Application & Web Security

CISC 3325 — CUNY Brooklyn College
Lecture Notes

Application & Web Security

In this chapter, we will:

1. List the characteristics that make an application secure.
2. Learn more guidelines on secure design and development of applications.
3. Explain the vulnerabilities of web browsers, web servers, and web applications, and discover the controls that should be used to eliminate these vulnerabilities.
4. Focus on web application attacks, including buffer overflow attacks, SQL injection attacks, command injection attacks, cross-site scripting attacks, cookie poisoning attacks, and session hijacking attacks.

Application & Web Security

Topic 3: Access Control demonstrated how critical it is to secure one's physical assets, and Topic 4: Cryptography showed that cryptographic methods ensure the confidentiality and/or Authenticity of data.

However, these aren't the only realms of computing where security measures must be taken: processing of data (a.k.a., by an application) and transmission of data (via a network) must be secured as well.

Moreover, compared to physical/hardware security, for example, making sure that an application is secure (that is, that it handles data securely) is far more difficult due to the enormous number of ways in which 'things can go wrong', such as app misconfiguration, invalidated inputs, errors in coding, man-in-the-middle attacks, man-in-the-browser attacks, session hijacking, weak encryption keys, weak passwords, weak authentication mechanisms, SQL injection, and buffer overflows.

In this chapter, we will review the various sorts of issues that can happen with the functionality of an application and the remedies to / prevention of these issues. Later, Topic 8 will touch on network security.

Characteristics of a Secure Application

A software application is as good as what it does. It should not do more than what it is expected to do. At the same time, it should not fail to perform what it is expected to do.

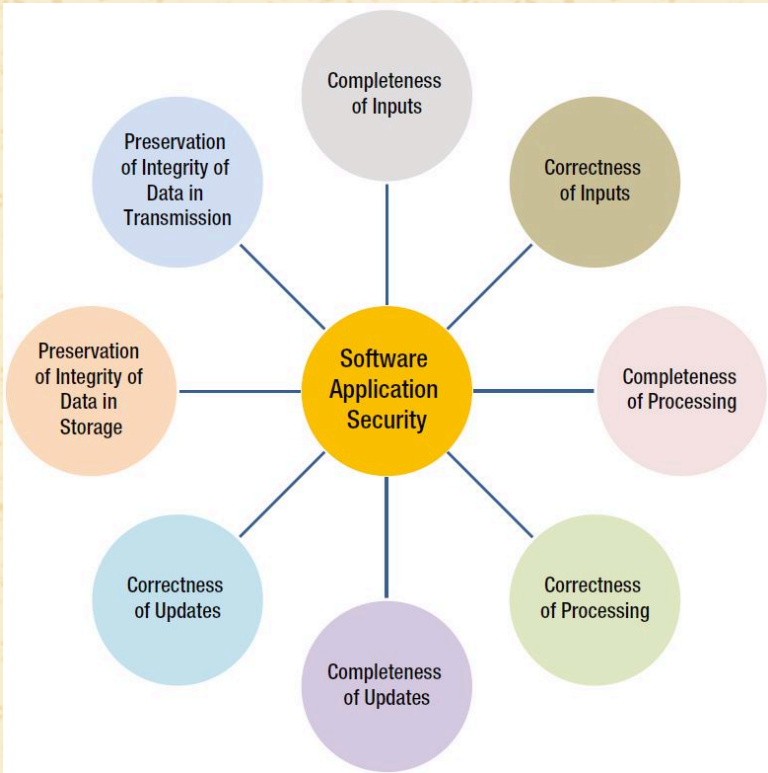
Here are 8 characteristics that fundamentally define a secure application:

1. **Completeness of the Inputs**: If an application requests only a part of all the necessary inputs (data), the app might not be able to carry out the right computations due to the lack of data, which will lead to missing or erroneous outputs.
2. **Correctness of the Inputs**: If an application doesn't check whether the entered input matches the required data format, it risks not only corrupting data but also the confidentiality thereof (e.g., SQL injection attacks) and the security of the users of the app (e.g., cross-site scripting attacks). Things can get wildly worse, e.g., when an incorrect medication dosage is entered, leading to a life threat.
3. **Completeness of Processing**: An application must verify that all the tasks that it was supposed to perform are indeed done. If the processing is partially complete, then the partially completed portion has to be reversed and the entire task has to be redone or the other portion has to be completed to ensure the integrity of the data.

