Multiplayer Online Games Insecurity

(never feel safe while playing online)

Luigi Auriemma[1] and Donato Ferrante[2]
ReVuln
http://revuln.com
info@revuln.com
http://twitter.com/revuln
25 January 2013

Abstract *Multiplayer online games security is an underestimated field, with an insane amount of players playing online games and companies pushing out new games at an incredible rate. In this ecosystem finding vulnerabilities in games turns to be a really attractive work.*

1 Why attacking games?

There are two main entities in multiplayer games: *players* and *companies*. Players play games for fun, companies make games for money. For each of these two entities there are different possible subsets of attackers. Let's start by considering players.

1.1 Attacking players

Some of the people who may be interested in attacking players systems while they play online games include:

- *Script kiddies* - people without any technical knowledge. They browse the internet looking for ready-to-use attacking tools, and then they use such tools to attack players. Primarily, their final goal is to generate Denial Of Service (*DoS*) conditions against players systems.
- *Others* - this category is composed by several entities. We may find people testing exploits, or people trying to build a games-based malware botnet. Others also include people interested in deploying "applications" on remote systems.

1.2 Attacking companies

As stated previously, companies make games for money. This is their business. Because of this possible attackers for game companies include:

- *Script kiddies* - this category of attackers makes no distinction between players or companies. They attack both for fun. Again, even in this case they usually tend to perform *DoS* attacks.

[1] http://twitter.com/luigi_auriemma
[2] http://twitter.com/dntbug

- *Competitors* - this is probably the most interesting category for companies, because companies' games are a very large maket. Companies (as attackers) may aim to create issues in their competitors' games, so that players will migrate to their systems. This migration has a big impact on companies because this is a monetary loss for the victim, and a huge revenue for the attacker (company). Please note that usually for people playing online games it's a big deal not to be able to play the game because the online servers are down, or because their computers always become infected by some malware. In this category, we may see *DoS*, *Distributed Denial Of Service* (*DDoS*) and *Remote Code Exectuion* (*RCE*) attacks.
- *Others* - people who may have interest in accessing the remote server infrastructure (users database, transactions database, etc.). Also, people interested in a specific target (player) playing games on a specific game server. In this situation, the most common type of attack performed is a *RCE* attack.

2 Possible scenarios

We have two main possible attack scenarios for online games: *client-side* and *server-side*. For *client-side* we mean attacking the game client itself, which is usually on the player system. For *server-side* we mean targeting the game server, which is usually hosted by companies or rental third party server providers. Please note that in several games, the server component can be hosted by the players directly.

2.1 Client-side scenario

Figure 1: Client-side attack scenario

The most common approach to exploit this kind of attack is the following:

a. The attacker is aware of a client-side vulnerability

b. The attacker obtains the victim IP address, or uses the server as forwarder, or hosts the malicious game server

c. The attacker exploits the client-side issue

d. The attacker takes control over the victim's system

2.2 Server-side scenario

More complicated than the previous one. This scenario can have multiple effects. Please consider the following server-side scenario for a company-hosted server:

a. The attacker is aware of a server-side vulnerability

b. The attacker exploits the server-side vulnerability

c. At this point the attacker has two different options:

   - Exploit a different client-side vulnerability against the clients
   - Try to inspect and obtain sensitive information from the server infrastructure itself.

