Multiplayer Game Programming

Chapter 10/11

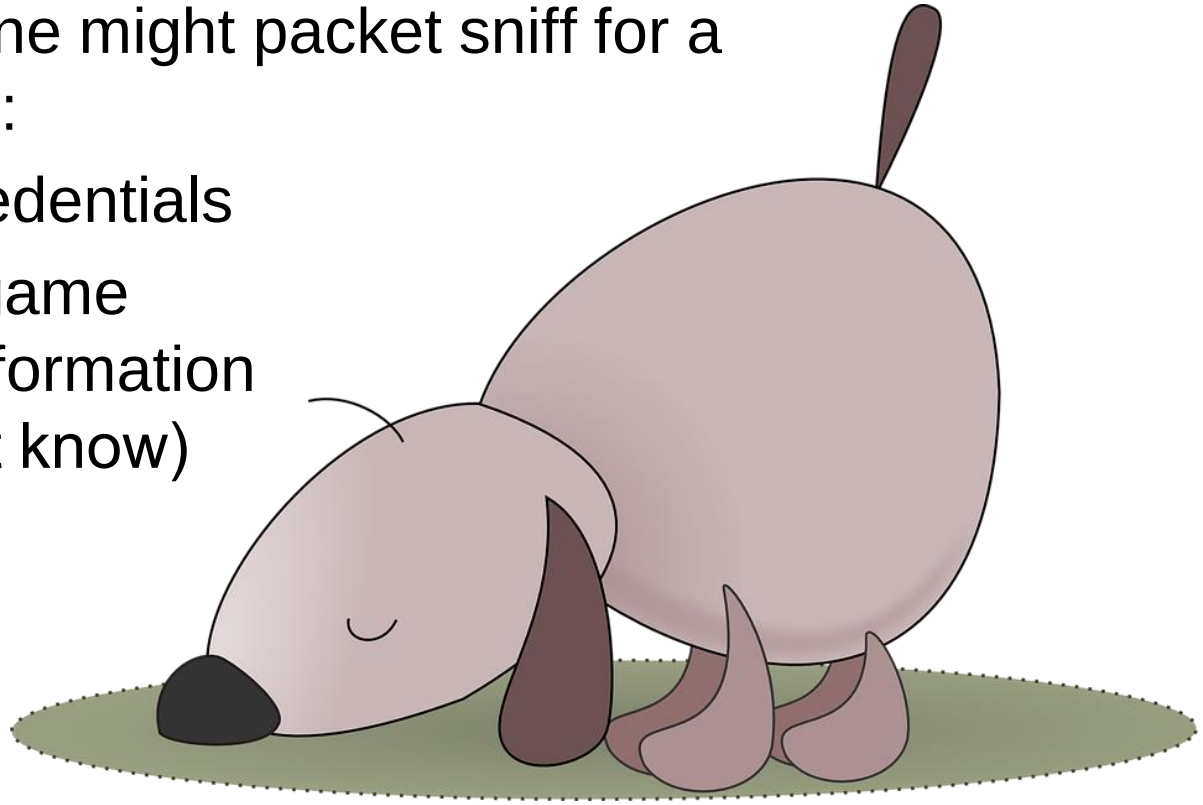
Security, Spawning & Syncvars

Chapter 10/11
Objectives

- **Packet sniffing**
  - How can packets be intercepted, and what can be done to counteract this?
- **Input validation**
  - How do you insure that inputs sent by players are valid?
- **Software cheat detection**
  - How can you determine whether cheat programs are currently loaded on a host machine?
- **Securing the server**
  - How should the server be protected against attackers?
- **Unity Game Networking**
  - NetworkManager Spawning
  - SyncVars
    - Function Hooks

