Sicurezza nelle applicazioni, parte 1

Corso di Sicurezza e affidabilità

Anno accademico 2024-2025

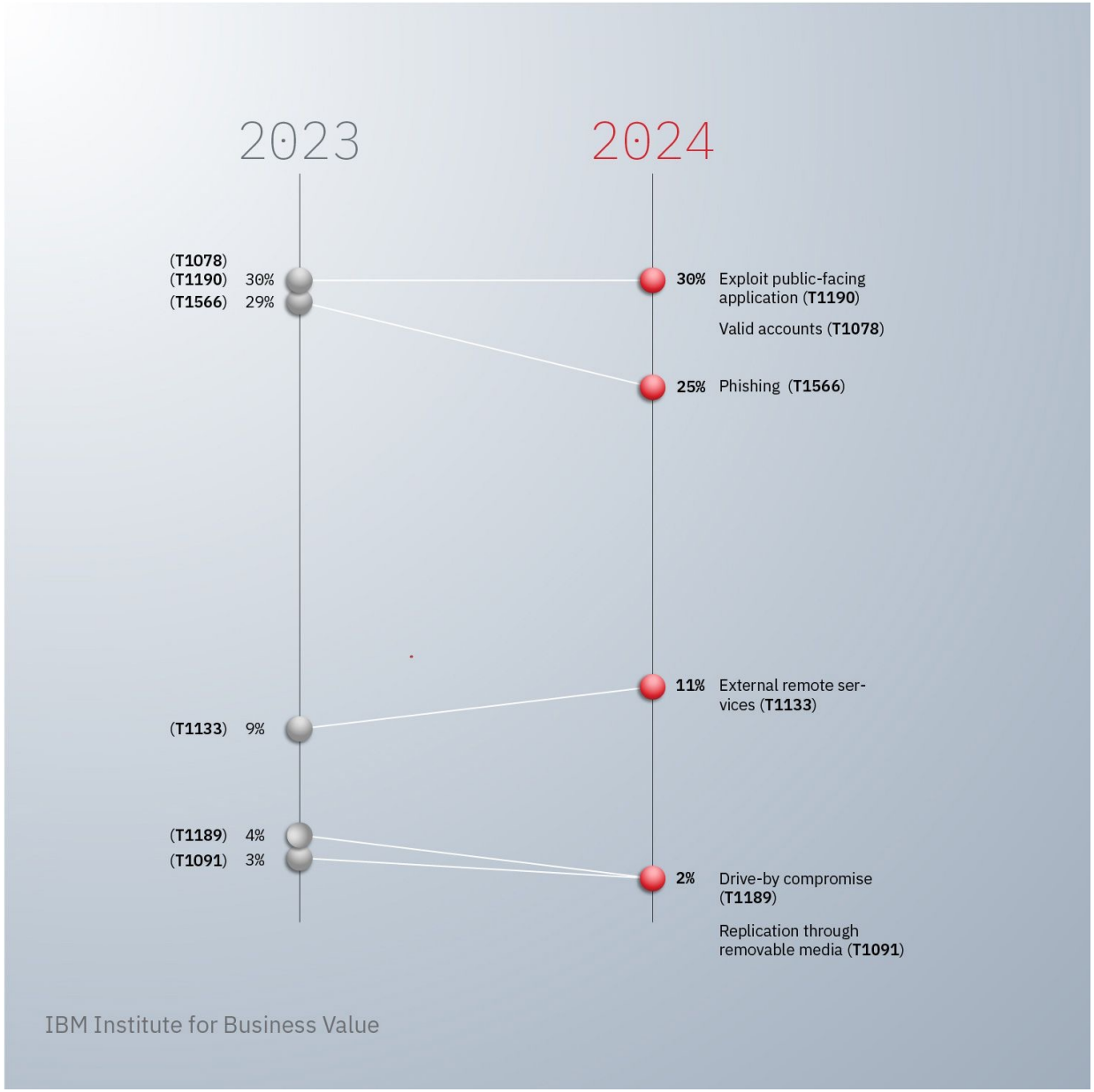
Argomenti

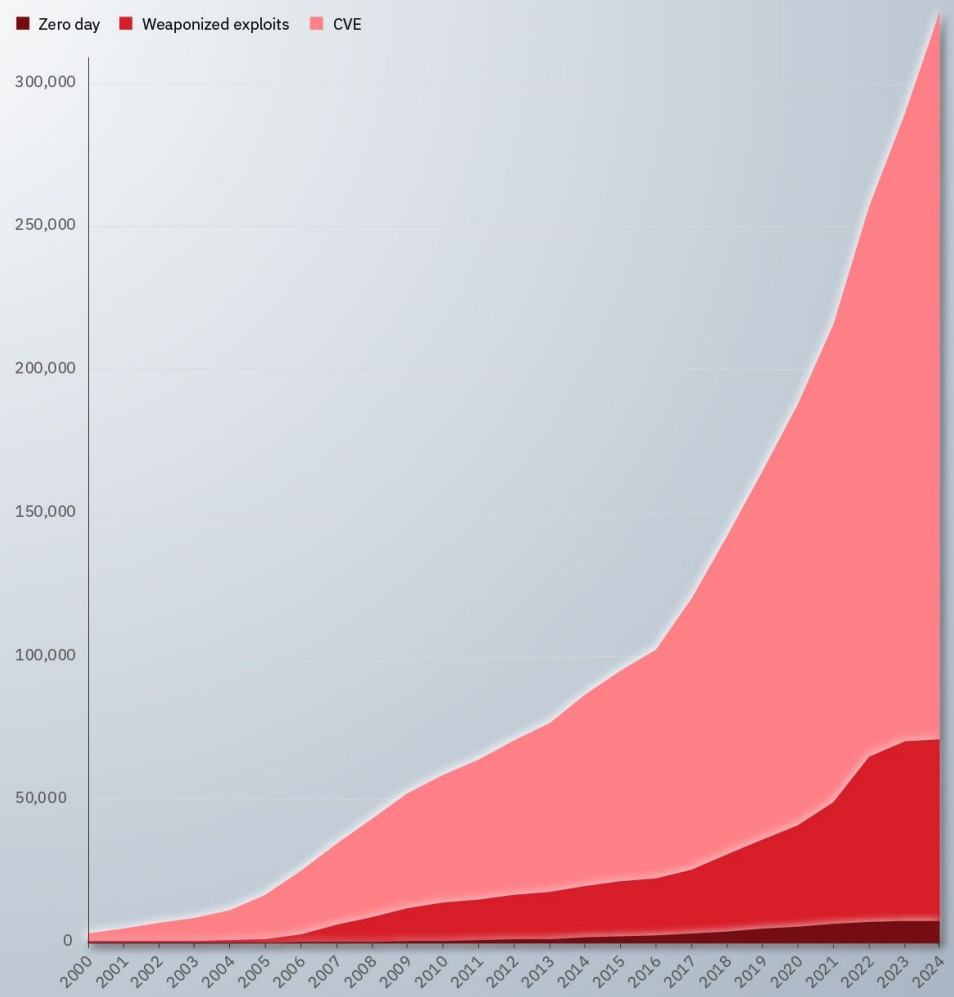
- Introduzione alla sicurezza nelle applicazioni
- Glossario e tassonomie
- Tassonomie CWE e CAPEC per weakness e schemi di attacco
- Intermezzo: la struttura delle applicazioni web
- Schemi d’attacco basati su injection:
  - Schema d’attacco XSS
  - Schema d’attacco SQL injection

Introduzione alla sicurezza nelle applicazioni

Sicurezza delle applicazioni

- Scopo principale delle applicazioni è eseguire delle funzionalità per determinate categorie di utenti utilizzando delle risorse messe loro a disposizione
- Non è pertanto strano che le applicazioni siano spesso attaccate, allo scopo di negare tali funzionalità, o di farle eseguire per conto di utenti non autorizzati, o di accedere a risorse per le quali non si hanno diritti
- Oggi le applicazioni sono sempre più esposte ad attacchi, dal momento che sono connesse alla rete globale, e quindi molto più facilmente accessibili


2023
2024
(T1078)
(T1190) 30%
(T1566) 29%
(T1133) 9%
(T1189) 4%
(T1091) 3%
30% Exploit public-facing application (T1190)
Valid accounts (T1078)
25% Phishing (T1566)
11% External remote services (T1133)
2% Drive-by compromise (T1189)
Replication through removable media (T1091)
IBM Institute for Business Value



IBM Institute for Business Value

Glossario e tassonomie

Il glossario della sicurezza

- Il glossario della sicurezza informatica è complesso
- Noi facciamo riferimento a:
  - **MITRE CWE glossary** (https://cwe.mitre.org/documents/glossary/)
  - **RFC-4949, Internet Security Glossary, Version 2** (https://tools.ietf.org/html/rfc4949)
  - **ISO 27000**

Glossario (1)

- **Security policy**: Un insieme di regole che determinano come un sistema o un'organizzazione protegge risorse critiche e sensibili
- **Weakness**: un tipo di errore nel progetto, implementazione o operazione di un sistema software che, in condizioni appropriate, può portare all'introduzione di vulnerabilità nel sistema
- **Vulnerabilità**: l'occorrenza di una weakness (o più di una) in un determinato sistema software, che può essere sfruttata da una terza parte per violare la security policy del sistema
- **Exploit**: una tecnica che permette di sfruttare una vulnerabilità per violare la security policy di un sistema

Glossario (2)

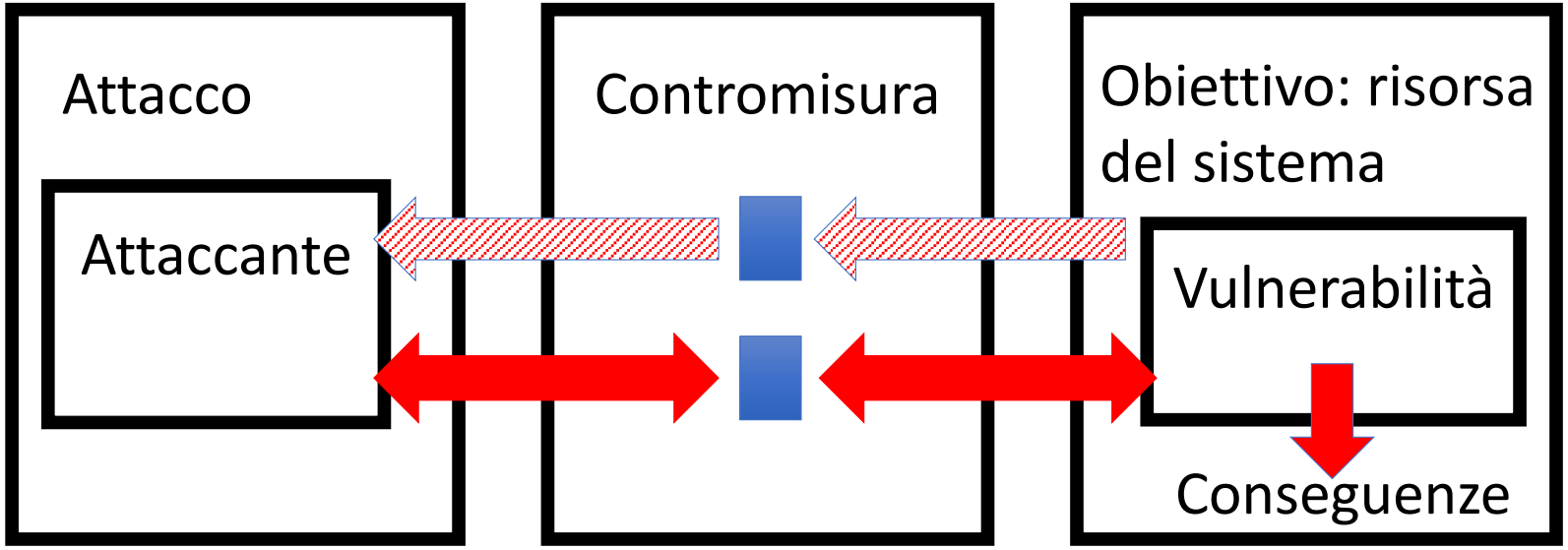
- **Attacco (attack)**: un’azione o una tipologia **(schema di attacco, attack pattern)** di azione intenzionale volta alla violazione della security policy di un sistema
- **Attaccante (attacker)**: un’entità che effettua un attacco
- **Contromisura (countermeasure)**: un’azione o tecnica che elimina, prevenisce, mitiga o individua una vulnerabilità, minaccia o attacco