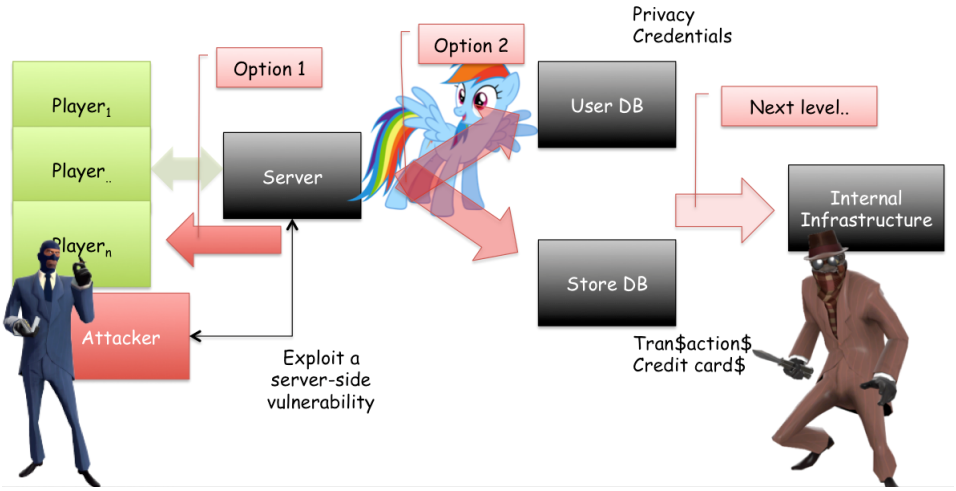

Figure 2: Server-side attack scenario

> Please keep in mind that most of the current games stores player information (name, e-mail, home address, phone number, etc.). Credit cards allow players to use features such as in-game markets, where players can buy additional in-game contents by using real money.

3 The missing ring

At this point of the paper we are aware of the possible victims: *players* and *companies*. We know about the possible attackers: *script-kiddies*, *competitors* and *others* entities. But there is still something missing. In order to have victim and attacker to interact, we need to have security vulnerabilities. There are two main ways to get game vulnerabilities: *finding or buying* them. That is one of the reasons why game vulnerabilities market exists.

3.1 The market

Differently for other markets, people on the games vulnerabilities market usually seek 0-day-only issues. This is due to the fact that well-known games updates are pushed to clients and servers as soon as a fix is available. Players are usually forced to update their client in order to play the game itself. In other words, the 1-day

market for games is of limited impact for well-known games. Please note that in this market even *DoS* issues are very valuable, as we explained before, taking down competitor's servers can be very valuables for companies. In this market the main buyers include:

- Players
- Server admins
- Game companies
- Others

4 HUNTING FOR GAME VULNERABILITIES

Games are usually built upon a so called game engine. A game engine is the core of games, it provides most of the game features, such as: game logic, physics, graphics, sounds, network support, and so on.

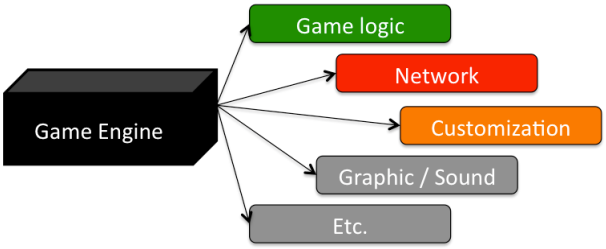

Figure 3: Game engine features

This is a very important concept for games and for bug hunters targeting games. The reason is that most of the games available on the market share the game engines. In other words, if you find a game issue affecting a game engine you are very likely to be able to exploit the same issue in (usually) the same way against several different games. To get an idea of the current status of *games and game engines* please refer to the following list:

- *Source engine* - "Source is a 3D game engine developed by Valve Corporation. It debuted in June 2004 with Counter-Strike: Source and shortly thereafter Half-Life 2, and has been in active development ever since. Source was created to power first-person shooters, but has also been used professionally to create role-playing, side-scroller, puzzle, MMORPG, top-down shooter and real-time strategy games."[3].
- *Unreal engine* - "The Unreal Engine is a game engine developed by Epic Games, first illustrated in the 1998 first-person shooter game Unreal. Although primarily developed for first-person shooters, it has been successfully used in a variety of other genres, including stealth, MMORPGs and RPGs. With its core written in C++, the Unreal Engine features a high degree of

[3] `http://en.wikipedia.org/wiki/Source_Engine`

portability and is a tool used by many game developers today.The latest release is the UE3, designed for Microsoft's DirectX 9 (for Windows and Xbox 360), DirectX 10 (for Windows Vista) and DirectX 11 (for Windows 7 and later), OpenGL for Mac OS X, GNU/Linux, PlayStation 3, Wii U, iOS, Android, and Stage 3D for Adobe Flash Player 11."[4]

- *Unity engine* - "Unity is a cross-platform game engine and IDE developed by Unity Technologies, targeting web plugins, desktop platforms, video game consoles and mobile devices."[5]
- *idTech engine* - "id Tech is the family of game engines designed and developed by id Software. Prior to the presentation of the id Tech 5-based game Rage, the engines lacked official designation and as such were simply referred to as the Doom and Quake engines, from the name of the main game series the engines have been developed for. id Tech numbers 1, 2, 3, and 4 have been released as free software under the GNU General Public License."[6]
- *CryEngine* - "CryEngine is a game engine designed by Crytek."[7]

For a complete list of game engines, please refer to the list[8] of games engines available on Wikipedia.

