Secure Channel Communication

By Chris Lomont, 2009, http://www.lomont.org

Introduction

This Gem is an overview of creating secure networking protocols. There is not enough space to detail all pieces, so instead a checklist of items is presented that covers necessary points.

Online games must prevent cheaters from using tools and hacks to their advantage, often to the detriment of other players' enjoyment. Cheating can be done through software addons or changes, often making the cheater too powerful for other players, or performing denial of service attacks making the game unresponsive for others, or gold, items, services, and accounts. Since any code running on the client can be disassembled, studied, and modified, security decisions must be made assuming the cheater has access to game source code. Any system should be designed with the twin goals of making it difficult to cheat and making it easy to detect cheaters.

The main reason game and network security is often broken is that security designers must protect against every possible avenue of attack, while a cheater only needs one hole for an exploit. This asymmetric warfare makes doing it right very hard and a continual arms race.

The goal of this Gem is to supply a checklist of items to consider when designing and implementing secure networking protocols. The ordering of topics is designed to be top down, which is a good way to think through security design. Many related security features are mentioned such as copy protection and code obfuscation, which, although not exactly networking, do play a role in an overall security of game networking by making it harder to change assets.

Architecture

The most important decision when designing a gaming networking protocol is to decide how the architecture is going to work before any programming is started. This architecture choice has profound effects on later choices; making a significant change to the networking component will have costly ripple effects throughout the rest of your game components. Three aspects of game engine design need up front thought: multithreading design, networking architecture, and code security. Each of these cannot be bolted onto an existing engine without severe problems and bugs, resulting in poor quality for all components. So fix these three design decisions up front, document them as gospel, and build the rest of the game around these choices.

Code Security

Code security is based on using secure coding practices. Without having this lowest layer done well it is impossible to get networking secure. Many games are written C/C++. Three good references are [Seacord05], [Howard03], and [Graff03].

Peer to Peer

The main choice in architecture is whether to be peer to peer or client/server. In a peer to peer architecture it is up to peers to detect cheaters, and of course such mechanisms can be subverted on a cheating client. For example, if there is a "feature" that allows a client to kick a suspected cheater off a network, then cheaters will subvert this to kick off legitimate players. In this case a secure and anonymous voting protocol should be used so numerous clients need to agree on a cheater before a ban occurs.

Client/Server

Most online games are a client/server architecture, where a central server is the authority on game state, and clients send player input to and receive game state back from the server. The main benefit of this architecture from a security point of view is the server can be assumed to be a trusted, central authority on the game state. Unless your server is compromised or there are fake servers for players to log onto, the server can be trusted to detect cheaters and ban them and their accounts.

The Unreal Engine [Sweeny99] uses what Tim Sweeny calls generalized client-server, where the server contains the definitive game state, and clients work on approximate and limited knowledge of the world. This information is synced at appropriate intervals. The limited game state supports the security principle of Least Privilege, covered later.

Protocol

The next big networking decision is selecting protocols to use and how to use them. A common tradeoff is between using slower TCP/IP for guaranteed delivery or faster UDP for speed. Ensure selected security methods work with the protocol chosen. For example, if your packets are encrypted in a manner requiring all packets get delivered, then UDP will cause you headaches. Good encryption needs an appropriate block-chaining method, covered later, but will cause problems if some packets in a chain are not delivered.

A recommendation is to use TCP/IP for login and authentication to guarantee communication, and then use UDP if needed for speed or bandwidth with symmetric key encryption during gameplay.

Network Security

Basic Security Principles

Follow basic security principles. Find the person on your team who is best versed in these areas, preferably someone who has experience cracking/hacking games, and have them teach the team how various techniques are insecure and how to create secure code. Have the team read a few books on code security, hacking, and network security.

Design security into every interface – your web portal, your UI, your message handlers. For example, don't assume your packets coming in to your server are from valid clients. Allowing buffer overflows and other malware into your server will definitely compromise the system. Assume all inputs to the system (whether client or server side) are malicious and validate them all. Have periodic security reviews with competent staff members to review input code. Definitely walk through interface code before shipping.

