20 Ways To Secure ColdFusion Applications

CFSummit Las Vegas 2024

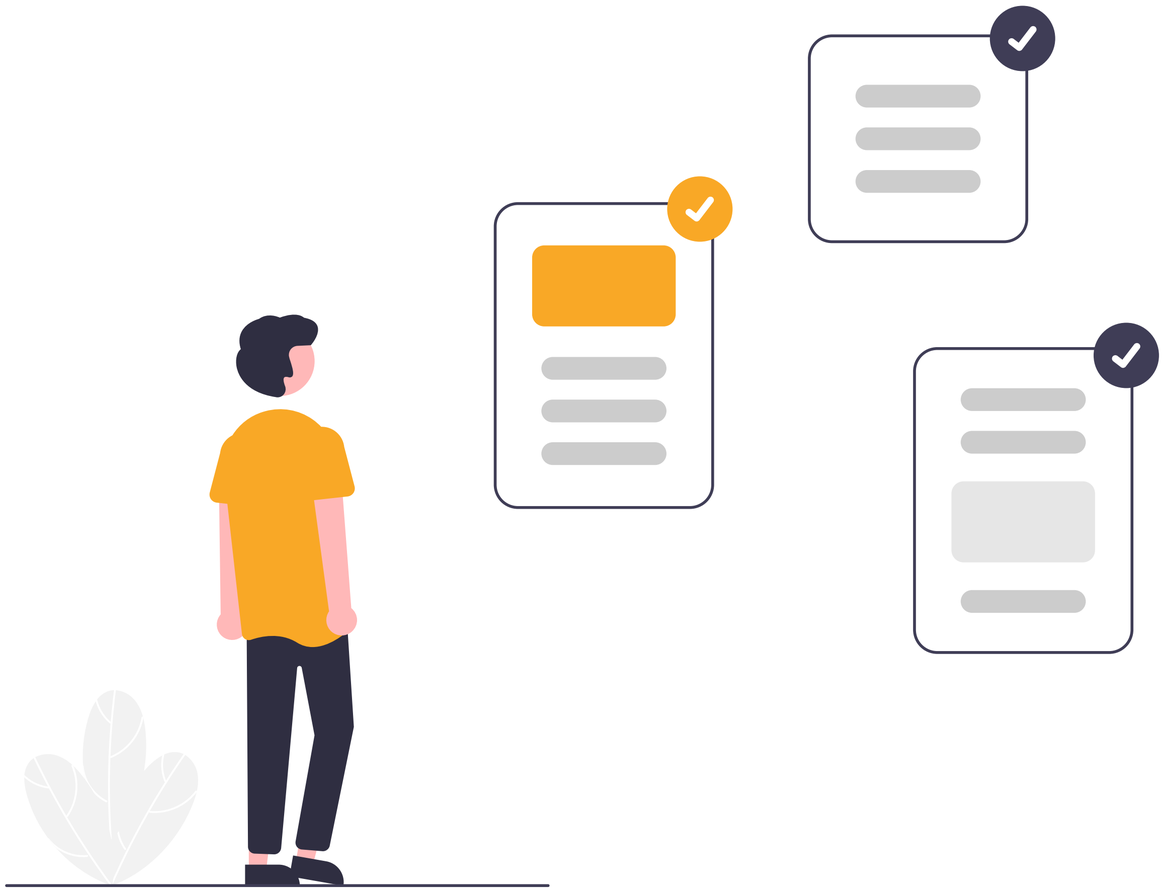
Pete Freitag, Foundeo Inc.

Hi, I'm Pete

- Working with ColdFusion for more than a quarter century
- Foundeo Inc. - sponsor of ColdFusion Summit
  - Fixinator - CF code security scanner
  - FuseGuard - web app firewall for CF
  - HackMyCF - CF server security scanner
- Other Stuff
  - cfbreak.com - weekly ColdFusion newsletter
  - cfscript.me, cfdocs.org, lockdown guides

Agenda

- 20 Tips
- No particular order
- Coding
- Tooling
- Admin



#1 - Block Remote CFC

- Check your CFC functions:
  - Tag <cffunction **access="remote"**>
  - Script: **remote** function x() {}
  - No remote functions? then block .cfc file extension
  - IIS: Request Filtering, Extensions, Deny File Extension
- Note: I'm not saying don't use CFCs

#1 - Block File Extensions

Example IIS web.config

```
<configuration>
  <system.webServer>
    <security>
      <requestFiltering>
        <fileExtensions>
          <remove fileExtension=".cfc"/>
        </fileExtensions>
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>
```

#2 - Add More Validation

- How often do you hear the word “unchecked” in a security vulnerability?
- Check variables before using them

```
<cfparam name="url.some_id" type="integer" default="0">

<cfif isValid("integer", url.some_id)> ... </cfif>

<cfif isNumeric(url.some_id)>...</cfif>
```