4.1 Tools of the trade

In order to start hunting for games vulnerabilities we usually need three things:

- A game, in other words our target
- A debugger/disassembler, like OllyDbg[9] and IDA Pro[10]
- Some network monitoring tools. Please note that we need to be able to simultaneously monitor and inject custom packets on-the-fly into the current network stream. In our opinion the best approach to do this task is to use a custom and scriptable DLL-proxy.

4.2 Targeting games

Games are very complex pieces of software, using custom protocols, various encryption and compression algorithms, which in several cases are custom implementations of known algorithms, complex data structures and network protocols. Moreover games tend to use anti-cheating protections to prevent cheating and also impact the vulnerability research process. In other words, games are very interesting and challenging targets for bug hunters.

4.2.1 Custom protocols

Because online games need to be fast and lag-free, it's very common to adopt TCP-over-UDP based solutions while developing the network subsystem for games.

One of the results of this approach is running into custom protocols. There are usually four things that define a custom protocol:

[4] http://en.wikipedia.org/wiki/Unreal_Engine
[5] http://en.wikipedia.org/wiki/Unity_(game_engine)
[6] http://en.wikipedia.org/wiki/Id_Tech
[7] http://en.wikipedia.org/wiki/CryENGINE
[8] http://en.wikipedia.org/wiki/List_of_game_engines
[9] http://www.ollydbg.de
[10] http://www.hex-rays.com/products/ida/index.shtml

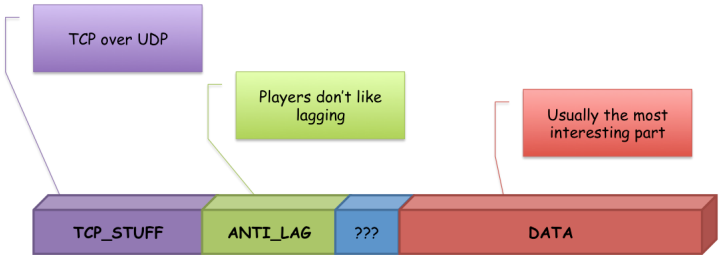

Figure 4: Typical custom protocol UDP packet format

- *TCP header*, sequence numbers and packets order information
- *Anti lag*, mostly time-related information
- *Misc*, protocol-dependent data, it can be interesting to dissect. Several games use this part to deal with game dependent data like channels, data reliability and so on.
- *Data*, the actual data contained in the packet

A very interesting child of custom protocols is the *fragmented packet*.

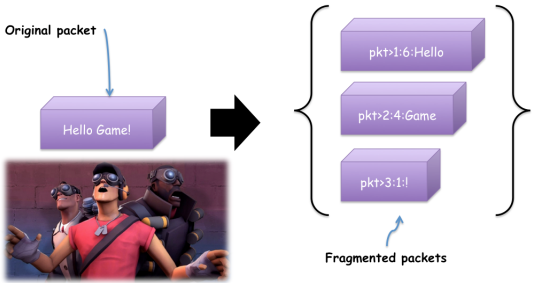

Figure 5: Fragmented packet

A fragmented packet is a UDP packet, which is used as base unit for TCP-over-UDP implementations, and it is usually composed of:

- *POS*, the position of the current packet in the given stream
- *LEN*, the length of the current packet DATA field
- *DATA*, the actual packet data
- *MISC*, other implementation dependent bits

If you are hunting for game vulnerabilities, fragmented packets are usually a good thing to check for interesting issues to exploit.

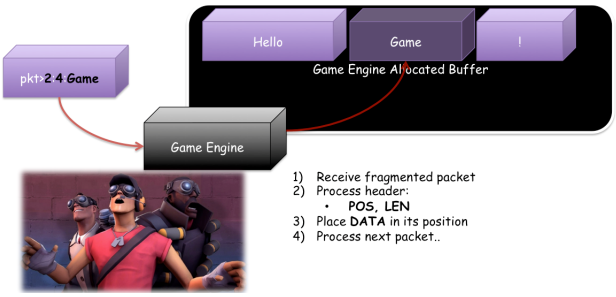

Figure 6: Fragmented packet

Rebuilding the data from fragmented packets is similar to a puzzle where the other endpoint specifies where each piece must be placed to obtain the original packet. The problem here usually arises because of missing checks while processing the packet headers, especially when the game engine deals with the *POS* and *LEN* fields of the packet.

