Web Application Security:

Input Validation

David J. Bernardy
Computer Science Department
Saint John's University
Collegeville, MN
djbernardy@gmail.com

***Abstract*— Web applications have become a vital aspect of our everyday lives and the security of information they access is of an upmost importance. There will always be people who aim to exploit these applications and their users for reasons such as curiosity, destructive intentions, or financial profit. A common flaw of the average web developer is the enforcing of weak input validation, or forgetting it all together. By simply submitting malicious strings to web applications that alter the intended use, hackers can gain access to sensitive information in databases and run scripts on your machine that they embedded in web sites. To demonstrate this, I have created two web applications which perform the same functions but one of which is insecure and vulnerable to basic SQL injections and XSS attacks while the other is not. If web developers take the basic precautions I have laid out, they will drastically decrease their web application security vulnerabilities. My research brings the awareness of web applications exploitations to the typical web programmer so they can prevent these common attacks and be proactive in staying up-to-date with emerging threats and newly discovered attacks.**

I. INTRODUCTION

As the Internet has evolved to meet the needs of our fast pace world, security must adapt at the same pace to protect sensitive data. Web applications have made the world more efficient by allowing people to access information and complete transactions from just about anywhere, at just about any time. As more security critical applications, such as banking systems, governmental transaction interfaces, and e-commerce platforms are becoming directly accessible via the Internet, security has become incredibly important. These web applications can be accessed by millions of anonymous Internet users which poses a very large security threat [17]. The root cause of SQL injections and XSS attacks is weak input validation or lack thereof. Input must always be considered untrustworthy and web developers must make security their top priority and continue to remain up-to-date with the latest security practices. An important step in securing web applications of tomorrow is to raise awareness and provide tools to web site administrators and web developers today, so they can proactively audit the security of their applications. Whenever an application needs to gather information from a user or a browser, that information must be validated carefully to remove potential malicious strings. Likewise, data sent back from the web server to a browser should be filtered to make sure that exploits that an attacker has managed to sneak onto a web server aren't served back to unaware consumers who visit the site. As the popularity of the web increases and web applications become tools of ever day use, the role of web security has become of utmost importance. Recent years have shown a significant increase in the number of web application attacks; we must be prepared for these attacks and stay one step ahead of those who have harmful intentions.

II. BACKGROUND

A. *Web Application History*

The Internet as we know it has changed significantly since its conception. In its early years, it consisted of static, unchanging web pages. These pages were read-only, informational portals that provided content on things such as businesses, locations, and individuals. Scrolling down a web site page to view more content was the only user interaction. Much like banks would have no use for security guards and cameras if they closed their doors to the public, security had no place in a cyber world that did not require people to communicate with web sites. As the world changed, so did users' needs, thus, a new era was born: the age of dynamically changing web pages that interact with users in various ways. "The World Wide Web is capable of delivering a broad range of sophisticated applications [13]."

Application development soon made the shift to the World Wide Web with web applications. Previously, applications needed to be separately installed on individual computers and could only be accessed on those machines. Each time an application was updated, it needed to be redistributed to all its clients, potentially costing an organization significant time, money, and productivity. A web application is a software application that is accessed through a web browser over the Internet or through an organization's Intranet. They became popular because they are easy to update and maintain without distributing and installing software on potentially thousands of client computers. Web applications typically follow a client-server architecture. The client side is the web page that is displayed in front of a user and the server is where the application is hosted. The client interacts with the server to perform various functions. A modern day example is WellsFargo.com. The client portion would be is the web page that you see when you open the browser. You must first log into the web application to confirm you are a valid user and then Wells Fargo knows which account information to display. After you've entered your information and hit submit, the data travels from the client-side to the server-side where Wells Fargo receives and processes it. When the data arrives at Wells Fargo, it is processed and formed into database queries. If you exist in their systems, these queries will return your information to the client-side where you can now check your account balances, transfer money, or pay bills.

Another benefit to web applications is that since they do not need to be install, they don't require the use of any disk space which means clients do not need to purchase larger hard drives and can access operate numerous applications. Another important aspect web applications is they are cross-platform compatible. This means they are accessible from various web browsers within different operating systems such as Windows, Linux, and Mac OS. Since operating systems vary greatly, to develop software that can be utilized by everyone (regardless of their platform) would require a large amount of financial commitment as well as capable programmers from each operating system. Web applications are simple solutions to the problem of availability to multiple operating systems.

