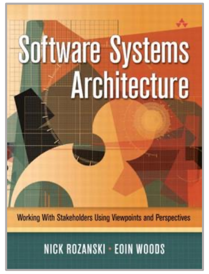
endava

BUILDING APPLICATIONS SECURELY

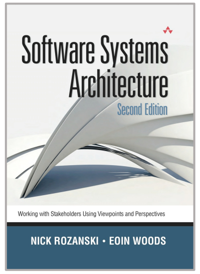
Eoin Woods
@eoinwoodz | www.eoinwoods.info

EOIN WOODS

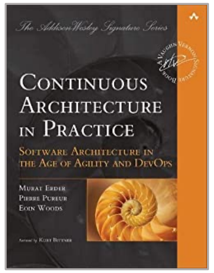
- Endava’s CTO, based in London (6 years)
  - 10+ years in products - Bull, Sybase, InterTrust
  - 10 years in capital markets - UBS and BGI
- Software engineer, architect, now CTO
- Long time security dabbler concerned at increasing cyber threats to systems
- Author, editor, speaker, community guy



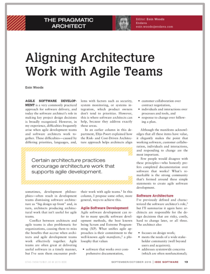
Software Systems Architecture
Working With Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI · EOIN WOODS



Software Systems Architecture
Second Edition
Working with Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI • EOIN WOODS

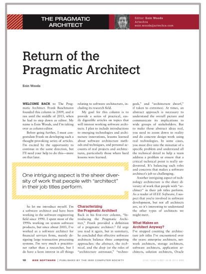


The Addison-Wesley Signature Series
Continuous Architecture in Practice
Software Architecture in the Age of Agility and DevOps
Murat Erder
Pierre Pureur
Eoin Woods

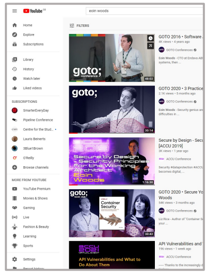


THE PRAGMATIC ARCHITECT
Aligning Architecture Work with Agile Teams
Eoin Woods
Certain architecture practices encourage architecture work that supports agile development.

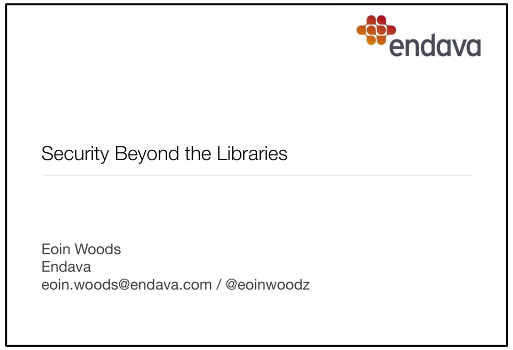
THE PRAGMATIC ARCHITECT
Architecting in the Gaps
A Metaphor for Architecture Work
Eoin Woods
Architects are often the only people responsible for bridging the gaps between a system’s elements.



THE PRAGMATIC ARCHITECT
Return of the Pragmatic Architect
Eoin Woods
One intriguing aspect is the sheer diversity of work that people with “architect” in their job titles perform.



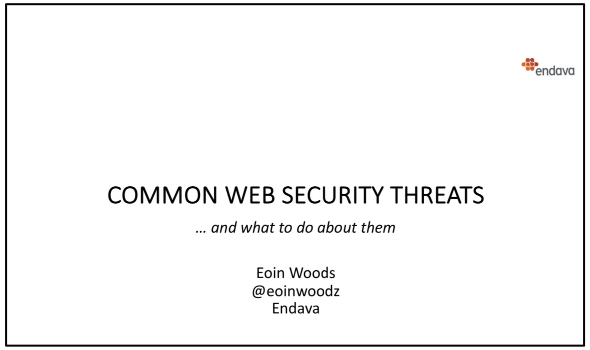
CONTEXT OF THIS TALK



Security Beyond the Libraries

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

[link](#)

