What is OWASP?

- The Open Web Application Security Project, or OWASP, is an international non-profit organization dedicated to web application security. One of OWASP's core principles is that all of their materials be freely available and easily accessible on their website, making it possible for anyone to improve their own web application security. The materials they offer include documentation, tools, videos, and forums. Perhaps their best-known project is the OWASP Top 10.

What is the OWASP Top 10?

- The OWASP Top 10 is a regularly-updated report outlining security concerns for web application security, focusing on the 10 most critical risks. The report is put together by a team of security experts from all over the world. OWASP refers to the Top 10 as an ‘awareness document’ and they recommend that all companies incorporate the report into their processes in order to minimize and/or mitigate security risks.

Below are the security risks reported in the OWASP Top 10

1. Injection

- Injection attacks happen when untrusted data is sent to a code interpreter through a form input or some other data submission to a web application. For example, an attacker could enter SQL database code into a form that expects a plaintext username. If that form input is not properly secured, this would result in that SQL code being executed. This is known as an SQL injection attack.
- Injection attacks can be prevented by validating and/or sanitizing user-submitted data. (Validation means rejecting suspicious-looking data, while sanitization refers to cleaning up the suspicious-looking parts of the data.) In addition, a database admin can set controls to minimize the amount of information an injection attack can expose.

2. Broken Authentication

- Vulnerabilities in authentication (login) systems can give attackers access to user accounts and even the ability to compromise an entire system using an admin account. For example, an attacker can take a list containing thousands of known username/password combinations obtained during a data breach and use a script to try all those combinations on a login system to see if there are any that work.
- Some strategies to mitigate authentication vulnerabilities are requiring two-factor authentication (2FA) as well as limiting or delaying repeated login attempts using rate limiting.

3. Sensitive Data Exposure

- If web applications don’t protect sensitive data such as financial information and passwords, attackers can gain access to that data and sellor utilize it for nefarious purposes. One popular method for stealing sensitive information is using an on-path attack.
- Data exposure risk can be minimized by encrypting all sensitive data as well as disabling the caching* of any sensitive information. Additionally, web application developers should take care to ensure that they are not unnecessarily storing any sensitive data.
- *Caching is the practice of temporarily storing data for re-use. For example, web browsers will often cache webpages so that if a user revisits thosepages within a fixed time span, the browser does not have to fetch the pages from the web.

4. XML External Entities (XEE)

- This is an attack against a web application that parses XML* input. This input can reference an external entity, attempting to exploit a vulnerability in the parser. An ‘external entity’ in this context refers to a storage unit, such as a hard drive. An XML parser can be duped into sending data to an unauthorized external entity, which can pass sensitive data directly to an attacker.
- The best ways to prevent XEE attacks are to have web applications accept a less complex type of data, such as JSON**, or at the very least to patch XML parsers and disable the use of external entities in an XML application.
- *XML or Extensible Markup Language is a markup language intended to be both human-readable and machine-readable. Due to its complexity and security vulnerabilities, it is now being phased out of use in many web applications.
- **JavaScript Object Notation (JSON) is a type of simple, human-readable notation often used to transmit data over the internet. Although it was originally created for JavaScript, JSON is language-agnostic and can be interpreted by many different programming languages.

5. Broken Access Control

- Access control refers a system that controls access to information or functionality. Broken access controls allow attackers to bypass authorization and perform tasks as though they were privileged users such as administrators. For example a web application could allow a user to change which account they are logged in as simply by changing part of a url, without any other verification.
- Access controls can be secured by ensuring that a web application uses authorization tokens* and sets tight controls on them.
- *Many services issue authorization tokens when users log in. Every privileged request that a user makes will require that the authorization token be present. This is a secure way to ensure that the user is who they say they are, without having to constantly enter their login credentials.