Packet Sniffing

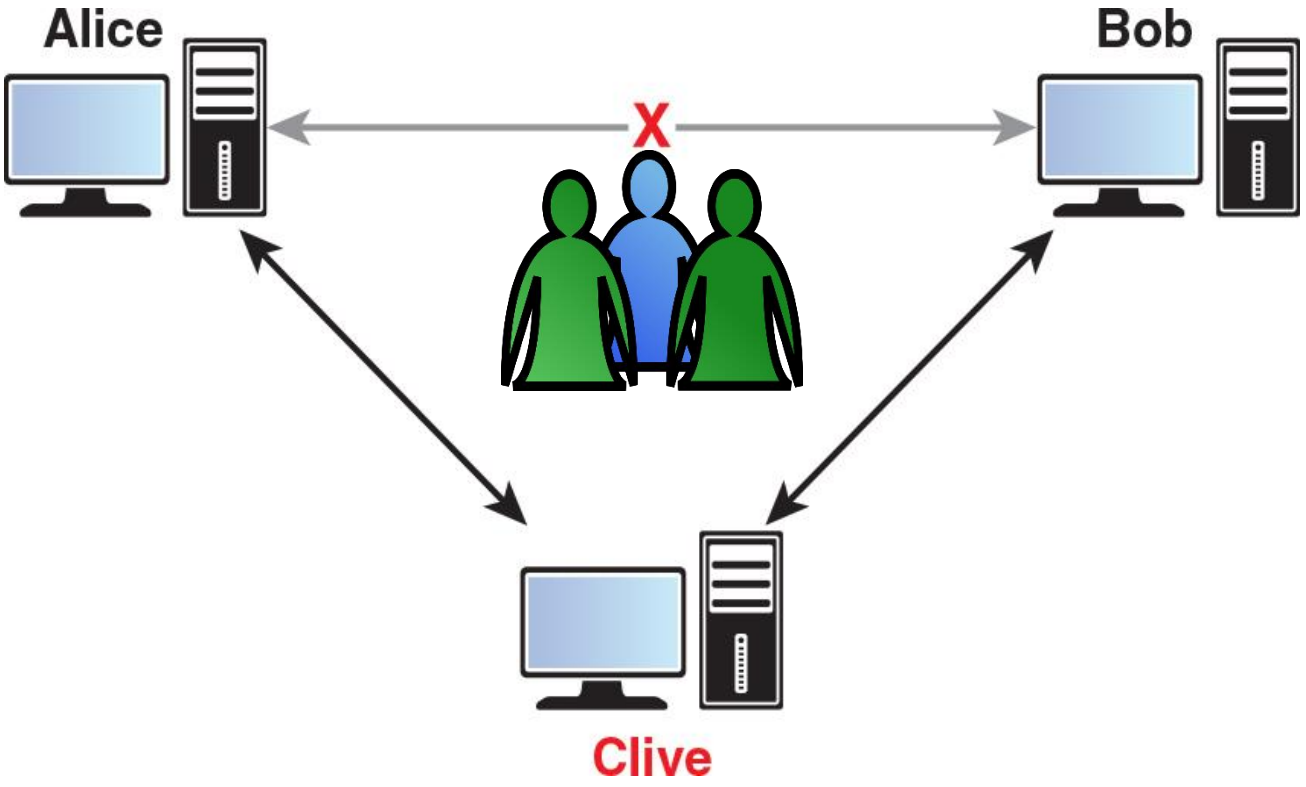
- When packet data is read for a purpose other than normal network operation
- Reasons someone might packet sniff for a networked game:
  - Steal login credentials
  - Cheat in the game (by gaining information they shouldn't know)



Man-in-the-Middle Attack

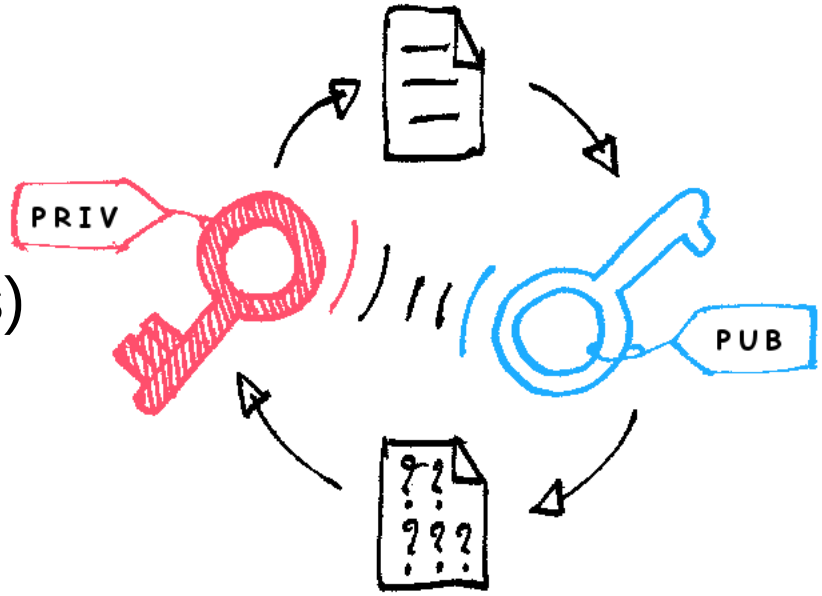
- A machine between a source and destination machine sniffs packets.
- Reasons this could happen:
  - On an unsecured wireless network such as WiFi at a coffee shop
  - A nosy system administrator
  - Government agents are targeting your game for some reason
- Technically, a player can set up a man-in-the-middle for their own machine, but this is overkill because players can just sniff packets locally.