Use best security practices. Design what data is given to which game sessions in a least privileges manner, that is, do not give data to a session unless it is absolutely needed by that device to do some computation. For example, in an online first person shooter you don't need to send actors to a given player that cannot affect that actor. Prevent hacked clients from having more information than needed.

Expect common attacks on networking such as

- Packet sniffing. Here a cheater on a network can see all packets, allowing reading and sometimes injecting other packets into the flow. Unencrpyted data such as login and passwords can be seen if in plaintext.
- Man-in-the middle (MITM) attacks. Here another cheater sits between the client and the server, pretending to be the server to the client and vice versa. The MITM may be an employee at an ISP, a university network, etc., and can steal user information from poorly designed protocols. This can be defeated using proper protocols, described below.
- Denial of service. Here a network is flooded with packets, making it unresponsive. The game should detect such activity and not penalize innocent players.

Finally, be sure to observe these basic security principles:

1. **Confidentiality**. Make sure system assets cannot be read by those who absolutely do not need to.
2. **Integrity**. Assets should not be modifiable or deleted by those who do not need to.
3. **Availability**. Assets must be accessible to those who need them in a timely manner. Failure of this leads to Denial of Service.

Encryption

Encryption of some or all data is needed for a good network protocol, so here are a few key points and common errors.

The first rule of encryption is: **never invent your own encryption methods**. Always use ones created and implemented by professionals. Programmers like to invent and implement algorithms, but security has a history of bad decisions made by developers, and it is even hard for security professionals to get right. It is very hard to create good implementations of popular algorithms. Knowledgeable crackers will break your implementations using a variety of tricks – poor random number generation, cache and timing attacks, and simple buffer overflows.

At this point make sure to use cryptographically secure random number generation where needed, such as salts for passwords, user key generation, etc. These are much slower than standard pseudo-random number generators (PRNGs) but don't suffer from allowing crackers to shrink your keyspaces for attacks. For non-secure PRNGs, basically a bit of the seed is given to a cracker per bit of decision he can see from the PRNG. Once he knows the seed (or part of it) he knows the future PRNG values (or some smaller subset).

Next you have to choose appropriate encryption methods for things you want encrypted. The two main classes are public key and private key encryption. Public key is slower, but useful for setting up connections to players and then agreeing on a (secure) random private key.

Private key AES is currently pretty secure, although recent attacks have some worried.

Most common private key methods are block ciphers, that is, encrypt a block at a time. How these blocks are linked together is called a mode of operation. Encrypting one block at a time (called Electronic CodeBook mode, or ECB for short) independently is inherently weak, and should be avoided unless absolutely necessary. [WikiCipher09] has a good example of how this leaks information. Almost any other chaining method on this page is acceptable, and Cipher Block Chaining (CPC) is recommended for simplicity and security.

Even with encryption, to prevent attacks by man in the middle a Message Authentication Code (MAC) must be used to ensure data integrity. This is left to the reader to pursue further.

Finally, a good method for exchanging data over insecure channels to agree on encryption keys is the Diffie-Hellman key exchange, which allows two parties without prior communication to jointly establish a shared secret key over an insecure communications channel. The rough idea is as follows, using a lock box metaphor. Alice and Bob want to share a message by sending a box through the mail. The box, if unlocked, can be opened and the message copied. So, Alice puts the message in the box and locks it with a lock for which only she has the key. Bob receives the box, and puts another lock on it for which only he has the key, and returns the box to Alice. Alice removes her lock, and sends once again to Bob. Bob removes his lock and reads the message. At no point was the box mailed in an unlocked state.

In practice the messages are numbers and locks are based on modular arithmetic and the hardness of factoring.

Finally, the downside of encryption is it slows down performance, so methods must be selected carefully.

Excellent references are [Ferguson03] and [Schneier96].

Hashes

Many secure operations uses hashes to reduce some large set of data into a smaller hash value, often 128, 256, or 512 bits. The purpose is to guarantee that the original data has not been tampered with; otherwise the hash would be an incorrect value. This can be used to check game assets, network packets, identities, and more.