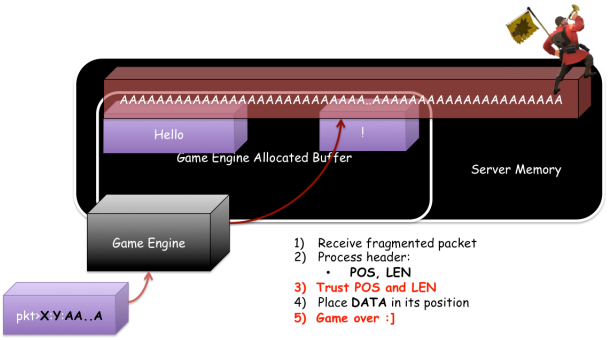

Figure 7: Fragmented packet handling

In fact, it may be possible to force the engine to load an attacker-controlled packet to arbitrary memory locations, by providing malformed fields or even trigger a classical heap overflow vulnerability by playing with the sequence of fragmented packets.

Nowadays fragmented packets issues are very commons in very popular games. A very good example of such issues is given by the Valve's Source Engine and a memory corruption bug exploitable via fragmented packets. To get an idea of the

impact of this bug, please note that we are talking about a *RCE* bug affecting a game engine having more than 10.000 online servers and used in well known games such as: Half-life 2, Team Fortress 2, Left 4 Dead, and others. The bug was caused by the usage of a small heap buffer (assigned to hold the entire packet), and by wrong checks on *POS* and *LEN* fields. Because the Source engine is mostly written in C++ it is trivial to find a function pointer to overwrite in order to gain code execution.

4.2.2 Cryptography and compression

As we are interested in network-based games, it's always useful to understand how the data inside the network packet is stored. This is in order to ease the amount of reversing work required to get an understanding of the network protocol and its packets. A quick strategy to get an idea of the type of compression or cryptographic algorithms in use by a given piece of software is checking for known constants or pattern into the binary or in memory. We do usually use tools such as signSrch[11] to do this task.

From a survey we conducted over a large set of multiplayer games, the following is a recap of the most used compression and crypto algorithms used by games. For cryptography we have:

- *RC4*, game-related software usually use customized versions
- *AES*
- *Blowfish*
- *TEA*, games usually use customized versions
- *XOR*, not exactly a crypto algorithm, but very common

For compression instead:

- *Zlib*
- *Lzss*
- *Lzma*
- *Lzo*
- *Huffman*
- *Others*, several proprietary custom algorithms

When dealing with reversing and tracing incoming packets, we may notice that quite often for games, the packets don't contain byte-aligned data, and this may cause some trouble for people not used to dealing with these kind of structures. In fact, several games use features such as *Bitstreams* and *Index numbers* to maximize the amount of information contained in a given packet, and also boost the network performances of the game itself.

4.2.3 Bitstreams

Bitstreams are used in several well-known games and even in streaming-related software, using a transport protocol such as: *MMS* or *RTP*.

[11] `http://aluigi.org/mytoolz.htm#signsrch`

4.2.4 INDEX NUMBERS

Index numbers are widely used in network-based games and the base idea is pretty interesting. If we want to transmit small (less then 4 bytes) integer numbers then using this approach we can save bytes in our packets. Please note that there are two different versions: *unsigned* and *signed*. Let's consider the *unsigned* case first.

The idea at the base of this approach is to use the byte unit as 7-bit for the value and 1-bit for the "has next" flag. If the number can be stored in less than 8-bit we just need 1 byte to be transmitted. In case we need to send more than 1 byte then we set the 8th bit of the byte to "1" so that the engine will know that it needs to concatenate the current byte with the next one in order to get the actual number value.

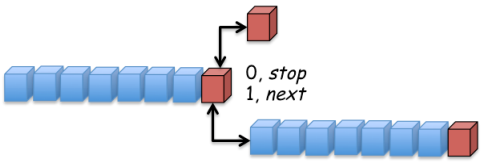

Figure 8: Index Numbers (unsigned)

On the average this approach is very useful because it can save a lot of data. The worst case scenario is when we need to send big numbers that will require 4 bytes for the value, so in that case we need to send 5 bytes instead of 4, but this is just a corner case for this solution.