Characteristics of a Secure Application

4. **Correctness of Processing**: If an application doesn't ensure that the computations and changes that it makes are accurate, the integrity of the data becomes questionable and data may become useless or not reliable.
5. **Completeness of the Updates**: When an update is only partially integrated into the application, or if the update is missing information, the application will suffer from outdated information, leading to wrong decision making.
6. **Correctness of the Updates**: Just as with incorrect input, incorrect updates result in corrupted data and wrong decisions (i.e., breach of integrity). Verification of all critical updates for correctness has to be provided through the application to ensure that no critical update is wrong and that the critical updates are fully accurate in all respects.
7. **Keeping the Integrity of Data in Storage**: Data stored in a database must be modifiable only through the interface of the controlling application, and never directly through the backend or through any command-line script. Accessing a database directly leads to the breach of Confidentiality, Integrity, and Availability.
8. **Keeping the Integrity of Data in Transmission**: Data often needs to travel from system to system, such as with inter-continental e-commerce transactions. To protect it from Man-in-the-Middle or Man-in-the-Browser attacks, the transmitting app could use mechanisms like encryption.

Characteristics of a Secure Application



The 8 Characteristics of a Secure Application. "Figure 6-1: Aspects of Software Application Security" (page 117), Nayak, U., & Rao, U. H. (2014). *The InfoSec handbook: An introduction to information security* (1st ed.). APRESS.

Important Guidelines for Secure Design and Development

1. Understand the Security Requirements of the application (functionality and data related) and document them as part of the Requirements Specifications Document.
2. Ensure that the Security Requirements are considered during architecture and design of the app.
3. Follow secure coding standards.
4. Validate all the inputs including the boundary checks, checks against allowed values, and format.
5. Ensure strong login mechanisms (including the need for strong passwords).
6. Encrypt transmitted data.
7. Ensure periodic mandatory change of passwords.
8. Apply the Least Privilege Principle when assigning access rights to parts of the application.
9. Handle errors correctly.
10. Handle exceptions correctly.
11. Configure the app correctly.
12. Use vetted (checked) algorithms.
13. Conduct counter-checks to ensure complete and accurate processing of data.
14. Delete all unused functions.
15. Ensure proper logout mechanisms.
16. Use secure protocols.
17. Ensure proper logging and auditing mechanisms.
18. Conduct rigorous testing.
19. Carefully control software releases.

Important Guidelines for Secure Design and Development

Secure Coding Guidelines. "Figure 6-3: Secure Coding Considerations" (page 123), Nayak, U., & Rao, U. H. (2014). *The InfoSec handbook: An introduction to information security* (1st ed.). APRESS.

Important Guidelines for Secure Design and Development

Any questions?

Browser Vulnerabilities + Controls

Browsers are client-side applications for accessing websites and the Internet. Various browsers exist on the market, and each browser is developed differently. As such, one browser might suffer from a vulnerability to a greater extent than another browser. In general, typical vulnerabilities that a browser is prone to are:

1. **Inappropriate Configuration**: Allowing pop-ups, cookies, or scripts on all websites without the user's consent/awareness might lead to the download of malware, which can impact data confidentiality, crash of applications, crash of the computer, or leakage of data being transmitted through the browser.
2. **Untrusted Add-ons**: Adding an extension from an unreliable source could lead to malware infection (if the creator of the extension is a hacker, or the extension has vulnerabilities). One should always disable unwanted extensions.
3. **Malware or Executables Run on the Web Browser**: Certain malicious apps might run on the browser, leading to theft of the data or system infection.
4. **Lack of Patching and Security Updates**: Security patches must be installed as soon as possible to prevent hackers from exploiting vulnerabilities that were recently detected in a certain browser.

Controls of Browser Vulnerabilities

The following precautions are some of the ways to overcome web browser vulnerabilities:

1. Configure the browser with appropriate settings; consult experts if required.
2. Keep the Web Browser regularly updated with Updates and Security Patches.
3. Disable or uninstall unnecessary or untrusted add-ons/extensions.
4. Understand the errors thrown up by the browser and then act on them appropriately. If something isn't fully clear, do not simply act on the same – consult experts.
5. Configure your operating system properly: change the default settings to appropriate settings.
6. Have a well-known and reputed anti-virus installed on your operating system.

