ISSN: 2277 – 9043
International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)
Volume 2, Issue 2, February 2013

Frontline Techniques to Prevent Web Application Vulnerability

Shashank Khandelwal, Parthiv Shah, Mr. Kaushal Bhavsar, Dr. Savita Gandhi

Department Of Computer Science, Rollwala Computer Centre
Gujarat University, Ahmedabad, India

***Abstract*— Today web applications are becoming the prime target for cyber-attacks. Web attacks are growing in numbers, with most of organizations in a broad survey reporting that they had recently suffered Web attacks.Last few years have shown a significant increase in the number of web-based attacks. Structured Query Language (SQL) injection, Cross Site Scripting (XSS), Insecure Direct Object Reference, Command Injection, Session manipulation and Parameter or URL Tempering are some of the major attacks which are application layer attacks. This paper demonstrates how attackers are discovered, exploit application-level vulnerabilities in a large number of web applications and present the different techniques to prevent web application attacks. Using this research paper researcher can examine how web application firewall is better technique for preventing web application vulnerability. This approach allows us to secure our web application.**

***Index Terms*— SQL Injection, Cross Site Scripting (XSS), Insecure Direct Object Reference, Command Injection, Session manipulation and Parameter or URL Tempering, Web attacks, Firewall, IDS, IPS, Web Application Firewall.**

I. INTRODUCTION

Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client, sometimes called a thin client. The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Today web application is very important part of our lives. Firewall, Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) are a system designed to prevent unauthorized access to or from a private network. They can be installed in both hardware and software, or a combination of both. These all Techniques are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. All messages entering or leaving the intranet pass through them which examine each message and blocks those that do not meet the specified security criteria. Usually, the firewall, IDS and IPS will only allow port 80 for internet connection and blocks other ports. To a certain extent, it is known that web applications are insecure. As port 80 is the only port available for Internet connection, the hackers will intrude the application layer by using Structured Query Language (SQL) injection, Cross Site Scripting (XSS), Insecure Direct Object Reference, Command Injection, Session manipulation and Parameter or URL Tempering. These Defense Techniques are not preventing application layer attacks. Blocking network attacks is not the same as interactively securing the vital resources and information made available by Web applications. Web application firewalls are standalone appliances that sit in front of application servers, doing what network firewalls and IPSs were never intended to do examine all Web application data in depth to ensure correct application behavior and block suspicious activity thus preserving the safety of sensitive information and systems.

II. WEB ATTACKS

Web attacks and their techniques by which way attacker attacks over web application

- **Injection:** Injection flaws, such as SQL, Operating System, and Lightweight Directory Access Protocol (LDAP) injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data. For example The application uses untrusted data in the construction of the following vulnerable SQL call.

  String query = "SELECT * FROM accounts WHERECustID='" + request.getParameter("id") +"'";

The attacker modifies the 'id' parameter in their browser to send: ' or '1'='1. This changes the meaning of the query to return all the records from the accounts database, instead of only the intended customer's.

http://example.com/app/accountView?id=' or '1'='1

In the worst case, the attacker uses this weakness to invoke special stored procedures in the database that enable a complete takeover of the database and possibly even the server hosting the database.

- **Cross-Site Scripting (XSS):** XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
  For example: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping.

  (String) page += "<input name='creditcard' type='TEXT'value='" + request.getParameter("CC") + "'>";

  The attacker modifies the 'CC' parameter in their browser to:

  '><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.

  This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

- **Broken Authentication and Session Management:** Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities. For example Airline reservations application supports URL rewriting, putting session IDs in the URL:

  http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii

  An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.

- **Insecure Direct Object References:** A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data. for example The application uses unverified data in a SQL call that is accessing account information:

  String query = "SELECT * FROM accts WHERE account = ?";PreparedStatementpstmt=connection.prepareStatement(query , …);pstmt.setString(1, request.getParameter("acct"));ResultSetresults = pstmt.executeQuery( );

  The attacker simply modifies the 'acct' parameter in their browser to send whatever account number they want. If not verified, the attacker can access any user's account, instead of only the intended customer's account.

  http://example.com/app/accountInfo?acct=notmyacct

- **Cross-Site Request Forgery (CSRF):** A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests that the vulnerable application thinks are legitimate requests from the victim. For example: The application allows a user to submit a state changing request that does not include anything secret. Like so

  http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243

  So, the attacker constructs a request that will transfer money from the victim's account to their account, and then embeds this attack in an image request or iframestored on various sites under the attacker's control.

  <imgsrc="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct#"width="0" height="0" />

  If the victim visits any of these sites while already authenticated to example.com, any forged requests will include the user's session info, inadvertently authorizing the request.

