CMSC 498M: Chapter 10c
Sockets Programming and Cheating

Sources:
- "Beej’s Guide to Network Programming"
- "RakNet Manual"
- Lecture notes by Okan Arikan from CS 387 at UT Austin.
- "How to Hurt the Hackers", by Matt Pritchard (Gamasutra article).

Overview:
- Sockets programming
- RakNet.
- Common cheating attacks and possible fixes.

Overview

- Socket Programming
- RakNet
- Cheating in Multiplayer Games
Sockets Programming

Sockets:
- Provide a means for programs (running perhaps on different machines) to communicate with one another.

Socket types:
Stream sockets: (TCP: Transmission Control Protocol)
- Reliable two-way connected communication streams.
- Items arrive in the order they were sent.
- Virtually error-free.

Datagram sockets: (UDP: User Datagram Protocol)
- No connection - Just generate a packet and send it.
- Packets may not arrive in the order they were sent.
- Packets may not arrive at all.

User Datagram Protocol (UDP)

Most real-time games use the UDP protocol:
- Faster and with lower overhead than TCP. Great!

But there are consequences: UDP packets...
... are not guaranteed to arrive.
... are not guaranteed to arrive in order.
... are guaranteed to arrive with correct data, but have no protection from hackers intercepting and changing the data once it has arrived.
... do not require a connection to be accepted. (Assists cheating. For example, intercept the packet "Give ...blah... invulnerability," generate a copy of the message, and send it to the server anytime.)

...and global consequences:
- Unlike TCP, UDP does not provide flow control or aggregation, and so it is possible to overrun the recipient's capacity.
Socket APIs

Low-level Socket APIs:

- Berkeley Sockets API: for Unix.
- WinSock: for Windows.
  - Both provide essentially the same functionality.

Basic capabilities: (for Berkeley sockets, but Winsock similar)

- `socket(...)`: Create a socket (of either type).
- `gethostbyname(...)`: Map hostname (e.g., "mysite.com") to IP address.
- `bind(...)`: Connect a socket to a port on your local machine.
- `connect(...)`: Connect to a remote machine and port.
- `listen(...)`: Wait for input to arrive from a socket.
- `accept(...)`: Accept a request from another host to connect to you.
- `send(...)/recv(...)`: Send and receive data. For stream sockets
- `sendTo(...)/recvFrom(...)`: Send/receive (for datagram sockets).
- `close(...)`: Terminate communication.

Sockets Program Structure

Program Structure: typical client-server program structure (assuming TCP connection).

UDP is even simpler: Since listen, accept, connect are not needed. Replace calls to send/recv with calls to sendTo/recvFrom.
Overview

- Socket Programming
- RakNet
- Cheating in Multiplayer Games

RakNet: More Reliable UDP

RakNet:
- C++-based, open-source toolkit for (higher-level) UDP-based socket programming.
- Supports client, server, and peer-to-peer communication.
- Provides a layer over UDP, which addresses many of UDP’s shortcomings.

RakNet Enhancements:
- Can automatically resend lost packets.
- Can automatically order packets that arrived out of order.
- Protects transmitted data, and inform the programmer if that data was externally changed.
- Provides a connection layer that blocks unauthorized transmission.
- Transparently handles network issues such as flow control and aggregation (grouping many small transmissions into one packet).
RakNet: Quick Overview

RakNet Standard Headers:

```
#include "MessageIdentifiers.h"
#include "RakNetworkFactory.h"
#include "RakPeerInterface.h"
#include "RakNetTypes.h"
```

RakPeerInterface:
- The main RakNet object.
- You will usually only generate one of these.

```
RakPeerInterface* peer = RakNetworkFactory::GetRakPeerInterface();
```
- Base class for more specific objects: RakPeer, RakClient, and RakServer.

RakNet: Client Connection

Connection as a Client:
- Start up the network threads.

```
peer->Startup(1, 30, &SocketDescriptor(), 1)
```
- 1: maximum number of connections. For a pure client, we use 1.
- 30: thread sleep time (in msec).
  - 0 msec: good for games that need fast responses, such as FPS.
  - 30 msec: good response times with little CPU usage.
- SocketDescriptor(): specifies the port/IP-address to listen on. Since we want a client, we don’t need to specify anything.
- 1: Force RakNet to use a particular IP as host.
RakNet: Client Connection

**Connection as a Client:** (cont.)
- Connect to the RakNet server.
  ```cpp
  peer->Connect( serverIP, serverPort, 0, 0 );
  ```
- **serverIP:** IP address of the server.
  - Use "127.0.0.1" or "localhost" to connect to your own machine (testing).
- **serverPort:** Port you want to connect to on the server.
  - Any unused port number (in the range 0 to 2^31-1). Many are used by existing applications (ftp, smtp, http, etc.)
  - Ports over 32000 are generally open to whoever wants them.
  - E.g.: serverPort = 60005, clientPort = 60006.
- Last two arguments used for **passwords**.
- Only initiates the asynchronous connection process. You will receive:
  - **ID_CONNECTION_ACCEPTED** if successful and
  - **ID_CONNECTION_ATTEMPT_FAILED** if not.

