# Vulnerability Report – Group 1, Team 4

###### 18075398 - Connor Bosanko

Tainted Data

Tainted data is a security risk which allows a hacker to trigger security vulnerabilities or potentially crash the program. This can happen when a program is reading from a risky source such as a file, user input or any other possible weak points in the dataflow. When an attacker is able to abuse this floor, they are able to execute unauthorized code and potentially remote control the program which is of course a huge security risk.

If tainted data makes it into the code from a lack of taint checking it is able to arbitrarily execute code by changing an instruction pointer or program pointer to point to piece of code in memory (normally injected by the attacker). Tainted data could also change something in memory which could cause a crash later like changing an integer to a string or causing an access violation by changing a pointer somewhere it does not have access to. There are multiple issues with allowing this data into the system so it is important to taint check any information coming into the system.

Taint checking is a preventive method of not allowing this data to enter the system by checking the input for signs of potential problems and then quarantining it (making untainted and ready for use). There Many different implementations of this exists however the underline factor is to understand the language and your code and notice the vulnerabilities and add special preventive measure by checking if a given input like a file, sensor, environment variable. etc. has signs taint and remove the risky code or ignore it. Taint analysis tools like can also be used to further investigate source code and find these attack surfaces so you can prevent them. Some languages have this feature built in such as Perl and asp not running a script if it senses that inputted data could be tainted.

This is a giant possible breach in a program so understanding your language and finding these breaches so come first when planning to fix your programs Vulnerabilities.

Firmware Vulnerability

Firmware is a type of computer software allowing for control over specific hardware. In computers, firmware is intended to provide functionality of the hardware to the software(kernel). Because firmware works with the hardware and on legacy systems is not always up to date; there are plethora of vulnerabilities, however I will talk about them in general; what they can do and how to avoid being a victim of one.

Because of the nature of firmware attackers could be able to directly exploit things like USB ports, gain access information from your storage directly (changing, deleting and possible editing encrypting data) or even totally brick hardware. In the past this has rarely been seen because gaining access and then finding a vulnerability of these devices is difficult and probably extremely difficult to perform outside of a lab with direct access to open hardware.

Some real example of real firmware vulnerability such as Windows Error Reporting CVE-2019-0863 allows for admin level arbitrary code execution and Intel SA-00191 allows for things like DDOS attacks and release of personal information

Some more recent devices use windows or software to automatically update firmware so as long you install hardware specific software and keep windows up to date you should not be susceptible to these types of attacks.

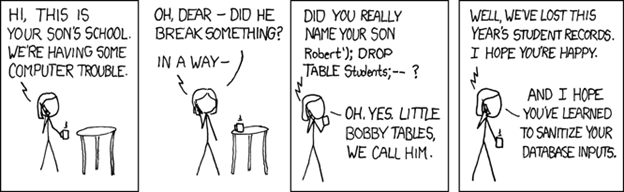
17046130 - Jack Clements

SQL Injection

Sql injection is a web security vulnerability that allows an attacker to interfere with queries that an application makes to it’s database. (Portswigger 2019) In modern day programming it’s very easy to prevent this type of attack however many newer developers still struggle to implement that protection. SQL injection works by modifying a string passed straight into an sql query to run sql code. This works because sometimes developers will use user input as part of a sql query for example: an application where the user will input an employee id then receive the corresponding data from a sql table. The program code for the query (simplified might look as such): data[] = sql.Query(“SELECT \* FROM Employees WHERE employeeID = ‘<user Input>’”);

On first glance this might look harmless as it just asks the user for an input then replaces <user input> with said input however if the user was to use a sql injection technique by ending the current statement and beginning their own. Let’s assume the user input is <’; DROP TABLE Employees;> This would return the current query by using the ‘; and then executing the user’s own code telling the database to drop the employees table. This can be used doing any type of sql command and can even display information back using more advanced means in conjunction with xss (cross site scripting) for website and utilising data displays in software applications.

We can easily protect against this using input sanitization. This simply means that a given user input regardless of any keywords or symbols will be treated entirely as a string stopping the execution of unwanted code.



13069284 – Abdulla Hassan

Unprotected Storage of Credentials

Username and password combination is the most widely used knowledge factor used in verifying authenticity of an entity. Any other entity who can gain knowledge on these credentials can pretend and pass the authentication checkpoints using these. This vulnerability involves greater risks of losing financially, permanent loss of access to sensitive and critical information, and damage to public image. To make things worse, exploitation of this vulnerability can go unnoticed if the attacker keeps on passively monitoring the accessed assets than performing any modifications or actively taking control of the assets.

