A Study on web Applications & Protection against Vulnerabilities

Padmaja K

Abstract

Web applications are widely adopted and their correct functioning is mission critical for many businesses. Online banking, emails, e-shopping, has become an integral part of today's life. Vulnerabilities in web application can lead to a variety of erroneous behavior at dynamic run time. We encounter the problem of forceful browsing in many web applications, username enumeration can help an attacker who attempts to use guessable passwords, such as test/test, admin/admin, guest/guest, and so on. These accounts are often created by developers for testing purposes, and many times the accounts are never disabled or the developer forgets to change the password, hacking reduces the performance or function of the application, further more, the modified system itself becomes a constraint to counter newer types of vulnerabilities that may crop up from time to time. Hence, the best solution would be to finds the steps to solve that are web-based (firewall) independent for protecting against vulnerabilities in web applications. In our work algorithm is to analyze vulnerabilities that are caused by breaking of the data dependency using problem which work efficient with existing one.

Keywords – Web Application, Vulnerabilities, Forceful browsing, Testing, Dynamic Testing

INTRODUCTION I

Web applications are collections of static files linked with each other by means of HTML references. With this dynamic feature was traditionally implemented by CGI scripts were added to web page to accept the user input, changing presentation and content of the pages accordingly. Currently more often web sites are created dynamically, to send the sites content is stored in database entries and present them to the user. While in the beginning user interaction was typically limited to simple request-response pairs, web applications today often require a multitude of intermediate steps to achieve the desired results. When developing software, an increase in complexity typically leads to a growing number of bugs. Of course, web applications are no exception. Moreover, web applications can be quickly deployed to be accessible to a large number of users on the Internet, and the available development frameworks make it easy to produce (partially correct) code that works only in most cases. As a result, web application vulnerabilities have sharply increased. For example, in the last two years, the three top positions in the annual Common Vulnerabilities and Exposures (CVE) list published by Mitre [3] were taken by web application vulnerabilities. To identify and correct bugs and security vulnerabilities, developers have a variety of testing tools at their disposal. These programs can be broadly categorized as based on black-box approaches or white-box approaches. White-box testing tools, such as those presented in [1] use static analysis to examine the source code of an application. They aim at detecting code fragments that are patterns of instances of known vulnerability classes [2]. Since these systems do not execute the application, they achieve large code coverage, and, in theory, can analyze all possible execution paths [4]. A drawback of white-box testing tools is that each tool typically supports only very few (or a single) programming language. A second limitation is the often significant number of false positives. Since static code analysis faces undecidable problems, approximations are necessary. Especially for large software applications, these approximations can quickly lead to warnings about software bugs that do not exist [5].

Attacks against Web applications come in a variety of forms, but it is important to understand that viewing Web application security with merely an attack-verses-vulnerability perspective results in an overly narrow focus. Studies such as conducted by Erickson and Howard reinforce the point that the overall security posture of a Web application depends on a variety of factors such as proper configuration, continuity within application logic and workflow, as well as factors such as competent administration and observance to security policies on the part of corporations that own and manage application data. Several organizations have published lists of the top categories of Web application vulnerabilities, notably the OWASP Top 107 and the WASC Threat Classification8. However, each list differs both in the level of abstraction and types of Web application vulnerabilities included among its top threats. The Cenzic Intelligent Analysis (CIA) Lab uses its own framework for classifying the top vulnerability threat classes, the methodology having been derived from its proprietary HARM system, the Hailstorm® Application Risk Metric (ARC™), which is explained in detail later in this document. Analysis provided below will show vulnerability information from all three categories, for comparative purposes, so that organizations using any one of the threat classification systems will have Q1 2007 data related to the methodology they are presently using.

SECTION II

2. Software vulnerabilities: vulnerabilities occurs in different ways hardware, sites, software, organizations, we focusing on software in input validation errors such as

2.1. Format String: The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application. In this way, the attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.
To understand the attack, it's necessary to understand the components that constitute it.

- The Format Function is an ANSI C conversion function, like printf, fprintf, which converts a primitive variable of the programming language into a human-readable string representation.
- The Format String is the argument of the Format Function and is an ASCII Z string which contains text and format parameters, like: printf ("The magic number is: %d\n", 1911);
- The Format String Parameter, like **%x %s** defines the type of conversion of the format function.

