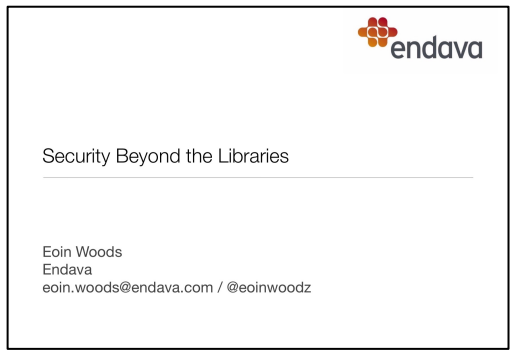
BUILDING APPLICATIONS SECURELY

Eoin Woods
@eoinwoodz | www.eoinwoods.info
careers.endava.com
Software Architecture Summit
Bucharest, May 2021

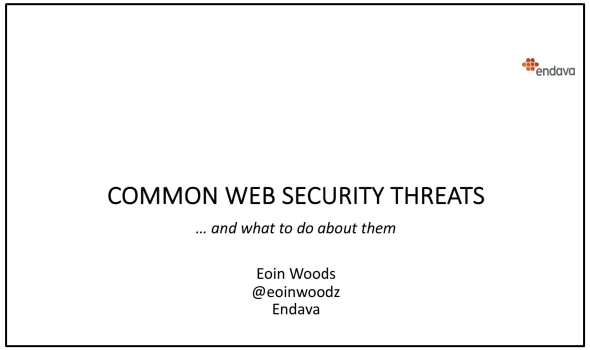
EOIN WOODS

- Endava's CTO, based in London (6 years)
  - 10+ years in products - Bull, Sybase, InterTrust
  - 10 years in capital markets - UBS and BGI
- Software engineer, architect, now CTO
- Long time security dabbler concerned at increasing cyber threats to systems
- Author, editor, speaker, community guy

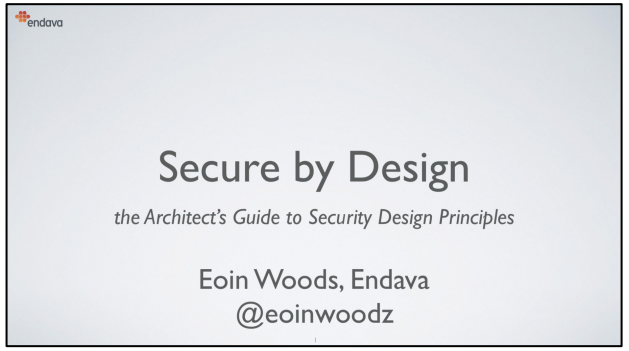
CONTEXT OF THIS TALK



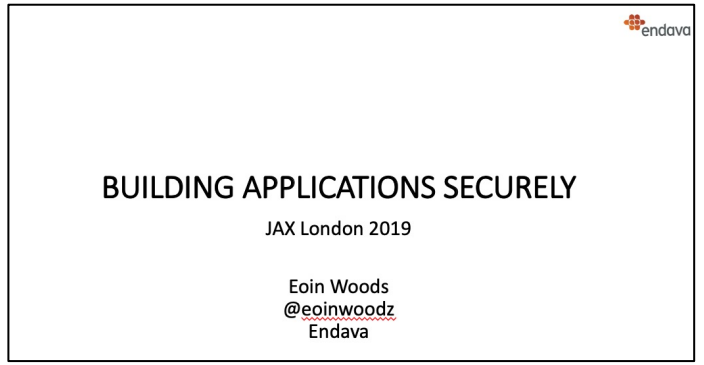
link



link



link



Agenda

1. The Threat
2. Mitigation via Software Security
3. Principles for Secure Implementation
4. Implementation Guidelines
5. Summary

