SANCTUM

ANATOMY OF A WEB APPLICATION: Security Considerations

White Paper

Steve Pettit, Sanctum Inc.

July, 2001

Sanctum, the Sanctum logo, AppShield, Policy Recognition and Adaptive Reduction are trademarks of Sanctum, Inc. Products mentioned herein are for identification purposes only and may be registered trademarks of their respective companies. Specification subject to change without notice.
©2001 Sanctum, Inc. All rights reserved.

TABLE OF CONTENTS

Overview

Providing a secure Web environment has become a high priority for companies as eBusinesses have increased the amount and the sensitivity of corporate information that can be accessed through the Web. This very increase in online traffic and transactions is also making it more difficult to secure a Web site. This paper will examine the threats to corporate information and resources that exist on Web sites. Specifically, this paper will examine the applications that access information over the Web, called Web applications, and look at the components of these applications and how they are vulnerable to attack.

First, the paper will address the question, what is a Web application? Many people are unaware of the complexity of these applications, the combinations of code that they contain, and the resources they access within the company. Then the paper will address vulnerabilities that exist at each of the layers of an application, and show how hackers can use those vulnerabilities to access public and nonpublic information other than what the application designer intended. Finally, the paper will examine solutions to the problem of Web application Security, looking both at the individual layers of Web applications, and at these applications as a whole. Not surprisingly, a number of vendors have released products with the tag line "We secure eBusiness." As we will see, these products vary significantly in the vulnerabilities they target and the success with which they mitigate them.

What is a Web application and what goes into building it?

Web applications are complicated entities. According to the IT reference Web site Whatis.com (a TechTarget site), an application is "a program designed to perform a specific function directly for the user or, in some cases, for another application program. Examples of applications include word processors, database programs, Web browsers, development tools and communication programs. Applications use the services of the computer's operating system and other supporting applications." In short, an application is a piece of code that does something, and that may access other pieces of code to do it.

Web applications are the business logic that enables user's interaction with the web site, and the transacting and interfacing with all the back-end data systems. Examples include applications that allow users to look up their account information at their bank and move funds; applications that allow users to buy things online, such as shopping carts and transaction software; supply-chain automation applications that link suppliers to a manufacturer, and many, many others. What these applications have in common is that they are composed of code that was written explicitly for the Web interface and code from many other sources that accesses internal data and performs transactions. Additionally, the databases that are accessed and the data the database contain are all crucial elements of the Web application. The components of a Web application are shown schematically in Figure 1. They are described in more detail in Section III.

The Web Application

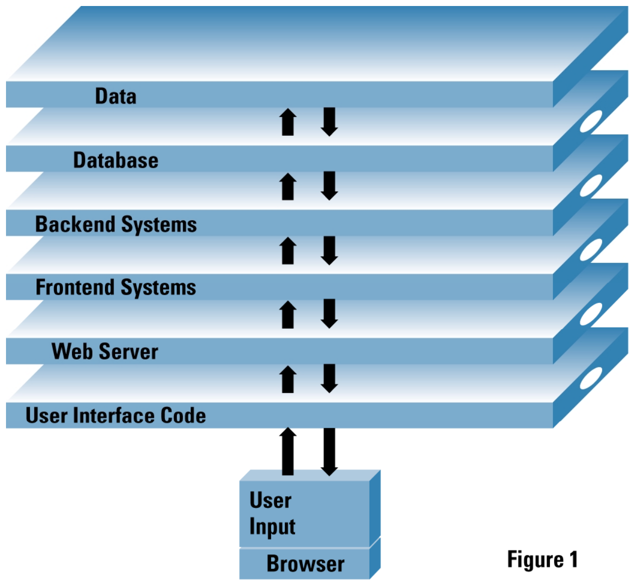

Figure 1

As you can see in Figure 1, Web applications include code that resides on the Web servers, application servers, databases, and backend systems of an organization. In short, they are any application that will be accessed in some way, shape, or form through the Web. The multiple parts of a Web application are often developed, supported and maintained by different departments within a company. While the term “application” connotes one discrete entity, in reality the “applications” driving the largest enterprises in the world are actually stacks of code coming from multiple places, some developed in-house and some from third party vendors, that all must work properly with each other. Integrating and managing these applications requires interdepartmental cooperation, and if the integration between the applications isn’t clean, or if any of the component pieces of code contains vulnerabilities, the Web application may be vulnerable to failure or to an attack. Given this complex equation, along with speed-to-market pressures and the current lack of skilled security professionals, the possibility of human error leading to vulnerable applications is significant. In fact, at the time of this writing, Gartner Group estimates that 75 percent of Web site hacks that occur today actually happen at this application level and this number is expected to increase.