The attack could be executed when the application doesn't properly validate the submitted input. In this case, if a Format String parameter, like %x, is inserted into the posted data, the string is parsed by the Format Function, and the conversion specified in the parameters is executed. However, the Format Function is expecting more arguments as input, and if these arguments are not supplied, the function could read or write the stack.
If the application uses Format Functions in the source-code, which is able to interpret formatting characters, the attacker could explore the vulnerability by inserting formatting characters in a form of the website. For example, if the printf function is used to print the username inserted in some fields of the page, the website could be vulnerable to this kind of attack, as showed below:
printf (username);

2.2. SQL Injection: It have more severe consequences than XSS due to the fact that a successful SQL injection can comprise the integrity of a database. Vulnerable in web application is a SQL injection if unvalidated user input is used to generate SQL queries . typical SQL query used to generate dynamic web pages

```
SELECT * FROM ARTICLES
WHERE id='<user input>'
```

An attacker user input control e.g enter

```
,;DROP ('articles');
```

Adds a command to the SQL query which then becomes:

```
SELECT * FROM articles WHERE id=' ';DROP('articles');';
```

These SQL commands will select some data, delete the table articles in the database and then generate an SQL error due to the single quotation mark. SQL injection gives an attacker the opportunity to manipulate the database and in special cases execute arbitrary code on the database server. It is therefore an effective attack on web applications. Typical attack are logins, search forms and the URL of dynamically generated pages e.g http://vulnerableSite.com/article?id=42 could result in a SQL query similar to the one in the example. SQL injection can be avoided through user input validation, ensuring appropriate handling of characters with a special meaning in SQL.

2.3. Cross-site Scripting: The main purpose of cross-site scripting is a XSS attacks [5]. To steal the credentials i.e cookies of an authenticated user. .request in the web contains an authentication cookie is treated by the server as a request of the corresponding user as long as does not explicitly log out. Everyone who manages to steal cookies is able to impersonate its owner for the current sessions. The browser automatically sends a cookie only to the web sites that created it, but with JavaScript program are restricted by the same origin policy. XSS attacks circumvent the same –origin policy by injecting malicious java script into the output of vulnerable applications. In this case, the malicious code appears to originate from the trusted site and thus, has complete access to all (sensitive) data related to this site. For example, consider the following simple PHP script, where a user's search query is displayed after submitting it:

echo "You searched for " . $_GET['s'];

The user's search query is retrieved from a GET parameter. Therefore, it can also be supplied in a specifically crafted URL such as the following, which results in the user's cookie being sent to "evilserver.com":

```
http://vulnerable.com/post.php?s=<script>document.location
='evilserver.com/steal.php?'+document.
Cookie</script>
```

All that the attacker has to do is to trick a user into clicking this link, for example, by sending it to the victim via email. As soon as the user clicks on this link, her browser visits the page post.php on the vulnerable site, with the GET parameter "s" set to the malicious JavaScript code. As a result, the malicious code is embedded in the application's reply page, and now has access to the user's cookie. The JavaScript code sends the cookie to the attacker, who can now use it to impersonate the victim. The particular type of XSS vulnerability discussed above is called reflected XSS,

since the attacker's malicious input is immediately returned (i.e., reflected) to the victim. There also exists a second type of XSS, where the application first stores the input into a database or the file system. At a later stage, the application retrieves this data through database queries or files reads, and finally sends it to the victim. For instance, such *stored* XSS vulnerabilities often occur in web guest books or forums, where a visitor leaves a comment that is later accessed by another visitor. In general, an XSS vulnerability is present in a web application if malicious content (e.g., JavaScript) received by the application is not properly stripped from the output sent back to a user. When speaking in terms of the sketched class of taint-style Vulnerabilities, XSS can be roughly described by the following properties:

• *Entry points* into the program: GET, POST and COOKIE arrays.

• *Sanitization routines*: PHP functions such as htmlentities () and htmlspecialchars (), and type casts that destroy potentially malicious characters or transform them into harmless ones (such as casts to integer).

• *Sensitive sinks*: All routines that display data on the screen, such as echo(), print() and printf().

