data, of course, affect not just people who play games. They are the bread and butter of the security business, so, naturally, anti-malware technologies react quickly when they detect them. The replicating nature of these viruses draws attention to them, so their lifespan is significantly shorter. This could be an additional reason for their decline in 2007. In general, it is a lot more likely that a static data-stealing program (especially used infrequently) would slip under the radar of security companies. On the other hand, self-propagation ensures that more computers get compromised, and unprotected systems will keep pumping out infections even after anti-virus protection is available for most users.

So far we have discussed ordinary parasitic viruses. A scarier scenario would be the appearance of an auto-executing worm. Such worms can exploit zero-day vulnerabilities in
client or server software and propagate similarly to, for example, W32/CodeRed20 or W32/SQLSlammer.21 It is not likely that a worm can take advantage of vulnerabilities in both client and server, so we can assume it will have to hop between only clients or only servers. A server-only worm would have just a handful of machines to propagate to (more later about peer-to-peer server clusters), and it would be quickly detected and countered with a fix. We imagine it would make a lot more sense for the bad guys to covertly use any zero-day exploit to steal or modify data rather than use it for building a worm (whether server- or client-based). That could be one reason why we have not seen explosive worms since CodeRed and SQLSlammer: If a vulnerability suitable for creating an auto-worm were found, it would not likely be used for that. Simply selling a zero-day exploit on a black market can yield $50,000 to $120,000.22

We can conclude this chapter by stating the obvious: Even though trading virtual money for real money (known as “real money trading” or RMT) in most virtual environments is considered a violation of the terms of service (TOS), we must assume that it will always be possible to find a way to convert virtual commodities into hard cash, and vice versa. Even if it were illegal (by law or according to the TOS), there would always be a black market for virtual commodities. And automated ways of extracting money from online games (password-stealing malware, bots,23 and similar cheats) would have an advantage over manual methods. So, unless we find ways to control this, we should probably expect as much growth in online threats as we have seen with other targets of malware. Now let’s have a look at how we can make online games more resistant to malware.

Client-Server Architecture and Scripting

In this chapter we will have a look at best practices in implementing scripting in general, and we’ll also focus on two scripting languages. First is Lua, because it is quite common and because it is used in World of Warcraft. Second is Linden Scripting Language (LSL) because it is a very rich scripting language used in Second Life and because this environment offers enormous flexibility in supporting commerce, advertising, and creativity. Therefore, we should expect many standard attacks that already occur in real life (phishing, spam, viruses, and more) to materialize there first.

Dangers of scripting

The flexibility of online games comes from the fact that they are using a client-server approach. On the server, there is a database that ensures persistency of the virtual world. There are also rules that define events and transformations related to the objects held in the database. All essential information should be stored on the servers (positions, contents of inventory, status of characters, and other data) because anything stored on the client can potentially be tampered with (and not just data stored in files—memory contents or flying network packets can be modified just as easily).

Thus, clients are usually designed to simply render the representation of the virtual world using pretty pictures after receiving all the necessary data from the server. Frequently the server-side rules are kept in form of scripts for added flexibility.24

There are four basic kinds of scripts (sometimes simple user scripts are called macros). Each has its own security issues:

• Vendor’s scripts on the server are isolated from users, and so are the safest of all
• User’s scripts on the server are the most dangerous because they need full sandboxing and strict control of what they can access and do
• Vendor’s scripts on the client, in essence, are the extension of the client, so they need the same level of checking as the game binaries, to prevent users from tampering with them. (In any case, having them in plain text for the users to read would not be a good idea. Some sort of p-code would be better.)
• User’s scripts on the client are complex. User access should be tightly controlled—either users get no access to server objects, or there is a strict access policy.

The golden rules to ensure safe scripting are:

• Scripts must not have access to other scripts (especially “write” permissions); otherwise, virus infections are possible
• Scripts should have as little persistency as possible; otherwise, viruses can survive
• There should be no auto-execution; otherwise, viruses can activate easily (automatically or on a certain event)

20 http://vil.nai.com/vil/content/v_99142.htm
21 http://vil.nai.com/vil/content/v_99992.htm
23 Note: A “bot” in online gaming refers to a program that plays a game in lieu of a human playing the game. A “bot” in security-industry terminology is a malicious, remotely-controlled program that is a part of a bot net, which comprises several controlled bots. Both terms have their origin in the word “robot.”
24 Note: By “script,” we mean any frequently modified code that can be dynamically changed, whether it is sandboxed, compiled, interpreted, or executed in native code (for example, via just-in-time compilation).
There should be no access to dangerous objects (email, instant messaging, Internet access) or, if necessary, such accesses can be throttled (for example, less than one email in 30 seconds is allowed) or require a recipient’s consent (for example, recipients can register on the server who can communicate with them); otherwise, viruses can propagate more easily and cause harm and disruption.

If these design decisions are taken into account, it will become very difficult to write effective viruses. Virus infections would only be able to take advantage of vulnerabilities in software that would allow them to achieve persistency, auto-execution, and propagation. Fortunately, finding a vulnerability is a highly non-trivial task and requires a lot of knowledge and luck. Plus, once one is found, it is not always possible to successfully exploit it so that code execution occurs. Frequently, it is only possible to cause a crash. This is, of course, still a security problem, but not as serious as a malware infection.

Lua scripts
The very first recorded case when scripting for online games was used to implement viruses25 was with scripts in the Lua language.26 These viruses were implemented for Garry’s mod (also known as GMod) for Half-Life2. The reason they were possible was that within GMod, some scripts are copied from server to the client and executed there. Persistency was achieved as scripts were saved as files. Plus, it appeared that GMod automatically executes scripts with “res” in the extension if they were in “maps/” folder. So, the viruses, when they received control, copied themselves there and renamed themselves from “.lua” to “.res.”

Lua is widely used for many other games27 because it is free of charge. WoW uses it,28 as do Nintendo game consoles and Warhammer Online (apart from already mentioned Half-Life2). In WoW, Lua scripts can be used to customize the interface of the game. The customizations (files with .XML and .LUA extensions) go into the following three folders: Interface, WTF, and WDB.