- **Security Misconfiguration:** Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application. For example your application relies on a powerful framework like Struts or spring. XSS flaws are found in these framework components you rely

on. An update is released to fix these flaws but you don't update your libraries. Until you do, attackers can easily find and exploit these flaws in your app.

- **Insecure Cryptographic Storage:** Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes. For example an application encrypts credit cards in a database to prevent exposure to end users. However, the database is set to automatically decrypt queries against the credit card columns, allowing an SQL injection flaw to retrieve all the credit cards in clear text. The system should have been configured to allow only back end applications to decrypt them, not the front end web application.

- **Failure to Restrict URL Access:** Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway. For example: The attacker simply forces browses to target URLs. Consider the following URLs which are both supposed to require authentication. Admin rights are also required for access to the "admin_getappInfo" page.

  http://example.com/app/getappInfo
  http://example.com/app/admin_getappInfo

  If the attacker is not authenticated, and access to either page is granted, then unauthorized access was allowed. If an authenticated, non-admin, user is allowed to access the "admin_getappInfo"page, this is a flaw, and may lead the attacker to more improperly protected admin pages.Such flaws are frequently introduced when links and buttons are simply not displayed to unauthorized users, but the application fails to protect the pages they target.

- **Insufficient Transport Layer Protection:** Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly. For example: A site simply doesn't use SSL for all pages that require authentication. Attacker simply monitors network traffic (like an open wireless or their neighborhood cable modem network), and observes an authenticated victim's session cookie. Attacker then replays this cookie and takes over the user's session.

- **Unvalidated Redirects and Forwards:** Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages. For example: The application has a page called "redirect.jsp" which takes a single parameter named "url". The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware.

http://www.example.com/redirect.jsp?url=evil.com

III. Different Techniques to prevent web application attacks

1. Firewall- Firewall as a set of related programs, located at a network gateway server that protects the resources of a private network from the users of other networks. The term also implies the security policy that is used with the programs. An enterprise with an intranet that allows its workers access to the wider Internet installs a firewall to prevent outsiders from accessing its own private data resources and for controlling the outside resources that its own users have access to.

There are several types of firewall techniques:

- Packet filters: Looks at each packet entering or leaving the network and accepts or rejects it based on user-defined rules. Packet filtering is fairly effective and transparent to users, but it is difficult to configure. In addition, it is susceptible to IP spoofing.
- Application gateway**:** Applies security mechanisms to specific applications, such as FTP and Telnet servers. This is very effective, but can impose performance degradation.
- Circuit-level gateway**:** Applies security mechanisms when a TCP or UDP connection is established. Once the connection has been made, packets can flow between the hosts without further checking.
- Proxy server**:** Intercepts all messages entering and leaving the network. The proxy server effectively hides the true network addresses.

2. Intrusion Detection System (IDS)- Intrusion Detection Systems help information systems prepare for, and deal with attacks. They accomplish this by collecting information from a variety of systems and network sources, and then analyzing the information for possible security problems. Intrusion detection provides Auditing of system configurations and vulnerabilities, abnormal activity analysis and Operating system audit

There are two main components to the Intrusion detection system

- Network Intrusion Detection system (NIDS): performs an analysis for a passing traffic on the entire subnet. Works in a promiscuous mode, and matches the traffic

that is passed on the subnets to the library of know attacks. Once the attack is identified, or abnormal behavior is sensed, the alert can be sent to the administrator.

- Host Intrusion Detection System (HIDS): takes a snap shot of your existing system files and matches it to the previous snap shot. If the critical system files were modified or deleted, the alert is sent to the administrator to investigate.

3. Intrusion Prevention System (IPS)- An Intrusion Prevention System (IPS) generally sits in-line and watches network traffic as the packets flow through it. It acts similarly to an Intrusion Detection System (IDS) by trying to match data in the packets against a signature database or detect anomalies against what is pre-defined as "normal" traffic. In addition to its IDS functionality, an IPS can do more than log and alert. It can be programmed to react to what it detects. The ability to react to the detections is what makes IPSs more desirable than IDSs.

Intrusion prevention systems can be classified into four different types

- Network-based intrusion prevention system (NIPS): It monitors the entire network for suspicious traffic by analyzing protocol activity. It examines network traffic to identify threats that generate unusual traffic flows, such as distributed denial of service (DDoS) attacks, certain forms of malware, and policy violations.
- Wireless intrusion prevention systems (WIPS): It monitors a wireless network for suspicious traffic by analyzing wireless networking protocols.
- Host-based intrusion prevention system (HIPS): An installed software package which monitors a single host for suspicious activity by analyzing events occurring within that host.