The *signed* version is very similar to the one we just discussed above, the main difference in this case is that we use 1-bit of the first byte for the sign, then 6-bit for the value and 1-bit as "has next" check. The remaining bytes will instead use a bit pattern like the one used in the *unsigned* case. Please refer to the next picture to get a quick overview of the *signed* approach:

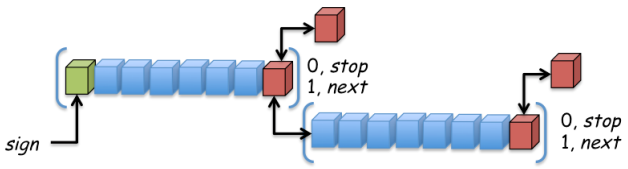

Figure 9: Index Numbers (signed)

As you may have argued if this solution is in place the developers already know the most frequent range of values that will be transmitted by the game over the network, so they know that the average size of the integer used in the network communication is usually less then 4 bytes.

For games using *index numbers*, a good idea to spot a vulnerability is to play with the "control" bits of packet sequences. Think about flipping the *has next* bit or the *sign* bit for signed index numbers.

4.2.5 Anti-cheating and anti-debugging

Game companies are pretty scared about people cheating on their games, more than people finding vulnerabilities. But since they adopt several techniques to prevent people from cheating, these techniques also usually impact people hunting for bugs in games.

The main features for anti-cheating solutions usually include the following (most of them used by Punkbuster[12]):

- Real-time scanning of memory for hacks/tools (including debuggers..)
- Random checks against player looking for known exploits of the game engine
- Calculate partial MD5 hashes of files inside the game installation directory
- Request actual screenshot samples from specific players
- Search functions to search players for anything that may be known as an exploit

But game protections are not only there to prevent people from cheating. Several times they provide an extension to the current attack-surface for the game they are trying to protect. There are several examples of thus fact. One interesting observation is that it was possible to trigger a format-string issue against games like *Quake 4* or *Doom 3* that were running protected by Punkbuster. Basically the issue was not a real problem if the game was not protected by Punkbuster, in fact the engine (alone) was able to escape all the malicious "%" chars in order to avoid the format string issue, when the protection was active on the game, the vulnerability was completely exploitable.

4.2.6 Game engine

The game engine is the core of the game itself. It's usually the most interesting location of the whole game from a security point-of-view. If we are looking for bugs, the best place to look at is how the game specific opcodes are processed by the engine. In fact, that is the place where most of the issues can be triggered. In order to reach the opcodes processing routine, a possible strategy is the following:

- Monitor network inputs
  - recv
  - recvfrom
  - WSARecv
  - etc.
- Locate and trace the recv'd packet in memory
  - defeat and bypass cryptographic algorithms
  - defeat and bypass compression algorithms
  - etc.
- Locate the opcodes processing routine

[12] `http://www.punkbuster.com`

– usually a switch (statement), where each (switch) case is usually a protocol opcode

The only issue in reaching such locations inside a given game is due to the level of protection the game has. And the ability to avoid, and understand crypto and compression algorithms that may be used on the incoming packets. We need to be able to craft and send packets that will target a specific opcode handler, each time, in order to verify if a possible issue can be triggered and later exploited in some manner.

4.3 Game specific vulnerabilities

There are several game specific issues that one needs to be aware of while dissecting game engines to spot security issues. Some of these issues are:

- Map loading attack - an attacker provides a malformed map to the victim in order to exploit a map parsing issues. This problem is fairly common due to the complexity of the parsing functions used to load maps content into the game itself. In this category we usually can find many *integer overflow* issues.
- Fake players attack - a *DoS* attack that is conducted by emulating a client-side protocol in order to fill online servers with fake players. This problem is usually due to the way the login procedure is performed.
- DOS forward via server - an attacker can trigger a given issue on all the clients connected to the server at a given time, by exploiting the ability of several server-side opcodes to perform opcodes broadcasting.

4.4 Finding victims

There is an interesting feature, called Master Server, that comes with multiplayer games. A Master Server is in charge of dealing with game servers and clients and keeping a list of all the games online at a given time. A Master Server usually does the following tasks:

- *Heartbeat handling* (from Servers) - handle join requests coming from new game servers
- *Queries handling* (from Clients) - handle queries coming from game clients. These queries use several filters, such as to include/exclude empty or full servers from the list of the servers available.

Master Servers may be abused by attackers to find vulnerable servers or clients, locate specific players, and gather a lot of additional information that can be used to perform targeted attacks.

