Lab 8

Author: Lukas Kmitas

Questions:  
1. Cheating in online games is the action of pretending to comply with the rules of the game, while secretly subverting them to gain an unfair advantage over an opponent. Describe an example of cheating from client side. Please include details of how this cheating works and a mechanism to prevent it.

2. Please provide an example of server-side attack. Please include details of how this attack works and a mechanism to prevent it.

**Q1:**

**Aimbots**

Aimbots automate the targeting process to give the player an unnatural accuracy. They work by scanning the game data within the client's computer memory to locate other players' positions. Once an enemy player is located, the aimbot automatically adjusts the player's aim to target the enemy, often aiming for high-damage areas like the head. This allows the cheater to shoot with inhuman accuracy, significantly reducing the skill gap between players.

**Mechanism to Prevent Aimbots**

Preventing aimbots requires a multi-faceted approach combining software solutions, community management, and game design adjustments. One effective mechanism is the implementation of server-side checks. Here's how it can work:

1. **Anomaly Detection:** The game server analyses players' behaviour patterns for anomalies. A player who consistently makes inhumanly accurate shots over several matches could be flagged for further review. This detection can be based on statistical analysis comparing the player's performance against average human capabilities.
2. **Heuristic Analysis:** Implement algorithms that specifically look for patterns indicative of aimbot use, such as instant aim adjustments, snapping to targets too quickly, or maintaining a 100% headshot rate over multiple engagements.
3. **Client Integrity Checks:** Regularly verify the integrity of the game files on the client's computer to ensure no modifications have been made that could facilitate cheating. This could involve hash checks or requiring a secure, unmodifiable client.
4. **Challenge-Response Tests:** Occasionally, the server can issue challenges that a normal client can respond to but would disrupt or be failed by an aimbot. For example, subtly altering hitboxes or player position data sent to suspected cheaters to see if their performance is affected.
5. **Encrypted Game Data:** Encrypting critical game data being processed on the client side can make it harder for cheat developers to create aimbots, as they won't easily find the data they need to manipulate.
6. **Community Reporting and Review:** Enable a system where players can report suspected cheaters, which, if combined with a review process, can help catch and penalize cheaters based on community monitoring.

These methods, combined with constant updates and patches to address newly discovered exploits, form a robust defence against aimbots and other forms of client-side cheating. However, it's a constant arms race between cheat developers and game developers, requiring ongoing efforts to maintain a fair play environment.

**Wallhacks**

**How They Work**: Wallhacks allow players to see through solid objects or walls, making it easier to spot opponents. These cheats work by manipulating the game's rendering functions to display enemy players in situations where they should normally be obscured. This can involve changing the properties of game textures to make them transparent or overlaying additional information (like skeletons or health bars) over players' models, regardless of their position on the map.

**Prevention Mechanisms**: Preventing wallhacks often involves server-side validation of what information should be sent to each client. By ensuring that the server only sends information about visible players to a client, the game can make it much harder for wallhacks to provide useful information. Additionally, the use of obfuscation techniques and regular updates to the game's rendering engine can disrupt the functionality of wallhacks.

**Speed Hacks**

**How They Work:** Speed hacks alter the game's understanding of time or the player's position, allowing a player to move faster than intended. This can be achieved by manipulating the game's client-side code to increase movement speed or by sending altered packets to the server that falsely report the player's position.

**Prevention Mechanisms:** Implementing strict server-side checks on player movement can prevent speed hacks. The server can calculate how fast a player should be able to move from one point to another, considering their current speed, abilities, and the game's physics. If a player exceeds this speed, the server can correct their position or flag the player for cheating.

**Botting/Scripting**

**How They Work:** Bots or scripts automate gameplay, performing tasks like farming resources, grinding levels, or even playing the game without human input. Scripts can also perform complex combos or actions with perfect timing, which would be difficult for human players.

**Prevention Mechanisms:** Detection mechanisms include analysing player behaviour for patterns that suggest non-human operation, such as repetitive movements or actions executed with inhuman consistency or speed. Captchas or interaction challenges unsolvable by bots can also be used sporadically to confirm human control.

**Resource Modification Cheats**

**How They Work:** These cheats involve directly modifying game resources or memory values to gain advantages, such as altering weapon damage, player health, or in-game currency. This is done by editing the game files or using third-party software to change values stored in memory during runtime.

**Prevention Mechanisms:** To combat this, games can implement checksum or hash verification to ensure game files and memory values haven't been tampered with. Secure and encrypted communication between the client and server, alongside constant integrity checks, can help identify and prevent these cheats.

**No-Recoil Cheats**

**How They Work:** No-recoil cheats remove the recoil effect from weapons, allowing players to shoot with perfect accuracy without needing to adjust their aim. This can significantly lower the skill required to play effectively.

**Prevention Mechanisms:** Detecting no-recoil cheats involves analysing firing patterns and the consistency of shot accuracy. If a player consistently manages to control recoil beyond what is normally possible, it could indicate cheating. Behavioural analysis and statistical comparison with the player's historical data can help identify abnormal performances.

**Q2:**  
An example of a server-side attack is a **SQL Injection (SQLi)** attack, which targets databases through vulnerabilities in a web application's software. SQL Injection can allow attackers to manipulate or steal data, escalate their privileges within the system, or even execute administrative operations on the database, like shutting it down or deleting data.

**How SQL Injection Works**

SQL Injection exploits vulnerabilities in the input validation of an application. When user input is directly included in SQL queries without proper validation or sanitization, an attacker can insert or "inject" malicious SQL code into these queries. This can alter the intended function of the SQL query, leading to unauthorized data access or manipulation. For example, consider a login form on a website that checks the username and password against a database. An attacker could input a specially crafted username or password that includes SQL code. If the application does not properly sanitize this input, the injected SQL code could be executed by the database, potentially allowing the attacker to bypass authentication or access sensitive information.

**Prevention Mechanisms**

Preventing SQL Injection requires a combination of secure coding practices, input validation, and the use of prepared statements:

1. **Input Validation and Sanitization:** Ensure that all user inputs are validated against expected formats and sanitized to remove or neutralize potentially malicious content. This includes not only form inputs but also data received from cookies, APIs, and anywhere external data can enter the system.
2. **Use of Prepared Statements (with Parameterized Queries):** Prepared statements ensure that an attacker cannot change the intent of a query, even if SQL commands are inserted by an attacker. In a prepared statement, SQL code is defined first, and then, parameters are passed into the query, preventing the execution of injected malicious SQL.
3. **Least Privilege Access:** Configure database accounts with the principle of least privilege in mind. Application database users should have only the permissions necessary to perform their tasks. This limits the potential damage in the event of a SQL Injection attack.
4. **Regular Security Audits and Code Reviews:** Regularly review and audit code for vulnerabilities, especially those related to SQL Injection. Automated security scanning tools can help identify potential vulnerabilities, but manual code review is also essential for complex issues.
5. **Web Application Firewalls (WAF):** Deploy a WAF to help detect and block SQL Injection attacks and other common threats. A WAF can filter out malicious data, providing an additional layer of protection against attacks.
6. **Error Handling:** Customize error messages to avoid revealing details about the database structure. Generic error messages help prevent attackers from gaining insights that could facilitate an attack.
7. **Regular Updates and Patch Management:** Keep all software, especially database management systems and web application platforms, up to date with the latest security patches.

By implementing these preventive measures, organizations can significantly reduce their vulnerability to SQL Injection attacks, protecting their data and systems from unauthorized access and manipulation.