project-2015-05-18.html

Programming project

Just reading about how to develop secure software isn't enough; you need to do it. Thus, a significant part of the course is to do a programming project.

For most students, I expect the programming project to be a game as described below. A few of you who have more experience developing secure software may want to develop something "more important" than a game, in which case, you may choose the alternate programming project. Each alternative is described below.

Normal project management can really help in projects, by the way. I strongly recommend every project use a version control system of some kind (e.g., git) for the source code, configuration, and documentation. That will really help you get work done, and as a side-effect, it can help track "who did what". Using an issue tracker (or at least email) to record reviews/comments can help record review credit... and also makes it more likely that problems someone finds won't get forgotten. And of course, create the software incrementally from a working starting point; DO NOT try to "integrate it at the end".

Game

Overall

Create a game system (e.g., to implement a board or card game) that can be *securely* played across a network between humans. This project is to be done by teams; each team must implement a different game. I strongly recommend pairs, so that you can review each others' work. Teams may have 1 or 3 only by instructors' permission.

Do not implement a game AI. This class is about designing and implementing secure software, not about writing a great AI, and the effort to make a great AI will detract you from the goal. Most people create a GUI, but even that is not strictly required.

You may use the programming languages Java, C#, Python, PHP, Perl, Ruby, Bourne shell/bash, C, C++, Objective-C, Javascript, Scheme, Common Lisp, Prolog, Smalltalk, or Ada. You may use another language with permission of the instructor (I need to be able to read it). I do not recommend using C, C++, or Objective-C, because it is harder to do correctly.

The purpose is not to create the ultimate game, but to get experience in writing secure software (while having some fun doing it). Writing a game that is cool, but really insecure, misses the whole point... and will not get a good grade.

Pro tip: Implement the SSL/TLS connection as soon as you can. Once you have that working, you can much more easily build on top of it. Then implement system logging; that makes it easier to debug things, and you'll end up with a robust logging system (you'll also reduce the risk of displaying information you should not).

Concept

You are part of a small development shop that has been asked to develop a network-enabled game system. The games will be played between competitors that do not trust each other and may try to cheat. The software must support at least 50 simultaneous players (in most cases this implies there will be a large number of simultaneous games).

The program must be designed to be secure. You must avoid vulnerabilities like SQL injection, buffer overflows, accepting invalid input, and passing sensitive data unprotected. You will also need to authenticate all users. You do not need to prevent out-of-game collusion.

Choice of game

The choice of which game to implement is up to you! A few words of warning though:

1. Games must support at least two humans, so you may not implement single-player games like Klondike Solitaire. The number of humans per game will depend on the game (chess is 2-player, bridge is 4, poker varies).
2. Game moves must be more complex than a single digit or letter as a possible move. So games like Tic-Tac-Toe, Connect 4, and so on are not acceptable. Basic blackjack is not acceptable (the only moves are "hit" and "stay", which can be encoded as "H" and "S"). Chutes and Ladders is especially unacceptable (there is only move possible move, "next", which can be encoded as "N"). I want you to demonstrate the ability to parse and test input more complex than that.
3. Some suggested games include Chess, Checkers, Go, Bridge, Hearts, and standard 5-card draw Poker.
4. Every team must have a different game, and I need to approve your choice (to avoid problems). You can even make up your own game, but if you do, try to make it fun.

Only one particular game is allowed per class (e.g. no two teams will be allowed to implement the same game). Thus, teams must register what game they plan to implement, and only implement one game. It's first come, first serve for game selection.

Basic Requirements

When (potential) users connect to the game system server, they must be able to either log in (with their username and password) or create a new account. Users must be able to pick their username and password when they create a new account; you should impose limitations on both. You must not let someone create an account that already exists. You must use an encrypted channel when transmitting passwords. You must store passwords using some salted hash system; you must not store passwords in the clear or as bare hashes. You don't need to support password reset for purposes of this project.

Once a user logs in, they must be able to:

1. see the win/loss statistics of themselves and others
2. see game moves of completed games,
3. start a new game, and
4. join an existing game that needs players.

Once a game has the required number of players, it begins. Once a player has joined a game, they must be able to re-log in and rejoin (e.g., if their computer crashes). However, if a player leaves, there must be a timeout where eventually they will forfeit (else players about to lose would just leave). Once a game ends, the win/loss/draw records must be updated. Obviously, different players will need to log in as different users, or they could easily cheat.

