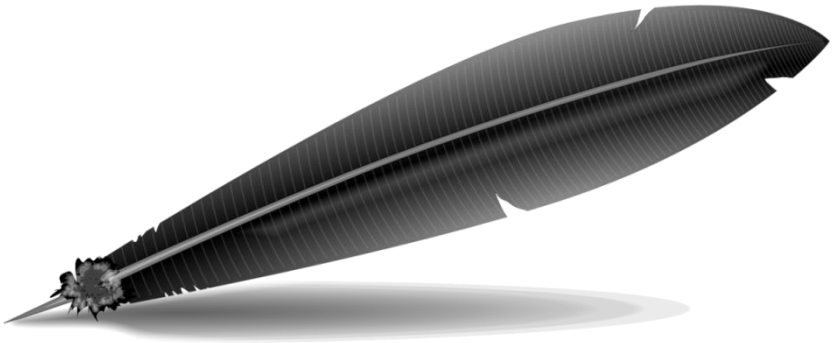
La sicurezza nella progettazione del software

a developer approach

ing. Stefano Turchi, Ph.D. stefano.turchi@unifi.it

Una vigile e provvida paura è la madre della sicurezza.

Edmund Burke



Cos'è la sicurezza?

Secondo il NIST (National Institute of Standards and Technology)[1]

- Integrità
- Confidenzialità
- Disponibilità

Integrità

*«…means **guarding** against improper information **modification** or **destruction**, and includes ensuring information non-repudiation and authenticity»*

Significa quindi *impedire* eventuali contraffazioni?

Funzioni Hash

Una funzione Hash è una funzione *deterministica* che mappa uno o più elementi del dominio in almeno un elemento del codominio

$H: X \to Y \; dove \; |X| \ll |Y|$ pertanto $\exists x_i, x_j \in X \mid H(x_i) = H(x_j)$

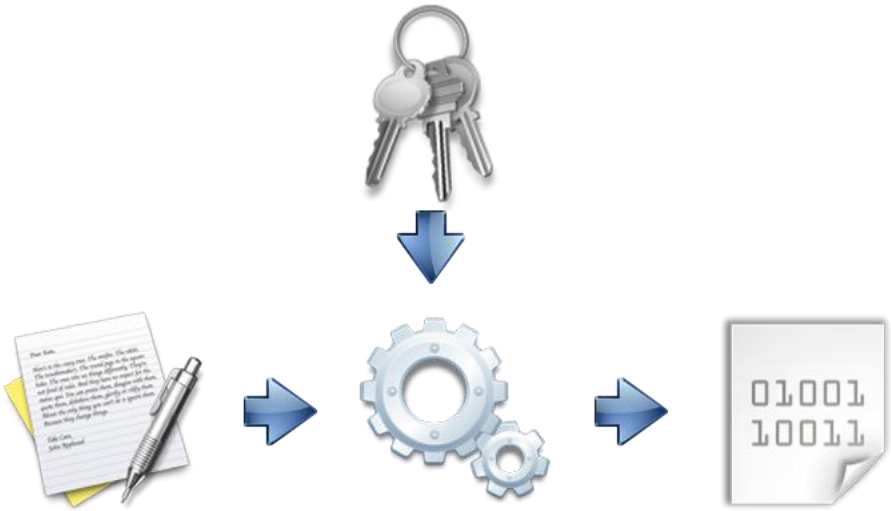
- Dato x è **semplice** calcolare H(X)
- Dato H(x) è «**impossibile**» ricavare x
  - Questa proprietà è chiamata «resistenza alla pre-immagine»

Confidenzialità

*«…means preserving **authorized restrictions** on access and disclosure, including means for protecting personal privacy and proprietary information»*

Crittografia

La crittografia tratta dei metodi per rendere un messaggio incomprensibile a coloro che non sono autorizzati ad accedervi.



- **Plaintext**: è il messaggio in chiaro
- **Cipher**: è il sistema (metodo, algoritmo) di cifratura
- **Ciphertext**: è il Plaintext processato tramite un Cipher
- **Key**: è il segreto utilizzato dal Cipher per produrre il Ciphertext

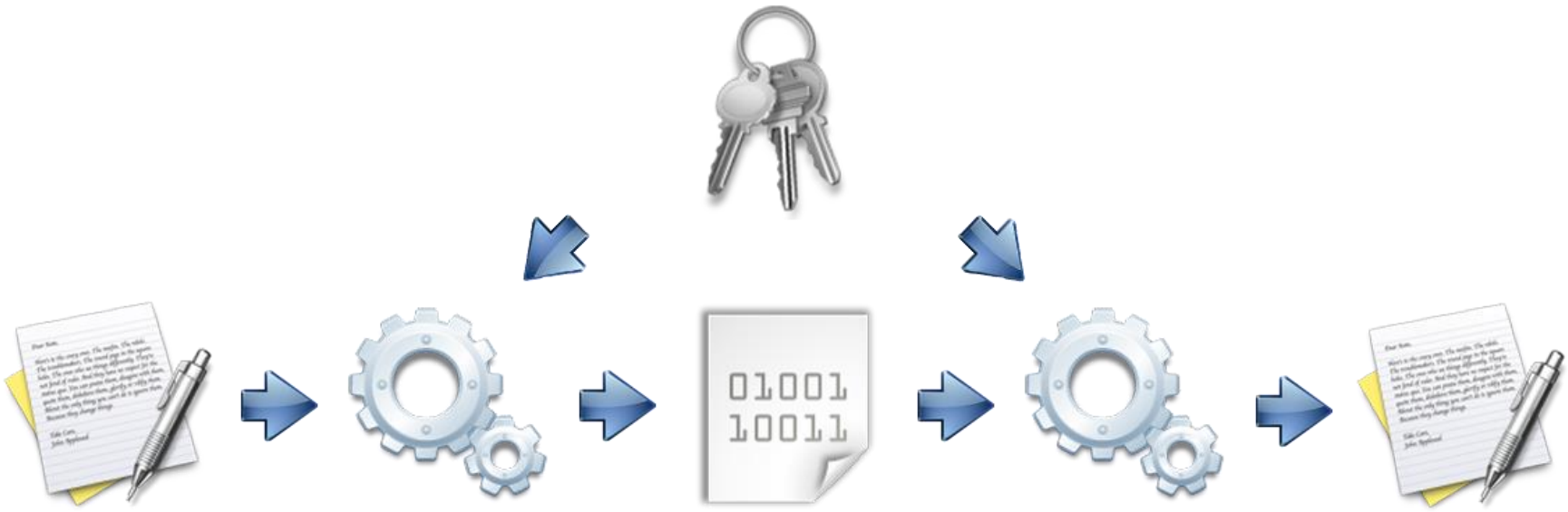
Crittografia

A cosa serve la Encryption-key?
Non può bastare soltanto il meccanismo di Encryption?

Principio di Kerckhoffs

In un sistema crittografico è importante tener segreta la chiave, non l'algoritmo di cifratura.

Crittografia Simmetrica



encryption

decryption

Crittografia Simmetrica

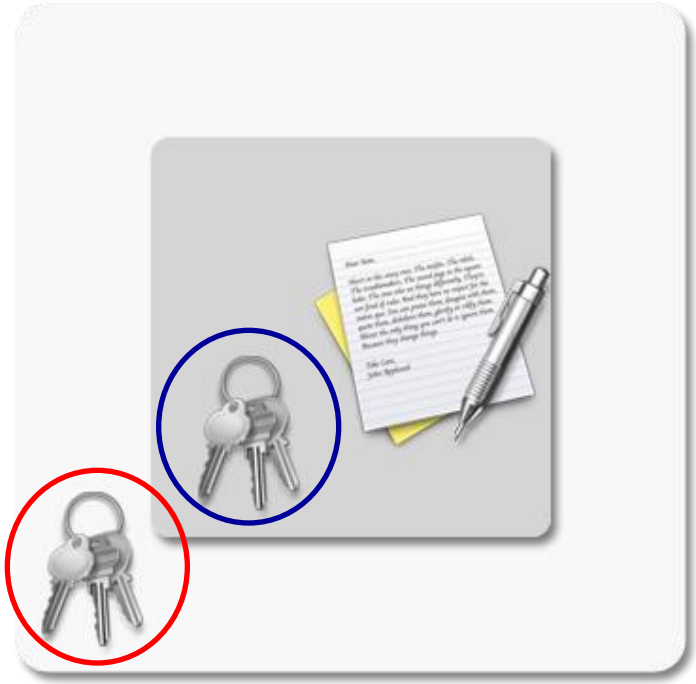
- Entrambe le parti coinvolte (A e B) devono condividere un segreto (la chiave)
- Il problema è come scambiarla:
  - **Scambio manuale**: improponibile per scenari distribuiti.
  - **Tramite Canale Sicuro**: Se A e B condividono un segreto possono usarlo per cifrare la chiave.
  - **Tramite Trusted Third Party**: Se A e B possiedono una comunicazione sicura con una terza parte C, C può distribuire loro la chiave.