6. Security Misconfiguration

- Security misconfiguration is the most common vulnerability on the list, and is often the result of using default configurations or displaying excessively verbose errors. For instance, an application could show a user overly-descriptive errors which may reveal vulnerabilities in the application. This can be mitigated by removing any unused features in the code and ensuring that error messages are more general.

7. Cross-Site Scripting

- Cross-site scripting vulnerabilities occur when web applications allow users to add custom code into a url path or onto a website that will be seen by other users. This vulnerability can be exploited to run malicious JavaScript code on a victim's browser. For example, an attacker could send an email to a victim that appears to be from a trusted bank, with a link to that bank's website. This link could have some malicious JavaScript code tagged onto the end of the url. If the bank's site is not properly protected against cross-site scripting, then that malicious code will be run in the victim's web browser when they click on the link.
- Mitigation strategies for cross-site scripting include escaping untrusted HTTP requests as well as validating and/or sanitizing user-generated content. Using modern web development frameworks like ReactJS and Ruby on Rails also provides some built-in cross-site scripting protection.

8. Insecure Deserialization

- This threat targets the many web applications which frequently serialize and deserialize data. Serialization means taking objects from the application code and converting them into a format that can be used for another purpose, such as storing the data to disk or streaming it. Deserialization is just the opposite: converting serialized data back into objects the application can use. Serialization is sort of like packing furniture away into boxes before a move, and deserialization is like unpacking the boxes and assembling the furniture after the move. An insecure deserialization attack is like having the movers tamper with the contents of the boxes before they are unpacked.
- An insecure deserialization exploit is the result of deserializing data from untrusted sources, and can result in serious consequences like DDoS attacks and remote code execution attacks. While steps can be taken to try and catch attackers, such as monitoring deserialization and implementing type checks, the only sure way to protect against insecure deserialization attacks is to prohibit the deserialization of data from untrusted sources.

9. Using Components With Known Vulnerabilities

- Many modern web developers use components such as libraries and frameworks in their web applications. These components are pieces of software that help developers avoid redundant work and provide needed functionality; common example include front-end frameworks like React and smaller libraries that used to add share icons or a/b testing. Some attackers look for vulnerabilities in these components which they can then use to orchestrate attacks. Some of the more popular components are used on hundreds of thousands of websites; an attacker finding a security hole in one of these components could leave hundreds of thousands of sites vulnerable to exploit.
- Component developers often offer security patches and updates to plug up known vulnerabilities, but web application developers don’t always have the patched or most-recent versions of components running on their applications. To minimize the risk of running components with known vulnerabilities, developers should remove unused components from their projects, as well as ensuring that they are receiving components from a trusted source and ensuring they are up to date.

10. Insufficient Logging And Monitoring

- Many web applications are not taking enough steps to detect data breaches. The average discovery time for a breach is around 200 days after it has happened. This gives attackers a lot of time to cause damage before there is any response. OWASP recommends that web developers should implement logging and monitoring as well as incident response plans to ensure that they are made aware of attacks on their applications.
- Reference: https://www.cloudflare.com/learning/security/threats/owasp-top-10/

What Is Frontend Application Monitoring?

- Frontend monitoring is what we use to describe the process and tools used by developers to track and maintain the health of the presentation layer of your applications. It's basically everything that a user would interact with, from the content and menu to APIs and other client-facing components.
- Application performance monitoring, or APM, focuses on what users experience rather than the communication between server and client. While both are worth keeping an eye on, understanding how the users interact with your application is crucial in delivering a good experience, which leads me to my next point.

Is Frontend Monitoring Important?

- Absolutely! We use many different monitoring tools and techniques to monitor our backend, databases, infrastructure, and frontend shouldn’t be any different. As the complexity of our websites and applications increases, the monitoring needs are going up too. Twenty years ago, you might have gotten away with just pinging your server a couple of times a day. Today that’s nowhere near enough. Any downtime or degradation of the performance of your website can have severe consequences.

What Performance Issues Can Frontend Monitoring Tools Fix?

