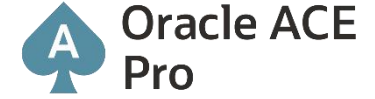


LET THE GAME BEGIN

SECURE PROGRAMMING IN APEX



Mirela Ardelean
Equip GmbH



Mirela Ardelean

@Mirela_RoOUG



.SYM L2



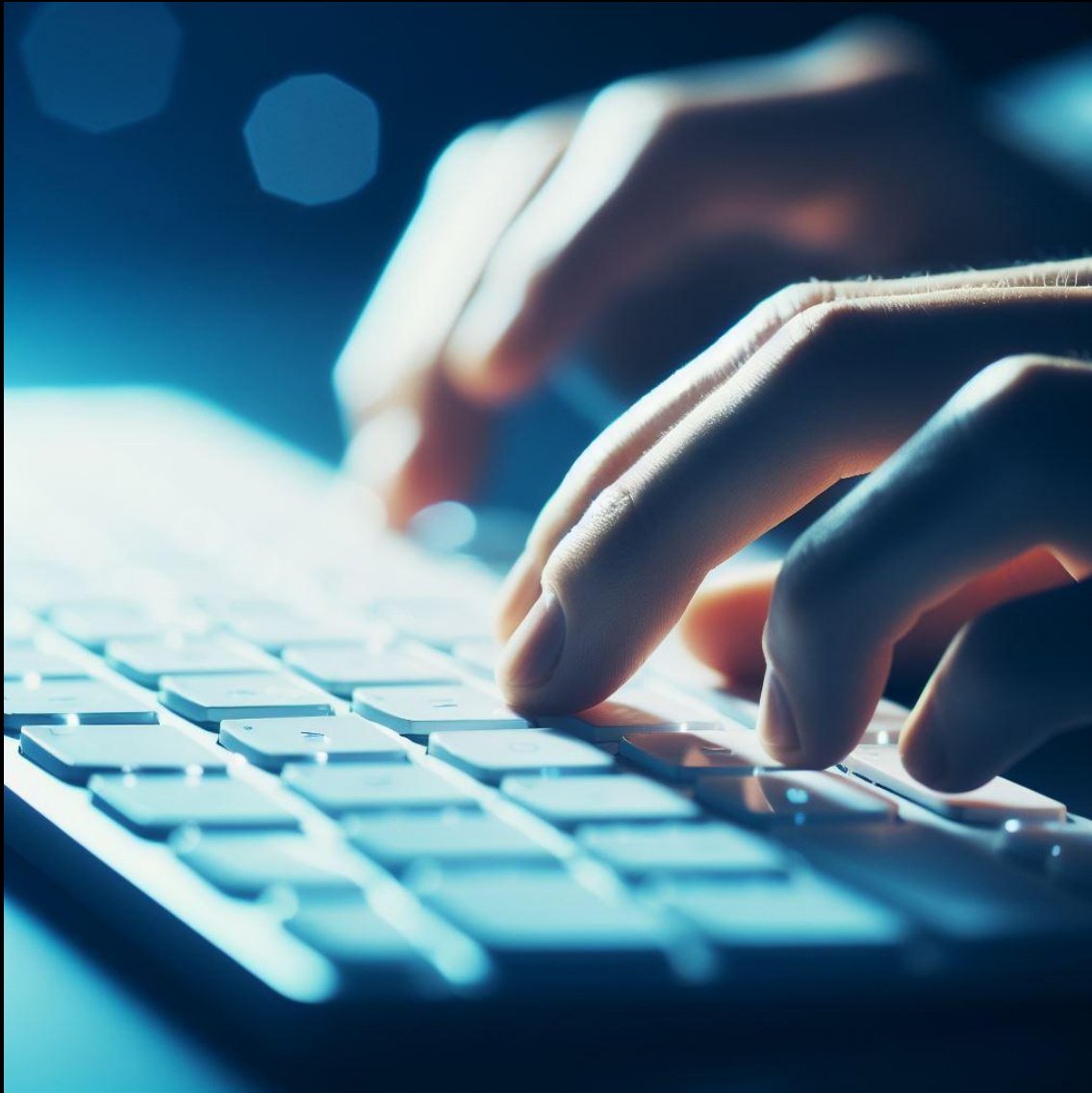


Mentor and Speaker Hub

Our goal is to *connect* speakers with mentors to assist in *preparing* technical sessions and *improving* presentation skills

