RESEARCH MANUAL

WEB BROWSER ADDON FOR DETECTING SQL AND XSS VULNERABILITIES

COLIN BROPHY
CYBERCRIME & IT SECURITY
30/04/21

Contents

INTRODUCTION

People are making web apps all the time. Any modern problem can be solved with a few lines of code. Previously, making the app and its core functionality was enough, but not in today's world. We have become more security focused than ever as we venture deeper and deeper into the digital age. Security has become just as important as the core functionality of today's web apps for without it, you can suffer downtime, or worse, data breaches. Developers run on tight deadlines every day, pushing out betas as they get to them, and using software that integrates well with their workflow is a must.

I envision to create a tool to help lock down the security side of web apps, while being easy enough to use that it will slot into any web dev's workflow seamlessly. I will begin this report by talking about what web browsers are currently being used, followed by the type of web vulnerabilities dev's face on a daily basis. I will then consider which browser has the best support for dev's looking to create addons and, finally, talk about existing software that accomplishes what I'm looking to do.

WEB BROWSERS

CHROME

HISTORY

[WIKb] Chrome is a web browser developed by Google, first released in 2008 for Windows XP and was solely designed for Windows. Later after its success, it was ported over to Linux, MacOS and Android and has become the most used browser of them all. Chrome began its life as an open source project which, at the time, was called "Chromium". It has since then become a fully licenses free-ware product.

The CEO of google, Eric Schmidt, never wanted to create Chrome, as he didn't want to get involved in "browser battles" which would involve legal battles. After a few Mozilla Firefox employees were hired and they created a demonstration of what would become Chrome, Eric said "it was so good that it essentially forced me to change my mind". It took Chrome around a year to release MacOS and Linux versions, in June 2009. These early versions that were ported were missing many features and were designed for users to test for feedback.

FEATURES

Chrome made the risky decision of making their user interface extremely minimal. This was while they were in competition with Firefox and Internet Explorer, both of which loved their uses of toolbars and buttons for everything. This is mainly what made Chrome appeal to a lot of end users, myself included. From the moment you install chrome and open it for the first time it is very unobtrusive.

Chrome has the basic features every browser needs and more. Tabs, Bookmarks, History etc are all features of every browser. The browser supports the latest HTML 5 and CSS3. Chrome adds to this with its use of settings synchronisation between browsers using the same G-Mail. More advanced features of Chrome includes a password manager, which can create secure passwords and store them automatically for sites you sign up to, making it easier to log into sites you use frequently, though its security is debatable. Chrome also supports Incognito browsing, where history, site data, form inputs and cookies aren't stored for that session.

FIREFOX

HISTORY

[WIKa] Firefox has been around a lot longer than Chrome but hasn't managed to grab the 1 spot. It first came out in November 2004 and was a main competitor to Internet Explorer that Windows had. By the end of 2009, Firefox had taken over Internet Explorer 7 as the most widely used browser, though not every Internet Explorer as a collective. Firefox went through numerous name changes in its early days.

Starting out as "Phoenix" this was quickly shit down due to a trademark claim from Phoenix Technologies. They then moved towards "Thunderbird" and insisted to the Firebird database software project that there would be no confusion as they would always call it "Mozilla Firebird". This never worked out and instead they chose the name we all know today, "Firefox". Firefox really took off as an alternative to Internet Explorer due to backlash on Internet Explorers poor program design and lack of security. Microsoft did release a service pack for XP to fix these issues, but they had already lost trust, and Firefox swooped in as a faster, more secure alternative. Why go back?

FEATURES

Like Chrome, Firefox contains the basic functionality every browser needs, history, private browsing, tabs etc. Firefox expands on these functions with extras such as developer tools (Error Console, DOM Inspector). Firefox has some other popular features that are available in other browsers, but they were early adopters. These features include an integrated pop-up blocker, custom themes and an extension mechanism to allow 3rd parties to develop addons for any features they feel are missing (like a web app vulnerability scanner!). The addons are primarily coded in web programming languages like HTML, CSS and JavaScript.