Intrusion prevention systems utilize one of three detection methods

- Signature-Based Detection: This method of detection utilizes signatures, which are attack patterns that are preconfigured and predetermined. A signature-based intrusion prevention system monitors the network traffic for matches to these signatures. Once a match is found the intrusion prevention system takes the appropriate action. Signatures can be exploit-based or vulnerability-based. Exploit-based signatures analyze patterns appearing in exploits being protected against, while vulnerability-based signatures analyze vulnerabilities in a program, its execution, and conditions needed to exploit said vulnerability.
- Statistical anomaly-based detection: This method of detection baselines performance of average network traffic conditions. After a baseline is created, the system intermittently samples network traffic, using statistical analysis to compare the sample to the set baseline. If the activity is outside the baseline parameters, the intrusion prevention system takes the appropriate action.

- Statefull Protocol Analysis Detection: This method identifies deviations of protocol states by comparing observed events with "predetermined profiles of generally accepted definitions of benign activity.

4. Web Application Firewall (WAF) - WAFs are designed to protect web applications/servers from web-based attacks that IPSs cannot prevent. In the same regards as an IPS, WAFs can be network or host based. They sit in-line and monitor traffic to and from web applications/servers. Basically, the difference is in the level of ability to analyze the Layer 7 web application logic.

Where IPSs interrogate traffic against signatures and anomalies, WAFs interrogate the behavior and logic of what is requested and returned. WAFs protect against web application threats like SQL injection, cross-site scripting, session hijacking, parameter or URL tampering and buffer overflows. They do so in the same manner as an IPS does, by analyzing the contents of each incoming and outgoing packet.

WAFs not only detect attacks that are known to occur in web application environments, they also detect (and can prevent) new unknown types of attacks. By watching for unusual or unexpected patterns in the traffic they can alert and/or defend against unknown attacks. For example if a WAF detects that the application is returning much more data than it is expected to, the WAF can block it and alert someone.

Every website is different, as is every business. Accordingly, WAF policies are all unique and customized for each website except for basic, universal security requirements like valid HTTP protocol enforcement. WAF policies are created to programmatically describe what a website should or should not do. They'll take the form of either a white list, black list, or a combination of both. A white list policy might allow only explicitly defined Web pages (URLs) to be served from the website, with all other requests rejected. A black list policy might deny any requests containing "' SELECT * FROM", which indicates a possible SQL Injection attack. The challenge is that websites are diverse, complex, and constantly changing, requiring policies with hundreds if not thousands of clear and precise rules. One can see how difficult this would be to manage with a typical security team. This makes policy generation an extremely important function to streamline the process.

WAF policy generation may take place in three ways via vulnerability assessments, or manually. "Learning" is a process whereby website traffic is passively monitored for what types of requests are normally received, then dropping the rest. This approach often requires weeks or months to complete initially, plus the subsequent correction of any mistakes. While very helpful at eliminating much of the initial grunt work, this method is often made complex when rogue attack traffic is mingled with good traffic in a production environment, confusing the learning engine. The site is also left vulnerable while the process plays out.

Another option is leveraging the results from the vulnerability assessment process, assuming it is an ongoing, methodical process. Vulnerability assessment generated policies provide a WAF with intelligence regarding where and what type of vulnerabilities exist on a website to narrow the WAF's focus and provide more clearly defined policies.

Lastly, and least common, is a purely manual approach in which each URL, parameter value and overall aspect of a website is defined by hand. While theoretically the most accurate approach, it's also the most time consuming and difficult to manage and therefore impractical for most organizations in the Web 2.0 world.

Authentication and Authorization in WAF

Authentication and authorization mechanisms could be hard-coded into the application, but a better method of control is to setup Web Access Control (WAC) in front of the web service. WAC may or may not be as good as WAF at protecting the web application against compromise, but it has the potential to effectively and efficiently control who has access to what information, and at maintaining an audit trail for compliance reporting or security monitoring purposes. And WAC mechanism is including into WAF for effectively control.

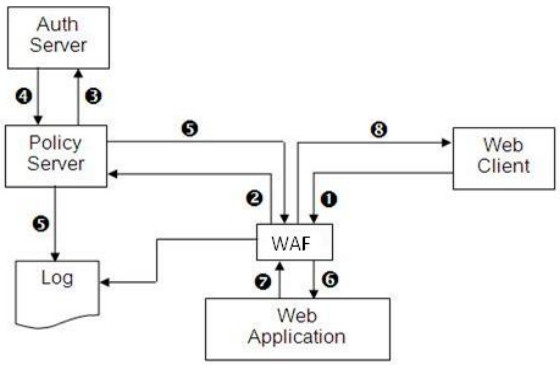

Figure 1 Authentication and Authorization

In the illustration above, whenever a client requests a page

Step 1.The request is first intercepted by the Web application firewall. The Web application firewall parses the request and determines the rules that apply to the requested access based on stored policy. If the requested page is protected and requires authentication, the Web application firewall presents the user with a logon page and forwards the user's credentials and access request to the policy server

Step 2.The policy server forwards user credentials to the authentication server

Step 3. The authentication server validates user credentials and informs the policy server

Step 4. The policy server creates an audit trail and sends back to the Web application firewall an 'Access Allowed' message

Step 5. Only then does the real web server receive the request from the Web application firewall.

