Video Game Information Security and RCE In Source Engine Games

Applied Information Security, KSAPINS1KU

Jonas Sylvain Jersild Balin
jbal@itu.dk

IT UNIVERSITY OF COPENHAGEN

IT University of Copenhagen
Denmark
23rd of February 2022

Contents

1 Introduction

In recent years, the video game industry has grown dramatically in size, surpassing both the global movie industry and North American sports industry combined [15]. The blockbuster gaming space is dominated by games with immense technical complexity, made by teams of hundreds of people spanning a large number of disciplines. Despite the seemingly ever-growing scale of video games, the development of blockbuster video games suffers from a culture of "crunch", where studios encourage employees frequently work overtime to reach deadlines [12]. In conjunction with the culture of crunch and rushed development cycles, video game QA contractors frequently find themselves lacking resources to properly ensure a game's quality before it ships [9].

With all of these factors taken into consideration, it is perhaps not surprising that video games are an appealing target for actors seeking to exploit security vulnerabilities in software.

This paper is a broad exploration of information security related concerns related to video game software. First, the paper will explore what assets can be harmed in video game software vulnerabilities, then there will be a section on common security vulnerabilities in video games. Finally, as a case study, there will be an examination of an RCE vulnerability discovered in Valve's Source engine games in 2019.

2 Video game software assets

As video games have gotten more complex, the assets malicious actors could target have equally become more significant and/or lucrative. Unlike other audiovisual media, video games' potential assets can range from the completely minor, e.g. an arbitrary score in a single player game, to trade-able digital items that can be worth thousands of euros. Beyond the assets that video games themselves carry, video game software vulnerabilities can also compromise the assets of the software's users, for example by committing RCE attacks on unwitting players.

Due to the wide range of assets in and around video game information security, broadly categorizing them could be beneficial. Some of these assets will be listed below:

- Game data/memory
- Digital items, currency
- Game servers
- Confidential game data/code
- Client machines

The categories above are not necessarily mutually exclusive nor far from exhaustive of the list of assets malicious actors could target. They do, however, serve well as a brief overview into the assets malicious actors could seek to gain/destroy.

Based on the aforementioned assets, numerous harm triples could be theorized.

- <modify, game data, loss of progress>
- <manipulate, servers/network packets, loss of availability>
- <breach, code leaks, loss of confidentiality>
- <overflow, client machines, RCE>

The next section will provide a brief overview of the typical vulnerabilities attackers will seek to use.

3 Brief overview of common video game software vulnerabilities

At the basest level, the most common vulnerabilities in video games are bugs and glitches that can be produced without external software, for example by abusing the game logic or the physics engine. A typical example of this could be stressing the game server by performing a high volume of intensive actions (e.g. dropping a large amount of items suddenly in a game that server-synchronizes ground items) [6].

Beyond vulnerabilities that can be performed within the game, one of the most common exploits of security vulnerabilities in video games revolves around modifying memory, often done through "trainers". These are external software that modifies or freezes the memory addresses of a game to alter its behavior. For example, freezing the value of the player character's hit-points, such that the player is invulnerable.

The most benign use of these trainers is in modifying single player data. In this instance, it could be argued that no harm is done since it is only impacting the player who is actively seeking out the trainer's effects. Memory freezing/modification can become more malicious when used in online sessions to disrupt the integrity or availability of other players' games. A particularly egregious example of this was when malicious actors in *Grand Theft Auto: Online* used trainers to non-consensually have other player characters engage in sexual animations, these being present in the code for another context [7].

A step up from using trainers are attacks based on packet manipulation. A vast majority of blockbuster online games use UDP as this protocol allows for faster packet transmission compared to TCP which requires a three-way handshake between sender and recipient[1]. Figure 1 illustrates the difference between TCP and UDP communication. Through packet sniffing methods not dissimilar from attacks in other contexts, malicious actors can create "hacks" that ruin the integrity of a competitive game. For example, "wallhacks" that allow players to see enemy players' positions through walls, with external software that sniffs the network for packets related to the game, extracts packets related to player coordinates and projects these onto the game.