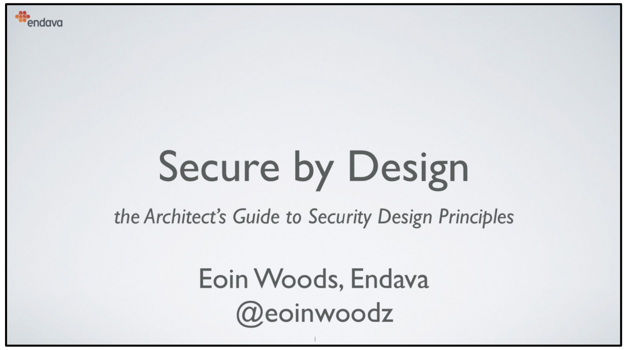


COMMON WEB SECURITY THREATS

… and what to do about them

Eoin Woods
@eoinwoodz
Endava

[link](#)

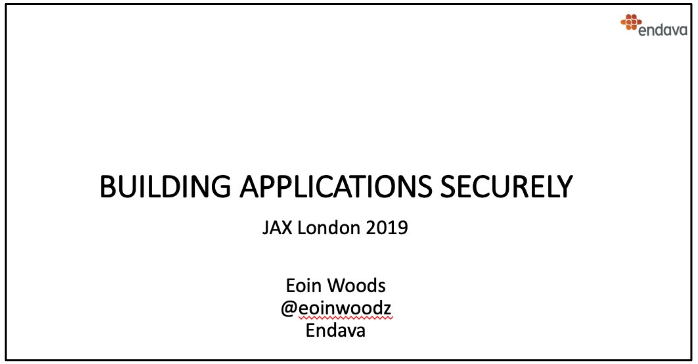


Secure by Design

the Architect's Guide to Security Design Principles

Eoin Woods, Endava
@eoinwoodz

[link](#)