B. *Web Application Downsides*

Although the ability of web applications to be used by a large number of clients seems like a good thing, there are downsides. Many users may use the applications in nearly anonymous ways [19]. This is where web application security becomes an important topic. "Today, data represents a valuable asset for companies and organizations and must be protected. At times, this data can be worth millions of dollars and organizations take great care at controlling access to it, with respect to both internal users within organization, and external users, outside the organization [18]." Web applications have made the world around us run quicker and more efficiently. They make it possible to bank online, access email from most computers, and shop with ease. It wasn't long before hackers began taking advantage of these new luxuries for fun and profit. As web applications become more and more important in our everyday lives, better security has become a pressing need. We live in a world where web sites have user-generated content like blogs and social network site as well as personal information such as bank account information and credit card numbers. Therefore companies and organizations needed to build up their defenses and protect their clients' as well as their own assets. "The world has witnessed a rapid increase in the number of attacks on web applications" [13] and there is not one cure-all that solves all security problems.

C. *Cyberspace and the Law*

Laws are in place that govern cyberspace just like they do in the physical world we live in. The computer Fraud and Abuse Act (1986), among others, protects users from credit card abuse, stolen property, and trespassing [6]. Althought these laws are in effect, just like in the real world, they are often broken which is why securing web applications is so vital. Validating and filtering input is not a guaranteed way to secure a system much like locking a house door. People often lock their doors when they leave their houses. This is a very simple and easy way to protect their house from a stranger walking in and stealing or damaging their property. The same principle applies to input validation; it is fairly easy to do and will protect against most users with malicious intent. On the other hand, there are users out there who possess the skills and resources to bypass input validation much like there are burglars who have the experience and resources to gain entry into your house. If we do not protect the access of our web applications, people with very limited hacking experience can easily walk right into our systems.

D. *Client-Server Model*

Multitier architecture refers to a client-server architecture that separates the user interaction, the application processing, and the data management into logically separate processes. It provides a model for developers to create flexible and reusable applications. By breaking up an application into tiers, developers only have to modify or add a specific layer, rather than have to rewrite the entire application over. There are typically three layers:

- Presentation tier
- Business logic tier
- Data tier

1) *HTML – Passing parameters*

A typical way of exchanging information between web pages is thoroughly the submission of forms. Input is sent from one page's form to another page by one of two ways. The method attribute of a form tag can be either POST or GET as denoted below:

```
<form name="login" action="login.php"
method="get">
  <label>Username:</label>
  <input name="u_name"/><br>
  <input value ="Login"
                   type="submit"/>
</form>
```

Figure 1. Example of a from using the GET method to send information

Information past by the GET method shows up in the URL of a web site.

```
/login.php?u_name=johnnies10
```

Figure 2. Example of the adress bar from a GET request

The name of the input item is displayed along with the input which allows these pages to be bookmarked. Imagine a user searching Google Maps for the location of all the restaurants in the area. Instead of typing in this search each day want to try out a new place, you can simply select the bookmarked search. Because input is displayed, it is a good idea to avoid this method when sending sensitive information to a page, such as a login password. Sending private information can be easily accomplished using the POST method as seen below:

```
<form name="login" action="login.php"
method="post">
  <label>Username:</label>
  <input name="u_name"/><br>
  <input value ="Login"type="submit"/>
</form>
```

Figure 3. Example of a from using the POST method to send information

2) Server-Side Scripting Language

PHP: Hypertext Preprocessor (PHP) is a server-side scripting language which was first released in 1995 and has been under continuous development ever since [20]. PHP is a free, open source language for producing dynamic web pages. PHP code is embedded into the HTML source document and it is compiled on the server and then sent back to the user as plain HTML text. It is similar to other server-side scripting languages that provide dynamic content from a web server to a client such as Microsoft's Active Server Pages (ASP), Sun Microsystems' JavaServer Pages (JSP). As of April 2007, over 20 million Internet domains had web services hosted on servers with PHP installed. Facebook is an example of a large company that utilizes PHP's functionalities. PHP variables do not need to be declared. As soon as a variable is given a value, it is automatically assigned a type.

```
<?php
      $counter = "Hello World";
      echo $counter;
?>
```

Figure 4. Example of casting a variable as a String

As you can see above, $txt is assigned the characters "Hello World!" and therefore casted as a String and therefore has all the functionalities of a String object (i.e. strLen(), trim() , or strtolower()). In the following example, the variable $counter is assigned the value 5 and is therefore casted as an integer. It now is possible to perform mathematical functions such as addition, subtraction, multiplication, etc.

Figure 5. Example of casting a variable as an integer

PHP provides web developers with predefined functions that assist with input validation, database calls, and, with the most recent release, classes to allow for object orientated programming (OOP). In PHP, To access user input passed from a HTML form, the GET or POST global array must but called using item ID keys to reference specific variables. For example if a form is submitted and a user inputs a username in an input box that is labeled ‘u_name’, in PHP you would simply assign that item to a variable like below:

```
$username = $_POST['u_name'];
```

Figure 6. Example of assigning the username submitted by an html form (via the POST method) to a variable in PHP

Now the variable $username will contain a user’s username and can be used in a wide variety of ways such as dynamic welcome messages or database queries.