Key Distribution Center

Un KDC condivide con ogni utente una master key utilizzata per garantire la sicurezza delle connessioni.

I dati trasmessi sono:

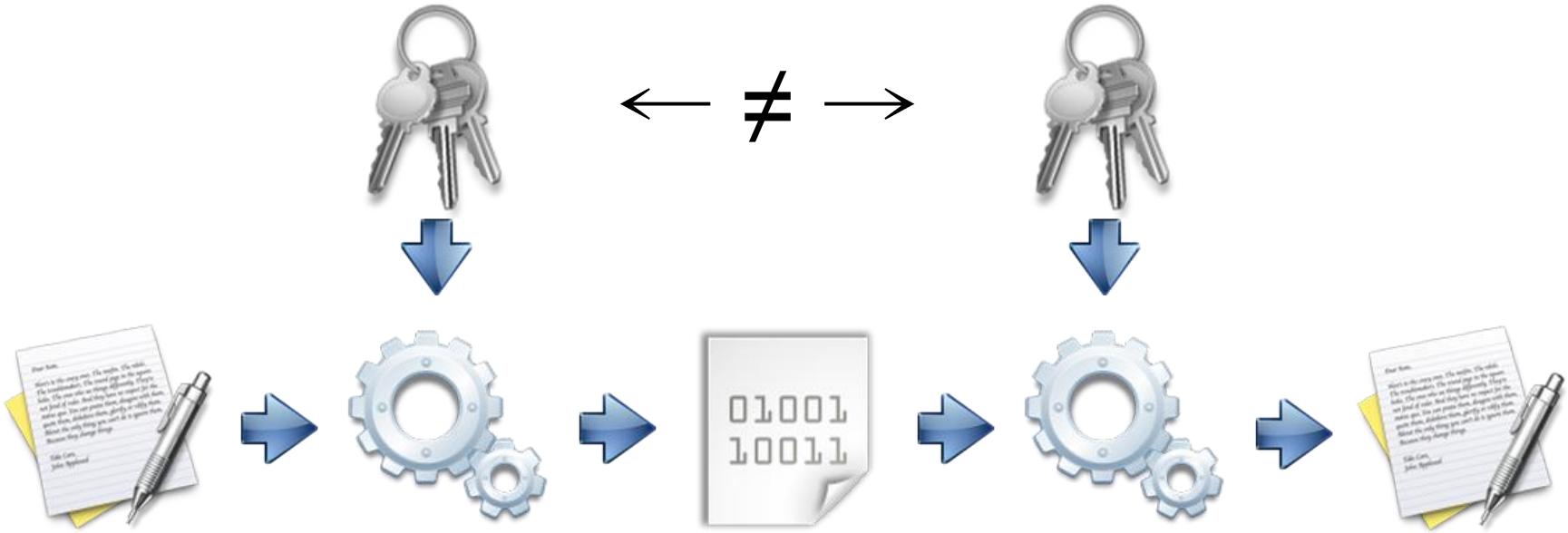
1. Protetti attraverso una session key temporanea generata per ogni nuova connessione
2. Il KDC comunica la session key cifrandola con la master key



Key Distribution Center

Sender	Receiver	Message
A	KDC	Request | N1
KDC	A	EkA {ks | Request | N1 | EkB(ks,IdA)}
A	B	EkB {ks | IdA}
B	A	Eks {N2}
A	B	Eks {f(N2)}

Crittografia Asimmetrica



Crittografia Asimmetrica

- Esistono due chiavi:
  - **Chiave pubblica**: distribuita pubblicamente dal possessore
  - **Chiave privata**: nota solo al possessore

- La chiave che cifra il messaggio *non è in grado di decifrarlo*!

La crittografia Asimmetrica contribuisce a risolvere il problema della distribuzione delle chiavi ed abilita la firma digitale.

Crittografia Asimmetrica

Problema: Chi certifica che ad una chiave pubblica sia associata una certa identità?

Soluzione: Public-Key Authority

- L'utente A interagisce con la PKA per ottenere la chiave pubblica di B
- È necessario che A conosca la chiave pubblica del PKA

Crittografia Asimmetrica

Sender	Receiver	Message
A	PKA	Request | Time1
PKA	A	EkRauth {kuB | Request | Time1}
A	B	EkuB {IdA | N1}
B	PKA	Request | Time2
PKA	B	EkRauth {kuA | Request | Time2}
B	A	EkuA {N1 | N2}
A	B	EkuB {N2}

Firma Digitale

Se esiste un'unica coppia di chiavi con cui eseguire l'operazione di cifratura e decifratura…

…perché non utilizzare la *chiave privata* per provare l'autenticità di un documento digitale?

1. A cifra il documento Dx con la propria chiave *privata*
2. A consegna Dx cifrato a B
3. B possiede la chiave pubblica di A e prova a decifrare Dx
   - Se ci riesce… sa che è stato proprio A a mandarglielo, dato che è l'unico a conoscere la chiave di cifratura!

In realtà le cose non stanno esattamente così…

Firma Digitale

Quello che B sa *davvero* è che Dx è stato cifrato con la chiave privata in coppia con la chiave pubblica in suo possesso.

Chi garantisce che l'identità di A sia associata alla coppia di chiavi?

Anche in questo caso è necessaria un'Autorità garante

Simmetrica VS Asimmetrica

Simmetrica	Asimmetrica
Necessita di: 1. Per cifratura e decifratura si usa lo stesso algoritmo e la stessa chiave 2. A e B devono condividere l'algoritmo e la chiave	**Necessita di:** 1. Per cifratura e decifratura si usa lo stesso algoritmo che opera con una coppia di chiavi diverse 2. A e B devono avere una chiave della coppia, rispettivamente
Per abilitare la Sicurezza: 1. La chiave deve essere segreta 2. Deve essere "impossibile" decifrare un msg senza informazioni sul segreto 3. La conoscenza dell'algoritmo e di campioni di testo cifrato non devono essere sufficienti per violare il sistema	**Per abilitare la Sicurezza:** 1. Una delle due chiavi deve essere mantenuta segreta 2. Deve essere "impossibile" decifrare un msg senza informazioni sul segreto 3. La conoscenza dell'algoritmo, di una delle due chiavi e di campioni di testo cifrato non devono essere sufficienti per violare il sistema

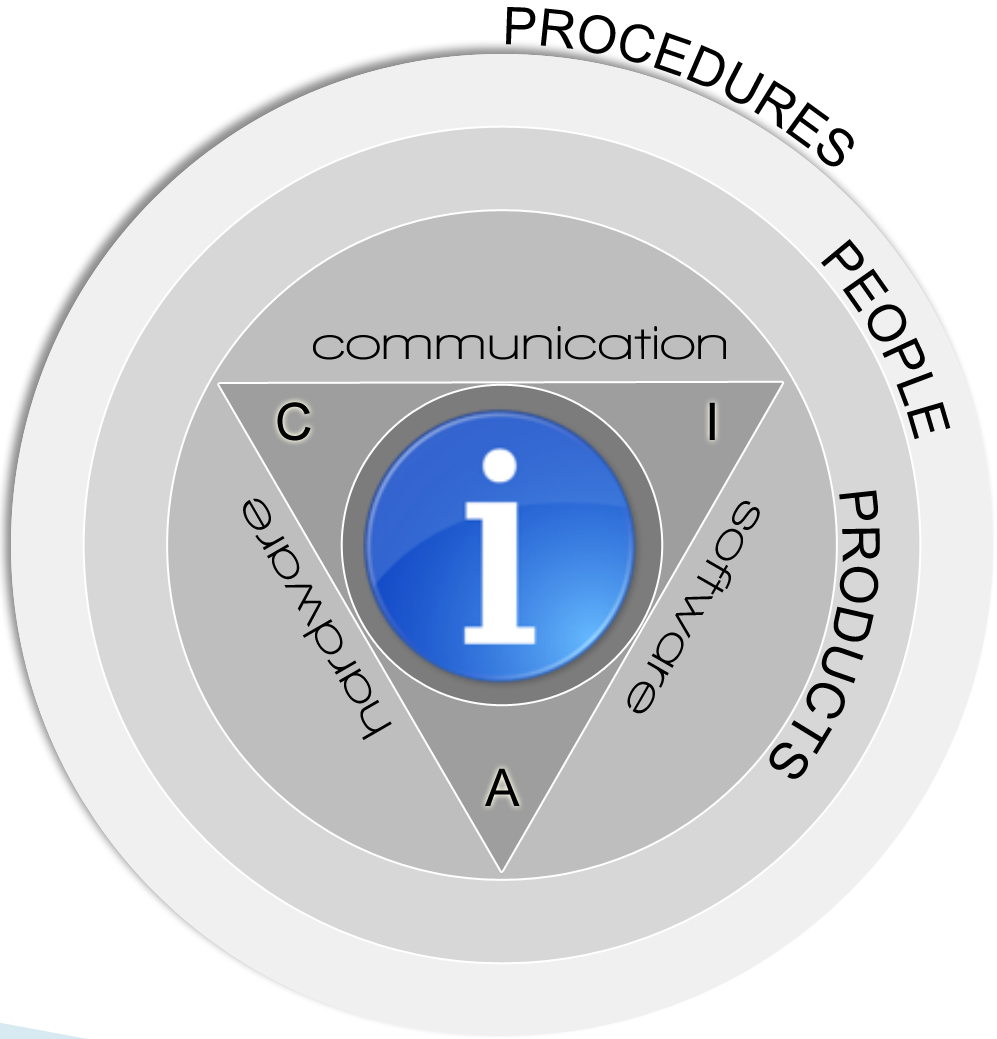
Disponibilità

«…means ensuring timely and reliable access to and use of information»

