access ON DEMAND

WHITE PAPER

Protecting Web Applications from Attack and Misuse

Introduction

The threat profile facing enterprise organizations has undeniably shifted from network-layer exploits to more formidable attacks against applications, primarily Web and Web services applications. This radical change has been recognized by numerous IT security vendors, which have rushed to deliver products that shield Web applications from a new generation of attacks.

Not surprisingly, this flurry of activity has resulted in general marketplace confusion — not only about the threat facing mission-critical Web applications, but also about the most appropriate technology to protect these vital online resources. This white paper will provide clarity regarding the actual requirements that must be satisfied by any viable Web application security solution.

CITRIX®

Table of Contents

Inspect Application Communications — Not Just IP Packets

Protecting an application from attack requires a complete understanding of all application communications. Unless a device can "see" the same data as the application it is protecting, it will be unable to identify application-layer threats. This means that to secure any common Web-based application, a security device must perform a full deconstruction of the HTML data payload, as well as track the state of each application session.

It is technologically impossible for any device to understand application communications or analyze application behavior via the deep inspection of IP packets, either individually or reassembled into their original sequence. Network firewalls and intrusion prevention systems (IPS) are useful for validating the format of application header information to ensure standards compliance.

In addition, network-level security devices may detect a small number of known, easily identifiable attacks by looking for pre-programmed patterns (i.e., attack signatures) in an HTTP stream.

Unfortunately, without any awareness of the HTML data payload or session context, devices that rely exclusively on the inspection of IP packets will fail to defeat the vast majority of application-layer exploits. For example, IP packet inspection will not detect a hacker who has maliciously modified parameters in a URL request.

The deployment of Web application firewalls is the only proven method of protecting critical Web applications. Application firewalls operate at the application layer, not the network (or IP packet) level.

They efficiently terminate all application sessions and perform a full, bi-directional parsing of all application data. By inspecting the actual HTML communications and understanding the context of all client requests and application responses in which these are sent to a Web server, an application firewall can enforce correct application behavior and block malicious activity. For example, only application firewalls can detect and prevent the injection of malicious scripts into applications via HTML form fields — a common means of executing a cross-site scripting attack.

Detect and Defeat Encrypted Application Attacks

Virtually all Web applications that process confidential customer or corporate information utilize Secure Sockets Layer (SSL) encryption to protect both the confidentiality and integrity of data while in transit.

Although SSL security has become a linchpin technology for e-commerce Web sites, it has also provided hackers with a useful tool to evade detection. It is often trivial for a hacker to establish an SSL-encrypted session with an Internet-facing Web application. With an encrypted session established, a hacker can launch an attack against the application knowing that the SSL tunnel will shroud all malicious activity. Intermediate network-layer security devices (e.g., firewalls and intrusion-prevention systems) do not participate in the SSL encryption scheme and are, therefore, relegated to blindly forwarding SSL traffic without inspection.

Application-layer security can only be performed if SSL-encrypted traffic is decrypted into its original clear text form prior to inspection. This requires full participation in the SSL encryption scheme. Even after decryption and security inspection, sensitive application environments may require re-encryption before forwarding traffic to the destination Web server — thus ensuring end-to-end data confidentiality.

Protect the Application Infrastructure and Application Users

Complete application security demands protection of all elements of an application infrastructure (e.g., server operating system, application program and back-end databases), as well as users of the application. However, application security requirements are often defined too narrowly, including only protection of the application program and application data. Trust relationships with users must also be closely guarded to ensure the continuing business viability of the application.

Most application-layer exploits can be classified as either:

1. Attacks attempting to compromise the integrity or availability of application resources; or
2. Attacks aiming to compromise the trust relationship between an application user and the application.

Types of Hacker Attacks	Successful Web application security solutions must defend against each type of exploit.
Buffer Overflow (attacking the Web application)	A common type of input validation attack that overflows a buffer with excessive data. Successfully executed, this attack allows the hacker to run a remote shell on the machine and gain the same system privileges granted to the application being attacked. Code Red and Nimda are well-known examples.
SQL Injection (attacking the back-end database)	An input validation attack that sends SQL commands to a Web application. The commands are then passed to a back-end database. When successfully executed, SQL injection attacks can provide a hacker access to a sensitive information store.
Cross-site Scripting (attacking the trust relationship)	An attack that results in an innocent application user unknowingly executing malicious code devised by a hacker and granting that code the same privileges as would be given to the trusted Web application. Successfully executed, a cross-site scripting attack can result in a hacker stealing the identity of innocent application users.

