Nick Forleo

INSC 561

**Task 1 - ZAP: Use ZAP (or Paros) to intercept some pages retrieved from a target website**

Fig. 1.1

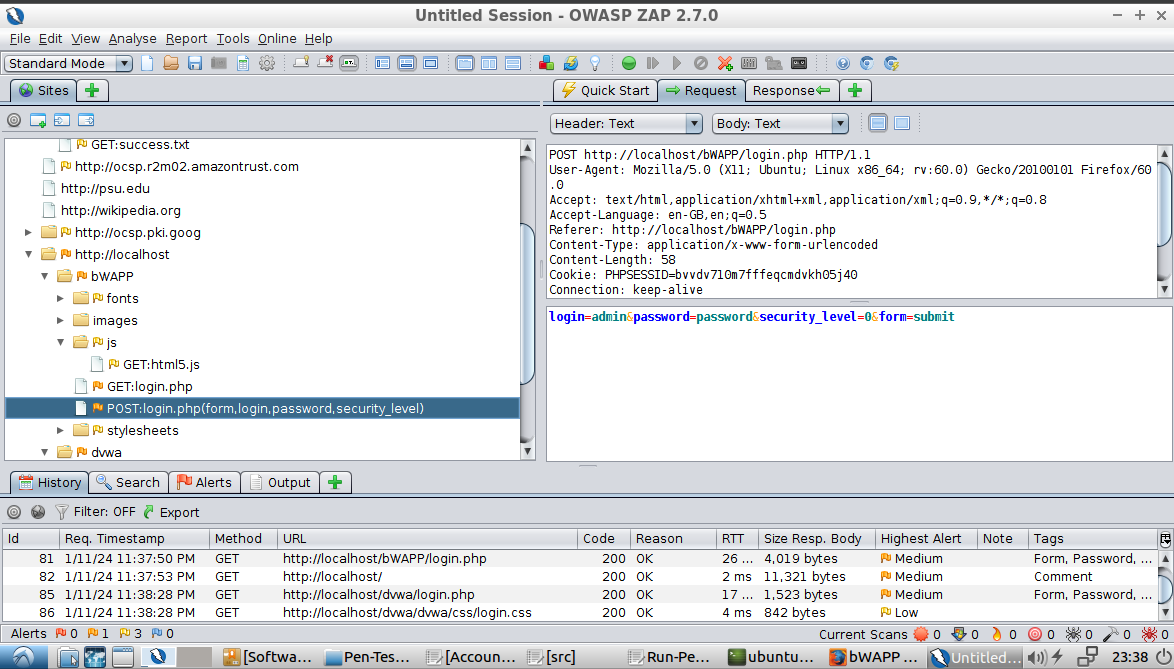
Fig. 1.2

Fig 1.3

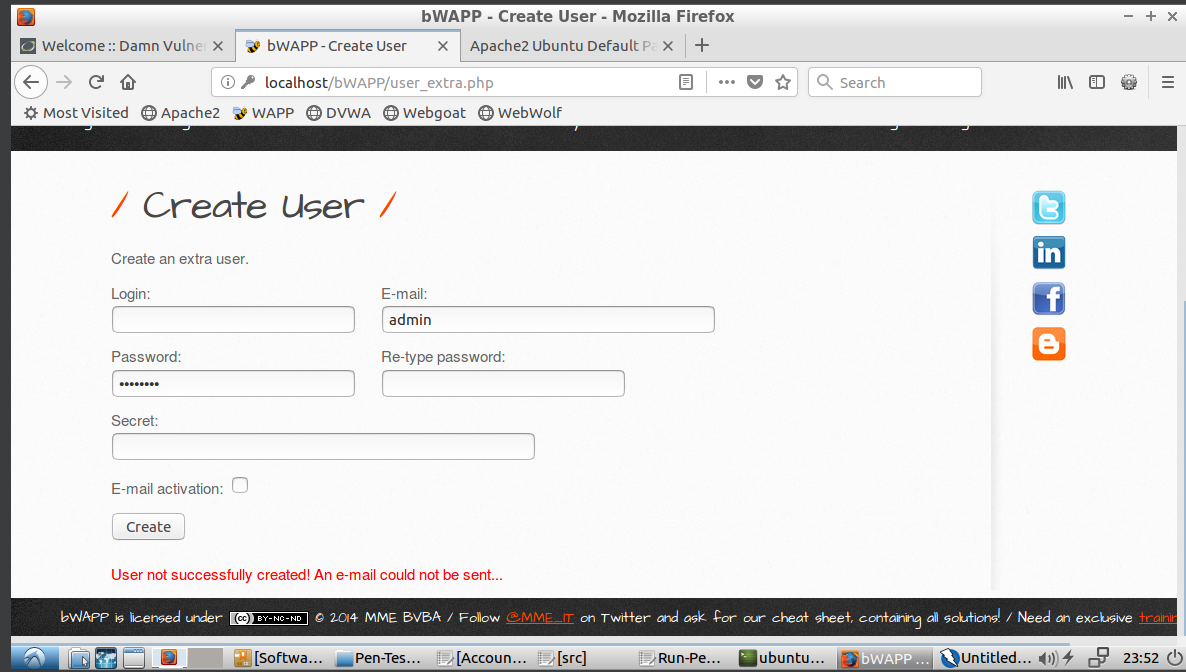