La Sicurezza è un Processo

Si utilizzano delle PROCEDURE (o policy) per dire alle PERSONE (amministratori, utenti, operatori) come utilizzare dei PRODOTTI per abilitare la Sicurezza dell'Informazione all'interno di un contesto.

La Sicurezza è un processo



Dalla teoria alla pratica…

teoria

pratica

Open Web App Security Project

- OWASP [2]
- È una community open
- Il suo scopo è supportare le organizzazioni nello sviluppo, acquisizione, manutenzione di applicazioni sicure
- Tutto il materiale prodotto da OWASP è **gratuito** ed **open**…
- … e noi lo sfrutteremo abbondantemente! [3] ;-)

OWASP top 10 (2013)

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

Injection

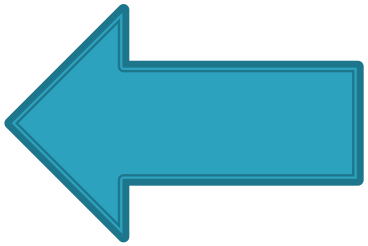
si parla di *injection* quando dati **non fidati** sono inviati ad un interprete ed **eseguiti** come un comando o una query.

- SQL Injection
- HQL Injection
- XML Injection
- Xpath Injection
- ….

SQL Injection

Precondizione per il verificarsi di questo attacco è l'impiego da parte dello sviluppatore di **query composte dinamicamente**.

SELECT credit-card
FROM users
WHERE user-id =''



… qualcosa proveniente dall'esterno

Classico attacco SQL-Injection

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

+

```
http://example.com/app/accountView?id=' or '1'='1
```

=

```
SELECT * FROM accounts WHERE custID='' or '1'='1'
```

Codice Insicuro

```
String query = "
               SELECT account_balance
               FROM user_data
               WHERE user_name = " +
               request.getParameter("customerName");

try {
Statement statement = connection.createStatement( … );
ResultSet results = statement.executeQuery( query );
}
```

SQL Injection: quali contromisure?

1. Utilizzare i «prepared statements»
2. Utilizzare le «stored procedures»
3. Eseguire l'escape di tutti gli input forniti dagli utenti [4]

Prepared Statements

Sono istruzioni **precompilate** in cui la struttura della query è mantenuta separata dai parametri (sostanzialmente statement con «?» Al posto dei parametri).

1. **Efficienza**: il DB-engine compila la stessa istruzione una volta sola

2. **Sicurezza**: tutti i parametri sono sottoposti ad *escape*!

Prepared Statements (Java EE)

```
String custname = request.getParameter("customerName");

String query = "SELECT account_balance
                    FROM user_data
                    WHERE user_name = ? ";

 PreparedStatement pstmt = connection.prepareStatement(
query );

 pstmt.setString( 1, custname);

 ResultSet results = pstmt.executeQuery( );
```

Prepared Statements (Hibernate)

```
Query unsafeHQLQuery = session.createQuery("from
      Inventory where productID
='"+userSuppliedParameter+"'");
```

```
Query safeHQLQuery = session.createQuery("from Inventory
      where productID=:productid");

safeHQLQuery.setParameter("productid",
      userSuppliedParameter);
```

Stored Procedures

Sono analoghe ai prepared statements (le strategie sono intercambiabili e si basano sullo stesso concetto): lo sviluppatore utilizza query predefinite e **memorizzate nel DB** alle quali passa i parametri in un secondo momento.

Stored Procedure (Java)

```
String custname = request.getParameter("customerName");

try {
        CallableStatement cs = connection.prepareCall("{call
sp_getAccountBalance(?)}");

        cs.setString(1, custname);
        ResultSet results = cs.executeQuery();
}
```

Stored Procedures Warning

L'impiego delle SP di per sé **non garantisce** di essere al sicuro dalla injection se… le vostre procedure includono query create **dinamicamente**!

Stored Procedure – dynamic query

```
PROCEDURE SafeGetBalanceQuery(@UserID varchar(20),
     @Dept varchar(10))
AS BEGIN
DECLARE @sql VARCHAR(200)

SELECT @sql =
       'SELECT balance FROM accounts_table WHERE '
     + 'user_ID = @UID AND department = @DPT'

   EXEC sp_executesql @sql,
     '@UID VARCHAR(20), @DPT VARCHAR(10)',
     @UID=@UserID, @DPT=@Dept
END
```

Bind variables

Gli input sono «dati» e non codice!

Input Escaping

Ogni DBMS supporta uno o più schemi di *escape* dei caratteri, per certi tipi di query. Se ogni input è processato in questo modo, il DBMS non lo confonderà col codice scritto dallo sviluppatore.

Tuttavia…

È sempre meglio utilizzare le prime due tecniche che scrivere un *escaper* a mano…

XML Injection

È un problema che affligge i database XML.

Sostanzialmente, i dati sono immagazzinati all'interno di un consistente file XML strutturato.

XML Injection – DB Esempio

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
	<user>
		<username>gandalf</username>
		<password>!c3</password>
		<userid>0</userid>
		<mail>gandalf@middleearth.com</mail>
	</user>
	<user>
		<username>Stefan0</username>
		<password>w1s3c</password>
		<userid>500</userid>
		<mail>Stefan0@whysec.hmm</mail>
	</user>
</users>
```

XML DB

Use case: registrazione di uno user

- Username: tony
- Password: Un6R34kb!e
- E-mail: s4tan@hell.com

I dati sono codificati nell' HTTP request URL

`http://www.example.com/addUser.php?`**`user=tony&pswd=Un6R34kb!e&mail=s4tan@hell.com`**

XML Injection – DB Esempio

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        <user>
                <username>gandalf</username>
                <password>!c3</password>
                <useri
                <mail>
        </user>
        …
        <user>
                <userr
                <passw
                <userid>500</userid>
                <mail>Stefan0@whysec.hmm</mail>
        </user>
</users>
```

```
<user>
        <username>tony</username>
        <password>Un6R34kb!e</password>
        <userid>500</userid>
        <mail>s4tan@hell.com</mail>
</user>
```

XML Injection Attack!

Use case: registrazione di uno user

- Username: **tony**
- Password: **Un6R34kb!e</password><!--**
- E-mail:
  --><userid>0</userid><mail>s4tan@hell.com

XML Injection Attack!

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        …
        <user>
                <username>tony</username>
                <password>Un6R34kb!e</password>
                <!--</password>
                <userid>500</userid>
                <mail>-->
                <userid>0</userid>
                <mail>s4tan@hell.com</mail>
        </user>
</users>
```

Admin privilege!

Cos'è XPath?

XPath è un linguaggio che utile per eseguire «interrogazioni» all'interno di un documento XML [5].

- Seleziona tutti i titoli
  - `/bookstore/book/title`
- Seleziona il titolo del primo libro
  - `/bookstore/book[1]/title`
- Seleziona tutti i prezzi
  - `/bookstore/book/price[text()]`
- Seleziona tutti i nodi «prezzo» maggiori di 35
  - `/bookstore/book[price>35]/price`

```
<bookstore>
      <book category="COOKING">
            <title lang="en">Everyday
            Italian</title>
            <author>Giada De
            Laurentiis</author>
            <year>2005</year>
            <price>30.00</price>
      </book>
            <book category="CHILDREN">
            <title lang="en">Harry
            Potter</title>
            <author>J K. Rowling</author>
            <year>2005</year>
            <price>29.99</price>
      </book>
      <book category="WEB">
            <title lang="en">XQuery Kick
            Start</title>
            <author>James
            McGovern</author>
            <author>Per Bothner</author>
            Nagarajan</author>
            <year>2003</year>
            <price>49.99</price>
      </book>
</bookstore>
```

XPath Injection

Scenario di autenticazione: un'applicazione utilizza dati inseriti dall'utente per verificare la sua esistenza nel DB XML