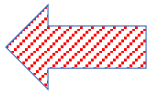
Caratteristiche attacco

- Tipologia
  - Attivo: altera le risorse del sistema e le sue operazioni
  - Passivo: acquisisce informazioni dal sistema
- Point of initiation:
  - Esterno: l'attaccante risiede al di fuori del perimetro di sicurezza del sistema
  - Interno: l'attaccante ha accesso alle risorse del sistema ma le vuole usare in modo differente rispetto a quanto concesso dalla security policy (ad es. scrivere un file)
- Method of delivery:
  - Diretto: attacca il sistema target direttamente
  - Indiretto: utilizza un terzo sistema per accedere al target o per amplificare l'attacco

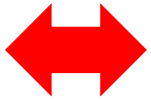
Relazioni tra concetti

Fonte: RFC-4949





Flusso delle informazioni durante un attacco passivo (lette)



Flusso delle informazioni durante un attacco attivo (lette/modificate)

Tassonomie e cataloghi (1)

- Diverse organizzazioni si occupano di collezionare e riportare weakness, schemi di attacco, vulnerabilità di software noti
- Le più note sono:
  - MITRE: organizzazione finanziata dal dipartimento della difesa americano
  - NIST: organizzazione finanziata dal dipartimento del commercio americano
  - Web Application Security Consortium (WASC): non legata ad organizzazioni governative
  - Open Web Application Security Project (OWASP): fondazione nonprofit sostenuta da diverse aziende

Tassonomie e cataloghi (2)

- MITRE mantiene:
  - Common Weakness Enumeration (CWE): tassonomia weaknesses
  - Common Attack Pattern Enumeration and Classification (CAPEC): tassonomia schemi d'attacco
  - Common Vulnerability and Exposure (CVE): catalogo vulnerabilità
  - ATT&CK: knowledge base di tattiche e tecniche avversarie
- NIST mantiene:
  - National Vulnerability Database (NVD): catalogo vulnerabilità
- WASC mantiene:
  - WASC threat classification: tassonomia schemi di attacco e weaknesses
- OWASP mantiene:
  - OWASP top ten: classificazione per gravità delle dieci più important weaknesses (chiamate "security risks", ma chiaramente correlate alle CWE)
- Analizzeremo quelle del MITRE (CWE, CAPEC e CVE)

Common Weakness Enumeration (CWE)

- Elenco formale di weakness
- Destinatari: sviluppatori, esperti di sicurezza
- Scopi:
  - Fornire un dizionario comune per descrivere le weakness nel software
  - Utile per misurare le potenzialità degli strumenti software di analisi/verifica
- https://cwe.mitre.org

Esempio di weakness riportata in CWE (1)

CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

Weakness ID: 77
Abstraction: Class
Structure: Simple

Presentation Filter: Complete

Description

The software constructs all or part of a command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.

Extended Description

Command injection vulnerabilities typically occur when:

1. Data enters the application from an untrusted source.
2. The data is part of a string that is executed as a command by the application.
3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Many protocols and products have their own custom command language. While OS or shell command strings are frequently discovered and targeted, developers may not realize that these other command languages might also be vulnerable to attacks.

Command injection is a common problem with wrapper programs.

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	C	74	Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
ParentOf	B	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
ParentOf	B	88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
ParentOf	B	624	Executable Regular Expression Error

Esempio di weakness riportata in CWE (2)

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality Availability	**Technical Impact:** *Execute Unauthorized Code or Commands* If a malicious user injects a character (such as a semi-colon) that delimits the end of one command and the beginning of another, it may be possible to then insert an entirely new and unrelated command that was not intended to be executed.	

Likelihood Of Exploit

High

Demonstrative Examples

Observed Examples

Reference	Description
CVE-1999-0067	Canonical example of OS command injection. CGI program does not neutralize "|" metacharacter when invoking a phonebook program.
CVE-2019-12921	image program allows injection of commands in "Magick Vector Graphics (MVG)" language.
CVE-2020-11698	anti-spam product allows injection of SNMP commands into confiuration file

Potential Mitigations

Weakness Ordinalities

Ordinality	Description
Primary	*(where the weakness exists independent of other weaknesses)*

Memberships

Notes

Taxonomy Mappings

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-136	LDAP Injection
CAPEC-15	Command Delimiters
CAPEC-183	IMAP/SMTP Command Injection
CAPEC-248	Command Injection
CAPEC-40	Manipulating Writeable Terminal Devices
CAPEC-43	Exploiting Multiple Input Interpretation Layers
CAPEC-75	Manipulating Writeable Configuration Files

Common Attack Pattern Enumeration and Classification (CAPEC)

- Catalogo di schemi di attacco (attack patterns)
- Notare la differenza:
  - CWE descrive le weakness che possono essere presenti in un software
  - CAPEC descrive come queste sono tipicamente sfruttate
- https://capec.mitre.org/

Esempio di schema d’attacco in CAPEC

CAPEC-248: Command Injection

Attack Pattern ID: 248
Abstraction: Meta

Status: Stable

Presentation Filter: Basic

Description

An adversary looking to execute a command of their choosing, injects new items into an existing command thus modifying interpretation away from what was intended. Commands in this context are often standalone strings that are interpreted by a downstream component and cause specific responses. This type of attack is possible when untrusted values are used to build these command strings. Weaknesses in input validation or command construction can enable the attack and lead to successful exploitation.

Relationships

Nature	Type	ID	Name
ParentOf	S	40	Manipulating Writeable Terminal Devices
ParentOf	S	66	SQL Injection
ParentOf	S	88	OS Command Injection
ParentOf	S	136	LDAP Injection
ParentOf	S	183	IMAP/SMTP Command Injection
ParentOf	S	250	XML Injection
ParentOf	S	676	NoSQL Injection

View Name	Top Level Categories
Domains of Attack	Software
Mechanisms of Attack	Inject Unexpected Items

Prerequisites

The target application must accept input from the user and then use this input in the construction of commands to be executed. In virtually all cases, this is some form of string input that is concatenated to a constant string defined by the application to form the full command to be executed.

Mitigations

All user-controllable input should be validated and filtered for potentially unwanted characters. Using an allowlist for input is desired, but if use of a denylist approach is necessary, then focusing on command related terms and delimiters is necessary.

Input should be encoded prior to use in commands to make sure command related characters are not treated as part of the command. For example, quotation characters may need to be encoded so that the application does not treat the quotation as a delimiter.

Input should be parameterized, or restricted to data sections of a command, thus removing the chance that the input will be treated as part of the command itself.

Related Weaknesses

CWE-ID	Weakness Name
77	Improper Neutralization of Special Elements used in a Command ('Command Injection')

Common Vulnerability and Exposures (CVE)

- Catalogo di vulnerabilità che riguardano software molto diffusi
- Probabilmente è il più noto e il più consultato in pratica
- https://www.cve.org/

Esempio di vulnerabilità riportata in CVE

CVE-ID	
CVE-1999-0067	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

phf CGI program allows remote command execution through shell metacharacters.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- AUSCERT:AA-96.01
- BID:629
- URL:http://www.securityfocus.com/bid/629
- BUGTRAQ:19960923 PHF Attacks - Fun and games for the whole family
- CERT:CA-1996-06
- URL:http://www.cert.org/advisories/CA-1996-06.html
- OSVDB:136
- URL:http://www.osvdb.org/136
- XF:http-cgi-phf

Date Record Created	
19990929	Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

This is a record on the CVE List, which provides common identifiers for publicly known cybersecurity vulnerabilities.

SEARCH CVE USING KEYWORDS: [ ] Submit

You can also search by reference using the CVE Reference Maps.

For More Information: CVE Request Web Form (select "Other" from dropdown)

Tassonomie CWE e CAPEC per weakness e schemi di attacco

Tassonomie CWE e CAPEC

- Vedremo in maggior dettaglio le classificazioni date da CWE e CAPEC per weakness e schemi di attacco
- Come avete visto dagli esempi, esse sono complementari:
  - Studiare le weakness aiuta a capire i tipici difetti nel codice che impattano sulla sicurezza
  - Studiare gli schemi di attacco aiuta a capire come tali errori possono essere sfruttati per violare le politiche di sicurezza del sistema
  - L'ingegnere del software studia le weakness per comprendere quali difetti evitare nello sviluppo
  - L'esperto di sicurezza studia gli schemi d'attacco per comprendere come individuare e sfruttare le weakness nei sistemi sotto esame
- Lo studio delle tassonomie è utile alla comprensione degli argomenti di questa parte del corso, ma le tassonomie non sono oggetto di esame

Weakness, research concept view (1)

- **CWE-284: Improper access control:** il software non restringe o restringe in maniera scorretta l'accesso ad una risorsa da parte di un attore non autorizzato
- **CWE-435: Improper interaction between multiple correctly-behaving entities**: entità con un comportamento corretto quando isolate, producono comportamenti non corretti quando interagiscono
- **CWE-664: Improper control of a resource through its lifetime**: il software non segue i protocolli corretti di creazione, uso e distruzione di risorse
- **CWE-682: Incorrect calculation**: il software effettua un calcolo che genera risultati scorretti o inattesi
- **CWE-691: Insufficient control flow management**: il software crea condizioni in cui il flusso di controllo può essere modificato in maniera inattesa
- **CWE-693: Protection mechanism failure**: il software non usa o usa in maniera scorretta dei meccanismi di protezione che fornirebbero una difesa sufficiente

Weakness, research concept view (2)

- **CWE-697: Incorrect comparison**: il software confronta due entità in un contesto rilevante per la sicurezza, ma il confronto è scorretto
- **CWE-703: Improper check or handling of exceptional conditions**: il software non individua o gestisce condizioni eccezionali che possono verificarsi durante le normali operazioni
- **CWE-707: Improper neutralization**: Il prodotto non assicura o assicura in maniera scorretta che i messaggi o i dati strutturati siano ben formati e che certe proprietà di sicurezza siano garantite prima che vengano letti da un componente a monte o inviati ad un componente a valle
- **CWE-710: Improper adherence to coding standards**: il software non segue certe regole di sviluppo software che garantirebbero l'assenza o mitigherebbero gli effetti di un certo tipo di vulnerabilità

Esempi di weakness per categoria

- Improper Control of a Resource Through its Lifetime
  CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')
- Incorrect calculation
  CWE-190: Integer Overflow or Wraparound
- Improper Check or Handling of Exceptional Conditions
  CWE-252: Unchecked Return Value
- Improper adherence to coding standards
  CWE-476: NULL Pointer Dereference

Schemi d’attacco, by mechanism of attack (1)

- **CAPEC-118: Collect and Analyze Information**: attacchi basati sulla raccolta/furto di informazioni
- **CAPEC-152: Inject Unexpected Items**: attacchi basati sull’invio di dati inattesi attraverso un’interfaccia di input, o sulla installazione di codice nel sistema
- **CAPEC-156: Engage in Deceptive Interactions**: attacchi basati su interazioni ingannevoli in cui l’obiettivo è convincere il target che sta interagendo con qualcun altro
- **CAPEC-172: Manipulate Timing and State**: attacchi in cui, manipolando lo stato del sistema e/o il timing delle sue operazioni, viene modificato il flusso di controllo del sistema
- **CAPEC-210: Abuse Existing Functionality**: attacchi basati sull’uso o manipolazione delle funzionalità del sistema per ottenere un obiettivo non previsto

Schemi d’attacco, by mechanism of attack (2)