- When it comes to measuring application performance, I suggest we split our focus into three main components that I believe are of equal importance.
- **Speed**
- This should be obvious. Slow loading resources will cause bottlenecks and a bad user experience, so it's crucial to resolve them as soon as possible. Before you ask how fast should a good site be and how slow is considered too slow, there's an application performance index called Apdex that helps you figure out the target response time.
- Your page speed will be significantly affected by the size of your resources and the number of requests. As a general rule of thumb, you should try to optimize all your resources and get the response times to be as fast as possible. These are just two of the many things you can do to increase website speed. You can find more techniques in our article about front-end optimization tips.
- If you don't know yet how fast is your website, read our blog post on how to properly check website speed.

Errors and Functionality

- I’m not talking about the obvious errors that you’ll get like 404 or 503. I’m talking about the errors that are less obvious like a 3rd party API that refuses connection or sending an HTTP request when you should be sending an HTTPS request. Issues like these can cause severe service disruptions that will break your users’ experience.
- Having a tool that alerts you whenever your website or app experiences an issue regardless of whether it’s a JavaScript error, network failure, or a framework-specific issue will make your life a whole lot better. Get a tool that does log monitoring and management such as Sematext Logs, and you’ll find it even easier to identify and fix the issue you are having.
- While some of the issues you’ll run into as a developer can be solved using solely the information you were provided in the console log, you will have some that aren’t as easy to figure out. In some cases you’ll have a single location for your logs but more often than not, your logs will be in separate locations, with separate timestamps, and probably have different redundancies that will make debugging quite complicated.
- Having a tool that can store, archive, and control the quality of your logs is essential to understanding the whole picture. You can use separate solutions, but having logging and monitoring in a single platform makes troubleshooting critical issues much faster and much simpler.

Availability

- Not to state the obvious here, but your website needs to be online and working for your users to use it. Duh! But availability can be difficult to ensure, especially once you've grown enough and you have a bigger pool of users that are located around the world. Having insight into how your users experience your website will help tremendously, which is the perfect scenario for using real user monitoring tools such as Sematext Experience.
- Such a solution will follow the user's progress throughout the application giving you information about how they interact with it and what kind of responses they're getting. Based on these UX metrics, you'll be able to decide if you need a faster machine, better scaling, or have to optimize your business logic.

Best Tools to Monitor Application & Website Performance

- **1. Sematext**
- **2. AppSignal**
- **3. Sentry**
- **4. Site24x7**
- **5. Pingdom**
- **6. Raygun**
- **7. SpeedCurve**
- **8. LogRocket**

How to Choose the Right Solution for You?

- This is not an easy question to answer as it does not have a universal answer. There are, however, a few things that are worth taking into account when choosing the right tool for your use case:
- Ease of use is essential as you really want to avoid spending time reading the documentation when you could be actively improving your app.
- Third-party monitoring solutions and API monitoring are crucial to understanding what is slowing your website down.
- Multiple testing locations is always a great feature. This will give you a better understanding of how your website performs for different locations around the world. If you want to learn more about how location affects user experience and website performance, read our blog post about latency.
- .

Continue..

- Logs support is awesome if you have it. It will supercharge your performance monitoring by providing key intel that isn't available otherwise. If you're new to logs, we recommend going through our introductory log management guide.
- Third-party integrations with your favorite tools will make the tool easier to adopt
- Infrastructure monitoring tools will give you the complete picture when it comes to performance
- Reference: https://sematext.com/blog/frontend-performance-monitoring/

What is OWASP ZAP?

- OWASP ZAP is a penetration testing tool that helps developers and security professionals detect and find vulnerabilities in web applications. OWASP ZAP performs multiple security functions including:
- Passively scanning web requests
- Using dictionary lists to search for files and folders on web servers
- Using crawlers to identify a site's structure and retrieve all links and URLs
- Intercepting, displaying, modifying, and forwarding web requests between browsers and web applications
- OWASP ZAP can identify vulnerabilities in web applications including compromised authentication, exposure of sensitive data, security misconfigurations, SQL injection, cross-site scripting (XSS), insecure deserialization, and components with known vulnerabilities.