A few points:

1. You must use a client-server architecture (unless given permission otherwise), such as web server application on the server and web browser on the client. Peer-to-peer is actually really hard to secure, and you won't have the time to do that well.
2. Games must be playable over a TCP/IP network.
3. Your program must reject invalid inputs including inputs that are syntactically correct but are semantically invalid based upon the current state of game play. Remember, the players may try to cheat.
4. The game communications and current game state of active games must not be accessible to a third party during play (other than trusted admins). E.G., use SSL/TLS.
5. There must be a UI sufficient to play the game, but a GUI is not required.
6. Any games that rely upon random values (e.g. dice, shuffled card deck, etc) will need a source of randomness that is trusted by both players and is actually random. If you use cards, double-check your deck-shuffling algorithm (it's not hard to do, but it's easy to do wrongly).
7. You don't need to pay for a signed certificate. Using a self-signed cert is fine for class purposes (if this were real, you could always switch later).
8. You may use external dependencies (software libraries, frameworks, operating systems, database systems, etc.). However, it is your job to ensure that there's a relatively low risk of them being security vulnerabilities, and justify that in your final report. (Don't go crazy on this, just show you thought a little about it.) You are responsible for the external dependencies you choose to use. If you include a library that has publicly-known vulnerabilities (perhaps transitively), and you do not counter those vulnerabilities, that is *your* fault. You have to do enough to show you wrote the (main) game yourself, though; just submitting a copy of someone else's game is not okay.
9. Do not rely upon anything that is not freely available without asking permission first (e.g., please make it so I can use MySQL or Postgres if you use an SQL database; don't make it depend on the very pricey Oracle). Exception: You can use Windows if you want to, I *do* have that. If you want to do otherwise, please talk with me in advance so we can work out some way for me to evaluate and grade your assignment. I can't afford to buy every library or framework out there!
10. *Keep it simple*. Simple tends to be more secure, as well as being more likely to work.
11. Don't try to integrate separate components at the end, that rarely works. Instead, build something early that works (e.g., communicates), then keep adding just a little bit of functionality at a time. I don't require it, but I would recommend using SCM tools like subversion or git so that you can constantly integrate the program into a cohesive whole from the team members.

Security requirements:

1. Confidentiality: You must create an audit trail of every game move, and these game moves must not be available to those not in the game until the game has completed. After the game is completed, the final audit record (including every move) must be publicly available to all authenticated users (but to not to unauthenticated ones). The win/loss/draw record of each user must be made available to authenticated users (but not to unauthenticated ones).
2. Integrity: Only the current player can make a move in a given game, and it must be a legal move (syntactically and given the current situation). Make sure that the audit trail of game moves can't be changed (except by legal moves during a game!), and that the win/loss/draw record of each user can only be changed by actually winning, losing, or drawing.
3. Availability: The player must not be able to make any game (including one he's playing) pause forever. A timeout eventually means forfeit, and sending 1 byte should not cause a stall forever. A simple packet or command shouldn't kill or stall a game. You can't prevent everything, so in the case of DDoS attacks, try to relatively quickly recover once the attack ceases or is filtered out upstream. If a player gets logged out for some reason (e.g. wireless failure), they must be able to reconnect to the game and continue to play (as long as the timeout hasn't expired). Your system must support multiple simultaneous games by different users (threads are permitted, but not necessary). It's fine if a given user can only play one game at a time.

You ***must*** use at least one tool to examine your program and fix any vulnerabilities it finds before submitting it. You can choose a dynamic analysis tool (such as OWASP ZAP) or a static analysis tool (such as PMD, FindBugs, or Fortify). It's best to use several tools, preferably of different kinds, since different tools find different things. This tool use should be included in your assurance case, of course. Using and responding to a tool report will not only help you make your program secure, but the process will probably help you understand how to make software more secure in general. Start applying the tool as soon you can; don't wait until the end of the project.

In addition, if you are directly calling an SSL/TLS library (instead of just building on a web browser and web server), you *must* demonstrate via tests that your program (1) rejects expired certificates and (2) rejects untrusted certs (a cert is only trusted if it is directly trusted or if there is a certificate chain rooted in a trusted certificate authority). These are *really* common mistakes for do-it-yourselfers.

Teams

For this project you are allowed and encouraged to work in teams of up to 3 students; 2-person teams are recommended. You may work alone if desired, but there will be no reduction in requirements because of this, so if you choose to work alone you will be solely responsible for everything. If you work in a team (as encouraged), be sure to fairly distribute the work among team members, and discuss the design and code among your team to find problems ahead-of-time.

If your team has more than one person, *please* be sure that you discuss the design and each other's code. Challenge each other on whether or not something is secure, and don't give up until the other person has convinced you it's okay. Be respectful of each person, and brutal on the code. I suggest using phrases like "this method..." instead of "your method..."; many people are naturally defensive when their code is critiqued, and I need everyone to get over that and accept constructive critique. Remember, an attacker is trying to break in — you want to find and fix problems before the attacker finds them. Think about using multiple mechanisms to make it harder on the attacker. Write down key points as you go, so you can include them in your final report explaining why it's secure. Again, be respectful of each other; everyone (even you!) makes mistakes, and peer review is often very effective at finding mistakes. This internal team discussion can be one of the most educational parts of the whole process, and it will almost certainly produce better results.

