

## INDICE

INDICE.....	1
INTRODUZIONE ED OBIETTIVO.....	2
METODOLOGIA OPERATIVA.....	3-8
CONCLUSIONE.....	9

## INTRODUZIONE ED OBIETTIVO

**INTRODUZIONE:** questo laboratorio prende in esame il comportamento di una applicazione web che consente agli utenti di caricare file sul server.

Attraverso l'analisi e la manipolazione delle richieste inviate dal browser viene osservato come un semplice meccanismo di upload, se privo di controlli adeguati, possa essere utilizzato per trasferire e rendere eseguibile un contenuto non previsto dall'applicazione. L'attività consente di seguire l'intero percorso del dato, dalla selezione del file sul client fino alla sua memorizzazione ed esecuzione sul sistema remoto.

**OBIETTIVO:** comprendere come una funzione di caricamento file possa essere utilizzata per far eseguire al server contenuti non previsti dall'applicazione.

Attraverso questa attività si osserva il passaggio da un semplice upload di file alla possibilità di interagire direttamente con il sistema che ospita l'applicazione.

## METODOLOGIA OPERATIVA

The screenshot shows two terminal windows. The left window is on Kali Linux with the command \$ ping -c 4 192.168.50.101. The right window is on Metasploitable with the command msfadmin@metasploitable:~\$ ping -c 4 192.168.50.100. Both show successful ping results with low latency.

```
$ ping -c 4 192.168.50.101
PING 192.168.50.101 (192.168.50.101) 56(84) bytes of data.
64 bytes from 192.168.50.101: icmp_seq=1 ttl=64 time=1.34 ms
64 bytes from 192.168.50.101: icmp_seq=2 ttl=64 time=2.42 ms
64 bytes from 192.168.50.101: icmp_seq=3 ttl=64 time=1.48 ms
64 bytes from 192.168.50.101: icmp_seq=4 ttl=64 time=2.95 ms
--- 192.168.50.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.339/2.045/2.949/0.667 ms

msfadmin@metasploitable:~$ ping -c 4 192.168.50.100
PING 192.168.50.100 (192.168.50.100) 56(84) bytes of data.
64 bytes from 192.168.50.100: icmp_seq=1 ttl=64 time=0.97 ms
64 bytes from 192.168.50.100: icmp_seq=2 ttl=64 time=1.99 ms
64 bytes from 192.168.50.100: icmp_seq=3 ttl=64 time=0.000 ms
64 bytes from 192.168.50.100: icmp_seq=4 ttl=64 time=0.000 ms
--- 192.168.50.100 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.000/2.741/8.971/3.688 ms
```

Nel primo passaggio dell'esercizio è stata verificata la raggiungibilità reciproca tra le due macchine coinvolte, quella che ospita l'applicazione web e quella da cui viene effettuata l'analisi.

Attraverso uno scambio di pacchetti di rete è stato confermato che i due sistemi si trovano sulla stessa rete logica e sono in grado di comunicare senza intermediari o blocchi di connettività; questo controllo iniziale è fondamentale perché l'intera attività si basa sulla possibilità di instaurare un dialogo diretto tra il browser che accede all'applicazione e il server che la ospita: senza questa connettività, qualsiasi richiesta web, caricamento di file o interazione applicativa non potrebbe avvenire.

The screenshot shows the DVWA login page with 'admin' entered in the Username field and '\*\*\*\*\*' in the Password field. Below it, the Burp Suite interface shows a list of network requests. Request 7 is a GET to the root URL. Requests 9 and 10 are GETs to /dvwa/. Request 11 is a POST to /dvwa/login.php with parameters 'username=admin' and 'password=123'. Request 12 is a POST to /dvwa/login.php with parameters 'username=admin' and 'password=123'. The Burp Suite interface also shows the response for request 11, which is a 200 OK status with the title 'Metasploitable2 - Linux'.