Man-in-the-Middle Attack



Fighting Man-in-the-Middle

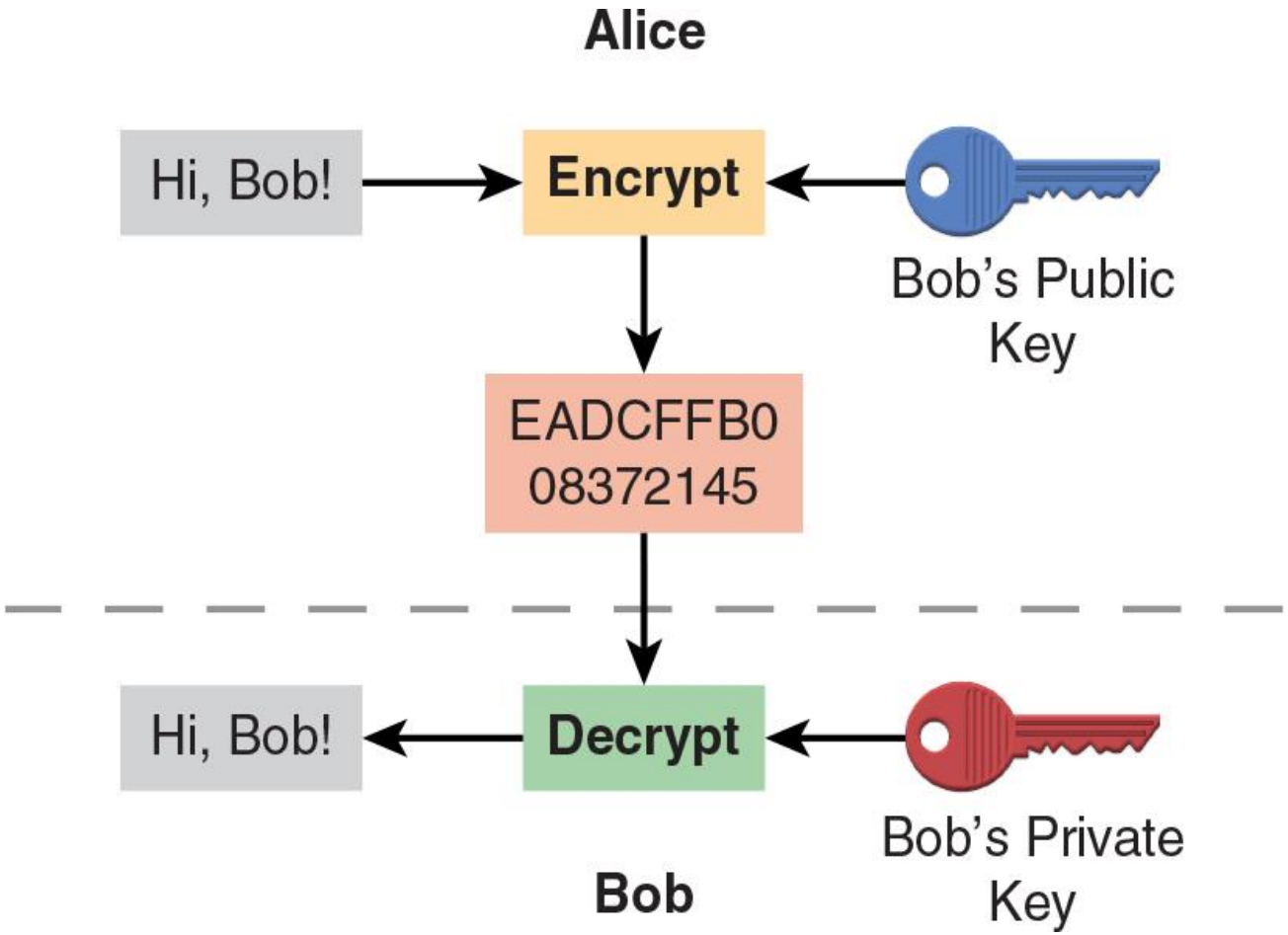
- The general approach is to encrypt transmitted data.
- May be overkill to encrypt all game data.
- Important to encrypt:
  - Login data
  - Credit card information
  - User information (such as billing address)



