Security Vulnerabilities

Contributors:

Jessica Fealy – 18024092 :

* Port Vulnerabilities
* SQL Injection
* Software Vulnerabilities
* Integer Overflow

Eduard – 18068235 :

* Buffer Overflow
* Data Vulnerabilities
* Logging Vulnerabilities

Harry – 18136044:

* Using components with known vulnerabilities
* Broken authentication

Port Vulnerabilities

Any port on a computer system can pose a potential threat to an actor wishing to penetrate the system unlawfully. Many computer users may overlook potential flaws in their ports. While the most commons and serious port are often lockdown and protected, many are overlooked as they are not seen as dangerous. However, any unsecured port will pose a security threat and this will be abused by malicious actors. Commonly hacked ports include: TCP port 21 and TCP port 22 (Beaver, n.d.), these ports respectively control the File Transfer Protocol and the Secure Shell, these are often over look, yet pose a serious security vulnerability.

Malicious actors will use port scanning tools, these tools echo out to listen for open ports on any network. Once an open port has been detected the actor will attempt to access this port and gain access into the system. However, the same tools malicious actors use to gain access the system administrators will use to watch for and close ports. By system admins scanning for open ports on their networks they can easily identify and close such ports reducing security threats on the network (Gelnaw, 2021). Simply by doing routine security scans on the admins systems the risk posed by port vulnerabilities is mitigated.

SQL Injection

SQL Injection is the principle of typing SQL code into a text entry field for this to be entered once the command reaches the database (SQL Injection, n.d.). As long as the actors knows the name of a table, such as “ ORDERS “ they can simply enter “ DROP ORDERS “ and this will result in an unsecure system having their orders database deleted. This vulnerability occurs from improper string error checking and just accepting any data the user enters without first validating the information.

However, a successful SQL injection can do more to a computer system then simply delete data. The injection can result in a data breach from the actor being able to pull user data from the database or copy the contents of the database (Academy and injection, 2021). Further attacks can result in a denial of service attack which will render the companies server inaccessible on the internet until the attack is over.

Steps, however, can be taken to mitigate SQL Injection attacks, these include (Rubiens, 2018):

* Privilege levels – having privilege levels inside the database, ensure that accounts connecting to the database do not have privileges they do not require at the time. This method of limited access can prevent larger amounts of data from the database during a successful SQL Injection.
* Constantly validating user fields – by validating the database inputted and to not just give the user blanket trust will ensure that any string taken in from the user will not contain any malicious code.
* As the actor cannot access any table without knowing the tables names initially, ensure that table names are obscure and kept secure from any body without a reason to know that information.

Software Vulnerabilities

Software vulnerabilities, or most commonly known as “ bugs “ can present in many different forms with each having a different level of severity. But can be non-serious, as a button on a GUI being non-functional; to more severe such as the login system encryption being bypassed. While many software bugs will just be inconvenient or displeasing to the user, some can be quite serious to the company. Many different types of vulnerabilities exist that present as software bugs, such as SQL Injections and Buffer Overflow, these each have their own solutions to solve. The main path to mitigating against many software vulnerabilities is to preform constant test and maintenance to fix the security vulnerability as soon as it is discovered.

Integer Overflow

This vulnerability occurs by real-time calculations or user inputted data exceeding the maximum or minimum value for an integer (Mohanty, 2018). For example, for an 8-bit signed integer value ranges from -128 to +127. If any real time calculation went over or under this value, then the result would wrap. For example, with an 8-bit integer if 10 is added to +120 the result would be -126, not +130.

While at face value, this vulnerability does not appear to be as threatening to a computer system, this can have great risk to a system. Such errors can cause program to crash or throw the rest of the program logic out.

Simple pre calculation methods can be employed to first ensure that calculations between 2 number will not result in an overflow.

Buffer Overflow

Buffer overflow is the act of trying to store too much data in a chuck of memory that is not equipped to holster this data (Synopsys, 2019). Trying to attempt to do this will result in unpredictable data being read from memory or causing the program to crash as a result of inaccessible memory issues.

The most often reasoning for this vulnerability to be cause is down to programmer error, this is where the programmer has failed to check the content size of the buffer before they attempt to copy the contents (Cwe.mitre.org. n.d. ). The programmer should instead attempt to restrict the amount of data drawn before the buffer to prevent this issue.

How to prevent buffer overflow:

* Avoid C/C++ - These languages allow the programmer to access and assign dynamic memory, doing so can result in buffer overflow. Other languages such as Python, does not allow direct memory control.
* Using secure STL methods when impossible to not use C or C++ - using secure string methods such as strcopy\_s and strcat\_s allows ways to copy a string into a buffer and add the contents of one buffer with the contents of another buffer. This will help negate the effects of buffer overflow (Team, 2017)

Data Vulnerabilities

Any program that is being hosted or connected to a database has a high likelihood of containing sensitive information regarding employees, or customers (Synopsys, 2019). Common data thefts steal customers banking information or personal details, commonly, while and inconvenience to the customer can be resolved by their banks issuing new bank cards etc. Sometimes information stolen can lead to individual identify theft, which can affect the livelihood of the person affected. Therefore, it is the job of the program developer to ensure that methods have been put into place to avoid data theft.

Vulnerabilities with data will usually lie with unencrypted data, meaning that an attacker can easily read the contents. However, even with encrypted data the attacker can steal encryption keys or attempt to breach the encryption. Breaching encryption is relatively easy if the methods used are not sophisticated.