Securing a Web application is difficult, not only because of the cross-departmental coordination involved, but because most security tools are not designed to address the Web application as a whole, including how the different pieces of the application interact with each other. The potential for a security breech exists in each layer of a Web application. Traditional security solutions, such as access control or intrusion detection systems, are specialized to protect different layers of the Internet infrastructure. While these tools are useful for their specific functions, they do not address all of the issues that Web applications present. And using these tools can give administrators a false sense of confidence if they do not realize that they are not addressing many of the vulnerabilities that exist. Additionally, some security solutions can impair the usability of the site by inhibiting the Web application flow. For example, host access controls that are improperly configured can prevent a user from accessing resources they should have access to.

One needs to look at the application life-cycle to fully understand the level of risk inherent in today's eBusinesses. At the start, Web application developers are generally not hired or known for their security expertise, and they develop the applications for functionality (including look and feel) and performance. Therefore, it is not surprising there are frequently flaws in design, implementation and testing of Web applications with respect to security. Quality Assurance (QA) follows the development where code is examined for functionality and efficiency over integrity and security. When the auditors come into play they generally check for syntax issues, but rarely for content logic. So applications commonly go into production with some vulnerabilities, often even published (known) vulnerabilities for which fixes exist. At the same time, hackers make a living at identifying vulnerabilities that can allow them to break into these applications. Once a site has been broken into and it is too late to avoid damage the application then goes back to the developer to patch (more time and money), through QA (more time and money) and back into production.

A new class of security solution can now address these concerns. One methodology to address Web application security is with an automated solution, where the security policy is driven directly by the Web application itself and the security solution understands the Web application as a whole. Ensuring the integrity of interactions between the user and the application is at the heart of application security.

Security: Establishing And Enforcing Trust

Security is all about trust. This is particularly true in the case of Web application security. The basic philosophy in security is that trust is established through the ability to enforce control over a resource. In the case of Web applications the resource is a function of the data being served. Protect the function and you protect not only the data but also the Web application. While there are different approaches to securing a Web application there are really just two philosophies: 1) establishing trust between the Web applications and the user and 2) establishing trust between the Web applications and data or resources being served back to the user. This will be defined more thoroughly in the rest of the paper.

Establishing and Enforcing Trust with the User

User trust is based on recognizing identity and granting authority. The Web is an anonymous medium, making the establishment of identity imperative. Validation of a user's identity is required in order to trust their identity. Authentication methods such as passwords, PKI, digital certificates, and cookie signatures are control techniques used for identity recognition.

User authority is established by limiting an identified user's access to information based on what resources that user can access, and when and how that user may access them. This is usually accomplished through a combination of role-based access control and encryption. The goal of these methods is to ensure that the resource trusts the user by ensuring that an authorized user can access only specific applications and resources.

Methods to Establish User Trust

Authentication – User identification or validation by use of password quality control, PKI, and/or digital certificates.

Access control – Declares user authority by defining who can access the application or resource. Access control can be coded into the application or provided externally to the application.

Intrusion Detection – Aimed at stopping threatening user behavior, based on a knowledge base of expected attacks. Some Intrusion Detection Systems (IDS) have adaptive learning capabilities that act after the fact. IDS sits at a layer external to the application flow and attempts to enforce rules that govern user behavior.

These methods establish trust between the resource and the user up to the point of access to the Web application. However, these methods do not address security after the user has been granted access and while they are using the Web application. Web applications are usually created such that a skillful user can "trick" the application into granting him or her authority beyond what they are authorized to do.

Establishing Trust with the Web Application

Trust between the resources being accessed and the Web application is based on authorized access, as previously discussed, and limiting what the application can directly access. The common methods used to accomplish this are access control and file permissions. The problem with these methods is that the Web application is actually a process composed of many integrated parts that own all of the data they serve. This process (the application) requires system resources that can go beyond simple data access.

The behavior of a Web application is defined during its design. However, a Web application can often be used in ways that take it beyond its design scope. A hacker can use the application to access resources that the application was not designed to access, but which it is not prevented from accessing. An authorized user can break an application's trust while using it in an authorized manner. The only way to ensure that resources can trust a Web application is to ensure the integrity of the application or the integrity of the inputs to the application.

Methods to Establish Application Trust

Access Control - URL or Web page access based on authenticated users is specifically known as Web Access Control. The main purpose is to control which pages a user can or cannot access. One of the drawbacks is that Web Access control has no control over enforcing Web application logic. If a malicious user finds a way to pervert an application they may be able to bypass the Access Controls.

