TROOPERS 10

IT-Security Conference & Workshops

Unbreakable ABAP?
Vulnerabilities in custom ABAP Code

Markus Schumacher, Co-Founder Virtual Forge GmbH

10.- 12. März 2010
Print Media Academy, Heidelberg

Virtual Forge GmbH - http://virtualforge.de

- „GmbH“ since 1.1.2006, headquarters in Heidelberg
- Long-lasting consulting experience
- Application security, focus SAP from day 1
- Code Profiler, http://www.codeprofilers.com
- SAP audits and code reviews
- Book: „Sichere ABAP-Programmierung“, http://sap-press.de/2037
- Trainings

Agenda

- ABAP development - risks in Web applications (example)
  - ABAP/BSP vs. OWASP Top 10
- Examples of vulnerabilities in custom coding
  - Business Server Pages
    - Inline ABAP in HTML
    - HTMLB-Tag-Library
  - Open SQL
    - Dynamic Open SQL
    - SQL-Injection
- Conclusion

ABAP in a Nutshell

- Exists since ~30 years
- COBOL-like syntax
- “grown language”
  → several programming paradigms at the same time
  → very context sensitive, no reserved keywords
- DB-independent SQL-dialect built in
- Code is stored in DB
- Development environment developed in ABAP
  - Code stored on server
  - Access via transaction SE80
  - Transport management

ABAP and Open Source (but not free!)

- Sourcecode is completely available in a SAP installation
  - „SAP standard“ code plus custom coding
- Customers can change code
  - Copy, rename, and modify code
  - Change SAP standard code („modification“)
- ABAP allows several development frameworks
  - Customers write their own code in order to adapt the standard to their needs („customizing“)
  - Custom development for non-standard business processes
  - 3rd party add-ons

Frontend-Technologies

- Dynpro
  - Written in ABAP
  - Requires proprietary UI (SAP GUI)
  - Similar to X11 paradigm

- Internet Transaction Server (ITS)
  - 1st Web-Technologie of SAP
  - Development almost stopped, but widely used

Frontend-Technologies

- Business Server Pages (BSP)
  - HTML with embedded ABAP (similar to JSP)
  - Several programming paradigms incl. MVC
  - Widely used, customers still build new applications

- Web Dynpro (ABAP | Java)
  - UI-independent framework, „point & click“ programming for UI design
  - Developer can‘t embed his own HTML/JavaScript
  - Developer can‘t cause a vulnerability. But he also can‘t avoid them

Frontend-Technologies

- Web GUI
  - HTML-version of regular Dynpros (SAP GUI)
  - Earlier version on top of Internet Transaction Server
  - Today as plugin of SAP Web Application Server

… external systems (via JCo or RFC), Adobe Flash, Microsoft Silverlight, PHP, Phython, etc.

Further Technologies

- File access
- Database access (OpenSQL, Native SQL)
- Remote access
  - HTTP, FTP, Email, …
  - Messaging (PI/XI)
  - Web Services (SOAP)
  - RFC - Remote Function Call

- Whatever you need – SAP has it, but be aware of the little differences

SAP Web Technology

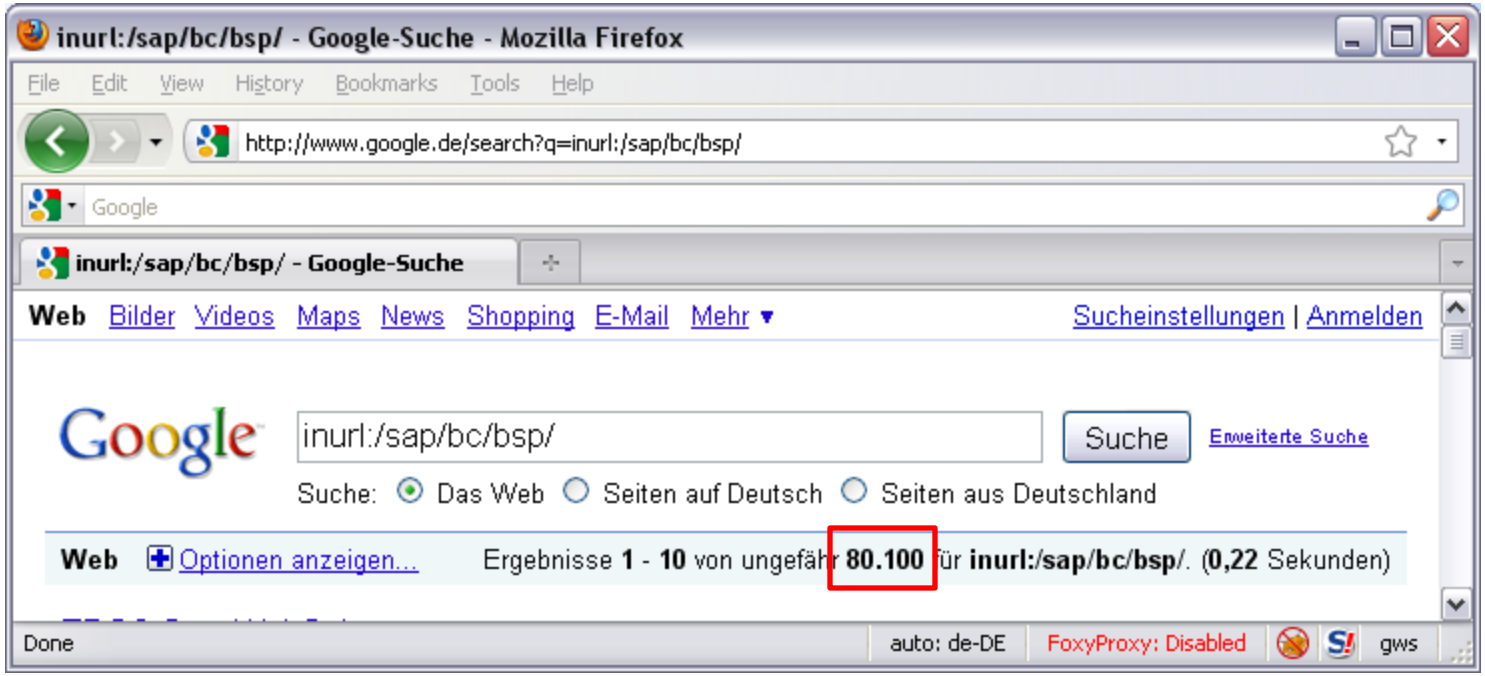
SAP NetWeaver Web Application Server (Web AS):