3) Structured Query Language (SQL)

SQL is a language designed for managing data in a relational database management system. It allows a user to retrieve information, delete records, insert records, and update records in a database. A hacker can use knowledge of how SQL is structured and how this query will be inserted into a dynamically formed query on a server to alter it's intended use (which will be discussed in further detail in the next section). Below is an example of a simple query:

```
SELECT title
FROM Books
WHERE price > 20.00
```

Figure 7. Example of a SQL query to retrieve the title of all books in the table that are more than $20

III. VULNERBILITIES

A. *HTML Injections*

There are two major web application security risks that exist today, SQL injections and cross-site scripting (XSS). Both depend on web sites’ failure to validate input being passed from a client to a server. Each of these attacks will be discussed in further detail below but the key point is that web developers must be skeptical of all input. The problem is application security is frequently overlooked. Unlike the legal system which states that everyone is innocent until proven guilty; all input should be considered malicious until proven trustworthy. Therefore all data should be sanitized and validated at the client level before it is sent to the server. Once the server receives the data it should be checked over once again to ensure that the proper parameters are being sent to databases. Web applications typically have rapid development cycles by developers who don’t have much experience in secure application development. “There are currently many poor coding practices that render web applications vulnerable to attacks such as SQL injection and cross-site scripting [13].” They need to identify all data entry points (HTML forms) and thoroughly validate each and everyone. As tools become more sophisticated and as corporate networks and applications are more interconnected and open, security must be built inside the application’s code, using secure programming practices. If not, hackers can do things such as access other people's web accounts and send emails on that user's behalf, acquire the list of contacts, and automatically BCC themselves on all outgoing emails; access banking or financial systems and transfer funds, apply for credit cards, or purchase checks; or purchase products on eCommerse sites.

1) SQL Injections

SQL injection attacks are based on web applications taking malicious strings read from clients and sending them into dynamically constructed database queries that alter the intended use. These database queries are statements that are aimed at retrieving data and manipulating records in databases. If the data is not properly processed prior to SQL query construction, malicious patterns can be injected that result in the execution of SQL or system commands. Apart from data retrieval and updates, SQL statements can also modify the structure of databases or delete them all together. It is important that client-side and server-side programs execute validation procedures prior to performing database access [13]. Developers need to consider all input malicious and sanitize it properly before using it to construct dynamically generated SQL queries [17].

SQL injection attacks have been responsible for stealing credit card numbers, usernames, passwords, and much more. In December 2009, an attacker breached a RockYou! plaintext database containing the unencrypted usernames and passwords of about 32 million users by using a SQL injection attack. He hacked the web site not for malevolent purposes, but to prove a point that most servers are susceptible to SQL injection attacks and furthermore their databases are poorly constructed and protected.

The string in the input box below alters the intended query's meaning.

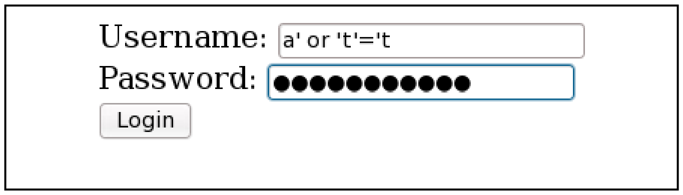
Username: a' or 't'='t
Password: ●●●●●●●●●●●

Figure 8. Example of a SQL Injection

The query was previously intended to place a username and password into a query and send it off to the server for processing.

```
  $queryString = "SELECT * FROM client
WHERE  u_name=".$_POST["u_name"].  "'AND
p_word='".$_POST["p_word"]."'";
```

Figure 9. The input from Figure 7 placed into a dynamic database query call

If we expand the variables to display the query that is actually being sent it looks as follows

```
$queryString = "SELECT * FROM client
WHERE u_name='a' OR '1'='1' AND
p_word='a' OR '1'='1'";
```

Figure 10. The query String from Figure 8 expaned

Due to the order of operations in SQL, this query will always evaluate to be true. When the database returns true to the web server, it grants the user improper access to the website. Developers need to consider all input malicious and sanitize it properly before using it to construct dynamically generated SQL queries [17]. "Depending on the web application, and how it processes the attacker-supplied data prior to building a SQL statement, a successful SQL injection attack can have far-reaching implications. The possible security ramifications range from authentication bypass to information disclosure to enabling the distribution of malicious code to application users [26]." Global variables can be utilized to discover information about databases. In the example below, the INFORMATION_SCHEMA.TABLES contains information on all tables in the database.

```
http://www.footlocker.com/items?id=10
UNION SELECT TOP 1 TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES--
```

Figure 11. Example of a more sophisticated SQL Injection to obtain the name of the first table in footlocker's database

We can get more specific and search for specific tables using the LIKE clause as in the Figure below.

```
/items?id=10 UNION SELECT TOP 1
TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES WHERE
TABLE_NAME LIKE '%25login%25'--
```

Figure 12. Example of a more sophisticated SQL Injection to obtain the exact name of the table that holds login information at footlocker

This example identifies the name of the table that holds login information. A user could then use this table name along with column names to identify specific users and their information. The effects of a successful SQL injection attack vary based on how the targeted application is implemented. SQL injections can be used for various reasons: to bypass application login, indirectly or directly obtain sensitive information, delete or alter contents, and even run commands against the database's host server.