Accidentally created viruses
We cannot resist mentioning an incident in 2005 when a bug in a game script for WoW caused a serious epidemic (“corrupted blood”) in a virtual world that killed scores of weak characters.29 This happened after the developers of the game introduced a new virtual disease. The source of this disease was a monster that could cast corrupted blood on victims that dared to attack it. The area inhabited by the monster is only visited by players of rather high level, so only such characters would be able to confront the monster. On top of that, if infected, only very strong characters would be able to survive until the infection was over. While carrying corrupted blood, they would not be able to infect many others because only relatively small parties of players confront this monster at any one time. Thus, the number of infected targets would be limited to the members of a party that attacked the monster. Because of these limitations, a natural assumption would be that the disease cannot occur anywhere far from the source of infection. This is where the miscalculation occurred. The oversight was such that some characters had virtual pets that could be summoned and unssummoned when necessary, and the disease could infect them, too. What happened is that someone’s pet was infected, and it was unssummoned before it died. That preserved the state of infection, so when the pet was later summoned back into an area densely populated by low-level characters (in a village or a city), there was enough time for the pet to infect some players before it died. And then an uncontrolled epidemic (and a lethal one at that) swept through the game environment. Scores of characters died, and some villages were completely wiped out.

It took game developers two days to fix the problem. It is curious that among WoW players, this incident is regarded as a rather exciting event.

That would be the first case of a virtual virus and, hopefully, the last. In a way, it was a case of an “accidentally” created virus, and the problem was caused by vendor’s server-side script. Ironically, according to our classification, it is the safest kind of scripting—and still it was the source of a viral infection. This underlines the fact that security, and especially replicating properties, should be treated with utmost care.

There is only a single known case of anything remotely similar in real life so far—a tool called Already.30 This tool is a short 70-byte program that updates its own memory copy with the current date and time and then writes it back to the disk. It compares the current date with the timestamp saved in its own body to determine if it was...
already executed today (hence the name, already.com) and returns the code that can be used in BAT scripts to achieve one-time execution on a specific date. If the program were executed from a folder different from the current one, then the program would create a modified copy of itself (this happens once, but if it is executed in another folder, it will create a third copy and so on) and so should be classified as a virus. If used from the current folder and under its original name (which is most likely how it was supposed to be used), the program is not viral but just modifies itself. So the replication property was a side effect of self-modifying code that had to save the modification. The same function can be achieved by using a configuration file but, admittedly, this will require a more complex algorithm; saving and reading both code and data in one go makes the program simpler.

The moral of the story is that when there is a buggy implementation of a curious game feature, you just might get a plague! A design mistake combined with a useful tool can turn into a computer virus. Game developers (like all other software developers) should never lose sight of security when coding software.

**LSL scripts and control of unsafe operations**

Scripting in some online games is very developed, and one of the most interesting examples is the exceptionally powerful LSL. Second Life was developed to allow players to create their own objects and define their behavior, thus giving users the tools to create the scripts that essentially define local game rules (normally only game developers would be able to do that). This exceptional flexibility makes LSL very interesting from a security perspective.

LSL is an event-driven, C-like language that gets compiled into byte-code and executed in a virtual machine on the Second Life server (so, according to our classification, this is the most dangerous kind of script). There is no explicit persistency, but scripts can be attached to in-game objects (to be precise, scripts are attached to so-called prims, many of which can be linked into an object), which can be saved and reused. In LSL you can even send emails (this function involves a 20-second delay—no doubt to prevent spamming, thanks to Linden Labs!). LSL has advanced networking functionality and supports XML-RPC (with a three-second delay to prevent denial-of-service, or DoS, attacks). Plus, there are also **llHTTPRequest** and **llLoadURL** (also throttled at one and 10 seconds, respectively). HTTP requests generated by LSL scripts to URLs out of the Linden Labs infrastructure get multiple “X-SecondLife-” headers automatically inserted. This lets Second Life interact with the Internet and opens many possibilities for building infrastructures that would interact with Second Life.

With LSL scripting, you can create really complex objects and video simulations. It was even used to create a visual simulation of a terrorist attack in Second Life (Figure 6).

The background of this event is that long-time Second Life player Marshal Cahill is campaigning to give voting power to players, so that they can decide the future of Second Life. He organized the “Second Life Liberation Army” to work towards that goal. Finally, he and his followers detonated a bomb (a complex LSL script, really) near two in-game stores (American Apparel and Reebok) to draw the attention of the media and Linden Labs.

A curious fact is that LSL functions related to visualizing explosions (llMakeFire, llMakeExplosion, llMakeSmoke) suddenly were deleted by Linden Labs. (Figure 7 shows these functions crossed off the list.) But there’s no conspiracy behind this; these functions were simply replaced with a better and more generic alternative—llParticleSystem.

![Figure 6. Visual effect of a script simulating an explosion in Second Life.](image)

![Figure 7. A list of LSL functions indicating “deprecated” (replaced) entries for explosions, fire, and smoke.](image)

32 In LSL scripts, all standard functions start with the double “ll,” which stands for “Linden Library.”
The point is, though, once scripting power is given to users, it must be tightly controlled. The most dangerous functions (email, IM, networking, messaging) involve a delay in scripting that is the main mechanism Linden Labs uses to reduce potential problems. A table in LSL Wiki lists the throttling limits that were imposed in LSL on certain functions deemed dangerous.

If you use Second Life® llRezObject and llGiveInventory, a new object can be created (a script attached to one of the prims will replicate the object). To control creation of too many objects, Linden Labs used more complex logic than trivial throttling. Both the frequency and quantity of created objects are measured, and a 12-second delay is introduced if a threshold is exceeded. The idea is to allow functions that support replication but throttle the yield of new objects.