Step 6. Serves the page back to the client through the Web application firewall path 7 and path 8.

In the logical data flow above, the real web server does not receive the client request until the user has been authenticated, the request authorized and an audit trail created.

IV. Comparison

Web Application Firewall can be differentiated from other security solutions that inspect traffic, most notably simple Firewall, Intrusion Detection System and Intrusion Prevention System. To be a WAF, a system should:

- **Have intimate understanding of HTTP** - while not of importance by itself, only by fully parsing and analyzing HTTP, breaking it to its elements including headers, parameters and payload, a WAF can effectively perform requirements and avoid evasion.
- **Provide a positive security model** - a positive security policy allows only things know to be valid to go through. This protection mechanism, sometimes called "white listing" provides an external input validation shield over the application. Too many web attacks cannot be reliably detected using signatures, making a signature only solution, not strong enough to protect web applications.
- **Application layer rules** - due to high maintenance cost, a positive security model by itself is not effective enough to protect web applications and should be

augmented by a signature based system. But since web applications are custom, traditional signatures targeting known vulnerabilities are not effective. WAF rules should be generic and detect any variant of an attack such as SQL injection.

- **Session based protection** - one of the biggest downsides of HTTP is the lack of a built in reliable session mechanism. A WAF must complement the application session management and protect it from session based and over time attacks.
- **Allow fine grained policy management** - most notably, exceptions should be applied to only minimal parts of the application, if a system does not allow minimal exceptions, false positives force opening wide security gaps.
- **Stops Data Leakage -** Data leakage can be caused by something as insignificant as a verbose error message presented to a public application user. If your application is harboring source code, credit card numbers, health information or other critical data, then a simple leak can turn into a catastrophe. In this instance, a WAF would like to scan everything that is returned as a response to your Web application users.

V. Conclusions and future works:

This research paper shows how easy it is to discover and exploit web application level vulnerabilities in a large number of web applications. Examples of such vulnerabilities are Structured Query Language injection, Cross Site Scripting, Insecure Direct Object Reference, Command Injection, Session manipulation and Parameter or URL Tempering. Although the majority of web vulnerabilities are easy to understand and avoid, many web developers are unfortunately not security-aware and there is general consensus that there exist a large number of vulnerable web applications and web pages on the web. Web Application Firewalls provide more security compared to the Firewalls, IDS and IPS. Using web Application Firewall we can protect our web servers more securely compared to the other techniques based on comparison in this paper. To the end, this paper helps you to suggest that, Web Application Firewall is the best technique to prevent Application Layer web attacks, but it cannot prevent distributed multi-platform web application servers with single web application firewall.

So in the future, this research will include creation of an architecture which can prevent the distributed web application with the single web application firewall.

REFRENCES

[1] Dafydd stuttard, and Marcus Pinto, "The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*", Wiley Publishing, Indana*, pp. 237-333, 2008.

[2] Naoto Katsumi, Isao Kaine and Katsutoshi Nakamura, "Web Application firewall", *Information-technology Promotion Agency japan*, pp. 9-20, 2011.

[3] Dr. Meshram B.B.,Patil Suchita, Kulkarni Pallavi and Rane Pradnya, "IDS vs IPS", *International Journal of Computer Networks and Wireless Communications (IJCNWC)*, vol 2 (1), pp. 86-90, 2012.

[4] Asaad Moosa, "Artificial Neural Network based Web Application Firewall for SQL Injection", *World Academy of Science, Engineering and Technology*, vol 40, pp. 12-21,2010.

[5] Lieven Desmet, Frank Piessens, Wouter Joosen, and Pierre Verbaeten, "Bridging the Gap between Web Application Firewalls and Web Applications", *DistriNet Research Group, Department of Computer Science Katholieke University Leuven Belgium*, 2009.

[6] Fuchsberger Andreas, "Intrusion Detection Systems and Intrusion Prevention Systems", Information *Security Technical Report*, vol 10, pp. 134-139, 2005.

[7] The Open Web Application Security Project. "OWASP Top 10 2010**",** Available at https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (Accessed on 15/02/2013).

Author's Profile

Shashank Khandelwal was completed B.E in Computer Engineering Ahmedabad Institute of Technology, Gujarat University, Ahmedabad in 2011. Currently doing M.Tech in Computer Engineering Department of Computer Science, Rollwala Computer Centre, Gujarat University, Ahmedabad, India. His M.Tech thesis has focus on frontline Defence of Distributed web Application. His areas of interest in security, cryptography, firewall, web application firewall and cloud computing.