Software Security

An Industrial Internship Report

submitted by

Atul Koirala (19BCE2666)

in partial fulfilment for the award of the degree of

BTECH.

in

COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

March 2022

DECLARATION BY THE CANDIDATE

I hereby declare that the Industrial Internship report entitled **"Software Security"** submitted by me to Vellore Institute of Technology, Vellore in partial fulfilment of the requirement for the award of the degree of **BTECH** in **COMPUTER SCIENCE AND ENGINEERING** is a record of Bonafede industrial training undertaken by me under the supervision of **Michael Hicks, University of Maryland, College Park.** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Signature of the student

Name: Atul Koirala

Reg.No.:19BCE2666

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

BONAFIDE CERTIFICATE

This is to certify that the Industrial Internship report entitled "Software Security" **submitted by** Atul Koirala (19BCE2666) to **Vellore Institute of Technology,** Vellore in partial fulfilment of the requirement for the award of the degree of **BTECH** in **COMPUTER SCIENCE AND ENGINEERING** is a record of Bonafide Industrial Internship undertaken by him under my supervision. The training fulfills the requirements as per the regulations of this Institute and in my opinion, meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Signature of the Supervisor

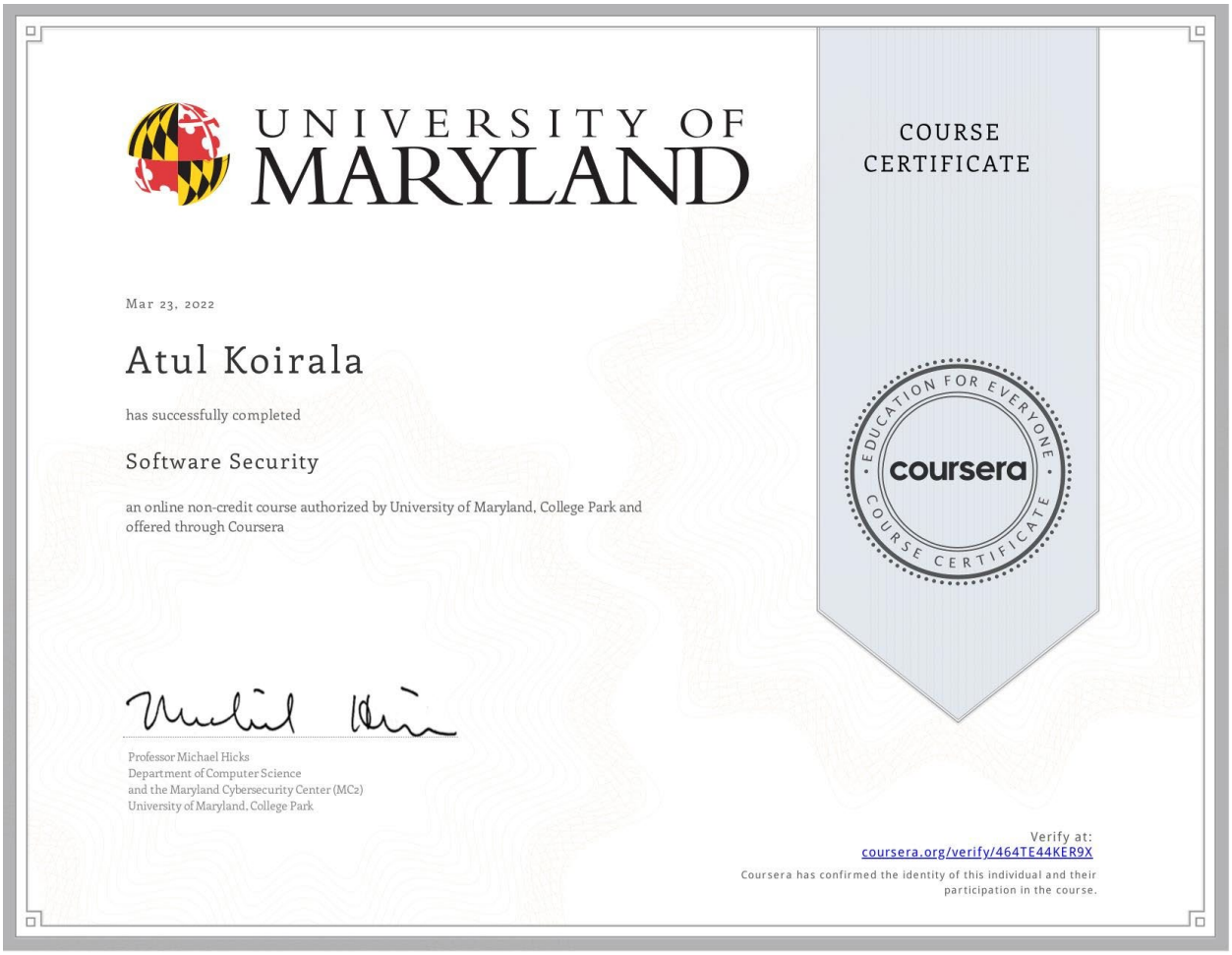
SUPERVISOR

Date:

Date:

Internal Examiner (s)

External Examiner (s)


UNIVERSITY OF MARYLAND
COURSE CERTIFICATE
Mar 23, 2022
Atul Koirala
has successfully completed
Software Security
an online non-credit course authorized by University of Maryland, College Park and offered through Coursera
EDUCATION FOR EVERYONE
coursera
COURSE CERTIFICATE
Professor Michael Hicks
Department of Computer Science
and the Maryland Cybersecurity Center (MC2)
University of Maryland, College Park
Verify at:
coursera.org/verify/464TE44KER9X
Coursera has confirmed the identity of this individual and their participation in the course.