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start respond.
7	http://192.168.50.101	GET	/			200	1124	HTML		Metasploitable2 - Linux			192.168.50.101		12:48:06 9 J... 8080	24	
9	http://192.168.50.101	GET	/dvwa/			302	483	HTML					192.168.50.101	PHPSESSID=cb1c...	12:48:20 9 J... 8080	22	
10	http://192.168.50.101	GET	/dvwa/login.php			200	1636	HTML	php	Damn Vulnerable Web...			192.168.50.101		12:48:20 9 J... 8080	25	
11	http://192.168.50.101	POST	/dvwa/login.php	username=admin	✓			HTML	php				192.168.50.101		12:48:32 9 J... 8080		
12	http://192.168.50.101	POST	/dvwa/login.php	password=123	✓			HTML	php				192.168.50.101		12:48:47 9 J... 8080		

Durante la fase di autenticazione all'applicazione web, l'accesso dell'utente non avviene come un semplice evento locale del browser, ma come una transazione di rete completa tra il sistema dell'utente e il server che ospita l'applicazione.

In questo passaggio è stata quindi effettuata una connessione al servizio web e l'invio delle credenziali di accesso sotto forma di una richiesta strutturata verso il server.

L'elemento chiave che emerge è che le credenziali e i dati di sessione non restano confinati nell'interfaccia grafica, ma vengono trasmessi come parametri tecnici all'interno di una richiesta http; tali parametri includono il nome utente, la password e gli identificativi di sessione che il server utilizza per stabilire l'identità dell'utente autenticato.

La schermata mostra che queste informazioni risultano integralmente osservabili, registrabili e riproducibili ciò significa che l'applicazione si basa su dati trasmessi in chiaro all'interno del flusso di comunicazione per determinare se un utente è autorizzato o meno.

Dal punto di vista del controllo dei rischi, questo implica che chiunque sia in grado di intercettare o replicare tali richieste può comprendere esattamente come viene costruita una sessione valida e, potenzialmente, riutilizzarla.

Questo passaggio dimostra quindi che l'autenticazione non è protetta da un meccanismo opaco o impenetrabile, ma è il risultato di una sequenza di scambi tecnici che possono essere analizzati e, se necessario, manipolati. È su questa esposizione del processo di login che si basa l'intera catena di eventi successiva dell'esercizio.

The screenshot shows the DVWA (Damn Vulnerable Web Application) security configuration page and the Burp Suite proxy tool interface. The DVWA page displays a sidebar with links like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, and SQL Injection (Blind). The main content area shows 'Script Security' settings with a dropdown menu set to 'Low'. Below it, a note states: 'Security Level is currently high. You can set the security level to low, medium or high. The security level changes the vulnerability level of DVWA.' A 'Submit' button is present. The Burp Suite interface at the bottom shows a timeline of a single request: a POST to http://192.168.50.101/dvwa/security.php. The request details show 'HTTP history' selected under the 'Proxy' tab, and the raw request body contains the value 'low' for the security level parameter.

Dopo l'accesso all'applicazione, il livello di protezione viene impostato su Low tramite l'interfaccia di configurazione.

Questa operazione genera una richiesta *HTTP POST* verso il server, visibile e intercettabile tramite Burp Suite, la richiesta POST trasporta al backend il parametro che definisce il livello di sicurezza dell'applicazione, che in questo caso viene esplicitamente abbassato al valore minimo.

L'osservazione è fondamentale perché dimostra che una configurazione critica dell'applicazione non è vincolata a un controllo interno del server, ma viene determinata da un dato fornito dal client.

In pratica, il sistema accetta che il browser (e quindi chiunque sia in grado di costruire una richiesta HTTP valida) decida il livello di protezione attivo.

Dal punto di vista operativo questo rappresenta un punto di svolta nell'esercizio: impostando *Low* tramite una richiesta *POST* intercettabile, l'applicazione entra in una modalità estremamente permissiva, nella quale i controlli sui contenuti caricati, sui parametri ricevuti e sulle operazioni consentite vengono drasticamente ridotti.

The screenshot shows the DVWA application interface and the Burp Suite proxy tool. The DVWA sidebar menu includes: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), **Upload**, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The current page is 'Vulnerability: File Upload'. The Burp Suite interface shows the 'Proxy' tab selected. A network request is listed: Time 13:05:10 9 Jan ... Type HTTP → Request Method POST URL http://192.168.50.101/dvwa/vulnerabilities/upload/. The DVWA page displays a file upload form with a 'Choose File' button highlighted in green, and a success message below stating '.../.../hackable/uploads/... successfully uploaded!'