This tool can only handle reflected XSS vulnerabilities. However, it is straight forward to use it for the detection of stored XSS as well, given a certain program policy with regard to the taint status of persistently stored data. For instance, it is customary that data is not sanitized before it is stored to a database or to the file system, which means that it has to be sanitized after its later retrieval. In the System, this can be modeled by adding the corresponding data retrieval functions to the set of entry points. Analogously, the application's policy can demand that all data is sanitized before it is stored. In this case, data storage functions have to be defined as sensitive sinks. Mixed policies are more difficult to handle. For instance, an application could expect a certain database table to contain only sanitized values, whereas some other table might also be allowed to contain unsanitized values. Here, the analysis would also have to resolve the names of the tables that are used for storage and retrieval.

2.4.HTTP Header Injection: It allows attackers to split a HTTP response into multiple ones by injecting malicious response HTTP headers. This can deface web sites, poison cache and trigger cross-site scripting.

Normally,

http://www.mysite.com/test/default.aspx?text=esiu

sets the cookie

//Query parameter text is not checked before saving in user cookie

NameValue collection request = RequestQuerystring:

//Adding cookies to the response

Response.Cookies["UserName"]Value=request["text"];

Set-Cookie header is used in HTTP response to request browser to save a cookie, %0D%0A is a new line character

On a HTTP response encoded by URL encoding, this is usually represented as "\r\n" in code.

2.5. HTTP Response Splitting: attack involved in 3 types

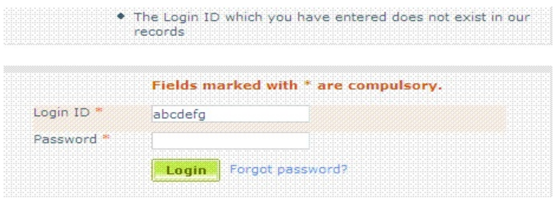
Web server which has a security hole enabling HTTP Response splitting

Target – Entity that interacts with the web server perhaps on behalf of the attacker. Typically this is a cache server forward/reverse proxy or browser attacker which initiates the attack.

2.6. Forceful Browsing: Forceful browsing is making several requests to the web server with the URL patterns of typical web application components such as CGI programs. The common with many of the exploits is that lack of server side validation makes them possible. Client side validation doesn't provide real protection as it is always possible to create a custom user agent or use an intermediary tool. These attacks against web[9] applications are that there are so many things can do wrong.

SECTIONIII

3. Problem definition: Today's life hectic with our schedules we go for online banking, emails, chats, e-shopping become an integral part. Thus web-based techniques are widely adopted and their correct functioning is mission critical for many business applications, vulnerabilities are readily exploited by attackers. Unfortunately software failures occur to reduce system availability and efficiency of the system as a whole. Here we have method to solve the vulnerabilities.



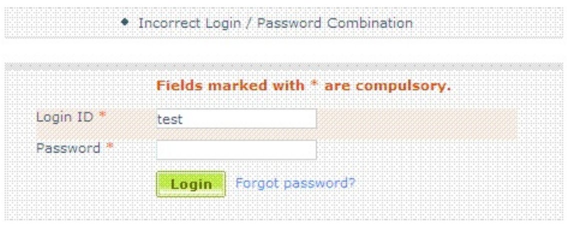
• The Login ID which you have entered does not exist in our records

Fields marked with * are compulsory.

Login ID * abcdefg

Password *

Figure 1. User Login Screen


Incorrect Login / Password Combination
Fields marked with * are compulsory.
Login ID *
Password *
Login
Forgot password?
test

Username enumeration can help an attacker who attempts to use some trivial usernames with easily guessable passwords, such as test/test, admin/admin, guest/guest, and so on. These accounts are often created by developers for testing purposes, and many times the accounts are never disabled or the developer forgets to change the password.
During testing assignments, have found such accounts are not only common and have easily guessable passwords, but at times they also contain sensitive information like valid credit card numbers, passport numbers, and so on. Needless to say, these could be crucial details for social engineering attacks.

3.1. Email Vulnerabilities: Email is one of the most widely used applications on the Internet due to its convenience, cost effectiveness, and time saving ability. Because of its ubiquitous capability it can be left open to many different types of vulnerability.

There are multiple ways that hackers can attack your email clients. Some of these methods include distribution of malware such as spyware, adware, Trojans, and viruses, to name a few types. Other attacks on your email client can include phishing, spam that is laced with malware, and denial of service attacks which are the result of sending a massive amount of messages to a server causing it to crash. Attacks can also cause a lot of damage to your other applications, data, and ultimately the PC operating system itself.

