Web Application Security and the OWASP Top 10
Web Application Security and the OWASP Top 10

This paper describes the most common vulnerabilities of web applications, as outlined in the OWASP Top 10. It also describes the security countermeasures that Sapient customers can implement to protect against web application hackers.

Overview: What is Web Application Security?

Web application security is a branch of Information Security that deals specifically with security of websites and web applications. It differs from the other branches of Information Security in that web application security is focused on vulnerabilities within the application code that is exposed during a user session on the web. The other areas of information security—that are not directly discussed in this document—are Network Security, Infrastructure Security, Database Security, and Operational Security.

A majority of the attacks against web servers are through network firewalls and through the http (80) or https (443) ports. Some of the most commonly used hacking techniques include denial of service, leakage, cross-site scripting, SQL injection and disclosure.

Due to the complexity of web applications and their supporting architectures (i.e. operating systems, databases, middleware, etc.), web attacks can be very sophisticated with serious, far-reaching implications. The complexity of web applications can also make web application security a more challenging endeavor than other branches of Information Security.

Hackers target web applications because it can be very lucrative for them to do so. For example, a successful attack on a bank’s web server could yield thousands of bank account numbers and user passwords information. The hacker could then use that information to gain a fortune by doing unauthorized money transfers and withdrawals. (An example of such an attack is described below.)

Web Attacks by the Numbers

Since 2005 there have been 297 publicly known breaches of web application security. In the 297 incidents, 297,573,821 records were known to have been compromised. 70% of the breaches were discovered by a third party (not by the organization that was breached).

One high-profile case occurred in 2010 at the University of Maine in Orono, Maine. Hackers compromised the personal information of 4,585 students who received services from the university’s counseling center. (The counseling center provides students with support and mental health services.) The information on the servers included names, Social Security numbers, and clinical information on every student who sought counseling services from the center between August 8, 2002 and June 21, 2010.

Another high-profile case in 2010 involved Small Dog Electronics in Waitsfield, Vermont. A hacker breached the web site and began stealing customer credit-card information after Small Dog began collecting and matching customer donations for Haiti relief efforts.

One of the most notorious breaches occurred in November of 2008, when hackers discovered vulnerability in the network of RBS WorldPay, a subsidiary of the Royal Bank of Scotland. The vulnerability was exploited to obtain a username and password which granted access to a database containing the account numbers and PINs of payroll debit cards.

44 fake debit cards were generated and distributed to 280 cities around the world to a pre-arranged network of
“cashers.” The hackers then accessed the accounts and increased the credit-card dollar limits and money-available on the cards. On Nov. 8, cashers in 280 cities around the world began accessing ATM machines using the fake cards. Within 12 hours, the hackers and cashers had stolen more than $9 million from RBS WorldPay.

Why Are Web Applications Vulnerable?

50% of all web traffic is now SSL-based: with SSL-based web traffic can come a false sense of security. Many people assume that the presence of a certificate on a web server means that the web server will create encryption—that there will be a tunnel from the user’s PC to the web server—and their transactions will be safe. In fact, it actually makes the web server less safe, because traditional firewalls do not detect Level 7 attacks. If the traffic is encrypted, there’s basically an encrypted tunnel going right through the firewall, passing any and all traffic. The existing IDS systems, for example, that may sniff the wire for bad or malicious traffic, can not do their job if the traffic is encrypted. For the most part, Layer 7 attacks can arrive directly at the web server undetected.

How Web Application Hacking is Conducted

When a web application firewall is installed—and it begins to register and sniff traffic—it soon becomes apparent that a great many spiders and bots are passing through the firewall. The spiders and bots in question are not the ones from Google and Yahoo, but those that have been customized to be somewhat malicious. They comb the site and look for certain things (e.g., e-mail addresses, domains, patch levels, and things of that nature.) When they find what they are looking for, they pull that data back to a central location (perhaps in China or Russia) and the hackers cull out any information that looks promising.

The hackers begin targeting that particular web application and domain to obtain more information. They do this by running targeted scripts to look at operating systems, application versions, open ports, and whatever is coming back from the server in terms of responses. When the hackers have enough information to execute an attack, they will do so, by SQL Injection, cross site scripting, or some other method. The end goal is obviously to exploit (i.e., to get credit card numbers, financial data, and credentials).
Figure 2 is a graphic example of a TCP packet going inbound and outbound. On an incoming packet, a traditional firewall looks at the host header information for the packet. On an outgoing packet, a traditional firewall looks at source/destination, IP address, etc., and some header information. Traditional firewalls are not designed to look at the payload section of the TCP packet, which is the section that is of interest to Layer 7. Thus, Layer 7 is where a lot of the web application attacks take place.