One feature of Firefox that makes it stand out is the ability to take a snip of a webpage from directly within the browser. It allows you to take a full screen snip, or create a scrolling screenshot for longer pages. Firefox's other party tricks include a handy "Reading Mode" which strips sites down to the basic text, removing any unrelated info from the webpage, allowing you to focus on what you're trying to do or read. Finally, Firefox comes with a built in PDF reader, so no going to external applications to open a PDF you downloaded from a site, it's all in-house.

CHROME AND FIREFOX, A COMPARISON

Browser	Similar Features	Benefits over Competitor	Downsides
Chrome	- Focus on security - Feature rich - Addons - Speed - Ease of Use	- Google account integration - More addons - Bigger user base - Simplicity	- Higher RAM usage - Poor thought for user privacy
Firefox	- Focus on security - Feature rich - Addons - Speed - Ease of Use	- Focus on privacy - Speed	- Less addons available - UI isn't as simplistic as some would like

Chrome and Firefox both have the same fundamental components as discussed above. Neither one really has any feature that makes it stand out more than the other. The reason some people tend to prefer one over the other is usually habit, which browser did that person use first? If it was Firefox from the get-go, it's hard to move from what you're used to.

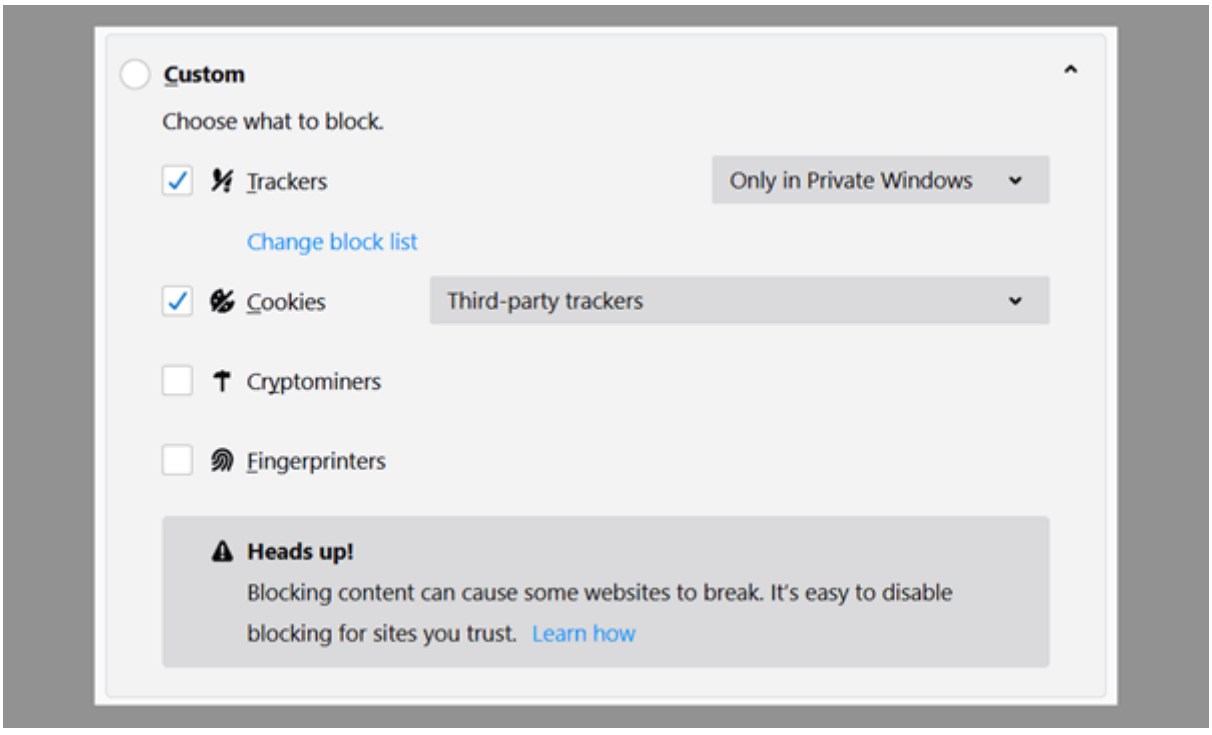
Personally, I've given both a fair shot and I'd prefer Chrome over Firefox for its simplicity and minimal design. Firefox does shine above Chrome in some ways regarding reliability. In

Chrome's early days, and even now and then, Chrome tends to eat up and available RAM on whatever machine it's running on, which can slow down the rest of the system depending on the available memory. Firefox hasn't had such issues and has gained a lot of trust among users that way.