ACKNOWLEDGEMENT

I would like to express my special thanks to Prof. **Michael Hicks** for instructing and explaining the concepts of the course very clearly. I would also like to thank **University of Maryland** and Coursera for hosting the course online. Also, I am very grateful to Vellore Institute of Technology (VIT) for supporting and providing me an opportunity to study the course.

Place: Vellore

Name: Atul Koirala

Reg. No.:19BCE2666

Date: 1st March 2022

TABLE OF CONTENTS

LIST OF FIGURES

1. SYNOPSIS OF THE REPORT

Coursera is an online learning platform that offers a variety of courses and partnerships with more than 200 universities. "SOFTWARE SECURITY" is a course offered by the University of Maryland, College Park. I had the opportunity to learn a lesson and learned about software security tools and their functionality. Throughout the course, I learned various new concepts about Software Security. Software security is a computer security component that focuses on secure build and software usage. It is the idea of designing software to keep it running smoothly in the face of vicious attacks. Software Security defines methods, frameworks, procedures, and strategies for increasing security and risk reduction in software and operating environment. First, in this course, I learned about low security. Where there is basic information about software security. I learned about memory structure, buffer overload, injecting code, and the risk of bullet unit. Then I learned to protect myself from low-level exploitation where memory security, how to avoid bullying and protect codes. And then after learning about web-related security. In this section on how hackers use tools to hack web information like sql injection session, cross-site text, web application fraud and web2.0, I later learned how to protect our web information despite these attacks.

We also learn about how to develop a secure software development that contains risk analysis where we need to find the least risk and resolve it. First, we must make documents of all security requirements and we must use all security resources. Designing secure software where data hacked by a hacker was minimal. we can design the best algorithm to protect useful and important data so that various hacking tools have failed to access our system software. Creating secure software that protects the data from unauthorized access. A great way to protect data. Avoid different types of web and mobile version attacks. And then we must analyze the structure of the system where repetitive code and incorrect code should be removed when static, flow and symbolic analysis are made.

Static Analysis automatically analyzes source code without using the app. flow analysis achieves balance between performance and accuracy to detect safety risks within seconds. Symbolic practice is a useful software testing tool to assist the production of test data and to ensure system quality. Finally, we need to check our software. There will be pen testing and

assembly. The function of the pen tester is to detect the risk in the test entry systems. Although fuzzing attacks are an automated process used to detect application risks.

2. About the organization

Coursera is an online e-learning platform that offers thousands of courses including expertise and degrees. It works with top Universities and businesses to facilitate its accredited online courses. Coursera is an online learning course founded by two professors of computer science at Stanford University. It offers thousands of online courses in partnership with 200 of the world's leading universities and companies, including Yale, Princeton, Penn, Google, IBM, Amazon, Facebook, and more.

The course offers individual courses as well as master's degree programs as well as degrees that reduce barriers to higher education. There are also professional certification programs designed to assist employees in finding new roles or promotions. Coursera study programs are divided into 6 main types: -Studies: Coursera offers more than 3,900+ online courses with 100 free courses taught by senior lecturers from Universities and Companies at world level. -Directed Projects: Allows us to acquire job-related skills and industry tools in less than 2 hours. Collaborative reading with step-by-step visual instructions with prices starting at $ 9.99. -Specializations: A series of courses suitable for someone who wants to learn a job skill. We may decide to take only one course that is closely related to the job or the series of lessons that may cover all aspects of that skill. -Professional Certificate: Training programs that prepare us for a particular job or industry. Some professional certificates are recognized by an organization or school. -Master Tracking: Semester length courses only available as a unit. We cannot take individual courses as part of Master Track and certificates are only issued after all Master Track has been completed. Online Degrees:

Coursera has partnered with top Universities such as Princeton and Yale to offer students the most affordable and accredited University Degrees. -3- Coursera courses last about four to twelve weeks, with one to two hours of video lectures per week. These courses offer questions, weekly tests, peer-reviewed and revised assignments, a special Honors assignment and sometimes a final project or completion exam. Hong Kong University of Science and Technology (abbreviated as HKUST) is a public research university on the Clear Water Bay

Peninsula, Hong Kong. Founded in 1991, Hong Kong University of Science and Technology (HKUST) is a world-class international research university dedicated to quality education and research. The University was established with the aim of advancing learning and knowledge through teaching and research, especially in science, technology, engineering, administration and business related to humanity and social science, and to assist Hong Kong's social and economic development. In just over 25 years, this young and highly successful University has risen dramatically to occupy its place among the highest institutions, rising to international prominence and gaining many accolades and recognition.

HKUST is ranked No.41 worldwide among 1,000 universities and No.1 in the top 350 new universities in the world by Times Higher Education (THE) in 2019. Our schools are also flying high one by one. The School of Engineering is ranked No.23 in THE World University Rankings for Title - Engineering and Technology (2019) and No.3 in Greater China. The HKUST Business School program required by the Kellogg-HKUST Executive MBA (EMBA) has also risen to the international level of Financial Times EMBA nine times in the last 12 years. HKUST provides strong strategic infrastructure support with students to overcome traditional barriers and field challenges in their research. It creates a multidisciplinary environment that fosters cooperation, which helps to generate fundamental ideas, and allows these ideas to be explored and realized.