For Linden Labs, such replicating scripts pose a DoS problem because a server with a quickly growing number of objects is likely to become very slow. Another interesting method employed to control how LSL scripts interact with objects is the concept of an object’s “energy.” Massive objects acquire energy slowly, while tiny objects acquire it quickly. At the same time, only when the energy level is 100 percent can scripts effectively affect an object. Objects with a level of energy close to zero are almost unaffected by scripts. It seems obvious that “energy” was invented for security and stability reasons.

To make scripts benign, all you need to do is remove persistency. If the persistency property is missing or throttled from user scripts, we can significantly reduce security risks (this is the case with many game consoles that do not store programs on writable media). Vendor’s scripts (scripts downloaded from the server for local execution on the client) can be verified using digital signatures and executed only if they are authentic. This checking would create a barrier against tampering with such scripts, especially if verification code in the game’s binary is protected with mechanisms such as GameGuard or Warden.

If we look at the history of security risks associated with scripting in general, we can see what makes scripting less prone to abuse—the lack of file, media, and storage access support (no persistency in files or other media) or access to dangerous objects (for example, the address book, raw memory, TCP/IP stack, other scripts, or VBA modules). Most of these problems can be addressed by proper sandboxing. (WoW does that for Lua just as Second Life does this for LSL.)

Open-source and proprietary scripting in MMOGs

There is a visible trend toward using open-source scripting in gaming more frequently. This is supported by open-source efforts like PyGame, which is a Python-based game library that is capable of talking directly to DirectX. Many other game-related scripting solutions are listed in the white paper Casual Gaming. As we mentioned previously, Lua is also an open-source project. Linden Labs plan to switch Second Life scripting to Mono—an open-source incarnation of .NET.

Meanwhile, an adequate balance between convenience and security can be achieved by using custom object models. Incompatible object models will ensure that the interoperability of such scripts is reduced. For example, in one game, a script can refer to the script attached to the fourth item in the inventory as “Inventory[4].Script,” but in another game, the same scripting language may have to select an object and then use “CurrentObject. RootElement.Script.”

For more about the security implications of open source, take a look at “Paying a Price for the Open Source Advantage,” July 2006, McAfee® Avert® Labs Sage Journal, Volume 1, Issue 1, which is available for download at http://www.mcafee.com/us/threat_center/white_paper.html.

Three Pillars of Security

Next, we will explore how technology solutions, economic measures, and human factors affect security and how none can fully succeed if the others are ignored.

Technology

Proper security measures are very important, but they can never solve all security concerns. This was demonstrated by

33 http://www.lslwiki.net/lslwiki/wakka.php?wakka=communications
34 http://www.lslwiki.net/lslwiki/wakka.php?wakka=GreyGooFence
35 http://www.lslwiki.net/lslwiki/wakka.php?wakka=energy
37 http://www.pygame.org/wiki/about
39 http://www.lslwiki.net/lslwiki/wakka.php?wakka=GreyGooFence
Securing Virtual Worlds Against Real Attacks: White Paper

Yee, Korba, Song, and Chen, who showed that most threats to online games (12 out of 17 that they identified) can be significantly reduced by properly designing a secure MMOG system.\textsuperscript{41} A simple example of what cannot be dealt with by technology is a physical threat to a player or a server administrator.

We have seen that currently the most troubling problem with online gaming is account theft achieved through the host operating system (password-stealing Trojans, backdoors, and malicious keyloggers that operate in the Win32 environment). Fortunately, this problem can be almost completely eliminated by applying certain improvements in the login process.

**Authenticating the client to the server**
The following measures are recommended for improving authentication of the client to the server. Some of them are in use or are being introduced in online banking, so the technology is already widely available and well tested:

- Allow users to tie their accounts to specific IP ranges or Media Access Control (MAC) addresses
- Send out notifications via email or text messaging about connections from unusual IP ranges
- Use a separate authentication channel (like Short Message Service, or SMS, messaging) in addition to standard user ID and password authentication
- Use one-time keys (a list of keys pre-shared between the client and server where each key is only valid for a single session and, possibly, for a specified period of time; these one-time keys can be distributed electronically via a web site, email, SMS, or, even better, snail mail)
- Use physical authentication devices, such as RSA tokens (see www.rsa.com for additional information) or similar methods (for example, biometrics with fingerprints, voice recognition, or face recognition)
- Use a public keys infrastructure (PKI) to verify users’ identity (users can confirm their identities by pointing the authentication server to their existing PKI records)
- Secure user input (via keyboard and mouse) during login (a curious obfuscation approach was demonstrated by Allen, Ford, and Saugere.\textsuperscript{42})

Some measures should be taken to minimize the risk of connecting to a spoofed server as well. We have seen attacks involving, for example, Domain Name System (DNS) changer\textsuperscript{43} malware where external accesses to financial institutions were redirected to replica phishing sites. It is very easy to redirect accesses to a game’s authentication server and capture all the login details. This can be done by manipulating the host’s file, by introducing a label-switched path filter, or even by manipulating a DNS server’s cache. If the workstation connects to the server by DNS and the IP resolution is manipulated, then the workstation will go to a replica site. The latter means that a redirection and compromise will occur even when the workstation is 100 percent secure! So, it is necessary for security reasons to use connection by IP, use DNS as a failsafe alternative, and verify the correctness of the returned IP. (Does it belong to the vendor’s IP range?) This means authenticating a server to a client is also important, as a compromised network infrastructure may connect a user to a fake phishing server.

It might be suggested that an Internet top-level domain (TLD) called “.game” should be established with a fairly high registration fee to reduce the risk of spoofed sites appearing in this TLD.\textsuperscript{44} This measure was suggested for “.bank” and “.safe” domains but the decision has not been taken. If, for instance, the “.safe” domain gets established, then it makes sense for major game vendors to use it.

