security-assessment.com

Secure Development

Things the OWASP Guide Didn't Tell You

OWASP NZ Day 2011

Member of Datacraft Asia

Who am I?

security-assessment.com

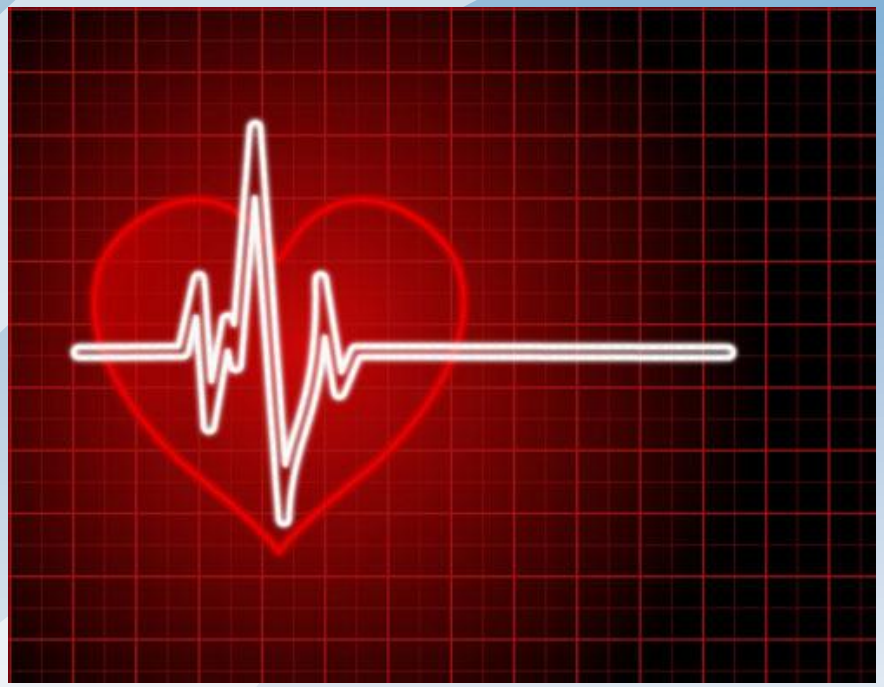
- I'm Blair Strang
- @ Security-Assessment.com
- Security Consultant
- Developer at large
- Third-party code reviews

The Goal

- To show the difference between “strong” and “outstanding” web applications (from a security point of view)
- Explain what’s missing from the OWASP Secure Development Guide to help you make your project more secure
- In particular, explain that not all the controls you might want are covered by OWASP (Yet, anyway)

Not Covered In This Talk

- **Policy Framework**
- **Security Buy-in from Management**
- **Methodology & SDLC**
- **Coding Standards**



From a developer's point of view,
These Are Life Support

Definition: Point Fix

- **Something you have to always remember to do (usually add) to your code to avoid a security problem**

- **XSS Example**
  - Response.Write(anything_untrusted) ← Needs HtmlEncode

- **LDAP Example**
  - ldapObj.DN = "ou=people,dc=spilab,dc=com“
  - ldapObj.SearchFilter = filter ← Needs LDAP Escaping

Juggling Chainsaws

security-assessment.com

- One time, I watched a dwarf juggle a chainsaw and a bowling ball while standing on a Swiss ball
- Point fixes everywhere remind me of this
- There's got to be an easier way to make a living…

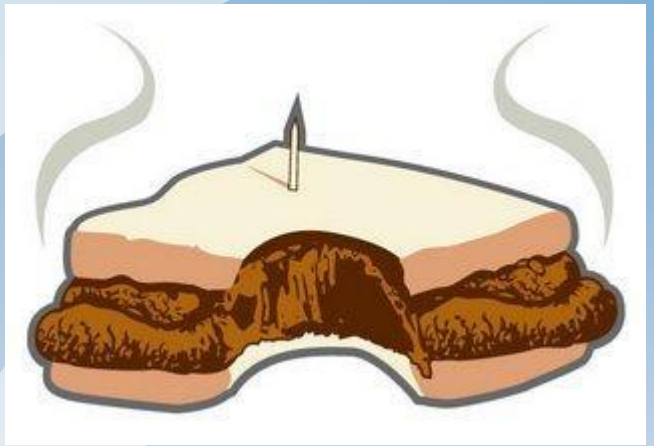
Juggling Chainsaws

security-assessment.com

- Demo
- Code search is great for this
- Let's watch people drop the chainsaw

Point Fixes

- What happened here?
- Someone handed the developer a crap sandwich (of an API)
- They ate it
- WHY?????