Manual Fixing – Addresses problems within the application source code. This solution requires that programmers have access to the source code (not possible with third party code), know how to fix a problem, and, of course, even know that the problem exists. With third party code, manual fixing would include monitoring for published vulnerabilities and immediately applying all patches, for each program, as the vendor releases them. The fact that this is still not done, even by large enterprises with large security groups, and even given that we all know it could prevent many attacks, attests to the difficulty in keeping up with patches.

Automatic Web Application Firewall – Aimed at application behavior, where the application interacts with a security solution to define the policy. The policy has knowledge of the application and how it is designed and functions. The application defines the security policy, thus if one makes a change to the application, the security policy is automatically modified. Automatic Web Control and Security will be discussed in more detail later.

Web Application Components

As has been discussed, a Web application is more than the underlying Backend Systems that resides on an operating system. It is the synergy and interaction of that code with the code that allows it to provide service, data, or a function to a Web user.

The main components of a Web application were shown in the introduction. They are described below.

User Interface Code – Written by a third party or in-house (custom code)

The User Interface Code is the presentation layer of the Web application. This code creates the look and feel of the site. Not only does it present code that interfaces directly with the server software, it also provides client-side code that can generate semi-automated features and responses on behalf of the user directly to the server. All user input is processed through the User Interface Code.

With the exception of third party code, which is patched, Web sites have been reliant on manual fixing as the only real solution to secure the interface. Developers must make sure that data input can be properly handled, so Web site security is limited to the skill level and security motivation of the developer. Most Web developers are not application security experts and are often not aware of some of the common exploits hackers may attempt against a site. Developers frequently introduce unnecessary security risks through features that appear to save time or promote ease of use, such as by using hidden fields to store important information like prices or leaving a backdoor open. In some cases there are security issues with libraries and code drivers that the developer has no control over. Keeping the libraries and support tools up to date is usually the responsibility of the administrator.

Common code drivers are: HTML, Java, JavaScript, ActiveX, and Visual Basic. The user interface code may be written by a third party, or in house with Graphical User Interface (GUI) tools. This code interfaces directly with the User, Web Server software, and the Frontend System.

Web Server Software – Written by a third party

The Web Server supports the physical communication between the user's browser and the applications that the user needs to access. It handles all of the in and out bound http/s requests, manages user session (timeouts, session state with cookies) and tries to make sure all sessions are properly processed.

Since virtually all companies use Web Server Software provided by third party vendors, such as Microsoft IIS, iPlanet, and Apache, they must rely on the vendors to supply error-free code and provide patch solutions when problems are identified. While a company's administrators can make sure the Web server software is configured properly, they are bound to work within the site environment they are supporting. In some cases the optimal security configuration may conflict with optimal usability. Because administrators are often evaluated on usability-related criteria such as site throughput and uptime, they are not always incented to promote the most secure configuration, and often they fail to do so. Additionally, even if a company is fortunate enough to have a site administrator who happens to be a security guru – a rare occurrence – they will still be limited by the availability of vendor patches, as patches are only released after a problem has been discovered and publicized – often days or weeks afterward it was discovered.

Web Server software interfaces directly with the Frontend systems, operating system, the Network, and the User Interface Code.

Frontend Systems – Written by a third party or in-house (custom code)

The Frontend System interfaces directly with the User Interface Code, the operating system, and the Backend Systems. Under normal circumstances a user will not interface directly with this layer; however, the data that the user passes to the User Interface Code will be passed through the Frontend System.

The Frontend System can be a combination of third party code and in-house developed custom code. In order to maintain the security of the Frontend System, operations support staff must keep current on the specific language bugs, exploits and patches and along with the libraries and development support tools. Frontend System developers need to make sure that invalid data and Meta code can be processed correctly. Vendor examples of Frontend System are ColdFusion and WebLogics. Custom code examples of Frontend Systems are Common Gateway Interchange (CGI), JSP and ASP.

Typically, CGIs are written with high-level languages such as PHP, Perl, C/C++, Python, and shell scripting scripts languages. Java Server Pages (JSP) written in Java and Application Server Pages (ASP) written in ActiveX are becoming more popular.

Backend Systems – Written by a third party or in-house (custom code)

The Backend Systems are the real driving piece of any Web application. The business needs drive the development of the Backend Systems, and the resulting code provides the business function, such as facilitating online transactions. User input is passed to this level via the User Interface Code and any associated Frontend System. The Backend Systems interface directly with the Frontend System, the Operating System, the Database, and possibly the data itself.