Interested? Read more and get in touch

<https://mashprogram.wordpress.com>



„Administrators are primarily responsible for ensuring the security of the Oracle APEX installation and

developers

are responsible for

building secure applications”

67% of developers knowingly release applications with vulnerabilities

A person is sitting at a desk, looking at a computer monitor. The monitor displays a code editor with various lines of code. A window titled "secure programming" is overlaid on the screen. The person's hands are visible on the desk, and their head is in the foreground, looking towards the screen.

secure programming

a way of writing code in a software so that it is protected from all kinds of vulnerabilities, attacks or anything that can cause harm to the software or the system using it



Coding Practice

Input validation: Never Trust User Input

Coding Practice

Input validation: Never Trust User Input

Sanitize Data First, Then Send the Inputs to Other Systems

Coding Practice

Input validation: Never Trust User Input

Sanitize Data First, Then Send the Inputs to Other Systems

Adopt the Principle of Least Privilege

Coding Practice

Input validation: Never Trust User Input

Sanitize Data First, Then Send the Inputs to Other Systems

Adopt the Principle of Least Privilege

Deny Access by Default

Coding Practice

Input validation: Never Trust User Input

Sanitize Data First, Then Send the Inputs to Other Systems

Adopt the Principle of Least Privilege

Deny Access by Default

Error handling and logging

Coding Practice

Input validation: Never Trust User Input

Sanitize Data First, Then Send the Inputs to Other Systems

Adopt the Principle of Least Privilege

Deny Access by Default

Error handling and logging



ā'pěks
(#orclapex)

is
SAFE

Never Trust User Input

Display Only Item

The screenshot shows the configuration for a 'Page Item' with the following details:

- Page Item** (Title)
- Filter** (Search icon)
- Identification**
 - Name: P15_DISPLAY_1
 - Type: Display Only
- Label**
 - Label: Display 1 (text)
- Settings**
 - Format**: Plain Text (highlighted with a yellow circle)
 - Based On: Plain Text
 - Show Line Breaks: HTML
 - Markdown: Markdown
 - Send On Page Submit:

The screenshot shows the 'Security' configuration section with the following details:

- Security** (highlighted with a yellow circle)
- Authorization Scheme: - Select -
- Session State Protection: Unrestricted
- Store value encrypted in session state:
- Restricted Characters** (highlighted with a yellow circle): All characters can be saved.
- Help**
 - Allowlist for a-Z, 0-9 and space
 - Blocklist HTML command characters (<>)
 - Blocklist <>*/;|= % and --
 - Blocklist <>*/;|= % or -- and new line

Hidden Item

The screenshot shows the configuration for a 'Hidden' item with the following details:

- Identification**
 - Name: P14_HIDDEN
 - Type: Hidden
- Settings**
 - Value Protected:

The screenshot shows the 'Security' configuration section with the following details:

- Security** (highlighted with a yellow circle)
- Authorization Scheme: - Select -
- Session State Protection** (highlighted with a yellow circle): Unrestricted
- Store value encrypted in session state: Unrestricted
- Restricted Characters: Restricted - May not be set from browser
- Help**
 - Checksum Required - Application Level
 - Checksum Required - User Level
 - Checksum Required - Session Level

Never Trust User Input

AVOID substitution syntax `&<ITEM_NAME>.` in the code

USE

embed bind variable into the string

`APEX_UTIL.PREPARE_URL` or `APEX_PAGE.GET_URL`

Never Trust User Input

Robert');DROP TABLE Students;--



Sanitize Data First

Use:

APEX_ESCAPE

&ITEM!JS.

&ITEM!HTML.

&ITEM!RAW.

&ITEM!ATTR.

whitelist/blacklist

Markdown

Template Directives

DOMPurify

Principle of Least Privilege

Deny by default

Application 109953 \ Edit Security Attributes

Definition **Security** Globalization User Interface

Application 109953

Show All Authentication Authorization Session Management Session State Protection Browser Security Database Session

Authentication

Authentication is the process of establishing each user's identify before they can access your application. You may define multiple authentication schemes for an application. The authentication logic of the current scheme is used when your application is run.