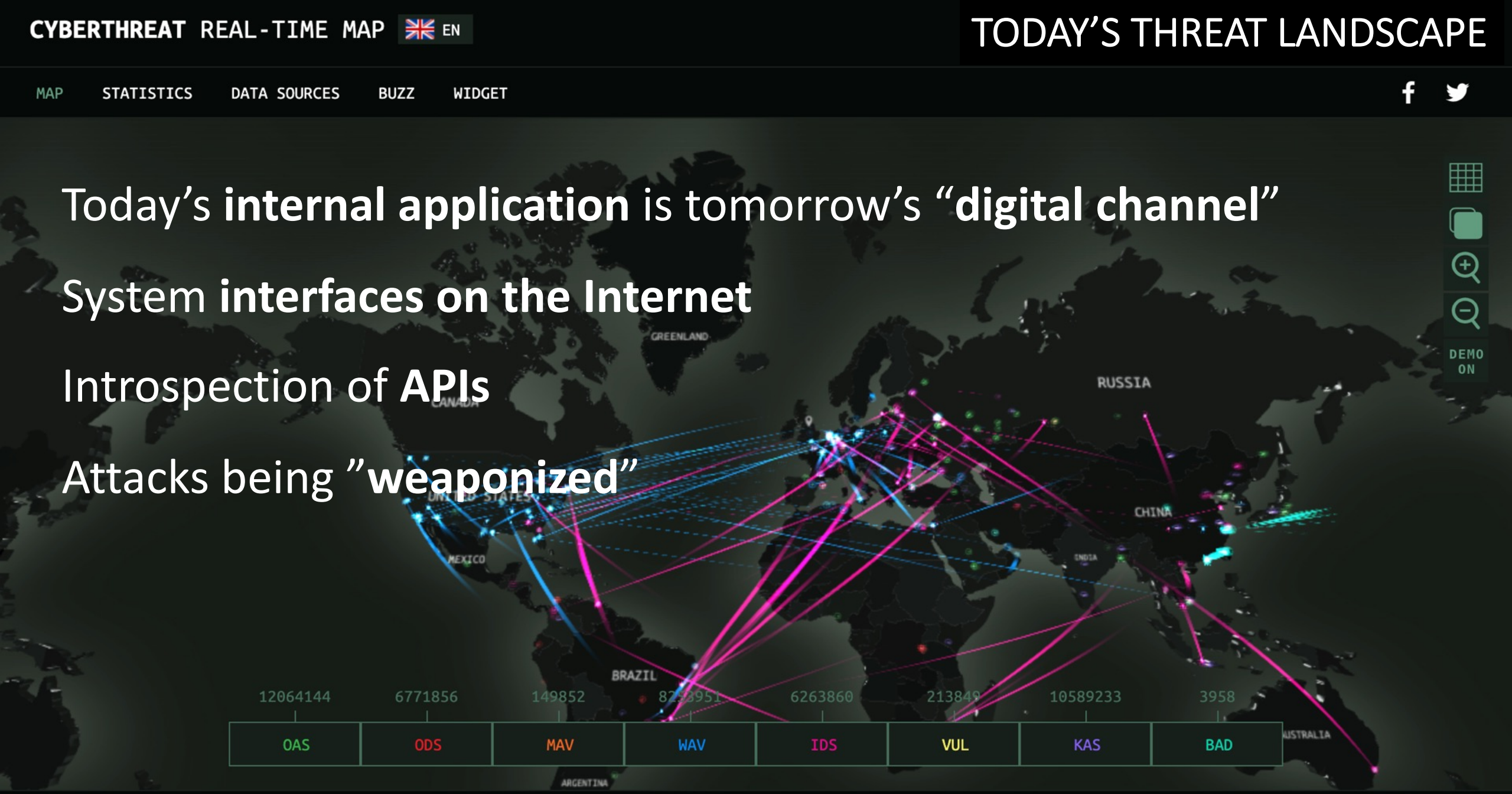
BUILDING APPLICATIONS SECURELY

The Threat

SECURITY THREATS

- We need systems that are **dependable** in the face of
  - Malice, Mistakes, Mischance
- People are sometimes **bad**, **careless** or just **unlucky**
- System **security** aims to **mitigate** these situations

TODAY'S THREAT LANDSCAPE



Today's **internal application** is tomorrow's "**digital channel**"

System **interfaces on the Internet**

Introspection of **APIs**

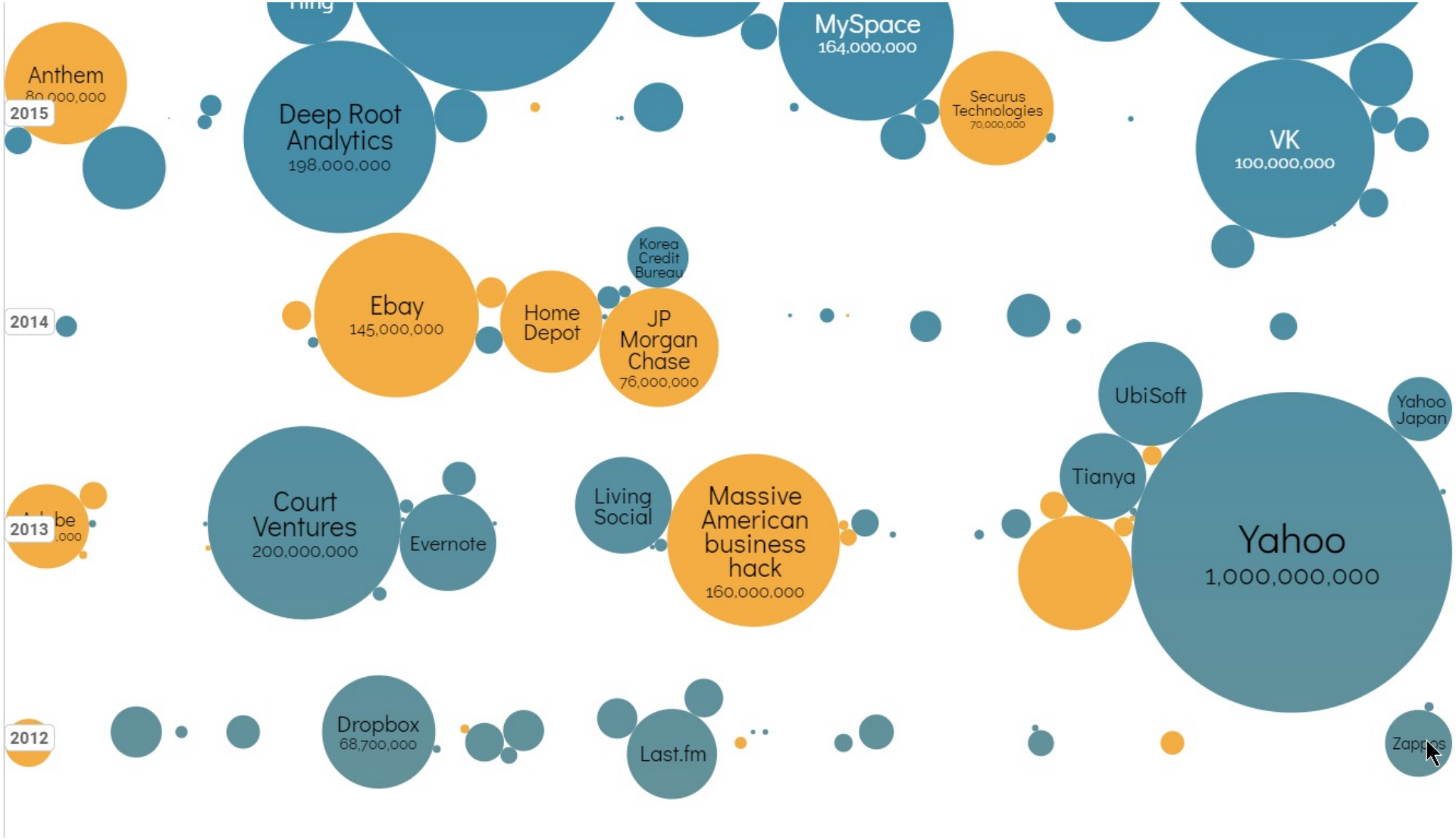
Attacks being "**weaponized**"