Security of each of the browsers is pretty much on par. Both have pop-up blockers but neither have ad-blocker, though this can be got for either one from their extensive addon libraries. Both browsers warn you when the site you're visiting isn't using HTTPS, which is always handy. Chrome is slightly ahead with this one, where the phrase "Not Secure" appears in the URL bar, unlike Firefox which just gives a "!" that most users would never notice.

When it comes to Privacy of each browser, Firefox is the clear winner. Google has a track record of disregarding user privacy and terrible policy standards, while Mozilla's has a much cleaner record with a very well thought out privacy policy. Google is well known for tracking everything and anything, hence why you get perfectly targeted ads. Chrome is also very sneaky with most data collection settings on by default, such as auto-fill URL prediction and search suggestions.

Firefox on the other hand is a non-profit organisation, meaning they don't make money from ads and therefore can be trusted when they say they aren't tracking you, they have no incentive to do so. The privacy settings are unmatched by Chrome with every setting being tweakable and toggleable.

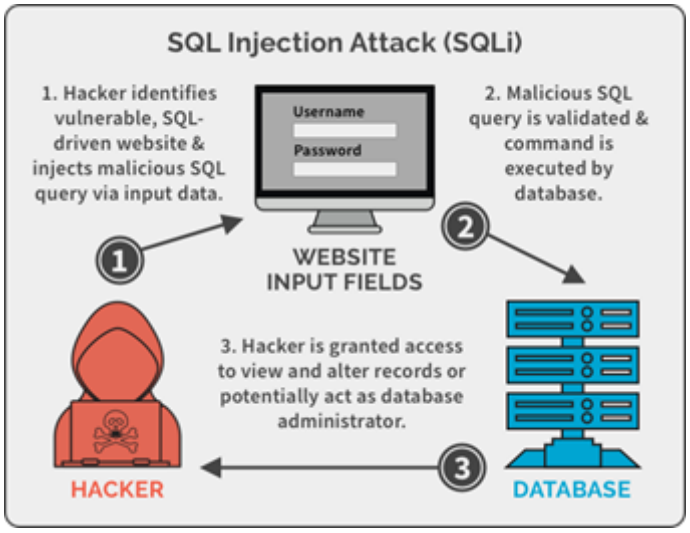


WEB APP VULNERABILITIES

WHAT IS A WEB APP VULNERABILITY?

A web app vulnerability is the exploitation of a flaw in a web apps code which allows a malicious user to manipulate it in such a way that it does something other than its intended use or makes available to the user hidden information that shouldn't be seen by anyone. These vulnerabilities pop up due to lack of focus on security. Web app exploits differ from the typical network or device exploits. They pop up due to the fact that web apps need to interact with multiple users across multiple networks, this level if accessibility then being taken advantage of by attackers.

SQL INJECTION



[HER] SQL Injection takes advantage of the Structured Query Language which is used all around the world for database management. SQL Injection occurs when an attacker is able to slip their own SQL commands into a database, usually through un-sanitized or validated user input forms. The inputted commands can then change, delete or print data as all SQL commands are run with full privileges.

If an attacker plays their cards right, entire databases with first names, surnames, bank details, addresses, etc can be printed out on-screen. Other uses for SQL Injection include privilege escalation. Using SQL, we can trick the password input field to return TRUE and log into any username without their password.

CROSS SITE SCRIPTING

[Impb] Attackers looking to target a web app in order to get data from it or gain access to it would use SQL Injection to do so. Malicious users looking to wreak havoc on the web apps users would utilise Cross Site Scripting instead. Similarly to SQL Injection, XSS utilises injecting code through user input forms in order to execute what we want. In the case of XSS, the code is run in the users browser when they visit the site, completely unknown to the user. There are three main types of XSS:

REFLECTED XSS

Reflected XSS is the simplest form of XSS attacks. An attacker formulates a URL with their payload in it and hides it in the form of a hyperlink to conceal its true form. Once the link is clicked, the payload is injected to the site and the result is reflected back to the user. This can be in the form of a fake login page for example.

