EXTRA

1. Mettere in comunicazione le due macchine e controllare che comunichino

Per prima cosa si impostano le macchine in modo che siano nella stessa rete. Dopo di che ci si accerta che siano in comunicazione con il comando “ping (indirizzo IP dell’altra macchina)”

IP Kali:192.168.104.100

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

IP Metasploitable: 192.168.104.150

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

1. Andare sulla web application DVWA da Kali

Bene, ora raggiungiamo la web machine DVWA scrivendo l’indirizzo IP della Metasploitable nella barra di ricerca di Firefox, dopo di che clicchiamo su “DVWA” per accedervi.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Adesso che siamo all’interno della DVWA andiamo a settare la difficoltà a “medium” cliccando su DVWA security, apriamo il menù a tendina e selezioniamo “medium” e diamo conferma con “Submit”.Immagine che contiene testo, elettronica, schermata, Pagina Web

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Sito Web, Pagina Web

Descrizione generata automaticamente

Ora ci spostiamo in “XXS stored” e procediamo con un primo test, lo scriveremo in JavaScript e vediamo cosa succede.

Immagine che contiene testo, Carattere, software, schermata

Descrizione generata automaticamente

Provando con l’approccio utilizzato in precedenza con sicurezza “low” non ci permette di ottenere i cookie come possiamo notare dall’immagine successiva. Notiamo che ha comunque accettato il codice, ma il livello di sicurezza ci impedisce di ottenere i cookie.

Immagine che contiene testo, schermata, software, Pagina Web

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, Pagina Web

Descrizione generata automaticamenteArrivati a questo punto esaminiamo il codice PHP per vedere se abbiamo parte di codice che blocca il linguaggio JavaScript. Per fare questo clicchiamo su “View Source”.

Verrà aperta questa pagina dove viene visualizzato il codice PHP, questo codice è progettato per gestire i dati di un modulo inviato tramite una richiesta POST. Questo script viene utilizzato in un'applicazione guestbook, dove gli utenti possono inviare un messaggio e il loro nome.

Il codice PHP si divide in:

* Verifica della richiesta POST
* Acquisizione e sanitizzazione dei dati (per prevenzione per SQL injection e attacchi XSS)
* Query SQL per inserire i dati nel database

Quindi il codice esegue questi passaggi per elaborare e archiviare i dati del modulo del guestbook in modo sicuro, utilizzando varie tecniche di sanitizzazione per proteggere l'applicazione da potenziali attacchi.

Notiamo anche che la parte di codice che si occupa del messaggio è stata rinforzata, ma notiamo che la parte dedicata al nome è ancora vulnerabile.

Allora con tasto destro del mouse andiamo a fare ispeziona sulla zona del nome per scoprire che i limiti di immissioni caratteri e il loro spazio visibile possono essere aumentati.

1. Preparativi per l’attacco

Scriviamo il payload d’attacco in un file di testo che andremo a posizionare nella cartella dove vogliamo che i dati vengano raccolti. Avviamo una shell di comando in questa cartella, questo passaggio serve anche per far funzionare correttamente il payload.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente**Payload XSS**

In questo documento è presente sia il payload per l’attacco XSS sia un comando di decriptazione che ci sarà utile più avanti per decriptare i dati in arrivo.

Questo payload tenta di iniettare uno script esterno (a.js) da un server remoto (http://192.168.104.100:4444) nella pagina web vulnerabile. Ora andremo a vedere lo script appena citato nel dettaglio.

Immagine che contiene testo, software, Software multimediale, Software per la grafica

Descrizione generata automaticamente

Questo codice in JavaScript raccoglie le informazioni target dal browser della vittima e le invia a un server remoto tramite un'immagine generata dinamicamente (tecnica spesso usata in attacchi XSS per l'esfiltrazione dei dati).

Cominciamo con la creazione dell’oggetto **info**

* **“cookies**: **document.cookie”**: Recupera i cookie del sito corrente. I cookie possono contenere informazioni sensibili, come token di sessione o preferenze dell'utente.
* “**userAgent**: **navigator.userAgent**”: Ottiene l’User-Agent del browser, che identifica il tipo di browser, sistema operativo e altre caratteristiche utili per il fingerprinting.
* “**ip: window.location.host**”: Recupera l'host della pagina corrente, che corrisponde al dominio o all'indirizzo IP del sito visitato.
* **“date: new Date().toISOString()**”: Inserisce la data e l'ora correnti in formato ISO 8601, utile per sapere quando l'informazione è stata raccolta.

