Software Security

Authors: Behnam Nikkhah, Brandon Yip, Clinton Cabiles, Guilherme de Oliveira Santos, Lester Dela Vega, Kevin Quan, Kyle Fruson

Abstract – Because of many security threats that threaten your software and privacy, software security is an important concept to learn. There are existing methods to enforce security which you can use to bolster your defenses. Whether its time or money, knowing more about protecting your software will save you from losses in the future.

Keywords—software; security; threats; life-cycle; attack;

I. INTRODUCTION

Software security is the idea and practice of building or engineering software to be secure and function properly when under malicious attacks. In the past, software was coded without great consideration to the security of the software. Attacks started increasing at some point, and so the concern to protect software from attackers also increased. At first, the protection to these threats were small fixes. Then they elevated to anti-virus, firewalls, anti-spyware, and many other forms of protection against attackers. Thus today we are now, more than ever, concerned about the confidentiality, availability, integrity and reliability of the software, and a new goal arises for software engineers, computer scientists and others to aim for.

The goal is to prevent attackers, whether they be amateurs or professionals, from exploiting vulnerabilities in our software.

II. WHAT IS SECURITY?

Software security addresses three aspects of any software and computer-related systems: integrity, confidentiality, and availability.

> Integrity: When one asks for data, and receives the right data and nothing different.
>
> Confidentiality: Access to the software is limited to certain authorized parties.
>
> Availability: The software can be accessed when one wants to access it.

Refer to Figure 1 for diagram [1].

These are the primary goals that are considered the utmost important to any secure software. This is because it is said that an asset is secure if these three properties are satisfied.

> Asset: A resource of value such as data, hardware or software [2].

Software can never be 100% secure. There are many vulnerabilities such as design flaws and implementation bugs in software.

> Vulnerabilities: Weaknesses in an asset that an attacker can exploit in order to cause loss or harm [3].

Because these vulnerabilities exist, attackers may exploit them to their advantage to threaten the security of your software. There are many different approaches the attackers can choose to do this.

> Attack: An action that exploits a vulnerability [3].

But, there are ways that can prevent and control these attacks from penetrating your defence. Certain security measures can be taken to prevent this and control vulnerabilities so that attackers cannot use them at their disposal and harm you in any way.

> Control: Removing or reducing a vulnerability by controlling it to prevent an attack or block a threat [3].

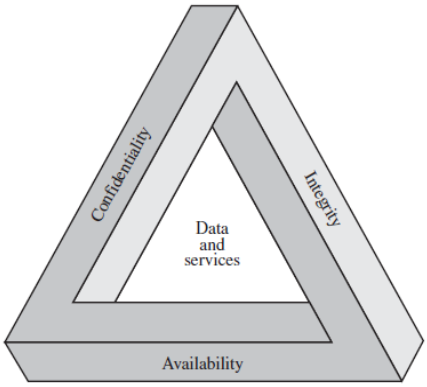

Fig. 1 - The Security Requirements Triad [4].

A preemptive measure to minimize the amount of vulnerabilities in software is starting in the software

development process, where we can focus on implementation flaws as well as create tests that that can test the security of the software.

There are also security tools that are provided to the general public to protect their computers and software. These are great to minimize the chances of the attacker's ability to attack your assets.

Most software access the Internet, in which case, network security is also an important concept that needs to be ensured. More details on this topic will be described in a later section.

There are many current events where attackers have successfully attacked assets resulting in losses for the affected party. Some examples will be given in a later section.

Finally, many types of security threats exist in the world today. The next section will give some insight on some of the types of security threats you may encounter.

III. Types of Security Threats

There are many types of security threats that exist in software. Common security categories include: input validation, authentication, auditing (logging), and exception management [5].

Input validation is the correct input which uses validation rules that check for correctness and meaningful entry data. For example, regular expressions make sure that the data entered is ordered, confined and correct. However, security threats such as injections and memory overflows cause confidentiality risks. One common type of injection threat is the *SQL injection*. For websites that use the DBMS *SQL*, attackers can enter malicious database queries for execution. This is mostly used for unauthorized account infiltration in websites that offer a log-in feature. Less subtle than the injection method, memory overflows attempt to overrun a system memory's capacity in order to alter the way it operates. Buffer overflows is a common example in which an attacker attempts to overrun a buffer's capacity to alter a program's behavior.