RakNet: Server Connection

**Connection as a Server:**
- Connect to the RakNet server.
  ```cpp
  peer->Startup( maxConnectionsAllowed, 30, 
  &SocketDescriptor( serverPort, 0 ), 1 );
  peer->SetMaximumIncomingConnections( maxPlayersPerServer );
  ```
- **maxConnectionsAllowed:** Maximum simultaneous connections.
- **30:** thread sleep time (in msec).
- **SocketDescriptor:** Which port to listen to.
- **maxPlayersPerServer:** Maximum incoming connections to allow.
RakNet: Read a Packet

Read a Packet:

```
Packet* packet = peer->Receive();
```

- Returns 0 (NULL) if no input.
- Data may come from engine or other RakNet instances.

Packet Structure:

```
struct Packet {
    SystemIndex systemIndex; // Server only: Sender index
    SystemAddress systemAddr; // Who sent the packet
    unsigned int length; // Data length in bytes (Deprecated)
    unsigned int bitSize; // Data length in bits (Use this)
    unsigned char* data; // The data (Cast as needed)
    bool deleteData; // Internal use
};
```

First byte indicates data type

RakNet: Send a Packet

Send a Packet:

```
const char* message = "Hello World";
peer->Send( (char*) message, strlen(message)+1,
            HIGH_PRIORITY, RELIABLE, 0,
            UNASSIGNED_SYSTEM_ADDRESS, true);
```

- message: data to be sent.
- strlen(message)+1: length of data. Allow one additional byte for string's null (\0) terminator.
- HIGH_PRIORITY: Packet priority (also "LOW" and "MEDIUM").
- Reliability options:
  - UNRELIABLE: may not arrive and order of arrival arbitrary.
  - UNRELIABLE_SEQUENCED: in order, but may not arrive.
  - RELIABLE: guaranteed arrival, but order is not.
  - RELIABLE_ORDERED: guaranteed arrival, and in order.
  - RELIABLE_SEQUENCED: out of order packets are deleted.
- Final arguments indicate a broadcast to all RakNet systems.
RakNet: Shutting Down

Shutting Down:

```cpp
peer->Shutdown(300);
...
RakNetworkFactory::DestroyRakPeerInterface(peer);
```

- **Shutdown**: closes the connection. The connection can be restarted using `Startup()`.
  - 300: Indicates how long to wait (in msec) for remaining packets to be sent.
- **DestroyRakPeerInterface**: Shut RakNet down and free all memory.

For more information:
- See *Raknet Manual and Tutorials* (Warning: Doxygen documentation appears to be out of date.)

---

Overview

- Socket Programming
- RakNet
- **Cheating in Multiplayer Games**
Why Care About Cheats?

Achieving financial advantage:
- Competitive games with prizes are the obvious example (casinos).
- Virtual Economies:
  - People play the game, build good characters, and then auction them on eBay.
  - If they can cheat to obtain better characters, they are achieving unfair financial advantage.

Ruin your game play ⇒ ruining your profits:
- Online gaming is big business.
- Players tend to have a strong sense of fairness.
- If they believe they are being cheated, they will stop playing, and you will not make any money.

Cheating is principally on online issue:
- Single player cheaters only affect themselves, so who cares?

Observations About Cheating

Pritchard's Rules (Gamasutra article):
1. If you build it, they will come—to hack and cheat.
2. Hacking attempts increase as a game becomes more successful.
3. Cheaters actively try to control knowledge of their cheats.
4. Your game, along with everything on the cheater's computer, is not secure — not memory, not files, not devices and networks.
5. Obscurity ≠ security.
6. Any communication over an open line is subject to interception, analysis and modification.
7. There is no such thing as a harmless cheat.
8. Trust in the server is everything in client-server games.
9. Honest players would like the game to tip them off to cheaters, hackers hate it.
Common Cheating Attacks

**Reflex Augmentation:**
- Improve physical performance, such as the firing rate or aiming.

**Authoritative Clients:**
- Clients issue commands inconsistent with the game-play, or mimic the server.

**Information Exposure:**
- Clients obtain/modify information that should be hidden.

**Compromised servers:**
- A hacked server biases game-play towards the group that knows of the hacks.

**Bugs and Design Loopholes:**
- Bugs and design flaws are exploited.

**Environmental Weaknesses:**
- Differences or problems with the OS or network environment are exploited.

---

Reflex Augmentation

**Reflex Augmentation:** (a.k.a. Aimbots)
- Turning yourself into "the Terminator."
- Examples:
  - Aiming proxies: intercept communications, locate players, and shoot.
  - Rapid-fire proxies: take each shoot packet and replicate it.