DATA BREACHES: 2005 - 2007

http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

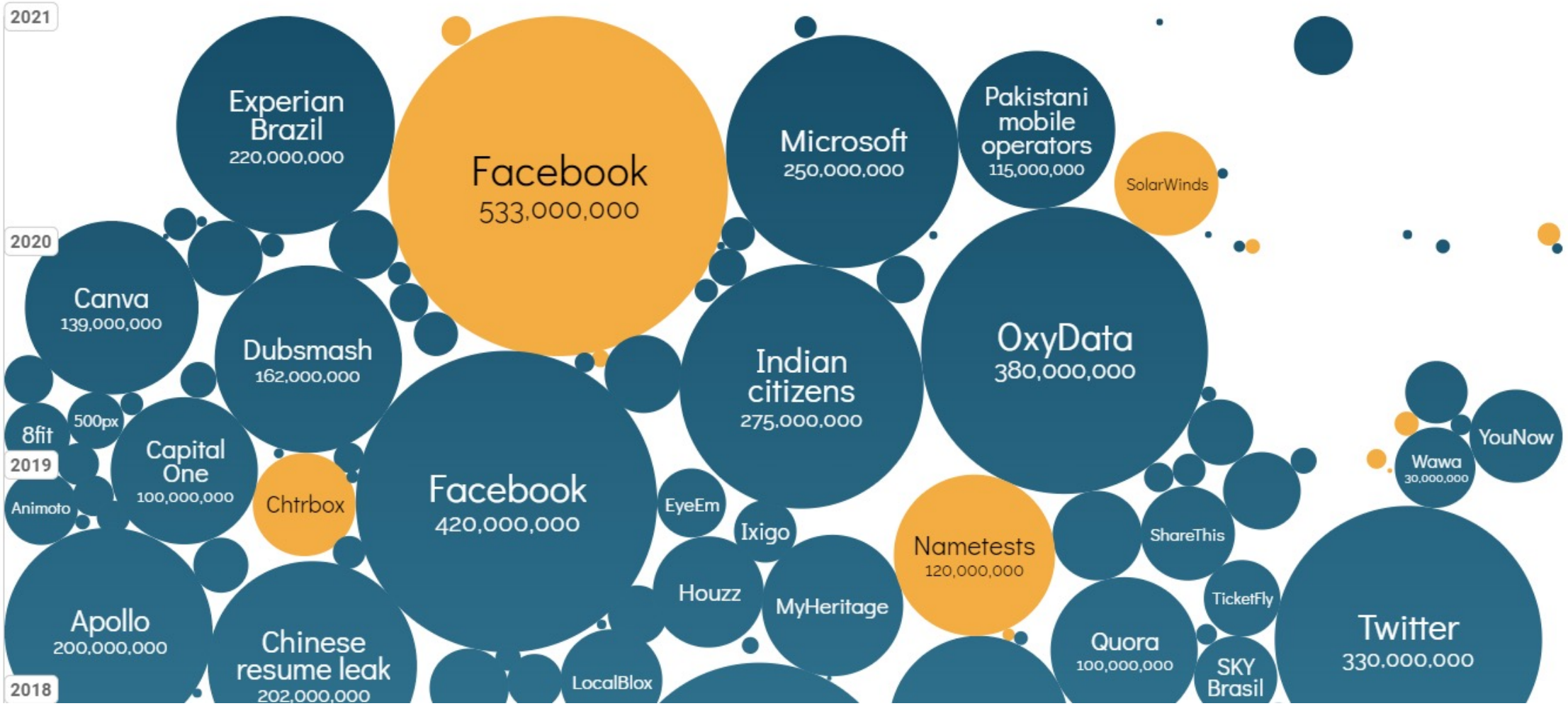
DATA BREACHES: 2008 - 2011

DATA BREACHES: 2012 - 2015



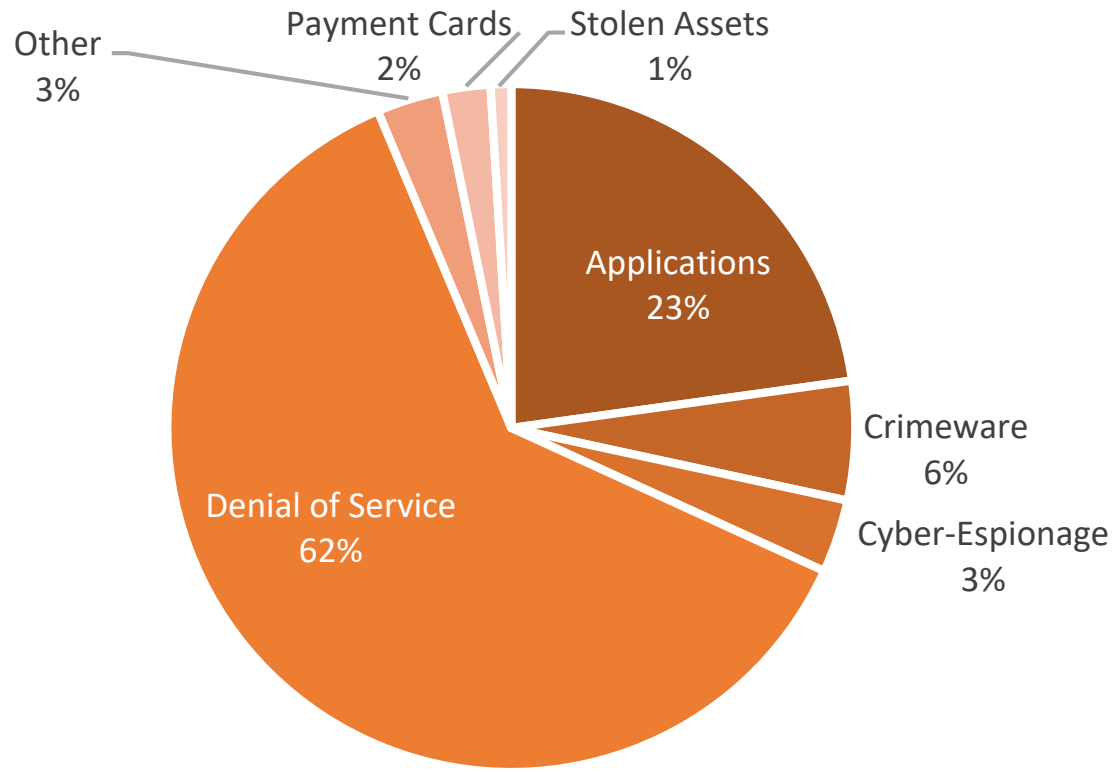
DATA BREACHES: 2016 - 2018

DATA BREACHES: 2019 – 2021



THE IMPORTANCE OF SOFTWARE SECURITY

- Verizon research security incidents annually
- There are many root causes
- Applications are significant
- This study suggests that about a quarter are application related