- Supports Single Sign On (SSO)
- SSO-ticket stored in cookie (`MYSAPSSO2`)
  - By default issued for path `/` and `domain.tld`
  - By default neither `httpOnly`, nor `secure`
- Development of your own HTTP-Handler possible
  - BSP, Web Dynpro, WebGUI are HTTP-Handler
- Configuration via profile parameter (*report* `RZ11`) and *transaction* `SICF`
- Blacklist implementation filters <script, %00 and other patterns

Business Server Pages (BSP)

Finding BSP applications:

- http://www.google.de/search?q=inurl:/sap/bc/bsp/



Business Server Pages (BSP)

- mentor.com
- erco.org
- sap-ag.de
- beiersdorfgroup.com
- mybayerjob.de
- heraeus.com
- wacker.com
- heidelberg.com
- knorr-bremse.com
- ottopersonalsysteme.de
- skyguide.ch
- eads.com
- bsr.de
- kuka.de
- kpmg.de
- daad.de
- euhreka.com
- umdasch.com
- celesio.com
- pflegedienst-navigator.de
- oebb.at
- salzburg-ag.at
- whirlpool.com
- volkswagen.de
- pharma.com
- wa.gov
- brucepower.com
- jetblue.com
- suzukiautoco.com
- singaporepower.com
- kaufland.de
- clavis-bonn.de
- albatha.ae
- vodafone.com
- iom.int
- wlw.de
- erecruiting-randstad.de
- lieferantensuchmaschine.com
- audi.de
- blanco.de
- festo.com
- vhv.de
- otto.de
- abb.de
- ruv.de
- holcim.com
- mannheim.de
- softsurvey.de

Business Server Pages (BSP)

OWASP Top 10	Potentially vulnerable?
A1 – Cross-Site Scripting (XSS)	Yes
A2 - Injection Flaws	Yes
A3 - Malicious File Execution	Yes
A4 - Insecure Direct Object Reference	Yes
A5 - Cross Site Request Forgery (CSRF)	Yes
A6 - Information Leakage and Improper Error Handling	n/a
A7 - Broken Authentication and Session Management	n/a
A8 - Insecure Cryptographic Storage	n/a
A9 - Insecure Communications	n/a
A10 - Failure to Restrict URL Access	n/a

Agenda

- ABAP development - risks in Web applications (example)
  - ABAP/BSP vs. OWASP Top 10
- Examples of vulnerabilities in custom coding
  - **Business Server Pages**
    - **Inline ABAP in HTML**
    - **HTMLB-Tag-Library**
  - Open SQL
    - Dynamic Open SQL
    - SQL-Injection
- Conclusion

Business Server Pages

Preventing Cross-Site Scripting by Encoding/Escaping

- in Plain-HTML-Pages
  - ABAP-Encoding-Functions (`CL_HTTP_UTILITY`)
  - BSP-Page Attribute (`forceEncode`)
- in Pages with HTMLB-Taglib
  - Tag-Attribute (`forceEncode`)

Business Server Pages – Plain HTML

```
1 <%@page language="abap" %>
2 <% DATA: name TYPE string.
3    name = request->get_form_field( 'name' ).
4 %>
5 <html>
6 <head><title>HTML mit eingebettetem ABAP</title></head>
7 <p>Hello <%= name %> </p>
8 <body>
9 </body></html>
```

Business Server Pages – Plain HTML

```
1 <html>
2   <head><title>HTML mit eingebettetem ABAP</title>
  </head>
3  <body>
4    <p>Hello Guest</p>
5  </body>
6  </html>
```

Business Server Pages – Plain HTML

Cross-Site Scripting Vulnerability:

```
http://.../example0.htm?name=<img src="abc" onerror="
   alert(document.cookie);">
```

```
1 <html>
2 <head><title>HTML mit eingebettetem ABAP</title></head>
3 <p>Hello <img src= onerror="alert(document.cookie);"> </p>
4 <body>
5 </body></html>
```

Business Server Pages – Plain HTML

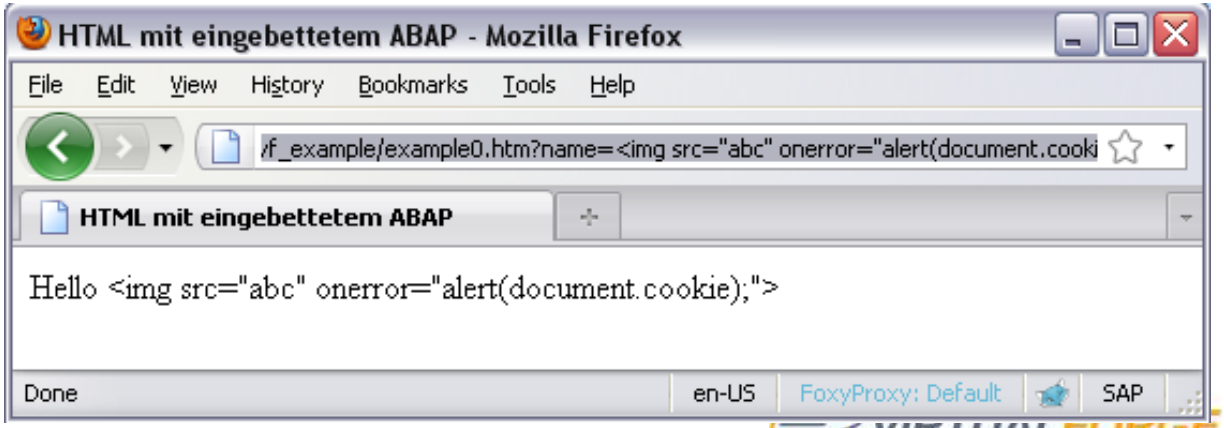
```
1 <%@page language="abap" %>
2 <% DATA: name TYPE string.
3    name = request->get_form_field( 'name' ).
4    name = CL_HTTP_UTILITY=>escape_html( name ).
4 %>
5 <html>
6 <head><title>HTML mit eingebettetem ABAP</title></head>
7 <p>Hello <%= name %> </p>
8 <body>
9 </body></html>
```

