Logo

Description automatically generated

Fuzzing

[What is Fuzzing? 2](#_Toc134920455)

[Benefits of Fuzzing 2](#_Toc134920456)

[Tools Used 2](#_Toc134920457)

[Examples of Fuzzing 2](#_Toc134920458)

[Results 5](#_Toc134920459)

Contents

# What is Fuzzing?

Fuzzing is a technique that is used to uncover vulnerabilities and potential weaknesses in a website, fuzzing tests different input variations to test the functionality of the website. Fuzzing can also be used to test for bugs that may not occur during regular testing, during fuzzing sessions it can cause apps/websites to crash or behave differently. OWASP (2020)

Fuzzing will inject random data in sections of a program (or in this case webpage) to test for input validation and we can see how the website handles unexpected pieces of data.

# Benefits of Fuzzing

Fuzzing can show the build quality of the website and can expose different security vulnerabilities (as mentioned above) that may not occur in regular testing scenarios. Fuzzing can also be quick to test compared to things such as stress testing and port scanning. Downsides of fuzzing can include not finding more common bugs. Synopsys (n.d.)

# Tools Used

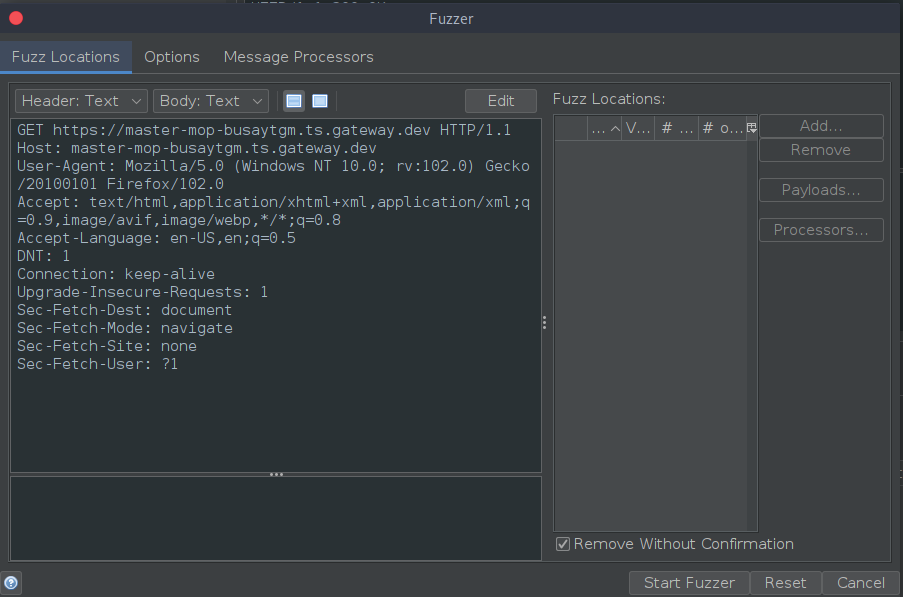
Parrot OS

Owasp ZAP

ChatGPT

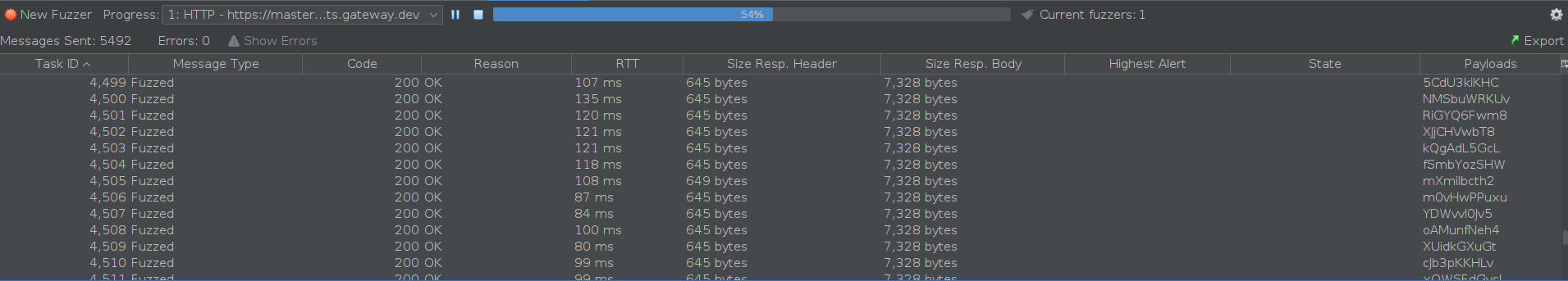
# Examples of Fuzzing

In my testing I used Owasp ZAP to first test the User-Agent header of packets. The strings that I used were just randomly generated from random.org, and surprisingly I was able to get a response with every one of