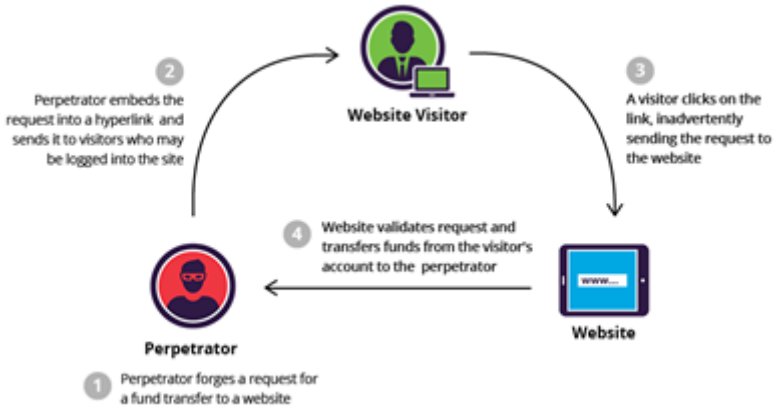
PERSISTENT XSS

Persistent XSS is similar to reflected but much more dangerous. In this case, the payload is maintained on the web page, in the form of a blog post or comment for example. Anyone who then visits the page has the code executed. This makes it a lot more dangerous as there is no relying on the user to click a link sent in an email.

DOM BASED XSS

With DOM Based XSS, the attack never reached the web server, all the code is executed in the users Document Object Model, or, the web browser itself. It is formulated similar to Reflected, a URL is created and sent to the user, only this time the payload never reaches the server to be reflected back.

CROSS SITE REQUEST FORGERY



[Impa] Cross Site Request Forgery is the third most prevalent Web App Vulnerability that web app devs need to aware of. This attack takes advantage of the browsers trust between open tabs. CSRF makes a user perform an action they never wished to perform. With a little social engineering, clicking a link on one website can perform an action on another tab, such as liking a Facebook page, or transferring funds from your bank to the attacker.

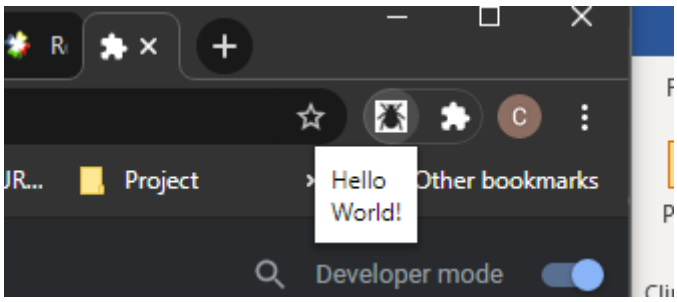
One way which this is accomplished is by having the bank web app form filled out and set to invisible on the malicious website, lining up elements of the malicious site with the invisible button on the banks web app can make a user unwillingly click “transfer”. Provided they have an authenticated session open in another tab, the request will work.

HOW ADDONS ARE CODED

BASICS

[Goo] I have noticed a far superior developer base with Chrome addons over Firefox addons. This could be due to the sheer number of Chrome users or down to how Google treats the browser as a priority. Addons for both Firefox and Chrome are both coded in HTML and JavaScript. I’m hoping that this way it may be a way to “code once, deploy twice” so to speak. If I focus on one browser, i.e. Chrome, I plan for the same code to work in Firefox with ease.

The coding of Chrome addons seems simple enough, judging by their in-depth guides. No fancy software is required, though a nice IDE would definitely make life easier. I am going to give Brackets a shot and see how I like it. With just three files and enabling developer mode in Chrome, I had my extension showing in the toolbar with its own icon and text when I click it!