Dopo aver portato il livello di sicurezza dell'applicazione sul valore più permissivo, la piattaforma ha accettato senza restrizioni aggiuntive le richieste in ingresso, rendendo possibile osservare e controllare in modo completo il flusso dei dati che dal browser venivano inviati al server.

In questo contesto, l'azione di caricamento di un file è stata trattata come una vera e propria transazione applicativa che il server riceve, interpreta ed esegue; l'intermediazione del proxy ha reso visibile questa transazione nella sua forma reale: una richiesta strutturata che contiene sia le informazioni del file scelto dall'utente sia i metadati che ne descrivono il nome, il tipo e la destinazione.

Nel momento in cui l’utente ha avviato il caricamento, il server ha ricevuto una richiesta di tipo “invio dati” indirizzata all’endpoint che gestisce l’upload.

Questa richiesta non trasportava soltanto un contenuto generico, ma includeva un campo che dichiarava esplicitamente il nome del file e un altro che ne indicava il tipo apparente; è proprio questo passaggio che evidenzia il punto critico: il sistema si è limitato a fidarsi di ciò che il client dichiarava, senza effettuare verifiche sostanziali sulla natura reale del contenuto , di conseguenza, un file che non aveva nulla a che fare con un’immagine è stato comunque trattato come se fosse legittimo.

La conferma visiva di questo comportamento si vede immediatamente nel messaggio di esito del caricamento, che indica chiaramente il percorso in cui il file è stato memorizzato all’interno dell’area applicativa, sotto la directory “*hackable/uploads*”.

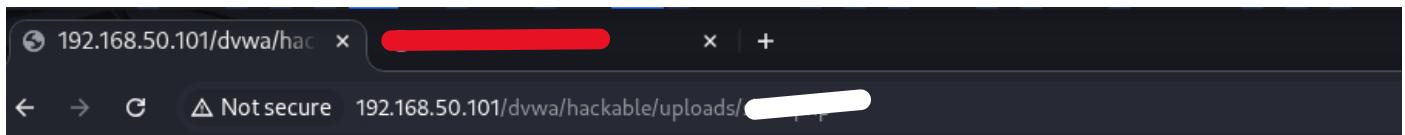
In pratica il server ha scritto sul proprio file system un contenuto controllato dall’utente, rendendolo accessibile tramite un normale indirizzo web; questo passaggio è fondamentale, perché segna il momento in cui un input esterno smette di essere un semplice dato temporaneo e diventa un file residente sul sistema.

L’unione delle due viste (da un lato la richiesta intercettata, dall’altro la conferma dell’avvenuto caricamento) dimostra che l’applicazione non applica alcun controllo efficace né sul nome del file né sulla sua reale funzione.

Il risultato è che il server, invece di limitarsi a conservare un’immagine, si ritrova a ospitare un file che può essere richiamato direttamente come una risorsa attiva.

In termini pratici, questo significa che ciò che è stato caricato non è più solo un oggetto passivo, ma qualcosa che il server può interpretare ed eseguire quando viene richiesto tramite il browser.

È esattamente questo il punto centrale dell’esercizio: mostrare come una semplice funzione di caricamento, se priva di controlli adeguati, diventi un canale diretto per introdurre codice eseguibile all’interno del sistema, trasformando un normale modulo di upload in una porta d’ingresso verso il cuore dell’applicazione.