BUILDING APPLICATIONS SECURELY

JAX London 2019

Eoin Woods
@eoinwoodz
Endava

Agenda

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

1

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

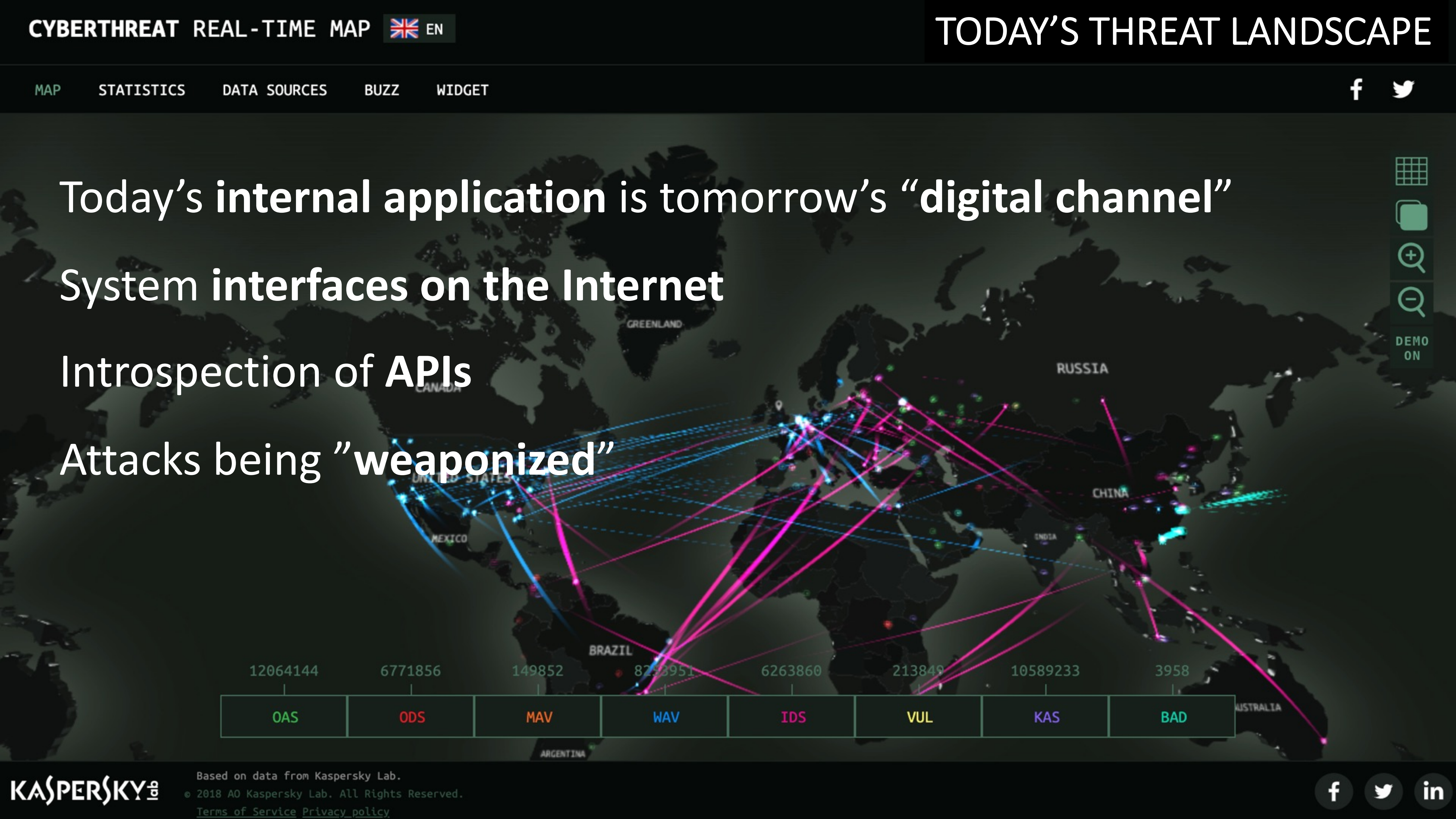
TODAY’S THREAT LANDSCAPE

Today’s **internal application** is tomorrow’s “**digital channel**”

System **interfaces on the Internet**

Introspection of **APIs**

Attacks being ”**weaponized**”