This is a promising start, with plenty left to do, but is a step in the right direction. It became quickly obvious that the main "manifest.json" file is the heart of the whole extension. In here is where we set the name, description, version, and what scripts get called to perform functions, among other things.

```
{
  "manifest_version": 2,
  "name": "Web App Vulnerability Checker",
  "version": "0.1",

  "browser_action": {
    "default_icon": "images/icon.png",
    "default_popup": "html/popup.html",
    "default_title": "Web App Vulnerability Checker"
  }
}
```

As we can see in my snip of code, the manifest.json file sets the name of the addon that is displayed to the user as well as the current version. We can see my basic "browser action" in the photo, which sets the icon shown in the top bar in chrome and the action, which is to popup the file "popup.html"

The popup.html simply has the words "Hello World!" inside, which is then displayed inside a popup from my addon button! I have done HTML, CSS and JavaScript in my studies, so these being the main languages that extensions are coded in is helpful. I will need to learn browser specific calls and names for things like permissions.

For example if I want to be able to control the page the user has open, I will need to ask for permission to "activeTab" in my manifest.json. Asking for permissions in the manifest means that when the user is installing the extension, they will be warned that the app may make changes to your open tabs.

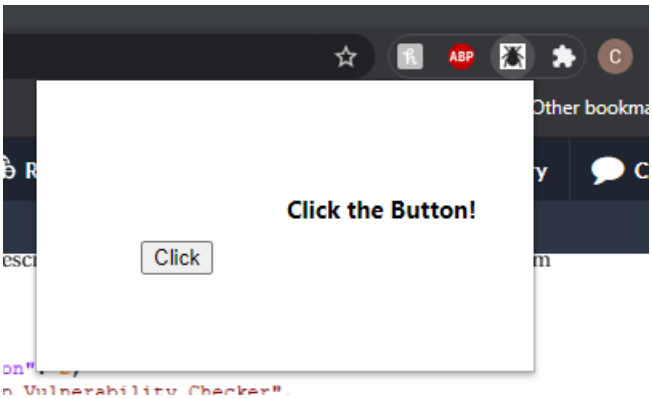
After we have the manifest done, it's just up to the HTML, CSS and JavaScript that I implement to make it look the part! I'm really looking forward to the design stage of thew project to decide how best to place the extension for best usability.

ADDING FUNCTIONALITY

Adding some extra functionality on top of what I already have was fairly straight forward, and we start to see how laying out my user interface will work. I added some HTML and css to the "popup.html" file I have.

```
<html>
  <head>
    <title>Extension Control Panel</title>
  </head>
    <style type="text/css">
    body     {
             margin: 5px;
             }
    h1       {
             font-size: 15px;
             text-align: center;
             }
    .box     {
             width: 300px;
             margin: 10%;
             padding: 10%;
             }
    </style>
    <body>
    <div class="box">
        <h1>Click the Button!</h1>
        <button id="buttonClick">Click</button>
    </div>
    </body>
    <script src="popup.js"></script>
</html>
```

As we can see in the code above, I added some CSS, extra text and a button to my popup.html file. This is the foundation for how all my buttons, input fields etc will be placed.