Figure 3. The Hacker’s Funnel

Spiders

Bots

admin@mail.acme.com
www.acme.com
Apache 216.27.178.28
Last modified – Dec 7, 2009

Operating system Headers
Application Open Ports
Versions Responses

SQL injection, XSS, parameter manipulation, etc.

Credit card numbers, financial data, credentials,
Web Application Security and the OWASP Top 10

Figure 4. A Successful Attack

Figure 4 illustrates a successful attack in flowchart form: the Threat Agents (hackers) launch a series of attack vectors (i.e., methods by which access to the system is gained). The attack vectors can be web pages, viruses, e-mail attachments, or instant messages, etc. One of the attack vectors is able to find a Security Weakness (e.g., weak authentication, lack of encryption, lack of data validation, etc). There is a failure in one of the Security Controls (Physical, Procedural, Technical/Legal, or Regulatory) that would otherwise stop the attack. The attack has a Technical Impact, such as a loss of any one of the following: Confidentiality, Integrity, Availability, or Accountability. Finally the attack has a Business Impact, such as Financial Damage, Non-Compliance, Privacy Violation, or Damage to the Organization’s Reputation.

By thinking of an attack in the way shown in Figure 4, you can put together a vulnerability model of how exposed a web application is, and what the potential impact of a web attack would be.

The Top Web Attack Methods

Figure 5. The Top Web Attack Methods
The OWASP Top 10 and PCI DSS requirement 6.6 have been linked together as a best practice implementation of web application security. Many organizations “cross reference” the two standards.

The following vulnerabilities, in descending order of severity, comprise the OWASP Top 10 for 2010:

- A1 – Injection Vulnerability
- A2 – Cross Site Scripting (XSS) Vulnerability
- A3-Broken Authentication and Session Management
- A4 – Insecure Direct Object References
- A5 – Cross Site Request Forgery (CSRF) Vulnerability
- A6 – Security Misconfiguration
- A7 – Failure to Restrict URL Access
- A8 – Unvalidated Redirects and Forwards
- A9 – Insecure Cryptographic Storage
- A10 - Insufficient Transport Layer Protection

Note that 32% of these attacks are either from cross-site scripting (XSS) or SQL Injection. Twenty percent of attacks are by unknown methods, which is alarming because we can not do an end-result or forensic search to determine exactly what happened. The rest of the attacks are by a smattering of miscellaneous methods, many of which are in the OWASP Top 10 (discussed below).

The Open Web Application Security Project (OWASP)

The Open Web Application Security Project (OWASP) is an open-source application security project. Its membership includes corporations, educational organizations, and individuals from around the world. The OWASP works to create freely-available articles, methodologies, documentation, tools, and technologies for web security. It is supported and managed by the OWASP Foundation, a 501(c)(3) charitable organization.

The OWASP Top 10 is a set of classes of vulnerabilities that are very high risk. Application developers can judge whether their applications meet best practices based on whether or not they has facilities to protect against these vulnerabilities. The OWASP Top 10 represents a broad consensus regarding the most critical vulnerabilities for web application security. A variety of security experts from around the world contribute their expertise to produce the OWASP Top 10.

An injection vulnerability can occur when a poorly-written program uses user-provided data in a database, or in a directory, query without first validating the input. You should always validate user input by testing the type, length, and range of the input. In addition, you should verify that the input is correctly formatted.

![Injection Vulnerability Diagram](image.png)
A1 - Injection Vulnerability
Injection vulnerabilities—such as to SQL, OS, or 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. Figure 6 illustrates an example illustrates an example if injection: the hacker is attempting to put things like SQL commands, or commands to bring things out of a database drop table, into a search form page. In this example, the hacker is trying to have the web server call the database and pull up the customer list.

A2 - Cross Site Scripting (XSS) Vulnerability
A Cross Site Scripting (XSS) vulnerability occurs whenever an application takes data that originates from a user or program and sends it to the browser without validating or encoding the data. XSS allows hackers to execute scripts in the victim’s browser, which can hijack user sessions, deface web sites, redirect the user to malicious sites, or conduct phishing attacks.