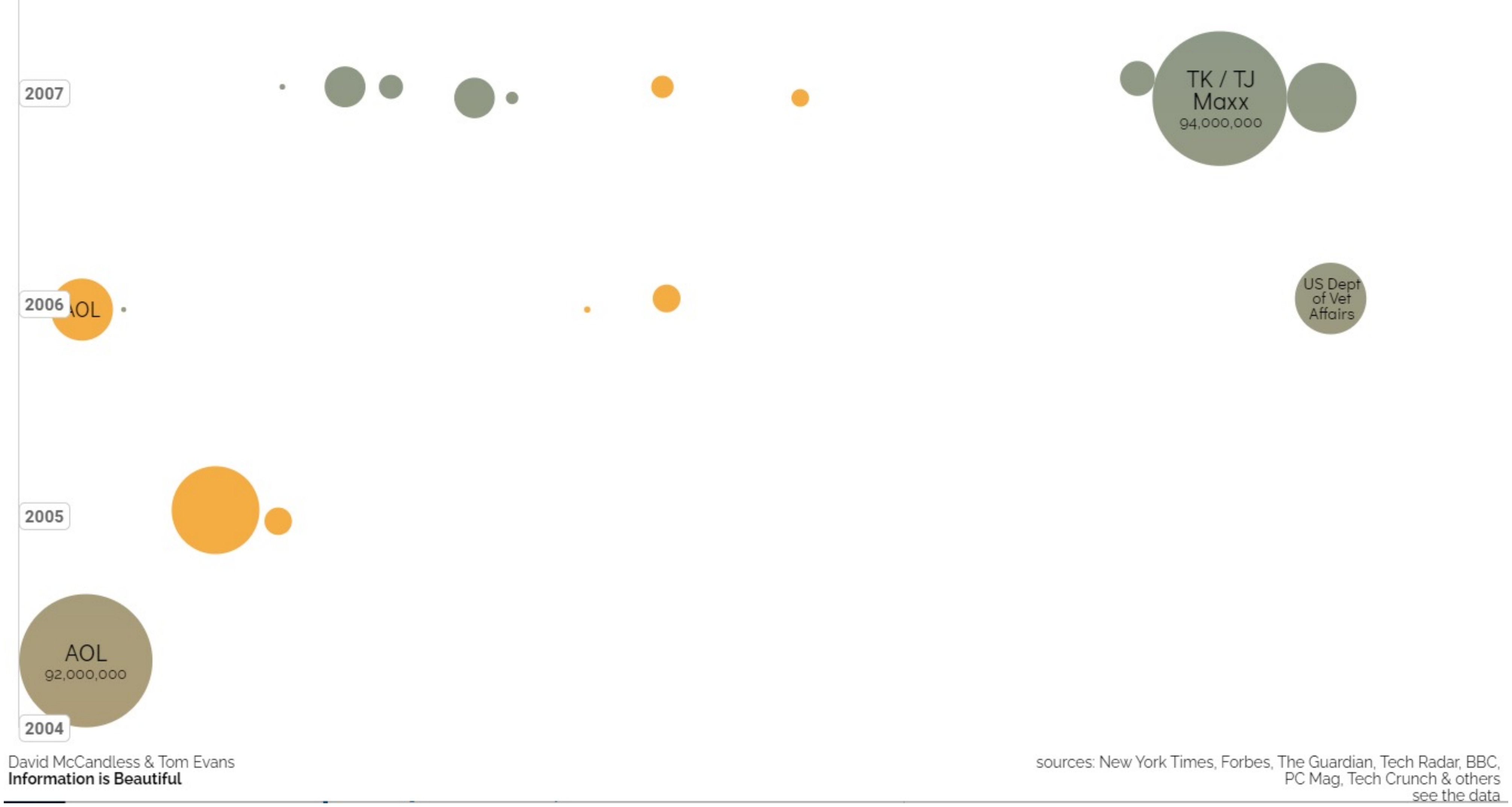


Based on data from Kaspersky Lab.

© 2018 AO Kaspersky Lab. All Rights Reserved.