With this level of criticality of user credentials, they need to be stored with utmost security when in virtual and physical storages. When considering the user base of a large enterprise, these credentials will be stored in LDAP(Lightweight Directory Access Protocol) implementations which provide features to securely store the credentials using salted encryption and hashing techniques or in a RDBMS using the same techniques. When a system is considered they can have passwords stored in configuration files[1]. When considering the virtual storages such as keeping the passwords in memory for the duration of execution, it also needs to be executed in a protected way to avoid exposing the credentials in plaintext to any other party monitoring the memory footprint. Adversaries exploit various other vulnerabilities such as SQL or LDAP injections and directory traversal attacks just to get hold of the credentials. When the credentials are protected using techniques such as salted encryption, even if the attackers get hold of the credentials they can’t readily make use of those to gain access.

The reputations of giant organizations have been in risk when this type of a vulnerability is exposed, Facebook being an example very recently in 2019[2][3]. In order to preserve non-repudiation properties of a system, not even an administrator should be able to have knowledge of a user’s credentials. When the passwords are stored in plaintext, it’s vulnerable to be misused by the internal administrators who have access to the data, which was the concern with Facebook, though an outside attacker was not able to get hold of these plaintext passwords.

As a best practice the systems are required to store these user passwords in salted encrypted mode to have the ultimate security levels as this vulnerability can open doors to a highly sensitive set of information leading enterprises totally shut down. Just depending on encryption is not very encouraged as an administrator will still be able to guess the password of a user if he/she can generate the same encrypted output for a plaintext password which can be avoided when a salt is used along with the password to perform the encryption.

XSS (Cross Site Scripting)

When a user access a particular domain in the web browser, it is supposed to provide content only from that trusted domain. Web browsers should not be accessing resources from other domains and submit to the users of the current domain as the it’s is unable to provide any guarantees on the other domains. To enforce this, modern browsers alerts the users when they detect the user is taken away from the current domain. However, with cross site scripting attacks, adversaries try to inject malicious scripts into the browser, so that the content will be run on the user’s trusted domain, without being caught as from an external domain. This misleads the browser and user to believe the content is from the trusted domain they intend to be using, but is not.

The severity of this attack depends on the next actions planned by the attacker which can be stealing the domain cookies to hijack the session, stealing tokens to conduct a CSRF(Cross Site Request Forgery) attack or launching a phishing attack disguising the user to be in the trusted domain while not and make them share some sensitive information. XSS attacks are triggered through user input fields, where the web application takes user inputs without validation and then submit it to the browser in dynamic contents, which make the malicious script to run on the browser within the trusted domain. XSS attacks can be categorized into 3 types[1][2] namely reflected XSS, stored XSS and DOM based XSS. In reflected XSS, the injected malicious script goes to the vulnerable web site’s server and then gets reflected back to the user’ web browser. The malicious data is not persisted in this case as opposed to the stored XSS attacks where malicious script is stored in the server’s database or logs. Then a separate request is triggered to make the server retrieve this malicious data from the database and submit it to the browser. In both of the above cases, the server injects the malicious script to the browser, but in the third type of DOM based XSS, the attack happens totally on the client side. This attack happens when the server provides a script that performs the sanity checks on the user inputs and submits the processed output to the web browser itself, so that malicious inputs provided in the client side will go through the sanity check and submitted back to the browser and get executed.

From the well known website, in 2010 Facebook was identified with an XSS vulnerability in one of their forms used by the developers as in [3]. There are several techniques used as best practices in web application development to mitigate this vulnerability. As the main method of injecting the malicious code is via user inputs, having proper user input validations to limit what can be submitted can reduce the risk of this vulnerability to a greater extent. Then as a second level protection output can also be validated and encoding to make sure it doesn’t have any malicious content and encoded so that it cannot automatically run on the browser. Modern browsers also provided special mechanisms that need to be enabled using headers, to mitigate this attack. A combination of these techniques can be used considering the level of criticality of the vulnerability in the domain and compromises to be made in user experience.

18134645 – Dafyd Phillips

Access Token Manipulation (ATM)

What – When a user attempts to manipulate the access tokens used by Windows to determine ownership of a currently running process. This results in a process appearing as though it belongs to a user other than the one who started it, taking on the security context associated with the new token.

How – Tokens can be manipulated one of three ways;

1. Duplicating an existing token to allow impersonation of a user’s security context or to assign the impersonated token to a thread.
2. Using a duplicate token to create a new process under a user’s security context.
3. Create a logon session to receive a copy of the session’s token, allowing the attacker to assign the token to a thread

(these methods require Administrative access to execute)

Why – Token manipulation is commonly used to elevate an attacker’s security context from admin to SYSTEM, giving them greater access to a system/network and its processes.

Mitigation & Detection:

* Since admin-level access is required to perform ATM, limiting users/user groups to the least privileges they require (such that they are unable to create/replace tokens) will reduce the risk of a user making use of this technique.
* Auditing command-line activity can help detect attacks from an attacker using standard command-line shell.
* Careful analysis of user network activity is the only way of detecting direct manipulation.

Buffer Overflow

What – If a buffer contains more data than its fixed-length will allow, that data will overflow into adjacent storage. This can corrupt or even overwrite data stored in that space. A system will typically crash as a result of buffer overflow, but it may also create entry points for cyberattacks.