B. XSS Attacks

Every day we interact with a large number of custom-built web applications. Many web sites publish content supplied by users in their communities. If this republished content contains scripts, visitors to the site can be exposed to XSS attacks. The standard defense is for the web sites to filter or transform any content that does not originate from the site itself, to remove scripts and other potentially harmful elements [15].

XSS attacks are essentially caused by web application failing to check up on user input before returning it to the client's web browser. Without an adequate validation, user input may include malicious code that may be sent to other clients and unexpectedly executed by their browsers, thus causing a security attack. "Ever since its conception, the World Wide Web has evolved towards an increasingly feature-rich, interactive and heterogeneous medium. Unlike early websites which were merely meant to deliver text in practical fashion, nowadays' Web 2.0 sites are not only capable of hosting rich content, such as images, videos, and audio material, but also provide platforms for users to contribute such data and share it with the rest of the world. Due to an increasing number of web sites offering features to contribute rich content, and the frequent failure of web developers to properly sanitize user input, cross-site scripting prevails as the most significant security threat to web applications." (Wurzinger p.33) They are by far the most common vulnerability found in web applications, identified in over 80% of all websites." Not even corporate giants like Google can completely escape these attacks. XSS enables the access and theft of web browser cookies, session IDs, and other sensitive information that the web site has access to which can then be reused to hijack online user accounts. (Whitehat Security)

Clients are unknowingly attacked by visiting a web page embedded with malicious JavaScript code. A hacker simply submits code to an area of a website that is likely to be visited by other users. These areas are often found on popular websites with community-driven features such as social networks, blogs, user reviews, message boards, chat rooms, web mail, wikis and numerous other locations. Once a user visits the infected web page, execution is automatic and the user has no means of defending himself. Therefore it is up to the server to filter out all potentially harmful data it displays to clients. (Whitehat Security)

XSS enables the access and theft of web browser cookies, session IDs, and other sensitive information that the web site has access to which can then be reused to hijack online user accounts. An example of an XSS attack would be posting the following script on a public bulletin board:

```
<script>alert("I stole your
cookies");</script>
```

Figure 13. Example of a XSS attack that displays a popup

This code, if not properly validated and character escaped, will be submitted to a server database and stored. Every time a new user views this community posting service, this entry will be displayed. The problem with this is that the post actually runs as JavaScript on an unsuspecting user's browsers. The example is innocent because it merely prompts all users viewing the page with a popup that says "I stole your cookies" This harmless JavaScript command can easily be

```
<script>alert(document.cookie);</script>
```

replaced with code that actually does send a user's cookie information directly to them.

Figure 14. Example of a XSS attack to display a user's cookie information that could contain a username and password

If the attacker doesn't have enough real estate to pull off a lengthy attack, they could simply use JavaScript to redirect you to their malicious page. Once you open this page, all the attacker's scripts can be run on your machine. Since we want websites to be very rich in content (bolded, italicized, display pictures/movies, etc), it is very difficult to filter out all the malicious input and still maintain the look and appeal of input from honest users. An example of this difficulty is you can hide html tags inside of html tags and when you 0use PHP's built in function strip_tags(), you will only capture the outside most tags and miss the embedded ones.

C. Historic Worms

XSS attacks have been well-known in the Internet community for several years. Samy is an XSS worm developed to spread across the social media site MySpace. The worm carried a script that would display the string "but most of all, Samy is my hero" on a victim's profile. When a user viewed that profile, they would have the script planted on their page. It was released on October 4th, 2005 and within 20 hours over one million users had run the script, making Samy one of the fastest spreading viruses of all time. The Yamanner worm is another example that was released on June 12th, 2006. Written in JavaScript, it targeted a vulnerability in the Yahoo! Mail service and spread through the Yahoo system, infecting the systems of those who opened the e-mails and sending the user's address book to a remote server. XSS attacks are simple to execute, but difficult to prevent and can cause catastrophic damage. Since the victim's browser receives malicious JavaScript code from a trusted web server, it trusts it and immediately executes. The key to stopping these entirely preventable attacks lie with software developers and information security professionals working with web applications.