6 Key Capabilities of the OWASP ZAP Tool

- 1. Active vs. Passive Scans
- 2. Running Scans: Desktop vs. API
- 3. Authenticated Security Scanning
- 4. WebSockets
- 5. OWASP ZAP Fuzzer
- 6. AJAX Spidering
- ZAP sits between a web application and a penetration testing client. It works as a proxy—capturing the data transmitted and determining how the application responds to possibly malicious requests. Professionals of various skill levels and job roles can use OWASP ZAP.

1. Active vs. Passive Scans

- ZAP offers two types of scans—active and passive. Passive scans check HTTP requests and application responses for known indicators of security vulnerabilities and cannot make changes to requests. Active scans can create and modify requests sent to the application, sending test requests that surface vulnerabilities you cannot catch using a passive scan.
- Active scans are generally considered more effective in finding application vulnerabilities because the testing suite injects various requests that surface vulnerabilities. However, these scans actively attempt to attack the application and might create or delete data.
- Passive scans pose a low risk, as they cannot change the data. However, these scans cannot catch many vulnerabilities, including aggressive vulnerabilities like SQL Injection (SQLi).
-

2. Running Scans: Desktop vs. API

- You can deploy OWASP ZAP as a desktop application or automatically via an API, depending on how you intend to use ZAP. Security analysts and penetration testers often run a one-off test utilizing the desktop application to detect vulnerabilities. Software development and security teams usually deploy ZAP via automation to ensure regular security testing of the application and its APIs.

3. Authenticated Security Scanning

- Many web applications require authentication. In this case, authentication must be configured in ZAP before running the scan. Otherwise, the check will not test any paths or paths that are behind authentication protection.
- ZAP supports a variety of authentication formats, including form-based authentication, script-based authentication, JSON-based authentication, and HTTP/NTLM-based authentication.

4. WebSockets

- WebSockets create an asynchronous communication channel between client and server, transmitting data in full duplex. This creates security vulnerabilities, because the WebSocket keeps the channel open, allowing attackers to eavesdrop or hijack the session. ZAP continuously scans WebSockets to identify vulnerabilities.

5. OWASP ZAP Fuzzer

- Fuzzing is a technique that sends large volumes of unexpected data inputs to a test application. OWASP ZAP enables fuzz testing of web applications. You can choose one of the built-in payloads, download a variety of payloads provided by the ZAP community, or create your own.

6. AJAX Spidering

- In a penetration test, AJAX scraping can help detect requests from AJAX rich web applications that normal crawlers cannot detect. ZAP provides an AJAX Spider window, accessible through the tools menu. The tool has configuration parameters such as maximum crawl depth, maximum crawl status, maximum duration, and other options to avoid infinite crawls.

Passive vs. active scanning

- ZAP uses two forms of scanning:
- **Passive scanning** investigates all proxy requests and responses, but does not change the response in any way and is considered safe. It can be done on a background thread so it doesn't slow down the application. This can find some vulnerabilities and can help you understand the basic security posture of a web application.
- **Active scanning** attempts to find additional vulnerabilities using known attack vectors against the selected target. Do not use active scans against targets you don't have permission to test, as active scans are real attacks that might cause damage to the
- Reference: https://www.hackerone.com/knowledge-center/owasp-zap-6-key-capabilities-and-quick-tutorial

10 Web Application Security Best Practices You Need to Know

- Reference: https://medium.com/@Imaginovation/10-web-application-security-best-practices-you-need-to-know-30a6e2fed1a2

owasp tools

- https://owasp.org/www-project-web-security-testing-guide/latest/6-Appendix/A-Testing_Tools_Resource