Terms of Service Privacy policy

DATA BREACHES: 2005 - 2007

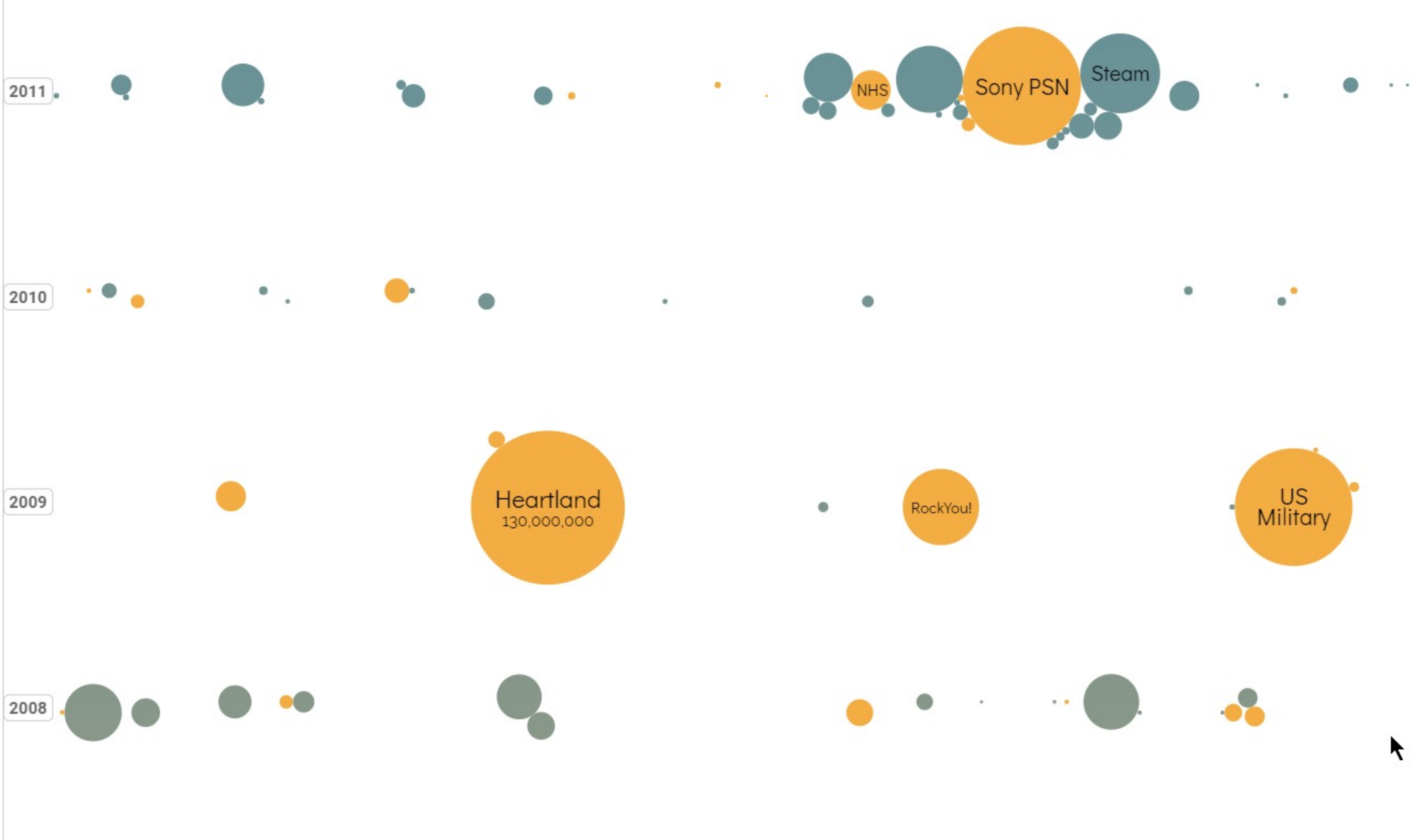


David McCandless & Tom Evans
Information is Beautiful