Authentication is accessing an account or some authenticated channel with the correct input validation. These types of threats are much more common in practice but normally infeasible. Most of these attacks deal with a (very) large set of data to be used for authentication testing. The most common of these threats include the brute force attack and the dictionary attack. The brute force attack, hence the name, scans through every single combination given its alphabet and string length. Although brute force attack is theoretically impossible to fail, the algorithm becomes extremely futile when string lengths increase; especially long passwords, for instance. Instead of going through every single combination of a string, the dictionary attack tries to lighten the load and search for pre-determined strings to test. The slight improvement comes with a cost, however, involving very large file sizes or strings that never match the target's authentication fields.

Auditing and logging requires the attacker to be hidden and hide their true identity while performing attacks. This category of threats is mostly known as *spoofing*. IP spoofing and *phishing* are two of such examples of concealing identity. Since the IP is a unique network identifier, this poses a risk for the attacker performing an attack. Therefore, to overcome this obstacle, the attacker comes up with a fake, random IP (i.e. 1.2.3.4) to divert attention to an IP that does not direct attention to the attacker; thus, the attacker can observe their attack while being hidden from public network view. A more involved maneuver is phishing or URL spoofing. Instead of going to an intended website, an attacker "phishes" a victim to enter a spurious website that poses to be the original. Information entered into a phished site can cause victims to enter personal information to be viewed by an information thief. For the unknowing victim, this poses a great risk since information is presented to the attacker in plain text.

Finally, exception management is mostly an attack on the availability or readiness of a software system. This attack usually involves an attacker populating large amounts of systems or amplifying a request into overwhelming amount of responses. Two such attacks include the DDoS attack and the DNS TCP amplification attack. The DDoS attack starts with one "zombified" system that is under the control of the attacker – a bot. From here, the bot searches for other prey systems to be part of a larger bot swarm called the *botnet*. After creating a sufficiently large botnet, the attacker then attacks a particular target with extensive amounts of requests causing it to slow or stop responding altogether. Somewhat analogous to the botnet and spoofing is the DNS amplification attack. The attacker performs two tasks to accomplish this amplification. First, the attacker spoofs the IP address to the victim's IP address. Then, the attacker finds an internet domain with many DNS records (i.e. 1.site.com, 2.site.com, 3.site.com, etc.). Henceforth, all the DNS replies avert their attention to the spoofed IP address (the victim server) which also causes a denial of service from that network's edge.

In Section IV, we will explore topics of some software security topics that try to prevent some of the attacks listed above.

IV. NETWORK SECURITY

With the increasing reliance on the Internet for daily activities, also comes the parallel increase of the possibility of an attack, or opening within network security. These threats to network security can be attributed to factors such as anonymity, complexity of software systems and also file sharing. These threats can be divided into main target areas such as authentication failures, programming flaws, confidentiality, integrity and availability [6].

Learning different methods of attacks would help in the prevention of them. For example, in the case of authentication, the authentication target is vital to the protection of a system. In this area, key credentials such as password or username can be guessed, or spoofed in a way where a hosted client gives false credentials and acts as an authorized user. This is typically done by IP address spoofing, or the interception of a connection and acting as the receiving client. While it is not possible to completely prevent credential guessing, you can increase the number of possibilities thereby making it extremely inefficient for a user to guess another password.

Prevention of these threats can be acquitted to architectural design choices, and encryption. However, in general there are 3 main steps to take in order to increase the security of a system [7]:

1. Analyze the components and how they interact with each other.
2. Consider possible damage to the main target areas defined above.
3. Brainstorm possible attacks on your system based on the possible targets.

Strong architectural design choices can be made in order to close the accessibility into these target areas. One possible design option is to segment your program. Having one centralized system would increase the possibility of failures due to the reliance of every individual component of the system; if one area were to fail, the others would subsequently follow. Another possible design decision to consider is to split functionality between units. By doing so, if one component were to fail, the other component could take over as the primary element. This creates further possibility for security. One aspect that both these design choices have in common is the ability to continue even after a failure. If you can locate an area of your system in which one failure would cause the system to crash (known as a single point of failure), you can then work to create a solution to stop it.