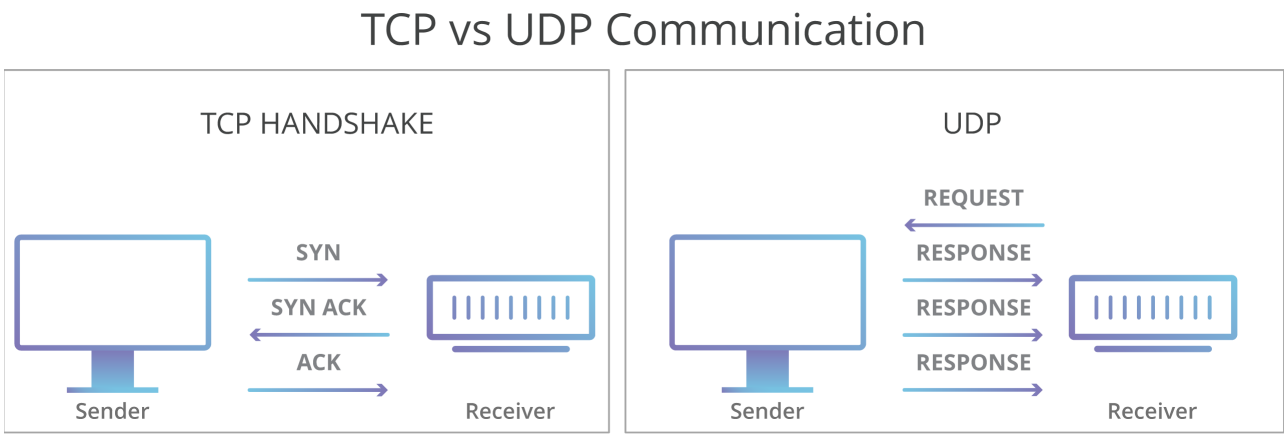

Figure 1: TCP and UDP network protocols[1]

Beyond all of the aforementioned vulnerabilities, any technology that is implemented in a given video game naturally carries over its vulnerabilities. For example, games that allow text inputs (such as chat functions) carry the potential of the same security vulnerabilities found in text inputs in other media. For example, malicious code injections in chat, e.g. HTML injections that can crash other players' games [4]. The same goes for DDoS attacks, which can damage availability of game servers or, if the game shares its clients' IP-addresses, can be used to target individual players' networks [3].

4 Source Engine Remote Code Execution

This section will describe a severe security vulnerability that exploited a game's engine along with the accompanying social network infrastructure. The specific case that will be discussed revolves around an RCE vulnerability that could be triggered by injecting malicious code into Steam invite strings, in games built on the Source engine. The vulnerability was reported in 2019 and patched in 2021, where a full write-up by the discoverers, Secret.Club, was publicized.

Source is a game engine developed by Valve software with initial release in 2004. The engine has powered all of Valve's mainline games, which include some of the most played games on the PC platform, including the *Counter-Strike*, *Half-Life*, *Left 4 Dead* games, and more. In 2015, the Source engine was replaced by the Source 2 engine as foundation for Valve's games, though any prior released games, including the aforementioned, still use the Source engine. Cumulatively these games host over one million concurrent players online each day [11].

A feature of the Source engine is the built-in developer console. This is a command-line interface that allows users to configure many aspects of their game experiences [13]. The console gives a high degree of customizability that are not accessible via the GUI. For example, creating complex button key-binds or adjusting server settings such as the gravity in the game (this setting being only available to admins of the given server).

Beyond creating Source, Valve is also the developer of the most popular PC gaming platform and marketplace, Steam. Steam serves as a point of entry for many PC gamers, as both a launcher, social media platform, and store for users' games. The social aspect of Steam allows players to befriend each other, track each others' activities across games, engage in VOIP and text chat across games.

Steam also allows for sending game invites, e.g. player A is in a session and can send player B an invite link. When player B clicks the link, the game will launch (if it is not already open) and it will connect B to A's session. This feature is widely used, but only available in games that integrate the Steamworks API.

A vulnerability discovered in 2019 found that coupled with the Source engine's developer console, Steam invites could be used to remotely execute code on other clients.

The following description is highly based on the vulnerability discoverers' write-up, which is accessible **here**.

4.1 How Steam invites work

Figure 2 shows the documentation of the `InviteUserToGame()` method. This is the method that is executed when games, using the Steamworks API, send session invites from one user to another. This function takes two parameters: the invitee's SteamID and a connection string. The connection string is a command-line function that gets added when launching the game, informing the game client of which session the user of the given SteamID wants to join.