3.2. Protect from email vulnerabilities: Our computer operating system is used as a platform for email client. Regardless of what type of client we use such as Microsoft Outlook, Outlook Express, Eudora, or other, there are steps we can take to protect email client against vulnerabilities

3.2.1. Plain *Text:* When checking our email message, use plain text format instead of formats such as HTML or rich text format that can open up email client to vulnerabilities hackers to exploit.

3.2.2. Automatic Updates: Always use the latest version of the mail client software and make sure you have the automatic update feature enabled.

3.23.3.Antivirus Software*:* Use antivirus software that includes a virus signature for monitoring your email files. Depending upon the program we are using, often can configure the automatic update for virus signatures.

3.2.4. Do Not Unsubscribe: If we receive unsolicited email do not click to unsubscribe to the list as it could contain malware or lead you to a website that is infected with malware. Simply delete the unsolicited message or if it ended up in your spam folder, clear the folder altogether.

3.2.5. Administrator: Avoid running email client under administrator privileges. If this is not possible, try to restrict the privileges while logged on as administrator. The administrator privileges can open up your email clients to exploits by a hacker.

3.2.6Attachments: Make sure attachments are scanned by your antivirus program before you open them. Most antivirus programs contain this feature and will let you know if there is a threat of a virus before you open the attachment.

3.2.7. Receipts and Confirmations: Configure the settings in email client so it does not automatically send return receipts or read confirmations. If an email is infected automatically opening or sending a message could spread the infection to the recipient's email client.

3.2.8.Use Encryption*:* To ensure that confidential information is secure, use encryption for sending these types of messages.

3.3. Protect from Web Vulnerabilities: All the Consequences of the most common web application security vulnerabilities we present a basic methods to protect against these vulnerabilities to secure coding security program

3.3.1Inject flaws: Injection occurs when user-supplied data is sent to an interpreter as part of a query. The attackers hostile data tricks the interpreter into executing unintended queries or changing data.

3.3.2. Malicious File Execution: Code vulnerable to remote file inclusion allows attackers to include hostile code and data resulting in devastating attacks such as total server compromise malicious file execution attacks affect PHP, XML and any framework which accepts filenames or files from users

3.3.3. Insecure Direct Object: A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file directory database record or key as a URL or form parameter. Attacker can manipulate those references to access other objects without authorization.

3.3.4. Cross Site Request Forgery*:* A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application which forces the victim's browser to perform a hostile action to the benefits of the attacker CSRF can be as powerful as the web application that it attacks

3.3.5 Information Leakage & improper Error: Applications can unintentionally leak information about their configuration internal workings or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data or conduct more serious attacks.

3.3.6. Broken Authentication session Management: Account credentials and session tokens are often not properly protected. Attackers compromise passwords keys or authentication tokens to assume other users identities.

3.3.7. Insecure Cryptographic Storage: Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes such as credit card fraud

3.3.8. Insecure Communications: Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.

3.3.9. Failure to restrict URL access: frequently an application protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directory

SECTION IV

4.1. Algorithms for Detecting vulnerabilities:

Content of text SQLMF are a set of n number of legitimate SQL queries (where, 1≤ n). Each query is expressed as a sequence of elements {s_1, s_2,.., s_n`}. Each element is a string of characters. The text pattern (P) of the dynamic query, is expressed as one or more elements {s`$_1$,s`$_2$,.. s`n}, where, 1≤ n. An element may have one or more sub elements, identifiers and values. A function element count(P) computes the number of elements in P. Each element is separated from others by semicolon (;).

Exact Matching

```
Input : T, P
        Output: Safe Query, Attack Alarm I
    [1] match_count <- 0;
    [2] For i= 1 to n do
            Begin
    [3] If (P=Ti) then {
            - Add 1 to match_count;
            - Declare 'Safe Query';
            - Exit; }
    [4] End if;
    [5] End For Loop;
    [6] If (match_count=0) then {
            - Declare 'Attack Alarm I';
            - Call Approximate Matching; }
    [7] Stop;
```

