M183 Applikationssicherheit Implementieren # 2

by Jürg Nietlispach

Recap # 1?

Recap

- Internet & Websystems
- Security Principles
- Motivation, Importance & Consequences
- Threats H2M & M2M Scenario
- Security Models, Detection & Prevention,
- Recommendations & Best Practices
  - Check often & Domain-Specific
- Hacker-Mindset: «What can i make this thing do»

Hacking Web Applications – where to start?

Consider a «classic» Web Application (Frontend & Backend)

1) Most interesting «things»?
2) How to get it?

Hacking Web Applications - Passwords

Passwords!

How?

- Traffic Monitoring (non-ssl)
- Try Default CMS Passwords (analyze HTML-Source for Wordpress, Typo3, …)
- Brute Force Passwords
- **Access Passwords through XSS Phishing / Keylogging**
- **«Classic» Email Phishing**
- Make the Application fail (hoping for good error-reporting!)
  - Known Plugin Issues (Wordpress)
  - Known Framework Issues (HTML-Response-Header)
  - Injections (Code, SQL) on Forms and Application
  - …

Hacking Web Applications - Documents

Data & Documents!

How?

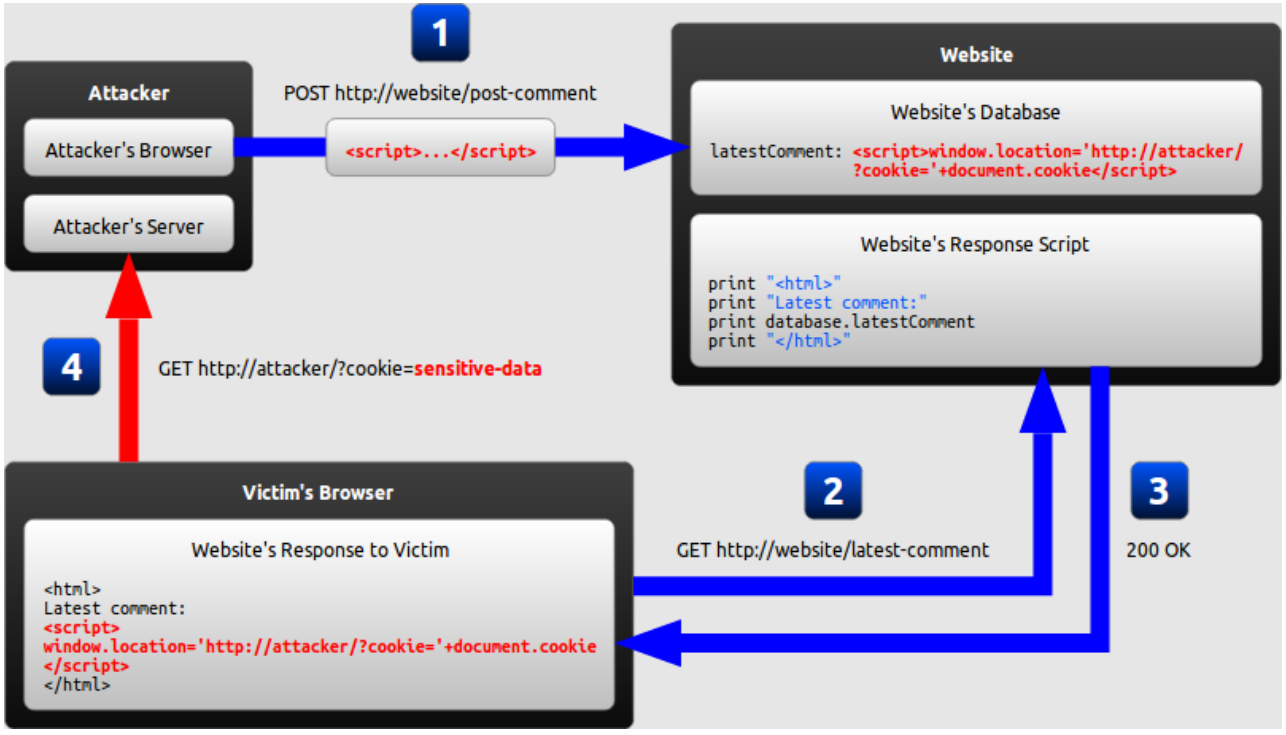
- Try default CMS-Document-Locations (/wp-assets/)
- Check «Static» Files
  - Robots.txt (-> Ressources, that should not be indexed),
  - sitemaps.xml (-> potential «hidden» content)
- Try Directory Listings (apache: allowIndex = false)
- SQL-Injections …

Password Phishing / Keylogging with XSS

XSS Scenarios?