Public Key Cryptography

- Alice has a public key that everyone knows, and a private key that only she knows.
- All data sent to Alice is encrypted via the public key.
- Only Alice can decrypt the data with her private key.
- Usually relies on prime number factorization.
- Most popular public key cryptography system is **RSA**.

Public Key Cryptography



Packet Sniffing on a Host Machine

- Assume that any data sent in your game's packets can be seen.
- This might be for information cheats to provide the player with extra information.
- Example: Finding out location of stealthed players.
- Send only the data that is critical to avoid this issue.
- Changing encryption/layout of packets regularly can make it harder (though not impossible) for players to sniff packets.

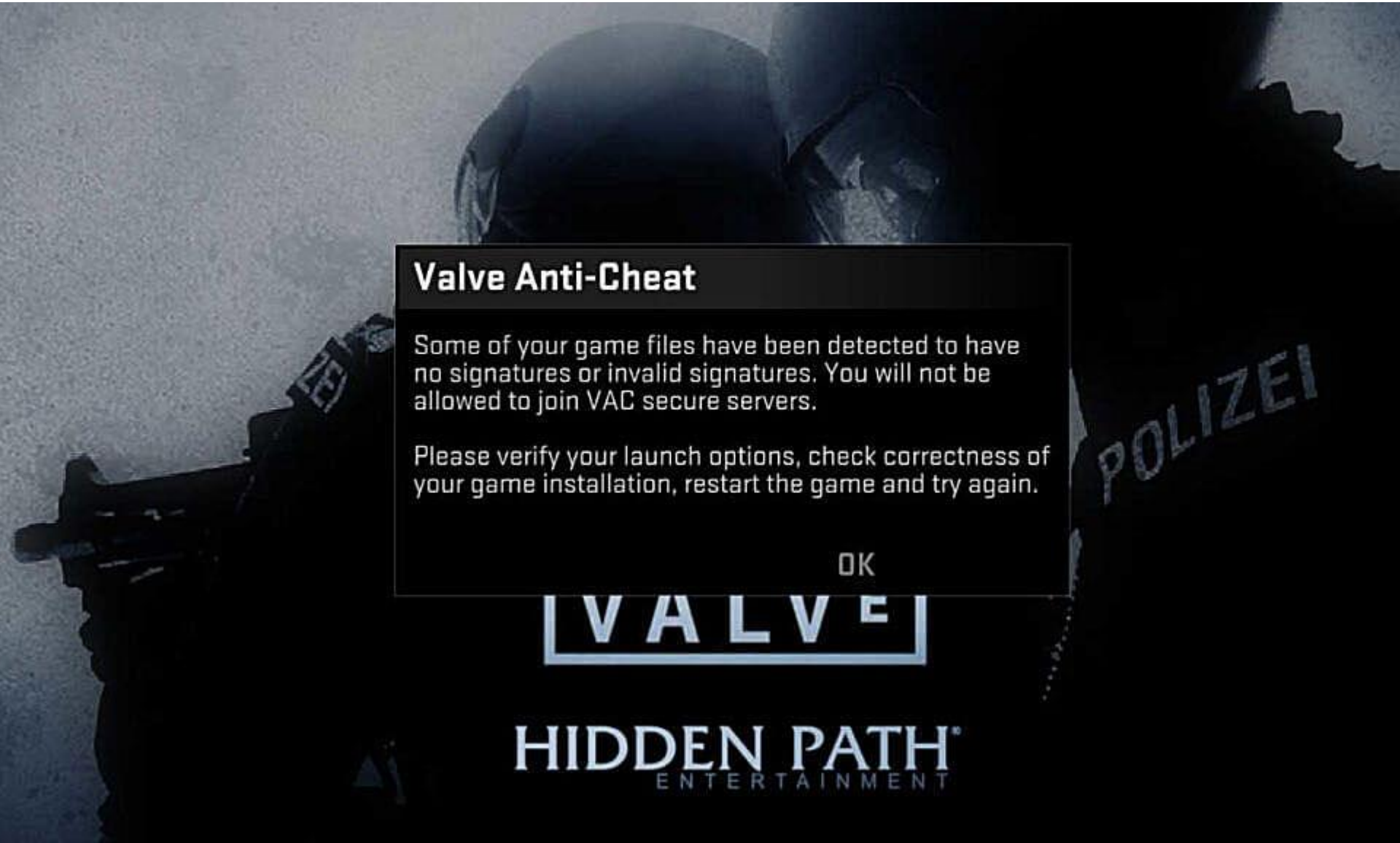
Input Validation

- Important to validate inputs received from clients or peer is valid.
- Make sure commands for a particular player are sent by that player.
- Easy for server to validate clients, because the server is authority.
- Hard for the clients to validate the actions of the server (in the case of listen server); solution is to run dedicate servers.
- Hard to validate in peer to peer.

Software Cheat Detection

- Software that runs as either part of or external to the game process and monitors the integrity of the game.
- Things that can be detected by software cheat detection:
  - Map hacking: Used to gain full visibility on the map
  - Bot: Program that either plays the game for the player, or assists in some way
  - Other client-side hacks/cheat programs
- Examples include Valve Anti-Cheat and Blizzard's Warden.

Software Cheat Detection



Securing the Server

- Equally important to protect the server from attackers