Approximate Matching

```
Input : T, P, W
        Output: Safe Query, Attack Alarm Final
    [1] k = element_count(P);
    [2] For i = 1 to n do {
    [3] For j = 1 to k do {
    [4] If (P[j] c T[i][j]) then
    [5] D[i] b D[i] + 1 ;
    [6] Enf if ; } }
    [7] Edit_Distance b 0 ;
    [8] For i = 1 to n do {
    [9] Edit_Distance = MIN (D[i]); }
    [10] If (Edit_Distance < W) then {
            - Declare 'Safe Query' ;
            - Execute P; }
    [10] Else {
    [11] - Declare 'Attack Alarm Final' ;
    [12] - Block P; }
    [13] End if;
    [14] Stop;
```

4.2. Dynamic Analysis for vulnerabilities in web applications:

In context of web applications, static approaches have limited potential because, web applications are often written in dynamic scripting languages that enables on fly creation of code the issue pose significant challenges to approaches based on static analysis. Testing of dynamic Web applications is also challenging because the input space is large and applications typically require multiple user interactions. The state of the practice in validation for Web standard compliance of real Web applications involves the use of programs such as HTML Kit5 that validate each generated page, but require manual generation of inputs that lead to Dynamic Analysis Testing Tools

4.2.1. DART*:* (directed automated random testing) integration of random testing and dynamic test generation using symbolic reasoning is best intuitively explained with an example.

Consider the function h in the file below:

```
int f(int x) f return 2 * x; g
int h(int x, int y) {
if (x != y)
if (f(x) == x + 10)
abort(); /* error */
return 0;}
```

The function h is defective because it may lead to an abort statement for some value of its input vector, which consists of the input parameters x and y. Running the program with random values of x and y is unlikely to discover the bug. The problem is typical of random testing: it is difficult to generate input values that will drive the program through all its different execution paths. In contrast, DART is able to dynamically gather knowledge about the execution of the program in what we call a directed search. Starting with a random input, a DART-instrumented program calculates during each execution an input vector for the next execution.

4.2.2. Apollo: Apollo first executes the Web application under test with an empty input. During each execution, Apollo monitors the program to record path constraints that reflect how input values affect control flow. Additionally, for each execution, Apollo determines whether execution failures or HTML failures occur (for HTML failures, an HTML validator is used as an oracle). Apollo automatically and iteratively creates new inputs using the recorded path constraints to create inputs that exercise different control flow. Most previous approaches for concolic execution only detect "standard errors" such as crashes and assertion failures. This approach detects such standard errors as well as uses an oracle which are interactively supplied by the user (e.g., by clicking buttons in generated HTML pages).

CONCLUSION V

Our work have presented an approach to improve the best functionality web applications by the absence of runtime errors, dynamically proposed solution prevent due to data dependencies on session data. Algorithm combines to develop program annotation verification and validation checking to protect against broken data dependencies in web applications. In addition, the proposed solution is interoperable with the existing web infrastructure and does not interfere with other web security solutions. Moreover, the proposed solution is able to leverage the power of existing web security by providing formal techniques guarantee to prove the absence of broken data dependencies in a given web protocol enforcement configuration. To the best of our knowledge, the research presented in this paper is the first to improve web application security by providing an appropriate solution to the specific problem of broken data dependencies on session data.

References

[1] A registration page had an an HTML comment mentioning a file named " _private/customer.txt"typing http://www.xxx.com/_private/customer.txt sent back all customers information

[2] Appending "~" or back or old to GCI names may send back an older version of the source code. For example www.xxx.com/cgi-bin/admin.jsp~ returns admin.jsp source code. Here hacking attempts that every serious business application should be able

[3] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990.

[4] Burp Spider. Web Application Security. http://portswigger.net/spider/, 2008.

[5] Acunetix. AcunetixWeb Vulnerability Scanner. http://www.acunetix.com/, 2008.

[6] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanov, E. Kirda, C. Kruegel, and G. Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *IEEE Security and Privacy Symposium*, 2008.

[7] Karl Forster, Lockstep Systems, Inc., "Why Firewalls Fail to Protect Web Sites," http://www.lockstep.com/products/ webagain/why-firewalls-fail.pdf, 2007.

[8] I. Ristic, "Web Application Firewalls Primer," (IN)SECURE, vol. 1, no. 5, pp. 6-10, Jan. 2006.

[9] Shah, Shreeraj. Hacking Web Services. 2007.

Padmaja K pursuing Ph.D from JNTU Kakinada. M.Tech(CSE) from JNTU Kakinada. She is having 10 years of experience in Academics. Has guided many UG & PG students. Her research areas include Software Engineering, Software Security