IV. Initial Defense

When the number of XSS attacks began to grow, the first suggestions to users were to have them disable scripting languages in their browsers and avoid unrestricted browsing on untrustworthy websites. Yet, we live in a world that has become reliant on scripting languages that enhance web sites, these solutions reduce the functionality that is executable by the client side of the application. This is unacceptable. The problem should be addressed by web developers who need to check on the input received from a client, and encode or filter the output returned to the user. The server is responsible for receiving the input data and outputting it, not the user. Programmers could simply filter out all the tags that are submitted, but, again, this reduces functionality. A string submitted by a user to a message board such as <b>Hello World</b> is actually displayed as **Hello World** (in bold). If a web site removes these tags, the words 'Hello World' would not appear in bolded text. Web sites leaning towards rich content must be careful because malicious JavaScripts can be easily embedded.

V. Input Validation

Developers either completely leave out input-output validation or they implement it very poorly so that it does not filter out a truly complete set of potential attack characters. There are many facets to validation. First, validation should be carefully developed and rigorously tested by other professionals. Do not define all possible bad characters in a String; instead accept only the good ones (deny all characters except for certain allowable characters). Limit the number of characters accepted, that way you limit the length of a string that the attacker has to execute an attack. You can't solely rely on security checks at the client-side; attackers may be able to bypass client side validation. This means there must be validation on the server-side acting as a backup. Always have other developers conduct a code review to see if any security issues can be discovered. With good filtering in place, most security concerns are mitigated. Attackers will hunt for weaknesses and exploit them, so cover all of your bases. (SANS article)

There are many ways of validating input. The important things to check for are input length, object type, and if the input follows a desired pattern. Input length deals with the maximum and minimum characters a user has inputted. If the strlen() of a string is not within that range, the application should not proceed and may choose to return an error to the user. For example if we ask for a user's first name, the application should only accept strings of length 2 to 20. For the object type, input may be a string of characters, numbers, Boolean value, etc. For our first name example, the only thing the application should accept is a string of alphabetical characters. Patterns are useful in ensuring users enter acceptable strings. For our example, users should only be able to input strings in which the first letter is uppercase and the rest are lower case.

These should first be checked on the client side with a scripting language, such as JavaScript, so a request is never even sent to the server if the input does not match our business logic requirements. Browser validation is faster and reduces the server load. After a client's browser deems input valid, it is sent off to the server where it should then be checked again. This is important because it is very easy to bypass client-side validation. All a user must do is go into their web browser's preferences and select the option that doesn't allow web sites to run scripts on their browsers. Another option is to create a new HTML page that sends a web server the correctly labeled input values, thus eliminating client side script validation and allowing them to enter any input they want.

The second line of defense occurs on the server-side. You should especially consider server validation if the user input will be inserted into a database. Validation that occurs on the server is very similar to validation that is executed on the client's browser. Again we're checking string length, type, and pattern. In addition, validation at the business logic layer can also incorporate organizations specific functionalities. For example, company X may not allow you to change your profile on the 1st day of each month.

Below are some PHP functions to validate input and sanitize it [5][21][22][23]:

```
isset($_POST['u_name']);
```

Figure 15. Checks to make sure an HTML input box has been filled out

```
strLength($_POST['u_name'];
```

Figure 16. Checks the length of a String. Use to make sure input is in the correct size range

```
isString($_POST['u_name'];
```

Figure 17. Check if user input is a String

```
ctype_alnum($_POST['u_name'];
```

Figure 18. A PHP function to check if user input is made up of numbers and letters (not symbols

```
strip_tags($comment);
```

Figure 19. A PHP function to remove all html tags (i.e. bold <b>, italics <i>, JavaScript <script>

```
htmlspecialchars($comment);
```

Figure 20. A PHP function to turn HTML special characters like > and < into their HTML entitites &gt; and &lt;.

The difficult part about preventing against XSS attacks is that you cannot simply remove all <script></script> tags and be safe. XSS attacks can come in a multitude of attack vectors:

- <div style="url('javascript:alert(Hello World)')"></div>
- <script src=http://ha.ckers.org/xss.js></script>
- <img src=`javascript:alert("Hello!")' />
- <script>window.open('http://www.evil.site.com', 'width=400, height=300');<script>

User input can also nest their script tags <scri<script>pt> so if you do not recursively call the validations, the inner <script> tag will remain intact.

VI. Future Trends

We would always be behind in security if we only dealt with the present and disregarded the future. Looking back over the years, the initial attacks on software were buffer overflow attacks. These were carried out by users entering data the program was not designed to handle. This data was stored in a buffer outside of the memory that the programmer set aside. If the input was too large for the allocated memory, it overwrote adjacent memory which often contains other data causing things like security breaches, memory access errors, incorrect results, or even complete program crashes. Programming languages began building in measures to protect against accessing or overwriting data in any part of memory. Buffer overflow issues still exist today, but have since been diminished. SQL Injections became the next big exploitation to gain recognition. SQL Injections are user inputs that maliciously alter database queries and change their intended meanings. To combat these attacks, programmers began enforcing strict input validation such as checking input lengths, data types, and patterns on the client and server. Ever since SQL Injection attacks have become commonly prevented, Cross Site Scripting (XSS) as emerged as the new threat. XSS attacks force a web site to display malicious code, which then executes in an unsuspecting user's web browser. It executes on the client's browser, not on the server. Users access trusted web sites only to fall victim to code a third party has injected into it via ads, message boards, wikis, user reviews, etc. The effects of XSS attacks are vast; they can hijack accounts, record keystrokes, steal histories, load trojans, and much more. The point is that as the computing world evolves, so must security.