Source games can be launched with command line options. The purpose of this can for example be for a map-maker to launch directly into a map they're working on.

```
hl2.exe -game cstrike -dev -console -fullscreen +sv_lan 1 +map MyMap
```

The above line is an example of a command line argument that launches *Counter-Strike*, enables developer mode, console, fullscreen, creates a LAN server and launches MyMap. A launch option to go directly to a server would look something like this:

`hl2.exe -game <game> +connect <64bit lobby ID>`. This is the kind of launch command that

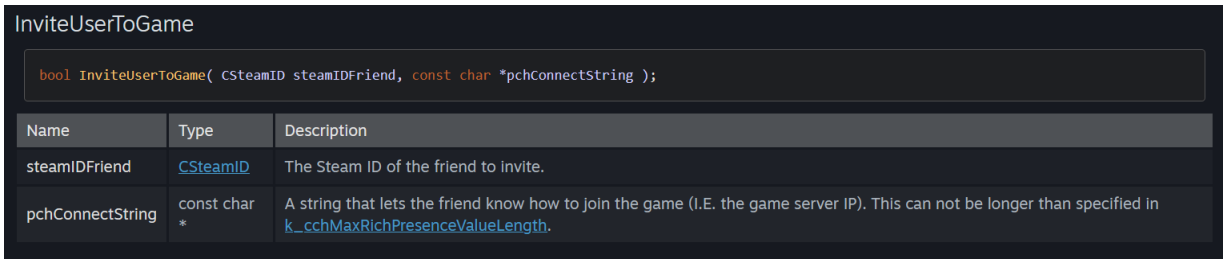

Figure 2: Steamworks API game invite method [14]

would be used when a player accepts an invite to a Source game session.[1]

The vulnerability here is that the launch parameters of Source games can include all the commands the console has. Given how powerful the Source console is, the vulnerability can reach critical levels.

4.2 RCON connections

Remote Console (RCON) is a Source protocol that allows console commands to be issues to a server without having access to the machine. For example, a server owner who wants to adjust the time limits of rounds do so by issuing the following commands:
`rcon_address: <server IP address> rcon_password <password>`
`mp_roundtime <value in seconds>`
However, by injecting RCON connect commands into the Steam invite, attackers could easily get a user's IP-address. Adding `+rcon_address yourip:yourport +rcon` to the Steam invite's second parameter will effectively leak the client's IP-address.

4.3 Source code

The source code for the Source engine is available to licensees, and beyond that, has had a long history of leaks, with some dating back to 2007 [2], and some as recent as 2018 [8]. This level of transparency means curious actors can poke around and potentially find security vulnerabilities.[2]. Even if some of the publicly available code is years old, large parts of the engine still rely on it. The following RCE vulnerability, discovered in 2019, relies on RCON protocol code that was present in the Source engine even in the 2007 iteration.

[1] Note that Source games always run on the Half Life 2 executable, hence the reference to `hl2.exe`

[2] It also means good-faith actors can find and warn Valve of vulnerabilities, which Valve encourages with a bug bounty program

Listing 1 shows the `SaveRemoteScreenshot` function from the `cl_rcon.cpp` file in the Source engine code. This function is present in every iteration of the Source engine SDK I could find from 2007 up to the most recent 2017 leak. This function, also, is crucial to facilitating RCE attacks via buffer overflow.

```
1  //-----------------------------------------------------------------------------
2  // We've got data from the server, save it
3  //-----------------------------------------------------------------------------
4  void CRConClient::SaveRemoteScreenshot( const void* pBuffer, int nBufLen )
5  {
6          char pScreenshotPath[MAX_PATH];
7          do
8          {
9                  Q_snprintf( pScreenshotPath, sizeof( pScreenshotPath ),
10                 "%s/screenshot%04d.jpg", m_RemoteFileDir.Get(), m_nScreenShotIndex++ );
11         } while ( g_pFullFileSystem->FileExists( pScreenshotPath, "MOD" ) );
12
13         char pFullPath[MAX_PATH];
14         GetModSubdirectory( pScreenshotPath, pFullPath, sizeof(pFullPath) );
15         HZIP hZip = OpenZip( (void*)pBuffer, nBufLen, ZIP_MEMORY );
16
17         int nIndex;
18         ZIPENTRY zipInfo;
19         FindZipItem( hZip, "screenshot.jpg", true, &nIndex, &zipInfo );
20         if ( nIndex >= 0 )
21         {
22                 UnzipItem( hZip, nIndex, pFullPath, 0, ZIP_FILENAME );
23         }
24         CloseZip( hZip );
25 }
```

Listing 1: SaveRemoteScreenshot function in Source engine code, C++ [14]

Listing 1 is a function used by the Source engine to handle a type of packets, `SERVERDATA_SCREENSHOT_RESPONSE`. This packet, theoretically, could contain anything, but the client handles the packets by treating them as ZIP archives, which it unzips and extracts a file called `screenshot.jpg` found within. This file is placed in the `pFllPath` variable, which is the root folder of the given Source game. These RCON packets can be sent without client request, opening up for the possibility of attackers uploading arbitrary files when a user accepts a game invite.