those random strings in the user agent. Fortunately this did not cause the server to break or having unexpected behaviour, this shows that the server can process a wide variety of different strings without having any errors. Doing this also showed that there aren’t any immediate vulnerabilities that can be exposed.

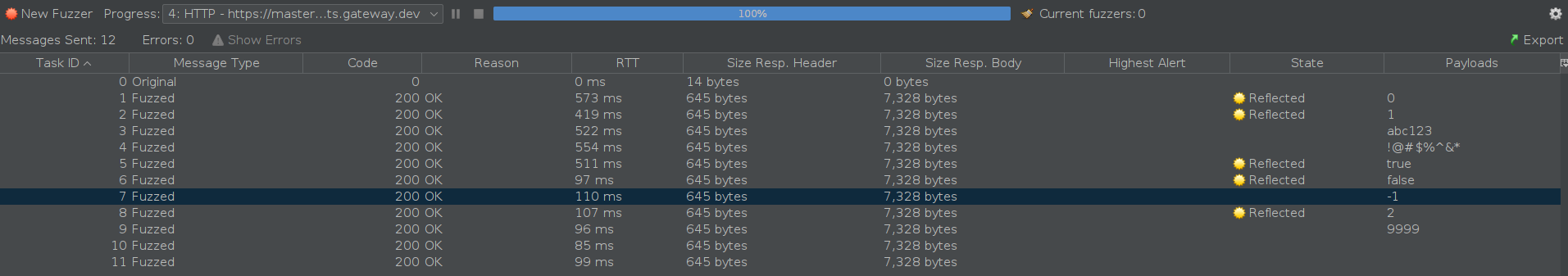


Above is a screenshot of the packet I was fuzzing, in my first lot of testing I was using random strings within the User Agent header.

I figured this was a good place to start since it shows it can handle different browsers (since they would all have different headers) without issue.



The next test to run was testing the Do Not Track field and seeing what the server reacts with



In this case (above) I fuzzed this so that there would be multiple values including things that should trigger out of bounds errors (-1, 2, 9999) and random characters. However upon further investigation of what the reflected status meant, it simply meant that there is a part of the code that reflects the value that is being fuzzed. This is nothing to be worried about, as for some parts of code it is impossible to avoid values like 1 and 0. The only significant thing to note here is that while the server may be configured to handle anomalies like the invalid characters and out of bounds integers we can also see that the server just straight up rejects the packet when the DNT header is set to blank (even though 0 is technically the same value and the packet still gets a response)