Figure 7 shows an XSS vulnerability that was detected on a Sapient client’s web site. In this case, Sapient cut and pasted the following string from an XSS cheat sheet into the client’s search field: ←TABLE BACKGROUND=“javascript” alert(Easy as 123)→. After clicking the Search button, the web site displayed a pop-up window that said “Easy as 123.”

An XSS vulnerability is not always dangerous in and of itself. However, an XSS vulnerability can show a hacker that due diligence may not have been conducted on the web site, and the web site may be susceptible to other more complex and more damaging attacks. The hacker can then search for other OWASP Top 10 vulnerabilities and find a point of attack.

A3 - Broken Authentication and Session Management
When application functions related to authentication and session management are not implemented correctly, hackers may be able to compromise passwords, keys, and session tokens—or exploit other implementation flaws—to assume other users’ identities.
A session hijacking occurs when a hacker takes control of a user session after successfully obtaining or generating an authentication session ID. This is done by using captured, brute-forced or reverse-engineered session IDs to seize control of a legitimate user’s Web application session while that session is still in progress. The causes of session hijacking include the following:

- An existing session ID is not invalidated, allowing the existing (previous) session ID to be used
- Forced use of a known session ID, as shown in Figure 8.

A4 - 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, hackers can manipulate these references to access unauthorized data. For example, in banking applications, it is common to use the account number as the primary key. Therefore, it is tempting to use the account number directly in the web interface itself. The best protection is to avoid exposing direct object references to users. You can do this by using an index, indirect reference map, or other indirect method that is easy to validate. If a direct object reference must be used, ensure that the user is authorized before using it. Keep the following in mind to avoid insecure direct object references:

- Whenever possible, avoid exposing to users private object references (such as primary keys or filenames).
- Validate any private object references extensively with an “accept known good” approach.
- Verify authorization to all referenced objects.
- Make sure that input does not contain attack patterns like ../.
A5 - Cross Site Request Forgery (CSRF) Vulnerability

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 hacker to force the victim’s browser to generate requests that the vulnerable application thinks are legitimate requests from the victim.

A practical example of CSRF is when a user is logged in to a banking application with the session still valid—the cookie has not expired or the session is still logged in from a time-limit perspective—and the hacker tries to trick the user into performing an action on the site. This would typically involve the hacker using malicious code to force the user to transfer money or withdraw money; the user would not even know that the transfer or withdrawal was occurring.

You can take the following measures to guard against CSRF:

- Verify transactions with tokens and challenges.
- Set short time limits for session IDs.
- Ensure that applications do not rely on credentials or tokens that are automatically submitted by browsers.
- Use cryptographic tokens to prove that the action formulator knows a session- and action-specific secret.

A6 - Security Misconfiguration

Security misconfiguration is new to the OWASP Top 10 in 2010. It consists of any non-compliance with the “best practices” for web application security.

The best practices for web application security include, but are not limited to, the following:

- Maintaining a secure system development life cycle—To the greatest extent possible, the test, QA, and production environments should be identical. Security must be integrated into the system development life cycle.
- Patch Management—Patches must be issued in a timely fashion to all mission-critical devices.

A7 - Failure to Restrict URL Access

Most web applications check URL access rights before rendering protected links and buttons. However, web applications need to perform similar access control checks each time a protected page is accessed, or hackers will be able to forge URLs to access the page.

If URL access rights are not protected each time a protected page is accessed, hackers will eventually gain access to restricted pages. The hackers will then use automated scripts to explore random directories or files until they find what they want.

The following URLs show some examples of sensitive data within the URLs themselves:

- http://example.com/app/admin_appinfo (admin access required for page)
- http://example/app/access=true (permissions located within the URL)
- http://example.com/app/user23/200023 (predictable information)

Do not assume that users will be unaware of special or hidden URLs or APIs. Always ensure that administrative and high privilege actions are protected.
A8 - Unvalidated Redirects and Forwards
Web applications frequently redirect and forward users to other pages and web sites, and use untrusted data to determine the destination pages. Without proper validation, hackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

The following are some examples of how hackers use this technique to redirect victims to phishing and malware sites:

- A hacker links to an unvalidated redirect and tries to trick users into clicking the link.
- Because the link is to a valid site, users are likely to click on it.
- Hackers can encode (and hide) the URL and even the savviest of end-users can be fooled into following the link.