- **CAPEC-223: Employ Probabilistic Techniques**: attacchi basati su tecniche probabilistiche per aggirare le proprietà di sicurezza del sistema garantite probabilisticamente
- **CAPEC-225: Subvert Access Control**: attacchi che sfruttano debolezze e assunzioni nei meccanismi che il target usa per gestire le identità e l’autenticazione
- **CAPEC-255: Manipulate Data Structures**: attacchi che sfruttano caratteristiche delle strutture dati del sistema per violarne l’uso inteso e la protezione
- **CAPEC-262: Manipulate System Resources**: attacchi basati sulla manipolazione di risorse quali file, applicazioni, librerie, infrastrutture e configurazioni

Esempi di schemi d’attacco per categoria

- Inject Unexpected Items
  CAPEC-63: Cross-site Scripting (XSS)
  CAPEC-66: SQL Injection
- Manipulate timing and state
  CAPEC-29: Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions
- Manipulate data structures
  CAPEC-92: Forced Integer Overflow
  CAPEC-100: Overflow Buffers

Problemi delle tassonomie CWE e CAPEC

- Non sono ortogonali: alcune categorie potrebbero sovrapporsi
  - CWE: Improper Access Control e Protection Mechanism Failure
  - CAPEC: Inject Unexpected Items e Manipulate data structures
- Sono imprecise:
  - CWE Improper Adherence to Coding Standards non chiarisce quali «coding standard» andrebbero considerati
  - CWE 'Type Confusion' non chiarisce quali usi (ad esempio delle union in C) costituiscono weakness e quali no
- Confondono «mistakes» (o sbagli, azioni umane che producono un risultato scorretto) con «faults» (o difetti, conseguenze degli sbagli nella produzione del software)
  - CWE NULL Pointer Dereference è un fault, CWE Improper Adherence to Coding Standards è un mistake

Intermezzo: la struttura delle applicazioni web

La struttura delle applicazioni web

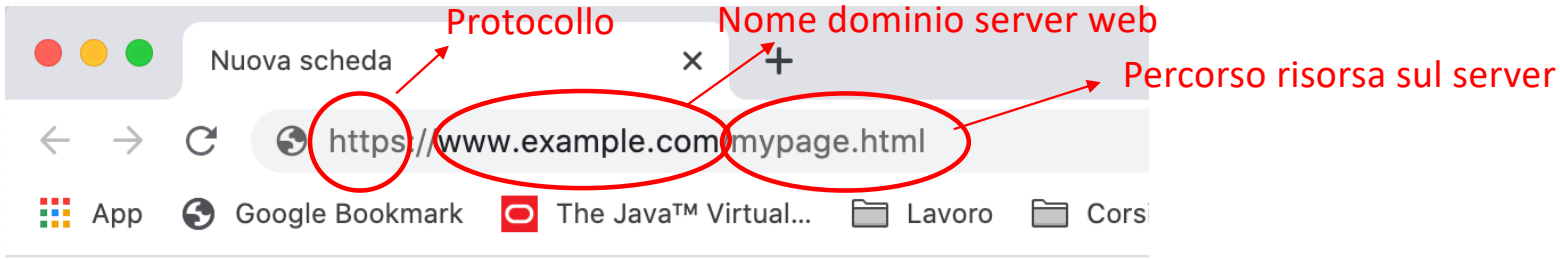
- Le slides che seguono non sono materiale di esame
- Servono per capire gli schemi di attacco di tipo XSS
- Essi infatti sono tipici delle applicazioni web, e quindi ci è utile riassumere come tali applicazioni sono strutturate

Il world wide web

- Il world wide web è basato su un'architettura client-server
  - Client: browser (Chrome, Firefox, Safari…)
  - Server: web server (Apache, NGINX, Tomcat…)
- Client e server comunicano attraverso il protocollo HTTP (su TCP)
- Funzionamento tipico:
  - L'utente inserisce un URL nel browser; l'URL identifica una risorsa in rete (di solito una pagina di testo codificata in linguaggio HTML)
  - Il browser si connette al server e gli chiede la risorsa utilizzando un opportuno comando HTTP
  - Il server invia la risorsa al browser, che inizia a visualizzarla
  - Se la risorsa contiene riferimenti (URL) ad altre risorse (immagini, video, audio…) il browser chiede le ulteriori risorse ai server che le ospitano

Gli URL

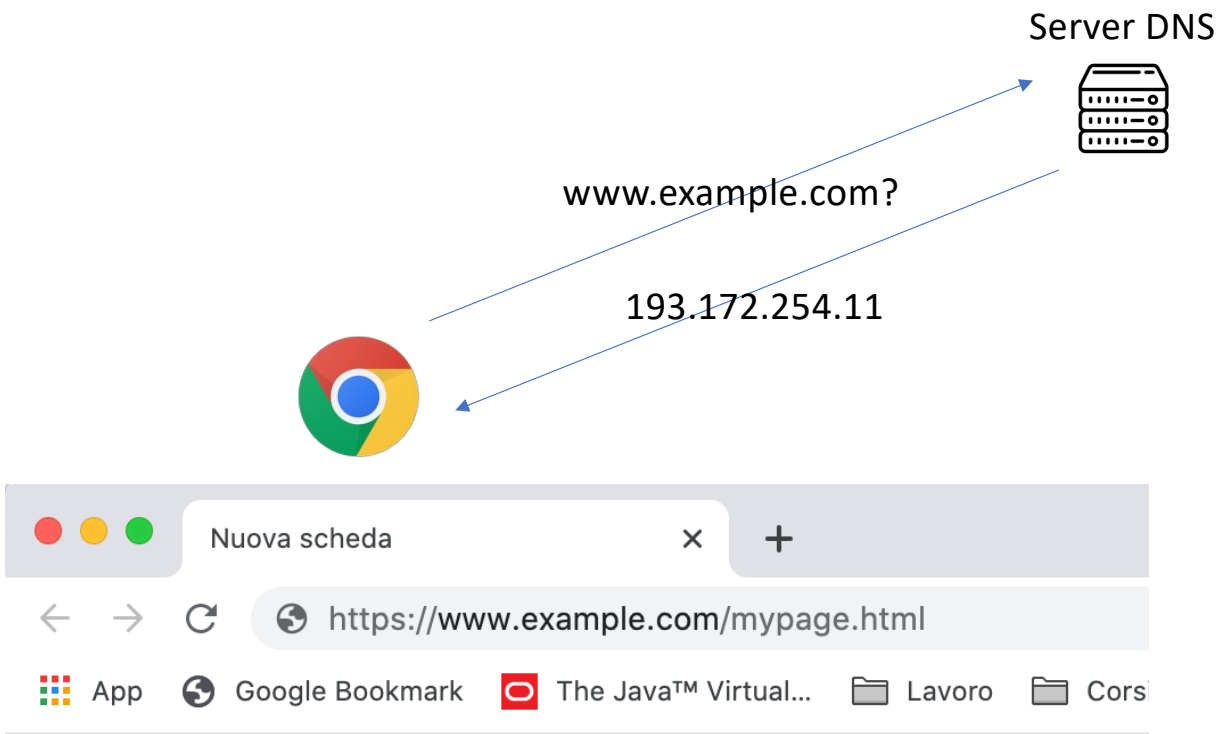
- Il browser identifica le risorse in rete attraverso gli URL
- Un URL è composto da:
  - Protocollo per comunicare con il server web che ospita la risorsa (normalmente HTTP o HTTPS)
  - Nome di dominio o indirizzo IP del server web che ospita la risorsa
  - Percorso della risorsa sul server
- Se nell'URL il server è indicato con il nome di dominio, il browser deve ricavare l'indirizzo IP corrispondente rivolgendosi ad un server DNS



Il protocollo HTTP

- È un protocollo basato su scambi testuali
- Nella sua versione più semplice (HTTP 1.1):
  - Il browser apre una connessione TCP
  - Quindi invia una richiesta HTTP al server
  - Il server risponde e chiude la connessione TCP
- Richieste HTTP essenziali:
  - GET: ottenere una risorsa dal server
  - POST: inviare una risorsa al server
- Svantaggio HTTP 1.1: occorre aprire una connessione TCP distinta per ogni risorsa che si vuole ottenere

Esempio di funzionamento



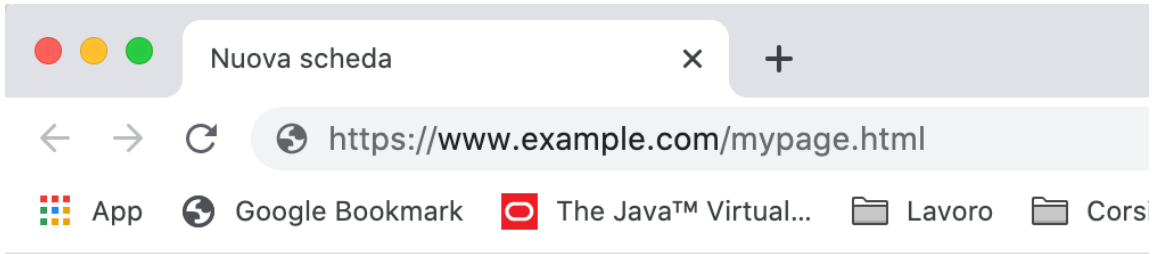
Esempio di funzionamento

Nome risorsa

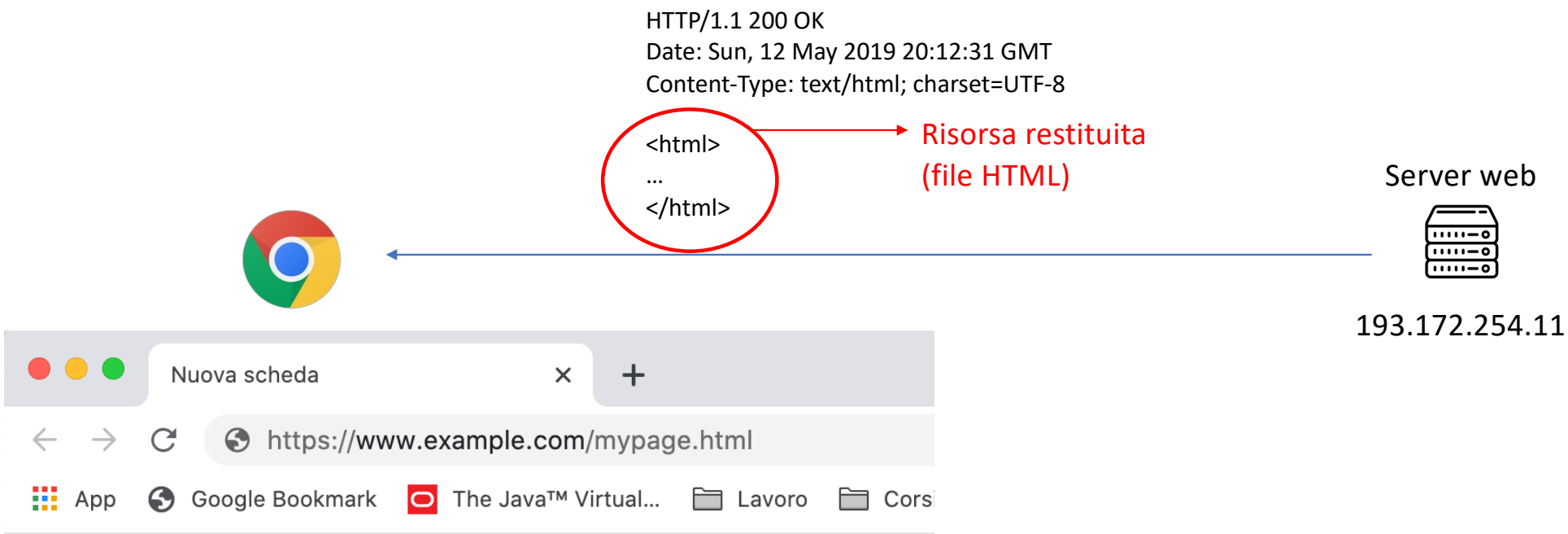
GET /mypage.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_4)
Accept: text/html, image/jpeg, image/png
Accept-Charset: utf-8