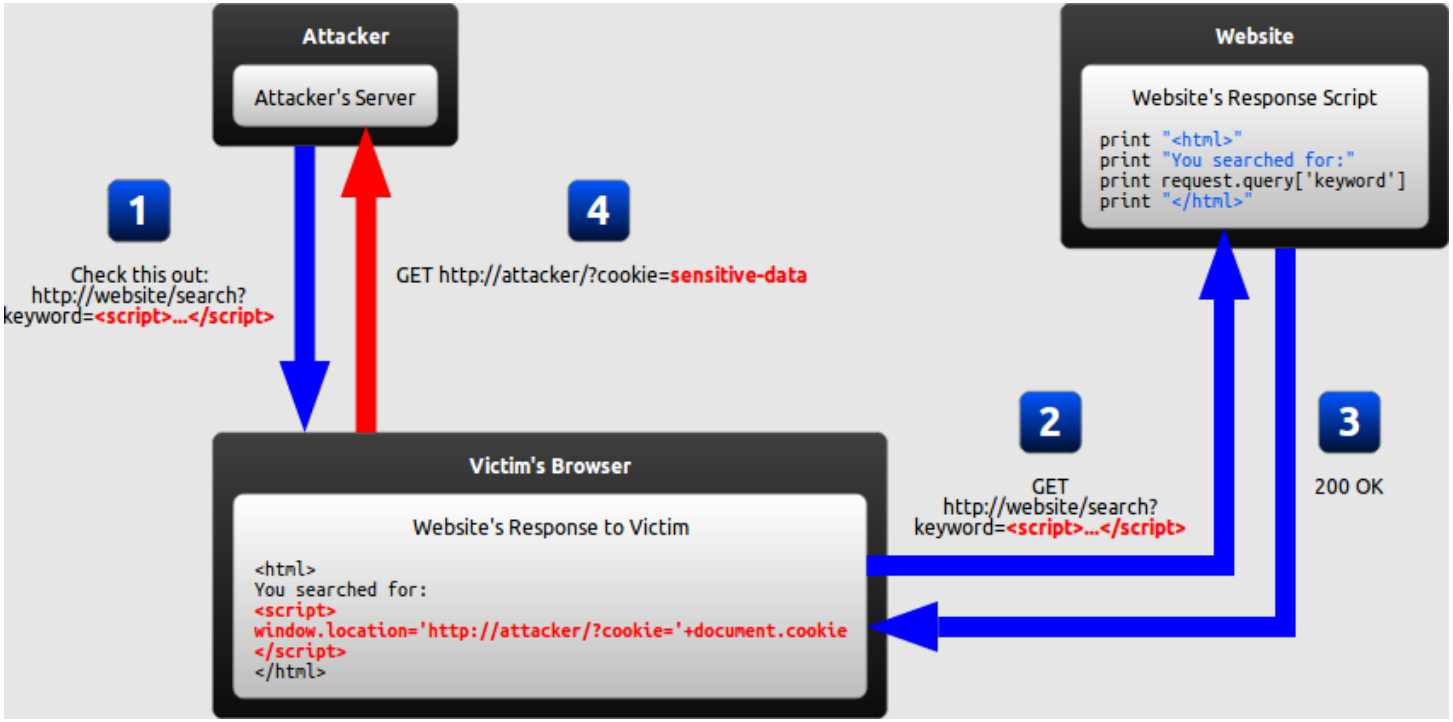
1 Persistent / Stored XSS Attack

1. Attacker injects JS-Snippet on target page (i.e. Blog-Post / Comment / etc.)
2. User gets JS-Snippet via HTTP-Response
3. User activity is monitored through the snippet or
   fake login is generated dynamically
4. Attacker receives desired information