The Backend application software may be generic off-the-shelf products; a customized off-the-shelf product (a combination of in-house developed and third party code), in-house developed code, or complicated systems built using specialized hardware and commercially tailored software such as SAP environments. Examples of Backend systems are application servers and e-commerce suites. They may be as complex environments such as Banking and Financial services, or less complicated environments such as online informational services like search engines and information retrieval.

Backend Systems are similar to Frontend Systems in that they may be a combination of third party code and custom code, often multiple groups are responsible for its security. Operations staff and application managers are responsible for keeping vendor applications current with patches and configurations. Development staff must make sure that any in-house written code is secure and can properly process Meta data and invalid code. However, operations staff are again responsible for support of the development languages, support tools and libraries used by development. And with complex hardware and software solution customers have to rely on vendor support for not only the integrity of the software, but also the integrity of the vendor service.

Database – Written by a third party or in-house (custom code)

A database is a collection of data that is organized so that its contents can easily be accessed, managed, and updated. The Database controls the data that the Web application uses and manages which data should be served.

Companies almost always use a database that is third party customized code such as MySQL, Oracle, or DB2. Once again, the database can require support from many areas. Operations staff should make sure the proper system parameters are set along with proper file permissions. In addition, any operating system support tools need to be properly configured and patched. Database administrators need to make sure the database is properly configured and up to date with the appropriate patches. Database Application developers need to make sure that the query language can properly handle invalid data and Meta code.

The Database interfaces directly with the Backend Systems, the data, and the operating system.

Web Application Threats

An attacker can target a Web application at a variety of different points. An attacker need not have direct access to the support infrastructure since the browser itself serves as an entry point into the Web application and its dataflow.

When a user makes a request at the Web interface, a flow of data is started. As data flows through the Web application layers that were described above, each layer must handle the request properly to avoid becoming a potential breach point. As was noted in the previous section, there are potential vulnerabilities that can exist at each layer. Figure 2 below shows these potential breech points. This section will describe some of these different vulnerabilities in detail and look at attacks a user can successfully accomplish when they are present.

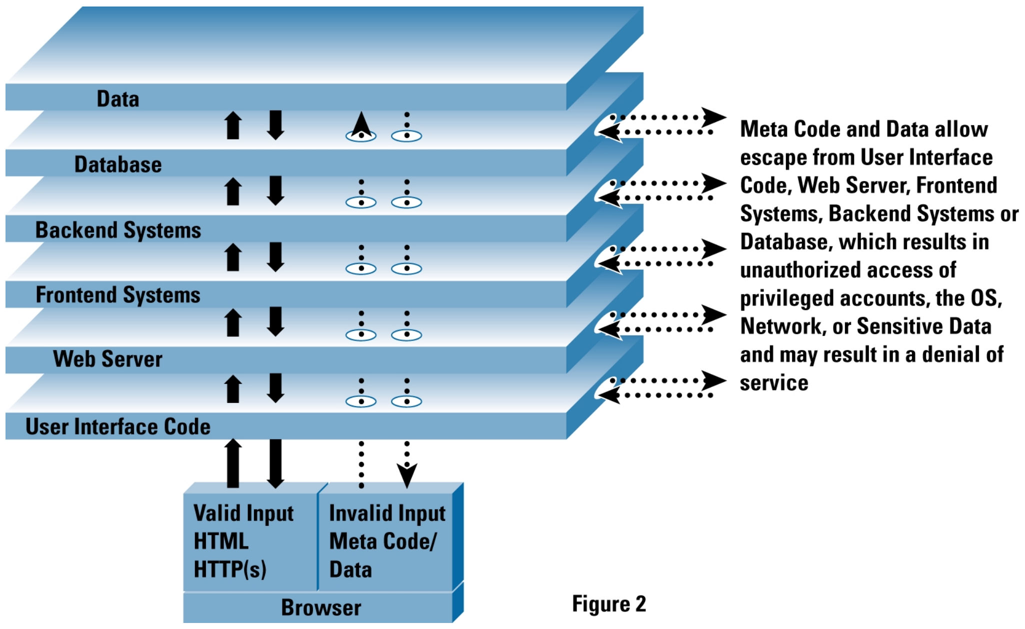
Data
Database
Backend Systems
Frontend Systems
Web Server
User Interface Code
Valid Input
HTML
HTTP(s)
Invalid Input
Meta Code/
Data
Browser
Meta Code and Data allow escape from User Interface Code, Web Server, Frontend Systems, Backend Systems or Database, which results in unauthorized access of privileged accounts, the OS, Network, or Sensitive Data and may result in a denial of service

Figure 2

Client Side Tampering

Web application vulnerabilities can be identified through *code scanning*, which gathers information about client side code, such as third party code used, the configuration of software, and even the languages used to drive the Web application interfaces. Hackers can gather information through freeware tools that will determine weaknesses in the code.