How – if a function doesn’t check or otherwise doesn’t know the buffer size or size of an input, an attacker may simply input more data than what was intended through string inputs. The get() method from C language is an example of such a method, with no safeguards or validation for input.

Why – data may overflow into parts of memory that contain certain variables, eg a boolean used to decide whether a user has entered the correct username/password, giving attackers access to high-permission user profiles. It may also be used to simply corrupt important data or crash a system entirely.

Mitigation & Detection

* The best detection is a keen eye – check any part of the code where buffers are used carefully.
* Implementing bound checking to user input will help in detecting and preventing overflows
* Using languages that do not allow these overflows (ie Python, Java, .NET) will reduce or even remove the need to perform checks in the first place

**Sources**

Wikiwand. (2020). Taint checking - Wikiwand. [ONLINE] Available at: <https://www.wikiwand.com/en/Taint_checking>. [Accessed 7 February 2020].

Laurel Stewart. (2020). What is Taint Checking?. [ONLINE] Available at: <https://blogs.grammatech.com/what-is-taint-checking>. [Accessed 7 February 2020].

Taint::Runtime - Runtime enable taint checking - metacpan.org. (2020). Taint::Runtime - Runtime enable taint checking - metacpan.org. [ONLINE] Available at: <https://metacpan.org/pod/release/RHANDOM/Taint-Runtime-0.03/lib/Taint/Runtime.pm>. [Accessed 7 February 2020].

Wikiwand. (2020). Arbitrary code execution - Wikiwand. [ONLINE] Available at: <https://www.wikiwand.com/en/Arbitrary_code_execution>. [Accessed 7 February 2020].

perlsec - perldoc.perl.org. (2020). perlsec - perldoc.perl.org. [ONLINE] Available at: <https://perldoc.perl.org/perlsec.html>. [Accessed 7 February 2020].

. (2020). . [ONLINE] Available at: <https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-0863>. [Accessed 7 February 2020].

Intel SA-00191 Firmware Vulnerabilities in NetApp Products | NetApp Product Security. (2020). Intel SA-00191 Firmware Vulnerabilities in NetApp Products | NetApp Product Security. [ONLINE] Available at: <https://security.netapp.com/advisory/ntap-20190318-0002/>. [Accessed 7 February 2020].

Portswigger (2019). *What is Sql Injection?.* [online]. Available at: <https://portswigger.net/web-security/sql-injection> [Accessed 7th January 2020]

[1] - Cwe.mitre.org. (2020). *CWE - CWE-256: Unprotected Storage of Credentials (3.4.1)*. [online] Available at: <https://cwe.mitre.org/data/definitions/256.html> [Accessed 7 Feb. 2020].

[2] - Canahuati, P. (2020). *Keeping Passwords Secure - About Facebook*. [online] About Facebook. Available at: <https://about.fb.com/news/2019/03/keeping-passwords-secure/> [Accessed 7 Feb. 2020].

[3] - Bauman, E., Lu, Y. and Lin, Z. (2020). *Half a Century of Practice: Who is Still Storing Plaintext Passwords?*. [online] Web.cse.ohio-state.edu. Available at: <https://web.cse.ohio-state.edu/~lin.3021/file/ISPEC15.pdf> [Accessed 7 Feb. 2020].

[4] - Cwe.mitre.org. (2020). *CWE - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (3.4.1)*. [online] Available at: <https://cwe.mitre.org/data/definitions/79.html> [Accessed 7 Feb. 2020].

[5] - Owasp.org. (2020). *Cross Site Scripting (XSS) | OWASP*. [online] Available at: <https://owasp.org/www-community/attacks/xss/> [Accessed 7 Feb. 2020].

[6] - XSSed.com. (2020). *Critical Facebook XSS bugs could be used to hijack accounts*. [online] Available at: <http://www.xssed.com/news/115/Critical_Facebook_XSS_bugs_could_be_used_to_hijack_accounts/> [Accessed 7 Feb. 2020].

Google Security Research (2018). *Microsoft Windows - Token Process Trust SID Access Check Bypass Privilege Escalation.* [online]. Available at: <https://www.exploit-db.com/exploits/44630> [Accessed January 21 2020]

Mitre (2018). *Access Token Manipulation.* [online]. Available at: [https://attack.mitre.org/techniques/T1134/](hjttps://attack.mitre.org/techniques/T1134/) [Accessed January 28 2020]

Bright, P. (2015). *How security flaws work: The buffer overflow*. [online]. Available at: <https://arstechnica.com/information-technology/2015/08/how-security-flaws-work-the-buffer-overflow/> [Accessed February 4 2020]

Kerestan, B. (2017). *How to Detect, Prevent, and Mitigate Buffer Overflow Attacks.* [online]. Available at: <https://dzone.com/articles/how-to-detect-prevent-and-mitigate-buffer-overflow> [Accessed 11 February 2020]