3. Skillset before Training

We have a course named “Information of Security Analysis and Audit” (ISAA) and "Information of Security Management"(ISM) in our curriculum. I have completed ISAA course, but currently I am studying ISM, before studying this course. So, I was known about all the prerequisites before taking this internship course. I already had a good foundation in the Software development which helps me to understand CSRF, XSS, SQL Injection etc. because of the course offered in our curriculum. But this course helps me to get the theoretical portion of the Software Security that How Coding should be done to reduce vulnerability, How to Secure a software from different attacks and so on.

4. Course Overview

This tutorial will examine the basics of software security. We will consider software risks and aggressive attacks - such as overcrowding, SQL injection, and hijacking sessions - and will consider defenses that prevent or mitigate these attacks, including advanced testing and systems analysis methods. Importantly, we take the attitude of "building security", considering strategies in each stage of the development cycle that can be used to strengthen the security of software systems. Successful students in this subject usually complete their sophomore / level-junior-level degree in technical fields, become proficient in editing, fluently in C / C ++ and other single "managed" programming languages (such as ML or Java), and have prior exposure. in algorithms. Students who do not know these languages but also others can improve their skills by using online web courses.

5. What is Software Security?

Software security is a concept used to protect the software from malicious attacks and other hackers so that the software can continue to operate smoothly under potential risks. Security is needed to provide integrity, assurance, and accessibility.

Software security is one major problem needed to build reliable software systems. Over the past few decades, we have seen a growing interest in the field of security research. Several researchers have explored this topic by offering new solutions regarding the security model, the development of security features, and the specification and implementation of security measures to be integrated into software programs. Along with the emergence of security concerns, security testing has also gained great interest as it needs to be developed simultaneously to enhance software security. In fact, it is important to ensure that the existing security measures are used properly. Testing these security measures is very important to avoid ending up with security errors within the system or operating system.

Fig: -1

Most Common Attacks on Software

Buffer overload, stack overload, command injection and SQL injections are the most common attacks on software. The Buffer and stack overload override the contents of the heap or stack respectively by writing extra bytes.

The command injection can be accessed in the software code where system commands are most used. New system commands are added to existing commands for malicious attacks. Sometimes a system command can stop apps and trigger DOS attack. SQL injections use malicious SQL code to obtain or modify important information on web servers. SQL injections can be used to pass login credentials. Sometimes SQL injections download important information from a website or delete all important data from a website.

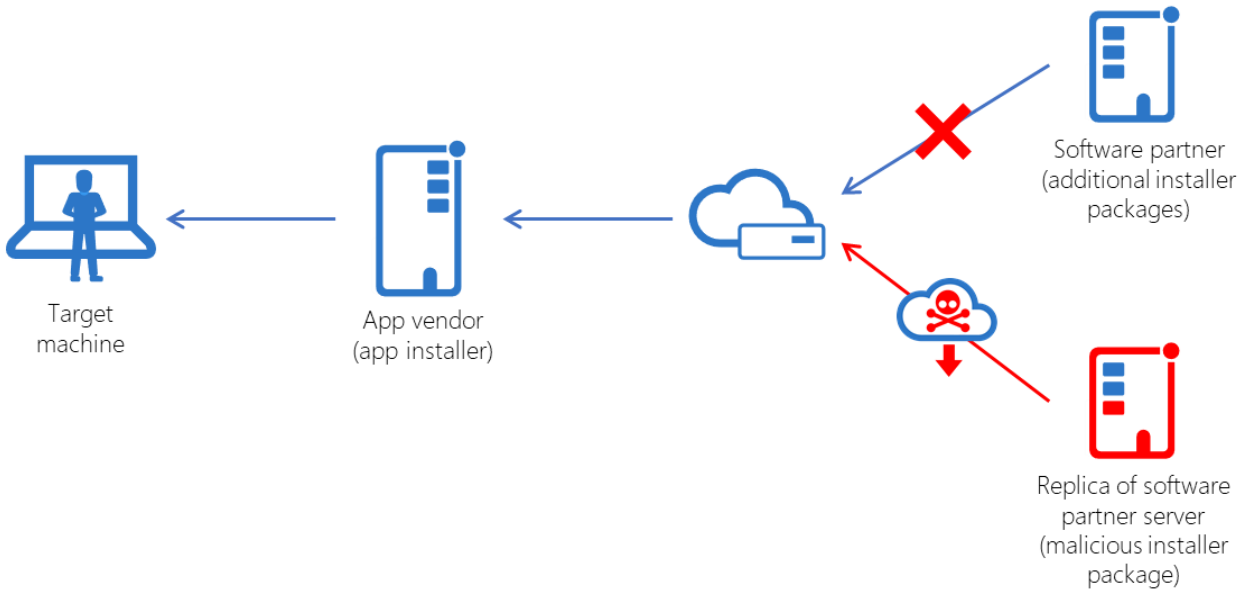


Fig: -2

6. KNOWLEDGE ACQUIRED

6.1 Week 1: Low Level Security

6.1.1 Buffer Overflow

The baths are archives that temporarily hold data when it is transferred from one place to another. Bath overload (or bath overload) occurs when the data volume exceeds the final memory capacity. As a result, a program that attempts to write data to the buffer writes over nearby memory locations.

For example, the login database may be built to wait for 8 bytes of usernames and passwords, so if the function includes 10 bytes of bytes (i.e., 2 bytes more than expected), the system may write excess. data exceeded bar border.