Metaphor switch in progress

Developers!

- The super powers of software developers are automation and abstraction
- Don’t eat the crap sandwich (more than you have to)
- If you have to apply a point fix more than a dozen times, STOP DOING THAT

Developers!

- Choose a better library, API or framework
- Wrap the API yourself

Developers!

- **Choose a better library, API or framework**
- **Choose templates that output encode by default**
  - For example, see RoR
- **Choose parameterised anything (separates metadata from data)**
  - You know this one

Developers!

- **Wrap the API yourself**
- **Encapsulate LDAP calls into:**
  - xSafeLDAPQuery() / xUnsafeLDAPQuery()
  - Accept e.g, key/value pairs, filter or encode by default

Developers!

- **Input Validation (the oddest point-fix)**
- **Choose a framework which /forces/ you to specify an input validation expression or function before you can access input**
  - safeVal = SafeInput.xQueryString(phone_no, my_phone_no_regex, INVALID_PHONE_NO)

Developers!

- Fix The Problem
- Go up a level, and fix the problem that causes the problem
- Often, the problem that causes the problem is unsafe APIs

Strong vs Outstanding Projects

Strong

Outstanding

During the course of code reviews…

Strong Projects

- Used their framework well
- Addressed the OWASP top 10
- Had conventions for input validation
- Applied point fixes consistently

Strong Projects

- Used their framework well
- Addressed the OWASP top 10
- Had conventions for input validation
- Applied point fixes consistently

This is enough to get your project past pen-testing

But it's not enough to ensure a truly resilient solution

Outstanding Projects

- **Were strategic as well:**
- **Minimised point-fixes**
- **Thought about their security requirements**
- **Implemented appropriate controls, based on requirements**

Designed to handle failure, too
(Unknown Unknowns)

Example A

One project, concerned mainly about <u>transaction integrity</u>:

- **Made the web application a client of a transaction service**
- **Implemented extra logging, access controls, and did those *well***
- **Fraud could still happen, but goal was a strong audit trail**

Once upon a time I would have said this was obvious and easy.

That would be before I saw so many other projects get this wrong

Example B

security-assessment.com

One project, concerned mainly about <u>privacy of user information</u>:

- **Public-key encrypted user data submissions**
- **Kept the key *offline***
- **Bit of a special case, not perfect, but compared to the norm, it was Outstanding**

Sure, everyone claims to care about your privacy.

Just not enough to spend budget or design effort on it….