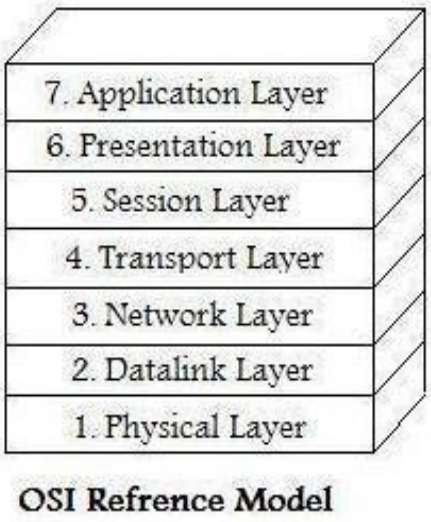

Fig. 2 - OSI Reference Model [9]

Following a good architecture, a powerful encryption method will also protect the system from intrusion, and spoofs. A strong encryption method integrated onto a fail-safe system architecture will create a greater wall in security of a network-based system. One possible integration of an encryption method is link encryption which deals primarily on the data link layer or physical layer of the OSI model[Fig.2]. This encryption method ensures that security is present within the links of a system, however since messages are decoded after getting to a link a security hole may be exploited and abused.[8]

Subsequently, since the encryption begins at the lower levels of model, it follows that the message is still unencrypted within the higher layers of the model. Another possible encryption method is "end-to-end" encryption. This deals with encryption from one end of a link to another. This encryption method begins within the user layer of the OSI model, and bypasses the unencryption at every link leading to the end links, making this encryption method stronger in certain scenarios. However since it relies on the end links, the possible encryption keys that this method uses will increase faster than when using link encryption. In conclusion, link encryption is faster to integrate and uses less encryption keys while end-to-end encryption is more versatile, and can be integrated in the application. While encryption helps as a countermeasure towards network security, other more general measures to security can be seen in the next section.

V. SECURITY MEASURES

In this section, we will discuss some of the general security measures to defend against the threats described in Section III.

A. General Overview of Security Measures

Due to the security vulnerabilities described in Section III, there is a need for methods for software developers to protect themselves against these attacks. There are several methods to defend against these threats such as cryptography, software controls, hardware controls, physical controls, and policies and procedures. The purpose of these methods of defence are to preserve the confidentiality, integrity, and availability as described in Section II. These types of controls can defend a software system in varying ways like preventing or mitigating an attack. Others can identify that an attack has occurred and the system's security has been compromised by detecting a breach real-time or after the damage has been done. The examples above can be generalized into five overlying methods to protect a system against malicious attacks. [10] We can seek to:

- Prevent it, by blocking the attack or closing the vulnerability
- Deter it, by making the attack harder but not impossible
- Deflect it, by making another target more attractive (or this one less so)
- Detect it, either as it happens or some time after the fact.
- Recover from its effects.

In general, we will want to combine many of these methods to defend against the same threat. A multi-layered defence is much safer than any one of these. Also, if a breach does occur even with all these defence mechanisms in place, there should be response procedures in place so the system can recover and make sure it does not happen again.

B. Examples of Security Measures

1) Prevention

The most effective way to protect a program from threats is to make sure that they cannot occur in the first place. Buffer overflows are problems that occur when you try to put too much data into a buffer that does not have the capacity to contain it. [11]

For example, in C you define a char array:

char buff[10];

then you try to assign a value:

buff[10] = 'a'

But the boundary of the buffer is "buff[9]" so data gets written beyond the boundary of the buffer.

This may just seem like a bad programming practice, but in the hands of an attacker, buffer overflows can be a major vulnerability. For example, when asking for a password, you can enter a value that will overflow the buffer that is getting input from the user, overwrite the memory location that saves the password, and gain access even with a wrong password.

To prevent these types of attacks, you must perform bounds checking to make sure that invalid inputs are not allowed. Checking that a user has inputted a valid request is called mediation. Another example of an attack that takes advantage of incomplete mediation are SQL injections. An attacker can input a malicious command in an input field and query the database and gain access to confidential information. Therefore, you must always mediate the inputs that a user can give to prevent intrusions. The following is an example of an SQL injection based on 1=1 is always true [12].

"A user enters this into an input field for a UserID: "105 or 1=1"

*The result from the server is: SELECT * FROM Users WHERE UserId = 105 or 1=1"*

The SQL statement shown above is valid and it will return all rows from the table "Users," since WHERE 1=1 is always true. In order to prevent this type of attack from succeeding, developers can blacklist certain words or characters from being entered. However, words like DELETE and DROP are also used in common language so sometimes blacklisting these is not a very good idea. The most effective way to protect an input field is to use SQL parameters. SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

For example,

"txtUserId = getRequestString("UserId");

*txtSQL = "SELECT * FROM Users WHERE UserId = @0";*

db.Execute(txtSQL,txtUserId);"

This way, the system is safe if the user tries to enter 1=1.

