Packet System:

I use a byte serialisation system to transmit data between clients and the server. This system uses a byte enum to define the type of information that any one packet contains, then though the use of a Packet class inheritance structure, provides processing for that type of packet, and in the case of the server, redistribution of that packet to the other clients on the network.

Packets start on the client as data attached to a game object. This data is translated into a packet that can contain that datatype (ie, TransformPacket for an object transform), then it is serialised using C# MemoryStream(1) and BinaryWriter(2) if it is generated in the server or on a unity client, and if the packet is generated on an unreal client, it uses FMemoyrWriter(5). This produces either a C# array, or an Unreal TArray (6) (depending on where the packet is generated), which is then sent to the server. The server then checks the packets origin (which is stored at position 0 in the packet), before sending the packet to all other clients currently connected. This is done to minimise the number of excess packets sent to the client.

[Diagram of Packet structure, possibly related to a binary dump of a packet]

//What would I do if I continued extending the engine and server

**Cheating and Exploitation**

Cheating is an ongoing problem in online multiplayer games. Though the use of either software or hardware, unscrupulous actors will use cheats to gain an advantage over the other players in the game.

**Types of Cheating:**

**Aimbotting:**

Aimbotting is the process of automating the aiming process, allowing for a cheater to target enemy players with unnatural speed and accuracy (7). Aimbots can be implemented at either the software or hardware level, with software level aimbots using the same device as the client to manipulate the players aiming ability, whereas hardware aimbots use external computing using either other computers to intercept and manipulate mouse input or moving the mouse using computer-controlled robotics.

//talk about modes of aimbot (passive, toggle, autoclick)

Aimbots are primarily found in multiplayer shooters, games that involve players in a competitive environment needing to aim at each other in the game to kill their opponents. Aimbots provide a clear advantage in these games, where precision and accuracy are both needed to be successful. Using a computer to do the aiming for you or to fix your aiming on the fly massively decreases the requirement of skill in these games, leading to players appearing to be of a much higher skill than they really are.

**Extra Sensory Perception (ESP)**

Extra Sensory Perception is a subclass of game hacks that reveal information to the player that they otherwise wouldn’t be able to see. This includes wallhacks, which shows enemies though walls though the use of an overlay and radar hacks, which shows enemy locations on a minimap component, allowing players to gain an advantage in having greater knowledge of their enemies positions.

ESP hacks can also provide cheaters with information specific to each other player, such as health total, weapons and consumable counts. In game, this allows cheaters to target players with particularly dangerous weapons or items first, or hunt down players with move valuable gear and equipment.

ESP cheats are used across the spectrum of competitive gaming, not just in shooters. They have seen extensive use in Real Time Strategy games, where knowing information about the opponent’s unit positions and status (health, upgrades), allowing for advanced knowledge of opponent movements and strategy, allowing for the cheater to optimise their own units positions and upgrades to respond perfectly to their opponent.

Their use in shooters (both first person and third person) allows for cheaters to see their opponents though walls, allowing for them to line up shots well in advance of seeing a player, then shoot much sooner when the player appears on the screen, as the cheater already knows where their opponents are going to appear from.

**Prevention and Punishment Methods:**

Just as there are many methods of cheating in games, there are many methods to prevent and/or punish cheating. These range from the relatively simple (allowing players to vote kick cheaters from servers based on an in game option) to the complex (statistical analysis of gameplay to determine likelihood of any individual player cheating) (7).

While it is a very simple method of punishing cheaters, allowing the players during a game to vote kick suspected cheaters can be an immediate way of punishing the cheater as well. However, it doesn’t do much too actively prevent cheating as it is a reactive rather than a proactive solution, and doesn’t provide any safeguarding against false positives or malicious use by players to remove people who are performing better than them from the game.

Another method of protection, executed on the server side is to not have the game clients manage the game state. By doing this, the server not only acts as a connection broker between clients, but also as arbiter of the game state, and the clients send inputs to the server. As the server now has full control over the game state, there can be extra checks implemented to ensure that the inputs the player is sending comply with the game rules. This also makes it impossible for players to utilise hacks that directly manipulate their position, as the player doesn’t have any control over their position in the game world, only the direction and speed they wish to move, both of which can be validated on the server before processing. At the point where the server detects illegal input attempts, the developer can decide how they wish to handle the attempted cheating.

The above methods are server side methods of preventing hacking, but the server is only half of the hardware in a multiplayer game. There are methods to prevent hacking on the client side, however these often require the use of more extensive security measures to prevent hackers from easily accessing the anti-cheat methods, and therefor bypassing them.

One such method of prevention is code encryption. Code encryption can be done in a number of ways, one of the simplest measures code packing (8). Code packing is the process of making the executable code unreadable to static analysis (the technique by which an executable code is analysed while the code is not running) then unpacking it at runtime. This provides a small amount of protection, but is vulnerable to reverse engineering attacks if the hacker can figure out what kind of packing tool was used, as many tools can also unpack code.

Code encryption can also be achieved by using an on demand encryption method (8), in which the executable decrypts code as it needs to use it. This form of inline code decryption makes static binary analysis more difficult, the decrypted code will only be able to be read in the brief time between it being called and consequentially decrypted and when it is re-encrypted. While this sounds like it would introduce significant overhead, but has been observed to have a small impact on the runtime speed of the executable (9). This method of protecting game code also means that attempting to modify the executable in place will produce unexpected behaviour at runtime and can reasonably be assumed that the unexpected behaviour will cause the game to crash.

1. <https://learn.microsoft.com/en-us/dotnet/api/system.io.binarywriter?view=net-8.0>
2. <https://learn.microsoft.com/en-us/dotnet/api/system.io.binaryreader?view=net-8.0>
3. <https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Runtime/Core/Serialization/FMemoryReader/__ctor>
4. <https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Runtime/Core/Serialization/FMemoryWriter>
5. <https://dev.epicgames.com/documentation/en-us/unreal-engine/array-containers-in-unreal-engine>
6. Cheating: Gaining Advantage in Videogames, Mia Consalvo
7. <https://www.i3d.net/countering-scourge-of-cheating-in-games/>
8. Comparative Study of Anti-cheat Methods in Video Games, Samuli Lehtonen
9. Towards Tamper Resistant Code Encryption: Practice and Experience Jan Cappaert1 , Bart Preneel1 , Bertrand Anckaert2 , Matias Madou2 , and Koen De Bosschere