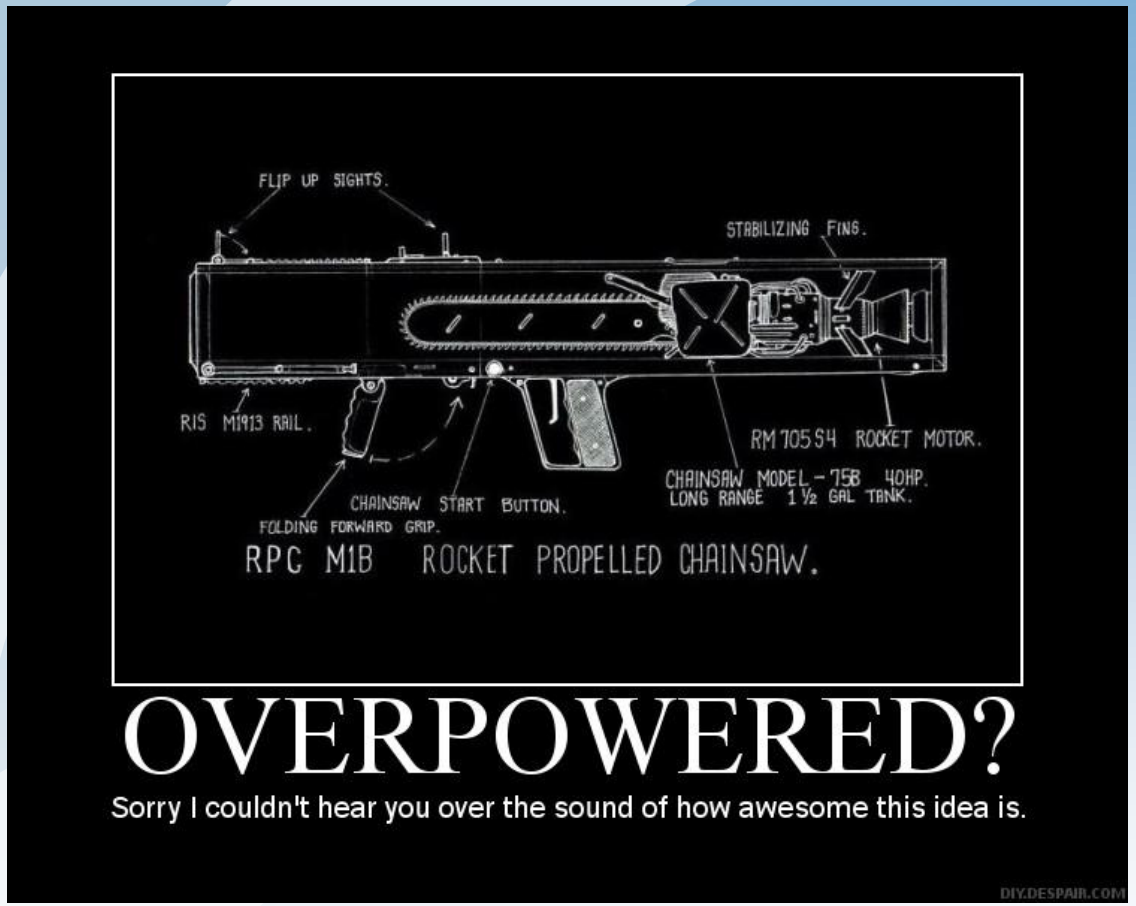
Outstanding Projects

- Designed to handle failures
- Which could be reasonably anticipated
- Didn't solve all the problems – spent effort on the most important

Why try to be Outstanding ?

security-assessment.com

- Isn't that more work?
- Not sure if we have the budget for that…
- You still have to make the cost/benefit trade-off
- Have to sell it to team
- But …



(Pic Unrelated)

Why try to be Outstanding ?

- Let's just step back for a minute

PBS

HBGary

citibank®

LOCKHEED MARTIN

SONY

twitter

FOX NEWS channel

Obligatory logo montage!

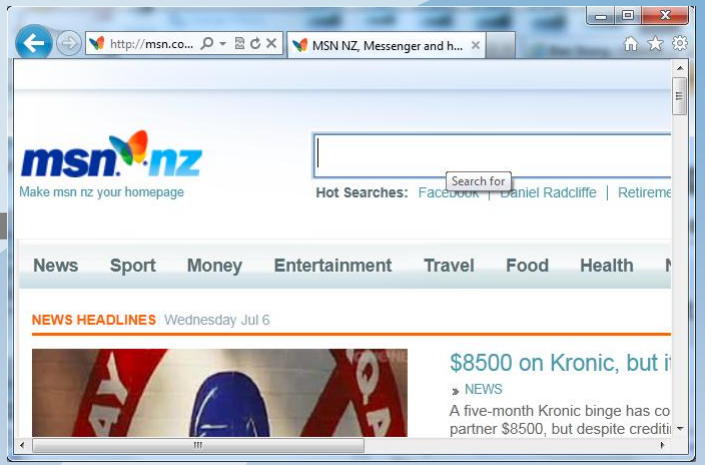
The Problem (server side)

security-assessment.com

- Most web developers are using at least 3 languages at the same time
- While trying not to drop the chainsaw
- Perched on top of a rickety stack of APIs, with unknown bugs
- Not to mention bugs or incorrect configuration of your server software
- Also, the server is likely in an outsourced datacentre, shared hosting, cloud