A9 - Insecure Cryptographic Storage
Many web applications do not properly protect sensitive data—such as credit cards, Social Security Numbers, and authentication credentials—with appropriate encryption or hashing. Hackers may steal or modify improperly protected data to conduct identity theft, credit card fraud, or other crimes.

You can take the following measures to guard against insecure cryptographic storage:

- Know how personal, private, confidential, and proprietary data flow in and out of your environment. (For example, the following is web site→database server→file server→onsite backup→offsite backup→retrieval→destruction.)
- Implement strong encryption, authentication, and authorization when transporting and storing sensitive data.
- Separate the duties of IT personnel such that database analysts that may have access to sensitive data are different from security analysts that manage the cryptographic keys, eliminate shared accounts, and log activity.
- Use strong encryption, established algorithms, strong keys, hashing functions and checksums.

A10 - Insufficient Transport Layer Protection
This vulnerability typically involves weak encryption, non-authentication, and a failure to protect the confidentiality and integrity of sensitive network traffic. When these kinds of weaknesses and failures exist, the Transport Layer will often support weak algorithms and use expired or invalid certificates.

The causes of Insufficient Transport Layer Protection can include the following:

- Links/hops from HTTPS to HTTP that have not been well thought out to ensure that no scenarios exist where an exception condition, browser back button or URL modification can allow a session to improperly remain in or change to HTTP mode when secure data is being transmitted.
- The use of default, or easy-to-guess, account names and passwords.
- Improper storage of sensitive data in the database.
- Incorrect key lengths—For example, several years ago, a graduate student at the University of California used a network of about 250 workstations to crack a 40-bit algorithm in less than four hours; using the same configuration, it would have taken 1.4 million years to crack a 128-bit algorithm. Using the incorrect key length makes it easy to hack a web application, as shown in Figure 10.
Web Application Security and the OWASP Top 10

By putting flaws into the Top 10 format, an organization can calculate risk factors and provide a framework to reduce overall risk exposure.

The OWASP Top 10 is comprehensive, but there are other important application security risks that are constantly being discovered. A limitless number of vulnerabilities have yet to be identified. Applications are often compromised by applying a series of these techniques and vulnerabilities. For example, XSS by itself has limited technical impact, but XSS can be used as a vector to steal cookies and break authentication.

To address multiple vulnerabilities, keep the following in mind:

- Adhere to the principle of least privilege (i.e., every module, process, user, or program should only access necessary and legitimate resources).
- Validate input where ever and whenever possible.

Reducing Risk with Countermeasures and Mitigation

Web security involves the integration of security into the Software Development Life Cycle (SDLC) and the implementation of the following: secure coding, scanning and vulnerability tools, and web application firewalls. The foundation for web security consists of longstanding IT security measures (i.e.,

Secure Coding

No application is completely secure, but adhering to the following principals will minimize risk:

- Minimize the attack surface area (plug the holes and minimize the access points).
- Establish and implement secure default settings with password expiration and timeouts, etc.
- Implement the principle of “Least Privilege”; don’t give users access to things that they don’t need to do their jobs.
- Implement “Defense in Depth” with re-authentication, tokens, and hidden IDs, etc.
- Applications should fail securely.
- Don’t trust services or 3rd parties.
- Implement “Separation of Duties” (e.g., an admin is not an auditor – and vice versa).
- Avoid security by obscurity (“hiding” is only a temporary fix).
- Keep security simple (humans will always bypass
Scanning and Vulnerability Testing

A vulnerability scanner is a program designed to assess computers, computer systems, networks or applications for weaknesses. To do its job, a vulnerability scanner relies on a database that contains all of the information required to check a system for security holes in services, ports, protocols, and anomalies in packet construction.

The primary function of a vulnerability scanner is to conduct network reconnaissance, which is typically carried out by a remote attacker attempting to gain information or access to a network on which it is not authorized or allowed.

- Optimally, a scanner should be able to do the following:
  - Maintain an up-to-date database of vulnerabilities.
  - Detect genuine vulnerabilities without an excessive number of false positives.
  - Conduct multiple scans simultaneously.
  - Perform trend analyses and provide clear reports of the results.
  - Provide recommendations for countermeasures to eliminate discovered vulnerabilities.

Web Application Firewalls (WAFs)

A web application firewall (WAF) is used as a security device to protect the web server from attacks. It sits between a web client and a web server, analyzing OSI Layer-7 messages for violations in the programmed security policy.