**Building secure software**
Many books have been written on how to build secure software. It is not something that can be learned from a book and then implemented at once. It is a long and difficult process. We would like to draw your attention to the following specific areas related to online gaming:

- Scripting with limited or no persistency, auto-execution, and access rights (for example, via sandboxing)
- Encrypted and authenticated transmissions (so that packets in transit between clients and servers cannot be tampered with)
- Proper segregation of what kind of data is stored on servers and what kind of data is stored on clients
- Not transmitting any information to clients that is not necessary. This may be tricky, as it is generally very hard to predict exactly what local rendering will require, but certainly no debugging information and sensitive data

\textsuperscript{41} http://iit-it.nrc.gc.ca/iit-publications-iti/docs/NRC-48457.pdf
\textsuperscript{43} http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_imuttik_vb2005_manipulating_the_internet.pdf
\textsuperscript{44} http://www.circled.com/posts/cicann_adopt_bank_domain/ , http://www.gss.co.uk/news/article/3938
should be transmitted to normal clients. It is possible to send additional data in encrypted format and only transmit appropriate decryption keys when the rendering engine needs access to specific portions of this data.

- Protection from wall-hacking and map-hacking. This is related to not transmitting unnecessary data to the client.

- If there are different levels of access to the server data (for example, for a game master, or GM, in server administration mode or for a developer in a debug mode), then the authentication should always be done on the server side. For example, a GM password can be hashed in the client and transmitted to the server for verification against a known hash. The comparison of two hashes should not be done on the client side even if the result is then only used by the server.

- Persistent data transmitted to the client should be protected by digital signatures. Asymmetric encryption can be used. The vendor will sign the files, objects, or records with the secret key, and the client will be able to verify the authenticity using a matching key built into the client. The key revocation mechanism should be implemented for cases where the secret vendor key gets compromised or stolen.

- In-game communication methods—instant messaging, voice transmissions, bulletin boards, voice over IP (VoIP), discussion forums, and others—should not allow any active content (executable code, scripts) or links to such objects (clickable URLs that activate browsers capable of executing such objects)

- The use of all potentially dangerous objects should involve a delay if it is impossible or impractical to deny the users access to such functions. Possibly, the delay can grow if the frequency of requests is not going down.

- User input—and that includes network packets from the client and any files loaded into the client software, as well as keyboard or mouse inputs—should be treated as untrusted data and every field, boundary, and limit should be checked every step of the way. No assumptions should be made that the input would follow any standard format or pattern. In fact, it is not even safe to assume there will be input when it is expected (for example, a file can be unexpectedly truncated, or access to the data can be denied where software expects data to be accessible and available).

- Client software cannot trust the environment. There can be a debugger present or a hypervisor trapping any specific execution step. And server software cannot trust a client. There can be no guarantee that client software has not been replaced in full or in part with an emulator, which would log every piece of information transmitted by the server and take action to maximize the yield of this information. We are talking not just about general information like the geometry of the virtual world, but also about personal information about the players that is in the scope of such a logging emulator: their location, appearance, and habits.

- Backups and convenient per-transaction roll-back functionality in the server database

- Secure logging of events up to a level where all events can be replayed and viewed via, say, a standard three-dimensional game client, logging of virtual transactions, and database modifications.

- Take measures against distributed denial-of-service (DDoS) attacks. Internet-based DDoS attacks are well known, and some defenses do exist, but consider also in-game attacks (for example, deliberately too many characters, objects, or scripts in one area or sim).

- Code reviews, code inspections for core binaries and all scripts, and penetration testing to improve the quality of software minimizes the number of exploitable scenarios known as “cheats” and the number of bugs in general, especially remote exploits. Quality assurance with adequate alpha and beta testing is also recommended.

It is important to find the right balance between security and speed. More functionality on the server side makes the game more secure, but it can decrease the speed of reaction on the client side. More network packets have to be sent, so more lags will occur.

We have to say that enforcing security for MMOGs should be, overall, easier than on real networks because the whole environment is fully controlled by game vendors. Updating of both a game client and server code is in the hands of the vendor—so explosive threats (worms, viruses, widespread cheats), critical bugs, and zero-day exploits will be very short-lived and can be rapidly and fully exterminated. Thus, if any security issue emerges, corrective action can be taken as soon as the solution is developed, except perhaps for some free-shards (third-party servers, usually running on
Securing Virtual Worlds Against Real Attacks: White Paper

pirated or emulated software) and emulated environments, such as L2J (the open-source Java version of Lineage2).\(^\text{49}\)

Still, of course, it is best to minimize the number of security issues by building software securely from day one.

**Economic measures**

Probably the most important security tenet is that the cost of a security breach should exceed the potential benefit. For example, the cost of equipment required to cut a safe multiplied by the risk of getting caught while robbing a bank ideally should be higher than the amount of money in this safe. Breaking this rule creates problems that can be seen, for example, in the phenomenon of spam. Emails are free, so spammers can afford to send millions of emails and still benefit from the business, even though very few people actually buy advertised goods or services. If emails cost even a penny each, the return on investment (ROI) would likely become too low for spamming to survive. Unfortunately, it is too late to change the rules because neither the Internet nor Simple Mail Transfer Protocol (SMTP) was designed with security in mind.

Fortunately, the MMOG situation is very different. For example, charging for in-game messages could easily be made part of a game—and even be presented as part of the game, for example, as a “royal mail stamp duty” or something along those lines. At the same time, our understanding of security implications is also growing. Our recommendation would be to introduce floating charges for literally all in-game services, so that the laws of the virtual economy would make most anti-social and dangerous behaviors unprofitable. It might seem that while the problem of account theft is unsolved, these economical measures would have no effect on virtual spam because spammers could send messages from compromised accounts. We believe this would make no sense because if an account has very little money in it, spammers would be limited in the number of messages that they could send. If, on the other hand, the account has plenty of money in it, it would make a lot more sense to grab the funds rather than invest in a spamming exercise with an uncertain return.

There is still a reason for using virtual spamming even if there is a charge for it—money laundering. As long as the price is not creating a negative, the bad guys can use dirty virtual money because sales of goods and services orchestrated by a spam campaign will clear their funds. Unfortunately, a solution for raising the cost of virtual mail can only work as long as normal law-abiding players happily accept such a raise. When this is not the case, we will have to consider, for example, only charging for mails that contain URLs or mails that are sent to more than one recipient.