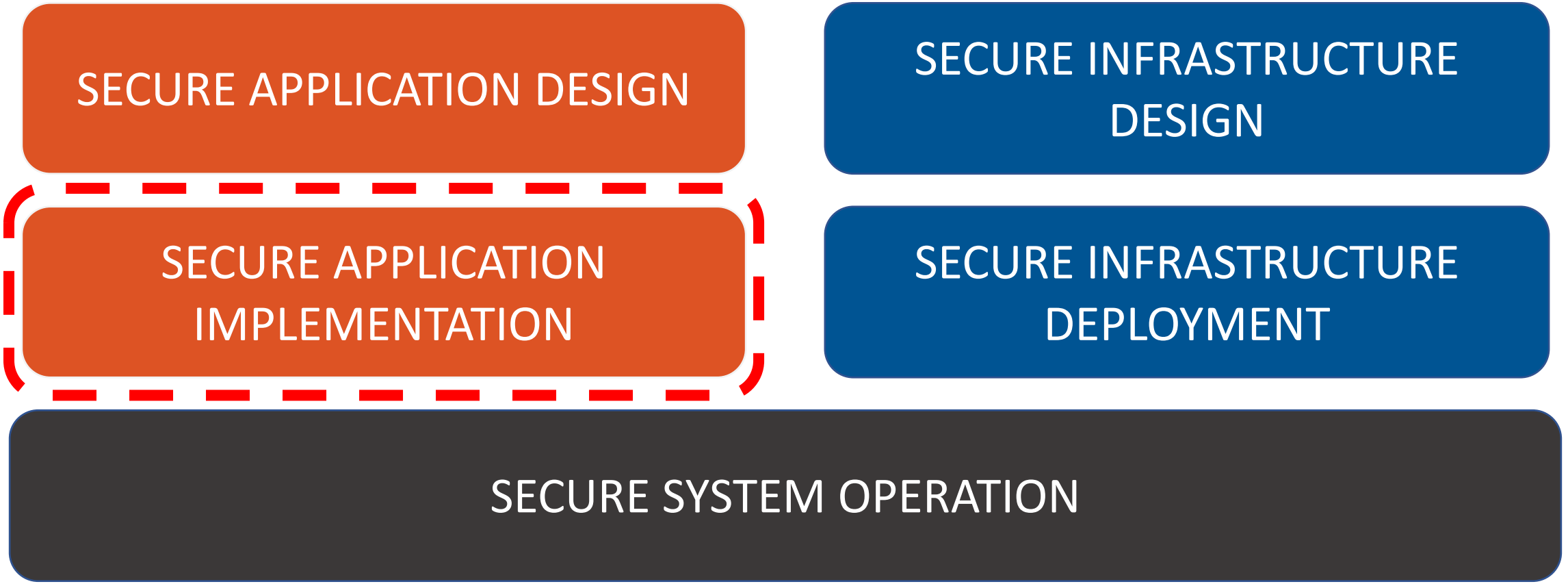


https://enterprise.verizon.com/resources/reports/dbir

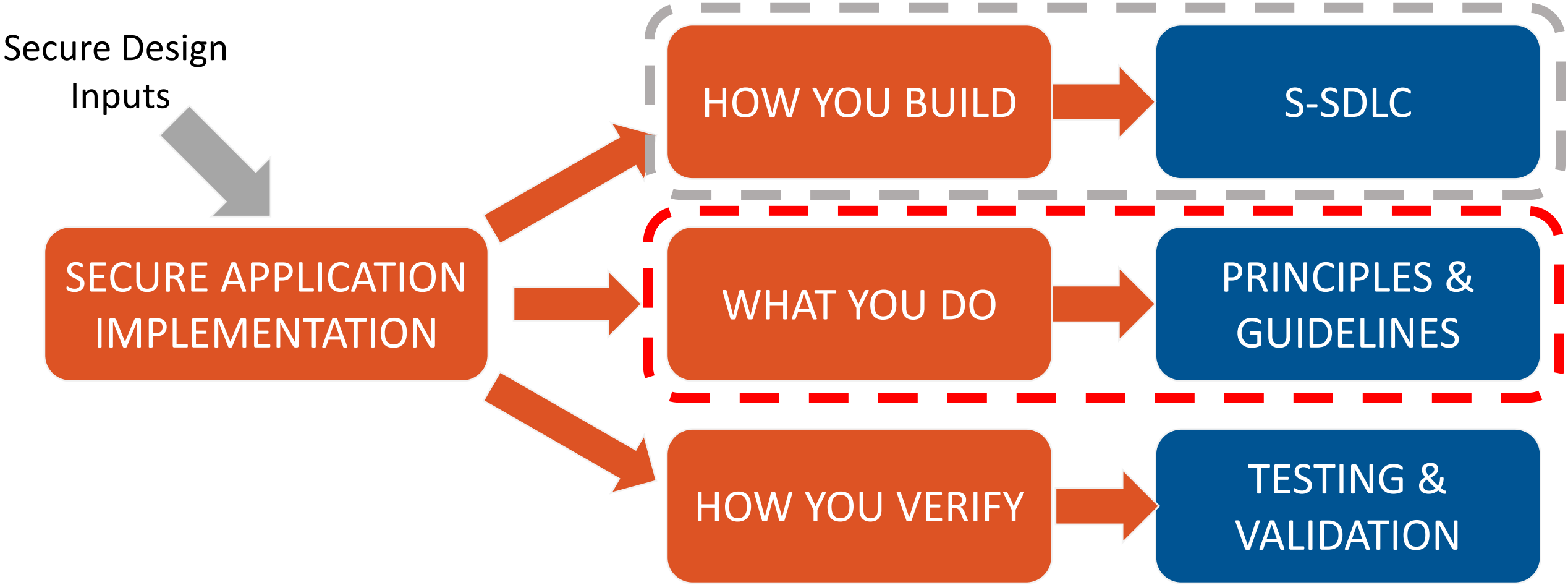
BUILDING APPLICATIONS SECURELY

Mitigation via Software Security

DIMENSIONS OF SECURITY PRACTICE



SECURE APPLICATION IMPLEMENTATION



SECURITY IN THE DEVELOPMENT LIFECYCLE

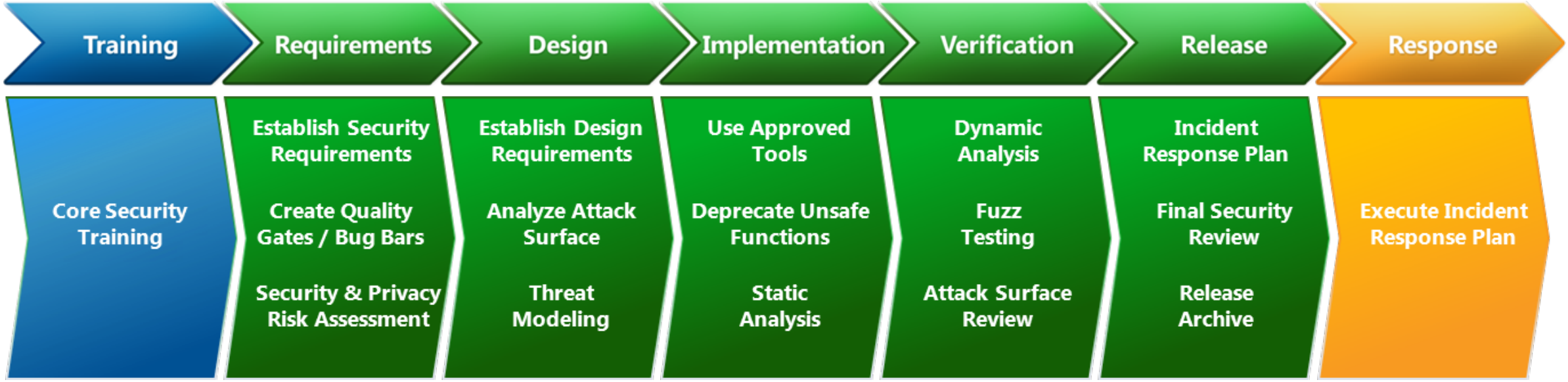
Microsoft SDL

OWASP SAMM

Building Security In Maturity Model

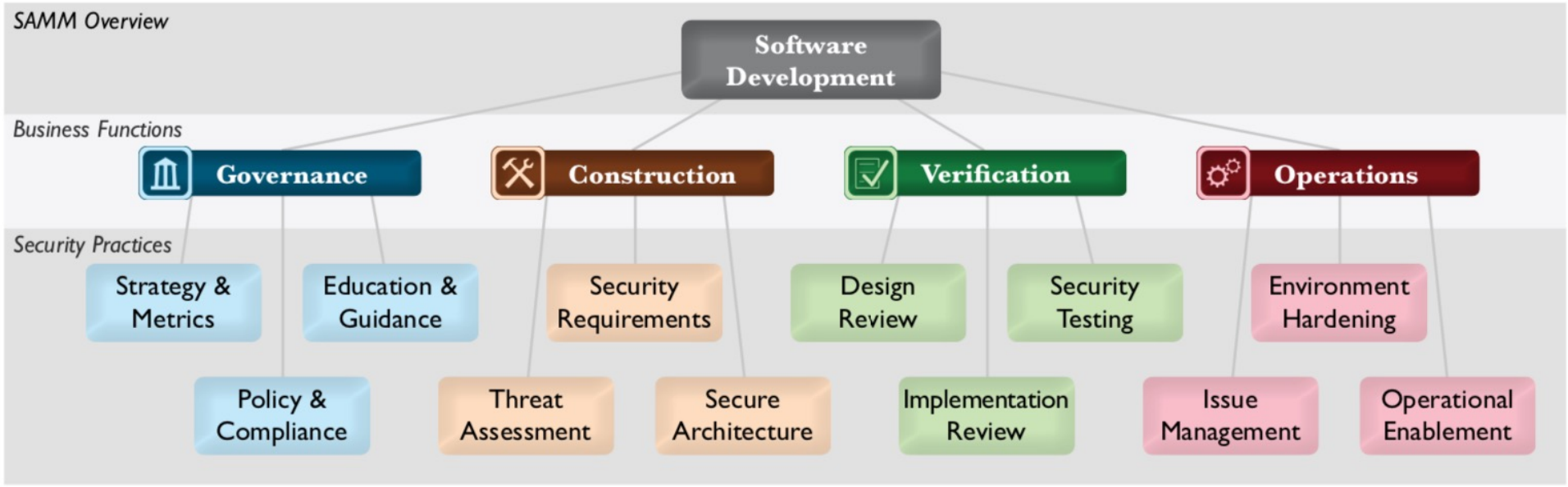
SAFECode Fundamental Practices

MICROSOFT SECURE DEVELOPMENT LIFECYCLE



- Developed by Microsoft for their product groups
- 17 practices across the lifecycle
- Good resources available from Microsoft
- Needs to be applied to your development lifecycle

OWASP SOFTWARE ASSURANCE MATURITY MODEL



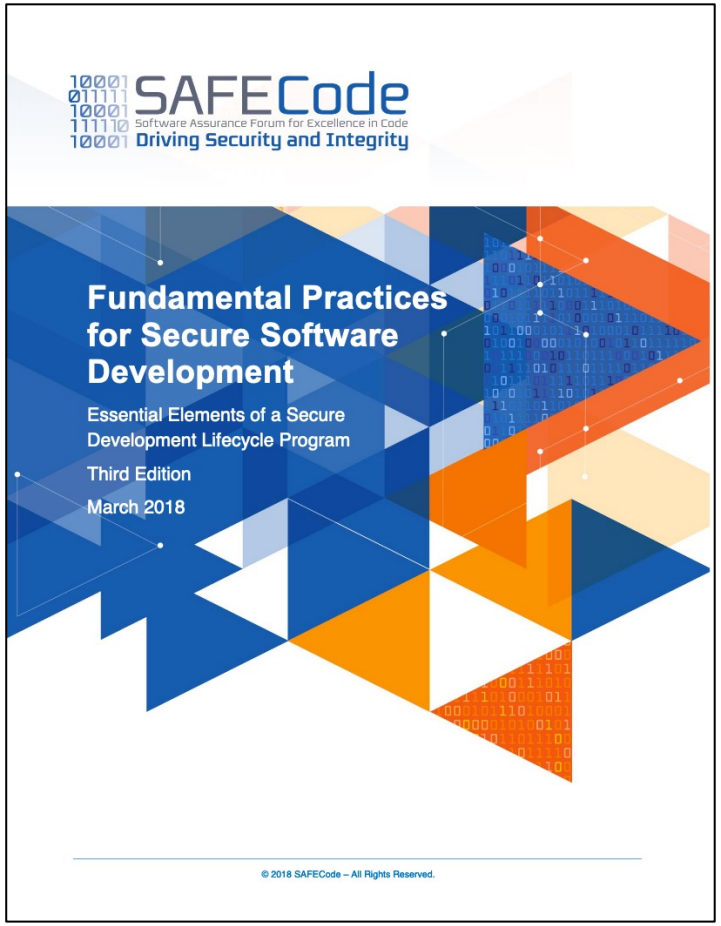
- Project from OWASP volunteers since 2008
- Governance, Construction, Verification & Operation
- Three key practice areas for each
- Maturity model rather than an SDLC