#2 - Add More Validation

- A simple cfparam prevents three vulnerabilities:

```
<cfparam name="url.some_id" type="integer" default="0">
...
<cfquery>
  SELECT things
  FROM somewhere
  WHERE some_id = #url.some_id#
</cfquery>
...
<cfset other_stuff = fileRead("../stuff/#url.some_id#.json")>
...
<cfset x = evaluate("checkbox_#form.some_id#")>
```

#2 - Add More Validation

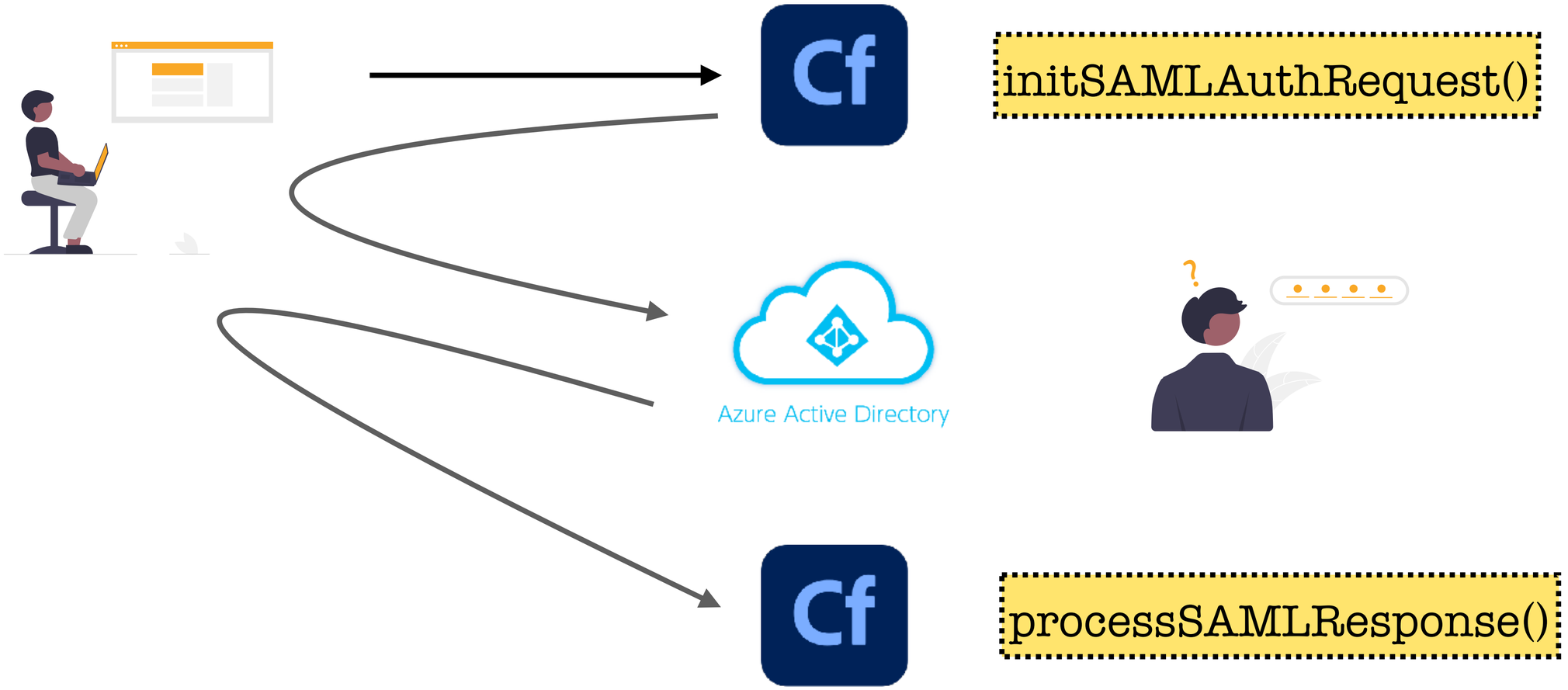
- The int() throws exceptions if input is not an integer
- The val() returns zero for non numeric values:

```
<cfset other_stuff = fileRead("../stuff/#int(url.some_id)#.json")>
...
<cfset x = evaluate("checkbox_#int(form.some_id)#")>
```

#3 - Outsource Authentication

- If possible leverage an identity provider
  - Users prefer single sign on
  - Reduces the attack surface of your app:
    - No more storing passwords
    - Forgot passwords
    - Dealing with password spraying attacks
- CF2021+ has SAML functions
- CF2025 - MS Graph API