Defeat Zero-Day Attacks

There are two types of zero-day attacks:

1. Attacks exploiting one or more security vulnerabilities in custom application code
2. Attacks exploiting one or more security vulnerabilities in packaged software applications for which the software vendor has not yet released a patch.

An effective application security solution must detect and defeat all forms of zero-day attacks.

Any network- or application-based security device that detects malicious activity based on either attack signatures or event correlation will fail to protect against zero-day attacks. By definition, signature-based ("black list") solutions cannot detect an attack for which there is no attack signature.

Application security managers should never rely on a security vendor to identify immediately every known application attack, develop a precise attack signature (i.e., one that will not generate false-positive security events), and distribute a new signature list to all customers.

Correlation-based security solutions attempt to correlate illegal client requests with their resulting application server error codes. Once a high degree of correlation is determined, the security solution adds the observed client behavior to a list of unauthorized actions. This list is then used to screen out all unauthorized application requests. This approach is fatally flawed because of the inevitable delay incurred when relying on multiple occurrences of similar request/response behaviors to determine correlation.

When a zero-day attack emerges, organizations rarely have the luxury of encountering numerous instances of the threat before initiating blocking action. Correlation-based security devices introduce an unacceptable threat window, and do not block early instances of zero-day threats.

The only viable defense for zero-day attacks is to utilize a positive security model that understands and enforces correct application behavior. Establishing an application "correctness" model enables a security device to distinguish legal ("good") application traffic from illegal ("bad") traffic in real time.

Application communications falling outside of this positive security model are immediately blocked. There is no need for attack signatures or correlation of multiple application events.

A well-designed positive security model will accurately identify and block all zero-day attacks in real-time.

Cloak Application Infrastructure Elements

Many attacks against Web applications are custom-crafted and designed to exploit security vulnerabilities in one or more elements of the application infrastructure (server operating system, application platform, or Web application code). Effective reconnaissance can help the hacker focus his attack methodology and target a smaller number of potential security weaknesses. In addition, greater knowledge of application design and implementation can enable a hacker to craft more damaging attacks, since the hacker can not only better identify security vulnerabilities, but also locate valuable application resources and data.

Best practices for Web application security mandate not only strong defenses against application-layer attacks, but also deep cloaking capabilities to obfuscate or mask details of the application being surveyed. At a minimum, a Web application protection solution should include the following cloaking capabilities:

- **Remove all unnecessary server response headers.** This increases the difficulty in fingerprinting the type of Web server (e.g., IIS or Apache).
- **Rewrite all application URLs.** Rewriting URLs can preserve application accessibility, while preventing the publication of potentially sensitive information (e.g., application directory structures and internal domain naming). For example, rewriting internal URLs before they are published to clients makes it more difficult to determine the location of back-up files and data directories.
- **Remove HTML comments.** Developers often include comments in the application code that can be used to deduce application design details. For example, an application may include comments that document application flow or point to a particular database table.
- **Encrypt cookie names and values, URLs and hidden form fields.** Each of these elements can be used to fingerprint the type of application server and narrow the list of potential attack vectors. The use of strong encryption makes identification of application components dramatically more difficult.

Prevent the Leakage of Sensitive Corporate or Customer Data

Application security is a bi-directional challenge. Only inspecting application data in-bound to the Web server is insufficient. Unquestionably, blocking attacks from untrusted clients is critical, but an application firewall must also be able to inspect application data in Web server responses.

The objective of many Web application attacks is to steal sensitive customer data (e.g., credit card numbers, account IDs, Social Security numbers).

An organization can never be completely assured that all avenues to a successful application-layer attack are blocked. The prudent strategy is to actively guard against the leakage of sensitive data in application server responses. When implemented smartly, the application firewall can serve as a last line of defense against attacks targeting confidential data.