```
String FindUserXPath;

FindUserXPath = "//Employee[UserName/text()='" +
        Request("Username") + "' And Password/text()='" +
        Request("Password") + "']";
```

Xpath Injection

```
Username: blah' or 1=1 or 'a'='a
Password: blah
```

FindUserXPath diviene

```
//Employee[UserName/text()='blah' or 1=1 or
        'a'='a' And Password/text()='blah']
```

Che è logicamente equivalente a

```
//Employee[(UserName/text()='blah' or 1=1) or
        ('a'='a' And Password/text()='blah')]
```

Broken Authentication and Session Management

Le funzionalità relative ad autenticazione e gestione della sessione spesso non sono implementate correttamente. Questo consente ad un attaccante di compromettere password, chiavi, session token, o generalmente di sfruttare errori di implementazione per assumere l'identità di un altro utente.

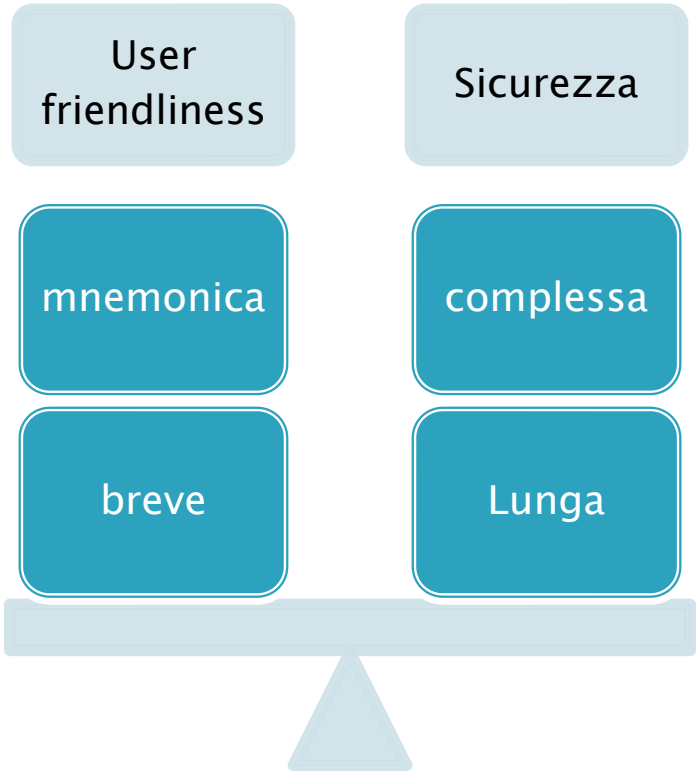
Autenticazione

È la **verifica** che un individuo o un'entità siano davvero **chi sostengono di essere** (claim).

Solitamente l'autenticazione è attuata sottomettendo un ID (username) e una o più informazioni private.

Password balance…

Usare una politica sicura per le password…



Password (ragionevolmente) sicure

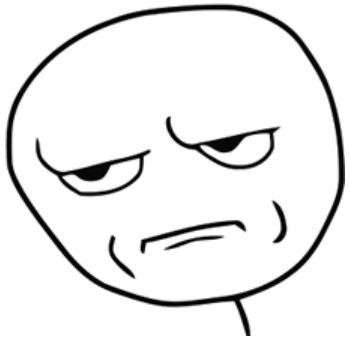
- Lunghezza $l_{inf} < l \leq l_{sup}$
  - $l_{inf} \geq 10$ per evitare l'uso di pswd troppo brevi
  - $l_{sup} \geq 128$ per non scoraggiare l'uso di passphrase

- Complessità (pick three!)
  - Almeno un carattere maiuscolo
  - Almeno un carattere minuscolo
  - Almeno un numero
  - Almeno un carattere speciale

Domanda di Sicurezza

Woah!!!

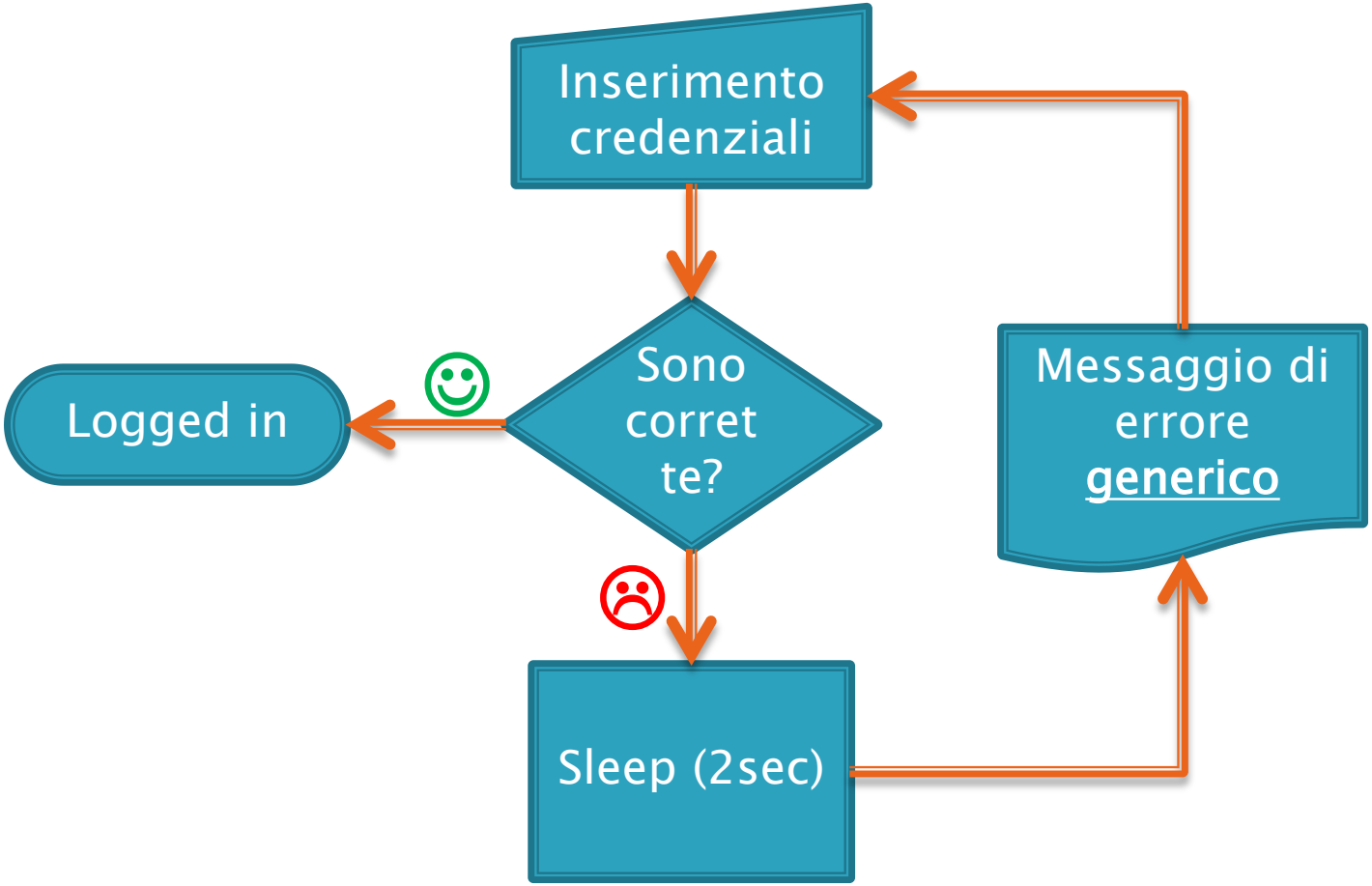
- User: Stefano_Turchi
- Password: zRGV(7_XHq-Qp<xv7{,[
- Domanda di riserva: Qual è il nome del tuo animale domestico?; Fuffy



Trasferimento di credenziali

- Solo attraverso protocolli **sicuri**
  - HTTP over TLS (HTTPS) è ok ☺
- In pratica…
  - L'invio di credenziali per login e registrazione devono riferire ad un URL di tipo `https://...`
  - Tutte le pagine private devono essere accessibili tramite URL di tipo `https://`

Gestire il processo di login



Gestione della Sessione

È un processo attraverso il quale il server mantiene **lo stato di un'entità** che interagisce con lui. È necessario affinché il server **ricordi come reagire a richieste successive attraverso una transazione**.

Le sessioni sono mantenute dal server attraverso un **session id** che può essere ritrasmesso dal client durante l'interazione col server.

Gestione della Sessione

In concreto, si tratta di gestire la comunicazione del session id.

- Una buona scelta è quella di affidarsi a framework consolidati
- …se questo non è possibile, almeno fare attenzione a **NON** comunicare il session id usando **URL-parameters** (gli URL si condividono!)

HTTPS → HTTP → HTTPS

- Attenzione alle transizioni da HTTPS ad HTTP perché **usciti dal canale sicuro il session id è esposto**!

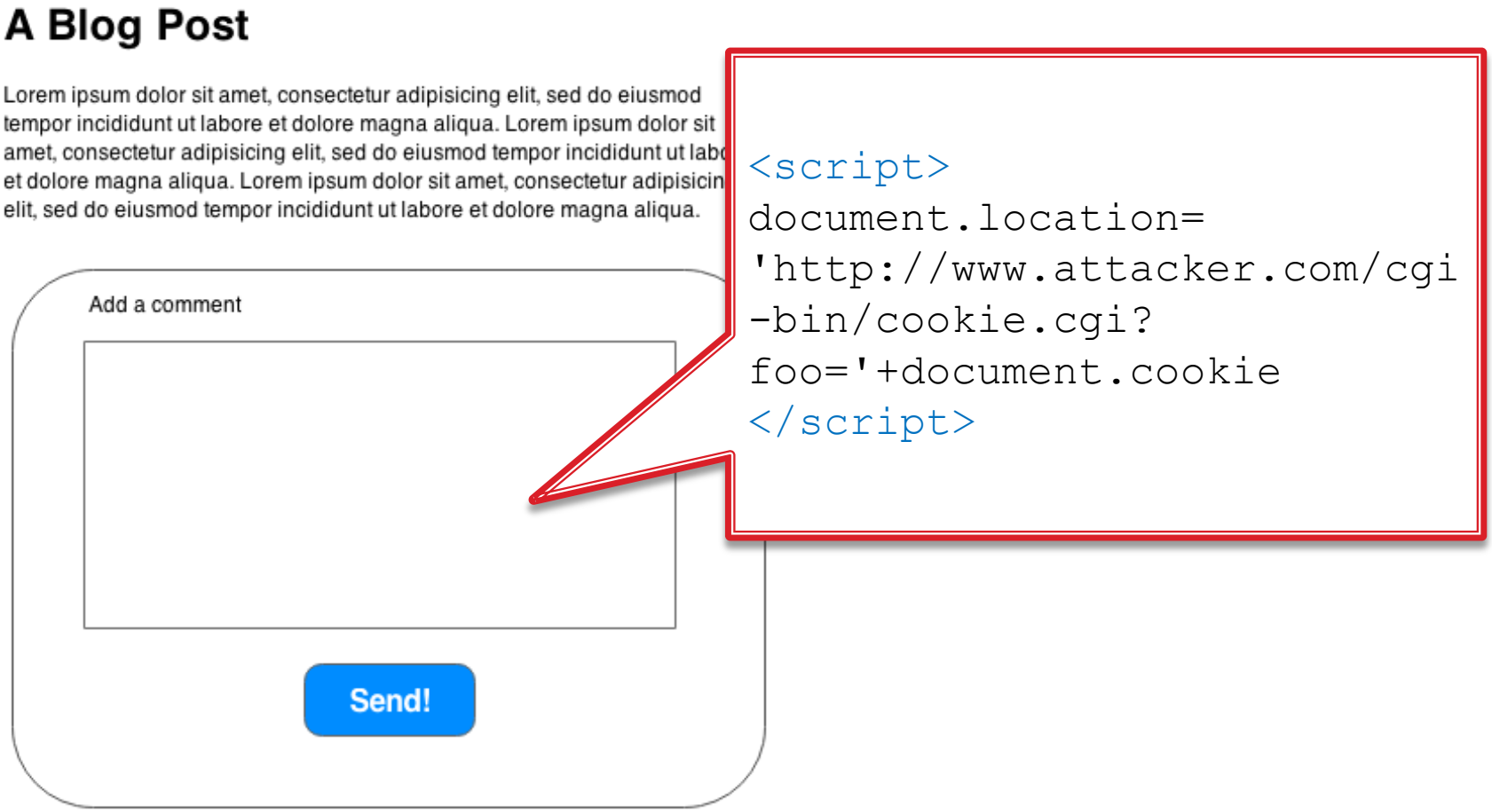
- Usare sempre l'attributo «**Secure**» per i cookie che istruisce il browser ad inviare il session id solo su canali sicuri.

Cross-Site Scripting (XSS)

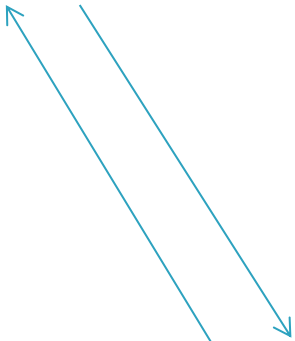
Attacchi di tipo XSS avvengono quando un'applicazione **importa dati non fidati** e li **invia ad un browser** senza un'appropriata validazione.

Un attacco XSS mira ad **eseguire script sul client** dell'utente allo scopo di sottrarre informazioni di sessione, compromettere siti web o redirigere l'utente verso siti malevoli.

Stored XSS Scenario



Reflected XSS Scenario



User: bob
Pswd: p4ssword

Sorry bob,
incorrect login

Reflected XSS Scenario

Sorry
<script>
evil
</script>,
incorrect login

Reflected XSS Scenario

Sorry
<script>
evil
</script>,
incorrect login

What's evil?

- Sostituire la pagina di login con una pagina clone che invii le credenziali all'attaccante
- Rubare il session id dalla variabile document.cookie

vittima

attaccante

Proteggersi da XSS

1. Mai inserire dati non fidati…