Fig 1.4

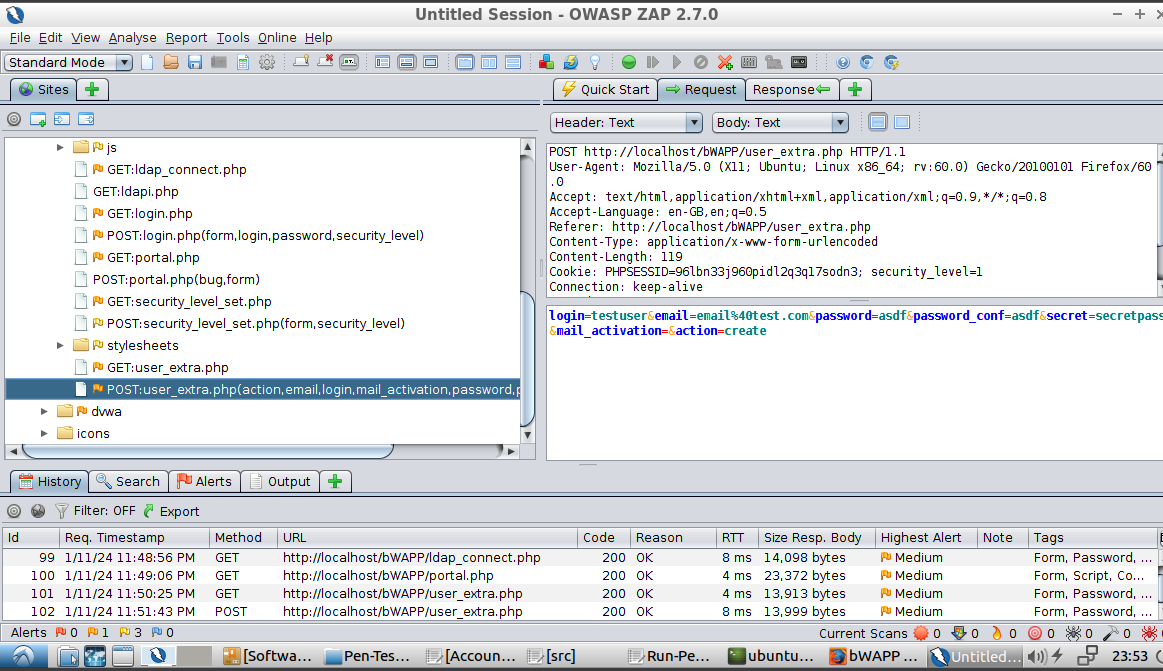
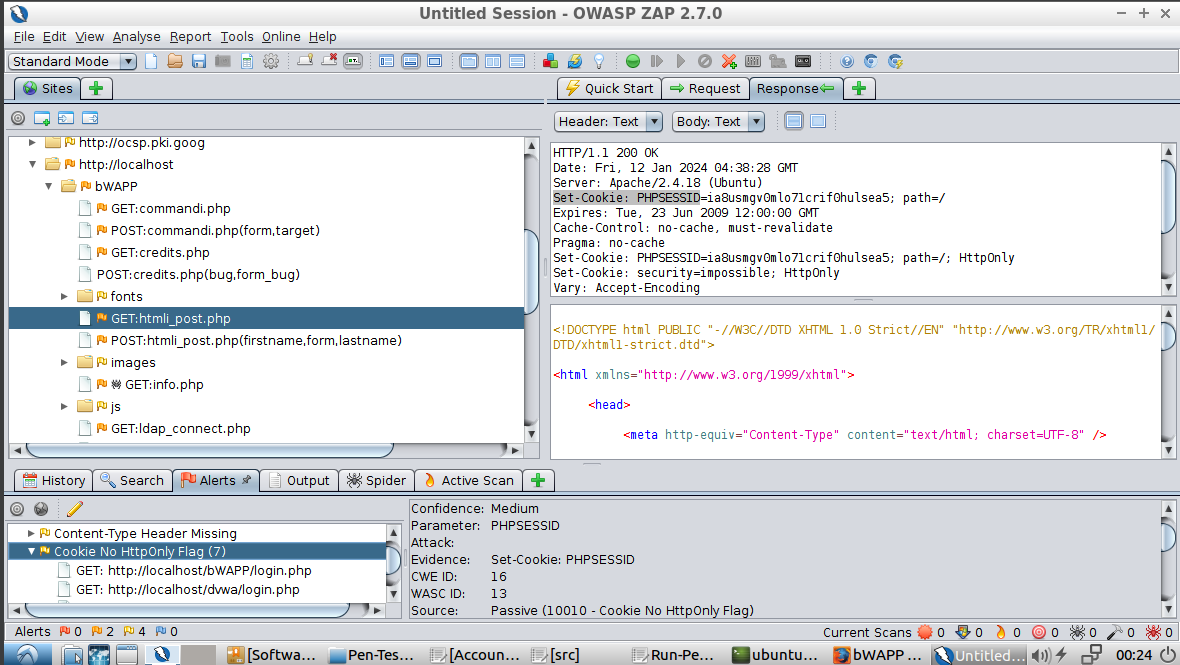