Server web

193.172.254.11



Esempio di funzionamento



Pagine web

- Sono scritte in un insieme di diversi linguaggi:
  - HTML: linguaggio di descrizione di pagina
  - CSS: linguaggio di fogli di stile
  - Javascript: linguaggio di programmazione general-purpose
- HTML è usato per descrivere il contenuto della pagina (testo e struttura)
- CSS è usato per descrivere come la pagina deve apparire (font, disposizione del testo, colori…)
- Javascript permette di far si che la pagina abbia un comportamento dinamico, e quindi di realizzare vere e proprie applicazioni

HTML (1)

- HTML è un linguaggio di markup
- Per markup si intende l’utilizzo di informazioni aggiuntive al testo per la descrizione del documento
- HTML fa uso di markup dichiarativo basato sull’utilizzo di tag
- I tag hanno nomi predefiniti racchiusi tra < … >
- Hanno un’apertura <tag> e una chiusura </tag>
- Il corpo di un tag (cioè la parte racchiusa tra l’apertura e la chiusura) può contenerne altri

HTML (2)

- Il tag <head> indica l’intestazione
  - Di solito contiene tag che non vengono visualizzati ma servono per gestire il layout o contengono meta-informazione
  - Ad esempio, il tag <title> contiene il titolo del documento
- Il tag <body> indica il corpo della pagina
  - Contiene il testo della pagina
  - I tag e <p> servono per strutturare la pagina in sezioni e paragrafi
  - I tag <a> («anchor») servono per creare link ad altre pagine
- I tag possono avere attributi con valore
  - Ad esempio, il tag <a> ha l’attributo href, il cui valore indica l’URL della pagina alla quale il link deve far riferimento
  - Esempio: <a href="http://www.example.com/...">text</a>

HTML: esempio

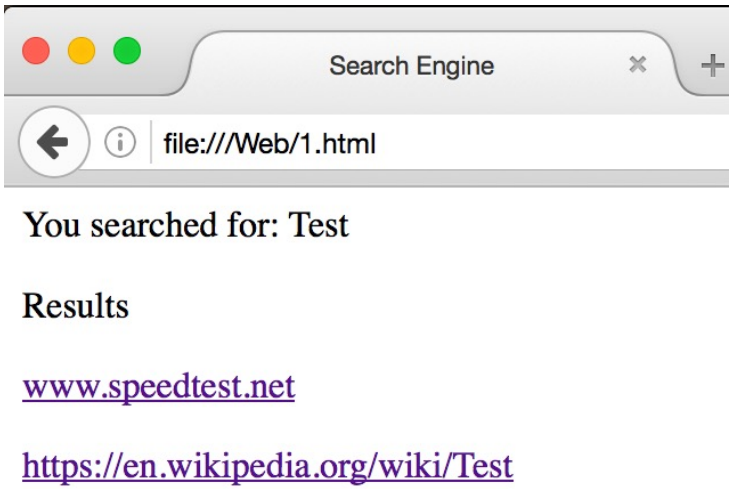
```
<html>
<head>
<title>Search Engine</title>
</head>
<body>
<div>You searched for: Test</div>
<div><p>Results</p>
<p><a href="http://www.speetest.net">www.speedtest.net</a></p>
<p><a href="https://en.wikipedia.org/wiki/Test">https://en.wikipedia.org/wiki/Test</a></p>
</div>
</body>
</html>
```

TAG (apertura)

TAG (chiusura)

ATTRIBUTE NAME (href in questo caso)

ATTRIBUTE VALUE

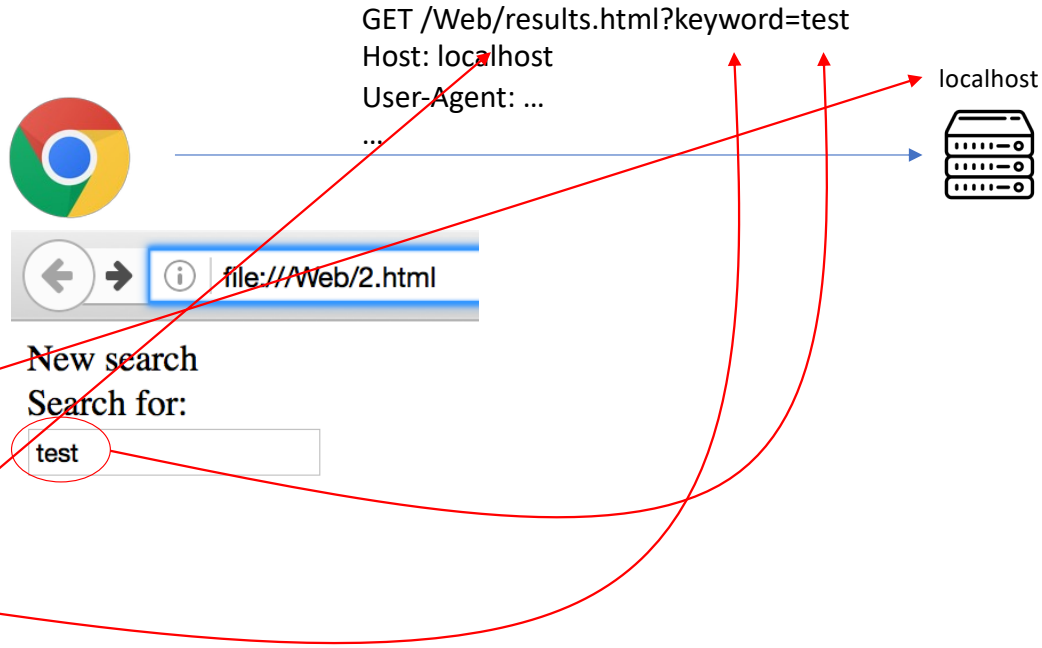


Generazione dinamica di pagine server-side

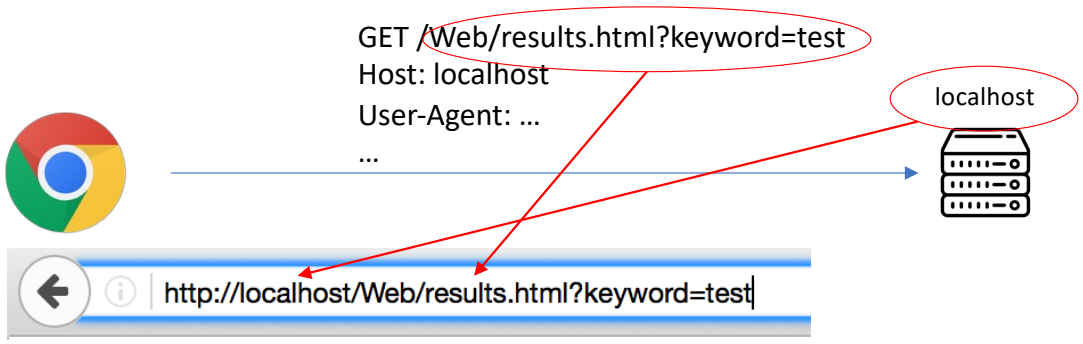
- È possibile che una certa risorsa non esista nel server, ma che sia il server a generarla dinamicamente una volta che gli viene chiesta
- Di solito questo avviene quando il contenuto della risorsa deve dipendere da un certo insieme di parametri che il client fornisce al server
- Il tipico esempio è quello dei motori di ricerca:
  - Il browser fornisce al server una keyword di ricerca
  - Il server effettua una ricerca su un certo database basata sulla keyword
  - Il server costruisce dinamicamente una pagina HTML contenente il risultato della ricerca, e la restituisce al client
- Due domande:
  - Come fa il client a trasmettere al server i parametri?
  - Come fa il server a costruire dinamicamente la pagina?

I form HTML

```
<html>
<head>
<title>Search Engine</title>
</head>
<body>
<div>New search</div>
<div>
 <form method="get" action="http://localhost/Web/results.html">
 Search for:<br/>
 <input type="text" name="keyword">
 </form>
</div>
</body>
</html>
```

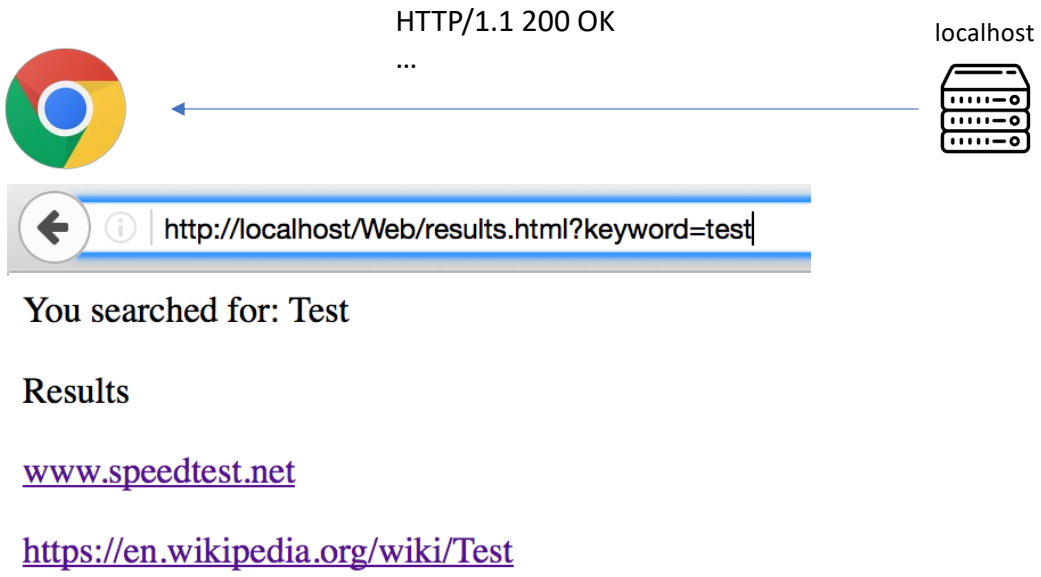


I form HTML


GET /Web/results.html?keyword=test
Host: localhost
User-Agent: …
…
localhost
http://localhost/Web/results.html?keyword=test

I form HTML

```
<html>
<head>
<title>Search Engine</title>
</head>
<body>
<div>You searched for: Test</div>
<div><p>Results</p>
<p><a href="http://www.speetest.net">www.speedtest.net</a></p>
<p><a
href="https://en.wikipedia.org/wiki/Test">https://en.wikipedia.org/
wiki/Test</a></p>
</div>
</body>
</html>
```



Invio richieste dal form al server

- Nella slide precedente abbiamo visto il metodo GET per inviare una richiesta al server:
  - La risorsa è una pagina fittizia di cui viene fatta una richiesta HTTP GET
  - Nell'URL dopo il nome risorsa e il carattere ? vengono aggiunti i contenuti degli input nel form
- Metodo alternativo: metodo POST
  - La risorsa è sempre una pagina fittizia, ma il contenuto dei vari input non è nell'URL
  - Stavolta la richiesta è HTTP POST
  - Il contenuto degli input nel form è passato nel message body della richiesta POST
- In entrambi i casi il server deve genera dinamicamente una pagina HTML con il contenuto appropriato e ritornarla al client

Scripting server-side

- La generazione dinamica della pagina è effettuata dal server eseguendo un opportuno programma al quale vengono passata gli input del form
- Oggi esistono appositi linguaggi di scripting server-side per la generazione dinamica delle pagine:
  - PHP
  - ASP.NET
- Ma si possono anche usare linguaggi general-purpose, di solito affiancati a framework appositi:
  - Java
  - Ruby
  - Python
  - …

Esempio di scripting server-side in PHP

GET /results.html?keyword=test
Host: …
User-Agent: …
…

```
<?php
print "<html>";
print "<head>";
print "<title>Search engine</title>";
print "</head>";
print "<body>";
$querystr = $_SERVER['QUERY_STRING'];
parse_str($querystr, $query);
$keyword = $query['keyword'];
print "You searched for: ";
print $keyword;
print "</body>";
…
print "</html>";
?>
```

