PREPARING FOR THE NEW OWASP TOP 10 AND BEYOND

THINK APP SECURITY FIRST

OWASP TOP 10 AND BEYOND

CROSS-SITE SCRIPTING

SQL INJECTION
INTRODUCTION

There’s a reason why web application firewalls have been getting so much attention lately. It’s the same reason we keep hearing about major security and data breaches left, right, and center.

Web application security is difficult—very difficult.

Not to mention time consuming and costly. Building and maintaining comprehensive web security controls can consume a large percentage of the limited budget you have for developing the actual application features users need to get useful work done.

In fact, web application security is so challenging that WhiteHat Security stated in its 2017 Application Security Statistics Report that the average web application has three vulnerabilities.² Are we are not investing enough in penetration testing and remediation? Do we not understand the risks? Are we not deploying the right tools to mitigate these vulnerabilities?

These are persistent, long-standing problems that remain omnipresent due to the difficulty of building and rebuilding remediations into every new application that is shipped. Understanding and defending against web application vulnerabilities typically requires focused security expertise, a skillset that few developers can realistically cultivate while getting actual development done at the same time.

Fortunately, there are options. Having the right tools and third-party controls in place can go a long way toward mitigating risk—and speeding development of your applications at the same time.


30% of the total breaches reported involved attacks on web applications.¹
THE OPEN WEB APPLICATION SECURITY PROJECT: CAN EDUCATION REDUCE VULNERABILITIES?

The pervasive nature of web application security shortcomings has not gone unnoticed. In 2001, a number of security professionals banded together to create the Open Web Application Security Project (OWASP) to educate developers and hopefully reduce these security shortcomings. OWASP is a nonprofit international group that produces publicly available methodologies, documentation, tools, and training addressing many aspects of web application security.

THE OWASP TOP 10: A TAXONOMY OF RISK

The most well-known of the OWASP projects is the “OWASP Top 10,” which is an ever-evolving list of the biggest security problems common to web applications. The goal of the OWASP Top 10 is to provide a basic taxonomy of risk with respect to web application vulnerabilities.

Future versions of the OWASP Top 10 are slated to be more closely aligned with widely accepted risk frameworks, such as ISO 31000:2015³, the intention being to boost the credibility and applicability of the project, which in turn may increase uptake and adherence to its philosophy.

PROTECTING YOUR APPLICATIONS: AN OVERVIEW OF THREATS

If you are responsible for the development, security, or operation of a web application, becoming familiar with the OWASP Top 10 can help you better protect that app. In addition, security testing against the OWASP Top 10 is a core requirement of numerous industry and regulatory standards such as the PCI DSS. The OWASP site also lists other relevant international security standards that reference the OWASP Top 10.⁴ By delving into the most common web app security problems and learning about effective mitigations, you can boost your organizational security posture, protect your critical applications, and help ensure the confidentiality, integrity, and availability of your data.

³ https://www.iso.org/iso-31000-risk-management.html
⁴ https://www.owasp.org/index.php/Industry:Citations
Injection is a common class of vulnerability where insufficently sanitized input provided by external sources contains hidden application commands from an attacker. Because the web application is not properly filtering the input, it allows injected commands to be passed through to either the local system or a dependent one. A common example is the SQL injection attack. Many applications rely upon user input to build SQL statements to fetch information or to log them in. For example:

```
select * from USERTABLE where USERID = '{userid-from-web-form}' and PASSWORD = '{passwd-from-web-form}'
```

Under normal circumstances this could match an entry in the data table USERTABLE, and if so, then the statement evaluates to “True” and the login succeeds.

But what if the user puts in “password’ OR 1=1” as the password on the web form:

```
select * from USERTABLE where USERID = '{userid-from-web-form}' and PASSWORD = 'password' OR 1=1
```

Without proper sanitization and escaping, the SQL server will evaluate the 1=1 conditional as TRUE and log the user in with the username provided regardless of what password was supplied. This is a very simple, and targeted, example. SQL injection attacks can get far more sophisticated and malicious, and have been used successfully to delete entire databases, modify records, and exfiltrate sensitive data.