Apart from spam (which includes phishing and IM spam), another class of security issues that can be solved by introducing small charges would be DoS attacks that originate from in-game scripts. For example, in LSL one can initiate access to remote (out-of-game) URLs every three seconds. If enough objects have scripts accessing one URL, they can create a DoS attack when the scripts initiate. The throttling would not help if there are thousands of such scripts activated at the same time. For 1,000 scripts, there will still be 333 TCP/IP connections per second—or more if more such objects were created. If there were a small charge introduced (for example, Linden $1 per request) for external XML-RPC accesses, the problem would likely disappear. An even more drastic measure would be to charge some minimal amount for any public script or allow access only to registered sites and Internet protocols and charge Linden dollars for such registrations.

Keep in mind, though, that economic pressures cannot prevent attacks mounted to prove a point or out of curiosity. That brings us to the question of how else we can improve human behavior in the metaverse.

**Human factors**

It has been known for thousands of years that the inevitability of punishment is the greatest deterrent from committing any crime. MMOGs are in a unique situation because constant and comprehensive monitoring of the entire population is actually technically possible, and such measures would potentially allow for reviewing of any event that took place in virtual world. Most likely, the problem here would lie in the lack of investment by game vendors in the area of logging in-game events (movements, actions, and transactions) and analyzing logs. Security and logging is not exactly an area where one would expect any immediate benefit or a rapid return on investment. But without proper security, the user experience in the long run will suffer. Let’s consider an analogy. It is cheaper and quicker to build a car without seatbelts, anti-lock brake systems, and airbags. It is troublesome and expensive to operate speed cameras and police patrol cars on an ongoing basis. But we know that if these measures are not taken, there would be more casualties on the roads. People would eventually stop buying from some manufacturers because there are safer alternatives. If the situation on the roads is dire, at some point even the authorities might intervene and enact laws to ensure better public protection. Undoubtedly, in the long run, similar developments could occur in digital worlds.

\(^{49}\) http://www.l2jserver.com
Securing Virtual Worlds Against Real Attacks: White Paper

Logging

If comprehensive logging is introduced, then it might even help in situations where a player is granted additional privileges in exchange for a bribe or if they are under a physical threat. Unfair methods of playing (for example, outsourcing and so-called power-leveling\(^5\)) can be detected by IP analysis and tracking virtual currency inflow into accounts. Countermeasures for these would also be based on logging. They will not affect law-obedient users but will severely increase the risk of getting caught for everybody who violates the laws and/or rules.

The biggest loophole of total logging is when virtual commodities are converted into real money (RMT is commonplace even for games where such trading is against TOS) and then need to be tracked in reality. Again, the throttling idea may help where, for example, there is a delay in the exchange of the goods and receipt of money. The game vendor would play the role of a safe escrow institution for financial transactions. Waiting for a transaction to go through would be quite risky for the bad guys because, during the delay, alarms can be raised. Thus, such a delay may help enormously in preventing virtual fraud; it would assist with analyzing the logs, and it would control RMT. It seems natural to impose limitations only when a certain currency limit is exceeded over a short period of time, allowing a few minor transactions.

If real money is converted into virtual currency (for money laundering, as an example), logging and delaying such transactions that exceed a certain amount may help law enforcement track down criminals. Modifications to a server’s databases after exploiting a vulnerability or successfully deploying a piece of malware on the server can be tracked down and undone if there are sufficient logs and backups.

Unfortunately, it may not be possible to log certain malicious acts. For example, a buffer overflow that comes in a TCP/IP transmission is likely to either crash the server or execute malicious code, but it would not be in the server logs. Web servers do not normally log all HTTP requests because they generate too much data to keep track of. Logging selected TCP/IP transmissions (originating from new IP ranges) can be done, however. Protecting servers with special in-line network devices from exploitations and DoS attacks, and providing some level of logging that shields the server from outside attacks may be a good idea. Logging at least some incoming traffic may reveal the nature and sources of attacks. This may be extremely important if, for example, a DDoS attack is launched to achieve an unfair advantage in a game. One scenario could be to initiate a DDoS attack when virtual competitors (the opposing team) are in a vulnerable situation. This is likely to create unacceptable lags that might lead to a higher level of virtual casualties within the opposing team. The alternative is to use a zero-day vulnerability in the server software to cause a crash at a convenient moment or to run malicious code on the server. Such attacks can be financially motivated. If it were possible to log these violations and analyze the sources, then the frequency of such attacks would undoubtedly diminish.

Special effort should be invested into ironing out bugs in the routines that deal with the network packets. Vulnerabilities in these routines can be responsible for remote code execution. This is the worst kind of compromise because when this happens, data on the computer cannot be trusted any longer. Intruders can modify the server database and remove logs describing the intrusion. For this reason, it would be beneficial to run logging on separate computers or devices. If such an intrusion is not spotted in time, then even the backup may have poisoned data. Sticking to a schedule and maintaining historical backups becomes essential.

Network packets, of course, should also be treated as untrusted user data. The fact that a packet is supposed to be coming from the client does not mean that it would in real life. There can be a packet switching attack or on-the-fly modification. Even if the packets are authenticated both ways (packets from the client are authenticated using a temporary per-session key obtained from the server), their contents should be trusted because modifications can be made to the client software itself and data can be manipulated at a stage before the packet gets encrypted.

Telemetry

Even the most comprehensive logging is unlikely to be used if all it does is accumulate vast amounts of logged data without sensible interpretation.

Apart from comprehensive logging, we would urge game developers to invest time in proper telemetry systems. By “telemetry,” we mean a set of scripted rules—perhaps even heuristic ones—that examine the logs and raise alarms when any violations occur. Usually, this involves teams of people implementing telemetry (developers of the system and of the rules) and monitoring personnel (watchers on duty).

Implementing a telemetry system for a peer-to-peer (P2P) server grid is not trivial at all because some centralized

50 http://en.wikipedia.org/wiki/Powerleveling#Powerleveling
collection and analysis is likely to be required. Building distributed logging systems from day one is essential, as bolting it on afterwards may be problematic due to many additional dependencies (additional workload on servers and increased bandwidth requirements, for instance).