The Problem (protocols/network)

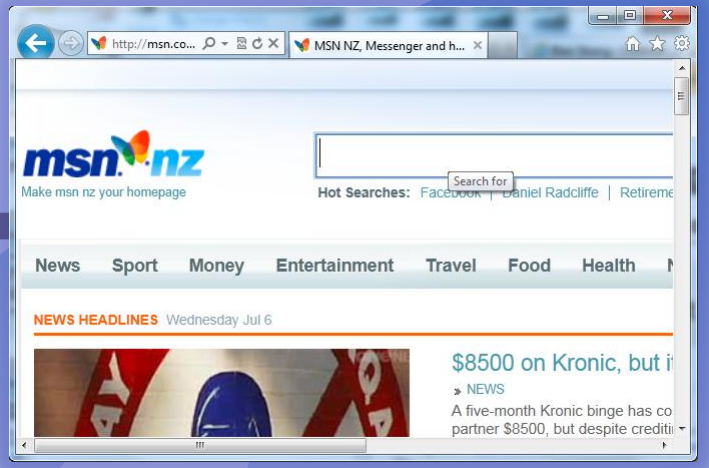
security-assessment.com



- HTTP was not really designed for your AJAX Web 3.0 cat social network
- Most of your traffic might as well be written on back of postcard (oh geez, Wi-Fi?)
- Your crypto certs are issued by the lowest bidder
  - Even though the support line is in Outer Elbonia
- For all you know, the Tubes might be routing your users' traffic through China

The Problem (client)

security-assessment.com



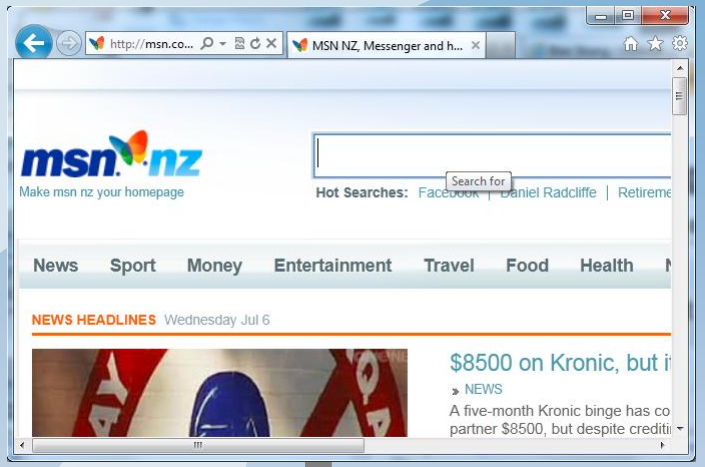
- Your user interface is running in a process you don't control
- Which has multiple unpatched software vulnerabilities
- You're probably hot loading third-party JavaScript into your page context
- On a host OS which, statistically speaking, is definitely owned (for some %users)

The Problem (users)

security-assessment.com

But it gets worse!

- Most of your users can't even tell if they're using your site or not
- They'll click on anything
- They'll type their password into anything
- Notice I said 'password', singular (they also use that password for Internet banking, and cat forums)



The Problem (You and Me)

- **This is BEFORE:**
  - You've had a chance to screw up the business logic - or drop the chainsaw
- **You're telling me you want to deal with:**
  - Money? Something convertible to money?
  - Personal information? Medical records?
  - Intellectual property? Business intelligence?
- **With *that* stack?**
  - Do you really value that stuff, or just say you do?