Buffer Overflow is a bug that affects low level code typically in C & C++. Normally, programs crash with this bug. But attackers can alter the situation and steal, corrupt, and run their own code. This occurs when the program tries to write more data to a buffer that it can hold. The string has seven characters. Whereas the buffer in the local function only allots four characters. so, we're going to buffer overflow when we call strcpy.

Fig: -3

6.1.2 Code Injection

Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application. This type of attack exploits poor handling of untrusted data. These types of attacks are usually made possible due to a lack of proper input/output data validation, for example: allowed characters (standard regular expressions classes or custom), data format, amount of expected data

Code Injection differs from Command Injection in that an attacker is only limited by the functionality of the injected language itself. If an attacker is able to inject PHP code into an application and have it executed, they are only limited by what PHP is capable of. Command injection consists of leveraging existing code to execute commands, usually within the context of a shell.

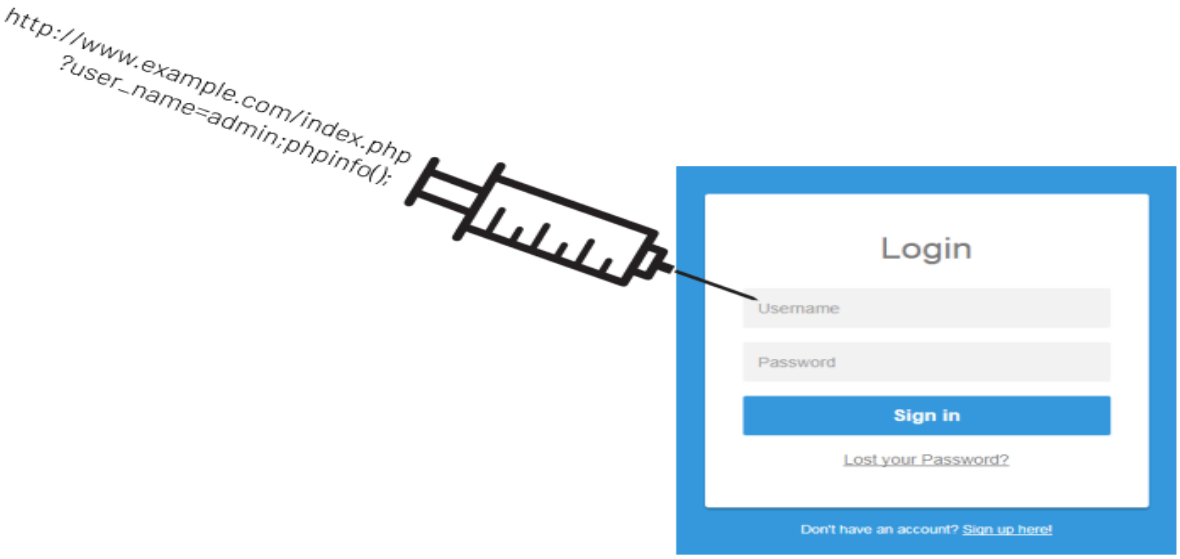


Fig: -4

6.1.3 Other Memory Exploits

Heap Overflows are a sub-class of the Buffer Overflow vulnerability that can affect applications written in many programming languages, and the name describes any situation in which the software attempts to move data from one location in memory into a fixed-length buffer allocated on the heap, which is too small to hold the data. Heap Overflows is a type of buffer overflow that occurs in the heap data area. A heap overflow is a buffer overflow, where the buffer that can be overwritten in the heap portion of memory. Heap memory is a part of memory allocated which is shared by all executing threads in the application. Generally meaning that the buffer was allocated using a routine such as malloc().

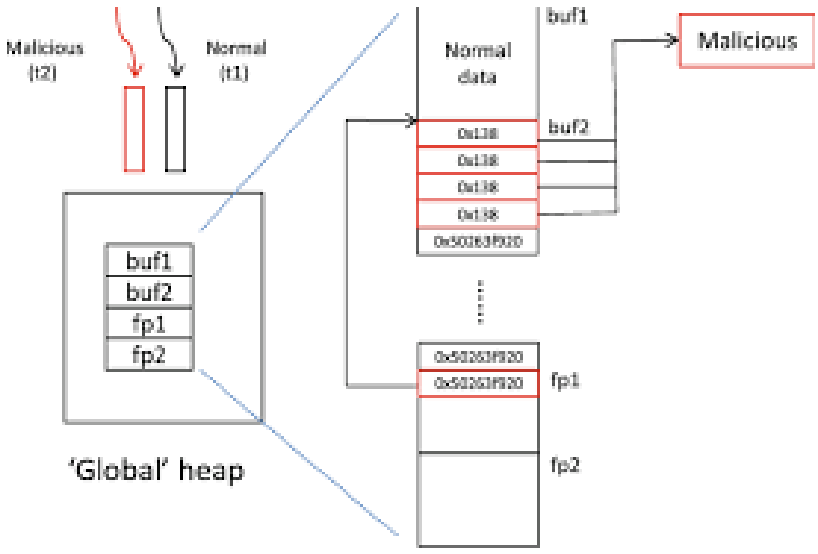


Fig: -5

6.2 Week 2: Defending against Low Level Exploits

6.2.1 Memory safety

Memory protection is a form of protection against various software errors and security risks when operating and accessing memory, such as overflowing bathtubs and hanging pools. For example, Java is said to be safe for memory because error detection checks list parameters, and non-references. Non-memory languages also have a negative impact on the stability, production, and performance of the application.