Business Server Pages – Plain HTML

Prevented Cross-Site Scripting Schwachstelle:

- `http://.../example0.htm?name=`**`<img src="abc" onerror="alert(document.cookie);">`**

```
1 <html>
2 <head><title>HTML mit eingebettetem ABAP</title></head>
3 <p>Hello &lt;img src="abc" onerror="alert(document.cookie);
  "&gt; </p>
4 <body>
5 </body></html>
```



Business Server Pages – Plain HTML

- Encoding of data by
  - **CL_HTTP_UTILITY=>escape_html( )**
  - **CL_HTTP_UTILITY=>escape_javascript( )**
  - **CL_HTTP_UTILITY=>escape_url( )**
  - **<%HTML=$VAR %>, <%URL= … %>, <% …**
- Pro:
  - Prevents XSS-Vulnerabilities
- Contra:
  - Every single output must be encoded according to the HTML-context
  - High effort, error prone

Business Server Pages – Plain HTML

```
1 <%@page language="abap" forceEncode="html"
2 DATA: name TYPE string.
3 name = request->get_form_field( 'name' ).
4 %>
5 <html>
6 <head><title></title></head>
7 <p>Hello <%= name %> </p>
8 <body>
9 </body></html>
```

Business Server Pages – Plain HTML

Preventing Cross-Site Scripting by

- `<%page forceEncode="{html|url|javascript}">`
- Global encoding via page attribute
- All output is encoded in the same way, no distinction between HTML-context (JavaScript, URL, ...)

Counterexample:

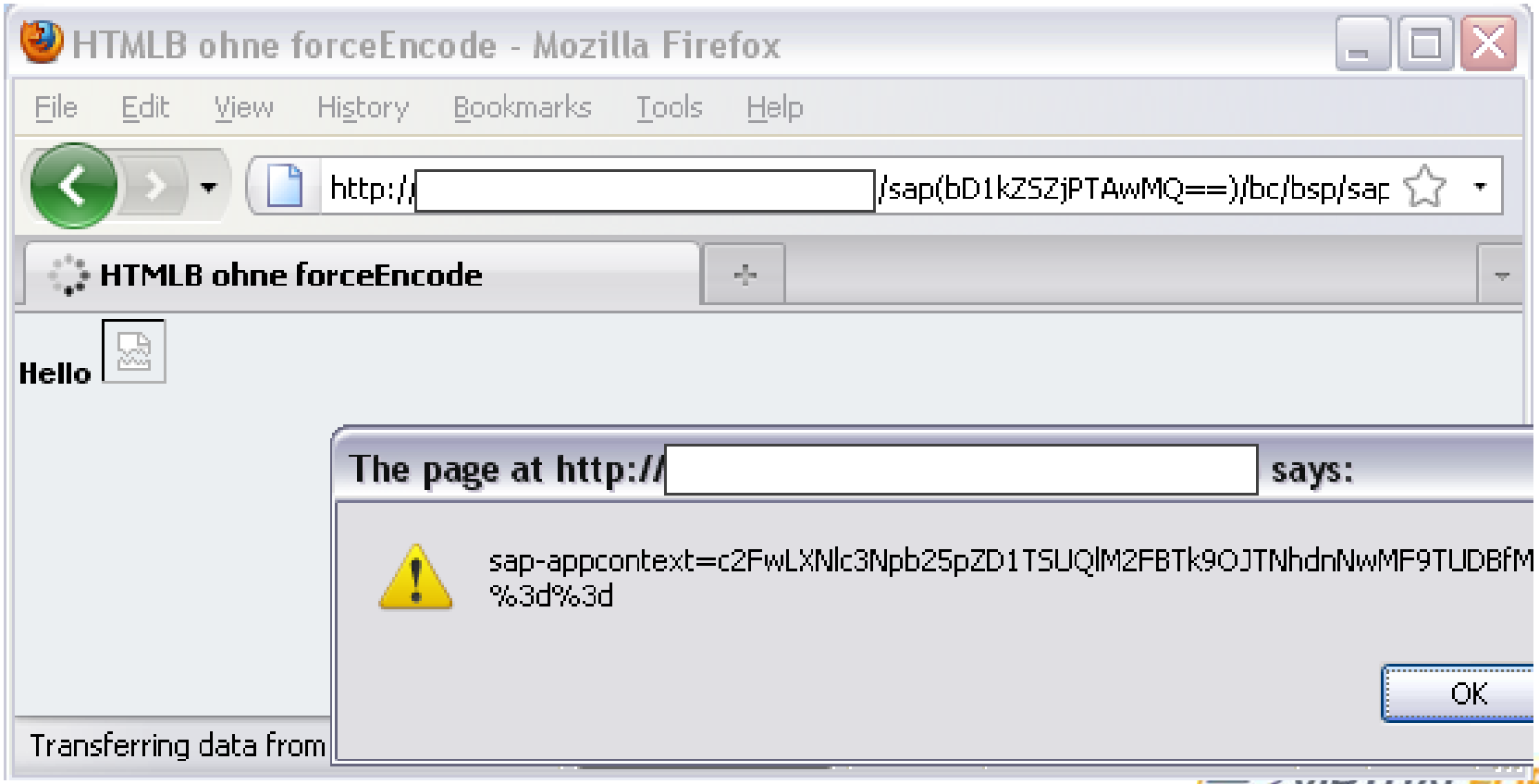
```
<a href="<%= request->get_form_field( 'user' ). %>">
   Link</a>
http://.../test.htm?user=javascript:document.write( ...
```