2) Deterrence

Another method of defence is to deter an intrusion attempt by making it more difficult or more time consuming for the attacker. An example of this is to use cryptography. You can protect your data by encrypting your data to make it unreadable to an attacker. If an attacker wants to access this data, they will have to invest more time and computing resources to try to decrypt your encrypted data. This is especially useful against brute-force attacks because the computing time is exponentially increased if you use a higher bit encryption.

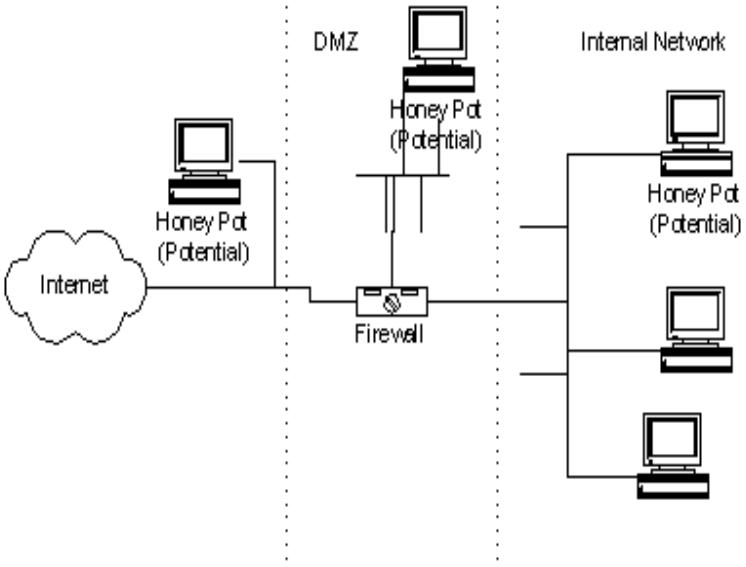

Fig. 3 - Network Honeypot Diagram

3) Deflection

A more indirect way of protecting your software from intrusions is to make yourself less appealing to attack or make something else more attractive to attack. An example of this is by making a honeypot**.** [13] Honey Pot Systems are decoy servers or systems setup to gather information regarding an attacker or intruder into your system. Honeypots are traps that are meant to detect and deflect attacks against systems. They are dummy systems that are purposely easy to intrude so that the owners of the honeypot can see information about the intrusion like the identity of the attacker.

4) Detection

This is not exactly a method of defence because the attack has already occurred, but it is still useful to have, just in case any of your previous methods failed. Detection is finding out that a breach has occurred so that the system can recover from it. You can manually program handlers for unexpected uses of the system or use and IDS (Intrusion Detection System).

In short, these are the general measures that can be taken to ensure secure software.

VI. SECURITY TOOLS

Many tools have been created to assess the source code behind applications. Ideally, analysis is done throughout the software development cycle.

These tools can be divided into static and dynamic testing tools.

A. Static Analysis Tools

Static analysis tools perform analysis of computer software on source code without the need to execute it. It may also perform analysis on a representation of the program instead, such as an abstract syntax tree, a dependence graph, a call graph, or another type of representation.[14] Sometimes called source code analyzers, static analysis tools promise to identify common programming bugs in an automated (or semi-automated) fashion. There are many benefits to using static analysis tools:

- Scalability - static analysis tools allow code to be reviewed at a speed eclipsing human reading ability. It greatly aids with manual code reviews.
- Superior knowledge base - such tools know more about security than humans.
- Ability to test unusual circumstances - static analysis tools possess abilities to go into deep areas of code without needing to run the code.

However, static analysis are not a magic bullet solution. Because static analysis look for sets of patterns or rules in the code, it won't find any problems if a pattern to find a particular issue has not been implemented yet. In addition, it still requires someone to evaluate the results of the analysis -- it cannot determine for you which issues must be fixed immediately or not. It also cannot critique software design.

Most importantly, a particular issue with static analysis tools are in regards to false positives and false negatives. False positives are reports that erroneously claim non-existent bugs exist, and false negatives are issues in which a bug goes unreported by the tool. False negatives are particularly nefarious because they produce a false sense of security.