The protection of sensitive customer or corporate data must encompass the following capabilities:

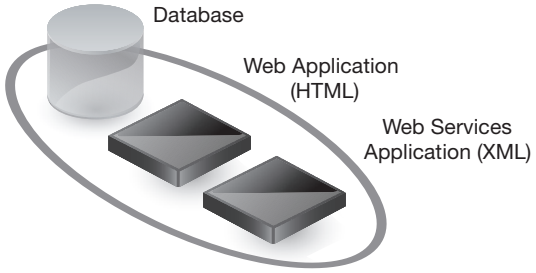
- **Inspect more than just HTTP Headers.** The entire data stream should be inspected for the presence of restricted data objects.
- **Ensure precision when matching data objects.** For example, credit card number protection should not automatically block all 16-digit numbers simply because they conform to the type and format of credit card numbers. Objects should be tested for validity, with algorithmic matching if necessary, before real-time action is taken.

The objective of many Web application attacks is to steal sensitive customer data.

- **Provide an option to transform matching data objects.** Instead of simply blocking data objects, e-commerce sites may, for example, want to reflect the last four digits of a customer's credit card number for verification purposes.

Don't Block Benign Traffic

Web application security is typically deployed to protect an organization's most business-critical applications. Frequently, protecting these important applications presents the greatest challenge due to their criticality. For instance, an application may be heavily relied upon to generate revenue or to process sensitive customer transactions. The potential for false positives (i.e., blocking benign traffic), is often deemed intolerable in these application environments.



Application Infrastructure

Web application security is typically deployed to protect an organization's most business-critical applications.

For example, mistakenly blocking transactions for a large number of legitimate clients may cost a financial-sector enterprise untold thousands of dollars in both real revenue and loss of customer goodwill. Not surprisingly, the foremost concern for the line of business manager who owns responsibility for the business success of the application is simply "do no harm."

The mitigation or complete elimination of false positives requires:

1. true application-layer traffic inspection
2. full communications context
3. a semantic understanding of all application data

The absence of just one of these important elements can easily result in blocking of benign traffic. SQL injection exploits, for example, are only harmful when they are contained within application input fields and incorporated into database query commands. However, if a SQL command or SQL special character were to appear in a non-input portion of the application it would pose no real threat to the application and should not, therefore, be blocked. Knowing the semantics of the application stream enables an application firewall to make intelligent security decisions.

Similarly, if an application utilizes client-side JavaScript to modify application cookie names, an application firewall cannot simply block a modified cookie under the assumption that the client request is malicious in nature. Performed correctly, the application firewall must understand the full context of client-to-application server communications to precisely determine when a modified cookie value or cookie name indicates legal application behavior representing an outright attack on the application session.

Deploy Consistent Security for All Applications

Most large enterprises have multiple Web applications requiring security. Although each application may possess unique application logic and support different traffic flows, they may be vulnerable to a common set of application-layer threats. For example, Web applications should never accept SQL commands in client-supplied fields and then use this data in the generation of data base queries. A common threat, such as a SQL injection attack, drives the need for consistent security defenses across applications.

Common threats to applications can be handled by "global" security settings. However, there may still exist a need to define per-application security rules. For example, a global security policy that prohibits any modification to data in a hidden form field may be effective in protecting one application, but may break an application that uses client-side JavaScript to legally modify form field data within the client's browser. In this scenario, different policies would be necessary.

Once policies are defined and put into production, policy management and data collection should remain granular. Even with global policy enforcement, security managers must be able to manage security defenses per application and perform all management functions per Web application. For example, security logging and reporting should be segregated by application.

Virtualization of application-security policies, with both global and per-application attributes, delivers a practical and flexible solution for protecting multiple Web applications.

Adapt Policies for Dynamic Application Environments

Many of today's Web applications utilize client-side JavaScripting to programmatically generate dynamic content within a client's browser that is then returned to the Web application. Prime examples of application content which may be legally modified on the client include URL parameters, application session cookies and form field data. Although this data is generated dynamically, it is done so within the bounds of the application logic and must be accepted as valid input to the Web application. A pure positive security model that is unable to accurately anticipate and accommodate dynamically-generated content from the client will potentially block legitimate application traffic.

