Iniziamo con la parte web:

Prima di tutto ciò introduciamo uno strumento che ci sarà molto utile.

SULLA MACCHINA VIRTUALE

Burp Suite -> online sul sito di portswigger noi usiamo la versione community, si deve avere il JAR perché è un software java.

Creiamo un progetto temporaneo -> start

Questo tool funge da proxy tra la VM e l'app che stiamo testando, tutte le richieste passano da questo tool.

Funziona un po’ come postman, permette di testare richieste senza utilizzo del browser.

Ci sono un sacco di plugin o cose che ci permettono di rendere ancora più potente lo strumento virtuale.

Nel tab proxy bisogna attivare “intercept on”, ATTENZIONE: burp gira sulla porta 8080, quindi deve essere l’unico processo a girare su quella porta.

Bisogna poi istruire il browser ad inviare le richieste su Burp.

Cerchiamo il plugin foxyproxy e lo aggiungiamo a firefox è un plugin che ci permette di passare al burp proxy o no.

FOXYPROXY:

option -> proxies -> add -> …

Type: HTTP

Hostname: 127.0.0.1

Port: 8080

Supponiamo di voler contattare un’applicazione nostra in locale o un sito esterno come google.com, il sito verrà chiamato nel proxy e poi burp comunicherà col browser.

Che succede quando attiviamo Burp Suite?

Innanzitutto, scaricare il codice della sua app creata per l’occasione su teams nella cartella XSS.

STUDIARE DOCKER.

Quando chiamiamo un server o un applicazione web, nel tab intercept, dove ci sono varie voci.

Ad esempio: user-agent è metterci <script>alert(1);<script> uno dei più famosi payload.

Posso lanciare una ricerca di payload per capire a quanto metterlo, mando la richiesta su repeater ctrl+R o tasto desto “lancia su repeater” e bombardare il sito di richieste.

Tramite burp suite nel repeater, possiamo vedere che tipo di tecnologia stiamo attaccando in repeater nella voce “server”, solitamente le moderne app web usano nel frontend node.js e poi nel backend quello che vogliono.

La tab “decoder”, è utile perché codifica le stringhe in vari modi, come la codifica URL o la codifica in base64.

I query parameter vanno generalmente codificati in formato URL perché ci sono caratteri speciali che possono dare fastidio al resto.

Possiamo anche usare intruder di burp suite al posto di molti altri script per trovare i query param nascosti. Quando cambia il tipo di risposta allora il parametro è giusto.

Gli attacchi alle applicazioni web si dividono in 2 macrocategorie:

1. Client-side: codice eseguito dalla nostra macchina sul server
2. Server-side.

Come funziona il cross sidescript (XSS):

1. Reflected
2. Stored
3. DOM based

CTRL+U -> mostra il codice frontend

Provo a “brutforzare” i query param, vado sul repeater ed eseguo ad esempio una GET  
CTRL + I -> passo ad intruder e gli spiego come andare a cercare i query parameter

Aggiungo all’url nel punto che devo scoprire il ?variabile ad esempio GET/reflected-xss**?Y**=1 http 1.1

Quando becco il query parameter corretto cambierà la risposta:

Evidenzion il query prameter da scoprire e clicco su § (simbolo di paragrafo, in questo caso indica il parametro sul quale fare fuzzing).

Possiamo aggiungere una lista di parametri anche a mano che possiamo anche prendere da siti online in cui ci sono queste wordlist payloadAllTHeThings on gitHub è aggiornato fino all’ultimo minuto e si trovano payload per qualsiasi cosa (ad esempio quella di XSS injection).

Dopo aver definito i payload possiamo cliccare su start.

Quello con lunghezza diversa è quello giusto (IN QUESTO CASO).

In questo caso abbiamo scoperto che il queryparam era name è posso porlo uguale a <script>alert(‘hello!!!!’);</script>, o ad altre cose, posso poi codificare questo script in URL sempre con Burp -> seleziona tasto dx URL encoder.

Lo strumento repeater vale più per i casi server side, quindi non vale nei reflected.

Tramite name =<script>alert(1);<script> se l’utilizzatore lo esegue potrei andarmi a prendere un cookie di sessione e fare danni.

Studiare la differenza tra stored e reflected.

Mentre nel reflected per generare l’XSS devo mandare un link che deve essere cliccato dall’utilizzatore con lo stored questo viene memorizzato nel DB.

Quidni se mettiamo il solito <script>alert(1);</script> anche se esco dall’app e ci rientro appare sempre e si resetta come nel reflected.

DOM based:

A differenza degli altri questo non coinvolge il server, in questo caso andiamo ad iniettare codice javascript sul browser e questo payload diventa un componente dell’applicazione, stiamo infilando un’automazione che viene lanciata a rtuntime nell’applicazione web.

In questo caso stiamo interaendo con il Document Object Model.

Nel caso in questione la vulnerabilità sta nel fatto che il DOM viene modificato tramite il campo modificato dall’utente.

Nel DOM based posso inserire codice javascript e non gli <script>, cerchiamo ad esempio su gitHub, esmepio:

<img src=x onerror=’alert(2);’</img>

Sto dando un percorso errato all’immagine di proposito, così darà errore e caricherà l’alert.

TUTTO BELLO E TUTTI FIGO MA QUESTI ATTACCHI FUNZIONANO SOLO SULLE APPLICAZIONI GIOCATTOLO, nel caso di quelle reali bisogna trovare il modo di bypassare i criteri di sicurezza.

Guardare la bacheca teams: esempi e siti.

A caso:

Se non ho abbastanza spazio per il mio payload posso istruire l’app a contattare un sito esterno che lo permetta.

Aprire un server php:

$ php -s 0.0.0.0 :8000

Per esercitarci leggere dal sito W/labs e burp swing .