- Particularly important for shared-world games, such as MMORPGs, but any game could be a target
- Should always have some contingency plans for a server attack

Distributed Denial of Service

- The goal of a distributed denial-of-service (DDoS) attack is to overwhelm the server with requests it cannot successfully fulfill.
- Causes server to be unreachable or unusable for legitimate users.
- Protecting against a DDoS requires contingencies in several areas:
  - Hardware
  - Internet service provider
  - Cloud hosting (if relevant)

Bad Data

- Your server should be able to handle bad data requests.
- One way to test for this is **fuzz testing**, which is automated testing designed to discover coding errors that normal testing may not.
- Fuzz testing involves sending lots of data, both structured and unstructured.

Timing Attacks

- The time a query takes can be used to try to glean information about hashing or cryptography system utilized.
- Prevention: Write comparison code that always takes the same amount of time regardless of how incorrect it is.
- If a password has a wrong first character, it should take as long to get rejected as a password with a wrong last character.

Intrusions

- Should also be concerned about intrusions to the server system
- Considerations
  - Keep software up-to-date.
  - Limit the services running on the server to the bare minimum.
  - Encrypt data using cryptographically secure algorithms.
  - Extensive logging and auditing are critical for identifying intrusions.

Spawning

- If we want to have the server control our objects in the game then the clients cannot instantiate them.
- Instead, the ideal way to handle this is have the server instantiate the object and propagate it out to all clients.
- In Unity we can accomplish this by using the **NetworkManager.Spawn()** function.