Basically, static analysis problems are mathematically undecidable in the worst case [15]. Thus, static analysis tools must approximate their results, finding a balance between issuing false positives and false negatives.

In the past, simplistic tools such as Cigital's ITS4, RATS, and Flawfinder perform simple lexical analysis of the code: they scan through files and find matches in syntax based on guidelines that indicated possible security flaws. Because of its simplistic approach, these kind of tools ended up flagging a lot of false positives due to a lack of consideration for the semantics of the code.

Today, there are more sophisticated static analysis tools, such as Coverity, Fortify, and Prexis, where each have an extensive set of rules far superior to the simple tools described above. Academic research continues to be done on improving the reliability of these tools.

B. Dynamic Analysis Tools

Dynamic program analysis involves executing the program in real time and observing it. It may implement fuzzing (a black-box approach to software testing, in which security flaws and bugs are found by submitting random, malformed input data into a system) to uncover flaws. Also known as dynamic application security testing (or DAST), dynamic analysis tools like Intel Inspector try to come up with properties that hold true for multiple executions.

Both kinds of tools are sophisticated aids that greatly help developers protect their software from implementation bugs. Keeping code as secure as possible will help avoid costly consequences later on in the software development cycle.

VII. SECURITY IN THE SOFTWARE DEVELOPMENT LIFE CYCLE

Software security should be considered during all the stages of the software development life cycle. The security aspect of software includes security mechanisms, such as user authentication, and more high level aspects as design of security patterns. Generally, security problems are commonly related to code issues because a feature was poorly documented or implemented. It is also very common to have problems with integration. A well-known example of security problem on web applications is the SQL Injection, which changes the code behavior by running malicious inputs. Other examples of common web-based security attacks are Cross-site scripting and denial of service. Gary McGraw [16] suggests a number of best practices applied to the software artifacts produced during the development life cycle.

A. Requirements and use cases

During the software conception, security issues should already be taken into account; evident security problems and security measures can be planned at this point. During the requirements stage, use cases are created to explain system features. In the same way, abuse cases can be created to describe the system behavior under attack, what should be protected, from whom it should be protect, and for how long.

B. Design

During Design and Architectural stages, security principles should be applied to create a reliable software architecture and design. A good part of application security design is about deciding what technologies to use for identification and authentication, how to specify access control, and how to manage authorizations [17].

C. Test plans

Testing is a very important stage of the software development, as well as for the software security. At this point, tests should run against security functionality described in the abuse cases, as well as risk-based testing based on attack patterns. "Representing vulnerabilities as patterns can be useful for illuminating a problem that can occur in multiple and different software applications. Generic solutions to the problems represented by attack patterns should be abstract enough to encapsulate solutions for the same problem in many contexts, as is done with design patterns" [18].

D. Code

During the implementation stage the focus should be on the code flaws. The book *19 Deadly Sins of Software Security: Programming Flaws and How to fix Them* [19], explains how to solve a set of common code flaws and can be used as a guide at this point. Another resource during this stage are the security analyses tools mentioned before in Section 6-A.

E. Test results

A useful activity at this point is to run penetration tests, when only architectural risk analysis are considered. This is because during tests, the system may still have uncovered issues. "Once an application is finished, its owners subject it to penetration testing as part of the final acceptance regimen" [20]. This is a form of black-box testing performed by tools found in the market. The tools perm canned penetration tests against the final solution; failing one of these tests shows that the software still has serious problems to be corrected.

F. Field feedback

Regardless of all the effort to create reliable software, attacks will happen. As a defensive technique, it is a good idea to keep an operational team monitoring possible attacks. Whenever a security break is identified, it is time to go back into the development life cycle and fix it. In this way, it is possible to mitigate the damages caused by the attacks, and protect the system from future breaks.

VIII. CURRENT EVENTS

Software security, otherwise known as application security, are the measures taken throughout a software's life cycle to prevent gaps in the security policy of a piece of code. In order to prevent vulnerabilities that can expose a program, there are several measures that one can take. Throughout history, we have seen such security vulnerabilities exploited by young, intelligent, and motivated computer "hackers". Computer hackers are individuals who seek out and exploit these weaknesses in the computer system or software. Hackers can be motivated to perform such acts for multiple reasons. Most notable exploited security flaws of interest are the Playstation Network outage, and the Heartland Payment Systems breach, which serves as one of the most famous and costly security breaches to date.