Mitigating Injection Risks

With respect to data input and security, you cannot inherently trust any data from the user. All input must be examined, escaped, sanitized, and filtered. Injection attacks can occur in normal user input forms as well as in hidden web fields. Leveraging parameterized SQL⁷ can go a long way toward mitigating this risk by compartmentalizing input data and distinguishing it from code, regardless of user input. It is also advisable to monitor outbound responses returned to the user in an effort to detect information leakage resulting from a successful injection attack.

25 MOST COMMON PASSWORDS IN 2017

1. 123456
2. Password
3. 12345678
4. qwerty
5. 12345
6. 123456789
7. jetmein
8. 1234567
9. football

10. iloveyou
11. admin
12. welcome
13. monkey
14. login
15. abc123
16. starwars
17. 123123

18. dragon
19. password
20. master
21. hello
22. freedom
23. whatever
24. qazwsx
25. trustno1

BROKEN AUTHENTICATION

Accurately knowing who a user is (authentication) and what they are allowed to do (authorization) are foundational concepts of security that complement each other. Since we’re talking about web application authentication, we should begin with passwords. Despite their rudimentary nature and the inherent risks and inconvenience that come with them, passwords are unfortunately still far and away the most common way of authenticating a user.

However, stolen credentials are the predominant method of web application compromise. The insecure nature of user passwords makes attacks like credential stuffing both easy and remarkably successful. A credential stuffing attack occurs when an attacker has gotten their hands on a large database of stolen user credentials. They then use automated tools to test those passwords against a variety of other sites and services to see what works. By some estimates, as much as 90% of all login attempts on web-based applications at Fortune 100 firms are credential stuffing attempts rather than legitimate logins. Attacks like credential stuffing are made possible by our persistent refusal to move past passwords, or more broadly adopt federated authentication.

> MITIGATING BROKEN AUTHENTICATION

One way to avoid the problems with passwords is not to use them. Client certificates, token-based two-factor and federated authentication are great ways to reduce reliance on passwords. Robust authentication can be difficult to build, secure, and maintain, so you can leverage federation to speed development and delivery of your application—and make users happy at the same time. Remember: no one wants yet another account and password to manage, no matter how cool your app may be.

---

Sensitive data exposure is an information leakage problem. The sensitivity of what is leaked can vary, but divulging any information about how a web application is designed to an attacker is a bad idea. This kind of information is low-hanging fruit for automated scanners and ripe for exploitation.

Some examples of the kinds of information commonly leaked by web applications that attackers find useful include the following:

- Error messages detailing how unexpected input is handled
- Physical locations of files on server
- Specific versions of components and libraries
- Stack traces from failed functions (can be decompiled and examined)
- “Forgot password” function error messages that reveal user ID validity, which can be used for discovery and brute-force attacks on user accounts and passwords

**Mitigating Sensitive Data Exposure**

There are several steps you can take to minimize your risk for data leakage. It is very common for web servers to report vendor and version information, among other things.10 Make sure that usernames cannot be validated from server response codes: an incorrect username error and incorrect password should generate the same error message. Ensure browser security directives are used to help protect sensitive data in transit. Avoid old and known weak cryptographic algorithms and methods. Transport Layer Security (TLS) is easy to use and is becoming the universal norm on the Internet.11 All sensitive data stored within a web application should be rendered unreadable—using techniques such as encryption or tokenization12—in case an attacker gains access through the application.

---

Apps that accept XML input can inadvertently allow external commands that cause XML processors to divulge data.

XML External Entities

Insufficiently hardened or misconfigured XML parsers or processors evaluate external entity references with XML documents. Apps, such as SOAP services, that accept XML input can inadvertently allow for inclusion of unanticipated external references and commands that cause XML processors to divulge data on internal file shares, execute code, initiate internal port scanning, and perform denial of service (DoS) attacks.

Mitigating XML External Entities

Carefully consider which APIs to expose in the first place, and ensure that vulnerable XML processors are upgraded, patched, or otherwise hardened. Once that is done, ensure that HTTP message sizes, versions, and methods are all enforced. Using a web application firewall (WAF) gives you the option to whitelist appropriate requests or block malicious ones from being sent to your XML processor in the first place. Additionally, a WAF can help validate JSON, XML, and SOAP requests, and apply signatures against known attacks that can cause DoS attacks and unauthorized information disclosure. Finally, a network firewall and traffic management can limit outbound requests to other endpoints, internal or external.
Security controls can be considered misconfigured for a variety of reasons, but a common cause stems from errors or omissions in user-facing documentation that result in gaps in controls, or missed steps. It could also be oversight and/or mistakes made by systems administrators who are, after all, only human. Dependent software and infrastructure can also be overlooked. Most web applications depend on other software (such as Apache, IIS, or Nginx) and may leverage other applications, libraries, and databases (such as PHP, ASP, or SQL).