4.4 Exploiting XZip to achieve remote code execution

This part of the exploit is significantly more complex than the previous sections. For the sake of brevity, the following description will condense parts of it.

The code in listing 1 uses an old C++ library called XZip to handle zip manipulation. Line 19 calls `FindZipItem()` to fetch information about `screenshot.jpg`. This method relies on `TUnzip::Get` to handle a zipped file and retrieve information about it. A part of this procedure involves parsing the ZIP file and processing the central directory file header.

ZIP files are structured in a specific manner. Each ZIP file is identified by the presence of an end of central directory record. As the name implies, the central directory contains file listings.

As part of the `TUnzip::Get` method, several functions are invoked in a chain to locate the requested file in the ZIP archive. This sequence can be illustrated as such:

1. `TUNzip::Get` parses the ZIP archive
2. The local file header offset is discovered (allowing locating the position of the compressed file in the archive)
3. `TUNzip::Get` calls `unzlocal_CheckCurrentFileCoherencyHeader` and passes the local file header offset
4. `lufseek` is called to set the internal file pointer to the local file header and returns
5. `unzlocal_getlong` is called to read the bytes that identify the local file header by reading each byte individually (invoking `lufseek` and `unzlocal_getByte` each time)

At this point, the ZIP archive is a byte BLOB[3] stored in memory.

In this process, the line `s->cur_file_info_internal.offset_curfile` can be modified by changing the corresponding field in the ZIP central directory. This opens the possibility of stack smashing, a well documented method of causing buffer overflow, allowing attackers (in this case) to execute assembly code on the target machine.

The stack smashing is achieved by manipulating values such that a calculation underflows, this is done by creating a ZIP archive with the local file header offset set to `0xFFFFFFFE`. Internally, the `lufread` function invokes `memcpy`. `memcpy` is a C++ function that copies a block of memory, specifying destination, source of copy, and the number of bytes to copy. Through manipulating the central directory and thereby the specified file such that an underflow is caused, the value given can be one such that the equivalent of this call executes: `memcpy(ptr, (char*)stream− >buf - 2, stream− >len + 2);`[4]

In this case, `ptr` points to a local variable declared by `unzlocal_getByte`. This variable is only a single byte, but the object to be copied into it is larger. This corrupts the stack.

At this stage, the stack is smashed, the buffer overflow has been achieved, and it is possible to execute code on the machine. From here, the discoverers of the exploit created a ZIP archive with the local file header offset, and ROP gadgets. ROP is Return Oriented Programming, and for the purposes of stack smashing, is often used to create programs by chaining existing instructions within a program's address space. This can aid in avoiding non-executable stacks (DEP) and code signing mitigations [10].

By trial and error, and based on faulting EIP values, the discoverers could observe where to put the gadgets. Furthermore, while the Source engine generally used Address Space Layout Ranzomization (ASLR) to increase security, a dll the game uses called `xinput1_3.dll` was found to have it disabled. This means that the file's address in memory is not randomized, making it easier for attackers to find memory addresses associated with the process, and execute the payload[5].

At this point, the smashed stack means that would-be attackers could attempt to move around in

[3] Binary Large Object

[4] `ptr` is the pointer, `char*(stream)− >buf-2` is the copy source, `stream− >len + 2` is the value to copy

[5] ASLR does not provide a security guarantee but does add a layer of obfuscation attackers would have to penetrate

memory and execute the payload delivered by the initial command. This is not a guaranteed success, though, as memory allocation can vary there is not a catch-all solution. Numerous attacks against a target could still be needed before success is achieved. However, attackers could easily avoid this problem by using the Source developer console.

4.5 Unlimited tries

As described, the vulnerability is essentially an exploit based on injecting Source engine launch parameters into invite strings using the Steamworks API. The developer console of Source is quite powerful and the parameters can be exploited to essentially grant attackers unlimited hack attempts without the user even being aware. By injecting this code:
`+bind "tab" "+showscores;rcon_address ip:port;rcon" +host_writeconfig`
The victim who accepts the invite is unwittingly binding their tab key to two functions.