Code that runs in the user's browser is not private. A user can scan the client source code looking for weaknesses and points of attack. Many sites contain code that the attacker can alter to attack the site; in effect using the site's code against itself. This is not limited to just HTML. JavaScript or any other client side code that runs on the user's browser is also vulnerable.

In addition to the code run in the browser, any data that is stored on the client can be examined altered and played back to the server. An example of this kind of attack is called *cookie poisoning*. HTML is stateless – it does not have the ability to retain information from one session to the next. Cookies were added to the HTML standard so that session information could be stored on the client workstation, in the user's browser directory. Cookies enable a server to recognize a user from a previous session. However, because the cookie information is stored on the user's desktop, savvy users can alter the information stored in the cookie. For example, a malicious user could alter the cookie and impersonate another user.

Something not so obvious is the ability to alter server-side HTML values with client-side modifications. Hidden values are a feature of HTML that allows Web developers to change values quickly. Unfortunately, the Web server does not validate the source of the change, making hidden fields a risky place to store critical information. Through a technique called *hidden manipulation* a user can download the HTML source page that contains a hidden field, change the value and play it back up to the server. The server will accept the change and process it as a valid request. Some shopping cart programs use hidden fields to store price information. This technique is behind some recently publicized hacks where attackers have been able to successfully charge themselves $0 for expensive items at Web e-tailers.

Forceful browsing is a technique where a user access a Web page they should not be able to get to by typing the URL directly into the address line instead of going through a link from the entry page. Many times, the access controls that would stop the user from accessing the page through the appropriate link do not stop forceful browsing.

Third Party Code and Vendor Tools

In the cases of third party code and vendor tools, the site owner cannot fix things manually because the site does not own the source code. The site owner's security is only as valid as the integrity of the vendor tools provided.

Vendors try to make their tools easy to install and use by providing out-of-the-box configurations, including default passwords and settings. Sometimes administrators fail to change these defaults, leaving themselves vulnerable to *third party misconfigurations*. *Third party misconfigurations* are easy for hackers to discover through *code scanning* and they pose a high risk for attack if not changed. Leaving default passwords and settings unchanged can be considered an open invitation for attacks, which can result in theft of data and potential holes for Denial of Service (DoS*)* attacks.

Known vulnerabilities, also called vendor bugs, present a similar opportunity for hackers. With vendor bugs, the risk is the integrity of the tools and how quickly the vendor responds to any identified problems. The key issue with *known vulnerabilities* is the vendor's response. Any time that elapses between when a bug is identified and when a patch is issued provides a window for attack, so the speed with which a vendor delivers patches and the speed with which the site administrator applies the patches is key. Some vendors are very responsive to security issues; others respond more slowly (causing a problem know as patch latency). Sometimes, a patch may not be issued at all (called zero patch availability).

Application and Language Issues - Invalid Data and Meta Code

Invalid data may be entered, but is not what is expected by the application. If the application receives invalid data, it may not be able to process it, leading to unpredictable results. In some cases the underlying compiler can have a design flaw that creates problems. For example, in the case of C/C++, when input values are larger than expected, a reaction called *buffer overflow* can crash the application (segmentation fault) and sometimes the operating system. In either case this may result in a number of different types of dangerous scenarios depending on the configuration of the system, including the ability for the attacker to access restricted data. Improper memory management (access violation) is another shortcoming that may allow access to restricted data or the operating system.

Parameter tampering is manipulating the URL string by changing the parameter to contain invalid data, for example changing a value to something very large, very small, negative when expecting positive values, wrong data type (text when expecting integer), or removing the parameter altogether. Any and all of these can have adverse effects on an application with unexpected results.

Meta code is control or escape code that is embedded inside of input data to allow control commands to be passed to the application or even the operating system. In effect, Meta code allows the user to control the application at a meta-level. Simple examples include command strings preceded with an exclamation mark "!" passed to Unix shell script, or percent sign "%" passed to a perl script that would then run the command at the operating system level. Use of Meta code can allow users to execute attacks through a technique called *stealth commanding,* which can allow the attacker to gain access to the operating system and gain access as a privileged user, crash the site, etc. The User Interface Code itself has a known similar weakness where use of special characters allows whole programs to be fed into input fields, a technique called c*ross site scripting (CSS)*. By exploiting this weakness, attackers can sometimes access sensitive corporate information that was not intended for access through the site. Different languages react differently to different escape characters and sometimes application sensitive escape characters may be part a valid URL string. Programmers are often unaware of the vulnerability that input fields can present. Meta code is not limited to specific characters but includes control strings that can act as switches that can trigger hidden (undocumented) *debug* code and *backdoors*.