Grading

Final Report: You will be required to provide a report on your implementation. This report should include the following sections:

1. Introduction (brief description of project; be sure to include the project name, collaborators, class, date
2. Design/Architecture of the code, including major components and a description of major functions/classes
3. Installation Instructions, including documenting any external dependencies
4. Operating Instructions
5. Game rules
6. Why you believe it's secure

The "Why you believe it's secure" section is especially important. Write a convincing argument for why it's secure (this is often called an "assurance case"). See the first lesson for more about assurance cases. You might walk through a list of common vulnerabilities and explain why your program isn't vulnerable to any of them. You might talk about peer review, static analysis tools you used to check it (including their results and what you did), dynamic tools you used to check it, how the design counters attack (including privilege limiting mechanisms), how the input filters counter attack, how the configuration counters attack, and so on. If problems were found and fixed, feel free to talk about them, and show that they provide evidence that your analysis was thorough enough to find and fix problems. If you can't write a convincing argument, think about what else you need to do to make it convincing. Be sure you include some justification that what you've done is enough; do not just include a random list of things

it convincing. Be sure you include some justification that what you've done is enough, do not just include a random list of things you did (since that does not justify that you did enough). I'm not looking for a formal mathematical proof that it's secure; I'm looking for a set of arguments that would convince a fair-minded decision-maker that the risk of vulnerabilities is low enough that it's ready to deploy.

The report should be around 12 pages or so, with about half of that the assurance case. Longer is fine.

Each team is also required to record a video demo of their project. Log in, start a game, play a few moves, show that it records things. The video demo *must not* be longer than 5 minutes.

Final project grading:

- Final Report including demo (25%)
- Functionality (20%)
- Security (35%) - much of this will be derived from looking at the report
- Project participation (20%).

The participation score is worth 20% of the total grade. It exists to deal with those few people who might be tempted to do a minimal amount of work and intentionally overload their partner. If no one says anything about their team partner, I'll presume that everything went well and everyone will get full credit. However, if I receive a confidential report from a team member, then I will ask that team member to describe the role and level of effort for the other team members. I may then make adjustments using that information, related information such as version control logs, and my direct observations concerning each team member's engagement and understanding of the project.

If I believe (based on the information I receive) that someone didn't do any significant useful work on the programming project, then the non-worker gets a 0 for the whole project. My goal is not to be punitive; my goal is to protect everyone from "partners who won't work". I think this is fair, indeed, it's not fair to give someone credit for work not done.

Submission requirements

You are required to provide the source code for your game system and your report, electronically, via Blackboard.

Alternate Programming Project

A few of you may want to develop something "more important" than a game. In that case, here is an alternate programming project, intended for the stronger students. You *must* have permission of instructor before proceeding.

The alternate programming project is to improve the implementation and security some existing open source software (OSS) project, and submit those changes to the project in the form expected by that project (e.g., compatible licenses, expected format for that project, and so on). The project need not accept the changes, as that is beyond your control, but you should carefully consider their comments. You should expect to take at least 50 hours, and you must document what you did (so that I can see that you really did the work).

This task could be to review and change an existing OSS project to harden its resistance to attack (typically by a combination of architecture changes, input validation, and code review to detect vulnerabilities and/or reduce risks). Alternatively, it could be to improve/implement a significant portion of an existing security library, e.g., you could implement a plug-in for OWASP ZAP, create a more rules for LLVM static analysis, or implement portions of the OWASP Enterprise Security API (ESAPI), that isn't already implemented in a given language (Perhaps OWASP ESAPI isn't available at all in that language, or perhaps it's available in that language but not that portion, or perhaps it is really immature. OWASP ESAPI is pretty mature for Java, but not necessarily for other languages.)

The rules are otherwise the same as the main project. In particular, you must describe *why* you believe it is secure.

You must do enough to be at least the same amount of effort as the game project above.

Since this is a more open-ended project, you *must* describe in detail for me what you intend to do (e.g., its scope). I need to approve your choice for it to be acceptable; that's true for games too, but since this is more open-ended, it's especially important to gain my approval for this case. Expect to interact with me throughout the work in this case.

Honor Code

This is an group assignment. **Group work IS allowed and encouraged** but only within the defined and approved teams.

This assignment, as with all assignments in this class must be performed in STRICT COMPLIANCE to the honor code!

This document is version 2015-05-18.