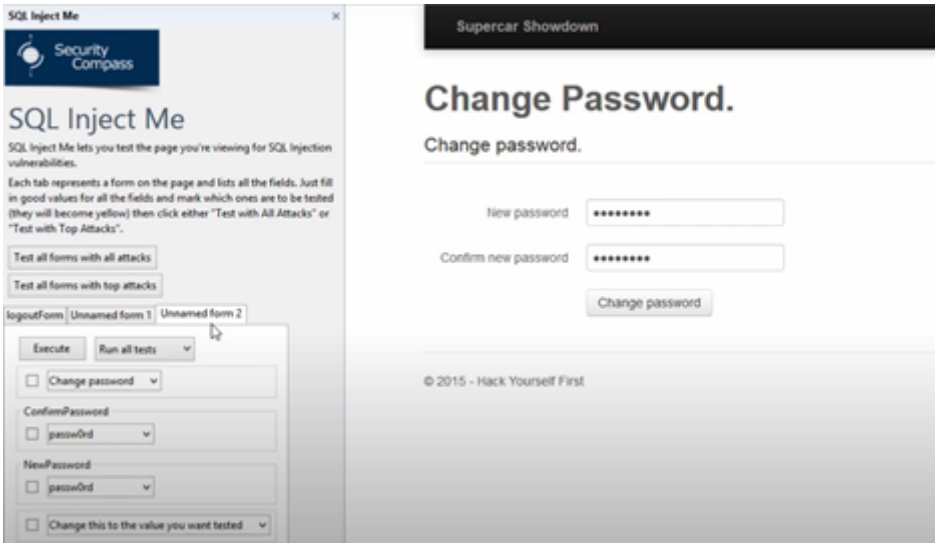


EXISTING SOLUTIONS

There are existing projects that carry out similar tasks to what I want to achieve, both addons and standalone applications. I'm looking to bring the best of each into one place, right in the browser. The following are a few I've come across, with a little bit about each and what I will take from them going forward.

SQL INJECT ME

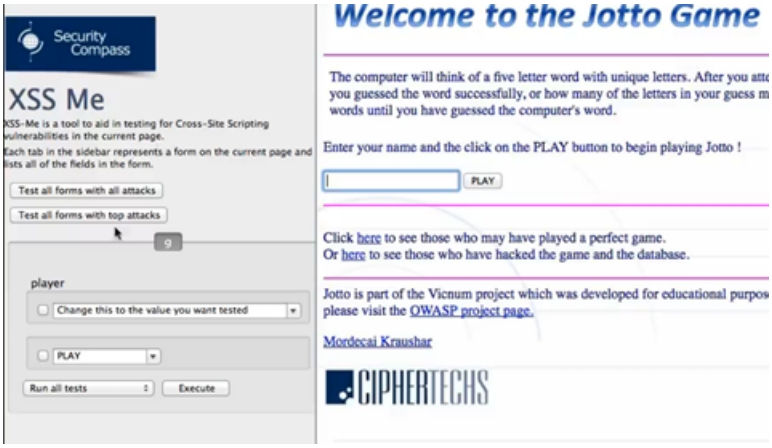
[SQL] SQL-Inject-Me is an example of a browser addons that exists for Firefox and manages some of what I set out to accomplish. This addon opens a side panel within the browser where all the user input takes place. Once it is opened, it scans the current web page for any forms and opens them up within the addon.



For example, with the above website for Supercar Showdown, once the addon is open and the change password page is loaded, the forms on the right are replicated within the addon. From there, we can select what input fields we want to run our test attacks against. SQL Inject Me uses a predetermined list of SQL Injection payloads. Once the payloads have been run against the site, the addon prints out a report, highlighting what worked and what didn't. SQL Inject Me is very much about discovery, rather than exploitation of the site itself, it leave that's up to the user.

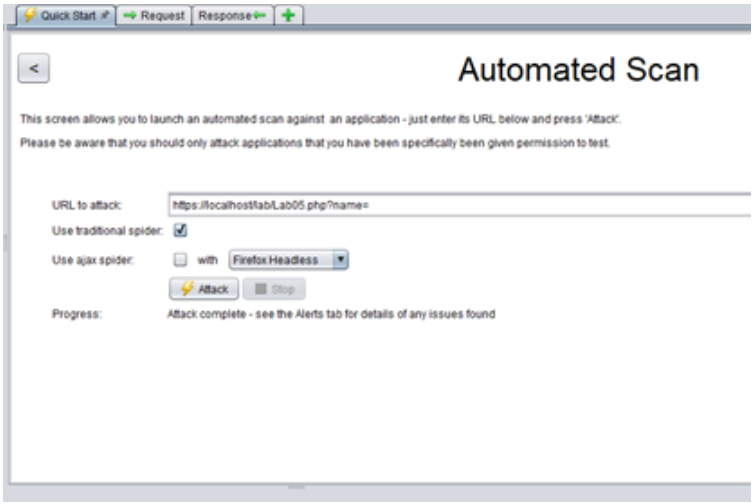
XSS ME

[XSS] XSS Me is made by the same company who made SQL Inject Me. It has the very same layout as SQL Inject Me also in that it opens a sidebar and allows the user to control it from there. With the sidebar open, we get the option to test all forms with all attacks, again returning a report at the end of what it found.

