Good morning,

Today I will be presenting you to Gentoo – my individual project focused on detecting web app vulnerabilities.

We start with the following motivation – does anyone know what this is?

A CVE – yes, a common vulnerabilities & exposures ID – but do you know which this is? I’d be a bit worried if you did…

If I add the following to the slide does that give more of a clue? (Apache Struts)

Apache Struts is an MVC framework for website design using Java’s Servlets. It was used by none other than Equifax – the credit score company which has recently been in the news.

Equifax’s data breach affected over 143Million Americans, and posed risks to users abroad as well. A host of sensitive personal data was stolen in this attack, including people’s names, birthdates, social security numbers, driving license scans, CC info, and more.

All of this data was stolen due to an interpreter bug in Apache Struts – to go into more detail, the value of the *Content-type* header was incorrectly interpreted, and thus resulted in a remote code execution (rce for short)

In the post-analysis of this attack, security analysts detected that there were 2 distinct attack types here – probing, and malware distribution

In the probing attacks, attackers test a proof of concept – as seen in the slide, the *whoami* command is injected to ascertain whether 1) commands can be injected and executed and 2) whether the machine treating the requests is running elevated privileges.

In the malware distribution phase, attacks become malicious – in this slide we see a command injected which disables the firewalls, downloads and runs a malicious file, and ensures the firewalls remain off upon reboot.

This particular attack falls is a type of injection. According to OWASP (the Open Web application security project), an authoritative organisation on security measures, Injections are of their latest report the largest security concern for web applications today.

“Surely the industry already has tools to minimise this damage” I hear you say. And they do – there are several tools for this, both commercial and open-source. Among these are OWASP’s own ZAP (Zed Attack Proxy), the well known Burpsuite, and Acunetix.

However, all of these tools are fully automated scanners, and all operate in the same way.

As we’ve already seen, probing and malware are delicate operations which are likely to require user analysis and response – inputs to these are hard to generate automatically.

Thus, our problem is set. However, we wish to explore how well a semi-automated scanner would perform (as opposed to fully automated). This makes sense as a pentester or web analyst would need to analyse and react to the data regardless.

Enter Gentoo!

Gentoo is also a web app vulnerability scanner, but the main difference between it and the others we have just considered is how closely it uses human interaction – Gentoo mainly uses the fact that vulnerability detection and exploitation is largely a human task.

It does this by exploiting the privileges given to a browser extension – in this instance I’ve selected Google Chrome as the browser to extend. This way Gentoo sits right between the website and the browser, and skips many of the difficulties associated with crawling that automated scanners have. For those who do not know what crawling is – it is a method used by scanners to find out the scope to which they can attack a website – a crawler finds any and every possible link on a site and builds up an internal database in the tool. The more there are, the better, giving more scope for an attack. If the scanner misses some however, the scan output and quality will be significantly degraded. Gentoo doesn’t need this because it operates where the user is browsing.

The key contributions of Gentoo are the following 3 modes – Recommendations, the passive mode, and the Action replay mode

Starting with the recommendations – the tool scans the HTML page for any <input> tags. Depending on the sensitivity desired, it creates a sibling node on the DOM (document object model) which can be clicked. Once clicked, this submits the form’s input with a probing payload, that would allow the user to ascertain further information from the website.

On this slide we can see how the test harness page we’ve created changes with the recommendations -> The “Investigate form” buttons are added

The probing payloads we send out are designed to reach the Request logger – if we’ve done that, we’ve successfully executed our own Javascript – which is a certain vulnerability.

As we can see, the request logger’s only duty is to report URL’s from which we have executed JS – meaning these are weak.

Now I’ll walk you through a demo of the recommendations at work.

So that’s it for the recommendations.

Now we look at the passive mode – this is designed to scan and analyse request and response headers. It flags up any insecure settings of select headers, and suggests the recommended secure setting. On top of this it also performs basic CSRF and Cookie security scans.

It also features a more experimental cross check mode – this is designed to find second-order vulnerabilities. Second order vulnerabilities are seen in attacks whose effect is only seen at some point after the malicious input injected – not immediately. Cross checks tries to combat this by analysing each new request against previous requests. The extent of the analysis is determined by a configurable window size, which dictates how many requests ago we should look at for any given request.

Now a little video showing what happens in passive mode

Now, onto the Action Replay mode

The principle here is to allow the user to select specific inputs of their own volition and generate focussed attacks against these inputs.