This document is merely an introduction to web application security. Input validation will continue to move towards standardization of functions that will encompass proven methods of validation. This will allow users to simply call a function and pass it the appropriate parameters versus writing their own validation for each form. This will drastically cut down on the number of mistakes programmers make. Input validation is just the icing on the cake! Security goes much deeper than that and there is much more to consider if we want to secure our information. We are currently unaware of the methods hackers will employ in the internet’s third decade but there are plenty of preventative measures our society can take to give us a fighting chance in securing information such as credit card numbers, social security numbers, bank accounts, etc. First, security must be a primary concern from the initial development of web applications. This means security cannot simply be tacked on at the end right before an application is shipped off. While creating web applications, developers must work in teams and thoroughly review each other's code. Not only does your own code need to be reviewed, you must also code review all the content you are displaying from 3rd parties. It is impossible to declare a system completely secure, especially when we still what it to be usable. A balance must be obtained between security and usability. Web applications must adapt a type of firewall that prevents brute force attacks from repeatedly searching a system until a vulnerability is found.

Security from the Beginning

Security must be considered and implemented into web applications from the beginning of their development. To demonstrate this, imagine you are baking a cake. To stand a chance at creating a delicious cake, you must add all the ingredients before putting it into the oven. It would be impossible to add the flour to the cake after it has been baked. Now consider web application development. If security is added at the very end of the process, it will not be as secure as a web application that chose to make security a priority from the beginning. "Security must be baked into the cake, not added afterwards [7]." How can this be accomplished? The answer is early education. Not only does security need to be implemented in many computer science courses, but courses must exist to take in-depth plunges into the world of security. Validating input is not something only people with multiple PhDs can perform. It does not take 10 years of experience; it is very simple to code and takes organization, meticulousness, early implementation, and professional code review. Companies should hold workshops or send their employees to security classes where they will learn the latest attacks and preventative measures. Security is imperative to an organization's success and it should be one of their top ethical concerns.

Code Review

It is very difficult to secure every aspect of a web application without help. The code you write must be thoroughly looked over by a professional. Coding a secure system is not like a solid block of American cheese, it is more like Swiss cheese. No matter how thorough you are there will always be holes that hackers may potentially be able to use to access your systems. Forgetting to validate even one input field could leave your entire system vulnerable which is why it is so important to team up with other programmers to limit the number and size of these holes. Even when you diminish the number of wholes in your system, if you directly display content from a 3rd party, you have opened up a huge vulnerability to your security. Since so many sites are now displaying catchy flash animation ads on their websites, it is more important than ever to review all code being displayed on your website. An ad designer could easily place dangerous code inside a flash animation that redirects you to a site of their choosing which can contain even more XSS attacks. You may have no intention of misleading users, but if you do not check the content your reputable site displays, you have a strong chance of harming others. Be smart. Use the resources around you and check all content being displayed on your web site whether it is your own, or someone else’s.

Usability vs. Security

It is very important to find the proper equilibrium between usability and security. This presents many problems. Consider the White House in Washington D.C. If gaining access to the oval office required someone to merely enter a one letter (a-z) password, White House staff would be happy. All employees, including the president of the United States of America, could easily enter the office within seconds, no problems right? But what if a terrorist walked in and entered a random letter? At worst case, a brute force attack of entering every letter (a-z) would take 26 tries. Now there is a problem. On the other end of the spectrum, a new security system has been implemented and gaining access to the oval office now requires a two hour process including five passwords that change daily, DNA tests, neurological monitoring, retinal scans, logical reasoning tests, full background checks, and more. Productivity would see a huge decrease due to the two hours a day people could not work. Employees would become annoyed having to be tested for two hours, every day, to confirm who they are. What is worse is people start fighting the system when they are unhappy. Now the president become tired of going through this completely unnecessary security check and he leaves a window open in a back room so he can bypass the security checkpoints every morning. If just one person walks by this open window during the night, climbs in, and shoots the president in the morning, all the time and money put into the security system would be for nothing and America will suffer a second rate Joe Biden presidency. As you can see there is a drastic difference in security approaches. A good security system should be invisible and balance usability and security.