Fatto ciò, pensiamo alla codifica e l’invio dei dati

* **new Image().src**: Crea un nuovo oggetto Image e imposta l'attributo **src** con un URL che contiene i dati codificati. Questo invia una richiesta GET al server remoto 192.168.104.100 sulla porta 4444, con i dati allegati come parametro della query string (?data=)
* **JSON.stringify(info)**: Converte l'oggetto info in una stringa JSON
* **encodeURIComponent**: Codifica la stringa JSON per garantire che possa essere inviata come parte di un URL senza problemi.

Questo è il server, ora lo andremo ad analizzare.

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

Cominciamo dall’importazione dei moduli

* **http.server**: Fornisce classi e funzioni per creare server HTTP semplici
* **os**: Modulo per interagire con il sistema operativo

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

Passiamo ora al **MyRequestHandler**:

* **BaseHTTPRequestHandler** è una classe di base per gestire le richieste http, la classe personalizzata **MyRequestHandler** si basa su di essa

Di seguito abbiamo la gestione delle richieste dove viene usato “**def do\_GET(self)**”

* Questo metodo viene chiamato ogni volta che il server riceve una richiesta HTTP di tipo GET

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

La prossima parte si occupa della richiesta ricevuta tramite “**print**”

* Stampa l'URL richiesto dal client, memorizzato in **self.path**

Ora viene l’invio della risposta per **/a.js**

* Restituisce un codice di stato **200 OK**
* Imposta l'intestazione HTTP **Content-type** come application/javascript per indicare che il file restituito è uno script JavaScript
* Legge il file a.js in modalità binaria (rb) e lo invia come risposta

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

La parte finale del codice tratta della configurazione e dell’avvio del server

* Configura il server per ascoltare su tutte le interfacce di rete (**""**) alla porta **4444**
* Utilizza la classe **MyRequestHandler** per gestire le richieste
* Stampa un messaggio per indicare che il server è attivo
* Chiama il metodo **serve\_forever()** per mantenere il server in ascolto fino a quando non viene terminato manualmente

In sostanza quando viene lanciato lo script il server si mette in ascolto sulla porta 4444. Se un client invia una richiesta a **http:/server\_ip>:4444/a.js**, il server restituisce il contenuto del file **a.js.** Per qualsiasi altra richiesta (es. http://<server\_ip>:4444/qualcosa), il server risponde con il messaggio Richiesta ricevuta correttamente!

All’attacco

Usando tasto destro sulla zona di input name e selezionando ispeziona andremo a sfruttare la vulnerabilità trovata in precedenza.

Per sicurezza aumentiamo la lunghezza a 100 e il limite di caratteri a 200Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Arrivati a questo punto proviamo a inserire lo script per ottenere il dump completo. Ma prima ricordiamoci di utilizzare la shell aperta dalla cartella dove si trovano tutti i file come detto in precedenza.

Ora siamo veramente pronti per tentare l’attacco. Attiviamo il nostro server e se non abbiamo errori potremo vedere che siamo in ascolto sulla porta 4444.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

prima di procedere assicuriamoci che la difficoltà sia impostata su “medium” e siamo pronti per immettere il payload nella zona vulnerabile inseriamo il messaggio e clicchiamo su Sign Guestbook.

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

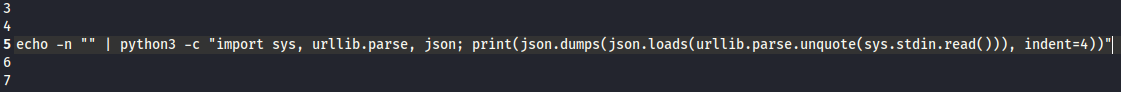
Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamenteDopo aver dato il comando se andrà tutto bene il nostro messaggio dovrebbe apparire il che significa che siamo riusciti ad impiantare un XSS stored con successo, ora vediamo se il nostro server ha captato “l’immagine”.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Come si può vedere il server ha fatto il suo lavoro ottenendo le info criptate della vittima. Ora non ci resta che decriptarle col comando che abbiamo lasciato in sospeso nel file di testo.