Be sure to select a hash that is considered secure by the security community. For example many still use MD5 as a hash algorithm in secure settings, even though it has been broken for that use for many years. SHA-1 is also falling out of favor with some attacks coming close to breaking it. SHA-2 is acceptable. Currently WHIRLPOOL, based on AES, is considered secure and NIST has a competition for a SHA replacement, and the competitors are currently unbroken (Skein seems to be a good one, designed by some very good researchers). Note MD6 is broken. So make hash function choices modular, versioned, and changeable since they might be changed over the life of your game.

Complete secure protocols

A final choice for a lot of secure protocols is to use an industry standard such as SSH-2 (not SSH-1, vulnerable to man in the middle) or SSL to perform encrypted networking.

Login

After connecting to a server or other players, the player usually has to login. Never send the username or password across the network unencrypted; this prevents man-in-the-middle attacks. Having to guess a username and password is more work than just guessing a password. Never send the password and definitely never store the password unhashed directly on the server. Generate a secure random salt, prefix the password, and then send/store a hash and the salt. The salt prevents pre-computed dictionary attacks such as rainbow tables. If you never heard of password salting or rainbow tables and are implementing a login system, then you are exactly someone who needs to read up on them.

Require the username to be visible characters, and enforce this to prevent players from using names that cannot be seen on some screens, or names that allow security breaches. Otherwise a player can choose a name consisting of non-printable or non-ASCII symbols, preventing for example, discussion about the player online or voting for a ban. You may go so far as preventing a player from typing their password into any user interface portion except the login box,

preventing phishing attacks where players are tricked into entering their login/password in game chat windows.

Once a login and password has been transmitted to the server, in future logins a challenge hash [Watte08] can be used, thereby avoiding sending passwords ever again over the network.

Make sure players choose secure passwords, for example by forcing them to be 10 or more characters, use uppercase, lowercase, numeric, and/or non alphanumeric characters. A brief internet search gives good guidelines.

Use a per session secure key to prevent packet playback attacks.

Authentication

Authentication is the process of making sure a player is who they claim to be and not an imposter. The Game Programming Gems 7 article [Watte08] has good coverage of this topic. A few ideas are using IP address or a token stored on the player computer to authenticate, and perform some authentication questioning if this changes.

Gaming

The network needs the highest performance during playing the game, and so encryption and security methods are usually kept to a minimum. The right balance must be determined on a case by case basis.

Attacks

The level of attack sophistication against your game is directly proportional to popularity and longevity. Hence more security is needed for a triple-A title than for a casual game. For example World of Warcraft (WoW) uses the Warden, a sophisticated kernel mode anti-cheat tool described in [Hoglund07], [Messner], and [WikiWarden09].

Reverse Engineering

Using tools such as IdaPRO and OllyDbg and a little skill one can disassemble game executables into assembly code for any platform, and there are plugins that reverse the code into C/C++ with good results. It only takes one cracker skilled in reverse engineering to remove or break weak security measures, who then distributes the tool/crack to everyone. Assume crackers have access to your code and algorithms.

Kernel Mode

Kernel mode is the security layer that operating system code runs in (for the Windows PC and many other protected OSes), and most games run in user mode. WoW's Warden and copy protection schemes like SecureROM run as kernel mode processes, giving them access to all processes and memory. However even kernel mode software can be subverted by other kernel mode software. Using kernel mode tricks like those used in rootkits and other malware, a

sophisticated cracker can run tools that hide under any snooping you might do and can watch/record/modify runtime structures. This is done to circumvent Warden and many of the CD-ROM and DVD protection schemes. To detect and prevent kernel mode attacks on your code you need kernel mode services, likely your own driver or commercial product to do the work for you.

Lagging

Also known as tapping, this is when a player attaches a physical device called a lag switch to an Ethernet cable, slowing down communication to the server, slowing down the game for all involved. However the player with the lag switch can still run around and act, sending updates to the server. From the opponents view the player with the lag switch may jump around, teleport, have super speed, and generally be able to kill opponents with ease.