Denial of Service Attacks (DoS)

Many people associate the term Denial of Service (DoS) attack with the Distributed Denial of Service (DDoS) attacks that have been launched against high profile sites over the past couple years. While DDoS attacks are often highly publicized, there are actually many types of Denial of Service Attacks that can be more easily launched by individual hackers. Although Denial of Service are attacks commonly thought of as attacks directed against a Web server's operating system, any attack that causes a site to deny service to its users is a Denial of Service attack; these are not limited to just the operating system. An attack that disables any component of the Web application will also result in a Denial of Service, specifically an Application Denial of Service, negatively impacting a site's Web presence. In fact, the host operating system may be running, but if the Web application cannot function, then the Web presence is in effect disabled. For example, if an attack is focused at the database and shuts it down, then the functionality of the Web server is removed in that the data can no longer be accessed.

Web Application Threat Summary

Table 1 sums up the exposures just described, illustrates the attack techniques and some of the consequences of the threat not being addressed.

Table 1 Threat Summary

Threat Category	Description	Consequence
Code Scanning Server/Client	Browsing source code	Learn vulnerabilities
Cookie Poisoning	Changing cookie content	User impersonation
Hidden Manipulation	Changing hidden HTML fields value	eShoplifting
Forceful Site Browsing	Use URL address line	Access sensitive data
Third Party Misconfigurations	Default or improper software configuration	Access OS or data
Identified (Known) Vulnerabilities	Published vendor bugs	Access OS, crash server/application/database, access sensitive data
Buffer Overflow	Overflow field input	Access sensitive data, or crash site/application
Debug Options & Backdoors	Change code setting	Access code/application as developer or admin
Parameter Tampering Server/Client	Removal or alteration of expected parameter fields	Access OS or sensitive data
Stealth Commanding	Use Meta code	Access OS or control application at OS level, site defacement
Cross Site Scripting	Use URL Meta code to insert Trojan code	Server-side exploitation, access sensitive data
Application DoS	Invalid data input	Crash server/application

Detail Web Application Threats: By Threat Category And Web Application Component

With the exception of code scanning, all of these threats result from either the use of Meta code or invalid data inputs. A single breach at one layer might not pose an immediate threat; however, it may provide information that can cause exposure at another layer, which can result in an immediate threat. For example, a breach at the User Interface layer might not reveal any data, but it might provide information about the underlying application giving an attacker enough information on how to pose a successful attack.

Table 2 and Figure 3 depict five attack points in the Web application where an attacker can attempt an exploit, before the endpoint data is even accessed. At each level there is a strong potential that an attacker can either directly or indirectly cause a site outage by taking down the server and/or the application. Worse yet, a skilled attacker can access data and resources that are outside of the application scope using methods not normally detected, which means they would neither be stopped nor logged.

Table 2
Five Attack Points

		Attack Points				
		User Interface	**Web Server**	**Frontend Systems**	**Backend Systems**	**Database**
Threat Category	Code scanning		Yes			
	Cookie Poisoning		Yes	Yes	Yes	
	Hidden Manipulation			Yes	Yes	
	Forceful Site Browsing		Yes			
	Third Party Misconfigurations		Yes	Yes	Yes	Yes
	Known Vulnerabilities		Yes	Yes	Yes	Yes
	Buffer Overflow		Yes	Yes	Yes	Yes
	Parameter Tampering		Yes	Yes	Yes	Yes
	Stealth Commanding		Yes	Yes	Yes	Yes
	Cross Site Scripting	Yes				
	Debug Options & Backdoors		Yes	Yes	Yes	Yes
	Application DoS		Yes	Yes	Yes	Yes

Securing A Web Application

Figure 3 illustrates the input flow request a user may make from a browser. Notice that each component of the application includes a potential failure point that could be exploited with invalid data or Meta code if it is not protected. Fixing vulnerabilities at each layer would require at least five separate solutions. Addressing security for each separate component of the Web application is not only tedious, but fixing problems by making manual programmatic changes has the potential to introduce more issues. Additionally, because no one single group within a company owns all of the Web application components, and each component requires a different type of security, a great deal of coordination is required to secure the application manually. A solution that checks the data at the Web entry point and verifies that it matches the expected input before allowing it to be passed to the Web application will invalidate these attacks and provide a single point of protection, making individual back end protection solutions redundant.