Vediamo in dettaglio cosa fa il comando, per fare ciò sezioneremo il comando in diverse parti:

* echo: Comando che stampa una stringa in uscita. In questo caso, l'output è una stringa vuota ("").
* -n: Opzione che sopprime il carattere di nuova linea (\n) alla fine dell'output.

 **|** (Pipe): Passa l'output di **echo** -n come input (**stdin**) al comando successivo (**python3 -c**).

 **python3 -c "..."**: Permette di eseguire uno script Python direttamente dalla riga di comando. Il codice Python è specificato come una stringa tra virgolette.

Ora prenderemo in esame la sezione del codice Python. Come prima cosa analizziamo l’importazione dei moduli.

import:

* **sys**: Utilizzato per accedere allo stdin (standard input) e leggere i dati in ingresso.
* **urllib.parse**: Fornisce la funzione unquote, che decodifica stringhe URL-encoded.
* **json**: Usato per manipolare stringhe JSON (JavaScript Object Notation).

Ora abbiamo la disamina dell'input

sys.stdin.read():

* Legge tutti i dati ricevuti tramite **stdin** (in questo caso, ciò che viene passato da **echo**).
* Se la stringa data=%x%1x%2x%3x%x2value%x%7x viene passata, sys.stdin.read() restituirà:

data=%x%1x%2x%3x%x2value%x%7

Passiamo alla decodifica URL:

* **unquote** converte i caratteri URL-encoded nella loro rappresentazione leggibile.

Es:

* Input: data=%x%1x%2x%3x%x2value%x%7
* Output: data={“key”:”value”}

Ad esempio:

Input: data=%7B%22key%22%3A%22value%22%7D

Output: data={"key":"value"}

Parliamo della parte finale del codice dove è presente Parsing JSON

* **json.loads** converte una stringa JSON in un oggetto Python (come un dizionario).
* In questo caso, estrae l'oggetto JSON dalla stringa.
* Input: **{"key":"value"}**
* Output: **{'key': 'value'}** (linguaggio Python)

Ed infine vediamo la formattazione tramite JSON

* **json.dumps** converte un oggetto Python (come un dizionario) in una stringa JSON formattata.
* **indent=4** aggiunge un'indentazione di 4 spazi per rendere il JSON leggibile.

Ecco i dati criptati estrapolati dalla risposta del server.

*%7B%22cookies%22%3A%22security%3Dmedium%3B%20PHPSESSID%3Da5c7498eeb839faf392dcc7b6b37229c%22%2C%22userAgent%22%3A%22Mozilla%2F5.0%20(X11%3B%20Linux%20x86\_64%3B%20rv%3A128.0)%20Gecko%2F20100101%20Firefox%2F128.0%22%2C%22ip%22%3A%22192.168.104.150%22%2C%22date%22%3A%222025-01-10T03%3A15%3A19.009Z%22%7D*

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteOra immettiamo i dati criptati all’interno del comando “echo”

Come ultimo passaggio apriamo una nuova shell dove copiamo prima il comando di decriptazione e all’interno di esso tra la doppia coppia di virgolette va inserita o copiata la stringa di file criptati. Dando l’invio i dati in chiaro appariranno a schermo.

E con questo ultimo passaggio abbiamo ottenuto le informazioni che volevamo grazie ad un attacco XSS stored. Un attacco di questo tipo è molto pericoloso perché permette agli attaccanti di inserire codice malevolo nei siti web che vengono poi visualizzati da altri utenti. Quando un utente visita una pagina compromessa, il codice dannoso viene eseguito nel suo browser, permettendo agli attaccanti di rubare informazioni sensibili come cookie, sessioni di accesso e persino di reindirizzare gli utenti verso siti dannosi.

In sintesi, l’XSS Stored è particolarmente insidioso perché il payload dannoso rimane memorizzato sul server e continua a colpire chiunque visiti la pagina compromessa.