WAFs are often called ‘Deep Packet Inspection Firewalls’ because they look at every request and response within the HTTP/HTTPS/SOAP/XML-RPC/Web Service layers (usually through ports 80 and 443).

Some WAFs look for certain ‘attack signatures’ to try to identify a specific attack that an intruder may be sending, while others look for abnormal behavior that doesn’t fit the website’s normal traffic patterns.

The use of a WAF is documented and recommended as a PCI DSS countermeasure.

Some of the common technologies and architectures in the WAF space include the following: Reverse Proxy, Transparent Proxy, Layer 2 Bridge, Network Monitor/Out of Band, Host/Server.

Commercial Tools
- Acunetix WVS by Acunetix
- AppScan by IBM *
- Burp Suite Professional by PortSwigger
- Hailstorm by Cenzic *
- MileScan Web Security Auditor by MileSCAN Technologies
- N-Stalker by N-Stalker
- Nessus by Tenable Network Security
- NetSparker by Mavitauna Security
- NeXpose by Rapid7 *
- NTOSpider by NTOobjects
- Retina Web Security Scanner by eEye Digital Security
- WebApp360 by nCircle
- WebInspect by HP *
- WebKing by Parasoft

Free / Open Source Tools
- Grabber by Romain Gaucher
- Grendel-Scan by David Byrne and Eric Duprey
- Paros by Chinotee
- PowerFuzzer by Marcin Kozlowski
- SecurityQA Toolbar by iSEC Partners
- Skipfish by Michal Zalewski
- W3AF by Andrei Riancho
- Wapiti by Nicolas Surribas

Software-as-a-Service Providers
- QualysGuard Web Application Scanning by Qualys
- Sentinel by WhiteHat
- Veracode Web Application Security by Veracode
- WebScanService by Elanize KG

* Also available as a service

Figure 12. Vulnerability Scanners

Figure 12 lists some of the more popular vulnerability scanners on the market. Vulnerability scanners are available as commercial tools, as freeware and open source tools, and as software-as-a-service tools.
Creating a Remediation Plan

To create a remediation plan, do the following:

1. **Identify assets and risks**, which consists of the following:
   - Obtain a full understanding of what you own (tangible and intangible).
   - Obtain a full understanding of the risks associated with those assets.

2. **Conduct a gap analysis and prioritize risks**:
   - Determine the biggest risks to your most expensive assets.
   - Prioritize the risks.
   - Draft and communicate those risks in order (i.e. high/medium/low) to the organization.

3. **Planning and Execution**:
   - Begin planning your remediation plan (i.e. budget, technology, team, timeframe).
   - Publish the plan and execute the plan.

4. **Track, monitor, and improve the plan**.
   This step will be time-consuming. (The amount of time that it takes depends on the GAP analysis.) This is an ongoing process; the program is never finished; it restarts and can always be improved. It is important to manage costs, time, and resources for solid execution.
Web Application Security and the OWASP Top 10

Subject Matter Expert:
Jon Panella

Jon Panella has over 25 years of technical, consulting and leadership experience in enterprise architecture planning, commerce strategy, product evaluation/selection, software development and technology implementation/support. He is currently responsible for technology direction within Sapient’s Global Commerce Practice, which includes oversight, planning and reviews for numerous commerce strategy and implementation engagements.

Jon has lead many strategy and implementation projects with Fortune 500 clients, including Target, Barnes & Noble College Booksellers, Sprint, JCPenney, and PetSmart. Some of his recent engagements include:

- eCommerce product evaluation, selection and roadmap for JCPenney
- Architectural assessment and eCommerce planning for Target
- Enterprise Architecture assessment and product recommendation for Sprint
- Design and implementation of an eCommerce solution for Barnes & Noble College Booksellers
- An architectural and business assessment, product selection, and roadmap for PetSmart

Jon has been with Sapient for eight years. Prior to joining Sapient, Jon was at American Airlines/Sabre for over 15 years, where he was responsible for a number of technology areas, including airline reservations, pricing and yield management, and ticketing. He was also Vice President of Technology for GetThere (now Travelocity Business). Jon has been a member of the IBM Websphere Commerce Product Advisory board for over two years and manages the technical components of Sapient’s partnerships with key Commerce vendors such as IBM, ATG, Endeca, Hybris, Bazaarvoice, and Omniture.