“BUILDING SECURITY IN” MATURITY MODEL

- Synopsys study of software security practice
- Member firms surveyed to establish practices
- Statistics & trends published
- Organisations can “benchmark” against aggregated findings

SAFECODE

- Membership organization of some leading software security firms
- Publish free on-demand training, blogs and guides



BUILDING APPLICATIONS SECURELY

Principles for Secure Development

SECURE DEVELOPMENT PRINCIPLES

1. Security is everyone’s concern
2. Focus continually through the lifecycle
3. Good design improves security
4. Use proven tools and technologies
5. Automate security checking
6. Verify your software supply chain
7. Generic and technology specific concerns matter

SECURITY IS EVERYONE'S CONCERN

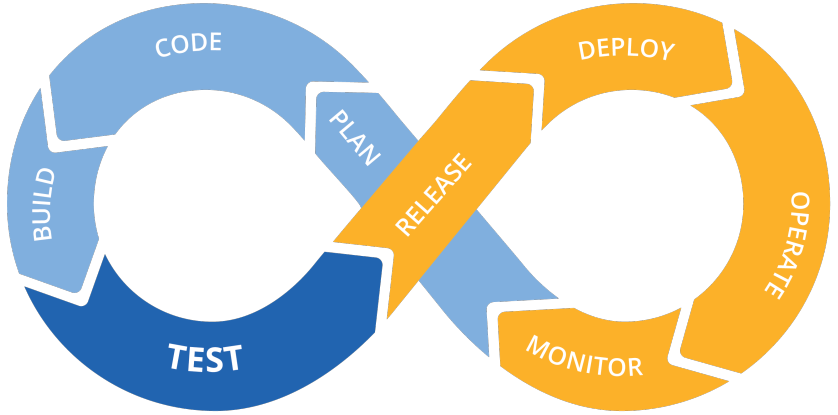
- A “concern“ not a ”feature”
- Needs team-wide awareness
- Avoid security being a ”specialist” problem
- Integrate security awareness into normal dev tasks