Business Server Pages – HTMLB

```
1  <%@page language="abap" %>
2  <%@extension name="htmlb" prefix="htmlb" %>
3  <% DATA: name TYPE string.
4    name = request->get_form_field( 'name' ). %>
5  <htmlb:content design="design2003">
6   <htmlb:page title = "HTMLB ohne forceEncode">
7     <htmlb:form>
8       <htmlb:textView     text          = "Hello <%= name %>"
9                           design        = "EMPHASIZED" />
10    </htmlb:form>
11   </htmlb:page>
12 </htmlb:content>
```

Business Server Pages – HTMLB

- `name=<img src="abc" onerror="alert(document.cookie);">`

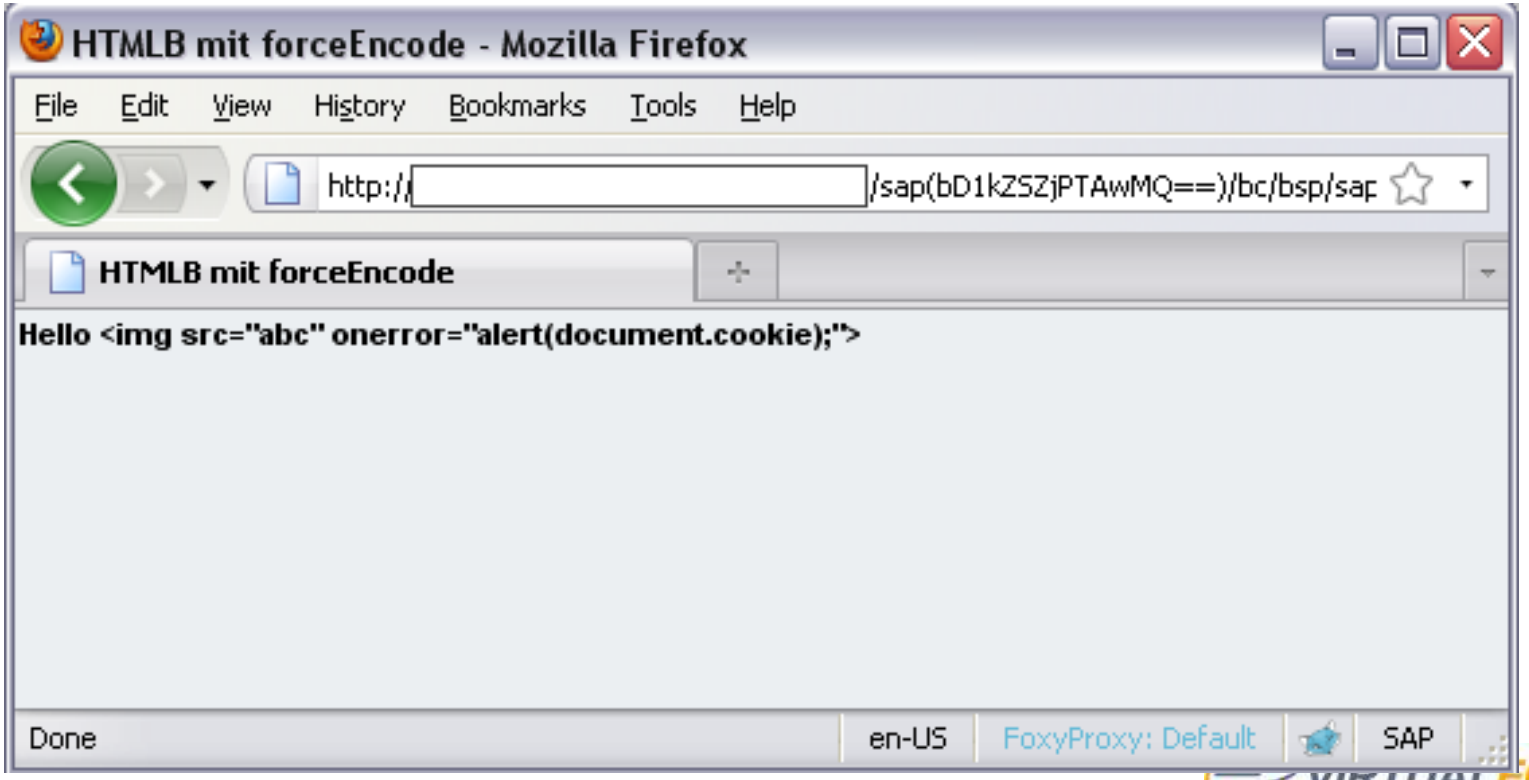


Business Server Pages – HTMLB

```
1  <%@page language="abap" %>
2  <%@extension name="htmlb" prefix="htmlb" %>
3  <% DATA: name TYPE string.
4    name = request->get_form_field( 'name' ). %>
5  <htmlb:content design="design2003" forceEncode="ENABLED">
6   <htmlb:page title = "HTMLB mit forceEncode">
7     <htmlb:form>
8       <htmlb:textView     text          = "Hello <%= name %>"
9                           design        = "EMPHASIZED" />
10    </htmlb:form>
11  </htmlb:page>
12 </htmlb:content>
```

Business Server Pages – HTMLB

- `name=`
  `<img src="abc" onerror="alert(document.cookie);">`



Business Server Pages

Preventing Cross-Site Scripting in Plain-HTML

- Encoding with methods (`CL_HTTP_UTILITY`)
  - High effort, error prone
- Encoding via page attribute (`forceEncode`)
  - Not speficic for HTML-context, no complete coverage of attacks