There are two approaches to "covering" an application so that dynamic content is properly handled while still enforcing strict policies governing client-to-application behavior:

1. Application Spidering
2. Adaptive Application Learning

Application Spidering

Application spidering is an untested technology that attempts to map an application exhaustively to learn and record all its objects (e.g., files, images, scripts) and application flows. Based on the mapping generated from this application spidering, the protecting entity that uses spidering can determine whether each client request is legal and is requesting authorized resources.

Spidering is only effective when input parameters submitted by a client are static. This is not the case with client-side JavaScript. Once JavaScript is introduced into the environment, an application spider cannot determine the safeness of client inputs. Consequently, application firewalls that depend on spidering are forced to relax their security posture such that they cease inspecting client values that are dynamically generated. This exposes the application to attack and misuse.

ADAPTIVE APPLICATION LEARNING

The only reliable method of protecting an application utilizing client-side JavaScript is to learn correct application behavior through a Bayesian analysis of representative traffic. In other words, analyzing the content in actual client requests to determine how JavaScript modifies application parameters (e.g., URLs, form fields, and cookies). Once correct application behavior is learned, policy recommendations should be automatically generated and manually applied (i.e., human in-the-loop) so that parameters dynamically modified by an application client is recognized and permitted per the security policy.

Conclusion

Application-layer attacks against business-critical Web applications are the most serious IT security threat facing enterprises today. A successful application attack can not only damage key Web servers, but also provide a hacker with a direct conduit to internal databases containing sensitive customer or company information. Further complicating the challenge facing security managers, nearly all Web application attacks are undetected by traditional network security devices, such as network firewalls or intrusion protection systems (IPS).

Effective Web application security requires complete awareness of application traffic and application threats. Not only must a solution protect all Web applications against known and unknown threats in real time, but also it is required to secure critical application data to prevent identity theft. When properly deployed, a Web application firewall can defeat the growing number of application-layer attacks, while preserving application access for legitimate users.

NOTICE

The information in this publication is subject to change without notice. THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTIES OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. CITRIX SYSTEMS, INC. ("CITRIX"), SHALL NOT BE LIABLE FOR TECHNICAL OR EDITORIAL ERRORS OR OMISSIONS CONTAINED HEREIN, NOR FOR DIRECT, INCIDENTAL, CONSEQUENTIAL OR ANY OTHER DAMAGES RESULTING FROM THE FURNISHING, PERFORMANCE, OR USE OF THIS PUBLICATION, EVEN IF CITRIX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES IN ADVANCE. THE USE CASES IN THIS PAPER ARE PROVIDED ONLY AS POTENTIAL EXAMPLES AND YOUR ACTUAL COSTS AND RESULTS MAY VARY.

CITRIX®

Best Access Experience. Anytime. Anywhere.

About Citrix: Citrix Systems, Inc. (Nasdaq:CTXS) is the global leader and most trusted name in on-demand access. More than 180,000 organizations around the world rely on Citrix to provide the best possible access experience to any application for any user. Citrix customers include 100% of the *Fortune* 100 companies and 98% of the *Fortune* Global 500, as well as hundreds of thousands of small businesses and individuals. Citrix has approximately 6,200 channel and alliance partners in more than 100 countries. Citrix annual revenues in 2005 were $909 million. Learn more at www.citrix.com.

32256/0706/2500

Citrix Worldwide

WORLDWIDE HEADQUARTERS

Citrix Systems, Inc.
851 West Cypress Creek Road
Fort Lauderdale, FL 33309 USA
Tel: +1 (800) 393 1888
Tel: +1 (954) 267 3000

EUROPEAN HEADQUARTERS

Citrix Systems International GmbH
Rheinweg 9
8200 Schaffhausen
Switzerland
Tel: +41 (52) 635 7700

ASIA PACIFIC HEADQUARTERS

Citrix Systems Hong Kong Ltd.
Suite 3201, 32nd Floor
One International Finance Centre
1 Harbour View Street
Central
Hong Kong
Tel: +852 2100 5000

CITRIX ONLINE DIVISION

5385 Hollister Avenue
Santa Barbara, CA 93111
Tel: +1 (805) 690 6400

www.citrix.com