Esempio di scripting server-side in PHP

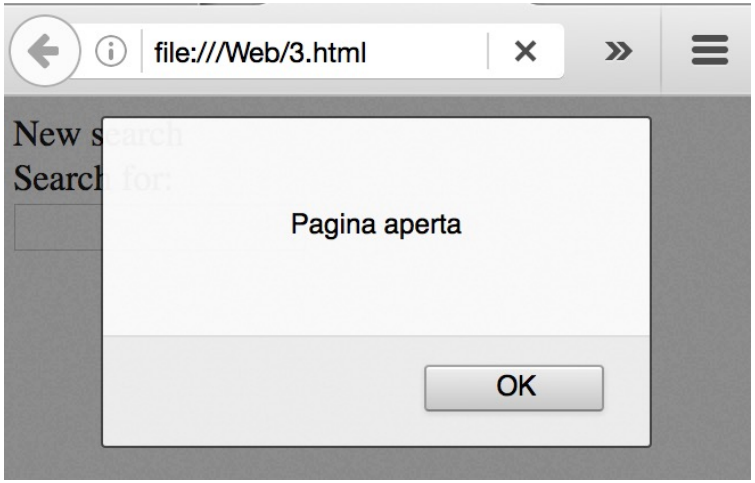
```
<html>
…
<body>
You searched for: test
…
</html>
```

```
<?php
print "<html>";
print "<head>";
print "<title>Search engine</title>";
print "</head>";
print "<body>";
$querystr = $_SERVER['QUERY_STRING'];
parse_str($querystr, $query);
$keyword = $query['keyword'];
print "You searched for: ";
print $keyword;
print "</body>";
…
print "</html>";
?>
```

Scripting client-side

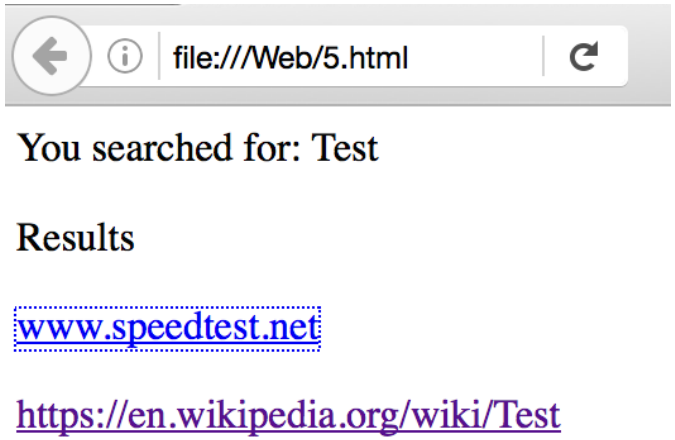
- Il linguaggio di scripting interpretato da tutti i browser è Javascript
- Attraverso tale linguaggio si possono programmare delle operazioni del browser tra cui:
  - Salvare o accedere ad alcune informazioni localmente nel browser (cookies)
  - Osservare o modificare lo stato del documento HTML (attraverso il DOM)
  - Inviare richieste HTTP tramite chiamate a XMLHttpRequest o altri meccanismi
- In tal modo la pagina HTML può comportarsi come una vera e propria applicazione
- Il codice Javascript viene incluso nella pagina web:
  - Tra tag <script>…</script>
  - Come valori di attributi appositi che specificano la reazione di certi elementi HTML a determinati eventi (es. onclick)
  - Anche in valori di attributi come href, se usiamo lo pseudoprotocollo javascript:

```
<html>
<head>
<title>Search Engine</title>
</head>
<body>
<div>New search</div>
<div>
 <form method="get" action="results.html">
 Search for:<br/>
 <input type="text" name="keyword">
 </form>
<script>
alert("Pagina aperta");
</script>
</div>
</body>
</html>
```

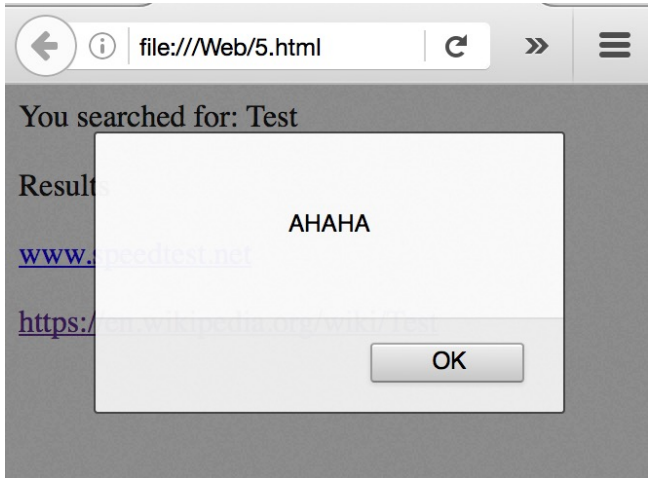



```
<html>
<head>
<title>Search Engine</title>
</head>
<body>
<div>New search</div>
<script>
function myFunction() {
 document.getElementById("inputK").style.backgroundColor = "red";
}
</script>
<div>
 <form method="get" action="2.html">
 Search for:<br/>
 <input id="inputK" type="text" name="keyword" onclick="myFunction()">
 </form>
</div>
</body>
</html>
```

```
<html>
<head>
<title>Search Engine</title>
</head>
<body>
<div>You searched for: Test</div>
<div><p>Results</p>
<p><a href="javascript:alert('AHAHA')">www.speedtest.net</a></p>
<p><a href="https://en.wikipedia.org/wiki/Test">https://en.wikipedia.org/wiki/Test</a></p>
</div>
</body>
</html>
```



Dopo click su speedtest.net



Schemi d’attacco basati su injection

Secondo la tassonomia…

- CWE-707: Improper neutralization
  - CWE-74: Improper neutralization of special elements in output used by a downstream component («injection»)
    - CWE-79: Improper neutralization of input during web page generation («cross-site scripting»)
    - CWE-943: Improper neutralization of special elements in data query logic
      - CWE-89: Improper neutralization of special elements used in an SQL command («SQL injection»)
- CAPEC-152: Inject unexpected items
  - CAPEC-242: Code injection
    - **CAPEC-63: Cross-site scripting (XSS)**
  - CAPEC-248: Command injection
    - **CAPEC-66: SQL injection**

La weakness di neutralizzazione impropria

- Avviene quando manca un controllo sugli input passati dall'utente
- Esempio: pagina web che richiede l'inserimento di parametri
- Cosa succede se inserisco un valore sbagliato, ad esempio, nel campo "data di nascita"?
  - In generale non sappiamo cosa può succedere, perché non sappiamo fino a dove tale valore penetra nell'applicazione
  - Se penetra in certi contesti d'uso il valore potrebbe essere utilizzato in modi imprevisti, e quindi il valore errato va "neutralizzato" prima dell'uso

Una non soluzione

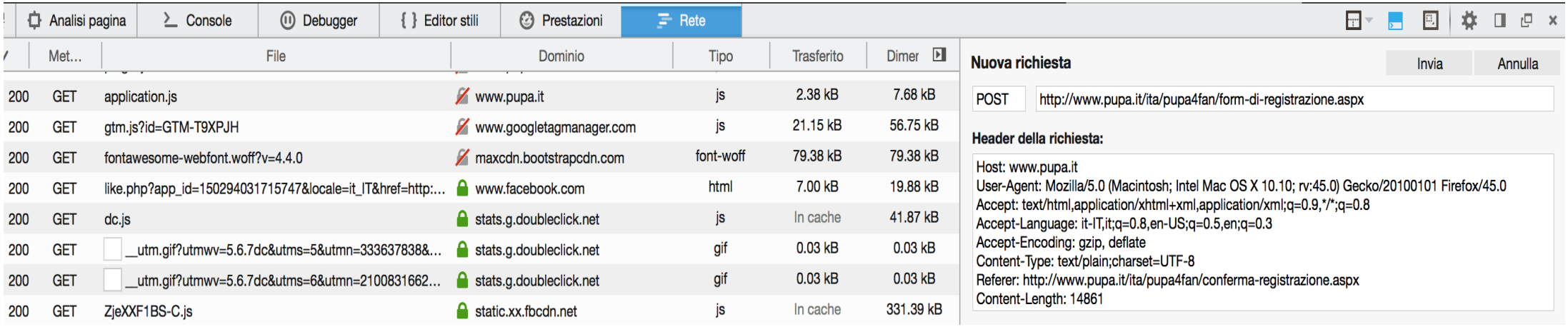
- Spesso viene aggiunto un controllo lato client (ad esempio nella pagina web stessa)
- Questo controllo è tuttavia insufficiente: un utente malevolo potrebbe inviare la richiesta direttamente al server web senza usare la pagina web, componendo un URL (o una richiesta HTTP POST)
- Vediamo ora un esempio

Esempio con richiesta di tipo GET

- Questo è un esempio di URL generata dal browser quando una form per ordinare prodotti da un sito web viene utilizzata da un utente.
- http://www.things.com/orders.asp?custID=112&itemId=20&qty=20&**price=200**
- Qui il problema è che il server si fa spedire il prezzo dal client, invece di calcolarlo lui
- Pertanto un attaccante potrebbe ottenere degli articoli gratuiti creando un URL che termina con …&**price=0**

Esempio con richiesta di tipo POST

- Lo stesso problema occorre anche nel caso di form che inviano richieste di tipo POST
- Sebbene il contenuto di una richiesta POST non si veda nella barra del browser, strumenti di web development permettono di creare richieste HTTP ad hoc (esempio: Firefox developer tools)



Soluzione

- Verificare la correttezza degli input utente sia lato *client* che lato *server*
  - Se applicazione client/server
  - Non solo applicazioni web!
- Necessario per prevenire diversi tipi di attacchi
  - XSS
  - SQL injection

Schema di attacco XSS

Schema di attacco Cross Site Scripting (XSS)

- È uno schema di attacco di tipo code injection
- Si verifica quando un attaccante riesce a eseguire del codice malevolo all’interno del browser di un utente sfruttando una vulnerabilità di un sito web ignaro dell’attacco
- Il codice malevolo è implementato in un linguaggio di scripting, ad esempio Javascript

Perché attaccare un browser (1)

- Uno script Javascript è eseguito in un ambiente protetto (browser) che non permette la modifica del sistema che lo ospita
- Tuttavia lo script, attraverso il browser:
  - Ha accesso ad informazioni sensibili, ad esempio i cookies
  - Può inviare richieste HTTP contenenti qualsiasi dato verso una qualsiasi destinazione
  - Può modificare l'HTML della pagina utilizzando funzionalità del DOM

Perché attaccare un browser (2)

- Conseguenze:
  - Furto di cookie
    - L’attaccante può accedere ai cookie tramite la variabile document.cookie
    - Inviarli al server
    - Usarli per estrarre dati sensibili, e.g. session IDs.
  - Keylogging
    - L’attaccante può registrare un keyboard event listener (tramite chiamata a addEventListener)
    - Quindi registrare ed inviare ad un proprio server tutti i caratteri premuti (potenzialmente password)
  - Phishing
    - L’attaccante può creare un fake login