Browser Vulnerabilities + Controls

This video originally targets cryptocurrency users, but it highly applies to any general browser user:

https://www.youtube.com/watch?v=cH6TbFOikFU

Server Vulnerabilities + Controls

Web Servers are backend computer programs running on a physical server and hosting websites or serving clients with requested web pages. Examples of commonly used servers are Internet Information Services (IIS) Windows Server from the Microsoft Corporation, Apache Web Server, Nginx Web Server, and Tomcat Web Server. Typical vulnerabilities of servers are:

1. **Unchanged Default Permissions**: Usually, servers like IIS define default users with default, weak, or no passwords at all, and with default permissions.
2. **Sample scripts are not removed**: Servers may come with outdated sample scripts, which hackers may misuse.
3. **Default Configuration is Not Changed**: Unnecessary and unwanted configurations may create loopholes in the server and let attackers exploit them.
4. **File and Directory Permissions are not Set Properly**: Inappropriate permissions will enable hackers to access directories on the Web Server that shouldn't, in principle, be accessible.
5. **Security Loop-Holes or Defects in the Web Server**: Servers become outdated from a security point of view when vulnerabilities in them are discovered and published. Not patching the server can lead to security attacks.

Controls of Server Vulnerabilities

1. Change or disable default users and default settings, and configure the server appropriately.
2. Regularly patch the Web Server.
3. Delete unnecessary and unwanted sample scripts.
4. Patch up security loopholes on the underlying Operating System.
5. Set up file and directory permissions only as absolutely required.
6. Do not perform server changes without impact analysis.
7. Ensure strong passwords for user and administrator accounts.
8. Ensure that only the needed ports are open and only the need-based services are active.
9. Encrypt the traffic as necessary.
10. Ensure data files are kept out of the Web Server.
11. Monitor the Logs regularly.
12. Delete unnecessary file shares.
13. Disable Tracing and Debugging.
14. Check for the validity of Certificates.
15. As much as possible, use a dedicated machine for the server. Other servers like database services should be installed on separate physical servers.
16. Disallow local logins on the Web Server machine.
17. Validate server-side session IDs.
18. Scan the server periodically to identify and fix vulnerabilities.
19. Perform boundary checking and validations on the inputs.

Controls of Server Vulnerabilities

Any questions?

App Vulnerabilities + Controls

Let's discuss some general app vulnerabilities:

1. **Buffer Overflow**: A **buffer** is a synonym of an array. Some lower-level languages, such as C and C++, don't check whether the programmer exceeds the boundaries of the defined array. That is, a programmer can flawlessly access a memory location *beyond* the array that the programmer declared. This situation is called a **buffer overflow**.

   Example: The program at BufferOverflow.c introduces a buffer overflow.
   Question: on which code line does the overflow occur?

   You can compile this program on any device by typing:
   `gcc -o BufferOverflow BufferOverflow.c`
   in the command prompt / terminal, and then execute it by typing:
   `./BufferOverflow` (on a Mac/Linux computer,) or: `BufferOverflow.exe` on a Windows computer.

App Vulnerabilities + Controls

1. ***Why should we spend time learning about a weakness of lower-level languages, if we can instead build projects in Java, Python, etc?***
   C and C++ are two of the most popular languages and are used in many critical platforms: in most Operating Systems' kernels, in web and database servers like Microsoft SQL, Mysql, and Apache, and even in embedded systems such as cars, Mars rovers, industrial robotic arms, etc. Ignoring buffer overflow threats can, therefore, lead to dire consequences!

In what ways could a hacker exploit a buffer overflow?

- Steal private/confidential information.
- Corrupt variables of the current program, another program, or the operating system.
- Write malicious code into Main Memory and execute it.

In the C program on the previous slide, we read data beyond the defined array. Can a hacker also overwrite data?

Unfortunately, yes. This is how a hacker can get to add and run malicious code.

App Vulnerabilities + Controls

1. Exploiting a buffer overflow flaw isn't an easy thing: an attacker first needs to spot this flaw somewhere in a program, and then carefully plan his or her actions, including how much data to overwrite, and what malicious code to insert. Operating Systems might have installed countermeasures, such as non-executable stacks (which are areas of Main Memory that an attacker can never get to run,) which require greater efforts on the side of the attacker.

   Besides exceeding array boundaries, as witnessed in the previous C program, another common source of buffer overflow are uses of C string functions, such as `strcpy()`, `gets()`, `sprintf()`, and `strcat()`, which don't perform any boundary checks. This is just a partial list of such functions in C.

   Thankfully, C also provides alternative functions that prevent buffer overflow from happening: `strncpy()`, `fgets()`, `snprintf()`, and `strncat()`. These functions take the number of characters, `n`, to be read/written/copied, and thereby don't exceed the target string's size (by the way, this is why the names of some such functions contain an extra letter '`n`'.) It is, therefore, advisable that you use the overflow-safe function versions in your coding.