A screenshot of a computer

Description automatically generated with medium confidence

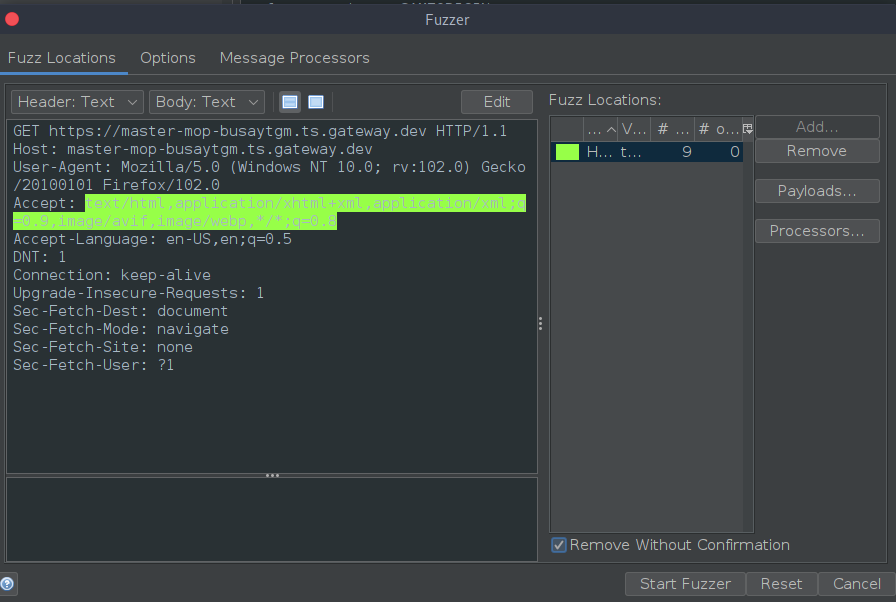
My next step was fuzzing the host value and seeing what the results were.

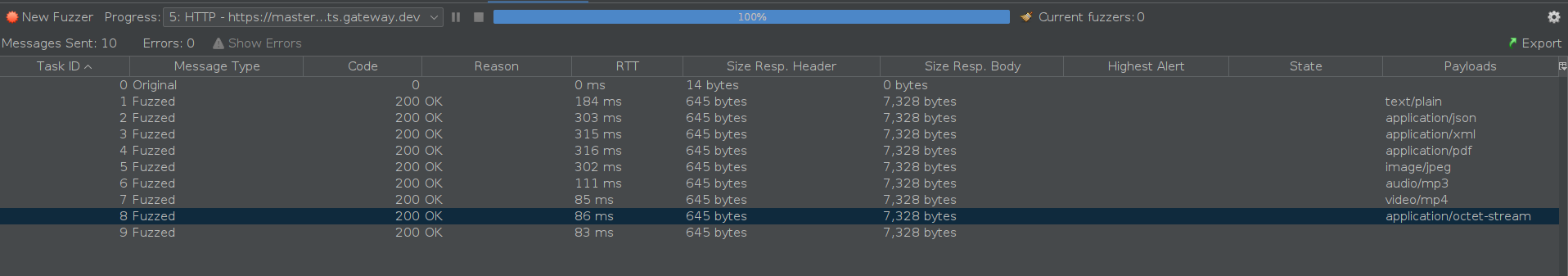
A screen shot of a computer

Description automatically generated with medium confidence

As we can see in the above screenshot we are getting a response from the website/server and at first this seemed confusing to me until I asked ChatGPT why this would be the case especially since those domains that I provided would be invalid and obviously not the actual host of the website. Upon checking with ChatGPT this is normal, since it just shows that the server is secure and it is handling routing requests without having to rely on the Host header, if it were relying on the host header then different occurrences such as error handling or falling back to the default page, or even declining the connection completely.

My next fuzzing attempt was testing the different types of file formats that can be accepted by fuzzing the Accept header (See highlighted area below).





And as shown in the results here it shows that this website can accept different content types without any hesitation, from regular application files all the way up to video. The website responds appropriately and doesn’t cause any issues. This can mean that when the time comes to scale up the website and add different types of content there will be no issues, only thing that will need to be done is to change the get request (as the fuzzing here has done) to accommodate the new file types.

# Results

The results of these tests show that the data for this website is being sanitized properly. This report has shown that there are safeguards in place to handle unexpected and potentially malicious host values. It has also been proven that different browsers and potentially spoofed headers fields can be used and there will be no impact to the website. The website can also take different media types and display them correctly. It shows that the implementation of request handling and content negotiation has been well implemented. And while fuzzing can’t show if a server is immune to all vulnerabilities, more tests will be conducted, and more results will eventually be found.