SECURITY CHAMPIONS

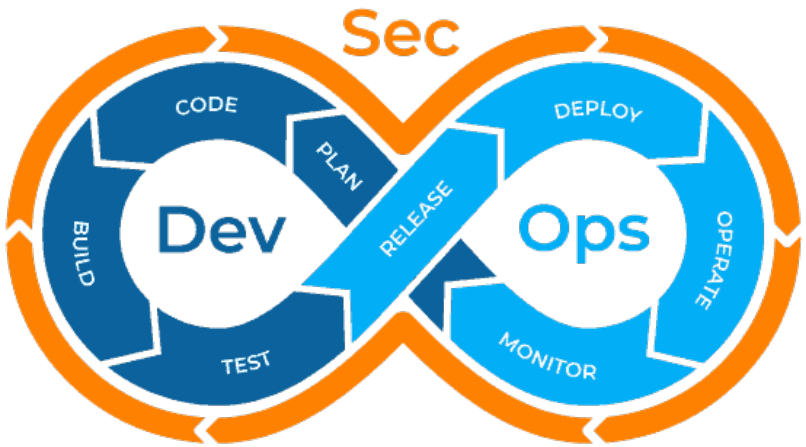
- Security is everyone's problem …
  but always someone else's
- You need enthusiastic advocates
  - People who will take ownership
- Self-selecting "security champions"
- Invest, involve, promote, support
  - don't isolate them!

FOCUS CONTINUALLY THROUGH THE LIFECYCLE

- Cannot “test security in”
- Traditional security testing delays deployment
- Need continual security work through lifecycle
  - analysis, design, dev, test, …



A WORD ON DEVSECOPS



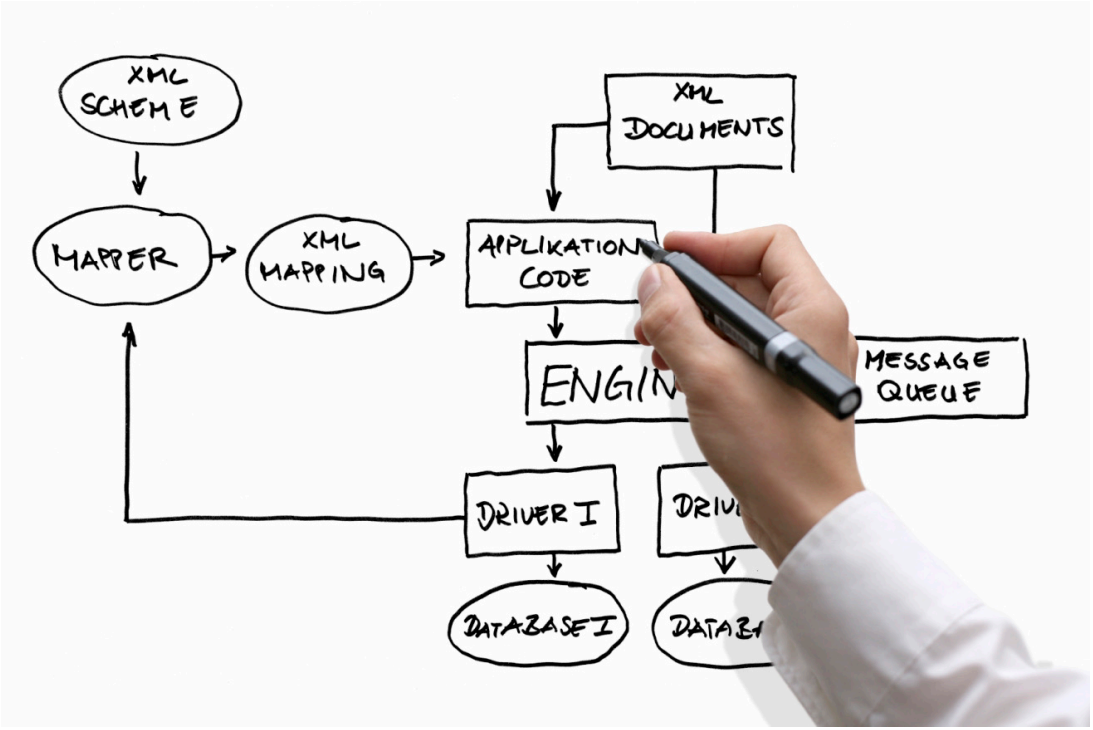
"Security says no"

We're all security engineers now

⇒ "Security" is another silo to integrate into the cross-functional delivery team

GOOD DESIGN IMPROVES SECURITY

- Careless design often creates vulnerabilities
- Strong types, simple mechanisms, well structured code all aid security
- Simpler implementation is easier to understand & secure



GOOD DESIGN IMPROVES SECURITY

```
public class OrderRequestHandler extends HttpServlet {

    private OrderService orderService;

    public void init() throws ServletException {...}

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        int qty = Integer.parseInt(request.getParameter( s: "order.quantity"));
        String sku = SecHelper.escapeStr(request.getParameter( s: "order.item.sku")) ;

        int ordered = orderService.orderItem(sku, qty) ;

        response.getWriter().println(renderResponse(sku, qty, ordered));
    }
```

Perfectly “reasonable” code … but with a potential security problem

… what happens if qty < 0 ?