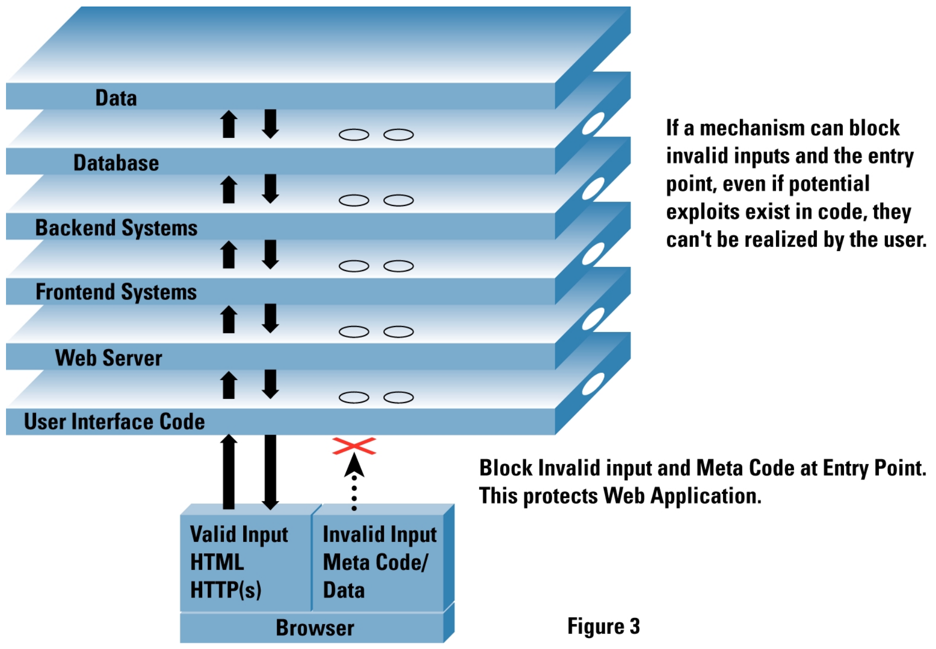
Data
Database
Backend Systems
Frontend Systems
Web Server
User Interface Code
If a mechanism can block invalid inputs and the entry point, even if potential exploits exist in code, they can't be realized by the user.
Block Invalid input and Meta Code at Entry Point. This protects Web Application.
Valid Input HTML HTTP(s)
Invalid Input Meta Code/ Data
Browser

Figure 3

Even though there are many components to a Web application, a successful security solution needs to treat the application as a whole in order to be effective. Because the vulnerabilities can all be exploited by input given to the Web application by an attacker, a solution that checks user input against expected norms before allowing it to be processed will protect the application against the many different attacks. However, for this solution to be effective, it has to be able to work automatically and respond automatically to changes in the application. Any solution that requires administrators to be aware of each potential breech and to set acceptable parameters individually will run into the same problems inherent in fixing each vulnerability individually. Trying to second-guess what an attacker will attempt is not effective, because no administrator can know all of the potential user attacks before they are attempted. Defining explicit access rules is an important step, but it does not address how the behavior of the application can be manipulated. An effective Web application security solution will address Web application behavior at the front end by only allowing valid input as defined by the application. This ensures Web application integrity so that internal application issues cannot be exploited.

Summary

A Web application is composed of many components, and users have both direct and indirect access to the internal workings of the Web application. A user's input flows from their browser into the User Interface down to the Backend Systems and beyond. In turn Web content flows to the user all the way from the Backend Systems to the User Interface where the user interacts with it on their local browser. This interaction exposes the Web application to malicious attacks. Insuring the trust and integrity of a Web application requires a security policy based on the inner working and content presentation of the Web application.

Table 3 summarizes the components of a Web application, potential exploits hackers can target, and the risks those attacks present.

Table 3 Detail of Web Application – Summary

Name	**User Interface Code**	**Web Server Software**	**Frontend Systems**	**Backend Systems**	**Database**
Examples	HTML Parser Client side code JavaScript ActiveX Visual Basic	IIS Apache iPlanet	PHP Perl C/C++ Python ColdFusion WebLogics	Online Finance Online Medical Customization Engines eTail (eShopping)	Oracle MySQL
Support Staff	Web Developers Web/App Developers Web Master	Operations - System Admins Web Master	Operations - support Web/App Developers Web Master	Application Developers Operations - Application Support, Web Master	DBA Application Developers
Exploits that can be used by an attacker	Meta Code, Invalid data, HTML bugs	Known Vulnerabilities, Meta code, Configuration	Meta Code, Invalid Data, CGI bugs Configuration	Known Vulnerabilities, Invalid Data, Configuration, Meta Code	Database Bugs, Configuration, Meta Code, Invalid Data
Directly at risk	Sensitive Data Other Sites	Sensitive Data OS Network	Sensitive Data OS Network	Sensitive Data OS Network	Sensitive Data OS
Application DoS		Yes	Yes	Yes	Yes