Application **109953** ?

Public User ?

Authentication Scheme ?

Authorization

Application authorization schemes control access to all pages within an application. Unauthorized access to the application, regardless of which page is requested, is denied.

Authorization Scheme ?

Run on Background Job ?

Source for Role or Group Schemes ?

Error handling function

Application 186826

Show All Name Properties Availability **Error Handling** Global Notification Substitutions Build Options Report Printing Application File Storage Copyright

Error Handling

Default Error Display Location Inline with Field and in Notification ?

Error Handling Function ?

```
--l_text := 'LAST MESSAGE: &P9_COMMENT..' || utl_tcp.crlf;  
l_text := 'LAST MESSAGE: ' || :P9_COMMENT || utl_tcp.crlf;
```

```
apex_util.prepare_url('f?p=&APP_ID.:&APP_PAGE_ID.:&APP_SESSION.::: :P9_MSG:&P9_MSG.')
```

```
'SELECT empno,ename,sal FROM emp WHERE job = '' || :P9_INPUT4 || ''';
```

```
http.p('<h1>Ups sorry, ' || :P10_ITEM6 || '.</h1>');
```

```
alert("Entered value &P10_ITEM.");
```

Ajax call returned server error ORA-02292: integrity constraint (ROOUG_MAIN.FK_SUBSCRIBERS) violated - child record found for Execute Server-Side Code.

Is my code secure?



APEX Sert



APEX Project Eye



APEXSec

Select * from < APEX Views >

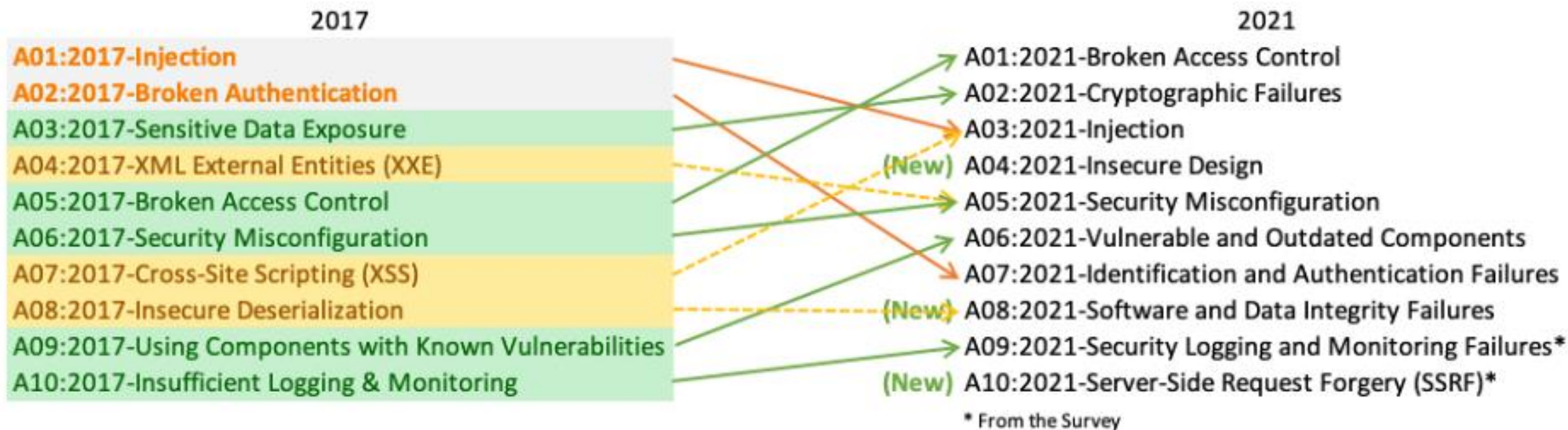
- **apex_applications**
- apex_application_processes
- apex_application_pages
- **apex_application_page_regions**
- **apex_application_page_proc**
- apex_application_page_items
- apex_application_page_buttons
- apex_application_page_reg_cols
- **apex_application_page_rpt_cols**