Web Application Firewalls

Hackers are often successful because they are able to attack a site as often as they want. By using brute force attacks they can try thousands, if not millions, of automated attacks within minutes via web-bots (website parsers). Given enough attempts, a vulnerability will be found because no system is 100% secure. To prevent these brute force attacks, web application firewalls can be implemented. These require much more skill than the average web developer and therefore tend to be more expensive and time consuming than basic input validation. These firewalls begin by learning the behavior of the application. They then profile the server's traffic and make intelligent decisions to determine if a user is not behaving as intended. The breakthroughs in the data mining field have provided some powerful tools to help with anomaly/ intrusion detection and behavior analysis. These firewalls can view access records and say "Hey, wait a second! This user just attempted 4,000 form submissions and accessed links very systematic within that last two minutes. I will not allow his IP to access the server anymore." Again, creating these defenses can be very resource heavy and in many cases might be out of an organization's scope. On the other hand, banking companies such as Bank of America, TCF, or Wells Fargo, which all deal with personal and corporate finances, are ethically obligated to invest the time and money into these firewalls. In the future, this technology will advance and become cheaper and more precise allowing smaller operations to utilize its incredible added security.

In conclusion, nobody can guarantee where web application security will be in the future, but we can set ourselves up for optimal success by properly educating the next generation of web developers and constantly raising security awareness. It must be common knowledge to begin implementing security from the conception of web applications. Since methods of attacks are constantly changing and being redeveloped, it is more important than ever to keep developers up-to-date on the latest forms of attacks and their prevention measures. It is imperative that programmers work together to mitigate the number of vulnerabilities in their code. The new frontier of internet security is like the Wild West. It is an ongoing battle between the good guys and the bad guys; anything can happen at any moment and momentum can shift without a moments notice. Pick up a badge and become a new sheriff in town. Do not let ignorance be an excuse. You now know how easy it is to attack a web application and, even more importantly you now know how easy it is to defend against. Take some security classes, or at least read a few security books, so that when you are asked to create a web application, you will know just how important security is and you will be able to implement a secure system.

Further Considerations

Even if the client and server validate input, further security precautions can be taken. Other things to consider:

- Output validation
- PHP Preferred Statements
- SQL Stored Procedures
- Data Encryption
- Secure Socket Layer (SSL)

VII. CONCLUSION

With web applications providing a plethora of quick and efficient services to users all around the globe, the security of their information has never been more important. People can access their bank and email accounts, purchase goods and services, and connect with other people around the globe from anywhere with an internet connection. Although this is very convenient, it also presents a problem. A bank like WellsFargo can receive thousands of web site hits a day from nearly anonymous users. These hits may be malicious attempts at logging in. These login submissions may contain special characters to alter the intended use and gain improper access to information. For this reason, when designing the security of one's web application, one must never trust user input. One must treat all data entering a web application from an outside network connection as potential harmful. XSS attacks are complex and are not going away anytime soon due to the increasing number of sites that mainly rely on user generated content. With proper input validation and sanitization, most of the minor security risks can be minimized. It is difficult to fully secure a web application when they heavily depend on user input which is why it is important to build security into an application from the start and have all code peer reviewed by a professional. This research is merely a gateway to the rest of the security implications web applications face; hackers are not going away any time soon and it is imperative that every web developer understands the basics that have been laid out and, furthermore, takes it upon themselves to pursue further protection means and stay up-to-date on current security practices.

References

[1] Anley, C., Grindlay, B., Heasman, J., & Litchfield, D. (2005). *The Database Hacker's Handbook: Defending Database Servers*. New York, NY: Wiley.

[2] Begg, C. E., & Connolly, T. M. (2009). Database Systems: A Practical Approach to Design, Implementation and Management (5th Edition) (5 ed.). Toronto: Addison Wesley.

[3] Bertino, E., & Sandhu, R. (2005). Database security-concepts, approaches, and challenges. IEEE Trans.Dependable Secur.Comput., 2(1), 2-19. Retrieved from http://dx.doi.org/10.1109/TDSC.2005.9

[4] Buehrer, G., Weide, B. W., & Sivilotti, P. A. G. (2005). Using parse tree validation to prevent SQL injection attacks. Paper presented at the SEM '05: Proceedings of the 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal. 106-113. Retrieved from http://doi.acm.org/10.1145/1108473.1108496

[5] Cannings, R., Dwivedi, H., Lackey, Z., & Stamos, A. (2007). Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions (Hacking Exposed) (1 ed.). New York: Mcgraw-Hill Osborne Media.

[6] Cavazos, E., & Morin, G. (1994). Cyberspace and the Law: Your Rights and Duties in the On-Line World. London: The Mit Press.

[7] Cross, Jason. Personal interview. 29 Mar. 2010.

[8] Daswani, N., Kern, C., & Kesavan, A. (2007). Foundations of Security: What Every Programmer

[9] Needs to Know (Expert's Voice). New York: Apress.

[10] Elmasri, R., & Navathe, S. B. (2006). Fundamentals of Database Systems (5th Edition) (5 ed.). Toronto: Addison Wesley.