GOOD DESIGN IMPROVES SECURITY

```java
public class OrderRequestHandler extends HttpServlet {

    private OrderService orderService;

    public void init() throws ServletException {...}

    public void doG
            throws

        int paramQt
        OrderQuanti
        SkuValue sk

        OrderQuanti

        response.ge
    }
```

```java
public class OrderQuantity {
    static private final int MAX_VALUE = 100 ;

    private final int value ;

    public OrderQuantity(int qty) {
        if (qty < 0) {
            throw new IllegalArgumentException("Quantities must not be negative") ;
        }
        if (qty > MAX_VALUE) {
            throw new IllegalArgumentException("Maximum quantity of " + MAX_VALUE + " exceeded by " + qty) ;
        }
        this.value = qty ;
    }

    // ...
```

Example of DDD improving security ”for free”

USE PROVEN TOOLS AND TECHNOLOGY

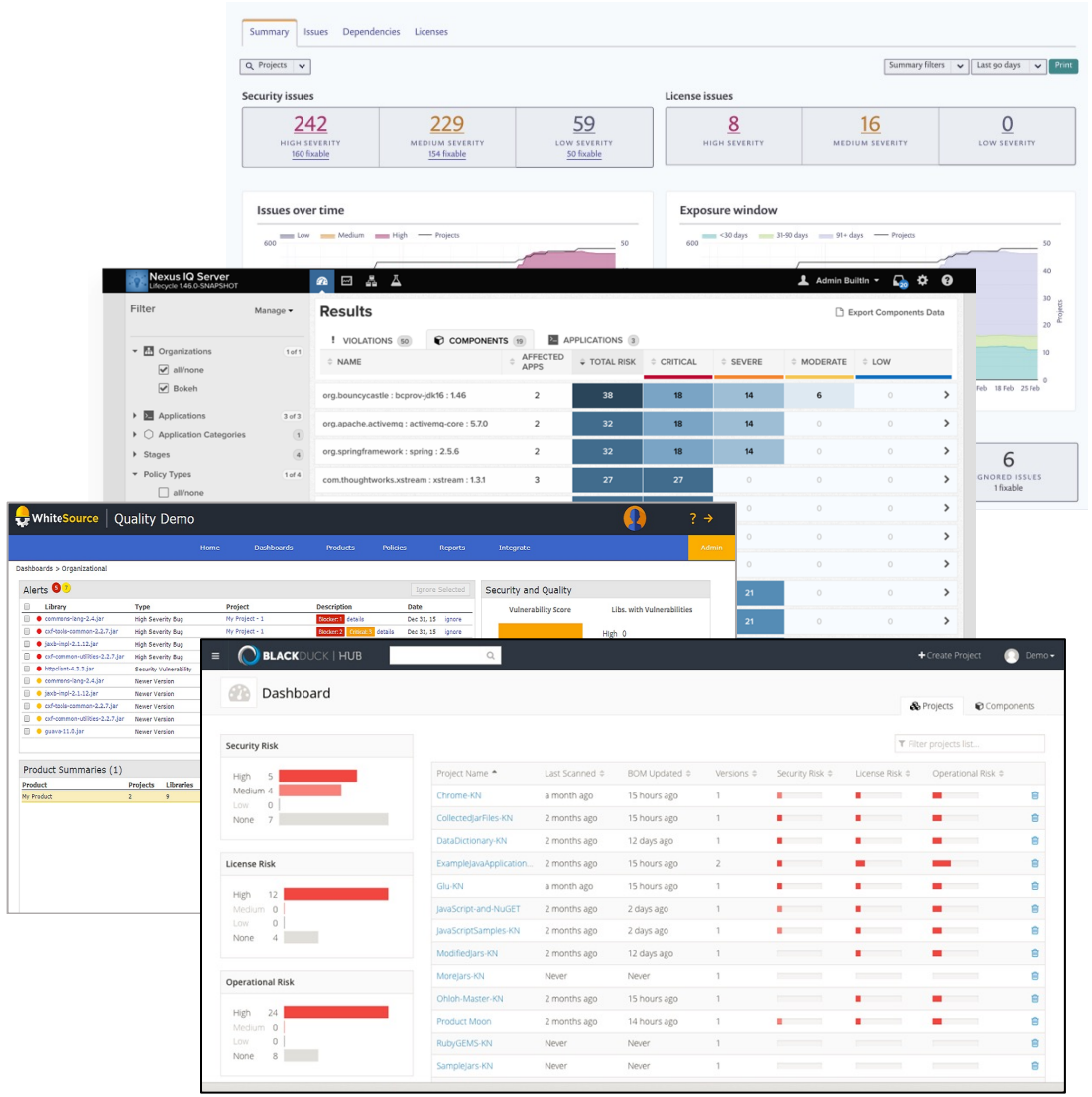
- Software is hard to secure
- Security software is very hard to secure
- Vulnerabilities emerge over time (from attacks)
- Proven tools & technology reduce production vulnerabilities

AUTOMATE SECURITY CHECKING

- Some security checks can be automated – SAST, DAST
- Consistency and efficiency
- Find (some) problems earlier
- Challenges include false positives and responding effectively
- Only ever part of the solution

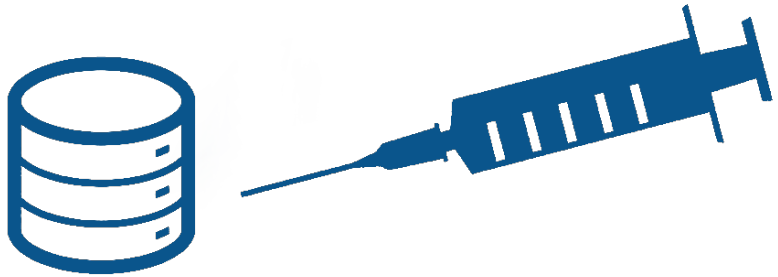
VERIFY YOUR SOFTWARE SUPPLY CHAIN

- 3rd party code is a possible risk – often open source
- Tools exist for OSS security, risk & compliance:
  - BlackDuck, Whitesource, Sonatype, Snyk, …
- Scan code to find dependencies
- Checks for known vulnerabilities
- Alerts and dashboards for monitoring



GENERAL AND SPECIFIC CONCERNS MATTER

- Many security concerns transcend technology
  - Injection, logging, …
- Technical stacks also have their specific weaknesses:
  - C/C++ - memory management
  - Java – reflection, serialisation
  - Python – module loading



SQL Injection

```
1  package my;
2
3  import java.io.IOException;
4
5  import javax.servlet.Servlet;
6  import javax.servlet.ServletException;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10
11 public class TestServlet extends HttpServlet implements Servlet {
12     static final long serialVersionUID = 1L;
13
14     public TestServlet() {
15         super();
16     }
17
18     protected void doGet(HttpServletRequest request,
19             HttpServletResponse response) throws ServletException, IOException {
20         doPost(request, response);
21     }
22
23     protected void doPost(HttpServletRequest request,
24             HttpServletResponse response) throws ServletException, IOException {
25         response.getWriter().println("blah");
26     }
27 }
```