Special teams will have to be involved in reviewing the logs on an ongoing basis to respond to complaints from users and to review specific situations from users and/or telemetry. The usability of logging systems (an ability to replay events in three-dimensional space) is important to minimize the cost of this effort.

A possible alternative to server-side logging and telemetry systems is to record the session locally. If there is a security incident, then the recording can be uploaded for further investigation by the vendor. This is probably fine if the authenticity of such recordings could be established, but, unfortunately, this is not always the case because a recording could be a fake. Unless the server itself holds the logs corresponding to the security event in question, there’s no way to verify whether the recording is genuine. A possible viable compromise could be to store on the server a log of all key events (messages, financial transactions) and use the client recording only as complementary data that presents the user’s point of view. By matching the client-side recording to the genuine event log from the server, we can verify the authenticity of the recording and user version would only add details to the event log that the server already has. Apart from presenting the user’s point of view, this method would also reduce the storage requirements for logging on the server side.

All in all, we expect that comprehensive logging and telemetry systems can protect online gaming from many violations and would make the environment significantly more law-abiding and welcoming.

**Predictions**

In 1997, Dr. Alan Solomon\(^\text{52}\) predicted that within a few years, the Internet would look similar to the then-current AOL client (lots of pictures, graphics, IM, a user-friendly interface). At the time, the Internet was used mostly by scientists and programmers, so this prediction was hard to swallow. There was no Internet commerce then, no Internet banking, no company web sites. But Solomon was right—the things that were attractive and convenient to the computing public won out.

**Convergence of metaverses and the web**

With the increasing capabilities of the Internet and advances in connectivity, we can expect that pure web-based entertainment communities will converge with traditional specialized client-server MMOG software. The signs of this are already present. For instance, Second Life allows you to use the “secondlife://” prefix to create HTML-style links to any location within the Second Life world (the alternative is to use a “http://slurl.com/” prefix, which will tell you how to install the Second Life client in case you don’t have one). Double-clicking on such a link launches the game at a specified location. That is generally a safe operation, but if, for example, when you arrive at a destination a script runs automatically, then it may pose a security risk.

The behavior of Second Life users seems to be very similar to Internet browsing. They visit location after location, stay there for a while in the more interesting venues (read text, look at pictures, watch movies, listen to music, chat with other visitors), and then move on to another place. Some people even deny Second Life is a computer game because there are no explicit goals or scores. It is more of a social networking site designed for communication and that makes it remarkably similar to many online sites. For example, the interface of Second Life has clickable objects, including web sites and links (Figure 8).

**Actions**

The next question for a game developer to consider is: What action should be taken once a violation has been established? For mild violations (for example, placing a character in an area where he blocks a narrow path or a doorway), a forceful relocation of an offending character can be applied. Sony used this method to handle a social unrest in a Sony Online Entertainment game when many players gathered in one place for a demonstration.\(^\text{51}\) For massive violations, a rollback can be applied, which will restore the state of the virtual world at some previous historic point and revert all modifications that took place after this moment in time, including movement of characters and/or transactions that took place.

\(^{51}\) http://en.wikipedia.org/wiki/Duping

\(^{52}\) Alan Solomon was the father of the company Dr. Solomon's Software, which produced one of the first anti-virus products in the 1990s.
Some may say that Second Life is a three-dimensional equivalent of the Internet, where owning a piece of land (a “sim” or “island”) is equivalent to owning a domain. And building on your land (LSL scripting and all) is equivalent to web design. We know the Internet is a dangerous place with a great many malicious websites that use the drive-by download method to load malware onto users’ computers. In the future, we might expect similar attacks to materialize in Second Life. Malicious sims may start cropping up or existing ones may be hacked. Tools that index a virtual world (à la Google) may get manipulated by the bad guys, and exploits and sophisticated social engineering (the combined power of scripting and human involvement) may eventually be unleashed on visitors.

We know that malicious links are now widely used to disseminate real malware and exploits. So when many online games start supporting clickable URLs and compatible scripting, then we would have a dangerous cocktail on our hands because clicking a link in a game could open a browser and cause trouble. We must make sure that appropriate measures are taken to prevent spammy advertising links to scripted virtual resources. And no doubt, we’ll be seeing an increase in this kind of spam in the near future. It’s likely that we will also see similar types of spam occurring via in-game communication mechanisms—“virtual spam,” if you will. It is quite possible that future web browsers will support links into many different games, and this, of course, will have security implications.

Now let’s look at online gaming from the point of view of web browsing. You can’t help noticing that there is an ongoing trend to create a richer browsing experience on the Internet: You’ll find more movies, sounds, and video clips. Standard browsers respond by handling more objects, and plugins, such as Flash, are already capable of creating MMOG environments. Have a look, for example, at offers from Neopets.com and ClubPenguin.com. (The latter is now owned by Disney.) These popular sites create virtual worlds using contemporary browsers as clients.

Here is how a browser window looks on the ClubPenguin site. If you compare Figure 9 with Figure 8, above, you’ll see a significant difference in that the images are two dimensional instead of three dimensional. Otherwise, the environments are very similar.

If you click any of the three yellow tablets located in this room, you would activate a Find Four game. So we can play “in-game” games. We should watch for the level of virtual nesting! Fortunately, security issues on all virtual levels should not really be different.

Clearly, new web technologies are paving the way to richer content. Recently, Microsoft released SilverLight, which is an alternative to Adobe’s Flash player. SilverLight is based on .NET, so it is cross-platform. Three-dimensional support in browser plugins was announced in August 2007 for Adobe Flash by using the third-party Swift 3D v5.0, produced by Colorado-based Electric Rain. Very soon three-dimensional functionality will be available for all browsers, and advances in broadband connectivity will allow three-dimensional content to be routinely used. Some attempts in this direction already exist with three-dimensional browsers, although they are currently more of a standard two-dimensional browser presented via a three-dimensional environment (for example, on the sides of a cube that can be manipulated in three-dimensional space), but this is still just a glimmer of a coming trend.

