Report Completo: Vulnerabilità PHP - POST / password

Analisi del comportamento di $_POST['password'], exploit tramite array e contromisure

Analisi generata per Giovanni Oliverio — ChatGPT (GPT-5 Thinking mini) — 2025-10-12

1) Introduzione

In questo documento analizziamo il comportamento di $_POST['password'] in PHP e come la ricezione di array
(es. password[]=...) possa portare a bypass di autenticazione quando il codice non verifica il tipo dell'input e utilizza confronti deboli. Vengono descritti gli esempi pratici, differenze tra versioni PHP e le patch consigliate.

2) Comportamento di $_POST['password']

Quando il client invia dati via POST, PHP popola $_POST con i valori ricevuti.

- Se invii password=abc → $_POST['password'] è la stringa "abc". - Se invii password[]=abc → $_POST['password'] è un array: array(0 => "abc").

Nota: la differenza tra il nome del campo nel form HTML e il valore è cruciale. Scrivere 'password[]=abc' come
testo in un campo non crea un array; l'array nasce dal nome del campo nell'HTML o dalla costruzione della payload HTTP.

3) strcmp e comportamento indefinito

La funzione strcmp($a, $b) si aspetta stringhe. Se le vengono passati tipi diversi (es. un array), si hanno comportamenti diversi a seconda della versione di PHP:

- PHP 7.x: - strcmp emette un warning e restituisce un valore 'falsy' (false o NULL). Se il codice confronta il risultato con '== 0' (confronto debole), false == 0 risulta vero → possibile bypass. - PHP 8.x: - strcmp su tipi non compatibili lancia TypeError, che è più sicuro, ma il codice dovrebbe comunque validare i tipi prima di chiamare funzioni stringa.

Questo è il vettore d'attacco principale: chiamare strcmp senza validare il tipo di $_POST['password'].

4) Esempi pratici con curl e debug

Esempi di richieste e output per debug:

Singolo valore (stringa): curl -s -X POST -d 'password=abc' http://tuo-sito/
var_dump($_POST['password']); //
string(3) "abc"

Array: curl -s -X POST -d 'password[]=abc' http://tuo-sito/ var_dump($_POST['password']); // array(1) {
[0]=>
string(3) "abc" }

Test strcmp su array: var_dump(strcmp($_POST['password'], $password)); // warning + ritorno false → confronto
debole può diventare true

5) Differenze tra versioni PHP

Riassunto rapido: - PHP 7.x: comportamento permissivo → warning + ritorno falsy → rischio di 'type juggling'
se si usa confronto debole. - PHP 8.x: TypeError su mismatch di tipo → più sicuro per questo caso ma non sostituisce la validazione dell'input.

Raccomandazione: non fare affidamento sulla versione per la sicurezza; validare sempre gli input.

6) Trucco principale (vettore di bypass)

Il bypass è tipicamente costruito su questa linea:

```
if (strcmp($_POST['password'], $password) == 0) { /* grant access */ }
```

Se $_POST['password'] è un array, strcmp restituisce false; con '== 0' si ha false == 0 ⇒ vero, quindi il ramo
di autenticazione viene eseguito e l'attaccante può visualizzare il FLAG o ottenere accesso.

7) Patch consigliata (snippet sicuro)

Esempio di patch consistente e robusta:

```
include_once('./secrets.php');

if (!isset($_POST['password']) || !is_string($_POST['password'])) {
    error_log('Invalid password type: ' . gettype($_POST['password']) . ' from ' . $_SERVER[
    echo 'Wrong Password';
    exit;
}

if (hash_equals($password, $_POST['password'])) {
    echo $FLAG;
} else {
    echo 'Wrong Password';
}
```

Spiegazioni: - is_string() evita array e altri tipi. - hash_equals() evita attacchi timing. - usare === quando si confrontano tipi conosciuti (o hash_equals per stringhe).

8) Best practices

- Disabilitare display_errors in produzione (display_errors = Off) e loggare errori in modo sicuro. - Non esporre sorgenti di codice in produzione (es. /?source). - Usare password_hash() e password_verify() per archiviare password reali. - Validare input con filter_input()/filter_var(). - Gestire array intenzionali esplicitamente quando previsto:

```
if (is_array($_POST['password'])) {
    foreach ($_POST['password'] as $p) {
        if (!is_string($p)) continue;
        // validazione di ogni $p
    }
}
```

9) Test rapidi suggeriti

Esempi di test da eseguire su un'istanza di staging:

POST singolo valore: curl -s -X POST -d 'password=50' http://tuo-sito/

POST array: curl -s -X POST -d 'password[]=50' http://tuo-sito/

Debug server rapido: var_dump($_POST['password']); die();

Questi test mostrano chiaramente la differenza tra stringa e array.

10) Checklist di mitigazione (rapida)

- [ ] Validare sempre il tipo di input prima di usare funzioni stringa. - [ ] Evitare confronti deboli '==' per risultati di funzioni che possono restituire false/null. - [ ] Usare hash_equals() per confronti di stringhe sensibili. - [ ] Abilitare logging sicuro di input non validi. - [ ] Testare in ambiente di staging su PHP 7.x e PHP 8.x. - [ ] Usare password_hash/password_verify per password persistenti.

11) FAQ / Note operative

Q: Perché false == 0 è true in PHP? A: In PHP i confronti deboli fanno coercizione dei tipi; false viene coerciuto a 0 per il confronto con numeri/stringhe, quindi false == 0 risulta true.

Q: Questo attacco funziona sempre? A: Non sempre. Funziona quando il codice non valida i tipi, usa strcmp o
funzioni simili e confronta il risultato con '== 0' (confronto debole). PHP 8 rende più difficile l'attacco grazie a TypeError ma non evita la cattiva pratica di non validare input.

12) Risorse e buone letture (suggerite)

- Manuale PHP: gestione array e variabili superglobali. - OWASP: Input Validation Cheat Sheet. - Documentazione PHP su hash_equals, password_hash, password_verify.

13) Conclusione

Il problema nasce dal type juggling di PHP: passare array dove si aspetta stringa, usare strcmp senza controllo tipo, e confronto debole '== 0'. La soluzione è semplice e robusta: validare il tipo dell'input e usare confronti sicuri (=== o hash_equals), oltre a seguire le best practices per la gestione delle password.