BUILDING APPLICATIONS SECURELY

Implementation Guidelines

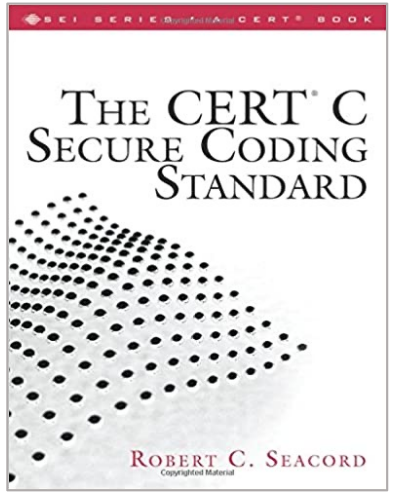
GENERIC SECURE CODING GUIDELINES

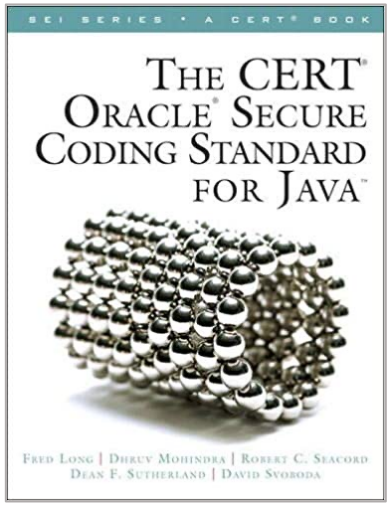
SAFECode Secure Coding Practices

OWASP Secure Coding Practices

Common Weaknesses Enumeration

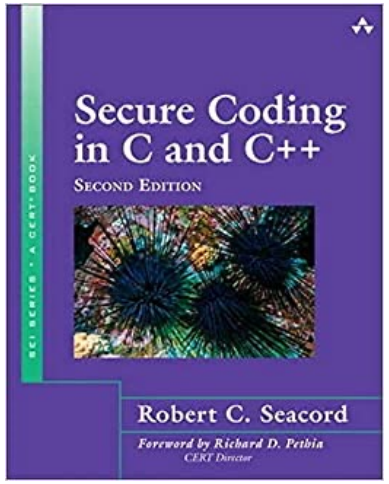
TECHNOLOGY SPECIFIC GUIDELINES

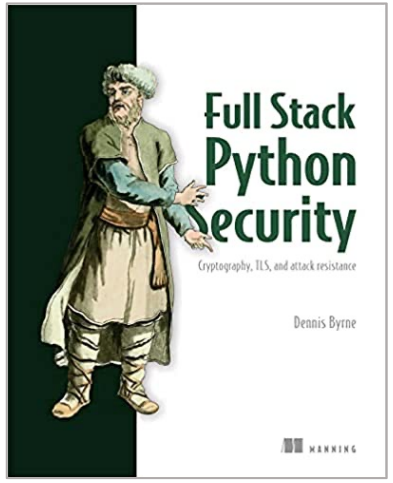




Microsoft

.NET Secure Coding Guidelines





Java™ ORACLE®

Secure Coding Guidelines

SECURE CODING GUIDELINES

- There are quite a few standards, which overlap significantly
- Need time to understand and apply
  - Oracle Java Security Guidelines contains 71 guidelines in 10 sections
- Something for your Security Champions to work through
  - you need the practical minimal subset for your threats and risks

GENERIC EXAMPLE – INJECTION ATTACKS

Unvalidated input passed to any interpreter

- Operating system and SQL are most common
- Configuration injection often overlooked

```
SELECT * from table1 WHERE name = '%1'
```

Set '%1' to **' OR 1=1 --** **… this results in this query:**

```
SELECT * FROM table1 WHERE name = '' OR 1=1 --
```

Defences include "escaping" inputs, bind variables, using white lists, …

JAVA SPECIFIC EXAMPLE – RANDOM NUMBERS

Java has two random number generators:

java.**util**.Random and java.**security**.SecureRandom

Guess which one isn't random but most people use?

```
Random rand = new java.util.Random() ;
SecureRandom secrand = new java.security.SecureRandom() ;

long utilTimeMsec = timeALambda( iterations: 100000, () -> rand.nextInt()) ;
long secTimeMsec = timeALambda( iterations: 100000, () -> secrand.nextInt()) ;
System.out.println("Util Random Execution Time:   " + utilTimeMsec);
System.out.println("Secure Random Execution Time: " + secTimeMsec);
```

```
$> java com.artechra.RandomTest
Util Random Execution Time:   7
Secure Random Execution Time: 49
```

PYTHON SPECIFIC EXAMPLE – UNPICKLING DATA

Python has a serialization system called “Pickle”

- Java, C# and others have similar mechanisms

A useful way of moving data around … and a security liability

```
# Don't do this at home
import pickle
malicious_cmd = "__import__('os').system('ls -l')"
pickle_txt = 'c__builtin__\neval\n(V{0}\ntR.'.format(malicious_cmd)
pickled_data = bytes(pickle_txt, 'utf-8')
pickle.loads(pickled_data)
```

To be fair, the docs clearly state:

“The pickle module is not secure. Only unpickle data you trust.”

SECURITY TESTING AND VALIDATION

- Like any other critical system quality **application security** needs to be **tested early** and **often** – mix of automation and manual techniques
  - Detailed description of testing is beyond this talk
  - But we need to be aware of it so that we know someone is doing it
- **Automated security testing**: Static Analysis (SAST) and Dynamic Analysis (DAST)
- **Automated functional testing**: do the application security features work?
- **Exploratory testing**: fuzz testing and penetration testing
- **Platform testing**: testing application's use of platform & configuration

Remember: security also needs to be tested from an infrastructure and operational perspective!

BUILDING APPLICATIONS SECURELY

Summary

SUMMARY (I)

- Much of the technology we use is inherently insecure
  - Mitigation needs to be part of application development
- Attacking systems is becoming industrialised
  - Digital transformation is providing more valuable, insecure targets
- Secure implementation is part of an end-to-end approach

SUMMARY (II)

- Three aspects to secure implementation
  - **HOW** do you go about **building** the software? (SDLC)
  - **WHAT** do you **do** to build the software? (Principles, Guidelines)
  - **HOW** do you **verify** what you build? (Testing, Validation)
- We explored a set of principles
  - Security is **everyone's concern**
  - **Continual focus** through the lifecycle
  - **Good design** improves security
  - Use **proven** tools and technologies
  - **Automate** security checking
  - Verify your **software supply chain**
  - **Generic** and **technology specific** concerns matter

SUMMARY (III)

- Both **generic** and **language-specific** concerns
  - A number of sets of guidelines exist … use them!
  - **SAFECode**, **OWASP** Secure Coding Practices, **Oracle** Secure Java Guidelines, **Microsoft** .NET Secure Guidelines, **CERT** Coding Practices
- We haven't explored security **testing** and **validation**
  - critically important and another area to learn about
  - involve specialist experts, particularly for manual aspects

BOOKS & PUBLICATIONS

WHAT DO I DO NEXT?

Get started …

Work out where you are …

Get some people interested …

Work out what to improve next …

Improve that thing …

REPEAT !

THANK YOU

Eoin Woods
Endava
@eoinwoodz
eoin.woods@endava.com
careers.endava.com