App Vulnerabilities + Controls

1. ***Another buffer overflow example:*** The C program gets_auth.c uses the `gets()` function to read the user's name from the keyboard. If the entered name starts with 'A' or 'a', or if a flag variable defined in the program isn't zero, the program 'authenticates' the user. Otherwise, the program doesn't authenticate them.

   Since the `name` string is of size 30 characters, any response from the user longer than 30 characters will overflow the `flag` variable (making it non-zero), thereby authenticating the user, even if their name doesn't start with an 'A' or 'a'!

   The solution, as we learned on the previous slide, is to use `fgets()` instead of `gets()`. The C program fgets_auth.c, uses this exact solution, thereby eliminating the buffer overflow bug of gets_auth.c.

   The above is a short demonstration of how an attacker who learns about the existence of a buffer overflow vulnerability in a program, or any person by accident, can corrupt the variables of that buggy program (and the OS in general.)

App Vulnerabilities + Controls

Amazing YouTube video explaining and exemplifying buffer overflows:

https://www.youtube.com/watch?v=AD-iXWANggo

App Vulnerabilities + Controls

2. **SQL Injection**: On Slide 5 above, we mentioned the threat of direct access to a database (not through the interface application,) which may cause data corruption and disclosure of confidential information. This attack attempts to do exactly that!

   SQL injection, which is an instance of a group of attacks called **code injections** is an attempt of an attacker to read from, write to, or re-configure a database illegally. **SQL** (Structured Query Language) is a programming language using which programmers communicate instructions to databases. In such an attack, the attacker will type SQL code into an input textbox (e.g., search box, username textbox, password textbox, etc.) and hope that the entered code will achieve its purpose.

   An SQL injection attack might succeed when the website or web app doesn't properly validate the format of its data. For example, most programming languages used for server development, e.g., Java, Python, PHP, etc., come with library functions that 'escape' special SQL characters in user data to minimize SQL injection chances.

App Vulnerabilities + Controls

Fun Interactive SQL Injection Demo: https://www.hacksplaining.com/exercises/sql-injection.

Tom Scott discussing SQL Injections:

https://www.youtube.com/watch?v=_jKylhJtPmI

App Vulnerabilities + Controls

Any questions?

App Vulnerabilities + Controls

3. **Cross-Site Scripting**: This is another attack resulting from unchecked and unfiltered user input.

   A user with malicious intentions might insert HTML or JavaScript code as their input into one of the input fields. If the entered input is unfiltered, the JavaScript code will execute on the pages of the users of the website. For example, the code might redirect the user to a hacker's infecting webpage. The attacker might also insert an HTML-only code, such as one that includes a link to such a dangerous website or, in some cases, simply annoys the users.

   Other instances of Cross-Site Scripting might even break into users' sessions, steal personal information, or infect the users' systems, without even having the users click anything.

   Let's look into the following fun interactive demo of Cross-Site Scripting: https://www.hacksplaining.com/exercises/xss-stored.

App Vulnerabilities + Controls

Tom Scott discussing Cross-Site Scripting:

https://www.youtube.com/watch?v=L5l9lSnNMxg

Controls of App Vulnerabilities

1. Always validate all the inputs including format checking, bounds checking, and acceptable values.
2. Always configure Web Applications appropriately.
3. Regularly patch up all the servers including Web Server, Application Server, and Database Server.
4. Do not save your Login credentials including passwords on the Web Browser.
5. Do not save unnecessary information on your Web Browser.
6. Log off immediately after the work on a Web Application is over – Do not keep the session open unnecessarily for long.
7. Use strong encryption keys and strong encryption where required - Do not store the encryption keys on the Web Server.
8. Define appropriate access rights.
9. Ensure appropriate Log Out mechanisms in the Web Applications.
10. Ensure that the Certificate is Valid and has not expired.
11. Implement effective Cookie Management including Cookie time-out, do not store passwords in a Cookie, authenticate Cookies.
12. Carry out regular Vulnerability Scans and Penetration Testing to understand and fix the underlying vulnerabilities.

Controls of App Vulnerabilities

Any questions?

Application & Web Security: Sources

Rao, Umesh Hodeghatta, and Umesha Nayak. "Chapter 6: Application and Web Security." *The InfoSec handbook: An introduction to information security.* Springer Nature, 2014, pp. 115-139. URL: https://link.springer.com/content/pdf/10.1007/978-1-4302-6383-8.pdf#chapter+6:+application+and+web+security