```
<script>...NEVER PUT UNTRUSTED DATA HERE...</script> directly in a script

<!--...NEVER PUT UNTRUSTED DATA HERE...--> inside an HTML comment

<div ...NEVER PUT UNTRUSTED DATA HERE...=test /> in an attribute name

<NEVER PUT UNTRUSTED DATA HERE... href="/test" /> in a tag name

<style>...NEVER PUT UNTRUSTED DATA HERE...</style> directly in CSS
```

Proteggersi da XSS

2. Prima di inserire dati in un elemento content di un file HTML esegui sempre l'esacape!

```
<body>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</body>

<div>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</div>

any other normal HTML elements


& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;
/ --> &#x2F;
```

Proteggersi da XSS

3. Prima di inserire dati in un attributo HTML esegui sempre l'escape!

```
<div attr=...ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE...>content</div>     inside UNquoted attribute

 <div attr='...ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE...'>content</div>   inside single quoted attribute

 <div attr="...ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE...">content</div>   inside double quoted attribute
```

Proteggersi da XSS

4. Prima di inserire dati in uno snippet JavaScript esegui sempre l'esacape!

```
<script>alert('...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...')</script>
inside a quoted string

 <script>x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...'</script>
one side of a quoted expression

 <div onmouseover="x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE...'"</div>  inside quoted event handler
```

Tuttavia esistono funzioni JavaScript che non sono mai sicure… ☹

```
<script>
 window.setInterval('...EVEN IF YOU ESCAPE DATA YOU ARE XSSED HERE...');
</script>
```

Proteggersi da XSS

5. Prima di inserire dati in un tag <style> esegui sempre esacape e validazione!

```
<style>selector { property : ...ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE...; } </style>     property value

 <style>selector { property : "...ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE..."; } </style>   property value

 <span style="property : ...ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE...">text</span>       property value
```

Tuttavia esistono contesti CSS che non sono mai sicuri… ☹

```
{ background-url : "javascript:alert(1)"; }  tutti gli URL devono avere
http: come protocollo!

{ text-size: "expression(alert('XSS'))"; }  solo in IE
```

Proteggersi da XSS

6. Prima di inserire dati tra i parametri di un URL esegui sempre l'esacape!

