

---

## 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.

Chapter 10, Slide 1  
Copyright © David Mount and Amitabh Varshney

## Overview

---

- Socket Programming
- RakNet
- Cheating in Multiplayer Games

Chapter 10, Slide 2  
Copyright © David Mount and Amitabh Varshney

## 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**.

Chapter 10, Slide 3  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 4  
Copyright © David Mount and Amitabh Varshney

## 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.

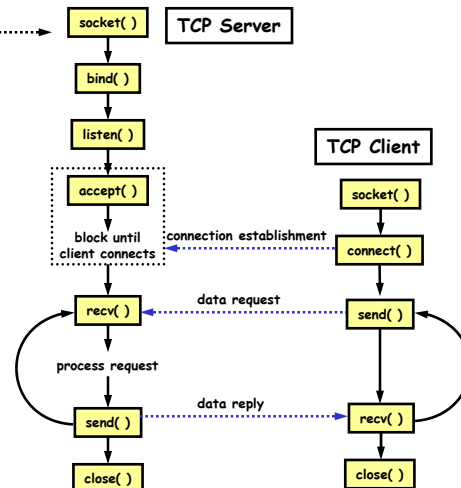
Chapter 10, Slide 5  
Copyright © David Mount and Amitabh Varshney

## Sockets Program Structure

### Program Structure: ..... → **socket( )** **TCP Server**

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**.



Chapter 10, Slide 6  
Copyright © David Mount and Amitabh Varshney

## Overview

---

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

Chapter 10, Slide 7  
Copyright © David Mount and Amitabh Varshney

## 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).

Chapter 10, Slide 8  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 9  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 10  
Copyright © David Mount and Amitabh Varshney

## RakNet: Client Connection

### Connection as a Client: (cont.)

- Connect to the RakNet server.

```
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  $2^{16}-1$ ). 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.

Chapter 10, Slide 11  
Copyright © David Mount and Amitabh Varshney

## RakNet: Server Connection

### Connection as a Server:

- Connect to the RakNet server.

```
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.

Chapter 10, Slide 12  
Copyright © David Mount and Amitabh Varshney

## 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

Chapter 10, Slide 13  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 14  
Copyright © David Mount and Amitabh Varshney

## RakNet: Shutting Down

---

### Shutting Down:

```
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.)

Chapter 10, Slide 15  
Copyright © David Mount and Amitabh Varshney

## Overview

---

- Socket Programming
- RakNet
- Cheating in Multiplayer Games

Chapter 10, Slide 16  
Copyright © David Mount and Amitabh Varshney

## 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.

### Ruining 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 an online issue:

- Single player cheaters only affect **themselves**, so who cares?

Chapter 10, Slide 17  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 18  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 19  
Copyright © David Mount and Amitabh Varshney

## 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.)

Chapter 10, Slide 20  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 21  
Copyright © David Mount and Amitabh Varshney

## 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:

- Alter **executable**. Change game **data** in other files. Hack **packets**.

### 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.

Chapter 10, Slide 22  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 23  
Copyright © David Mount and Amitabh Varshney

## Environment "Tweaks"



Return to Castle Wolfenstein

Chapter 10, Slide 24  
Copyright © David Mount and Amitabh Varshney

## 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).

Chapter 10, Slide 25  
Copyright © David Mount and Amitabh Varshney

## 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.

Chapter 10, Slide 26  
Copyright © David Mount and Amitabh Varshney

## 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 **clip-board** 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.)

Chapter 10, Slide 27  
Copyright © David Mount and Amitabh Varshney

## 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**.

Chapter 10, Slide 28  
Copyright © David Mount and Amitabh Varshney

## Summary

---

### Summary:

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