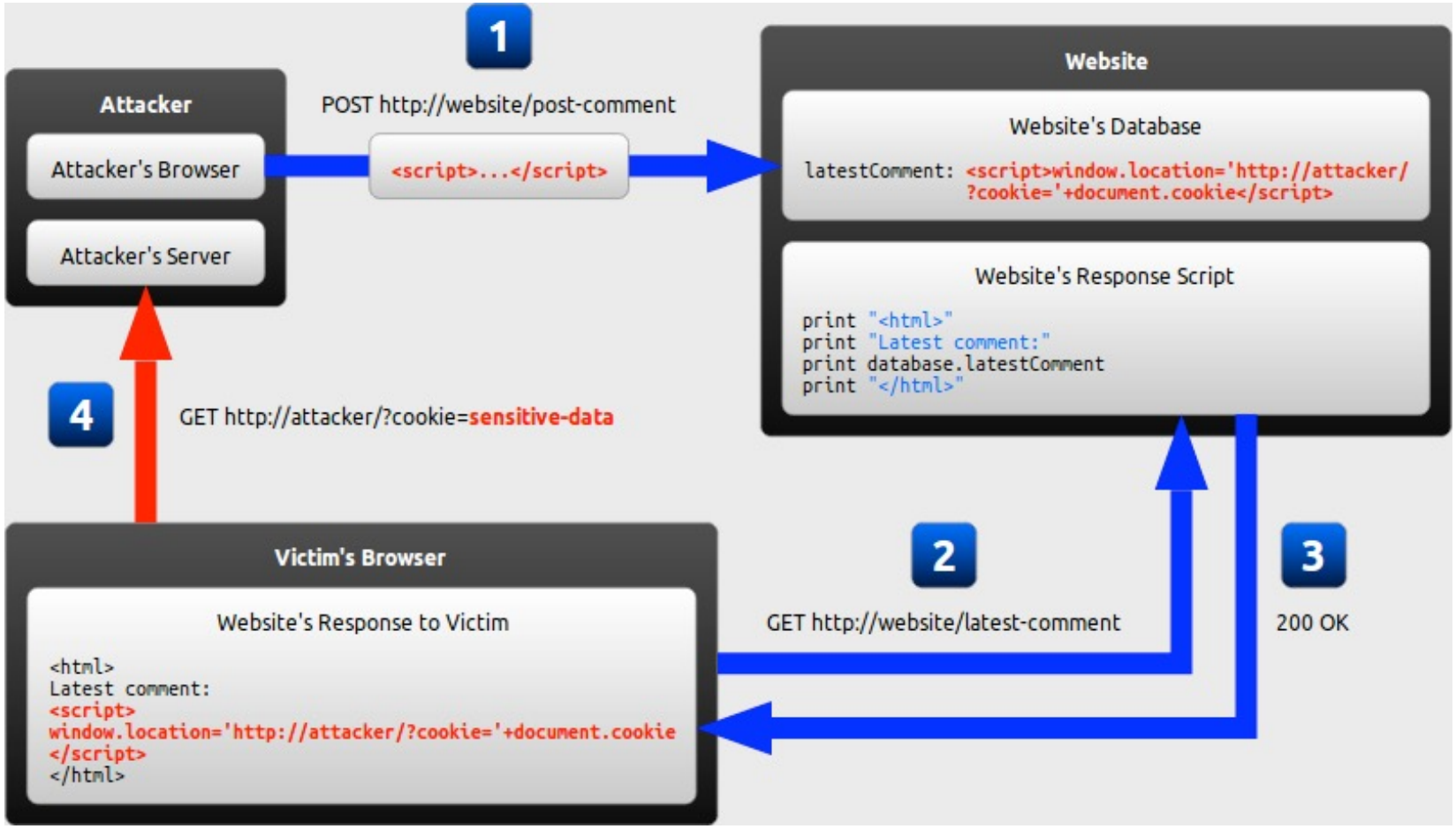
Le tipologie di attacchi XSS

- **Persistent XSS**
  - Il codice malevolo risiede nel DB del sito web
  - La vulnerabilità risiede nel codice di script eseguito dal server
- **Reflected XSS**
  - Il codice malevolo risiede nella richiesta HTTP della vittima
  - La vulnerabilità risiede nel codice di script eseguito dal server
- **DOM-based XSS**
  - Il codice malevolo risiede nella richiesta HTTP della vittima
  - La vulnerabilità risiede nel codice di script eseguito dal client
  - (quest'ultimo viene fornito dal server, quindi la vulnerabilità ha sempre origine nel server!)

Persistent XSS

- L’attaccante utilizza una form del sito web per inserire del codice malevolo nel database del sito
- La vittima richiede una pagina dal sito web
- Il sito web (ignaro) include nella pagina web il codice malevolo inserito dall’attaccante ed invia la risposta alla vittima
- Il browser della vittima esegue lo script malevolo contenuto nella pagina web

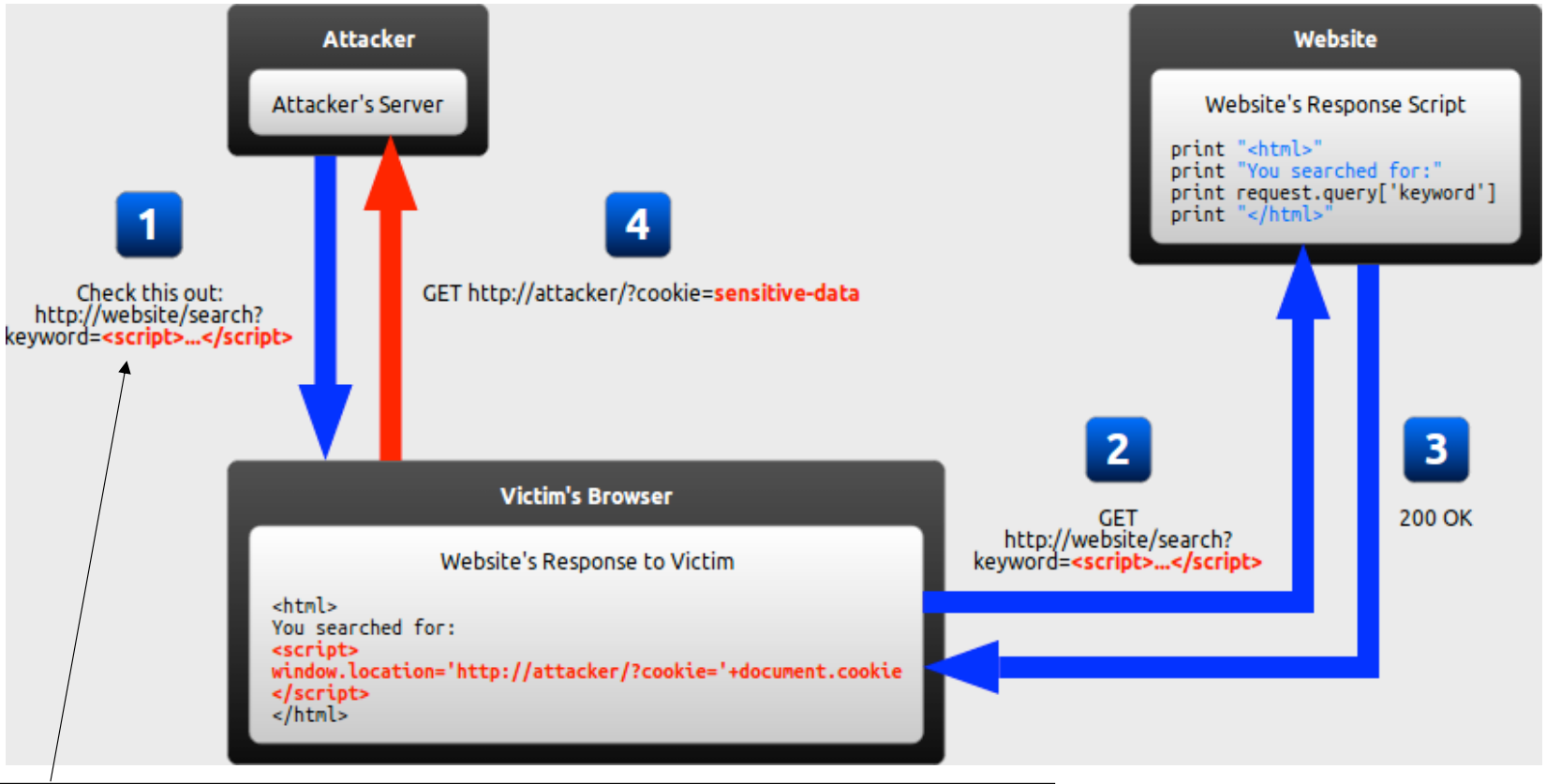
Persistent XSS: schema



Reflected XSS

- L’attaccante crea un URL contenente una stringa malevola e lo invia alla vittima, ad esempio in un messaggio email
- La vittima clicca sull’URL
- Il sito web invia una pagina di risposta che include la stringa malevola
  - Questo dipende dall’implementazione del sito web
  - Ad esempio il sito è un motore di ricerca che mostra la keyword di ricerca nella pagina dei risultati, e la stringa malevola è contenuta nella keyword
- Il browser della vittima esegue il codice malevolo

Reflected XSS: schema

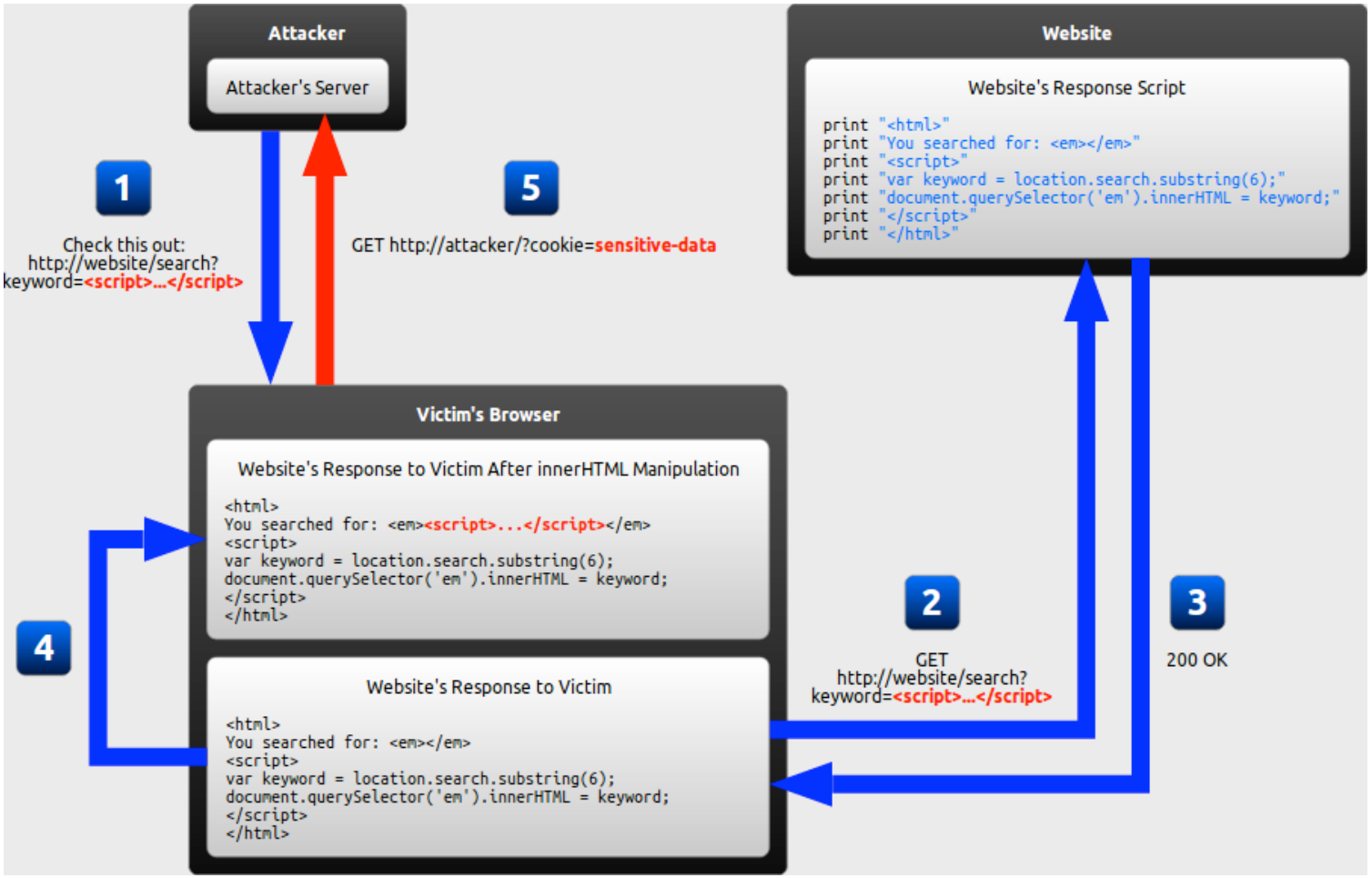


La vittima viene convinta a cliccare su un link tramite E-mail / SMS / Link da pagina web. Il link stesso contiene lo script malevolo.