A5 BROKEN ACCESS CONTROL

A broken access control vulnerability refers to a flaw in the design of the web application where unauthorized access to a sensitive object (like a directory or record) is improperly or insufficiently enforced. For example, it could be that any anonymous user could view certain files on a website simply by knowing what URL to request; alternatively, the application could execute a function that assumes some level of authentication or authorization has occurred without first validating that is the case.

> MITIGATING BROKEN ACCESS CONTROL

All web application objects and pages should have an enforcement mechanism that denies access by default. From there, the system should enforce explicit rights and only grant them to users associated with specific user roles.

A6 SECURITY MISCONFIGURATION

Security controls can be considered misconfigured for a variety of reasons, but a common cause stems from errors or omissions in user-facing documentation that result in gaps in controls, or missed steps. It could also be oversight and/or mistakes made by systems administrators who are, after all, only human. Dependent software and infrastructure can also be overlooked. Most web applications depend on other software (such as Apache, IIS, or Nginx) and may leverage other applications, libraries, and databases (such as PHP, ASP, or SQL).

> MITIGATING SECURITY MISCONFIGURATION

To properly secure a web application, not only must the software itself be properly locked down, but so must all the other integral components. It is also important to conduct regular and thorough audits to ensure that controls have been implemented properly and remain firmly in place.

93% OF ATTACKS ON WEB APPLICATIONS WERE FINANCIALLY MOTIVATED, PERPETRATED BY ORGANIZED CRIMINAL GROUPS.¹³

Cross-site scripting (XSS) refers to an input validation vulnerability that lets attackers run their own malicious scripts in a victim’s browser within the trusted context of a site they’re visiting. XSS can be used to steal session tokens, initiate hidden transactions, or display falsified or misleading content. More sophisticated XSS scripts can even load key loggers to monitor victims’ passwords as they type them in, relaying that information to command-and-control servers operated by the attackers.

XSS can occur anywhere an external user can contribute content to a website, which makes it one of the most common types of vulnerabilities. XSS is also hard to identify and eliminate because it uses the same HTML commands required by a website to render its pages. Furthermore, XSS attacks can be encoded in a variety of ways. For example, a basic attack script to pop up the message “XSS” on a page could look like this:

```html
<script>alert('XSS')</script>
```

But encoded, it could also look like this:

```html
%253Cscript%253Ealert('XSS')%253C%252Fscript%253E
```
or this:

```html
<IMG SRC="jav&#x0D;ascript:alert('XSS');">
```
or even this:

```html
<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x3A;&
 #x61&#x6C&#x65&#x72&#x74&#x28;&#x27&#x58&#x53&#x27&#x29>
```

With all these encoding possibilities supported by native browsers, it’s easy to see how XSS vulnerabilities can slip through. Complicating things further is the fact that there are a number of different XSS delivery methods that can be employed, making this a very flexible attack vector.

> MITIGATING CROSS-SITE SCRIPTING

Stopping XSS goes back to the idea that any external data supplied by the user is untrustworthy. Since any data input can have a malicious payload embedded in it, it is imperative to filter everything coming in, although this can be very difficult to accomplish comprehensively on your own. Fortunately, a good WAF can do it for you, freeing up time and resources with infrastructure that is reusable.
INSECURE DESERIALIZATION

This is a new addition to the top 10, and deals with object serialization, which is the process of turning an object into a data format that can be restored later. Think about how files are saved to a disk or how data is transferred over a network. The data is saved in a given structure using formats such as JSON or XML. Deserialization is the opposite—reading this structured data and building an object from it.

Many programming languages offer native serialization or allow customization of the serialization process, which can in turn be used maliciously. Insecure deserialization has been found to allow remote code execution, denial-of-service, replay, injection, and privilege escalation attacks.