5 Case of study: Steam

We covered the Steam protocol insecurity in detail in one of our previous papers[13] and we released a video[14], as proof-of-concept of the issues. In this section we are going to discuss additional information regarding the vulnerability we found.

[13] `http://revuln.com/files/ReVuln_Steam_Browser_Protocol_Insecurity.pdf`
[14] `http://vimeo.com/51438866`

5.1 What is Steam?

From Wikipedia: "Steam is a digital distribution, digital rights management, multiplayer and communications platform developed by Valve Corporation".
With Steam users can buy games, download demo and free-to-play games, find multiplayer matches, communicate with other users, share stats and so on. Steam is the digital delivery platform having the biggest user base (approximately 50 millions users) and supporting several platforms: Windows, MacOS, PS3, mobile devices and Linux.

5.2 The Steam Browser Protocol issue

We found a way to remotely exploit local bugs by using Steam as attack vector. Our idea was simple, as most of the games allow users to provide custom command line arguments to them on game launch. We found a way to start remote games with arbitrary "bad" parameters by using Steam as vector.

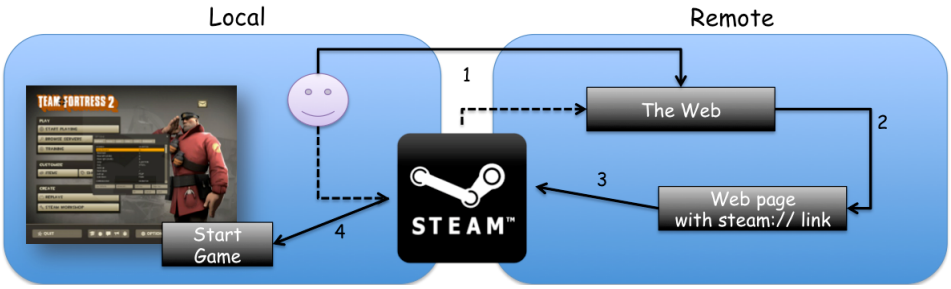

Figure 10: Steam Browser Protocol

To remotely trigger a local vulnerability via command line parameters in any vulnerable game, we just need to use one of the following four commands:

- steam://run/id/language/url_encoded_parameters
- steam://rungameid/id/language_bug/url_encoded_parameters
- steam://runsafe/id
- steam://rungame/id/lobby_id/parameters

5.3 Is the issue fixed?

When releasing our initial advisory we provided in detail several suggestions on how to fix the issues:

- Fix for users - disable steam:// URL handler
- Fix for Steam - avoid games command-line and undocumented commands accessible from untrusted sources
- Fix for games developers - secure programming and certificate validation for game update

After our public disclosure of the issues we are aware of just two games-related fixes: one for *Team Fortress 2*[15] and one for *APB Reloaded*[16]. But nothing for Steam itself. Steam can still be used as a huge attack vector. An attacker just needs to pick another game and find a new vulnerability in the game or Steam itself, like we did in one of the issues we found, before remotely triggering the issue via Steam.

5.4 Finding more games to exploit via Steam

One of the possible ways to find games exploitable via the Steam browser protocol, is to download one of the 2000 games available on Steam itself, and start hunting for local bugs. A good strategy is usually to look for command line options available for the current target. For instance, we may want to try to locate interesting ways to deal with: *maps*, *patches*, *configurations*, etc.

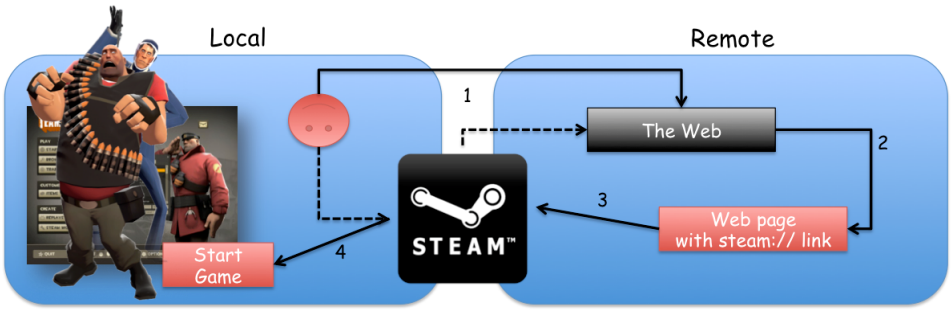

Figure 11: Steam Browser Protocol Exploit

Once we have a local bug we can remotely trigger it by crafting a Steam-link, using one of the four commands detailed above, and publishing the link on some web page ready to be served to the target.