```
<a href="http://www.somesite.com?test=...ESCAPE UNTRUSTED DATA BEFORE
PUTTING HERE...">link</a >
```

Per tutto questo c'è…

ESAPI

(The OWASP Enterprise Security API)

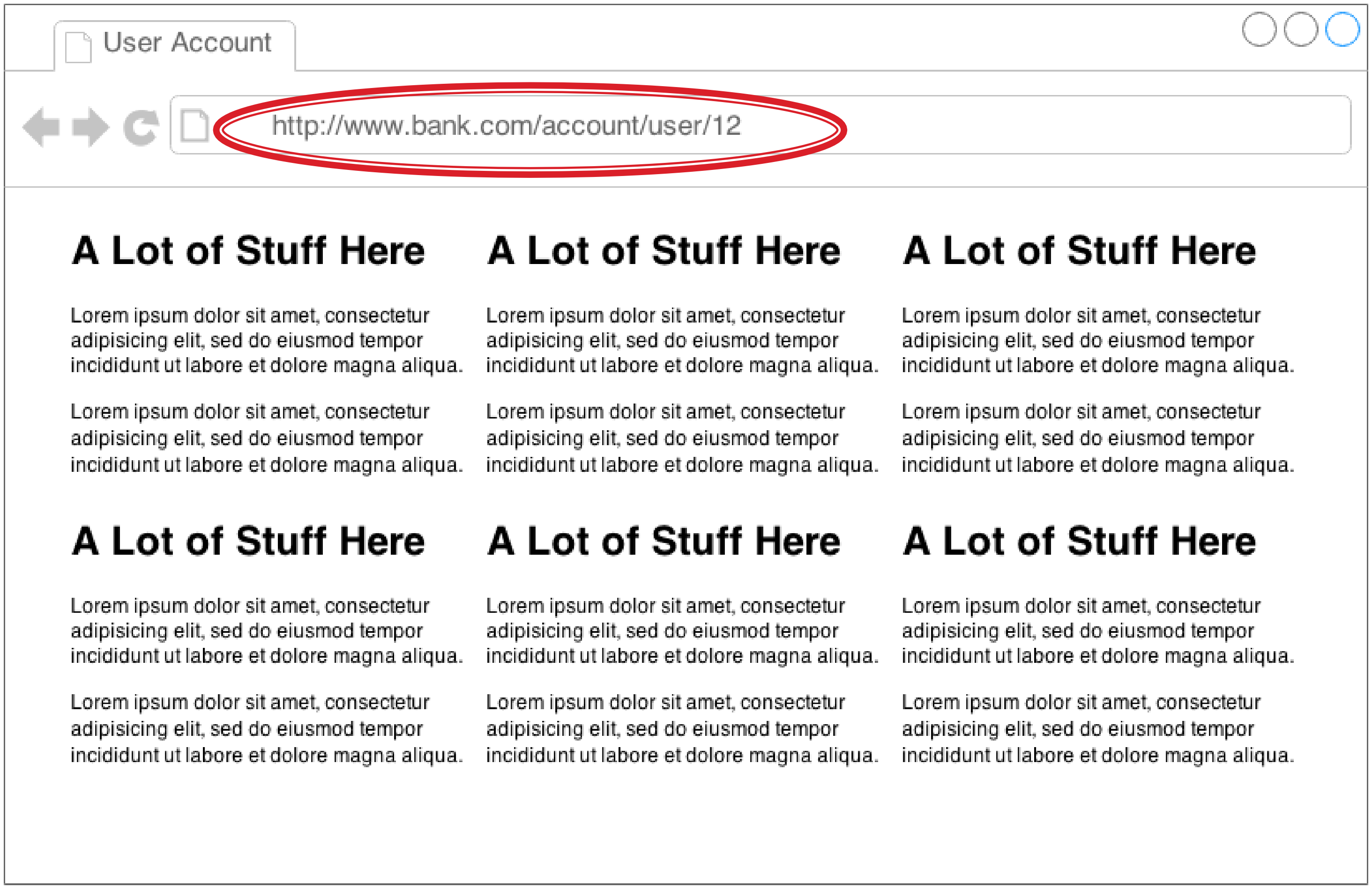
https://www.owasp.org/index.php/Esapi

Insecure Direct Object References

Si ha questo tipo di vulnerabilità quando uno sviluppatore **espone un riferimento** verso un **oggetto interno** come un file, una cartella o la chiave di un database.

Senza un appropriato controllo degli accessi, un attaccante può manipolare questi riferimenti per accedere a dati non autorizzati.

IDOR Scenario



IDOR Scenario

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt = connection.prepareStatement(query , … );

pstmt.setString( 1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

…ma l'utente semplicemente digita…

