# Software Security

CSC424: Software Engineering II

Rebecca Grantland

April 23, 2021

## Introduction

Software security has been a rising issue since computers were invented [3]. It is almost as if as soon as something exists you have people that want to use it as intended, and others that want to find a way to break or use it maliciously. Since computers are now almost all connected to this thing called the internet where data can be transferred across the world in seconds, there is a lot more at risk than there used to be. People enter credit card information, addresses, email, and plenty of other personal information into online forms now, and of course there are people that would like to access that information to steal money, blackmail, or otherwise use this information for illegal purposes. Now different techniques have been invented for software creators to use to prevent malicious attackers from accessing users' information [3]. This is software security in a nutshell.

## Making Safe Software

There are a million and one different ways to help keep software safe. Many people know this already, but the question is how is it done? If there are so many different ways, then how come the everyday person can hardly list three to five of them? A lot of security happens in the background of programs, so it is hard to see how programs are secure or how secure they are unless you are a hacker that wants in. So let's go over some different techniques that help to make software secure.

Probably the most common thing a programmer will think of is inputs [1]. If you are creating a software you should test any and all inputs and injection methods you can [1]. SQL injection can be something awful if a hacker uses it to get username and password information from typing a couple lines of code in your input bars [1]. Testing for injection is not very difficult to learn if you have the internet, and it is not very difficult to learn methods to thwart this basic attack. There are things like JavaScript regular expressions where you block the user from using certain characters or patterns on input. There are now normally built in methods to change special characters to their HTML equivalents to keep them from working like it normally would in code without changing how it looks on screen. For SQL specifically, there are some methods like real_escape_string() in PHP that break up the user's input string with '\' before a special character to prevent injection.

Another common practice that some people might think of is store basic information [1]. If you can only store unimportant information for your program to work, that is fantastic. If someone manages to hack it, then all they get is some useless random information that you use for your program. This is not always the case though and databases will probably need to store important information for their functions. Like a bank app or webpage needs to store your username, password, and other bank information. In cases like this it is important to remember to use lots of other methods to protect the information, but also only store things you absolutely have to to make the program run and nothing more [1]. Back to the bank app, it needs your username, password, account number, and dollar amount in your account. It does not have to include your home address, full name, or social security number. The bank still needs to have all that information on file, but it does not need to be kept in the same database as the app's information. By separating the information, it makes it harder for those that want your private information, to get it.

Similarly, it is always a good idea to limit access as much as possible [1]. Sure, it is very nice to hand someone all the privileges they might need right off the bat, but who's to say they know what to do with those privileges, or happen to know all too well how to use them maliciously [1]. It can be a hassle to find ways to limit access, but it pays off tremendously when it comes to the software's security by limiting the number of access points a hacker can have [1]. Later, you might run into issues with many users asking for more access privileges, but this is why it is important to think about what privileges users will need to do their work in your software [1]. You could also change this later if need be when you push out a new version of the software [1].

I am sure everyone is now aware of softwares not only trusting a username and password anymore. We all get those pesky emails labeled "Is this you?" when we log in on a new device, or get a pop up about checking your phone for a key to type in after your password. These things get annoying sometimes, but they are there for a reason. Many people forget their passwords and have to reset them, and they will reuse the same password or variant of the same password for multiple accounts [1]. Since there has to be a way for a user to retrieve or reset their password and make sure that the user requesting the reset is the user and not a hacker, methods like these are used in attempt to make sure the one requesting is the user and not a hacker [1]. They are not flawless by any means, but it is an added piece of security that helps programmers vet the one requesting the change is the user and not an attacker [1].

Encryption is another method that can be used to protect information [1]. Can it be cracked? More than likely, the answer is yes. Is it enough to deter most hackers from trying to use the information they got? In most cases, this would also be a yes. Encrypted information cannot be read by a person unless they take the time to decode it [1]. Computers are much better at decoding information and making it readable to humans, but even they can take a while to crack a code [1]. Unless someone really wants to spend time decoding information that may or may not be useful to them, they probably will not bother unless they know they are going to get something good. The more you encrypt, the more garbage they would have to dig through to get to the things they would want as well as figure out how to crack your code.