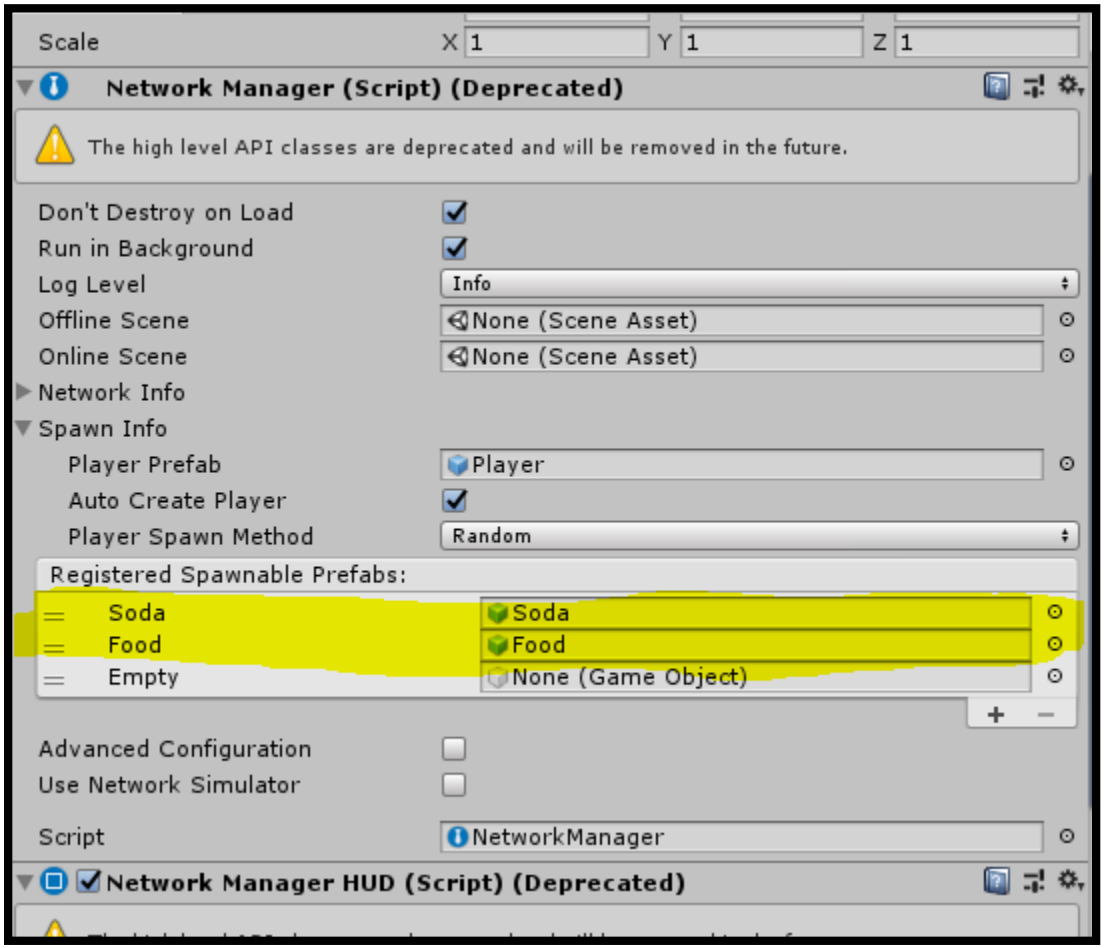
```
//Instantiate tileChoice at the position returned by RandomPosition with no change in rotat
randomPosition.x -= 4.0f;
GameObject go = Instantiate(tileChoice, randomPosition, Quaternion.identity);

NetworkServer.Spawn(go);
```

  - This function takes one parameter, the **GameObject** to spawn on the clients.
  - Because of this the server must first instantiate the object locally, then call Spawn to send the object to the clients.

Spawning

- Spawning an object has two prerequisites:
  - The object must be a prefab that has a **networkIdentity** Component attached to it.
  - The Spawned gameobject **must be in the list of spawnable objects** inside of the NetworkManager script inside your scene.



SyncVars

- SyncVars allow us to synchronize variables across the network.
  - Only works from server to client.
    - Must use Commands to update the server of local variable changes
  - You can make a variable a syncvar by adding the [SyncVar] tag directly above any variable within a NetworkBehaviour Object.

```
[SyncVar]
private int food;
```

  - This will now automatically update all clients when the value is changed on the server.

NOTES:

There is a limit of 32 syncvars per object.

Note you can only sync simple variable types and Unity based types, not custom objects.

SyncVar Hooks

- SyncVars also have a secondary way of updating the client.
  - Hooks allow clients to have some control over whether an object is updated or not and how.
  - Essentially it is a form of network encapsulation.
- When can set a hook function up to a Syncvar so whenever the server updates the variable it will now call this new hook function with the updated value as a parameter:

```
[SyncVar (hook="updateFood")]
private int food;
```

- Parameters to the tag can be added in parentheses and the “string” value should be a local function within the file to call when updating the variable.

SyncVar Hooks

- If you do not update the syncvar while using a hook on the client, then the variable will not sync.
- The new hook function should always have a single parameter of the same type as the syncvar.
  - This will be populated with the changed data the server has observed.
  - It is the responsibility of this new function to update the syncvar within it.
    - Once a hook function is used the syncvar is no longer updated automatically.

```
public void updateFood(int newfood)
{
    food = newfood;
```

Summary

- Server Security is always evolving as attackers find new ways to analyze, intercept and hack data / servers.
  - Staying on top of industry trends for security is a requirement of Network Programmers.
- Servers can and should Spawn objects which are universal to the online world
- Syncvars can be used to update clients of changes to variables.
  - Hooks can be added to syncvars to control / customize the updates