#3 - Outsource Authentication



#4 - Add Audit Logging

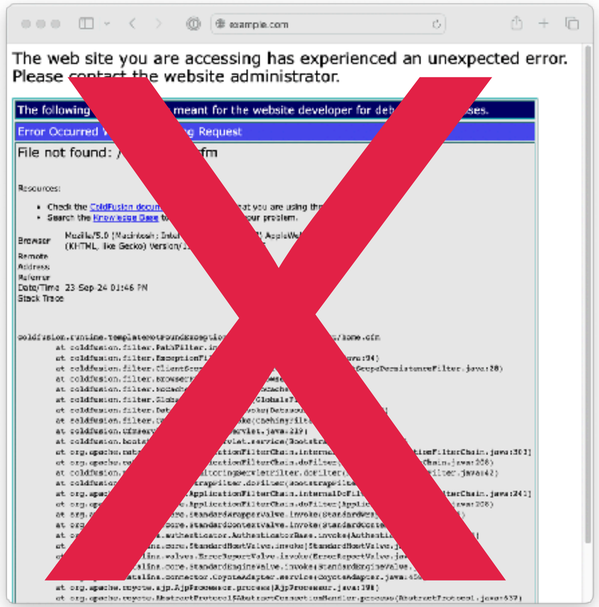
- Not all attacks are successful on the first try
- Log authentication, authorization or other security failures
- Block IP / user if too many failures within X minutes

#4 - Add Audit Logging

- What goes in the audit log?
  - IP address
  - Timestamp
  - Event Type (login, login_failure, forgot_password_request, etc)
  - User ID (if applicable)
  - User Agent

#5 - Add onError()

In your Application.cfc




```
public void function onError(exception) {
    writeOutput("Oops");
    logError(exception);
}
```

#6 - File Uploads

- Always check file extension
    - Use a list of allowed types
- Don't rely on the mime type in accept attribute
    - Hint you can use extensions in accept attribute instead
- Upload destination always outside the web root
    - Copy to web root only after validation and verification of file.
    - Tip: use `getTempDirectory()`

#7 - Encode Outputs

Use the correct context to prevent XSS

Context	Method	Example
HTML	`encodeForHTML()`	#encodeForHTML(url.name)#
HTML Attribute	`encodeForHTMLAttribute()`	<input value="#encodeForHTMLAttribute(url.name)#">
JavaScript	`encodeForJavaScript()`	<script>x = "#encodeForJavaScript(url.x)#";</script> <button onclick="doIt("#encodeForJavaScript(url.x)#")">
CSS	`encodeForCSS()`	<style>p { color: #encodeForCSS(url.color)#; }</style>
URL	`encodeForURL()`	go

#7 - Encode Outputs

What Happens Here?

```
<cfoutput encodefor="html">
    Hi #qry.name#
    <script>
      var name = "#encodeForJavaScript(qry.name)#";
    </script>
</cfoutput>
```

#7 - Encode Outputs

What Happens Here?

encodeForHTML(qry.name)

```
<cfoutput encodefor="html">
  Hi #qry.name#
  <script>
    var name = "#encodeForJavaScript(qry.name)#";
  </script>
</cfoutput>
```

encodeForJavaScript(qry.name)

Does not double encode on ACF

#8 - Content Security Policy

(CSP)

- Response header or meta tag:
  - Defines allowed scripts, css, images, etc.
- Can be tricky if you use tags like cfform, cflayout, etc
- Reference: content-security-policy.com

CSP Demo

#9 - PDF Injection

- Variables in PDF files generated with cfdocument, or cfhtmltopdf need to be encoded too.
  - Use encodeForHTML, etc.
  - Unprotected variables can lead to server side issues such as SSRF.

PDF Demo

#10 - Avoiding SSRF

Server Side Request Forgeries

- Many Tags / Functions can make network requests (cfhttp, cffeed, cfdocument, cfhtmltopdf, xmlParse, etc)
    - Avoid allowing untrusted inputs
- Guide: https://foundeo.com/security/guide/server-side-request-forgery/
    - See Brian Reilly's list of tags / functions that may allow SSRF

#11 - RCE via Evaluate

- Avoid the Evaluate Function!
- RCE Evaluate Guide: https://foundeo.com/security/guide/evaluate/

```
evaluate("form.#name#")
```

Vulnerable!

```
form[name]
```

Safe

#12 - RCE via IIF

- Avoid the IIF function!
- RCE IIF Guide: https://foundeo.com/security/guide/iif/

```
iif( len(name), de(name), de("Anonymous") )
```

```
len(name) ? name : "Anonymous"
```

IIF Demo

#13 - SQL Injection

```
<cfquery>
  SELECT x FROM y
  WHERE id = #url.id#
</cfquery>
```

```
<cfquery>
  SELECT x FROM y
  WHERE id = <cfqueryparam value="#url.id#">
</cfquery>
```