6 What's next?

The biggest category of games that is driving the market at the moment is composed by *Massive Multiplayer Online Games* (MMOG). "A massively multiplayer online game (also called MMO and MMOG) is a multiplayer video game which is capable of supporting hundreds or thousands of players simultaneously. By necessity, they are played on the Internet. Many games have at least one persistent world, however others just have large numbers of players competing at once in one form or another without any lasting effect to the world at all. These games can be played on any platform it, be it the personal computer, a game console such as the internet capable PSP, PlayStation 3, Xbox 360, Nintendo DS, PS Vita or Wii, or mobile devices and smartphones based on such operating systems as Android, iOS and Windows Phone."[17]

The most interesting and attractive features of MMOG include:

- Huge player-base
- Complex network protocol

[15] `http://www.teamfortress.com`
[16] `http://apbreloaded.gamersfirst.com`
[17] `http://en.wikipedia.org/wiki/Massively_multiplayer_online_game`

- Complex game engines
- Linked to social networks
- Linked to credit cards

One of the reasons why MMOG require credit cards details from players is that it's pretty common for such kind of games to have at least one of the following features:

- Monthly subscription fee, in order to play the game
- In-game store, to buy in-game objects

The in-game market using real money is usually built upon a *microtransaction* model. "Microtransaction (also referred to as in-app billing or in-app purchasing) is a business model where users can purchase virtual goods via micropayments. Microtransactions are often used in free-to-play games to provide a revenue source for the developers, although they can also occur in non-game software. While microtransactions are a staple of the mobile app market, they are also available on traditional computer platforms such as Valve Corporation's Steam platform."[18]

MMOG are usually composed by a server part, which is kept by the vendor itself, and a client part, which is given to the customers (players). There are several things to keep in mind while trying to find vulnerabilities in such games. Let's start by considering the first part: the servers.

6.1 Server-side vulnerabilities

As stated previously the main problem while hunting for vulnerabilities in MMOG server-side is that the server is not (usually) shipped along with the client, which means that the server is hosted by the vendors or a third-party hosting service. Because of this performing *live* testing on third-party systems is not usually a good choice as it may end up in legal issues. An alternative strategy can be using a server emulator. As you may know there are several emulators for almost every MMOG available on the market. The main issue in using such approach is that a bug found on an emulator is usually a emulator-only bug, which means that it will not work on real servers.

6.2 Client-side vulnerabilities

Game clients are delivered to the players, because of that a bug hunter gets easy access to its content, so it is usually like auditing a normal application, but with a few caveats. The first one especially MMOG, tend to use anti-cheating solutions, which are getting smarter and they tend to use rootkit like approaches. An interesting example of such kind of protection, which is used by Blizzard's World Of Warcraft[19] is the Warden. "Warden (also known as Warden Client) is an anti-cheating tool integrated in many Blizzard Entertainment games. While the game is running, Warden uses operating system APIs to collect information about certain software running on the user's computer and sends it back to Blizzard servers as hash values to be compared to those of known cheating programs or simply as a yes/no response (whether a cheat was found). Some privacy advocates consider the program to

[18] http://en.wikipedia.org/wiki/Microtransaction
[19] http://worldofwarcraft.com/

be spyware."[20] Because of these defenses the game can detect our testing attempts and slow down our analysis. Another caveat is due to the monthly subscription fee. In most cases to test the client of a MMOG we need to pay a monthly fee to have access to the game itself, but if the anti-cheating subsystem detects our debugging activities we are very likely to be banned, and in some cases, we will waste our money.

7 Conclusion

Vulnerabilities in games is a very interesting field, which consists of thousands of possible *attack vectors* (games) and millions of *targets* (players) that are not aware of the risk of playing online games. To make this field even more interesting there is the fact that games usually run on players' systems with full privileges, in most of the cases this is due to the anti-cheating solution used to protect the game itself.

8 Game Over

Figure 12: Play safe :]

[20] http://en.wikipedia.org/wiki/Warden_(software)

9 Revision History

- 15 March 2013: Version 1.2 released.
- 25 January 2013: Version 1.1 released.