Preventing Cross-Site Scripting in HTMLB

- Tag-attribute `forceEncode` per default deactivated, must be set explicitly

Agenda

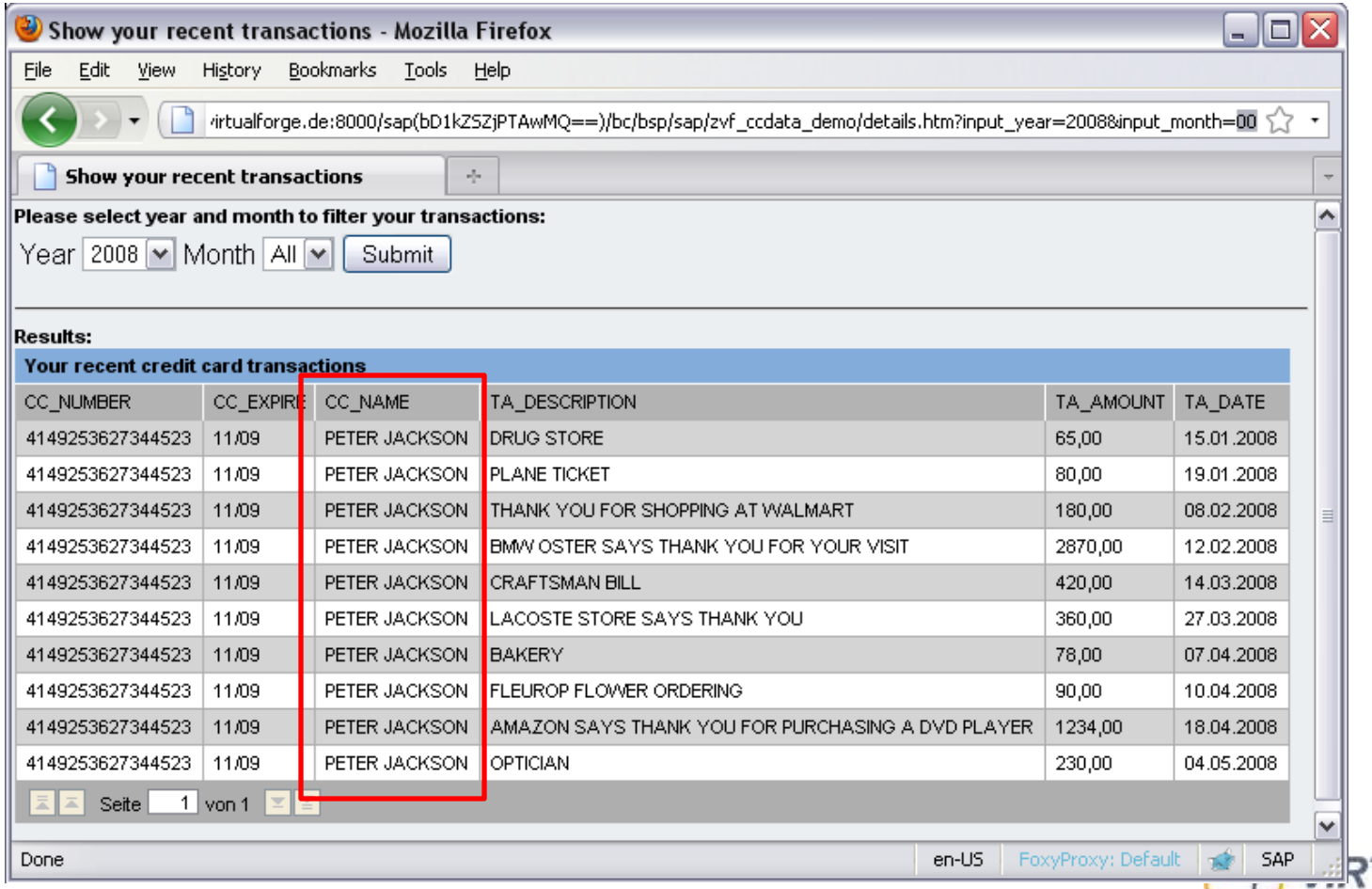
- ABAP development - risks in Web applications (example)
  - ABAP/BSP vs. OWASP Top 10
- Examples of vulnerabilities in custom coding
  - Business Server Pages
    - Inline ABAP in HTML
    - HTMLB-Tag-Library
  - **Open SQL**
    - **Dynamic Open SQL**
    - **SQL-Injection**
- Conclusion

Open SQL

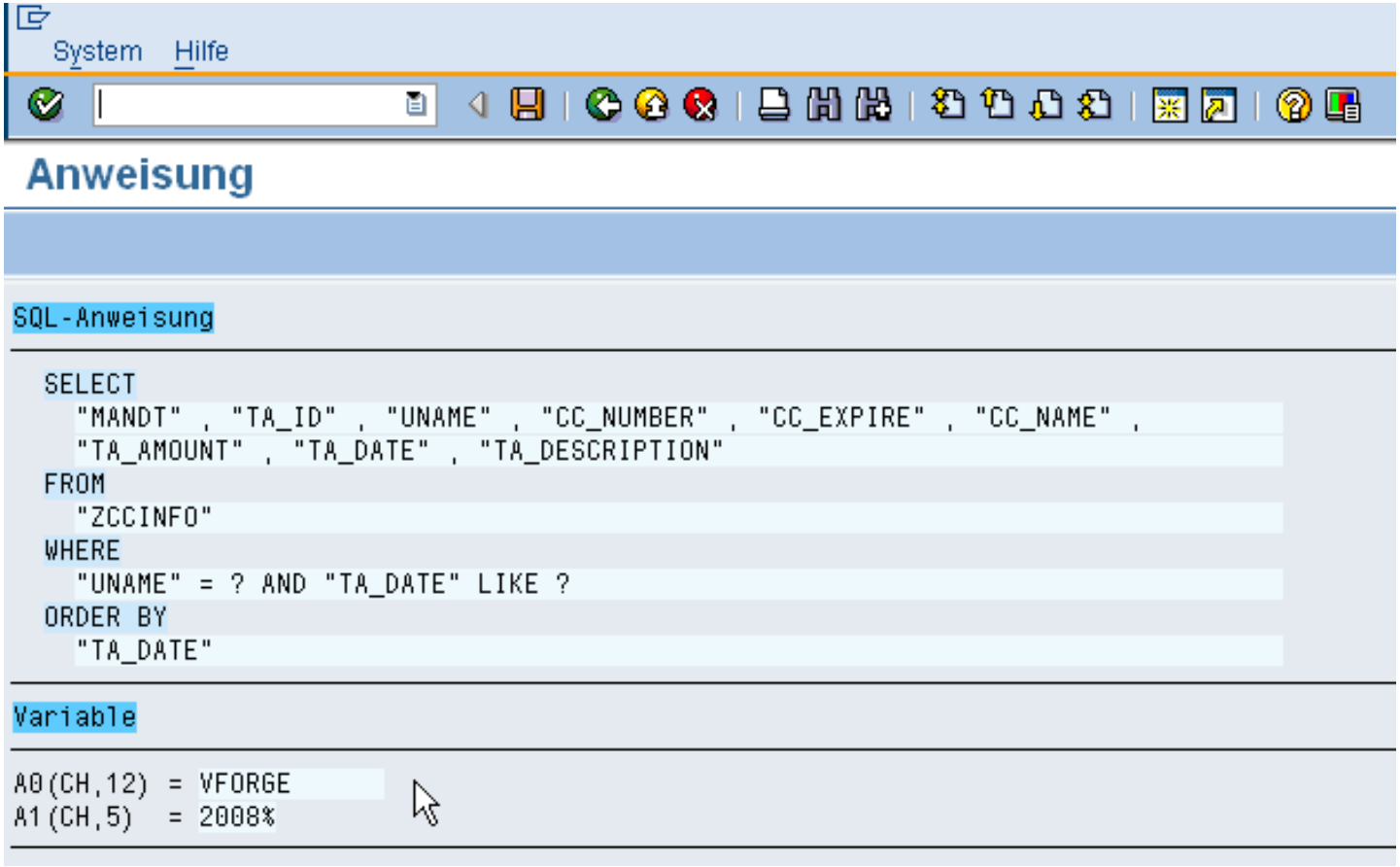
- Open SQL built in ABAP
- Internally converted to prepared statements
- SQL-statement and user data separated, no SQL-Injection possible

```
1  SELECT * FROM ZCCINFO
2  INTO l_zccinfo
3  WHERE uname = l_uname
4  AND ta_date = l_date.
```