In this mode, a session begins recording the user’s inputs.

Once the user is happy that the input they are looking at has been ‘attacked’, then they can stop the session

At this point, the replay begins, and Gentoo generates several attacks on the user chosen input, each with slightly different payloads.

Let’s see a demo of this in action

After seeing all these 3 modes, you might wonder whether there are any real world applications to Gentoo. I will now guide you through an example of a real vulnerability I’ve uncovered on a live website as a result of using Gentoo.

As part of development, it would often be the case that the Recommendations mode would be left on, and I would sporadically attempt different probing payloads against a number of websites whilst normally browsing.

It would often be the case that I received interesting outputs from websites as a result of submitting these payloads.

As an example, one of these generated a CSRF error from a website, despite the payload not successfully executing.

More often than not there were also results which indicated some information about the website which would otherwise not be readily available.

In this slide, we gathered from this attempt that the website is running Wordpress, because they’re using the Wordfence extension to stop security attacks.

In the next example we see how the recommendations payload has unexpectedly affected the layout of the form.

I will attempt a live demonstration of this vulnerability and how it can be exploited.

Now, we analyse the progress made by Gentoo in relation to other tools. To do so, we’ve benchmarked Gentoo against 2 other scanners, ZAP and w3af. We’ve done this across 4 different vulnerable applications – the Test Harness we’ve made, DVWA, WebGoat and WackoPicko.

The benchmark involves testing 3 different metrics for success.

Firstly, the time to first vulnerability. This mainly tests the speed of the extension against the other scanners.

Then, we measure the number of replays required to find the first vulnerability. This metric should measure both the speed and efficiency of the scanner – if they are performing the best attacks first then this metric should be lower.

Finally, we measure the interaction volume of a scanner. This also measures the scan efficiency, and can be used as a comparison point for the stealth of any given scanner.

For all of these, the lower the measurements the better the tool is performing.

We performed different scans. In blue is Gentoo’s XSS detection using the Action replay algorithm Orange is the passive header scan, grey is ZAP’s performance, and yellow is w3af. As you may notice, I was unable to configure w3af’s authentication module to sign into dvwa and WebGoat, which is where most of the interaction required for the scan is. The action replay scan also didn’t work in WebGoat.

From these statistics, we can see that Gentoo’s scans are fairly consistent – As expected the action replay takes significantly longer than the passive header analysis. However, we also take into consideration the human interaction in the Action Replay scan. Despite this, it still achieves a competitive performance when pitted against other scanners.

Now we look at the number of replays (otherwise attacks) required to detect the first vulnerability. Clearly, ZAP seems to take far longer than any of the other scans to dfirstly detect a vulnerability. However, it’s also made clear here the difference between Gentoo and other tools – Gentoo is targeting a focussed attack, whereas the attacks done by ZAP and w3af target the entire application; Gentoo reports one vulnerability type per scan, whereas the other scans report any vulnerabilities they might have found in the application. Thus, to make this comparison fairer, we standardise the results by comparing at what percentage of the scan completion we have found the first vulnerability.

Doing so makes comparison easier.

We see here that w3af has very low scans, ZAP has high fluctuations across the board.

However, skewed against smaller scans – for the webgoat header scan the first attack detected a vulnerability. But because there were only 3 attacks, this generates a misleading 33%.

Interaction volume ridiculous for zap. Skewed against larger scans. Normalise the data by dividing the respective numbers by the number of confirmed detected vulnerabilities found by the scanner.

Even after normalisation, it’s clear that ZAP creates a considerably larger gap between Gentoo and w3af.

Gentoo has consistently very low numbers for this figure.

We sum up with the following final remarks – it’s clearly difficult to compare Gentoo against other fully automated scanners, as they are tools of different natures. As you have seen from the metrics, exceptions and normalisations were attempted to make the comparison a bit more fair, though this is still difficult.

Gentoo strongly benefits from its position as an extension, which increases its compatibility because it goes where the user goes, no crawling or extra authentication to configure, which is a major advantage over the other scanners, though some of their features are already matching this level.

However, Gentoo is still currently a bit difficult to use – as was shown by preliminary user research, it was unclear how to use the tool to detect vulnerabilities without my guidance. A help or tooltips page would have been much more useful

Nonetheless, Gentoo was still the driving force behind finding a live vulnerability, and thus performed its job correctly in this sense – it was a success in this aspect.