Security for a Web application is usually addressed as an afterthought, not a design issue. And unfortunately, Web application security is often addressed with the wrong methods. The application components have different vulnerabilities and must be addressed as a whole in order to be secure. Each component of the Web application from the browser all the way to the end data must be secured in order to be effective. While existing security products provide important functionality, most do not address the security issues inherent in Web applications. The uses and limitations of several types of security offerings are shown in table 4.

Web Application Solutions

Table 4

Threat Category	Access Control	Manual fixing	IDS	Automatic Web Application Firewall
Cookie Poisoning	No	Yes	No	Yes
Hidden Manipulation	No	Yes	No	Yes
Forceful Browsing	Yes	No	No	Yes
Third Party Misconfigurations	No	No	No	Yes
Known Vulnerabilities	Yes	Yes	Yes	Yes
Application Buffer Overflow	No	Yes	No	Yes
Parameter Tampering	No	No	No	Yes
Stealth Commanding	No	Yes	No	Yes
Cross Site Scripting	No	Yes	No	Yes
Debug Options & Backdoors	No	Yes	Yes	Yes

Automatic Web Application Firewall solutions address Web application security properly because they provide the security policy for each application based on information extracted from that application. This allows the solution to address the behavior of each specific Web application automatically. The security policy is not defined by a list of known attack patterns, but how the application functions. By observing the application, the Automatic Web application firewall solution gains internal knowledge of the application and what it expects, and then uses this information to determine whether the input and the application behavior are acceptable, eliminating the risks associated with application breeches. The solution generates policies that are in sync with each Web application, automatically defining these policies before the user can even attempt an attack. The policies are unique to each application, and attacks are stopped before users are allowed to enter the Web application on the server.

Conclusion

Ebusiness has enabled new and exciting uses of the Web, from online customer self-service applications that save money and promote customer intimacy to business-to-business transaction software that streamlines relationships with suppliers and partners. However, every application that links corporate information and resources to the Web gives hackers a new potential entry-point into your organization. In the race to develop online services, these Web applications have often been deployed with minimal attention to security risks, with the result that most corporate sites are surprisingly vulnerable to hacking or industrial espionage.

As we have seen, the applications that provide online functionality are actually complex layered entities, and potential vulnerabilities exist at many levels, not just the user interface or the Web Server. Fixing all of these vulnerabilities manually in each Web application a company uses is not only an extremely time consuming endeavor, but often not possible. Most common security tools that companies have in place, such as firewalls, intrusion detection systems, access controls, and antivirus software, do not address the issue of application vulnerabilities.

EBusinesses need a security solution specifically designed to protect Web applications and the issues they present. As shown above, most application vulnerabilities rely on a hacker's ability to input invalid data or malicious code into the application, through the variety of techniques described in Section V. This means that a security solution that sits between the user and the application(s) and checks the validity of user inputs before passing them into the application dataflow, can serve to prevent a significant variety of attacks. To be effective, however, such a solution needs to provide several features:

- Ability to understand individual applications – The solution needs to be able to analyze each Web application individually. Because each application is unique, and is built on a unique collection of customer and third party code, the security solution needs to be able to understand each application, instead of relying on predefined concepts of hacking attacks.
- Automatic policy generation – The solution needs to use its understanding of the application to develop appropriate input policies automatically. Any solution that requires an administrator to generate the policy or rules manually requires that he or she know every potential vulnerability in every application, and is as time consuming and failure-prone as attempting to manually fix each problem.
- Automatic updates – For the same reason, the application security solution needs to be able to recognize and adapt to any changes in the Web applications, whether they be 3rd party or proprietary applications.
- High accuracy – Because all unacceptable inputs are stopped, the security solution must be highly accurate, to avoid stopping legitimate uses of the application.
- Scalable – Because Web applications are often built for high user volumes, the security solutions need to scale to support heavy usage requirements.
- Automated Forensics –The solution needs to automatically generate the forensics required to analyze the source of the attack and provide the details needed to aid in catching the hacker.

Fortunately, as eBusiness continues to grow, there is a new breed of security solutions that target Web applications. These solutions vary significantly in levels of automation and depth of protection, and users need to examine vendor offerings carefully before purchasing.