[11] Grossman, J., Hansen, R., Rager, A., Fogie, S., & Petkov, P. (2007). XSS Attacks Cross Site Scripting Exploits &Defense - 2007 publication. Rockland: Syngress Pub., 2007.

[12] Guimaraes, M. (2006). New challenges in teaching database security. Paper presented at the InfoSecCD '06: Proceedings of the 3rd Annual Conference on Information Security Curriculum Development, Kennesaw, Georgia. 64-67. Retrieved from http://doi.acm.org/10.1145/1231047.1231060

[13] Huang, Y., Huang, S., Lin, T., & Tsai, C. (2003). Web application security assessment by fault injection and behavior monitoring. Paper presented at the *WWW '03: Proceedings of the 12th International Conference on World Wide Web,* Budapest, Hungary. 148-159. Retrieved from http://doi.acm.org/10.1145/775152.775174

[14] Huseby, S. H. (2004). Innocent Code: A Security Wake-Up Call for Web Programmers (1 ed.). New York, NY: Wiley.

[15] Jim, T., Swamy, N., & Hicks, M. (2007). Defeating script injection attacks with browser-enforced embedded policies. Paper presented at the *WWW '07: Proceedings of the 16th International Conference on World Wide Web,* Banff, Alberta, Canada. 601-610. Retrieved from http://doi.acm.org/10.1145/1242572.1242654

[16] Juillerat, N. (2007). Enforcing code security in database web applications using libraries and object models. Paper presented at the LCSD '07: Proceedings of the 2007 Symposium on Library-Centric Software Design, Montreal, Canada. 31-41. Retrieved from http://doi.acm.org/10.1145/1512762.1512766

[17] Kals, S., Kirda, E., Kruegel, C., & Jovanovic, N. (2006). SecuBat: A web vulnerability scanner. Paper presented at the *WWW '06: Proceedings of the 15th International Conference on World Wide Web,* Edinburgh, Scotland. 247-256. Retrieved from http://doi.acm.org/10.1145/1135777.1135817

[18] Kamra, A., Bertino, E., & Lebanon, G. (2008). Mechanisms for database intrusion detection and response. Paper presented at the *IDAR '08: Proceedings of the 2nd SIGMOD PhD Workshop on Innovative Database Research*, Vancouver, Canada. 31-36. Retrieved from http://doi.acm.org/10.1145/1410308.1410318

[19] Lucca, G. A. D., Fasolino, A. R., Mastoianni, M., & Tramontana, P. (2004). Identifying cross site scripting vulnerabilities in web applications. Paper presented at the *WSE '04: Proceedings of the Web Site Evolution, Sixth IEEE International Workshop,* 71-80.

[20] PHP. (n.d.). Wikipedia, the free encyclopedia. Retrieved March 12, 2010, from http://en.wikipedia.org/wiki/PHP

[21] PHP Security. (n.d.). PHP: Hypertext Preprocessor. Retrieved March 13, 2010, from http://us.php.net/manual/en/security.php

[22] PHP Tutorial. (n.d.). W3Schools Online Web Tutorials. Retrieved March 12, 2010, from http://www.w3schools.com/php/default.asp

[23] PHP Tutorials. (n.d.). phPro.org. Retrieved March 14, 2010, from http://www.phpro.org/

[24] Rietta, F. S. (2006). Application layer intrusion detection for SQL injection. Paper presented at the ACM-SE 44: Proceedings of the 44th Annual Southeast Regional Conference, Melbourne, Florida. 531-536. Retrieved from http://doi.acm.org/10.1145/1185448.1185564

[25] Roichman, A., & Gudes, E. (2007). Fine-grained access control to web databases. Paper presented at the SACMAT '07: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, Sophia Antipolis, France. 31-40. Retrieved from http://doi.acm.org/10.1145/1266840.1266846

[26] Sammut, T. (n.d.). Understanding SQL Injection. Cisco Systems, Inc. Retrieved March 14, 2010, from http://www.cisco.com/web/about/security/intelligence/sql_injection.html

[27] SQL Injection Walkthrough. (n.d.). SecuriTeam.com. Retrieved March 14, 2010, from http://www.securiteam.com/securityreviews/5DP0N1P76E.html

[28] Wassermann, G., & Su, Z. (2008). Static detection of cross-site scripting vulnerabilities. Paper presented at the *ICSE '08: Proceedings of the 30th International Conference on Software Engineering,* Leipzig, Germany. 171-180. Retrieved from http://doi.acm.org/10.1145/1368088.1368112

[29] Wurzinger, P., Platzer, C., Ludl, C., Kirda, E., & Kruegel, C. (2009). SWAP: Mitigating XSS attacks using a reverse proxy. Paper presented at the *IWSESS '09: Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems,* 33-39. Retrieved from http://dx.doi.org/10.1109/IWSESS.2009.5068456