Because languages that are not protected from memory often allow for multiple distractions and crashes, application stability can have a significant impact. Even if a crash does not protect the defender, it is still a very bad experience for users.

Worse still, these bugs can be very difficult for engineers to track. Memory damage can often cause the crash to occur far away from where the disturbance is. If more mixing is involved, the further disruption can be caused by the small difference in the chain function there, making it even more difficult to produce bugs. The result is that engineers often have to stare at hourly reports to determine the cause of corruption. These bugs may remain unchanged for

months, and the developers are absolutely convinced that an error exists, but they have no idea how to fix the cause and fix it.

6.2.2 Avoiding Exploitation

An exploit is a code that takes advantage of a software vulnerability or security flaw. It is written either by security researchers as a proof-of-concept threat or by malicious actors for use in their operations. The first way we will look at it, is to find abundance through stack canaries. Launch standard risk scan. Practice secure network engineering. Bullying involves being bullied, forced, or forced to do something you do not want to do for someone else. It is a complex and hidden story. Victims may find themselves in situations where thcey are subjected to violence and may be forced to participate in criminal activities.

6.2.3 Secure Coding

Secure encoding is the practice of making computer software in such a way as to prevent accidental transmission of security risks. It can be done in at least two ways.

First, engineers can apply the discipline, limiting their own risk-taking coding patterns. Such patterns are written in coding levels such as CERT C code level or other collected intelligence.

Second, developers can also use advanced code reviews and testing methods to detect risk before shipment. Therefore, using a secure cable library, we will catch this error. Strlcpy will write zero in third place.

Secure encryption is a set of processes that consider security of how the software will be encrypted and encrypted to better protect you from online attacks or vulnerabilities. Errors, bugs, and logical errors are the main cause of frequently used software vulnerabilities, and security experts have found that most of the risks stem from a small number of common software system errors. Secure coding standards introduce protections that reduce or eliminate the risk of leaving security risks in the code.

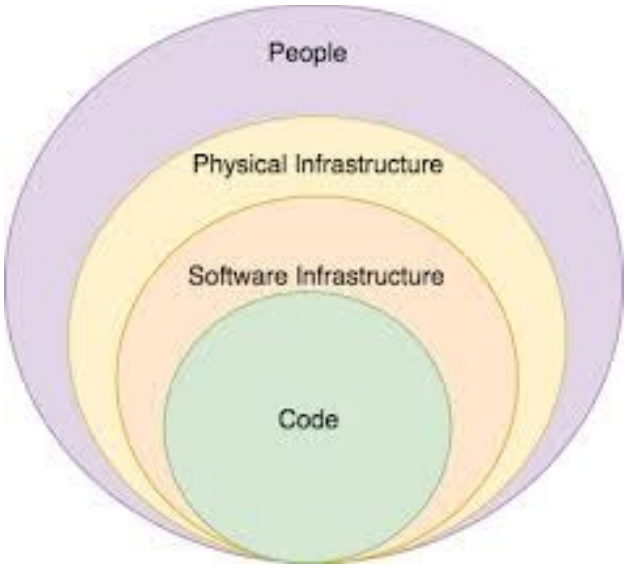


Fig: -6

6.3 Week 3: Web Security

6.3.1 SQL injection

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure or perform a denial-of-service attack.

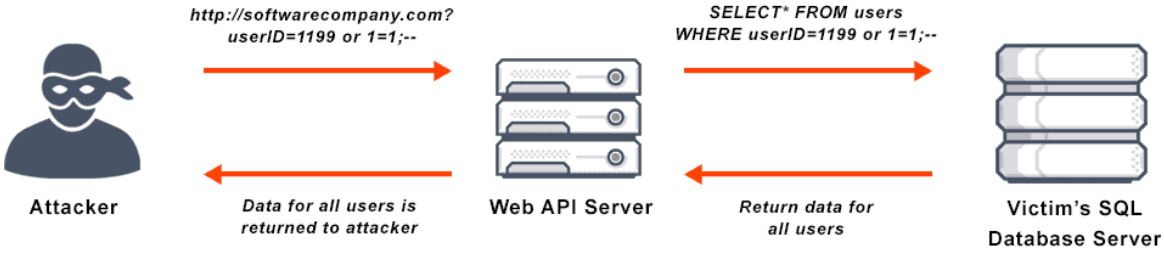


Fig: -7

6.3.2 Session Hijacking

Session Hijacking Attack involves exploitation of the web-based control system, which is usually controlled by a session token.

Because http connections use many different TCP connections, a web server requires a way to detect individual user connections. The most effective method depends on the Web Server token that it sends to the client's browser after successful client verification. A session token is usually constructed of a series of variable widgets, and can be used in a variety of ways, such as in URLs, in the http request header such as a cookie, in other sections of the http request title, or in the body of the http request.

Sack hacking session damages a session token by stealing or predicting a valid session token to gain unauthorized access to a Web Server.

6.3.3 Cross-site request forgery

A fraudulent multi-site application (also known as CSRF) is a security compromise that allows the attacker to entice users to perform actions that they do not intend to perform. It allows the attacker to partially avoid the same root policy, designed to prevent different websites from interfering with each other.

Thus, the victim is committing a vicious act on his own but is unaware of it. How? The victim is loading a page somewhere that contains a malicious link to html (i.e., img src) or a targeted website that contains XSS vulnerabilities, and it is a good point to upload harmful JavaScript to external and uninstall ajax applications.