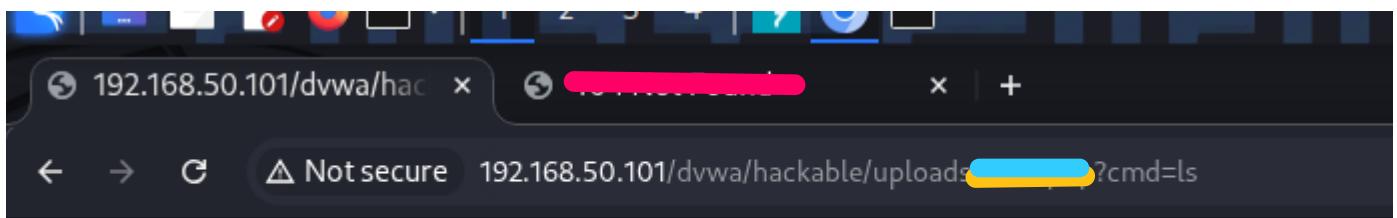


Warning: system() [[function.system](#)]: Cannot execute a blank command in **/var/www/dvwa/hackable/uploads/** on line 1

Il file caricato viene effettivamente eseguito dal sistema che ospita l'applicazione, non semplicemente archiviato.

Il messaggio restituito indica che il codice presente nel file è stato interpretato dal server e che il sistema ha tentato di avviarlo, ma non ha ricevuto alcuna istruzione operativa; il percorso riportato conferma che il file si trova nella directory degli upload dell'applicazione, una posizione che dovrebbe essere solo di archiviazione ma che in questo caso è trattata come area eseguibile.

Questo significa che l'upload non è confinato in uno spazio sicuro ma entra direttamente nel perimetro di esecuzione del server, trasformando un normale meccanismo di caricamento in un punto di ingresso operativo: ciò che è stato caricato non è più un contenuto passivo ma una componente attiva del sistema, capace di reagire a input esterni e di eseguire istruzioni sul server stesso.



dvwa\_email.png dvwa\_email.txt

Questo passaggio rappresenta il punto in cui l'attacco diventa **operativo** e non più solo dimostrativo.

Il file caricato non si limita a esistere sul server, ma viene utilizzato come interfaccia di controllo; inserendo un parametro nella richiesta web, il server esegue realmente l'istruzione richiesta e restituisce il risultato direttamente nella pagina.

L'elenco dei file visualizzato dimostra che il codice sta interrogando il file system della macchina che ospita l'applicazione e che ha accesso alla directory in cui sono stati memorizzati gli upload, confermando che il controllo non è teorico ma concreto.

In pratica, il caricamento del file ha creato un punto di accesso che permette di osservare e manipolare l'ambiente del server dall'esterno, trasformando una funzione apparentemente innocua in una vera porta di ingresso verso il sistema.

Pretty Raw Hex

```
1 GET /dvwa/hackable/uploads/?cmd=ls HTTP/1.1
2 Host: 192.168.50.101
3 Accept-Language: it
4 Upgrade-Insecure-Requests: 1
```

Questo elemento mostra in modo esplicito come l'interazione con il file caricato avvenga tramite una normale richiesta web in cui, oltre al percorso del file, viene passato un parametro che contiene l'azione da eseguire sul server.

In pratica la pagina non viene più usata come una semplice risorsa statica, ma come un punto di ingresso che riceve istruzioni dall'esterno, le interpreta e le inoltra al sistema operativo della macchina che ospita l'applicazione; il fatto che il parametro sia accettato e processato senza alcun controllo dimostra che l'upload ha trasformato il server in un componente attivo dell'attacco, capace di eseguire comandi e restituirne l'output, rendendo evidente come una funzione pensata per caricare file possa essere sfruttata per ottenere un controllo diretto sull'ambiente che esegue l'applicazione.

## CONCLUSIONE

Nel complesso l'esercizio ha mostrato come una funzionalità apparentemente banale, come il caricamento di un file, possa trasformarsi in un punto di accesso completo all'infrastruttura che ospita l'applicazione quando non è progettata e governata con criteri rigorosi.

È emerso con chiarezza che non è necessario aggirare sistemi complessi o sfruttare tecniche esoteriche: basta poter inserire contenuto arbitrario in un flusso previsto dal sistema perché questo venga poi eseguito con i privilegi del servizio che lo gestisce, rendendo possibile leggere dati, avviare processi e interagire con l'ambiente sottostante.

In termini concreti ciò significa che una singola debolezza logica può annullare ogni altra barriera, trasformando un semplice servizio web in un canale di controllo remoto, con un impatto potenzialmente totale sull'affidabilità, sulla riservatezza e sulla continuità operativa dell'intero sistema.