DOM-based XSS

- Sfrutta lo scripting dinamico lato client
- Come nel reflected XSS l'attaccante crea un URL contenente una stringa malevola e lo invia alla vittima
- La vittima clicca sull'URL
- Il sito web riceve la richiesta, ed invia in risposta una pagina dinamica che non include il codice malevolo ma include codice Javascript vulnerabile
- Il codice Javascript incluso nella pagina:
  - Estrae il codice malevolo dall'URL
  - Lo aggiunge dinamicamente alla pagina
- Il browser dell'utente interpreta lo script contenente il codice malevolo

DOM-based XSS: schema



Contromisure

- Consistono principalmente nell’eseguire operazioni di purificazione dell’input utente
- Configurate in base a
  - Contesto
  - Momento del controllo (inbound/outbound)
  - Locazione del controllo (client side/server side)
- Metodi
  - Encoding (codifica)
  - Validation (convalida)

Il contesto impatta sulla soluzione

- Gli attacchi possono essere diversi a seconda del contesto:

Application code	<a href="**userInput**">
Malicious string	**http://foo.com"><script>...</script><div id="**
Resulting code	<a href="**http://foo.com"><script>...</script>

- In questo esempio la stringa di attacco sfrutta il fatto che il carattere " è un terminatore del valore di un attributo, e puo essere usato per inserire del codice
- Una soluzione è impedire l'inserimento del carattere " nell'input utente
- In un altro contesto (diverso dal valore di un attributo) tale soluzione potrebbe essere inutile

Alcuni possibili contesti

Context	Example code
HTML element content	`<div>userInput</div>`
HTML attribute value	`<input value="userInput">`
URL query value	`http://example.com/?parameter=userInput`
CSS value	`color: userInput`
JavaScript value	`var name = "userInput";`

Momento del controllo

- L’applicazione può effettuare i controlli in due possibili momenti:
  - Inbound: ricezione dell’input dall’utente
  - Outbound: invio dell’output all’utente
- Dipende dall’architettura:
  - Se il sistema ha un unico punto di ingresso è piu semplice inbound
  - Se l’output viene costruito tramite diverse chiamate è piu semplice outbound

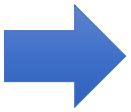
Locazione del controllo

- Controllo lato server: protegge da persistent e reflected XSS
- Controllo lato client: protegge da DOM XSS

Encoding

- Consiste nella trasformazione (escaping) degli input di modo che il browser non possa interpretarli come codice
- Esempio: HTML escaping
  - Converte **<** e **>** in < e >
- Effettuato tramite librerie apposite

```
print "<html>"
print "Latest comment: "
print encodeHtml(userInput)
print "</html>"
```



```
<html>
Latest comment:
&lt;script&gt;...&lt;/script&gt;
</html>
```

Encoding con Javascript

- Quando si usa Javascript per inserire degli input in un documento, alcune funzioni fornite dal sistema implementano l'encoding in maniera trasparente:

Context	Method/property
HTML element content	*node*.textContent = **userInput**
HTML attribute value	*element*.setAttribute(*attribute*, **userInput**) or *element*[*attribute*] = **userInput**
URL query value	window.encodeURIComponent(**userInput**)
CSS value	*element*.style.*property* = **userInput**

Convalida dell’input (validation)

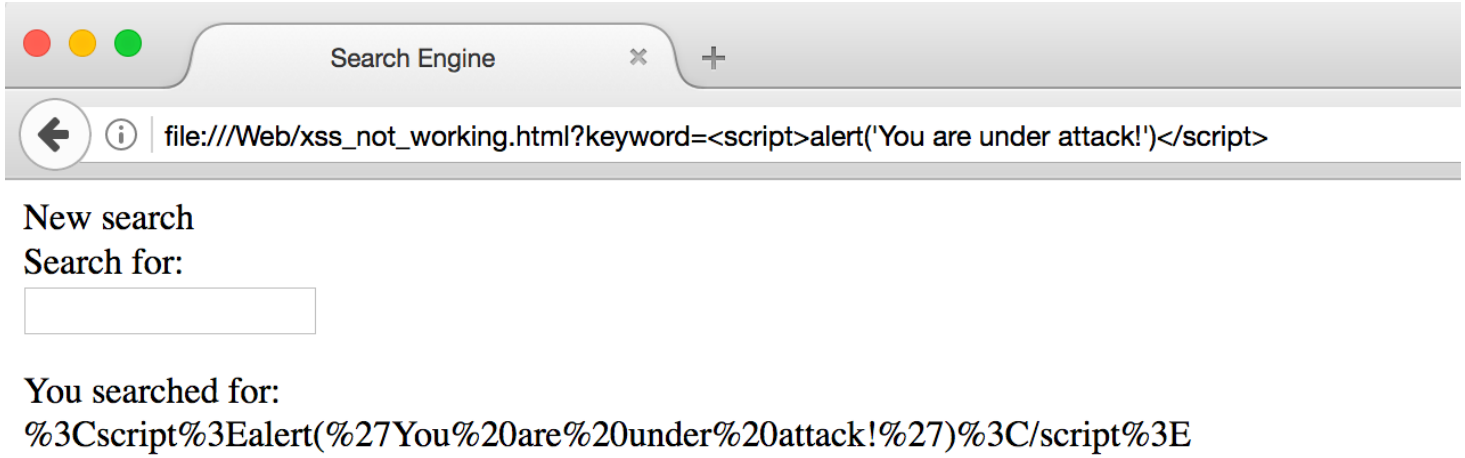
- Gli approcci si distinguono per:
  - Strategia di classificazione
  - Outcome
- La strategia di classificazione determina come sono identificati gli input validi da quelli non validi; ci sono due strategie:
  - Blacklisting: uno o piu pattern (regular expressions) identificano gli input pericolosi
    - Complesso: molti pattern da elencare
    - Invecchia velocemente
  - Whitelisting: i pattern identificano gli input validi
    - La complessità dipende dal tipo di input
    - Più longevo del blacklisting
- L’outcome determina il comportamento quando un input non è convalidato:
  - Rejection: l’input viene rifiutato
  - Sanitization: si cerca di rimuovere la parte non valida dall’input

Esempio

Questo codice è vulnerabile rispetto ad attacchi di tipo XSS.

```
<html>
<header>
<title>Search Engine</title>
</header>
<body>
<div>New search</div>
<div>
   <form method="get" action="">
   Search for:<br/>
   <input type="text" name="keyword"/><br/>
   </form>
</div>
<div>You searched for:</div>
<script>
   var url=document.URL;
   var pos=url.indexOf("keyword=")+8;
   if (pos > 8) {
      var keyword=url.substring(pos);
      document.write(keyword);
   }
</script>
</body>
</html>
```

Risultato



Sebbene ci aspettassimo l'esecuzione del codice javascript riportato nella barra URL del browser, questo non accade.

Il motivo è che la maggior parte dei browser moderni include meccanismi di difesa trasparenti all'utente e allo sviluppatore. In questo caso osserviamo il risultato del meccanismo di difesa implementato in "document.URL", che restituisce una stringa codificata affinché i caratteri speciali, ad esempio il carattere <, siano trasformati nel codice unicode corrispondente.

Questa è una soluzione di encoding.

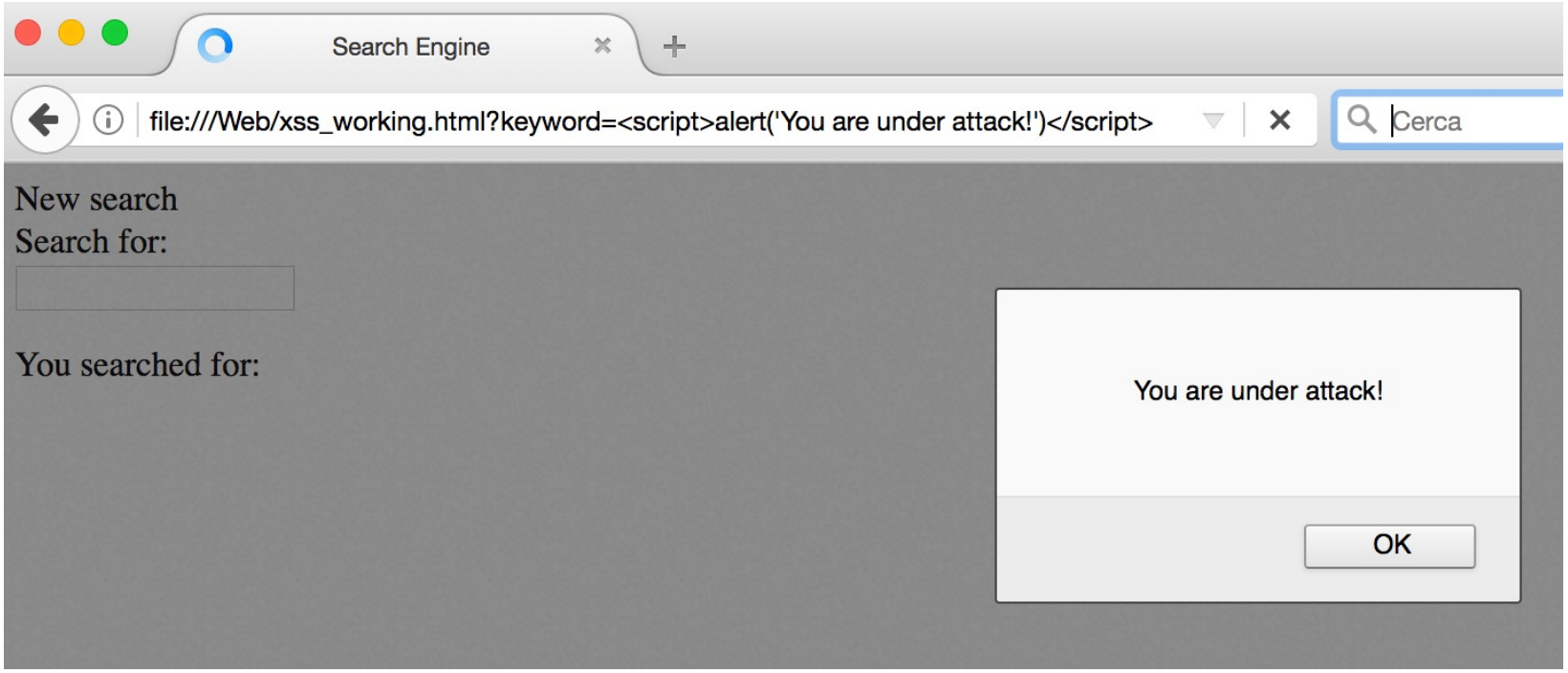
Attenzione, nel momento in cui sviluppate una pagina web non potete fare troppe assunzioni sul browser utilizzato dagli utenti, quindi è meglio non fare assunzioni sula presenza di meccanismi di difesa integrati nei browser.