Fig 1.5



These screenshots are of bWAPP or data being sent between me (the client) and the server (the website) via the ZAP application. Figure 1.2 shows the data being sent in the login request and Figure 1.1 shows where the user would click to initiate the request. In this case, it is a POST request that contains the following data fields: login, password, secutiry\_level, and form. Figure 1.4 shows the data being sent in the request that creates a new account while Figure 1.3 shows where on the UI the request is triggered. This request contains the necessary data to create an account, which in this case is login, email, password, password\_conf, secret, mail\_activation, and action are sent. For both of these requests, a hacker that is snooping on sent packets can see the data in plain text. This means that whether you are logging in or creating a new account, the information can easily be taken and used elsewhere. The final screenshot, Figure 1.5 is the output of running an automated scan. This was interesting because it scans the web page and finds vulnerabilities. One that caught my eye was in the highlighted GET request. It found that the response from the server that sets the cookie for PHPSESSID doesn’t include the HttpOnly flag. This means that a hacker can spoof your session ID outside of the browser (i.e. executable code) and can access your data and attack it programmatically.

**Task 2 - Burp Suite: Use Burp suite to intercept some pages retrieved from a target website.**

Fig 2.1

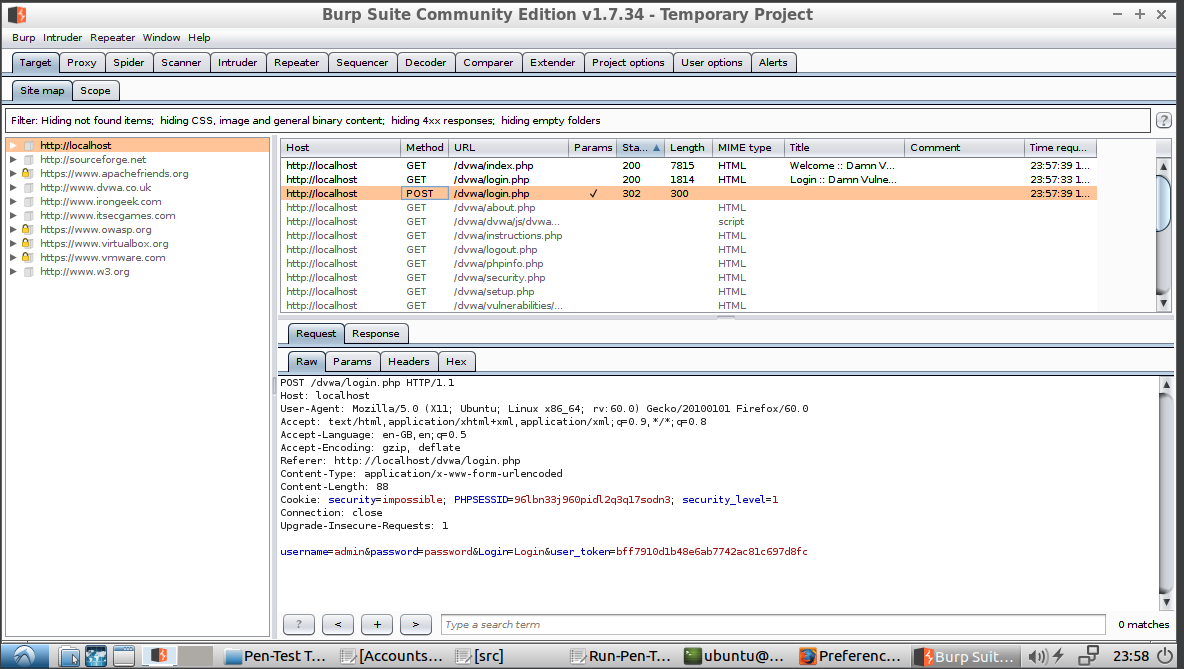


Fig 2.2

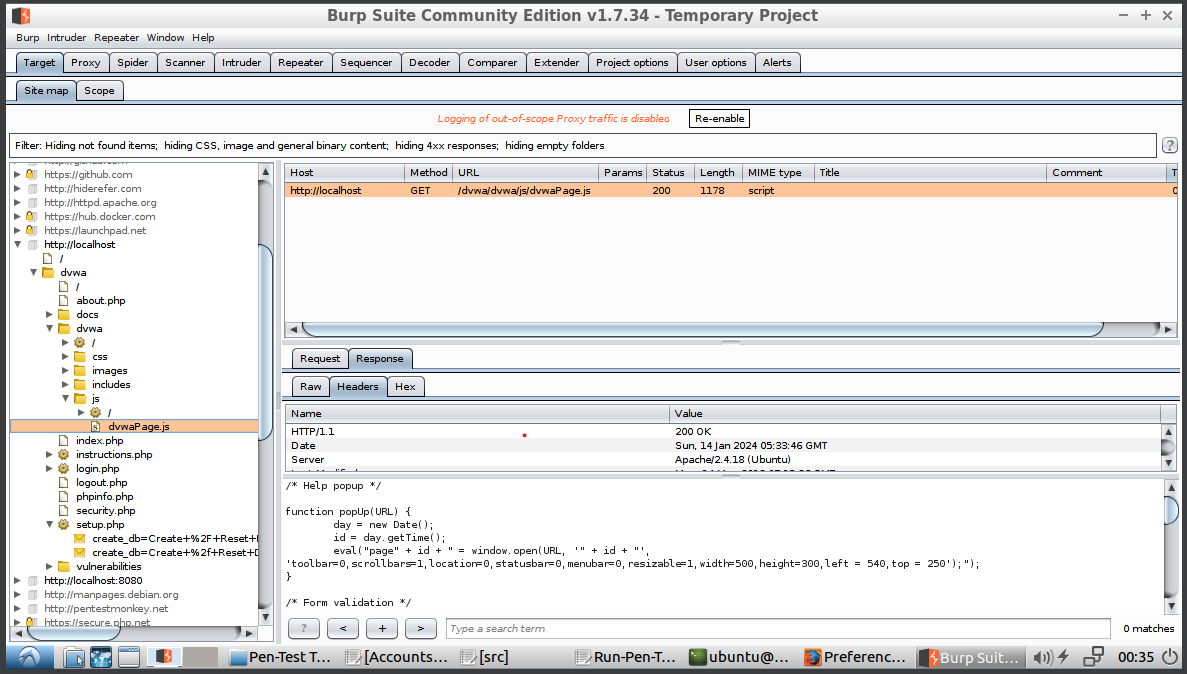
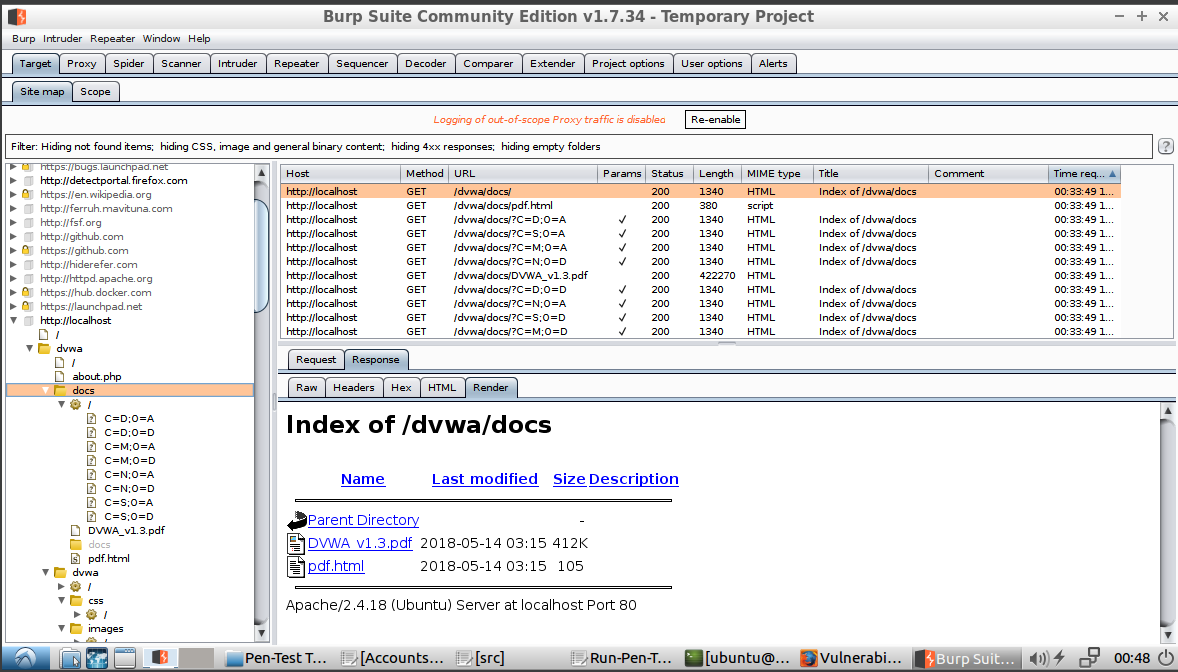


Fig 2.3



The three figures shown above include data being sent between the client and DVWA captured via BURP. Fig 2.1 is the client piece of the conversation that is sending the login information to the server (username, login, password, user\_token). Similar to bWAPP’s request, this information is transmitted in plain text which can make it easy for a hacker to steal your information and log into your account. Fig 2.2 is a screenshot of all of the pages found on the site after doing a spider crawl. I specifically highlighted the javascript code that is returned. A hacker with an encrypted view of the code that the front end executes may be able to exploit it to find weaknesses by analyzing the code. Finally, the third screenshot (Fig 2.3) is of a request to get a document stored on the host. This request can be used to expose the storage location of other files where someone may be able to gain access to files that they shouldn’t be able to see.