> MITIGATING INSECURE DESERIALIZATION

To protect against a deserialization attack, you should not accept serialized objects from untrusted sources. When that isn’t possible, you can enforce HTTP message sizes, versions, methods, and required headers. Also, an advanced WAF can validate JSON, XML, and SOAP requests against known attack signatures as well as validate data length, value, and structure. Your WAF should also allow you to customize handling of data designated as sensitive.
ATTACKERS RELY ON COMPLACENCY AND BLIND SPOTS TO REMAIN UNDETECTED, AND GAIN FOOTHOLDS INTO APPS AND NETWORKS.

A9 USING COMPONENTS WITH KNOWN VULNERABILITIES

This is another one of those risk areas that may seem obvious, but it is worth addressing since many software components are chosen solely for their utility in fulfilling some basic operational requirement. Such components may not have known vulnerabilities when implemented, and it is common for there to be resistance to upgrades out of fear of breaking functionality or losing a valuable legacy feature.

These are valid concerns, but a successful exploit against a known security vulnerability can result in a significant loss of service or breach of customer confidence, so this risk must be weighed against the perceived risks of upgrading.

> MITIGATING USING COMPONENTS WITH KNOWN VULNERABILITIES

The key here is to keep components updated wherever you can. Where you can’t, compensating controls such as a strong WAF will enable you to block the exploitation of known vulnerabilities while allowing you to keep your software intact and operational. A WAF can also buy you time while patches are developed and rolled out.

A10 INSUFFICIENT LOGGING AND MONITORING

While most applications are built to log some level of access and authorization information, many are not, or do not do so by default. The risk here is that log data and access information that is not collected or not analyzed will not help detect a breach, or facilitate expedient service recovery in the event of an incident.

Attackers rely on complacency and blind spots to remain undetected, and gain footholds into apps and networks. Diligent monitoring of this data can help detect attacks early, allowing you to build or further fortify your defensive posture, and ultimately, minimize damage or impact to your organization.

> MITIGATING INSUFFICIENT LOGGING AND MONITORING

Start by identifying which apps and endpoints are likely to generate the most useful logging information and can be relied on to provide early warning of anomalous behavior. You will need to define what must be collected and how it should be analyzed and monitored. Good WAF solutions will allow you to standardize logging for all of your web applications—and to log that information off-box for further analysis and reporting.
THE OWASP TOP 10:
ONE PIECE OF THE APP SECURITY PUZZLE

When evaluating the myriad of potential attack vectors, one thing becomes clear: your web applications face a number of complex threats that are difficult and costly to defend against. Most development teams simply do not have the resources to sufficiently protect against the variety of attacks that are relevant to each vector—and the level of expertise required is such that it will be difficult to come by even if your project has the time and budget for it.

The good news is that advanced WAF technology is more accessible and affordable than ever before. Modern, full-featured WAFs can help organizations of all sizes defend their critical apps—whether they’re deployed in data centers or hybrid cloud environments. Unique and flexible deployment options can simplify implementation and make it easy to customize protection for your app.

While you’re addressing your security needs to protect against the OWASP Top 10, it’s also essential to consider all the other threats to your applications: DDoS attacks, bot-enabled attacks, intellectual property theft, and fraud, just to name a few.

JUST BECAUSE A THREAT DIDN’T MAKE THE OWASP TOP 10 DOESN’T MEAN IT IS NO LONGER RELEVANT.

Take, for instance, cross-site request forgery (CSRF), which involves tricking a victim using a browser into clicking on a benign-looking link that actually performs a malicious action, such as initiating a transfer within your banking app. CSRF doesn’t appear on the list this year, but you can be sure that criminals are still out looking for opportunities to exploit it—as well as innumerable clever gambits—to compromise your apps. Defending against the OWASP Top 10 is a great—and necessary—step, but it’s only one piece of a comprehensive, defense-in-depth strategy that will help you protect your apps, your data, and your business.

For more information on the threats that affect your applications and your organization—as well as what you can do to defend against them—visit f5.com/security.

---

Always-on, always-connected apps can help power and transform your business—but they can also act as gateways to data beyond the protections of your firewalls. With most attacks happening at the app level, protecting the capabilities that drive your business means protecting the apps that make them happen.

Learn more about application security at the AppProtectLibrary.