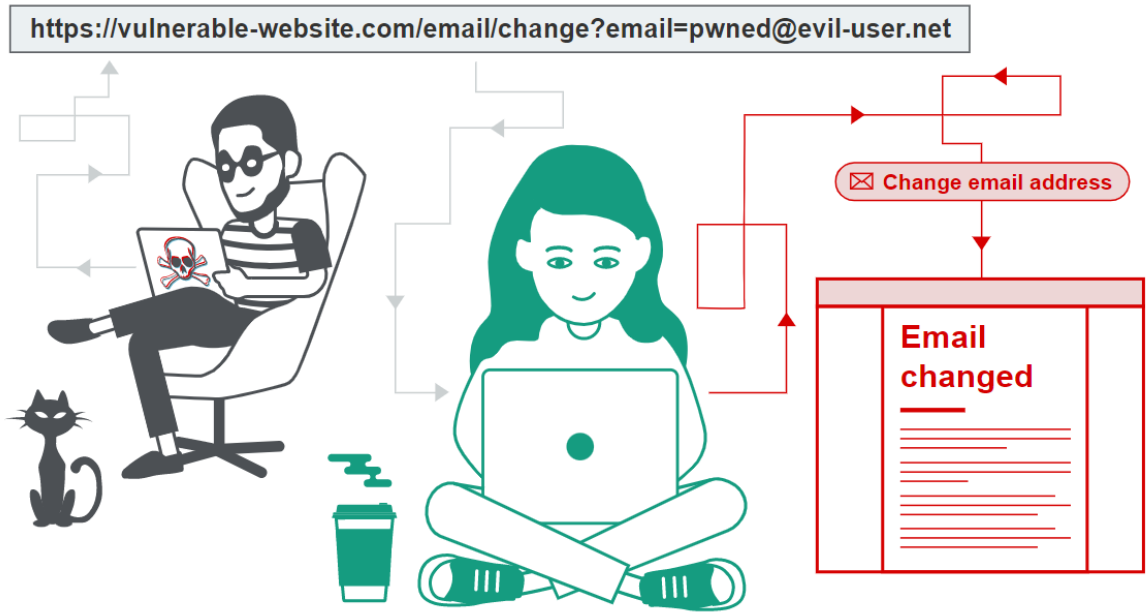


Fig: -8

6.4.4 Cross-Site Scripting

Cross-Site Scripting (XSS) attacks are a form of injection, in which dangerous texts are injected into websites if they are not and are reliable. XSS attacks occur when an attacker uses a web application to send malicious code, usually in the form of a separate browser text, to a different end user. Errors that allow these attacks to be successful are very widespread and occur anywhere when a web system uses input from a user within the output that you produce without verifying or encrypting.

Attacker can use XSS to send malicious script to an unsuspecting user. The end user browser has no way of knowing if the script should be trusted and will use the script. Assuming that

the text is from a trusted source, a malicious script could access any cookies, time tokens, or other sensitive information stored on the browser and used by that site.

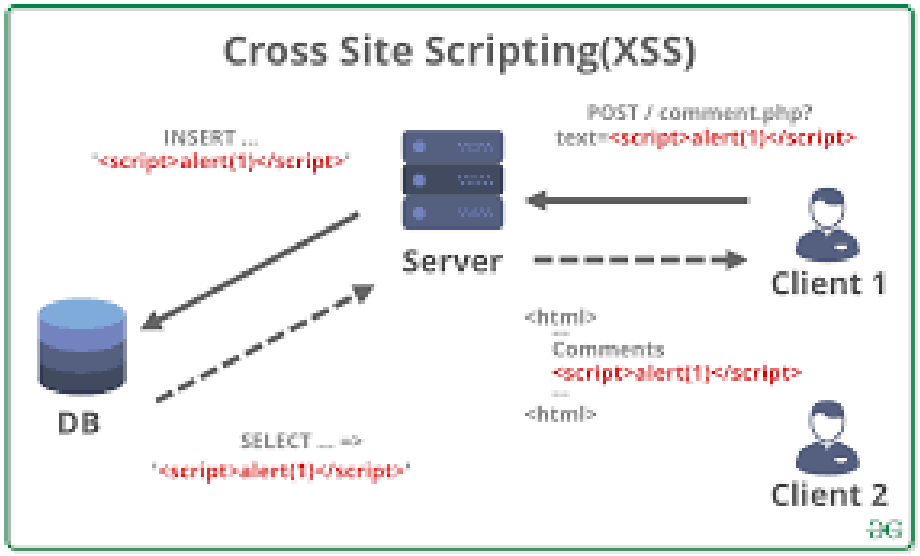


Fig: -9

6.4 Week 4: Secure Software Development

6.4.1 Designing and Building Secure Software

Secure Software Design incorporates Loren Kohn Felder 'experience of more than two decades into a concise, effective guide to improving the safety of technology products. Written for a variety of software experts, it emphasizes building security in early software development and involving the entire team in the process.

The book begins with a discussion of key concepts such as trust, threats, mitigation, secure design patterns, and cryptography. The second part, perhaps the book's unique and most important contribution to the field, covers the process of designing and updating the software design with consideration in mind. The last section provides details on the most common code-generating errors, using code captions written in C and Python to illustrate the use of the code.

Designing a secure software where the data hacked by hacker was minimum. we can design a best algorithm to protect the useful and important data so that the different hacking tools which failed to reach into our system software. Building secure software which cause protecting data from unauthorized which. A good way to protecting data. Avoid different types of webs as well as mobile version attacks.