Ajax technology can use more atomic client-server interactions, which are very useful for gaming. Ajax is based on XMLHTTPRequest—supported by both Internet Explorer and Firefox. At the same time LSL supports XML-RPC, so there is some support of Web 2.0 in LSL.

As we can see, web and MMOG technologies are converging. Given that the browsers have been and

54 http://en.wikipedia.org/wiki/Drive-by_download
55 http://news.thomasnet.com/fullstory/529168/rss/2585
56 http://news.thomasnet.com/fullstory/529168/rss/2585
57 http://www.browse3d.com/
58 http://www.lslwiki.net/lslwiki/wakka.php?wakka=xmlrpc
continue to be plagued by severe security issues, there is a significant risk that such convergence will make the attack surface on MMOGs much wider than it is currently. It would seem natural to assume that in the future, more MMOGs will be based on browsers (no need to develop a client, so it is much cheaper), thus making malicious attacks on the online games easier. Attacks can use vulnerabilities in browsers and steal, for instance, virtual identities and commodities in addition to real ones.

Integration of metaverses
Online gaming is booming. New players, large and small, are surfacing in the marketplace all the time. It seems as if everybody wants a piece of the pie. It would not be surprising to see a rather low level of coordination and standardization during this phase. But user demand for communication that works across different games will inevitably lead vendors to provide interfaces to existing communication media (browsers, email, IM, VoIP, SMS) and among each other. People tend to play for long periods of time and would prefer to have all communication methods integrated and easily accessible. Because of that, we expect that access to existing communication vehicles will be organized first via in-game mechanisms by using appropriate application program interfaces (APIs) to respective protocol providers. The first report of this has appeared recently.59

There will also be strong user demand for increasing interoperability of virtual games, and we should expect the appearance of virtual gateways, similar to border checkpoints, quite possibly even with customs and taxes. Such gateways will likely be able to transmit messages initially. Later, it may become possible to send parcels with virtual commodities between virtual worlds and transmit virtual currencies via these bridges.

It is important to make sure that no active content (executable code or scripts) can be transmitted. If necessary, active content should be inspected (automatically and/or manually), charged for, and logged. Transmitting messages with links (even with standard HTTP links) without any vetting or control would assist the bad guys because traditional real-life spam would then become virtual.

The peer-to-peer shift
When the number of users for a single MMOG server exceeds 10,000 to 20,000, new subscriptions are usually redirected to a new server (frequently called a “shard”). This has to be done to allow the servers to cope with the load. New players are isolated from the old crowd and populate a replica of the existing world. The drawback, of course, is that in many cases new and old users do not share the same metaverse and cannot enjoy simultaneous playing. Economies are also separate, which is reflected even in slightly different exchange rates, as we have seen in the eBay screenshot in Figure 2. This is how things were. Modern technologies, though, allow for server clustering and can be infinitely expandable.

A server cluster for EVE Online60 supports around 40,000 simultaneous users. A similar number is quoted as the average count of Second Life users at any given time. Second Life uses a grid of servers. Each supports one sim (island) with all objects and avatars on it. When objects or avatars move to another sim, their database records are passed to another server. We expect similar P2P approaches to become more common as the market drives developers to support more players and a common metaverse. Perhaps at some point in the future, P2P technology will also use P2P-connected clients to process data. If we use digital rights management, it might even become possible to store parts of the main database on the client, which may initiate a trend toward using clients’ computing capacity to support wider and more powerful P2P environment. This, of course, might introduce security risks, such as data tampering.

Within massive P2P server grids, the risk and implications of an auto-executing worm could also become very significant. For this reason, even if a server cluster communication is done on a separate network or via a virtual private network (VPN), it would make sense to use encryption and authentication for inter-server communication. That would also provide some protection from internal compromises.

Windows Vista has built-in support for P2P networking, so this should either be disabled to prevent propagation of P2P worms or P2P communications should be strictly controlled.

While P2P clusters are more resistant to DDoS attacks, measures should be taken to allow the filtering of unwanted traffic—in-line network appliance hardware would probably be a good option.

Traditional anti-virus, threats to online gaming, and cooperation
As we know, malware has increased exponentially in recent years. This may seem paradoxical, but this growth is likely related to the effectiveness of anti-virus software in blocking known threats. As a result of scanners detecting existing threats, new malware is constantly being written to replace previous versions that were rendered ineffective by anti-virus software.

60 http://en.wikipedia.org/wiki/Eve_online
Currently, the involvement of security companies in protecting the gaming environment is mostly limited to detecting password stealing malware. This malware is written for the Win32 environment and does not require malware researchers to possess any special knowledge about the game’s internal operation. In most cases, just knowing the name of the game’s program files, folders, and running processes is enough to determine what is targeted.

There are also optimizations in on-access scanners designed to optimize the performance of gaming software. For example, while a game is running, the on-access scanner (commonly called OAS) might temporarily add the game to the list of exclusions.

Security research related to the in-game threats, such as Lua scripts, requires detailed knowledge of the inner workings of the targeted environment. To handle threats developed for different games, security researchers need:

- Better than average knowledge of the environment
- Access to the environment, preferably with debug capabilities; an isolated server-client pair may be needed, for example, to examine how a remote exploitation works
- Clearance from the employer to run tests for malware in various gaming environments
- Enough demand from customers to justify research and development for such security solutions.

For game vendors, players comprise 100 percent of their customer base, for security firms, only a small fraction of customers are gamers, so game-related security problems are frequently not significant enough to receive priority attention.

It’s difficult to meet all these requirements, so existing knowledge is simply inadequate in coverage and in depth. For example, the first viral Lua scripts were discovered by GMod players and only then brought to the attention of security companies simply because this was a new propagation vector that was not monitored by anti-virus software.

With the number of incompatible virtual worlds expanding as the market grows, it is natural to assume that initially we will be facing a multitude of problems specific to this or that virtual environment. At this stage, it’s unlikely that security companies will be able to keep up and cover all the security issues for the vast multitude of games.