Ways to mitigate data vulnerabilities:

* Strong and secret encryption methods, ideally, one way encryption such as RSA (Lake, 2018). This means that any attacker can only see the public key and would need access to the private keys before being able to read the data. With the company keeping their encryption methods and especially their private keys hidden this will heavily mitigate against data vulnerabilities.

Logging Vulnerabilities

A log is a common function of any program, and serves the purpose of keeping track of any changes to a program. These logs are normally done automatically and usually writing to a file without the use of encryption.

However, attack can take advantage of unprotected log files in an attack known as a “ Log Injection “ (OWASP Foundation, n.d.), this form of attacks takes advantage of an unprotected log file or a program that can take log files from unknown sources. Such attacks can generate new events on a system that never happened, and can even inject malicious code into a system.

Very simple steps can be taken to prevent the occurrence or ability of log injection attacks, these include: Preforming input validation on any data coming from any log file, doing so will render malicious code inert. Having all data being written to a log file to be encrypted, doing so will prevent any malicious actor being able to observe what a program is currently doing. (Affinity, n.d.)

Using Components with Know Vulnerabilities

Using components with known vulnerabilities in pieces that have unchecked for security causes a small backdoor to compromise for accessing sensitive area inside the software.

Most software using pre-existing components (framework) instead of being coded entirely from scratch. Often time, developer of components that parts not checked for secure before use. This ending up causes a problem for companies or individuals who use pre-written frameworks to make more appealing and interactive programs without awareness that they may introduce a vulnerability to the application.

How to avoid this threat:

Awareness is the best defence against the risks from known vulnerabilities. thehackerish suggesting that using an automated tool to find common vulnerabilities and exposures will discover the threat (thehackerish, 2020).

It is removing the threat and unnecessary files or components that are not using any part of this application.

Broken Authentication

Broken authentication is when authentication poorly designed, making threat agent easier to access. This was used in many worst data breach cases (Martin G.). Threat agent uses phishing attack to capture the valuable information that allows them to access to the system.

How to prevent this:

Using a strong case of authentication, most threat agent use software to crack down the most common password to allow them to log in while agent out around looking for a valuable person to phishing with. If the common password does not match, then it generates a new password. See if it fit. Using long characters and symbol will make the threat agent last millions of years to crack it.

Bibliography

Beaver, K., n.d. *Commonly Hacked Ports - dummies*. [online] dummies. Available at: <https://www.dummies.com/programming/networking/commonly-hacked-ports/> [Accessed 20 February 2021].

Gelnaw, A., 2021. *Open Port Vulnerabilities: What's the Big Deal?*. [online] BitSight. Available at: <https://www.bitsight.com/blog/open-port-vulnerabilities-whats-the-big-deal> [Accessed 20 February 2021].

W3schools.com. n.d. *SQL Injection*. [online] Available at: <https://www.w3schools.com/sql/sql\_injection.asp> [Accessed 20 February 2021].

Academy, W. and injection, S., 2021. *What is SQL Injection? Tutorial & Examples | Web Security Academy*. [online] Portswigger.net. Available at: <https://portswigger.net/web-security/sql-injection> [Accessed 20 February 2021].

Rubiens, P., 2018. *SQL Injection Prevention | How to Prevent an SQL Attack*. [online] eSecurityPlanet. Available at: <https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks/> [Accessed 20 February 2021].

Mohanty, S., 2018. *5 Important Software Vulnerabilities - DZone Security*. [online] dzone.com. Available at: <https://dzone.com/articles/5-important-software-vulnerability-and-attacks-tha> [Accessed 20 February 2021].

Team, S., 2019. *Top 10 software vulnerability list for 2019 | Synopsys*. [online] Software Integrity Blog. Available at: <https://www.synopsys.com/blogs/software-security/top-10-software-vulnerability-list-2019/> [Accessed 26 February 2021].

Cwe.mitre.org. n.d. *CWE - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (4.3)*. [online] Available at: <https://cwe.mitre.org/data/definitions/120.html> [Accessed 26 February 2021].

Team, S., 2017. *How to detect, prevent, and mitigate buffer overflow attacks | Synopsys*. [online] Software Integrity Blog. Available at: <https://www.synopsys.com/blogs/software-security/detect-prevent-and-mitigate-buffer-overflow-attacks/> [Accessed 26 February 2021].

Lake, J., 2018. *What is RSA encryption and how does it work? | Comparitech*. [online] Comparitech. Available at: <https://www.comparitech.com/blog/information-security/rsa-encryption/> [Accessed 26 February 2021].

Owasp.org. n.d. *Log Injection Software Attack | OWASP Foundation*. [online] Available at: <https://owasp.org/www-community/attacks/Log\_Injection> [Accessed 26 February 2021].

Find and Fix Your Vulnerabilities. Discretely. Now. n.d. *How To Prevent Log Injection*. [online] Available at: <https://affinity-it-security.com/how-to-prevent-log-injection/> [Accessed 26 February 2021].

thehackerish. 2020. *Using components with known vulnerabilities - thehackerish*. [online] Available at: <https://thehackerish.com/using-components-with-known-vulnerabilities/> [Accessed 19 February 2021].

Martin, G. 2020. *The 11 Biggest Data Breaches of 2020 (So Far).* [online] available at <https://auth0.com/blog/the-11-biggest-data-breaches-of-2020-so-far/> [access 21 February 2021]