- apex_appl_page_igs
- **apex_appl_page_ig_columns**
- apex_application_page_ir
- **apex_application_page_ir_col**
- apex_application_page_da_acts
- apex_application_list_entries
- apex_application_lists
- apex_application_lovs

Resources

<https://owasp.org/www-project-top-ten/>

Top 10 Web Application Security Risks



Resources

https://security.web.cern.ch/recommendations/en/checklist_for_coders.shtml

General

- Think of the security of your software as a design goal.
- **Architecture:**
 - **Modular interface:** Design a modular interface between components.
 - **Isolation:** Isolate components from each other and from requests.
 - **Defense in depth:** Protect data and resources against all possible attacks.
 - **Simplicity:** Keep the design as simple as possible.

Design

- Make **security-sensitive** parts of your code **small**.
- **Least privilege principle:** Don't require more permissions than you need to run your code as the least privileged user necessary (e.g. use a non-root user flag). Make sure that the account on which you run your code has the minimum access and execute privileges that your code really needs with admin privileges from your software;
- Choose **safe defaults:** For example, a random password rather than a standard default password;
- **Deny by default:** For example, when validating user input, deny by default what you expect rather than trying to block known "bad" input;
- **Limit resource consumption,** to limit the likelihood of denial of service attack;
- **Fail securely:** For example, if there is a runtime error, assume the user has no rights, assume she has none;
- In distributed or Web applications **don't trust** user input, perform security checks or authentication on the server side; remember that HTTP response headers (e.g. referrer etc.) and HTTP query string values (from cookies, etc.) are forged/manipulated;
- Cryptography: **Use trusted, public algorithms**; don't invent your own cryptographic algorithms or protocols; reuse trusted code.

Coding

- **Don't trust input data:** Data coming from potentially malicious users is the single most common reason of security-related incidents (buffer overflow, SQL injection, Cross Site Scripting (XSS), code inside data etc.). Input data includes command-line arguments, configuration files (if accessible by not-trusted users), environment variables, cookies and POST/GET arguments etc.;
- **Validate all input data:** Consider all input dangerous until proven valid, deny by default if not sure, validate at different levels, for example at input data entry point and before really using that data;
- **Don't make any assumptions about the environment:** make sure your code doesn't break with modified/malicious PATH, CLASSPATH and others environment variables, current directory, @INC Perl variable, umask, signals, open file descriptors etc.;
- **Beware of race condition:** Can your code run parallel? what if someone executes two instances of your program at the same time, or changes environment in the middle of its execution?
- **Deal with errors and exceptions:** Don't assume that everything will work (especially file operations, system and network calls), catch exceptions, check result codes; don't display internal error messages, failing SQL query, stack trace etc.;
- **Fail gracefully:** If there is an unexpected error that you can't recover from, then log details, alert the administrator, clean the system (delete temporary files, clear memory) and inform the user;
- **Protect passwords and secret information:** don't hard-code passwords (it's hard to change them and easy to disclose), use external files instead (possibly encrypted) or already existing credentials (like certificates), or simply ask the user for the password;
- **Be careful when handling files:** If you want to create it, report an error if it already exists; when you create it, set file permissions; if you open a file to read data, don't ask for write access; check if the file you open is not a link with the lstat() function (before and after opening the file); use absolute pathnames (for both commands and files); be extra careful when the filename (or part of it) comes from a user;
- **Temporary files:** Avoid them whenever possible. Pipes are a safer and more efficient way of communicating information between processes. If you really need temporary files, don't fall for the symbolic link attack (someone guesses the name of your temporary file, and creates a link from it to another file e.g. /bin/bash, that your program overwrites). Temporary files must have unique names that are hard to guess! (use tmpfile() for C/C++, mktemp shell command etc.);
- **Be careful with shell calls, eval functions etc.:** Such functions evaluate the string argument as code and interpret it, or run it on the shell. If an attacker managed to inject malicious input to that argument, you're executing his code.

General

- Think of the security implications of what your code does;

Assume nothing

Believe no one

Confirm everything



ā'pěks
(#orclapex)

is
SAFE

What questions do you have?