Esempio (no encoding)

Questo codice è vulnerabile rispetto ad attacchi di tipo XSS in quanto lo sviluppatore ha deciso di decodificare l'URL preventivamente codificato dal browser (questo potrebbe dipendere da esigenze di sviluppo)

```
<html>
<header>
<title>Search Engine</title>
</header>
<body>
<div>New search</div>
<div>
   <form method="get" action="">
   Search for:<br/>
   <input type="text" name="keyword"/><br/>
   </form>
</div>
<div>You searched for:</div>
<script>
   var url=decodeURIComponent(document.URL);
   var pos=url.indexOf("keyword=")+8;
   if (pos > 8) {
      var keyword=url.substring(pos);
      document.write(keyword);
   }
</script>
</body>
</html>
```

Risultato: attacco riuscito

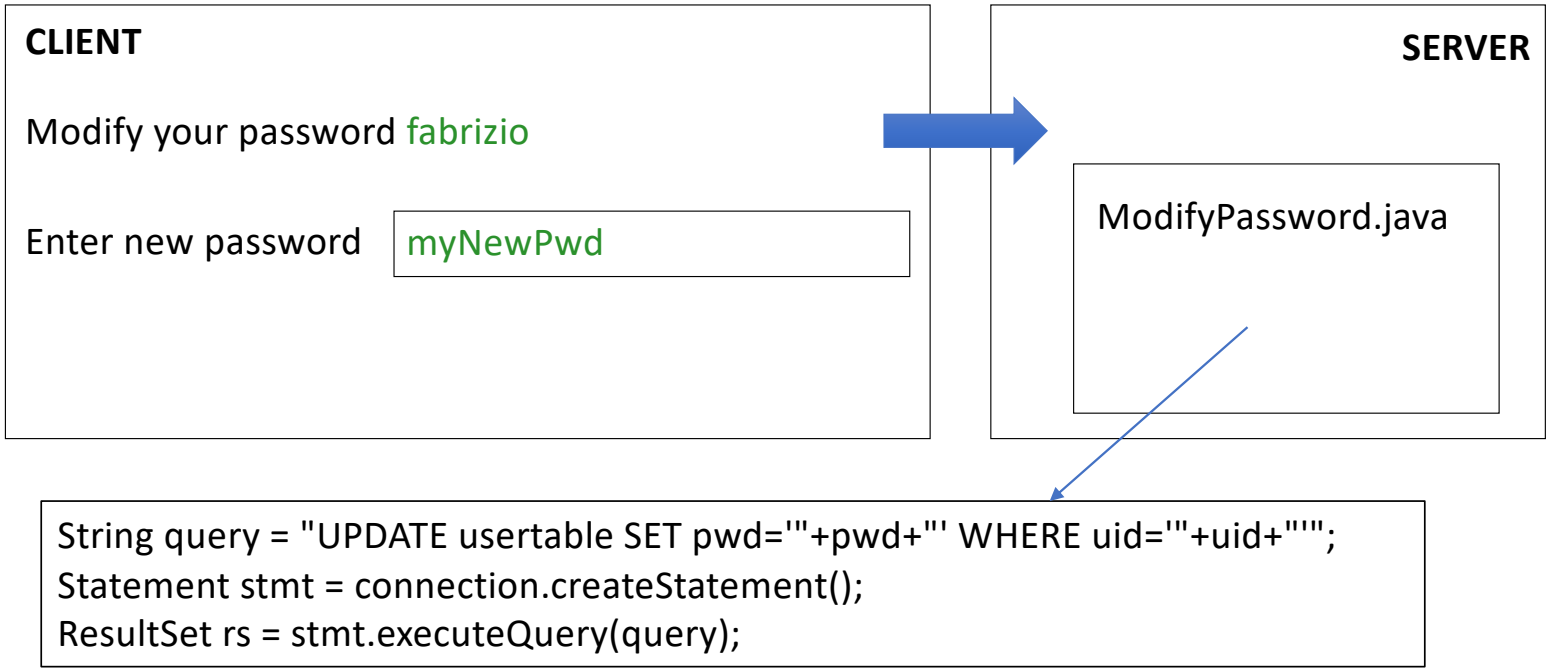


Schema di attacco SQL injection

Schema di attacco SQL injection

- È uno schema di attacco di tipo command injection
- Obiettivo: modificare l'esecuzione di un comando SQL parametrico all'interno dell'applicazione, agendo sul parametro
- Consiste nell'inserimento nel sistema di una (porzione di) query SQL attraverso l'input utente
- Un attacco eseguito con successo può:
  - Leggere/modificare dati sensibili da un DB
  - Eseguire operazioni di amministrazione del DB (ad esempio shutdown)
  - Recuperare il contenuto di un file presente nel filesystem del DBMS

SQL injection: esempio Java

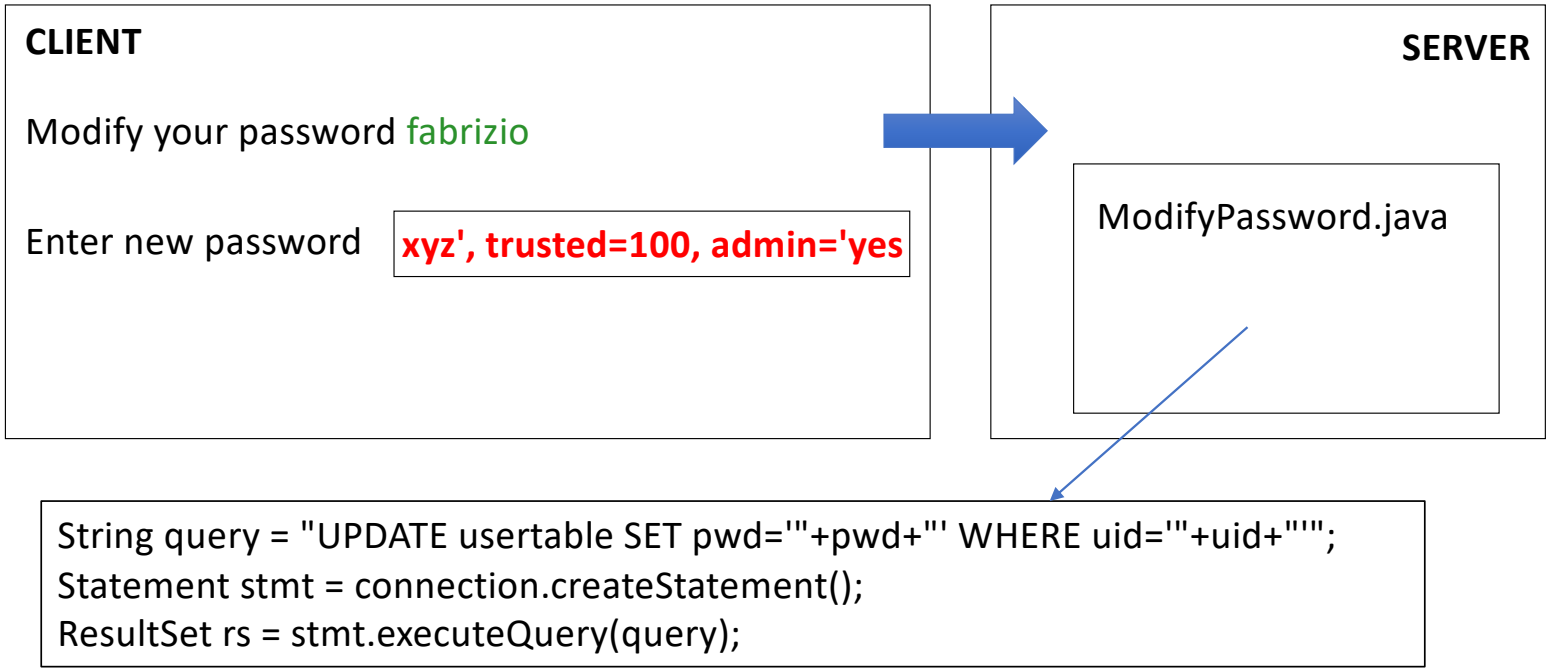



```
String query = "UPDATE usertable SET pwd='"+pwd+"' WHERE uid='"+uid+"'";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

In condizioni normali la query porta al risultato atteso (aggiorna il campo pwd)

query="UPDATE usertable SET pwd='myNewPwd' WHERE uid='fabrizio'"

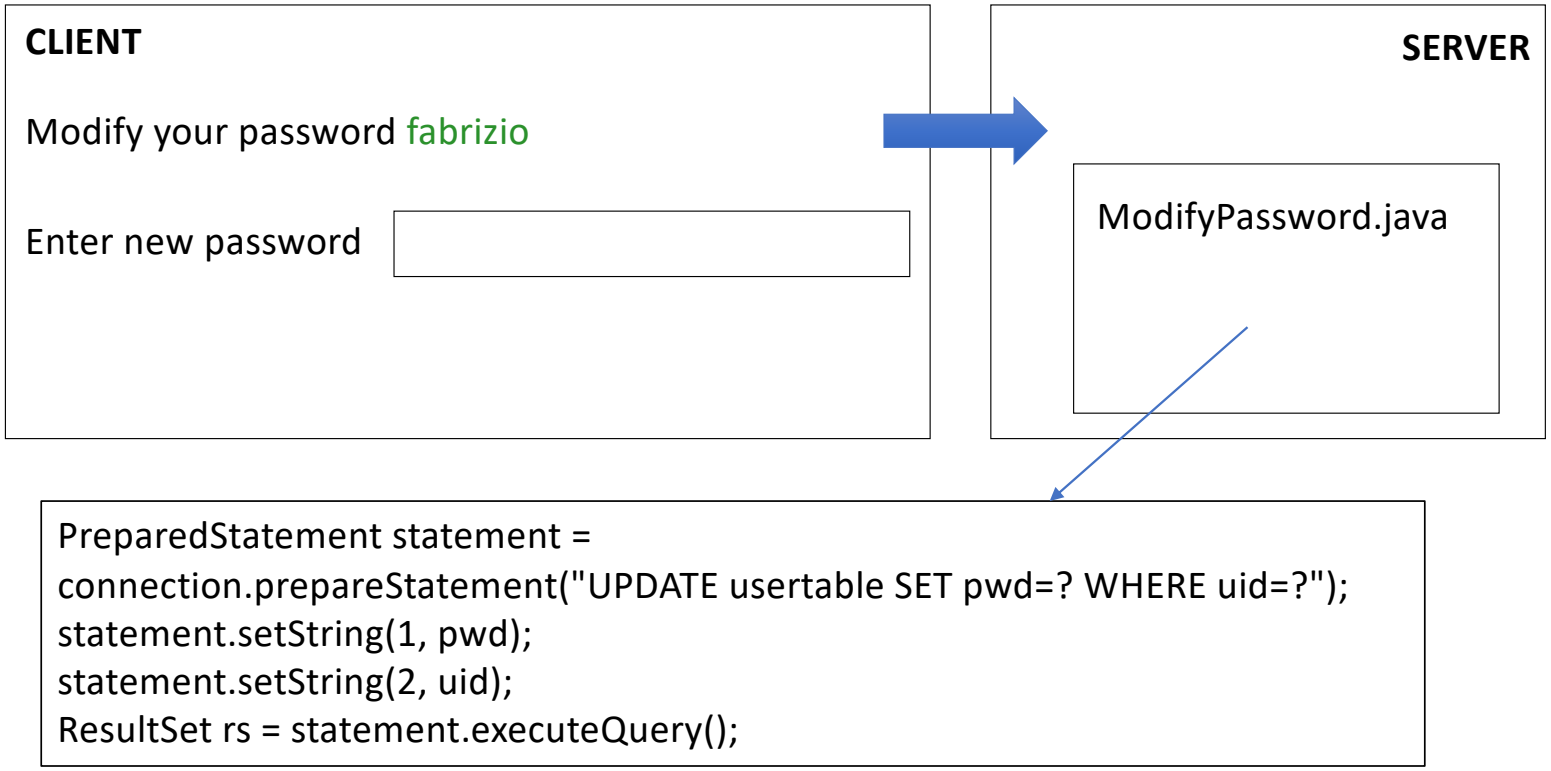
SQL injection: esempio Java



La SQL injection fa si che vengano cambiati altri campi (in questo caso modifica i privilegi)

query="UPDATE usertable SET pwd='**xyz', trusted=100, admin='yes**' WHERE uid='fabrizio'"

SQL injection: contromisure




```
PreparedStatement statement =
connection.prepareStatement("UPDATE usertable SET pwd=? WHERE uid=?");
statement.setString(1, pwd);
statement.setString(2, uid);
ResultSet rs = statement.executeQuery();
```

La libreria JDBC si occupa di effettuare l'encoding attraverso l'oggetto PreparedStatement

SQL injection e linguaggi di programmazione

- Questo tipo di weakness si può manifestare in tutti i programmi che si interfacciano con DBMS
- Storicamente sfruttati nel caso di web applications scritte con linguaggi quali
  - Perl
  - PHP
  - Python
  - ….
- Librerie di encoding per ovviare al problema sono disponibili per i maggiori linguaggi