The Playstation Network, otherwise known by its abbreviated initials PSN, is an online gaming and digital media delivery service provided by Sony Computer Entertainment for the use of their various gaming consoles, the most notable and latest being the Playstation 4. The PSN had one of the most popular security breaches to date that was handed to them by the Information Commissioner's Office (ICO). This breach would cost the company $400,000 USD, and even though this would serve as chump change to the multi-billion dollar industry, it was the third-highest fine ever handed out by the ICO [21]. Such a rare slip-up for an industry which was reflected in the penalty received by the PSN from the ICO, as such a company should have all their bases covered, to say the least. In April 2011, computer hackers known as Anonymous attacked the PSN, and over 77 million names, ,passwords, email addresses, and credit cards of those who had their information saved to their accounts were exposed [22]. This resulted in one of the largest security breaches to date. PSN was forced to shut down and rebuild from the issue, which approximately lasted 24 days [22]. Once Anonymous got wind of a weakness in Sony's security system, they were able to breach the database and implement a simple SQL injection into the code [23]. As previously mentioned in Section III, a SQL injection is a type of injection threat where a piece of malicious software is inserted into an entry field for execution, and in this case would dump the contents of the database, containing all the customers information, to the hacker. The significant flaw of the PSN came from the lack of security mechanisms in place in the internal workings of the system.

The PSN team thought their security system was impenetrable and did not take any further measures past the initial security system setup. This is akin to having one lock on the door, and having everything exposed in the house. Once Anonymous was past the initial system (the lock on the door), they had free reign over the open contents as they pleased. A lack of security in the internal workings compromised the system, so better protection methods should have been put in place. Given the size and net worth of Sony Computer Entertainment, such security measures should have been implemented to cover the internal workings to protect against the possibility of a breach.

The most famous and detrimental breaches a company has faced to date was the breach of the Heartland Payment Systems. An unknown gang composed of five men (four Russian and one Ukrainian) were able to penetrate a weakness in software security [24]. This event was recorded as the largest-ever criminal breach of credit card data. It is known that over 100 million credit cards were compromised as a result of the security breach [24]. While Heartland tried to cover all their bases, much like other big companies, they put too much emphasis on what they thought were the most critical servers and neglected the less important ones. These neglected components offered a weak spot and loophole for hackers to take advantage of. The hackers took a very similar approach to the tactics that Anonymous used on the Playstation Network, which was to gain an entry point and perform a SQL injection into the database. Again, a lack of security in the internal workings of the system ended up being very expensive. Heartland ended up taking an initial stock hit of 50% and then continued to plummet up to 77.6% [25]. The company was able to gain some ground since the stock plummet, but the stock is still down 50% today since the initial breach [25]. Due to this breach, the company also racked up 12.6 million USD in expenses related to the intrusion of the system, for things such as compensation, defending against litigation, etc. [24].

The two above examples only scratch the surface of the possible consequences one faces for the lack of security mechanisms in place.

IX. Conclusion

In conclusion, the three security topics we covered are Confidentiality, Integrity and Availability. In Section II, we discussed that with confidentiality, the user's space is secure. Integrity ensures the correctness of data. Lastly, availability allows the user to access the requested information at any time. These areas are the target of possible security threats and attacks against them can hinder the performance of a system and cause damage to both the software and social aspects as seen in Section III. However these problems are often solvable. We also discussed the various ways to prevent these threats in Section V. Such countermeasures include protection from buffer overflows and injections. We then delved into various security tools we can use to help assess our systems in Section VI. Using these tools, we are able to explore unusual or abnormal circumstances which may pose security threats to the software system. Furthermore, software systems grow to expanding networks such as the Internet for fast data transfer. There exists a large percentage of attackers in the open-net; thus, we need to ensure a secure network in all of the network layer which we have explored in Section IV. Finally, during the development life cycle of a software system, we must consider the security aspects at each step of the process. We also discussed some of the most devastating current events that occurred in the history of software security. Every application presents challenges in preventing attackers from accessing sensitive data. Thus, software security is essential to the confidentiality, integrity and availability of a complete system.