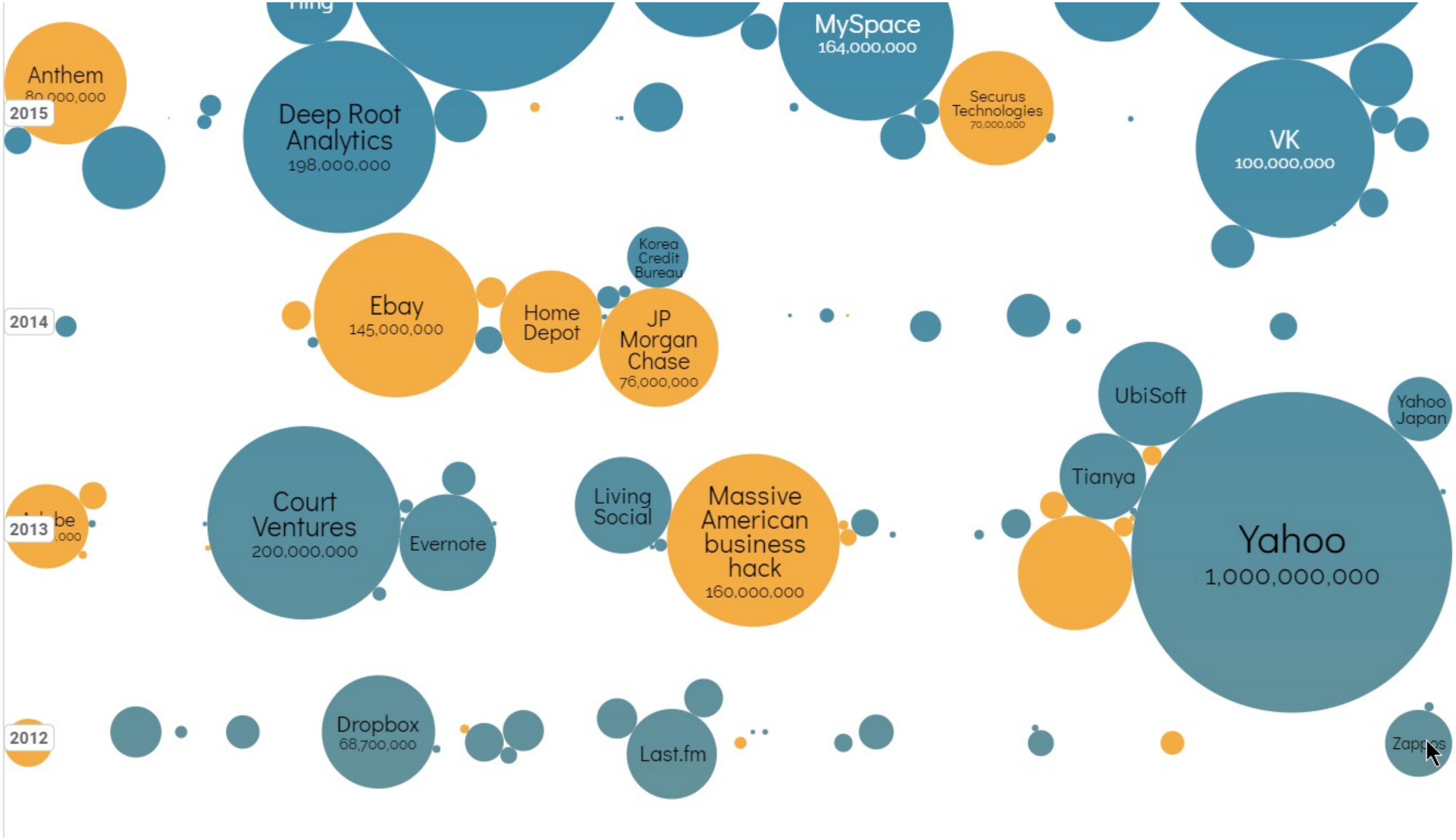
sources: New York Times, Forbes, The Guardian, Tech Radar, BBC, PC Mag, Tech Crunch & others
see the data