Dynamic Open SQL - Example



Open SQL

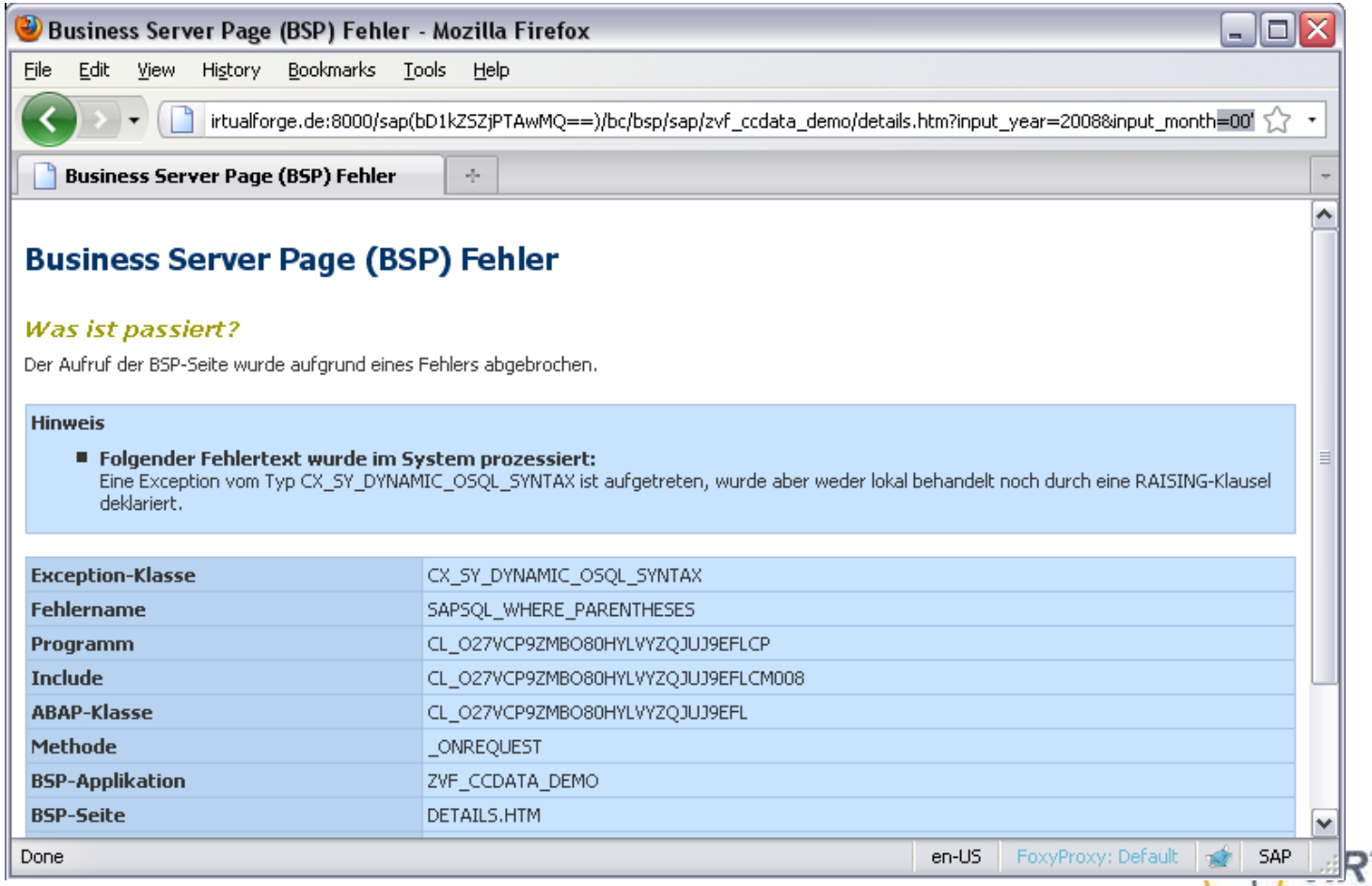

System Hilfe
Anweisung
SQL-Anweisung
SELECT
"MANDT" , "TA_ID" , "UNAME" , "CC_NUMBER" , "CC_EXPIRE" , "CC_NAME" ,
"TA_AMOUNT" , "TA_DATE" , "TA_DESCRIPTION"
FROM
"ZCCINFO"
WHERE
"UNAME" = ? AND "TA_DATE" LIKE ?
ORDER BY
"TA_DATE"
Variable
A0(CH,12) = VFORGE
A1(CH,5) = 2008%