#13 - SQL Injection

```
queryExecute("
  SELECT x FROM y
  WHERE id = #url.id#
");
```

```
queryExecute("
  SELECT x FROM y
  WHERE id = :id
", { id: url.id } );
```

#13 - SQL Injection

- You can use **`cfsqltype="integer"`** now instead of **`cfsqltype="cf_sql_integer"`**
- Get to know cfqueryparam: https://www.petefreitag.com/blog/mastering-cfqueryparam/
- SQL Injection Guide: https://foundeo.com/security/guide/sql-injection/

#14 - Use CI

- A script that runs whenever you commit code
  - Run checks on your code for security (e.g. fixinator), syntax, etc.
  - Functional testing to avoid errors
  - Add security checks to your unit tests (esp. authorization checks)
    - "Test that user A cannot read record owned by user B"
    - "Test that user A cannot delete record owned by user B"
- My CI Talk: https://www.petefreitag.com/blog/better-cfml-code-ci/
- Example CI: https://github.com/foundeo/fixinator/wiki/Running-Fixinator-on-GitLab

#15 - XML Entity Injection

```
<?xml version="1.0" ?>
<!DOCTYPE d [
   <!ENTITY xxe SYSTEM "http://httpbin.org/uuid">]>
<tag>&xxe;</tag>
```

XXE Demo

#15 - XML Entity Injection

The Billion Laughs Attack

```
<!DOCTYPE root [
<!ELEMENT root ANY>
<!ENTITY LOL "LOL">
<!ENTITY LOL1 "&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;">
<!ENTITY LOL2 "&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;">
<!ENTITY LOL3 "&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;">
<!ENTITY LOL4 "&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;">
<!ENTITY LOL5 "&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;">
<!ENTITY LOL6 "&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;">
<!ENTITY LOL7 "&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;">
<!ENTITY LOL8 "&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;">
<!ENTITY LOL9 "&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;">
]>
<root>&LOL9;</root>
```

#15 - XML Entity Injection

xmlParse(xml, false, **{allowExternalEntities: false}**);

isXML(xml, **{allowExternalEntities: false}**)

More Info: XXE Guide: https://foundeo.com/security/guide/xml-external-entities/

#16 - Clear-Site-Data

- HTTP Response Header
  - "cookies" - Remove all Cookies for the site
  - "storage" - clear localStorage / sessionStorage
  - "cache" - clears browser cache related to the site
  - * - all of the above
- clear-site-data.com

```
<cfheader name="Clear-Site-Data" value="*">
```

Clear-Site-Data Demo

#17 - Avoid End of Life (EOL) Versions

Version	Released	End of Core Support	End of Extended Support
ColdFusion 8	2007-07-30	2012-07-31	2014-07-31
ColdFusion 9	2009-10-05	2014-12-31	2016-12-31
ColdFusion 10	2012-05-15	2017-05-16	2019-05-16
ColdFusion 11	2014-04-29	2019-04-30	2021-04-30
ColdFusion 2016	2016-02-16	2021-02-17	2022-02-17
ColdFusion 2018	2018-07-12	2023-07-13	2024-07-13
ColdFusion 2021	2020-11-11	2025-11-10	2026-11-10
ColdFusion 2023	2023-05-17	2028-05-16	2029-05-16

Source: https://helpx.adobe.com/support/programs/eol-matrix.html

1 Year Away	< 2 years ago	3-5 years ago	5+ years ago

#18 - Block File Extensions

- Allow uploads?
  - Ensure your web server blocks cfm, cfc, cfml, etc
  - Or better yet: only serves jpg, png, gif, pdf files
  - IIS: Use Request Filtering

#18 - Block File Extensions

Example IIS web.config

```
<configuration>
  <system.webServer>
    <security>
      <requestFiltering>
        <fileExtensions allowUnlisted="false">
          <add fileExtension=".jpg"/>
          <add fileExtension=".png"/>
        </fileExtensions>
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>
```

#19 - Path Traversals

- Never use untrusted variables in file paths.
- Look at all file tags / functions (there are many)

```
<cfinclude template="html/#url.file_name#">
```

example.cfm?**file_name=../../config/secrets.txt**

#20 - Tooling

Lots of security tools, here's my tools

- **Fixinator** - ColdFusion code security scanner.
    - 6 releases in 2024, more on the way
- **FuseGuard** - web app firewall for ColdFusion
    - Blocks malicious requests before it hits your CF code
- **HackMyCF** - ColdFusion server security scanner
    - Monitor CF, Java, Tomcat for updates & misconfig

Questions?

Thank You!

Feel free to email me, or visit my booth
pete@foundeo.com

Try Fixinator: fixinator.app/try/