```
http://www.bank.com/account/not_my_account
```

Evitare IDOR

- Se possibile, evitare di esporre riferimenti diretti, aggiungendo un livello di indirezione interno all'applicazione
- Ma **soprattutto** implementare una politica di controllo degli accessi funzionale e capillare

Security Misconfiguration

Un buon livello di sicurezza richiede una configurazione sicura ben definita per l'applicazione, i framework utilizzati, application server, web server, database server e piattaforma.

Una configurazione sicura deve essere implementata e mantenuta poiché **quella di default è spesso insicura**. In aggiunta, il software dovrebbe essere mantenuto aggiornato.

Scenari di Attacco

- La console di amministrazione dell'application server non è stata rimossa e continua ad essere accessibile con le **credenziali di default.**

- Il *listing* delle directory non è disabilitato, un attaccante scopre che può eseguire il comando per cercare un file, trova i sorgenti, li decompila e fa *reverse-engineering*

Scenari di Attacco

- Quando si verifica un errore l'applicazione restituisce lo *stack-trace* siano all'utente, esponendo potenzialmente alcuni security flaw

- L'istallazione dell'application server prevede una **sample application** (con security flaw noti) che non è stata rimossa dall'ambiente di produzione

Le domande da porsi

- Tutti i componenti del mio ambiente di produzione (OS, Web/App Server, DBMS, applications, librerie) sono aggiornati?
- Ho qualche feature non necessaria installata nel mio ambiente?
- Gli account di default sono ancora attivi?
- Durante la gestione degli errori sono comunicate all'utente informazioni non necessarie?
- Le configurazioni di sicurezza del mio framework sono abilitate?

Sensitive Data Exposure

Molte web application non proteggono i dati sensibili (carte di credito, nominativi, account) in modo appropriato. Se un attaccante riesce a sottrarre queste informazioni, può compiere diversi atti criminali.

SDE Scenarios

- Un'applicazione cifra i numeri di carta di credito usando un sistema di cifratura **a livello DB**. Questo significa che i dati sono anche **decifrati** una volta recuperati dal DB! Usando la SQL Injection, un attaccante recupera tutti i dati sensibili

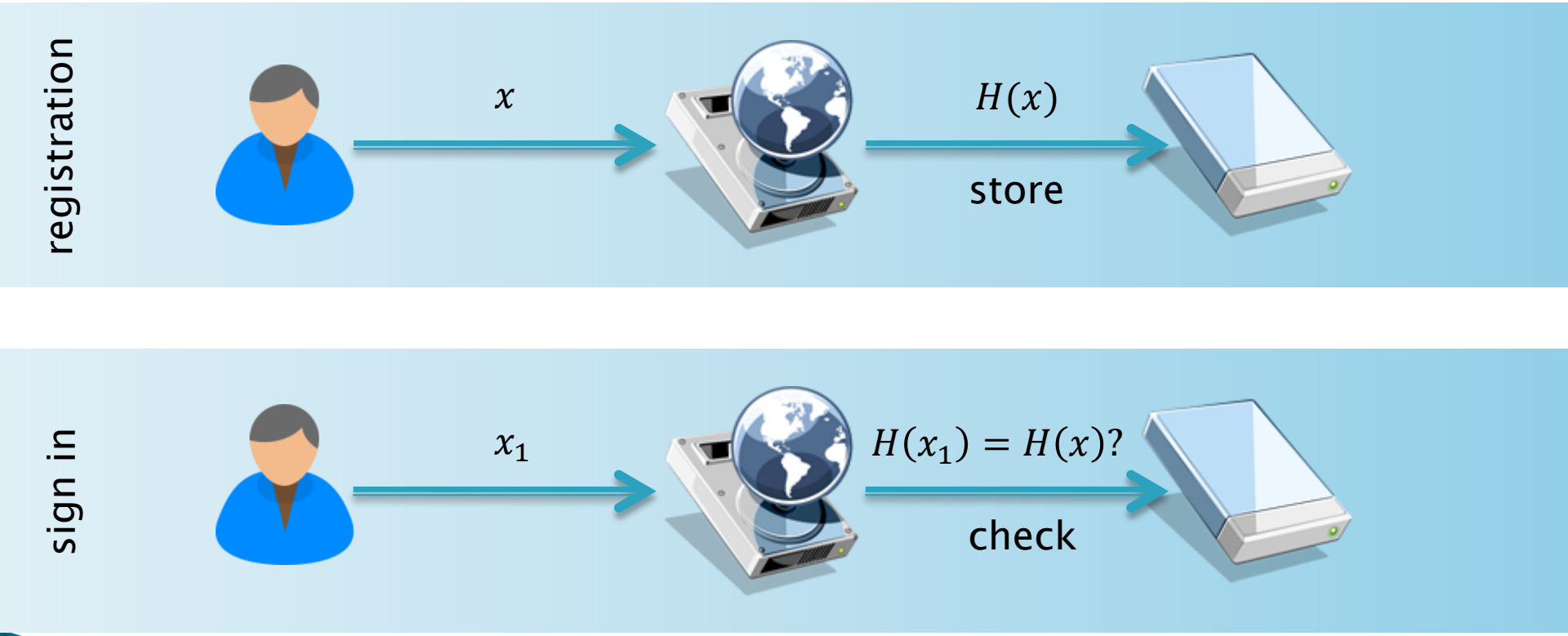
- Un sito **non utilizza TLS** per le pagine autenticate, un attaccante monitora la rete e ruba i contenuti del traffico HTTP

Come gestire i dati sensibili

- Memorizza solo i dati sensibili di cui hai bisogno!
- Usa solo algoritmi crittografici forti (☺ AES, RSA, SHA-256; MD5, SHA1 ☹)
- Assicurati che i numeri casuali siano crittograficamente forti
- Quando devi eseguire più cifrature per scopi diversi, usa chiavi diverse
- Assicurati che ogni *secret key* sia protetta da accessi non autorizzati
- Memorizza le password processate attraverso *salted hashing*

Funzioni Hash

$x = p4ssword$ $H(x) = 93863810133ebebe6e4c6bbc2a6ce1e7$



Salare gli Hash delle password!

Il «sale» è un codice casuale crittografico di lunghezza fissata

Scegliete un sale diverso per ogni credenziale e memorizzatela così!

$$H(salt, x)$$

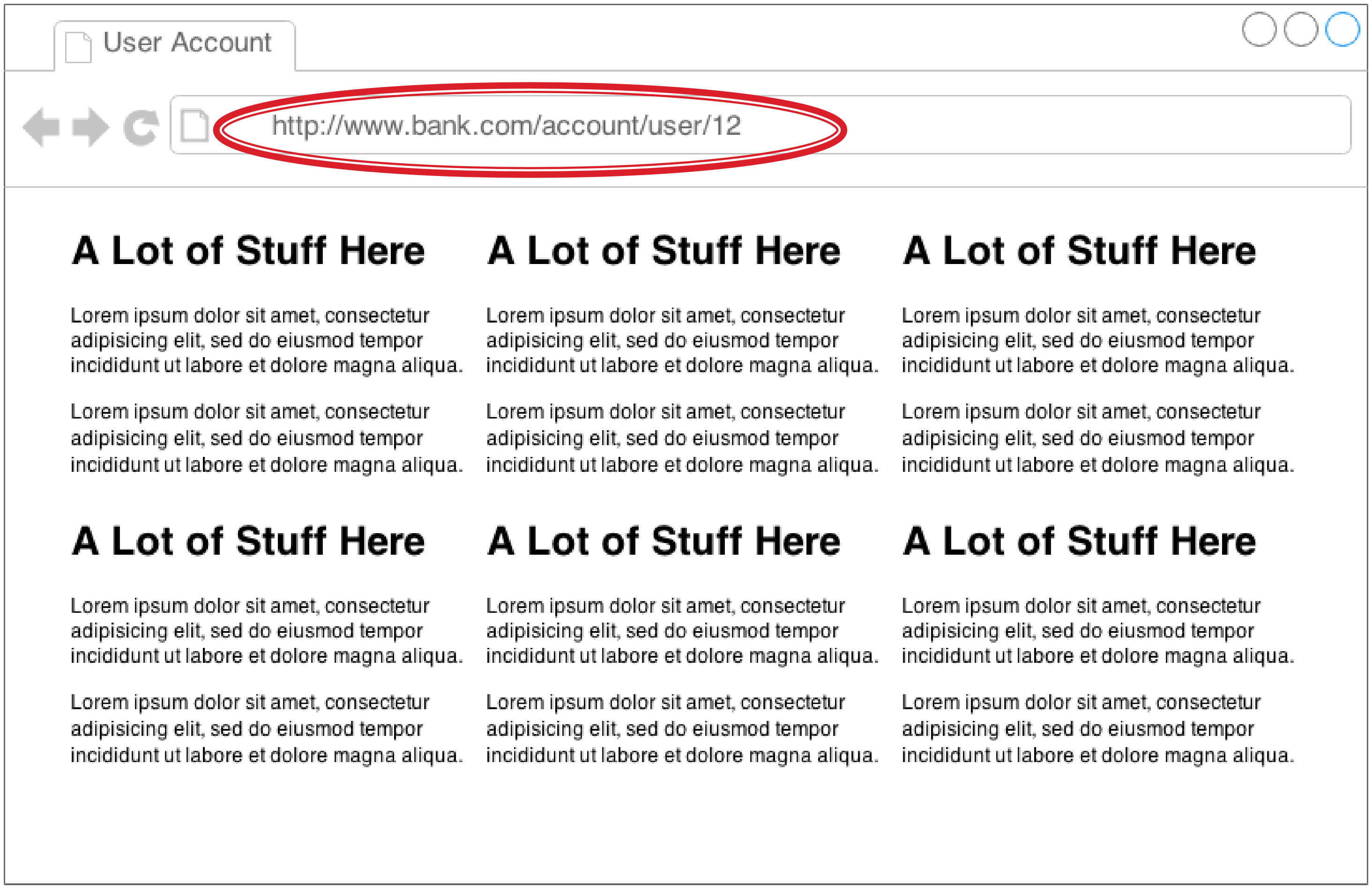
Le vostre credenziali saranno molto più al sicuro da attacchi a forza bruta e *rainbow tables*!

Missing Function Level Access Control

Molte web application verificano i diritti di accesso a certe funzionalità appena prima di mostrare la relativa funzione sull'interfaccia.

Tuttavia è necessario eseguire gli stessi controlli sul server quando ognuna di queste funzionalità è acceduta. Se le request non sono verificate, un attaccante può forgiarne una per accedere a funzionalità a lui precluse.

Attack Scenario

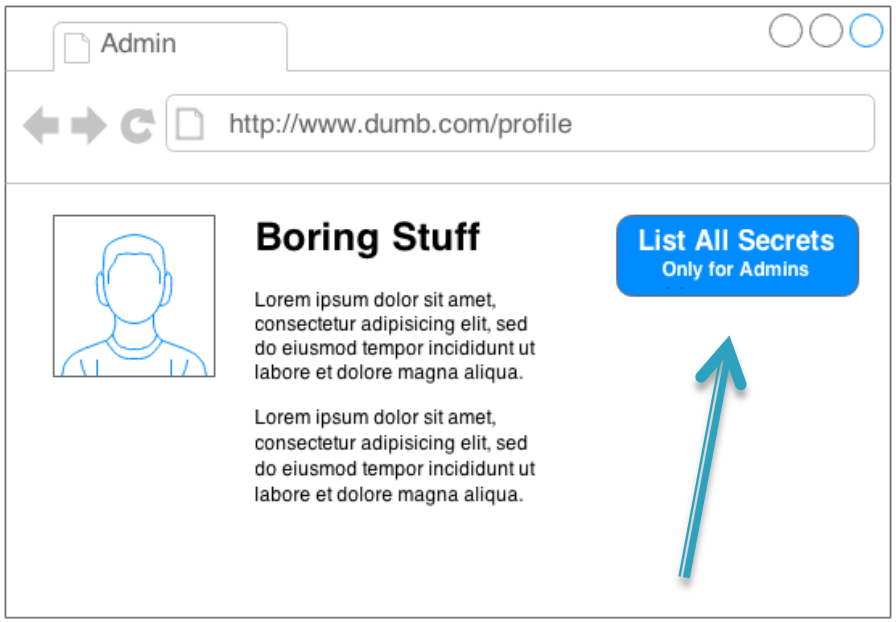


Una questione di verifica…

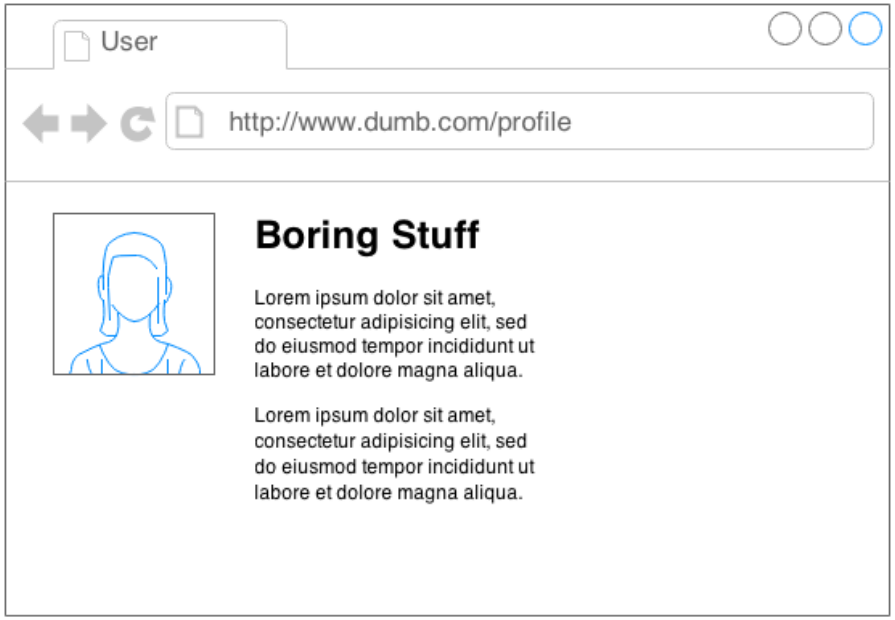
- Verificare i ruoli
- Verificare i singoli utenti
  - Due utenti con il medesimo ruolo possono avere accesso a spazi disgiunti
- NON attuare un meccanismo di *security through obscurity*
  - Ciò che non mostrate nell'interfaccia non è detto che sia inaccessibile!

Security through Obscurity

Admin



User