**Task 3 - WebScarab (NG): Use WebScarab to intercept some pages retrieved from a target website.**

Fig 3.1

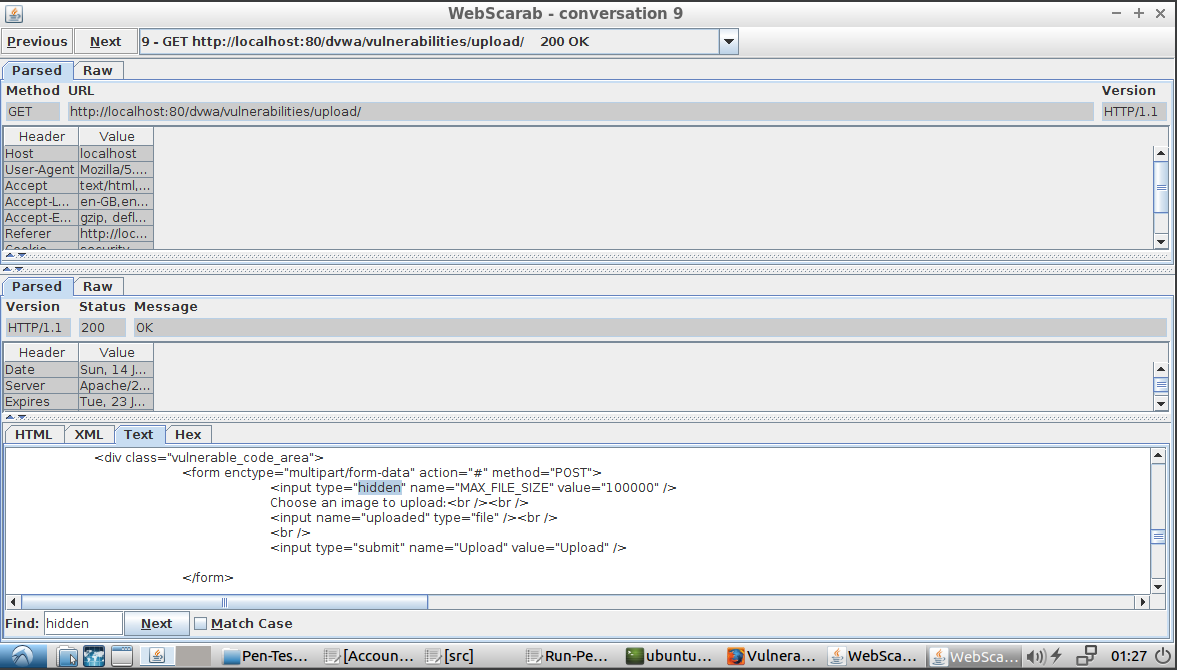


Fig 3.2

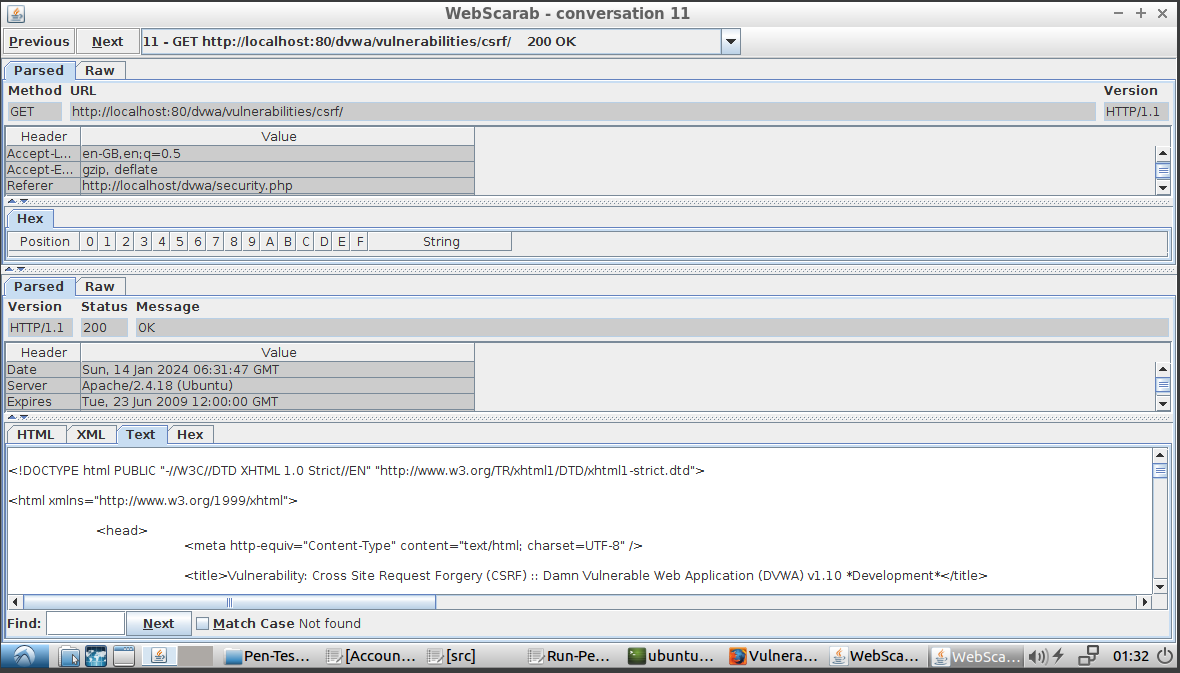
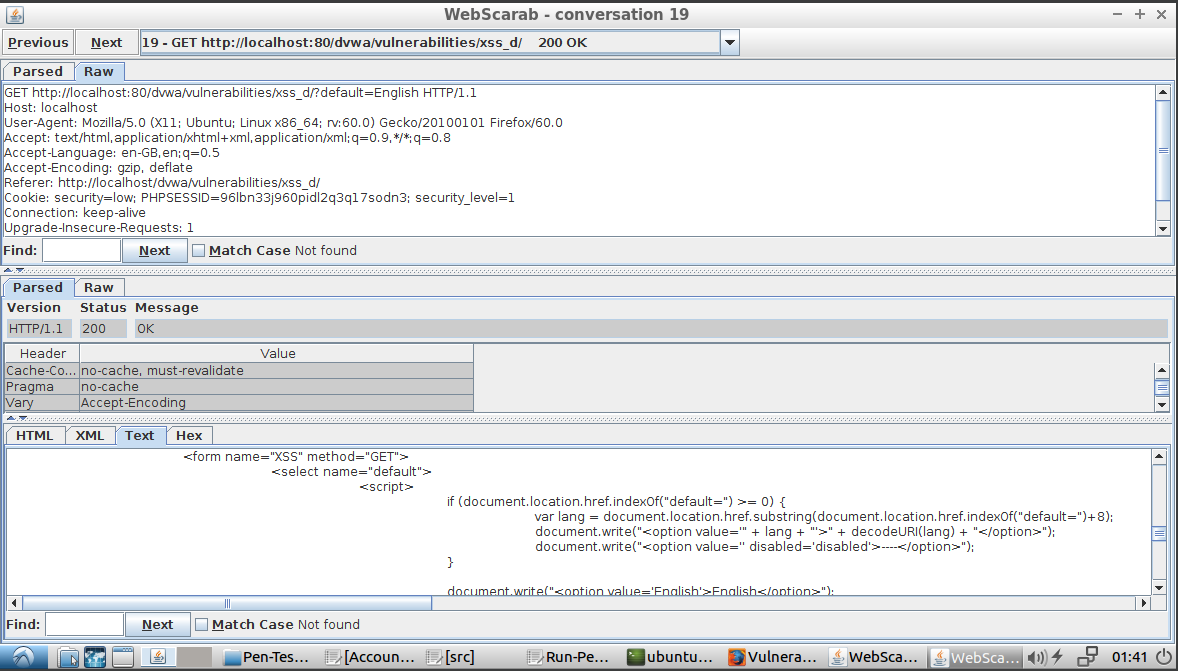


Fig 3.3



The three screenshots above were also captured from DVWA, but this time using WebScarab NG. Figure 3.1 captures the request when loading a file upload page. In this image, you can see that there is a hidden input used to store the max file size allowed to be uploaded. A hacker can modify this hidden input field to increase the maximum allowable file size. Of course, this assumes that the verification is only done client side and there is no server side verification as well. The second figure (Fig. 3.2) is a screenshot of a request that includes the Referrer header, the data in the request isn’t all that relevant in this example. All requests use the Referrer header, but this can be useful to a hacker if the user workflow is dependent on the Referrer address. For example, let’s say you are purchasing an item online. One request has a Referrer address for adding the item to cart, then a second address for checking out, and a third one for confirming payment. Each address is unique to the step and the server trusts that all previous steps were done correctly. But what if there was an address for adding a discount or promo code? It can only be accessed by clicking a link. Well, if a hacker had access to this address, it can manually be inserted as the Referrer address after adding to cart. A discount code could then be applied when it isn’t supposed to be (assumes the hacker has access to a valid code). The last screen, Figure 3.2, is a screenshot of the request when loading a page with a drop down. The response shows that the dropdown is built using a <script> tag. A hacker can use this to execute code which may allow them unauthorized access to the system or system data.