The last method I would like to make mention of is to have someone and something look over your code [1]. By having someone look over your code I mean it is good to have another pair of human eyes look over your code to see if they can catch your mistakes and make suggestions on how to better go about accomplishing tasks in your code [1]. By having something look over your code I mean that using software tools scan your code for little mistakes everyone tends to make when they code [1]. Most of the time the mistakes made in code that tools catch are not major, but occasionally they could be detrimental to security [1]. Regardless of the mistake, it does not hurt to fix it.

All of these methods can help defend software, but I think the most important thing to remember is to use multiple methods together. There is no such thing as over doing it when it comes to secure software. For example, our project does not include any vital user information, but we still used some methods, such as preventing SQL injection, to protect our database. This was done with the intent to make sure that the information in our database is safe and cannot be added to, edited, or deleted by users that might think it's funny to pull a prank on a zoo webpage.

## Testing Tools

As mentioned before, there are some testing tools that can be used to scan code for simple mistakes like misspellings and some vulnerabilities [1]. Different testing tools will test for different things so misspellings and vulnerabilities are not the only things they can test [2]. So let's look at some different kinds of testing these tools can do. There are three main groups and they are automated unit testing, automated web service and API testing, and GUI testing [2]. Now that we have three general groups, let's look at them in more detail.

First off, we have automated unit testing. This would cover any tests that involve the code itself [2]. For example things like typos, warnings, and bugs would all be part of unit testing [2]. There are many testing tool softwares out there, and some can work with multiple languages while others only work with a few [4]. The only one I have worked with in the past and would actually very highly recommend is Microsoft's unit testing framework that comes in Visual Studio as part of the compiler [4]. It works really well with C++ from what I have experienced, and whenever some code compiles with warnings or errors, there is normally an error code that is well documented online. Some other examples of automated unit testing software would be Check for the C language, Mocha for the JavaScript language compiled in NodeJS, and SimpleTest for the PHP language.

Next is web service and API testing. This type of testing involves connections to other softwares [2]. One of the only similarities between web service and API testing and unit testing is that both can be tested manually, or using tools [5]. These tools work based on which web services protocol your program is using [5]. The two main protocols that are followed for web services are SOAP (Simple Object Access Protocol) and REST (Representational State Transfer architecture) [5]. Some tools like SoapUI only test one kind of protocol, and others like Postman can test both [5, 2].

Lastly, we have GUI testing. As you could probably guess, this is testing the graphical user interface [2]. Just a quick reminder of what that is, it is the part that the user interacts with, and is generally what plays the largest role in the look and feel of the software [2]. These tests involve making sure everything is aligned properly, error messages come up and display in a readable manner, instructions and labels are clearly displayed and easy to understand and read, text size and color do not interfere with readability as well as look appealing, and the user interface looks overall appealing and easy to understand [6]. There are a few different testing tools for this including Ranorex, Selenium, QTP, and Cucumber [2, 6].

All these tests are necessary in creating software, but tools like these may not be. Most of them perform checks that we can already do manually with time [2]. The major benefit to these tools are when development is rushed and there is no time to manually check every line of code [5]. Sometimes, they can catch smaller errors that a person would not catch. An example of this would be when a GUI is off by one or two pixels. It can be very difficult for a person to see that one button in a row of them is off by such a small amount while a program checking their alignment by pixel would notice without a problem. I also wish I had a tool for HTML while working on our senior project. I had one small misspelling that caused me to be unproductive for hours while I looked for it.

## References

1. Safeguard Your Code: 17 security tips for developers
   https://www.infoworld.com/article/2078701/safeguard-your-code--17-security-tips-for-developers.html

2. 14 of the Best Automation Testing Tools Available Today
   https://dzone.com/articles/14-of-the-best-automation-testing-tools-available

3. What is Software Security
   https://medium.com/the-framework-by-tangram-flex/what-is-software-security-e03a5ee7a6b5

4. 20 Most Popular Unit Testing Tools in 2021
   https://www.softwaretestinghelp.com/unit-testing-tools/

5. Web Services Testing Tutorial
   https://www.guru99.com/webservice-testing-beginner-guide.html

6. GUI Testing Tutorial
   https://www.guru99.com/gui-testing.html