```
href='http://www.dumb.com/admin/all_the_yummy_secrets'
```

Il bottone non c'è, ma…
cosa accade se l'utente inserisce quell'URL nel browser?

Fail Securely

```
boolean isAdmin = true;

try
{
        codeWhichMayFail();
        isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex)
{
        log.write(ex.toString());
}
```

Fail Securely

```
boolean isAdmin = false;

try
{
        codeWhichMayFail();
        isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex)
{
        log.write(ex.toString());
}
```

Cross-Site Request Forgery

CSRF è un tipo di attacco in cui la vittima è indotta ad eseguire a sua **insaputa** alcune operazioni **verso una web application presso la quale è autenticata**.

CSRF Scenario

www.bank.com

existing session

victim

CSRF Scenario

www.bank.com

existing session

victim

CSRF Scenario

Super cool KITTEN video! **OMG!**

```
<img src="http://www.bank.com/
transfer.do?acct=Bob&euro=100000">
```

CSRF non funziona solo in GET

Data questa request richiesta dall'applicazione

```
POST http://bank.com/transfer.do HTTP/1.1

acct=BOB&amount=100
```

L'attaccante può implementare un form

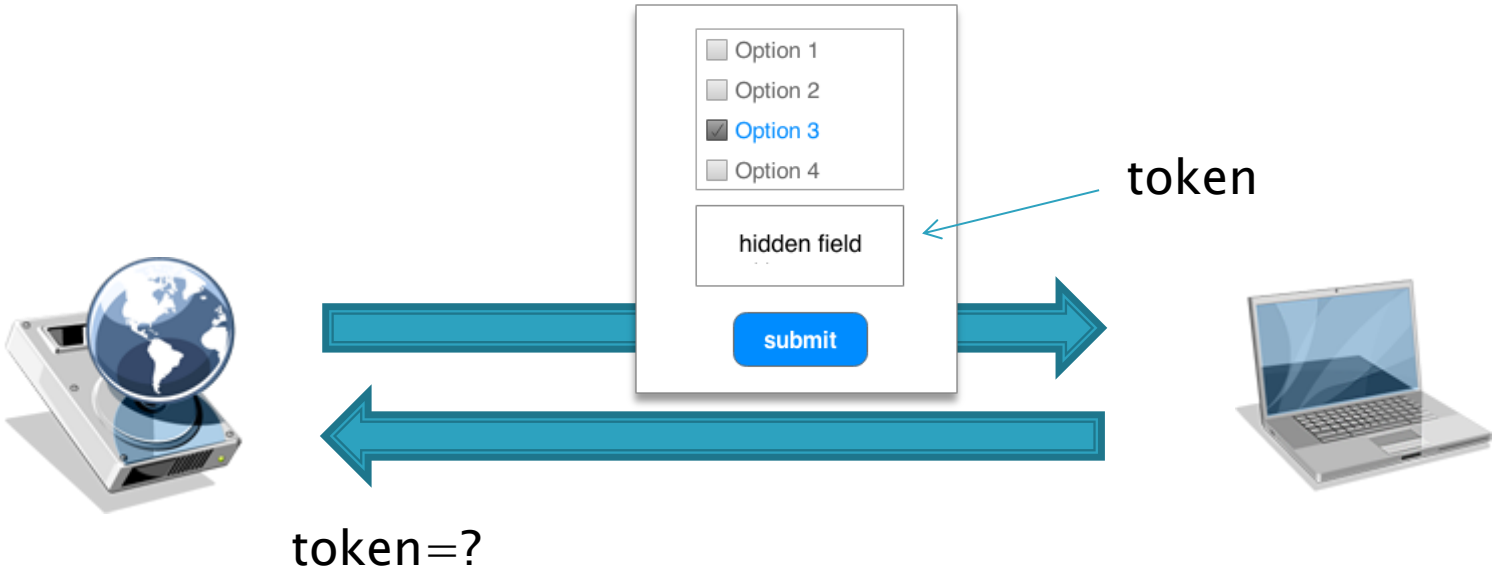
```
<form action="http://bank.com/transfer.do" method="POST">
   <input type="hidden" name="acct" value="MARIA"/>
   <input type="hidden" name=amount" value="100000"/>
   <input type="submit" value="View my pictures"/>
</form>
```

Che può essere sottomesso al caricamento di una pagina!

```
<body onload="document.forms[0].submit()">
<form...
```

CSRF quali contromisure?

- Sostanzialmente, è bene che il server associ ad ogni richiesta un token specifico col quale validare un'eventuale interazione.



Using Components with Known Vulnerabilities

Componenti come librerie, framework ed altri moduli software **sono eseguiti quasi sempre con il massimo dei privilegi**. Se un attaccante riesce a sfruttare la vulnerabilità di uno di questi, è possibile incorrere in problemi molto seri.

La applicazioni che utilizzano componenti con vulnerabilità note possono compromettere le difese applicative e dare luogo ad una serie di attacchi.

Buone Pratiche

- Identifica tutti i componenti che stai usando (incluse le dipendenze)
- Per ognuno di questi, verifica il suo stato di aggiornamento
- Ove necessario considera di creare dei wrapper che disabilitino le funzionalità non utilizzate

Unvalidated Redirects and Forwards

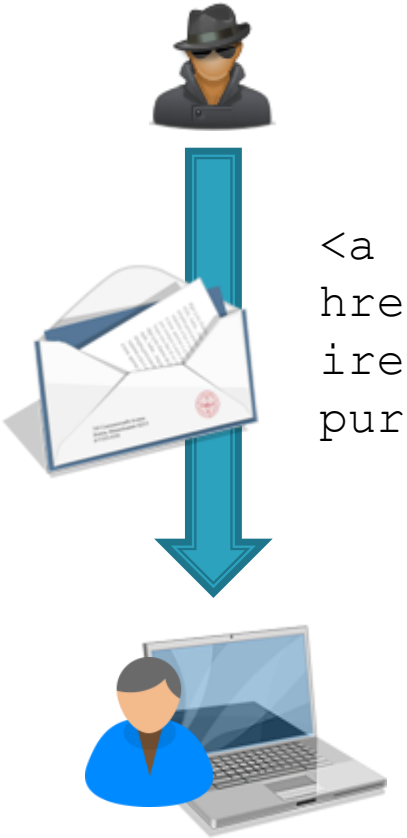
Spesso le web application eseguono *forward* e *redirect* per guidare la navigazione degli utenti.

Senza ricorrere ad una validazione appropriata, un attaccante può redirigere un utente verso siti malevoli o utilizzare i forward per accedere a pagine non autorizzate.

Attack Scenario

`http://www.dumb.com/redirect.jsp`

Redirect.jsp



```
<a
href='http://…/red
irect.jsp?url=www.
pure_evil.com>
```

Attack Scenario

```
http://www.dumb.com/boring.jsp?fwd=admin.jsp
```

Se il controllo degli accessi non è ben implementato è possibile che un forward del genere porti l'attaccante ad una pagina di amministrazione!

Contromisure

- Evitare i redirect e i forward :D
- Se non è possibile, non utilizzare parametri manipolabili dall'utente per decidere la destinazione
- Se anche questo non è possibile, assicurarsi che tali parametri siano validi e autorizzati per l'utente

grazie per l'attenzione!

Reference

- [1] Richard. Kissel, National Institute of Standards e Technology (U.S.) Glossary of key information security terms [electronic resource] / edited by Richard Kissel. English. U.S. Dept. of Commerce, National Institute of Standards e Technology, [Gaithersburg, Md.] : 2006, 87 p. :
- [2] www.owasp.org
- [3] www.owasp.org/index.php/Top_10_2013-Top_10
- [4] www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- [5] www.w3schools.com/xpath/xpath_examples.asp

Resources

- [slide 22] “Information" by Visual Pharm is licensed under Attribution-NoDerivs 3.0 Unported
- [slide 36]”Simplicio" by Neurovit is licensed under Creative Commons Attribution 3.0
- [slides 91,98] ”Agent" by Pixel Mixer
- [slide 92]"Retouched Kitty" by Ozan Kilic is licensed under Creative Commons Attribution 2.0
- [slide 92]“Social media" by Lotcos is licensed under Creative Commons Attribution 3.0