**Fix #1:**
- Server validates player actions. Disqualifies players "too good" to be human.

**Fix #2:**
- Make it hard to insert invalid network packets.
- Encrypt packets: Must be fast, and so may be easy to crack.
- Encryption dependent on the game state or some random value.
- Serialize packets with a unique sequence of numbers. Hacker cannot copy or insert extra packets. (Requires reliable protocol.)
Encryption

Typical Encryption:
- A key, known only to intended users, is used to convert regular data into something that appears random.
- Hard to use encrypted data to obtain key or the original data.

How to come up with the key:
- Agree on it ahead of time, e.g. when software is purchased.
- Transmit it — key-exchange algorithms.
- Derive it from somewhere else in such a way that all parties derive the same key (e.g. from game state).

Most encryption algorithms work on blocks of a fixed size:
- Split large amounts of data into smaller blocks.
- Pad blocks that are too small.

Authoritative Clients

Authoritative Clients: (Example)
- One player's game generates bogus definitive event: e.g. "Player 2 just got 10,000 hit points."

How to Hack the Client:

Fix: Insert command request steps:
- Player requests an action, its validity is checked, it is sent out on the network, and added to the player's pending event queue.
- Incoming actions also go on the pending queue.
- Actions come off the pending queue, are validated again, and then are implemented.

If validation is hard to get right, try synchronization:
- Occasionally send complete game state around, and compare it.
- More practical: send something derived from complete game state.
Information Exposure

Accessing/Modifying Hidden Parameters:
- Modify the renderer to make walls transparent, modify maps to remove the fog of war.
- Display variables are stored/modified in memory, or read out and displayed elsewhere.
- Hackers use debugging tools to find the locations of key data in memory, and modify them transparently.

Fixes:
- Check that players agree on the values of certain variables, and the validity of actions (synchronization again).
- Check for invalid actions based on the correct display. (E.g., aiming through walls.)
- Compute statistics on drawing, and check (e.g. # of polygons drawn).
- Encrypt data in memory to avoid passive attacks.

Environment “Tweaks”

Decluttering the environment by disabling texture mapping.

Return to Castle Wolfenstein
Compromised Servers

Customizable Servers:

Fact: Some servers have customization options, and the community is allowed/encouraged to modify the server.

Fact: This is completely legal.

Naïve Users:
- Do not have the skills or knowledge to check whether the server they are playing on is altered.
- Will grow frustrated, blame the developer, and complain to friends.

Illegal Modifications:
- if (player.name->contains("My_Clan")) Damage = Damage * 0.80;

Solution:
- Warn players as they connect to the server, of any non-standard modifications (discovered through validation).

Exploiting Bugs and Design Flaws

Bugs:
- Some bugs enable cheating, such as a bug that enables faster weapon reloading, or one that incorrectly validates commands.

Poor Design Decisions:
- Embedding cheat codes in single-player mode makes it easy for a hacker to track down the variables that control cheats.
- Poor networking or event handling can allow repeat commands or other exploitations.
- Example (Age of Empires and Starcraft): All resource management is done after all events for a turn are processed. Poor networking allowed multiple cancel events on the queue, which restored multiple resources.

Solution:
- Avoid bugs and think carefully about the implications of design decisions on hacking.
Environmental Weaknesses

Environmental Weaknesses:
- Facilities to deal with the OS or network may leave you vulnerable to some forms of attack.
- Example: (Firestorm) Generate a message from system’s clipboard containing non-printing characters. Sending it to another user causes his program to abort.
- Interaction with almost any scripting language may leave you open to hacks not related to the game. (Your game could be a way in.)
- Network connection drops or overloading can cause problems.

Targeted/Indiscriminant Cheats:
- Some cheats destroy the game for all players, which can be useful if you are losing.
- Others affect a specific opponent. (E.g., your worst enemy.)

The Moral of the Story

Final Thoughts:
- You cannot prevent cheating completely.
- Try to make cheating as hard as possible (e.g., as hard as writing a new game).
- Do not trust information from others.
- Limit the potential damage.
- Test for anomalous/unrealistic behavior.
Summary

Summary:
- Socket programming
- RakNet
- Common cheating attacks and possible fixes.