6.4.2 Risk analysis

It defines what type of possibility risk are present on the software so that we can make a proper risk-free software which not able to hack by hackers. Risk analysis is the process of identifying risk in the application and prioritizing them to test. Reduce the impact of a negative event. Evaluate whether there are more benefits to a project than risks before initiation. Plan the company's response to emergencies or other adverse events. Eliminate risks during a process.

Fig: -10

6.4.3 Security Requirements

Security Requirements (SR) practices focus on key security requirements in the context of secure software. The first type addresses common software-related requirements for specifying goals and expectations for protecting services and data at the core of your application. The second type addresses the requirements of supplier organizations that are part of the application's development context, especially outsourced development. Outsourced development can have a significant impact on application security, so it's important to streamline expectations for secure development.

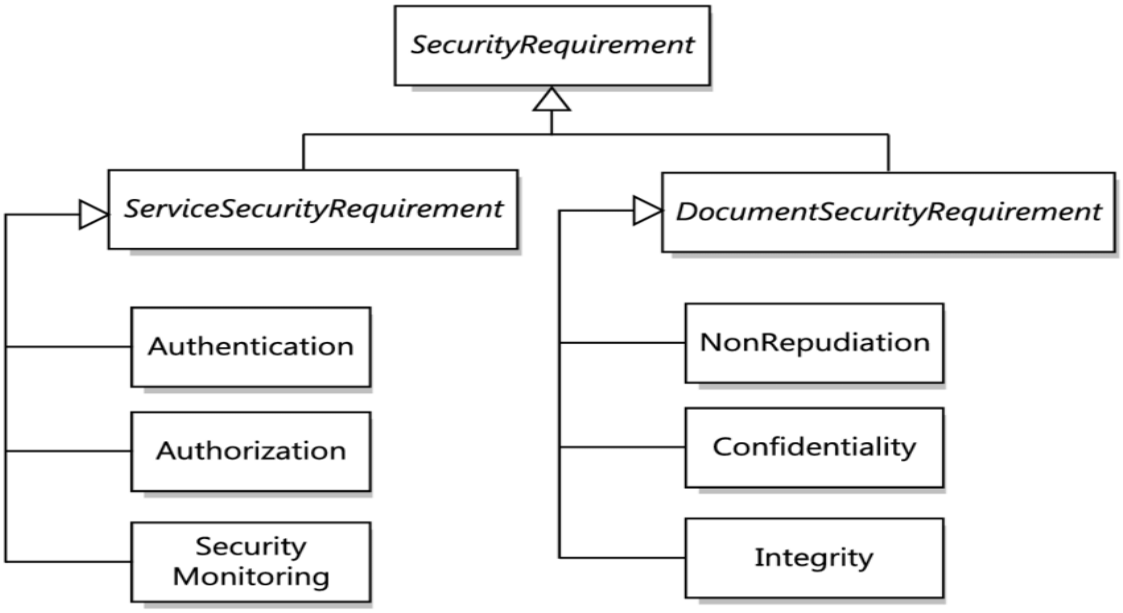


Fig: -11

6.5 Week 5: Program Analysis

6.5.1 Static Analysis

Static Analysis automatically analyzes source code without using the app. If the analysis is done during the execution of the program, it means that the analysis is robust. Strict analysis is often used to detect security risks. Performance issues.

Static analysis, also called static code analysis, is a computer error correction method performed by checking the code without using a program. The process provides insight into the structure of the code and can help ensure that the code complies with industry standards. Static analysis is used in software engineering by software development teams and quality assurance teams. Automatic tools can assist programmers and developers in performing static analysis. The software will scan all code in the project to check the risk while verifying the code.

6.5.2 Flow Analysis

Flow analysis science to perform quantitative analysis chemistry in flowing streams. This article provides an overview of the key aspects of flow analysis, features and methods of flow analysis, theory, and application. The main features of flow analysis techniques are that the sample management can be reproduced, the flow sample analysis method is closed, the sample time is short, the reagent consumption is low and concentration gradients established within several flow can be used. The main methods of flow analysis considered in this article are the analysis of the flow and the flow analysis separately (flow injection and consecutive injection).

6.5.3 Basic Symbolic Execution

Symbolic execution is a software testing technique that is useful to aid the generation of test data and in proving the program quality analyzing a program to determine what inputs cause each part of a program to execute. Symbolic execution cannot proceed if the number of iterations in the loop is known.

In computer science, symbolic execution (also symbolic evaluation or symbol) is a means of analyzing a program to determine what inputs cause each part of a program to execute. An interpreter follows the program, assuming symbolic values for inputs rather than obtaining actual inputs as normal execution of the program would. It thus arrives at expressions in terms of those symbols for expressions and variables in the program, and constraints in terms of

those symbols for the possible outcomes of each conditional branch. Finally, the possible inputs that trigger a branch can be determined by solving the constraints.

6.6 Week 6: Penetration Testing

6.6.1 Pen testing

Entry testing is also known as pen testing or behavioral robbery. Describes the deliberate introduction of simulated online attacks that seek usable risks to computer programs, networks, websites, and applications. Although the main purpose of pen testing is to detect security vulnerabilities, access control tools can also be used to assess the organization's security policy, compliance with its rules, safety awareness of its employees, and the organization's ability to identify and respond to security. incidents as they occur.

Pen testing can be done automatically using protective tools or it can be done manually. To provide important information on an organization's ability to fine-tune its security policies and to amend risks identified, the entry test must identify risks that would allow access to the attackers' system. The process includes collecting information about potential intentions, identifying potential hotspots, attempting to break into - or possibly actually - and reporting the findings to the organization's security team.