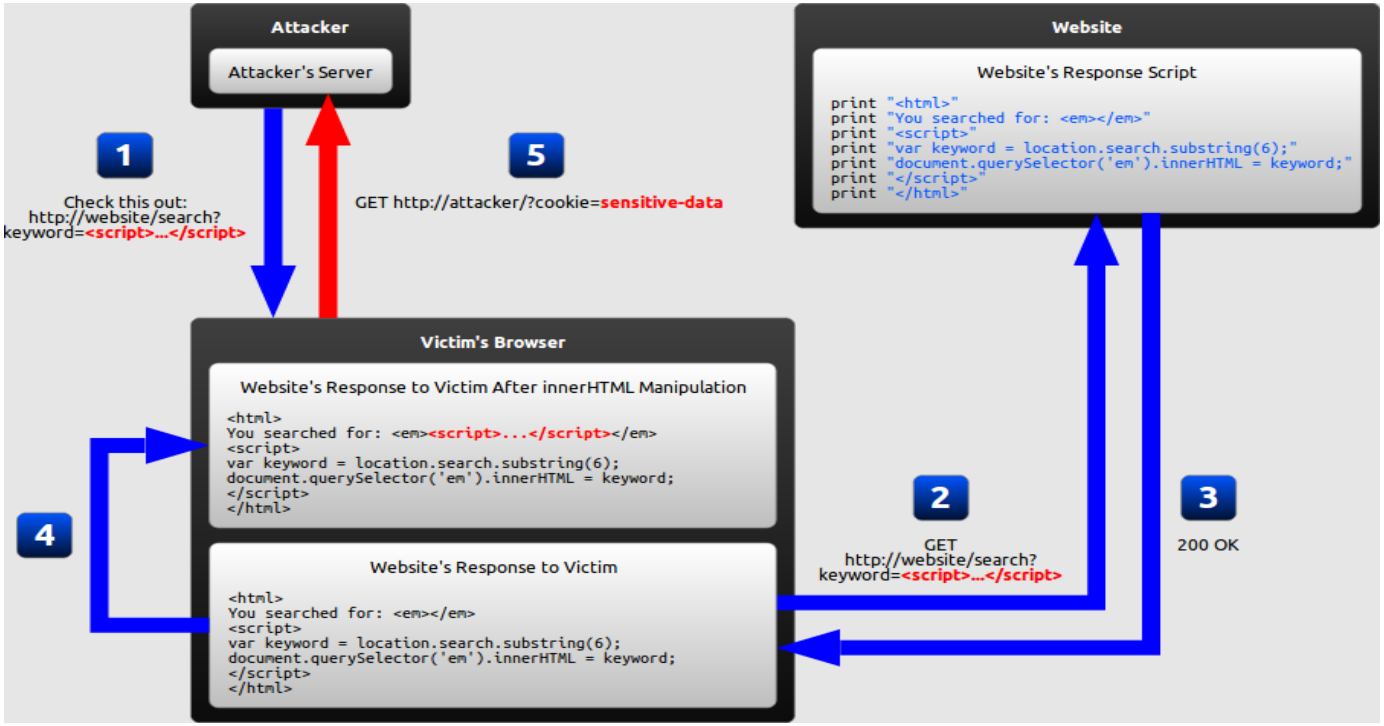
2 Reflected XSS Attack

1. Attacker creates a link containing the script and sends it to the user
2. User visits link
3. The user executes (Keylogging, fake Login) the JS-Snippet as part of the website's response
4. Attacker receives desired information



3 DOM Based XSS Attack

1. Attacker creates a link containing the script and sends it to the user
2. User visits link
3. The website does return (among other data) a script of its own
4. The malicious script is then executed «as a parameter» for the proper website script
5. Attacker receives desired information



Example DOM Based XSS Attack

1. Attacker creates a link
   `http://www.example.com/test.html#<script>alert(1)</script>`

2. User visits link
3. The website does return (among other data) a script of its own

```
<script>
    document.write("<b>Current URL</b> : " + document.baseURI);
</script>
```

4. The malicious script is then executed in the `document.write()`- Function
5. The Attacker gets his information

XSS Attack Prevention?

XSS Attack Prevention

1. **Stored/Persistent XSS Attack**

-> Secure Input Handling

    1. Escaping / Encoding
    2. Validation / Filter

2. Reflected XSS Attack

-> Secure Input Handling

    1. Escaping / Encoding
    2. Validation / Filter

3. DOM Based XSS Attack

-> Secure Input Handling

    1. Escaping / Encoding

-> Secure Coding Practices

    1. Do not use certain functions like eval() etc.

“Classic” Email Phishing

Phishing Scenarios?

“Classic“ Email Phishing

Phishing Scenario 1 & 2

1. User receives fake email
2. User clicks on link
3. User enters password in fake page
    1. «fake page»: Web Application is reverse-engineered 1:1
    2. «fake page»: Web Application is consumed inside an iframe (Cross-Origin)

“Classic“ Email Phishing Prevention?

“Classic“ Email Phishing Prevention?

Emails:

- Identify Suspected Phishing Emails
- Source of Information From Incoming Mail
- Do not click on hyperlinks or links attached in the email, as it might direct you to a fraudulent website
- Enter Your Sensitive Data in Secure Websites Only
- Use Google-Mail ☺
- …

Iframes:

- Control for Iframe Embedding:
  - X-FRAME-OPTIONS (DENY, SAMEORIGIN)
- Javascript Solution: `top` vs `self`

Übungen

Allgemeines

«Code From Scratch» mit verständlichen Kommentaren & Gedankenschritten!

Empfehlung: Pair Programming

Angabe des entspr. .git Repos für Code (Public, regelmässige pushes)

Gegen Schluss der Stunde kurze Präsentation Stand der Arbeiten

Übung «XSS in Practice»

Implementierung Keylogger-Snippet (Plain Javascript)

1. index.html-File erstellen
2. <script></script> - Tag vor dem </body> - Tag
3. «Events» für Keyboard-Keys aktivieren und via console.log() ausgeben («Buchstaben sammeln»)
4. «Wörter» sammeln (Wörter in console.log() ausgeben)
5. «Sätze» sammeln (Sätze in console.log() ausgeben)
6. Problem: Page Relaod! -> Cookie oder Local-Storage
7. «Sätze» in Cookie & Local-Storage speichern (und in console.log() ausgeben)
8. «AJAX» Routine vorbereiten, welche dann die «Sätze» an einen Endpunkt (noch zu definieren) sendet.

Übung «XSS in Practice»

Fragen / Konzeptuelle Gedanken zum Keylogger-Snippet

- Was ist beim Endpunkt für den Push der Benutzerdaten zu beachten (Hintergrund: Persistente XSS Attacke)?

- Was ist bei der Wahl des Zeitpunkts des Datenpushes zu beachten?

- Bei welchen Webseiten funktioniert das Snippet (10’ max)?