Reality

- **Something will go wrong, eventually**
- **Maybe more than one thing**

- **Outstanding projects actively plan for that**
- **You can buy off a lot of risk**
- **But *sometimes* the technical fix is cheaper**

Outstanding Projects

security-assessment.com

- **Did three extra things:**
- **Minimised point-fixes**
- **Thought about their security requirements**
- **Implemented appropriate controls, based on requirements**

Would be nice to have more guidance available on useful security controls

Where OWASP Could Use Help

security-assessment.com

Reference

- How To...
- Principles
- Threat Agents
- Attacks
- Vulnerabilities
- Controls
- Activities
- Technologies
- Glossary
- Code Snippets
- .NET Project
- Java Project

Attacks – 62

Vulnerabilities – 164

Controls – 43

Where OWASP Could Use Help

security-assessment.com

Pages in category "Control"

The following 43 pages are in this category, out of 43 total.

.

- .Net CSRF Guard

A

- Address space layout randomization (ASLR)
- Authentication

B

- Blocking Brute Force Attacks
- Bounds Checking
- Business Justification for Application Security Assessment
- Bytecode obfuscation

C

- Canonicalization
- Clickjacking Protection for Java EE
- Concurrency
- Controls
- Cryptography

E

- Encoding
- Encryption
- Error handling

E cont.

- Executable space protection

H

- HTML Entity Encoding
- Hashing
- History Isnt Always Pretty
- How to protect sensitive data in URL's

I

- Identity Management
- Input Validation
- Intrusion Detection
- Intrusion Prevention

L

- Logging

M

- Memory Management

O

- Output Validation

P

- PDF Attack Filter for Apache mod rewrite
- PDF Attack Filter for Java EE
- Parameterized Command Interface

P cont.

- Protecting code archives with digital signatures

Q

- Quotas

R

- Randomization
- Resource Locking

S

- SSL
- Safe Libraries
- Session Fixation Protection
- Session management
- Signing jar files with jarsigner
- Stack-smashing Protection (SSP)
- Static Code Analysis