1. showing the scoreboard (standard keybind for all Valve Source games)
2. RCON connecting to the target that hosts the payload

In other words, without the victim noticing or being informed of any modification to their settings, every time they press tab to view the scoreboard, they are unwittingly subjecting themselves to an IP logger and a possible RCE attack. Further compounding this issue is that the keybinding is saved to disk and will persist across sessions. Meaning that regardless of which server the client is connected to, and no matter how much time has passed, the vulnerability will remain.

Fortunately, ridding the attack is as simple as just unbinding the key with a simple console command, yet the stealthy nature of this attack means it is very unlikely most would ever discover this.

5 Conclusion

This paper has (very) briefly laid out the assets related to video game software, harms that can be committed to them, and the typical attacks malicious actors would do. As a case study, an RCE exploit found in Valve Source engine games with the Steamworks API was examined. This exploit is far from the only one of its type - RCON protocols exist for many games and exploits using them have been found in other games [5]. Beyond RCON vulnerabilities, the complexity of video game software means they have numerous points of entry for malicious actors. Another typical use-case being packet sniffing and manipulation.

Despite gaming's recreational nature, they are no less subject to security vulnerabilities, exploits, or malicious actors than other types of software. In a development culture that is already pressured by crunch and lacking QA, it is not surprising if information security is not always top priority for game publishers. Nonetheless, it should be prioritized due to the high potential for exploits and vulnerabilities in software as complex as AAA games tend to be.

References

[1] Cloudflare. *What is User Datagram Protocol (UDP/IP)?* URL: https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/ (cit. on p. 4).

[2] Gbps. *Exploiting the Source Engine (Part 1).* URL: https://ctf.re/source-engine/exploitation/reverse-engineering/2018/08/02/source-engine-1/ (cit. on p. 6).

[3] Richard Hummel. *DDoS Attacks Against Online Gamers and the Damaging Ripple Effect.* 2021. URL: https://www.infosecurity-magazine.com/opinions/ddos-attacks-online-gamers/ (cit. on p. 4).

[4] Matt Kim. *New World Reportedly Has a Vulnerability That Makes It Possible To Crash Players Through the Text Box.* 2021. URL: https://www.ign.com/articles/new-world-text-box-html-vulnerability-game-crash (cit. on p. 4).

[5] Donato Ferrante Luigi Auriemma. *Owning Multiplayer Online Games.* 2012. URL: https://revuln.com/files/Ferrante_Auriemma_Multiplayer_Online_Games_Insecurity.pdf (cit. on p. 9).

[6] Meloncube. *Minecraft lag types and how to fix them.* 2021. URL: https://www.meloncube.net/clients/knowledgebase/105/Minecraft-lag-types-and-how-to-fix-them.html (cit. on p. 3).

[7] Rob Brooks Michael Kasumovic. *Virtual rape in Grand Theft Auto 5: learning the limits of the game.* 2014. URL: https://theconversation.com/virtual-rape-in-grand-theft-auto-5-learning-the-limits-of-the-game-30520 (cit. on p. 4).

[8] Lindsey O'Donnell. *Valve Confirms CS:GO, Team Fortress 2 Source-Code Leak.* 2020. URL: https://threatpost.com/valve-confirms-csgo-team-fortress-2-source-code-leak/155092/ (cit. on p. 6).

[9] Jason Schreier. *Quality Assured: What It's Really Like To Test Games For A Living.* 2017. URL: https://kotaku.com/quality-assured-what-it-s-really-like-to-play-games-fo-1720053842 (cit. on p. 2).

[10] Ben Simmonds. *ROP (return oriented programming) chains.* 2019. URL: https://www.bencode.net/posts/2019-09-07-rop-chain/ (cit. on p. 8).

[11] *Steamcharts.* 2021. URL: https://steamcharts.com/ (cit. on p. 5).

[12] Michael Thomsen. *Why is the games industry so burdened with crunch? It starts with labor laws.* 2021. URL: https://www.washingtonpost.com/video-games/2021/03/24/crunch-laws/ (cit. on p. 2).

[13] Valve. *Developer Console.* URL: https://developer.valvesoftware.com/wiki/Developer_Console (cit. on p. 5).

[14] Valve. *ISteamFriends Interface.* URL: https://partner.steamgames.com/doc/api/ISteamFriends (cit. on pp. 6, 7).

[15] Wallace Witkowski. *Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic.* 2021. URL: https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990 (cit. on p. 2).