Dynamic Open SQL

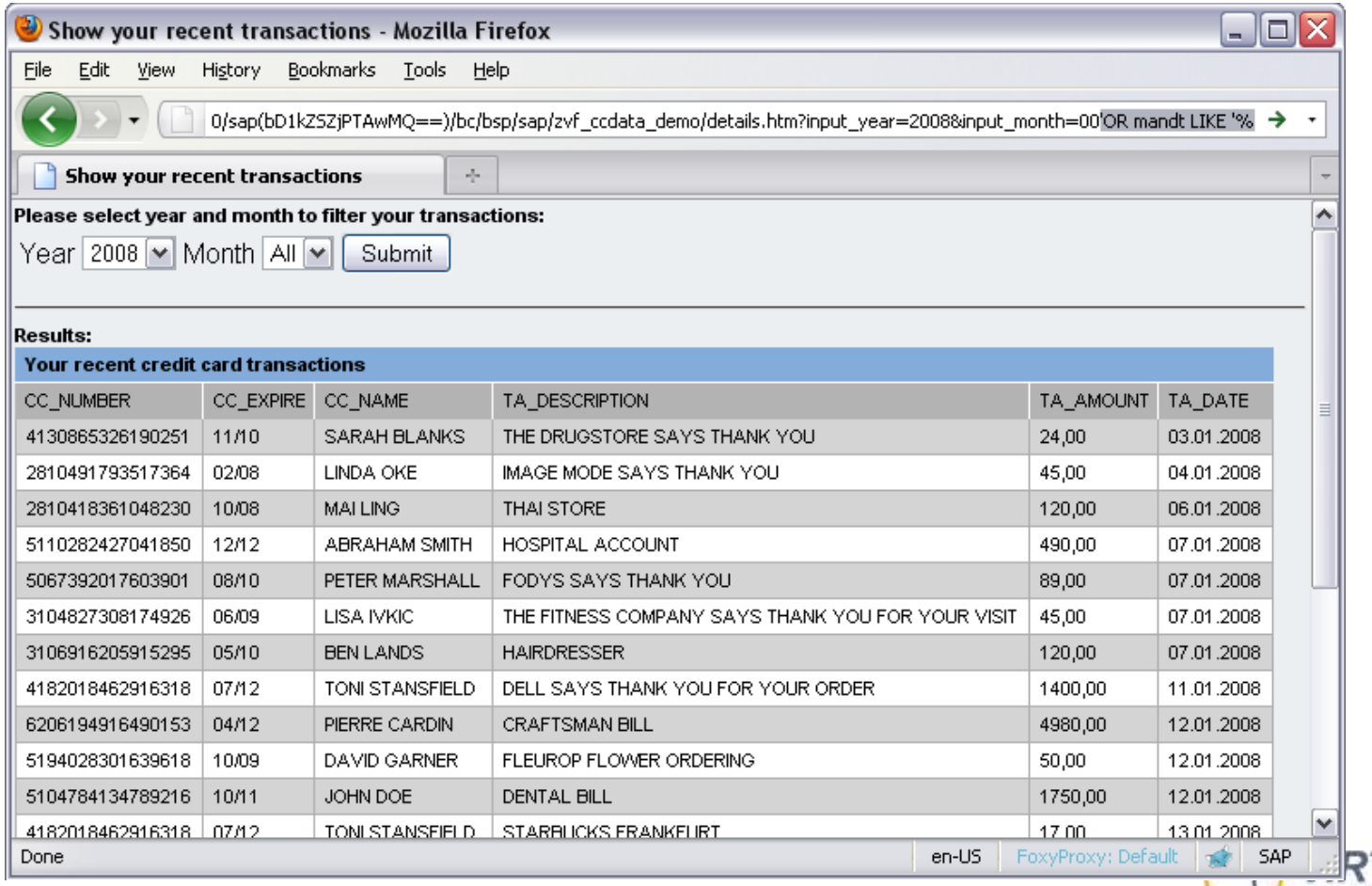
- Interprets String literal as SQL-Statement
- No encoding functions
  - User data can‘t be separated from SQL-commands
  - SQL-Injection very likely, when user data is part of dynamic SQL-Statement

```
1  SELECT (l_felder) FROM (l_table)
2  INTO l_zccinfo
3  WHERE (l_where).
```