REFERENCES

[1] J. Ransome and A. Misra, "Software Security and the Development Lifecycle," in *Core Software Security: Security at the Source*, 2014, Ch.1, Sec 1.2, pp. 7

[2] C. P. Pfleeger and S. L. Pfleeger, "What Does Secure Mean?" in *Security in Computing*, Fourth Edition: Prentice Hall, 2006, Ch. 1, Sec 1.1, pp. 19

[3] C. P. Pfleeger and S. L. Pfleeger, "Attacks," in *Security in Computing*, Fourth Edition: Prentice Hall, 2006, Ch. 1, Sec 1.2, pp. 24 - 25

[4] W. Stallings and L. Brown, "Computer Security Concepts," in *Computer Security Principles and Practice* 2nd Edition: Prentice Hall, 2008, Ch. 1, Sec 1.1, pp. 11

[5] Microsoft Corporation. "Chapter 2 - Threats and Countermeasures". Improving Web Application Security: Threats and Countermeasures. Microsoft Press, © 2003.

[6] C. P. Pfleeger and S. L. Pfleeger, "Is There a Security Problem in Computing?," in *Security in Computing*, Fourth Edition: Prentice Hall, 2006, Ch. 7, Sec 7.2, pp. 527

[7] C. P. Pfleeger and S. L. Pfleeger, "Is There a Security Problem in Computing?," in *Security in Computing*, Fourth Edition: Prentice Hall, 2006, Ch. 7, Sec 7.3, pp. 530

[8] What is Network Encryption (network layer or network level encryption)?: http://searchsecurity.techtarget.com/definition/network-encryption

[9] OSI Model Diagram: http://www.buzzle.com/articles/osi-model-diagram.html

[10] C. P. Pfleeger and S. L. Pfleeger, "Is There a Security Problem in Computing?," in Security in Computing, Fourth Edition: Prentice Hall, 2006, Ch. 1, Sec 1.5, pp. 43.

[11] H. Arora. (2013, June, 4). "Buffer Overflow Attack Explained) [Article]. Available: http://www.thegeekstuff.com/2013/06/buffer-overflow/

[12] W3Schools "SQL Injection Tutorial" http://www.w3schools.com/sql/sql_injection.asp

[13] L. R. Even. (2000, July 12). "Intrusion Detection: What is a Honeypot?" [Article]. Available: http://www.sans.org/security-resources/idfaq/honeypot3.php

[14] P. Jalote et al., "Program partitioning," in [WODA '06], [2006] ©[ACM].

[15] G. McGraw "Code Review with a Tool," in *Software Security: Building Security In*, 1st ed. Boston: Addison-Wesley Professional, 2006, ch. 4, pp. 122.

[16] G. McGraw, "Software security," in *Security & Privacy*, IEEE , vol.2, no.2, pp. 80,83, Mar-Apr 2004.

[17] P. Kumar, *J2EE Security for Servlets, EJBs and Web Services: Applying Theory and Standards to Practice*. Upper Saddle River, NJ: Prentice Hall, 2003.

[18] M. Gegick, L. Williams, "On the design of more secure software-intensive systems by use of attack patterns", in *Information and Software Technology*, Volume 49, Issue 4, April 2007, pp. 381-397.

[19] D. LeBlanc et al.. *19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. New York, NY: McGraw-Hill/Osborne, 2006.

[20] B. Arkin, S. Stender, G. McGraw, "Software penetration testing,", in *Security & Privacy*, IEEE , vol.3, no.1, pp. 84,87, Jan.-Feb. 2005.

[21] S. Richmond (2011, April 26) "Millions of internet users hit by massive Sony PlayStation data theft" [Article]. Available: http://www.telegraph.co.uk/technology/news/8475728/Millions-of-internet-users-hit-by-massive-Sony-PlayStation-data-theft.html

[22] A. Sebastian. (2011, April 27) "How the PlayStation Network was Hacked" [Article]. Available: http://www.extremetech.com/gaming/84218-how-the-playstation-network-was-hacked

[23] Associated Press (2009, January 20) "Heartland Payment Systems hacked" [Article]. Available: http://www.nbcnews.com/id/28758856/ns/technology_and_science-security/t/heartland-payment-systems-hacked/

[24] B. Brenner. (2009, August 12) "Heartland CEO on Data Breach: QSAs Let Us Down" [Article]. Available: http://www.csoonline.com/article/499527/heartland-ceo-on-data-breach-qsas-let-us-down?page=3