When traditional anti-virus programs provide inadequate protection—and frequently because they do not have adequate visibility, as with Lua viruses—measures will have to be taken by the game vendors. It is very likely that in the short term, security problems in MMOGs, apart from data-stealing Trojans, will be predominantly covered by security and policy departments of respective game developers. Cooperation among security departments of security companies and game vendors is certainly welcome and, in my opinion, would be mutually beneficial.

Game vendors’ protection systems are quite commonplace because tampering with clients would otherwise be more widespread. Of major concern is using bots or other cheating additions to game clients. These cheating systems exist as software (bots) and hardware devices (USB simulators of a keyboard and mouse that automatically play instead of a real person). For example, Warden for WoW is a scanning tool that detects unwanted programs (bots, cheats, and the like). Detecting hardware simulation devices is trickier but can certainly be done on a case-by-case basis. This is an area where vendors’ scanning tools would behave very much like traditional anti-virus scanners. The emphasis, though, is a little different, as anti-virus is focused mainly on malware, while the cheating software and hardware are not malicious. So they cannot even be classified by anti-virus producers as potentially unwanted programs (PUPs) because from the users’ point of view, these cheats are desirable! From the point of view of the game vendors, though, they violate the rules (TOS), ruin the balance of the game, and create unfair advantages. It is quite clear from the above that there won’t be user demand for anti-virus software to detect and block cheats. At the same time, game vendors need proven security technology.

Before too long, we will start seeing polymorphic or metamorphic bots (bots in the MMOG sense). Bots that make use of stealth and rootkit technologies already exist and even became a point of legal battle.61 We soon might see cheating software that uses virtualization, too. These technologies are already widely used in malware, so we should expect them to find their way into unwanted programs that plague online gaming on a significant scale. Detecting these threats is not always a simple task and proper cleaning, to be honest, is frequently a real pain. Another area where security companies have a great amount of knowledge to share is anti-spam solutions. Many games already have serious problems with abuse of internal messaging systems. The knowledge of security companies about how to deal with these security issues should help. For example, cooperation can take form of licensing anti-virus/anti-spam engines for game security or providing

---

61 http://www.realpoor.com/Blizzard_to_Sue_Creator_of_MMO_Glider_Bot_t147334.html
APIs to anti-virus/anti-spam engines to interrogate in-game scripts and other objects.

**Legal conflict within online gaming**

By now, it should be clear that the interests of game developers and players are often conflicting. A gaming bot—which goes against the TOS and is also undesirable from the point of view of the gaming community—is usually very desirable for the owner of the computer. Detecting such a bot with anti-virus software as a PUP does not seem appropriate because the game vendor, who considers it unwanted, may never get to see the detection. Normally, a user can suppress all PUP detections and ignore them going forward, so there would not be any point in having anti-virus software report game-related PUPs anyway—a user would always choose to ignore and/or suppress such detections.

When a user agrees to the TOS or end-user license agreement (EULA), the game vendor is interested in monitoring the violations (running a bot or other cheating software), so tools like Warden and GameGuard may unconditionally report bots through a live Internet connection, and this may even result in termination of a player’s account. There are two sides in this equation: the owner of the computer hardware who also bought a game and the game vendor. And they will be fighting for who has the authority to use specific software or hardware on the user’s machine. What a tricky legal battle this could turn into!

**Follow the money and get more cooperation**

We know that profit has been the main factor driving up production of real malware and fuelling many new attacks since at least 2003. That means, of course, that commercialization of the virtual environment should lead to a similar wave of virtual malicious activity. And when virtual worlds start merging into a virtual universe, it will be the time for global security companies to step in—because everybody will expect them to! We are bound to see virtual spam, abundant and annoying advertisements, phishing scams, virtual identity theft, and so on unless, by design, MMOG developers make the environment technologically, economically, and socially resistant to these attacks.

Currently, to track down cybercriminals, we use the “follow the money” principle. Cooperation of different bodies in different countries is frequently needed to do the police work. Occasionally, this turns into an impossible task because of differences in legislation, language barriers, and even time zones. It would be a very good thing if major game developers could reach agreements about preserving the logs and cross-checking inter-game transactions when they become a reality. It would seem beneficial to encourage such inter-game exchanges and discourage conversions into real money because that would make transactions significantly more difficult to trace. We can even envision the appearance of a virtual Interpol (or, perhaps, the real Interpol would invest in a virtual criminology department?). Sooner or later, cooperation among real law enforcement and virtual police will have to take place—most likely on an international scale.

It seems essential that cooperation among major online gaming companies will be required to achieve the goal of merging different games into a single metaverse. From the security standpoint, cooperation is, perhaps, even more critical for thwarting similar kinds of attacks elsewhere once they have been identified for the first time.

**Conclusion**

During the explosive growth of online gaming when vendors struggled to reduce their time to market, we fully expected that security issues would get a bit sidelined, at least for a while. But we urge game developers to build the basic security foundation from the very beginning. As we know, bolting security onto an existing product is a far-from-perfect approach.

Most of the attacks that we have witnessed in real life will surface in virtual worlds unless the environment is built with security in mind. We need to leverage our knowledge and work together—security vendors and gaming vendors—to avoid falling into the same trap again. It is possible to make most attacks in virtual life impossible or uneconomical. There are no good reasons why virtual characters should suffer from the same troubles—spam, phishing, adware, spyware, Trojans, viruses, worms, and other malware—that currently plague our real day-to-day lives.
If we succeed, then the population of merging metaverses will grow. Users will flock to virtual worlds because they will feel more secure and protected. If we fail, well, it is a virtual world, so we can erase and start from scratch, can't we? I am afraid not. We really have to get it right the first time.

Acknowledgements
I am very grateful to my colleague François Paget for statistics on data-stealing malware.

Dr. Igor Muttik is Senior Architect for McAfee Avert Labs. Dr. Muttik holds a Ph.D. in physics and mathematics. His studies of the first computer viruses led to his career at Dr. Solomon’s Software, which was later acquired by McAfee, Inc. In addition to his research work on malware, Dr. Muttik is a frequent presenter at security conferences around the world.