to the attackers website along with their token used for authentication. Account takeover report incoming!

One common problem people run into is not encoding the values correctly, especially if the target only allows for /localRedirects. Your payload would look like something like /redirect?goto=https://zseano.com/, but when using this as it is the ?goto= parameter may get dropped in redirects (*depending on how the web application works and how many redirects occur!*). This also may be the case if it contains multiple parameters (via &) and the redirect parameter may be missed. I will always encode certain values such as & ? # / \ to force the browser to decode it **after** the first redirect.

Location: /redirect%3Fgoto=https://www.zseano.com/%253Fexample=hax

Which then redirects, and the browser kindly then decodes %3F in the BROWSER URL to ?, and our parameters were successfully sent through. We end up with: https://www.example.com/redirect?goto=https://www.zseano.com/%3Fexample=hax, which then when it redirects again will allow the ?example parameter to also be sent. You can read an interesting finding on this further below.

Sometimes you will need to double encode them based on how many redirects are made & parameters.

https://example.com/login?return=https://example.com/?redirect=1%26returnurl=https%3A%2F%2Fwww.google.com%2F

https://example.com/login?return=https%3A%2F%2Fexample.com%2F%3Fredirect=1%2526returnurl%3Dhttps%253A%252F%252Fwww.google.com%252F

When hunting for open url redirects also bear in mind that they can be used for chaining an SSRF vulnerability which is explained more below.

If the redirect you discover is via the “Location:” header then XSS will **not be possible**, however if it redirected via something like “window.location” then you