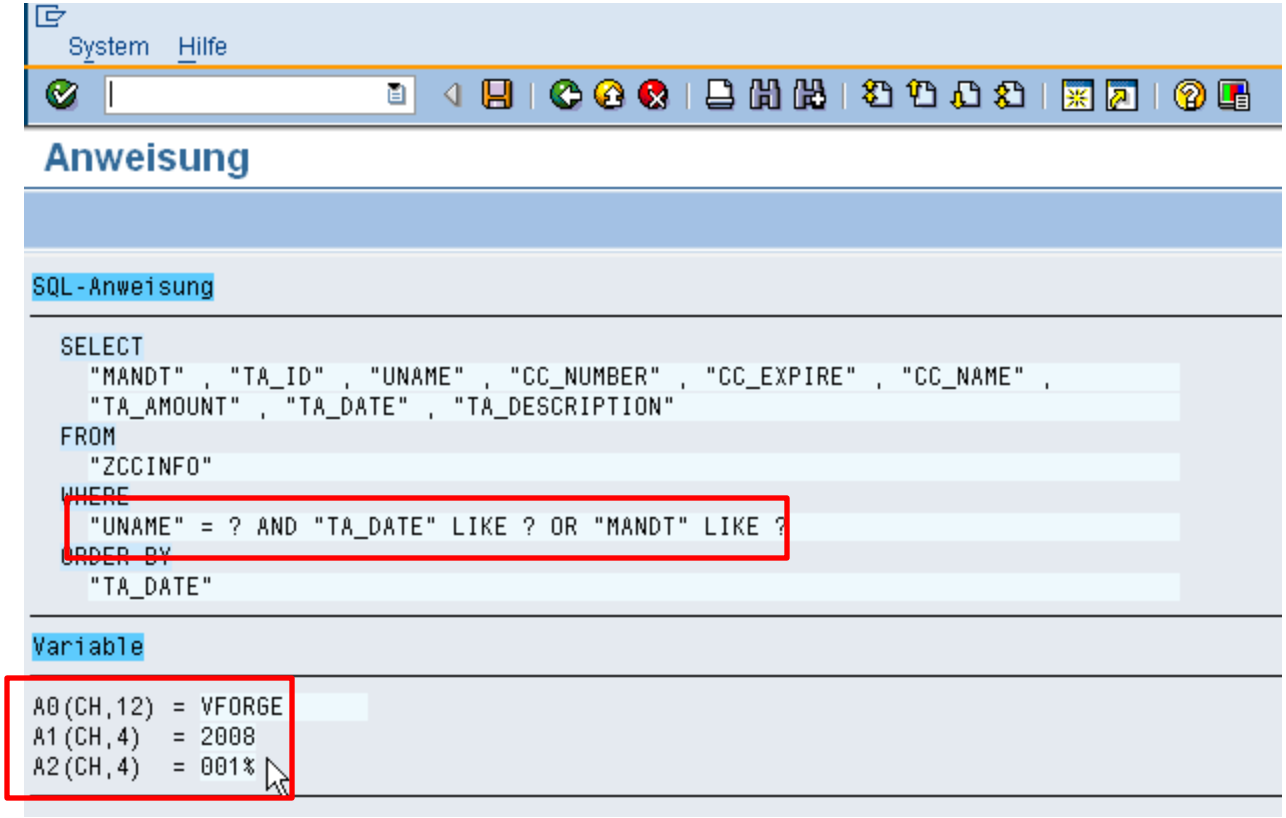
Dynamic Open SQL - Example



Dynamic Open SQL - Example



Dynamisches Open SQL



Summary Open SQL

- Dynamic Open SQL can easily lead to SQL-Injection-Vulnerabilities
  - No encoding functions
  - Prepared-Statement-Injection
- Avoid dynamic Open SQL in ABAP whenever possible

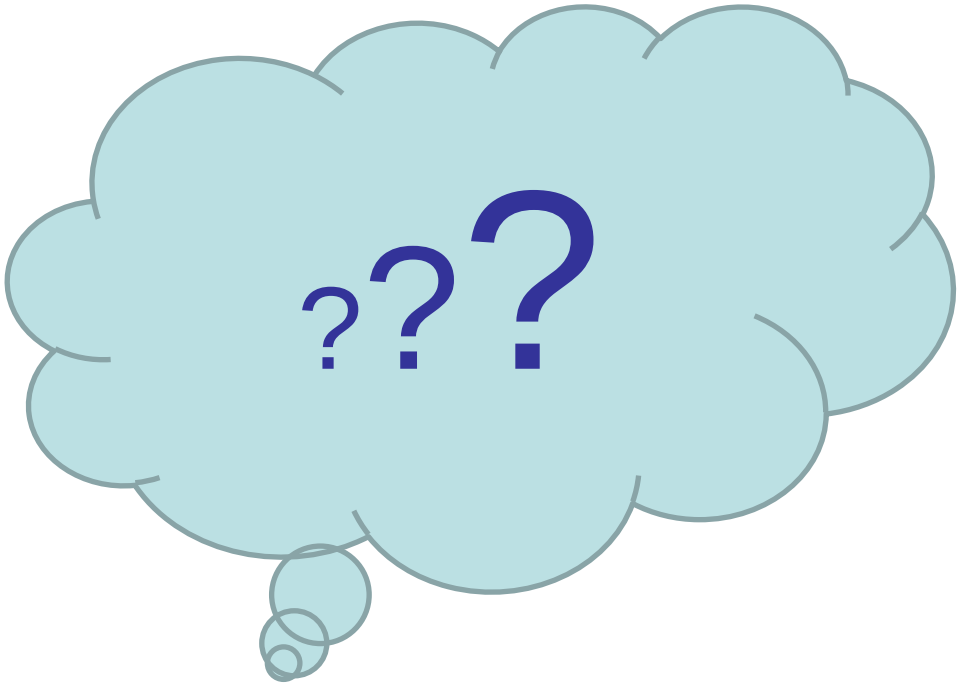
Conclusion: Covered Topics

OWASP Top 10	TODO
A1 - Cross Site Scripting (XSS)	☑
A2 - Injection Flaws	☑
A3 - Malicious File Execution	X
A4 - Insecure Direct Object Reference	X
A5 - Cross Site Request Forgery (CSRF)	X
A6 - Information Leakage and Improper Error Handling	-
A7 - Broken Authentication and Session Management	-
A8 - Insecure Cryptographic Storage	-
A9 - Insecure Communications	-
A10 - Failure to Restrict URL Access	-

Conclusion: Take Aways

- SAP-Web-Frontends as example
  - Widely used, processing of business-critical data
- SAP-Web-Frontend-Technologies covered in this talk:
  - ☑ **Business Server Pages (BSP)**
  - X Web Dynpro
  - X Internet Transaction Server
  - X Own HTTP-Handlers
  - ...
- High efforts for writing secure ABAP code!
  - Step 1: Understand how known vulnerabilities relate to SAP
  - Step 2: Understand what to do

Questions



Literature

- *“Sichere ABAP-Programmierung” - SAP Press*, 2009
  Wiegenstein, Schumacher, Schinzel, Weidemann
  http://www.sap-press.de/2037
- *“ SAP Documentation”* - http://help.sap.com/
- *“Secure Programming – ABAP” - SAP AG, 2004*
  http://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/17a4f828-0b01-0010-8da1-d18bb60ec2bf&overridelayout=true
- *“Security Scanner for ABAP”* - http://codeprofilers.com/
- *“vMovie: Security Knowledge on Stage”*
  http://secure-abap.de/media