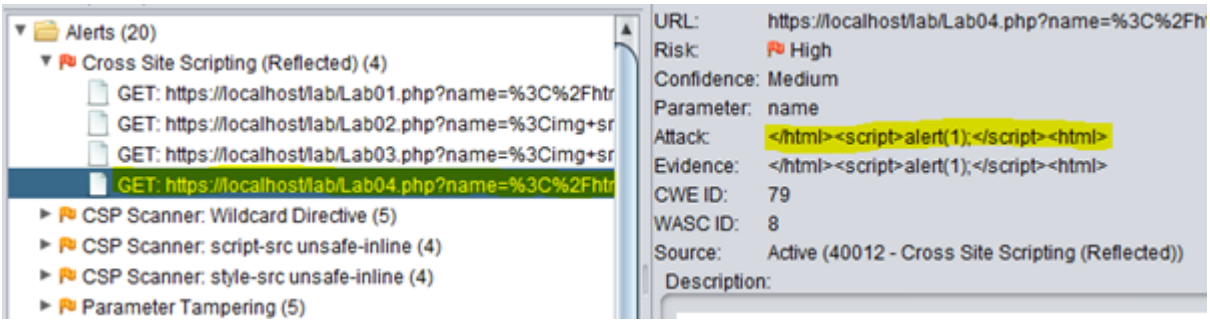


OWASP ZAP

[OWA] ZAP Proxy by OWASP is a standalone software designed also for discovering SQL vulnerabilities of a web app. It works differently to the others I talked about in the sense that it isn't something that runs within the browser, but rather it proxies internet traffic through itself and catches faults/ sends exploits that way. When you launch OWASP you can then launch a browser using a button in the toolbar. That browser session is then proxied through ZAP, it sees all traffic coming in and all traffic going out. One of the more basic uses for ZAP is to run an automated scan on a site:



ZAP then goes through every payload it has at its disposal and checks for ANY security vulnerability it comes across, including SQL Injection and Cross Site Scripting. Once ZAP has finished running its test, they all get reported in an easy to read folder format for each type of attack. In my example, there was a Cross Site Scripting vulnerability in the page, to which ZAP printed out what payload it successfully hit it with.



ZAP is handy in that it uses colour coding to alert us to the severity of each problem it found. As we can see in my photo above, XSS attacks are Red, High priority.

Although ZAP is a standalone application, it can use what they call the HUD (Heads Up Display) within the browser to run attack payloads and alert us to any it found, again using the colour coding to show severity.

Bibliography

[Goo] Google. *Getting Started - Chrome Developers*. URL: https://developer.chrome.com/docs/extensions/mv3/getstarted/. (accessed: 04.11.2020).

[HER] MAX HERSCHAP. *SQL-injection-attack-example*. URL: https://spanning.com/blog/sql-injection-attacks-web-based-application-security-part-4/sql-injection-attack-example/. (accessed: 06.11.2020).

[Impa] Imperva. *Cross site request forgery (CSRF) attack*. URL: https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/. (accessed: 06.11.2020).

[Impb] Imperva. *Cross site scripting (XSS) attacks*. URL: https://www.imperva.com/learn/application-security/cross-site-scripting-xss-attacks/. (accessed: 06.11.2020).

[OWA] OWASP. *OWASP ZAP Zed Attack Proxy*. URL: https://owasp.org/. (accessed: 01.11.2020).

[SQL] SQLInjectMe. *SQL-Inject-Me*. URL: https://github.com/SecurityCompass/SQL-Inject-Me. (accessed: 04.11.2020).

[WIKa] WIKIPEDIA. *Firefox - Wikipedia*. URL: https://en.wikipedia.org/wiki/Firefox. (accessed: 08.11.2020).

[WIKb] WIKIPEDIA. *Google Chrome - Wikipedia*. URL: https://en.wikipedia.org/wiki/Google_Chrome. (accessed: 08.11.2020).

[XSS] XSS-ME. *XSS-Me tool html frames*. URL: http://travisaltman.com/xss-me-tool-html-frames/. (accessed: 02.11.2020).