T

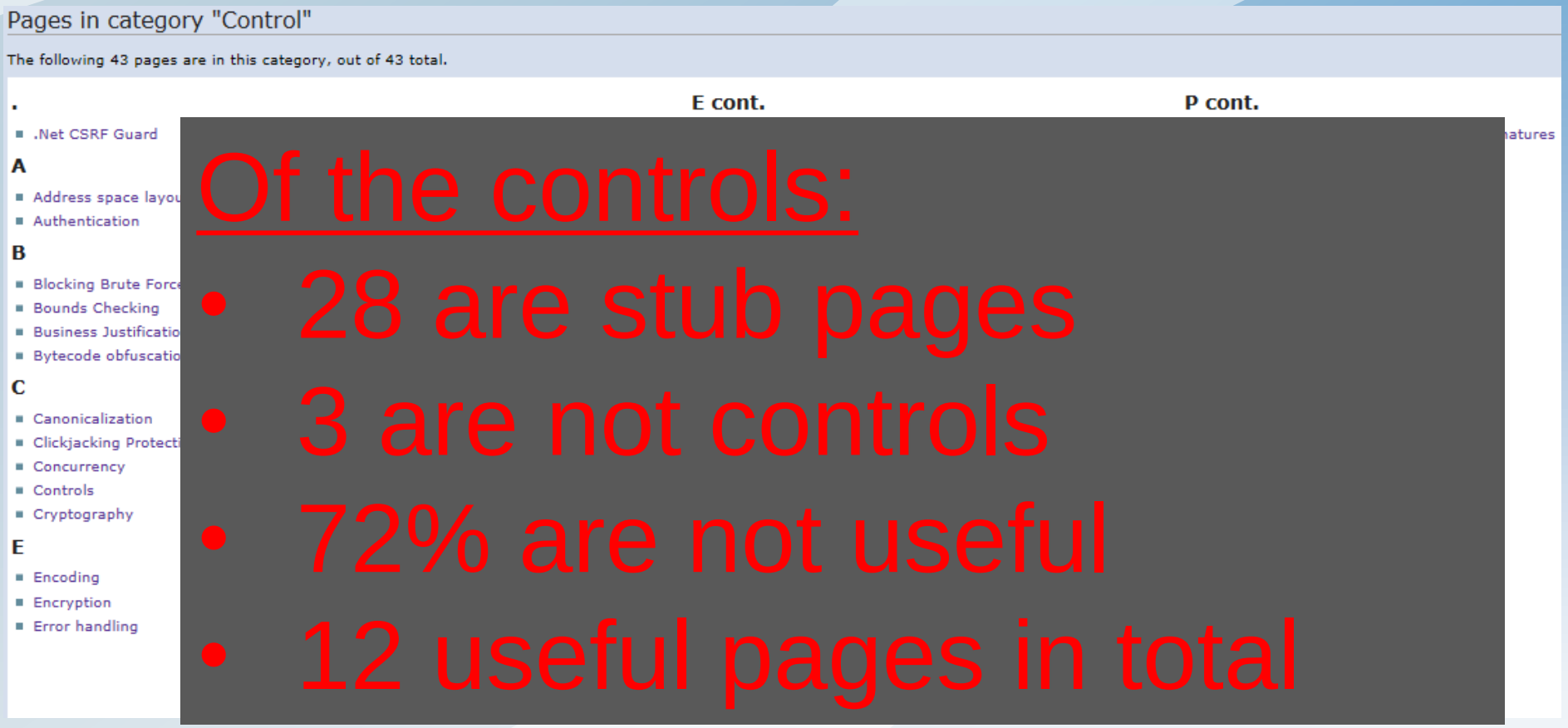
- Tokenizing

W

- Web Application Firewall

Where OWASP Could Use Help

security-assessment.com



OWASP Conclusion

security-assessment.com

- **OWASP Development Guide has:**
  - Good coverage of web application vulnerabilities, attacks
  - Good explanations of how to ‘point fix’ flaws
  - Reasonable discussion of threat/risk modelling

- **OWASP Development Guide does not have:**
  - Useful information on controls
  - Discussion of higher level security design patterns
  - This is what OWASP isn’t telling you (yet)

Examples of Controls

Let me give you some examples of what I consider to be security controls, some common, some not

- **Compartments / Tiers**
- **Honey Tokens**
- **Encrypted Object References**
- **User Visible Account History**

Compartments / Sandboxing

Motivation:

You want to isolate high-risk areas of your application.

Examples:

IE9 puts rendering in a low-integrity process

Pros:

Failure is contained, two things have to go wrong

Cons:

More overheads, isolation is never complete

Note: When applied to web applications, the 'site' is high risk ;)

Compartments: Stored Procs

security-assessment.com

Motivation:

You want to compartmentalize; this is an implementation pattern

Description:

All database access through stored procs

Procs implement access controls and auditing

Pros:

Low overhead compared with many solutions

Relatively simple

Cons:

Works better with staff DBA

Doesn’t play nicely with many frameworks (e.g, ROR)

Compartments: Service Layer

Motivation:

You want to compartmentalize; this is an implementation pattern

Examples:

Your web-site is effectively a rendering front end for a web service

Pros:

Can provide very strong isolation, arbitrary API

Cons:

Now you have two problems ;) (more tractable though)

Heavyweight, performance, more code

Honey Tokens

Motivation:

Detect data loss or disclosure

Description:

Put magic strings in source code, plant interestingly named files, have 'sentinel' database rows

Pros:

You get early warning of intrusions

You can make that expensive IDS do something useful for once

Cons:

Requires a potentially expensive IDS

User Visible Account History

security-assessment.com

Motivation:

Improve security experience for users. Allow *users* to know if their account has been subject to unauthorised access.

Description:

Allow users to access details of their account history (password changes, last login times and IP addresses) from the UI

Pros:

Users are more likely to notice inconsistencies

Can act as an early warning system for something you missed

Cons:

It's another feature you could get wrong, extra effort

Potential support issues

Some Conclusions

security-assessment.com

- Avoiding point-fixes will help you A LOT
- Implementing appropriate controls in your application can make it much stronger. There's room to be creative here.
- Work BOTH tactically AND strategically.
- **If you really value that data, work to make your application security Outstanding.**