In peer-to-peer network architecture this can also be achieved with packet flooding the opponents since each client sees other IP addresses.

Other Attacks

- **Internal Misuse.** A large release game must protect against employee cheating. For example an online poker site was implicated in using inside information for some individual to win an online tournament using full knowledge of other player hands [Levitt07]. When these prizes reach tens or hundreds of thousands of dollars, there is a lot of incentive for employee cheating.
- **Client Hacking**. Client hacks are changes made to a local executable, such as making wall textures transparent to see things a player should not (called wallhacking).
- **Packet Sniffing**. To reverse network protocol, looking for weaknesses such as playback attacks, DDoS, usernames passwords, chat packet stealing. Some networks allow some users to see others packets, such as colleges and others on the same LAN, making packet sniffing attractive.
- **Bots**. Numerous bots help using auto aiming, auto firing, collect goods such as gold and other items. Several years ago this author created a (unnamed) word game playing bot that was very successful, climbing to the top of the ranks during the course of the experiment.
- **Aids**. Aids assist a player, such as auto aiming, auto firing when aimed well, backdoor communication, poker stats, poker playing games, etc. Imagine how easy it is to cheat online for chess, checkers, Scrabble, and similar games where computers are better than humans.
  Other tools give multiple camera angles, better camera angles, player highlighting, etc.
- **Design Flaws**. Game design can be exploited. For example, on a game with scoring, if a player about to record a bad score can quit before the score gets recorded and not be penalized then this is a design flaw.

Responses

In response to all the attack methods, here are some methods to help defeat cheaters.

Code Integrity

First of all write secure code. Then key code assets can be further hardened using methods from the malware community such as code packing, encryption, and polymorphism. These all slow the code down but could still be used for infrequent actions like logging on or periodic cheat detection. Further tools and methods along these lines should be researched on the internet, starting at places like www.openrce.com and www.rootkit.com.

One method to check code integrity is to integrate a small integrity scripting system, and have the server (or perhaps other clients in a peer to peer setting) send snippets of script code to execute. These snippets perform integrity checks like hashing game assets, checking game process memory for problems, etc., returning the answer to the server for verification. The queries are generated randomly from a large space of possible code snippets to send. To defeat this technique a cheater has to answer each query correctly. This requires keeping correct answers on hand, or keeping a copy of modified game assets and remapping the scripting system, or something similar. Although doable, this adds another level of complexity for a cracker to work with since the script language is not built into existing reversing tools. A variant of this takes it further and randomizes the language per run and updates the client side as needed.

A final method of code protection is integrated into commercial copy protection schemes like SecureROM, Steam, and PunkBuster.

An interesting attack on PunkBuster (and likely would work on other anti-cheat tools) was the introduction of false positives getting players banned from games. The false positives were caused by malicious users transmitting text framgments from known cheat programs into popular IRC channels, and PunkBuster's aggressive memory scanning would see the fragments and [Punk08] ban players. This is likely fixed by the time this reaches print.

Again, any code will eventually be reverse engineered given a determined cracker. A good place to start reading is [Eliam05]. [Guilfanov09] shows some advanced code obfuscation techniques from the creator of IdaPRO.

Kernel Mode Help

As mentioned above, this gives the highest level of computer control, but also can crash a computer. Operating system may ban access to kernel mode code for gaming in the future, making kernel mode code a short term solution. Kernel code mistakes often crash computer, not just the game process, so code must be extremely well tested before shipping.

Cheat detection

Having the server detect cheaters through statistics is a powerful technique. Statistics from players should be kept and logged by username, including time online, game stats like kills, deaths, scores, gold, character growth, kill rate, speed, etc. An automated system or moderator should investigate any players with stats too many standard deviations outside the norm. A rating system could be implemented behind the scenes like ELO scores in chess, and players who suddenly show profound skills can be detected and watched.

Continued Vigilance

Every current solution requires vigilance from the game creators to patch games, update cheat lists, and evolve the game as cheats evolve. So far there is no one-shot method for preventing cheating in online games. However, following all the advice and reading deeper into each topic will make protecting the game much easier by making cheats much harder to implement. Game creators should monitor common cheat sites like http://www.gamexploits.com/ and per-game forums looking for cheats and techniques.