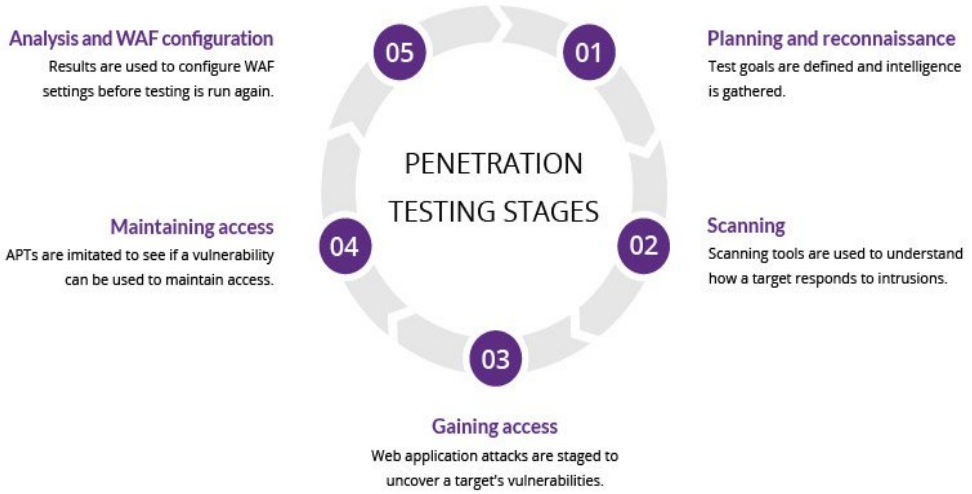


Fig: -12

6.6.2 Fuzzing

A fuzzing attack is an automated process used to find application vulnerabilities. Fuzz testing is basically a kind of random testing which is a kind of testing in which inputs for a test case are generated randomly or semi randomly. An example of a network-based fuzzer is SPIKE. SPIKE is a fuzzer creation kit and it provides a C language API for programming fuzzers in C interact with remote servers using network-based protocols. Another example network fuzzer is Burp Intruder. It's one element of the Burp suite.

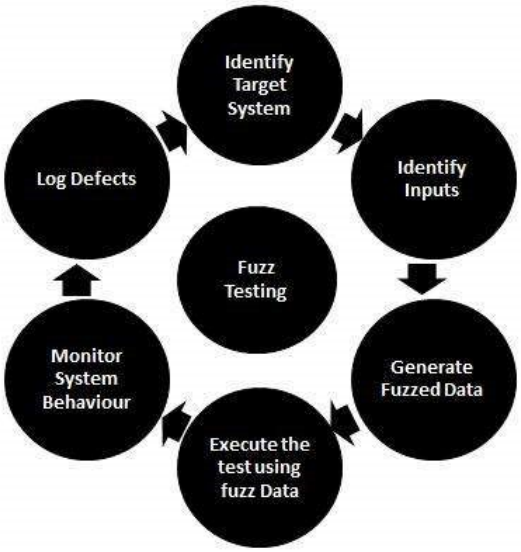


Fig: -13

7. Enhancement

I would have liked to have the material in written form as well. It is much easier to review text than to re-watch videos. Also, a more detailed syllabus would have been good, so you know what you can expect to learn. At least this course had a written syllabus, though; some other Coursera courses don't have any sort of a syllabus, which is obviously worse (an introductory video is no substitute, in my opinion).

There were also some interviews with people in the industry. This is a great idea in theory, but unfortunately, I didn't find the ones I watched very interesting. I have had the same experience for other courses, so this problem is not unique to this course. If there were more practical than theory then that would be best for me.

8. Conclusion

All in all, a good course that did not take much time to complete. Although there is a lot you can learn about software security, it is still a good introduction for all engineers to benefit from.

Good software security practices can help ensure that software behaves properly. Safety-critical and high-assurance system designers have always taken great pains to analyze and to track software behavior.

It also made me remember all the kinds of problems you might have in C programs that just disappear in other programming languages, and it made me happy to move from C / C ++ to Java, and then to Python!

9. References

[1] K. Tsipenyuk, B. Chess and G. McGraw, "Seven pernicious kingdoms: a taxonomy of software security errors," in IEEE Security & Privacy, vol. 3, no. 6, pp. 81-84, Nov.-Dec. 2005, doi: 10.1109/MSP.2005.159.

[2] S. Barnum and G. McGraw, "Knowledge for software security," in IEEE Security & Privacy, vol. 3, no. 2, pp. 74-78, March-April 2005, doi: 10.1109/MSP.2005.45.

[3] B. Arkin, S. Stender and G. McGraw, "Software penetration testing," in IEEE Security & Privacy, vol. 3, no. 1, pp. 84-87, Jan.-Feb. 2005, doi: 10.1109/MSP.2005.23.

[4] G. McGraw, "From the ground up: the DIMACS software security workshop," in IEEE Security & Privacy, vol. 1, no. 2, pp. 59-66, March-April 2003, doi: 10.1109/MSECP.2003.1193213.

[5] G. McGraw, "Software security," in IEEE Security & Privacy, vol. 2, no. 2, pp. 80-83, March-April 2004, doi: 10.1109/MSECP.2004.1281254.

[6] B. Potter and G. McGraw, "Software security testing," in IEEE Security & Privacy, vol. 2, no. 5, pp. 81-85, Sept.-Oct. 2004, doi: 10.1109/MSP.2004.84.