Backups/Restore

To prevent worst case damage to a game environment, have a well schedule and system for backups in case of server hacking.

Disciplinary Measures

When a cheater is caught, the game has to have a well defined punishment system. Most games and anti-cheat systems currently ban accounts either temporarily, permanently, or with some resolution process.

Examples

Here are two examples of current online game security features.

WoW

World of Warcraft uses a module called the Warden to ensure client integrity. From [Hoglund07], [Messner], and [WikiWarden09] the following is found

- It checks the system one about every 15 seconds.
- It dumps all DLL's to see what is running.
- It reads the text of all Windows title bars.
- These are hashed and compared to banned item hashes.
- It hashes 10-20 bytes for each running process and compares these to known cheat program hashes, such as WoWGlider.
- Looks for API hooks

- Looks for exploitative model edits.
- Known cheating drivers and rootkits.

Unreal Tournament

Sweeny [Sweeny99] lists the following network cheats as having been seen in Unreal Tournament:

- Speedhack – exploits the client's clock is used for movement updates. Fixed by verifying client and server clock stay nearly synced.
- Aimbots - UnrealScript and external versions.
- Wall hacks and radars - UnrealScript and external versions

Conclusion

To develop a secure networking protocol for gaming, securing all game assets from code to art and networking data is important. Performance versus security tradeoffs must be designed into encryption and message protocols from the beginning.

Securing an online game is a constantly evolving war, and whatever methods used today may fail tomorrow. Developers must constantly monitor the servers and communities to detect, mitigate, and prevent cheating. This involves tools to update clients, protocols, servers, and assets as needed to provide an enjoyable, level playfield for all customers.

Finally, throughout the game development process, keep a list of security checkpoints, and follow them religiously.

References

[Eliam05] Eliam , Eldad, Reversing: Secrets of Reverse Engineering, Wiley, 2005.

[Ferguson03] Ferguson, Neils, and Schneier, Bruce. Practical Cryptography, Wiley, 2003.

[Graff03] Graff, Mark, and Van Wyk, Kenneth, Secure Coding: Principles and Practices, O'Reilly Media, 2003.

[Guilfanov09] Guilfanov, Ilfak, "IDA and obfuscated code," http://www.hex-rays.com/idapro/ppt/caro_obfuscation.ppt, 2009.

[Hoglund07] Hoglund, Greg, "4.5 million copies of EULA-compliant spyware," http://www.rootkit.com/blog.php?newsid=358, 2009.

[Howard03] Howard, Michael, and LeBlanc, David, Writing Secure Code, 2nd Edition, Microsoft Press, 2003.

[Levitt07] Steven D. Levitt, "The Absolute Poker Cheating Scandal Blown Wide Open," http://freakonomics.blogs.nytimes.com/2007/10/17/the-absolute-poker-cheating-scandal-blown-wide-open/, 2007.

[Messner] Messer, James, "Under the Surface of Azeroth: A Network Baseline and Security Analysis of Blizzard's World of Warcraft," http://www.networkuptime.com/wow/, 2009.

[Punk08] "netCoders vs. PunkBuster," http://bashandslash.com/index.php?Itemid=78&id=297&option=com_content&task=view, 2009.

[Schneier96] Schneier, Bruce, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition, Wiley, 1996.

[Seacord05] Seacord, Robert, Secure Coding in C and C++, Addison-Wesley, 2005.

[Sweeny99] Sweeny, Tim, http://udn.epicgames.com/Three/NetworkingOverview.html , http://unreal.epicgames.com/Network.htm, 2009.

[Watte08] Watte, Jon. "Authentication for Online Games," Games Programming Gems 7, Charles River Media, 2008.

[WikiCipher09] "Block Cipher Modes of Operation, "http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation , 2009.

[WikiWarden09] "Warden (software)," http://en.wikipedia.org/wiki/Warden_(software), 2009.