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

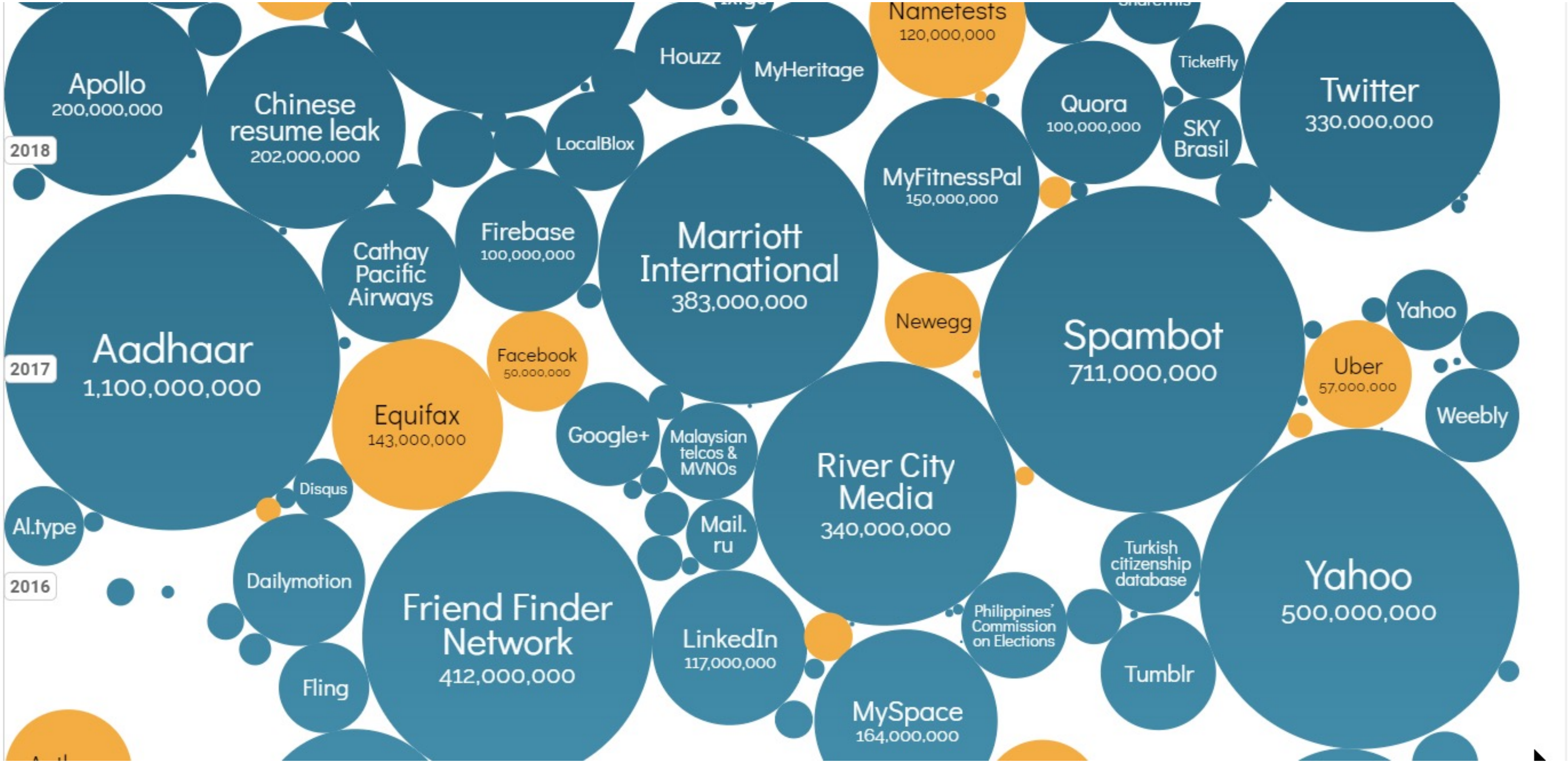
DATA BREACHES: 2008 - 2011



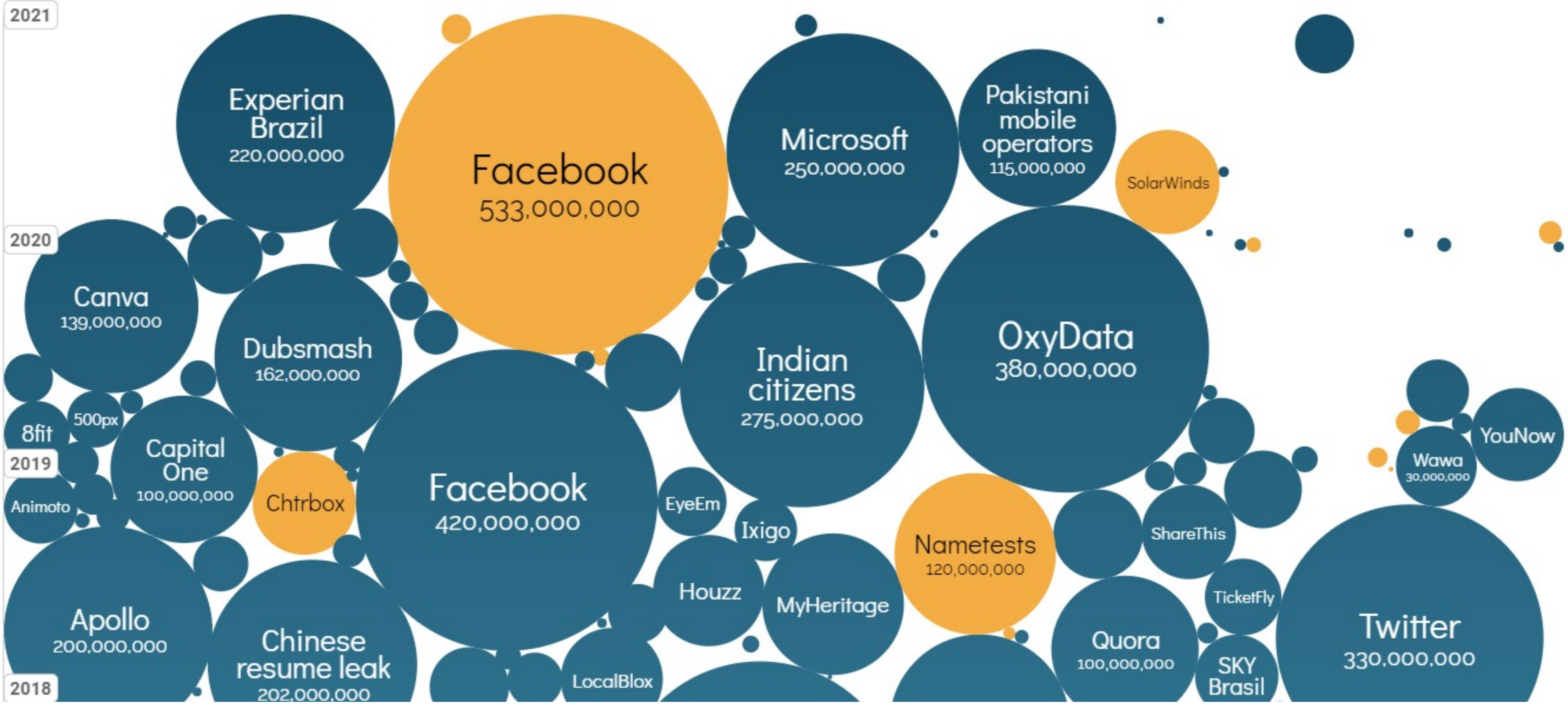
DATA BREACHES: 2012 - 2015



DATA BREACHES: 2016 - 2018

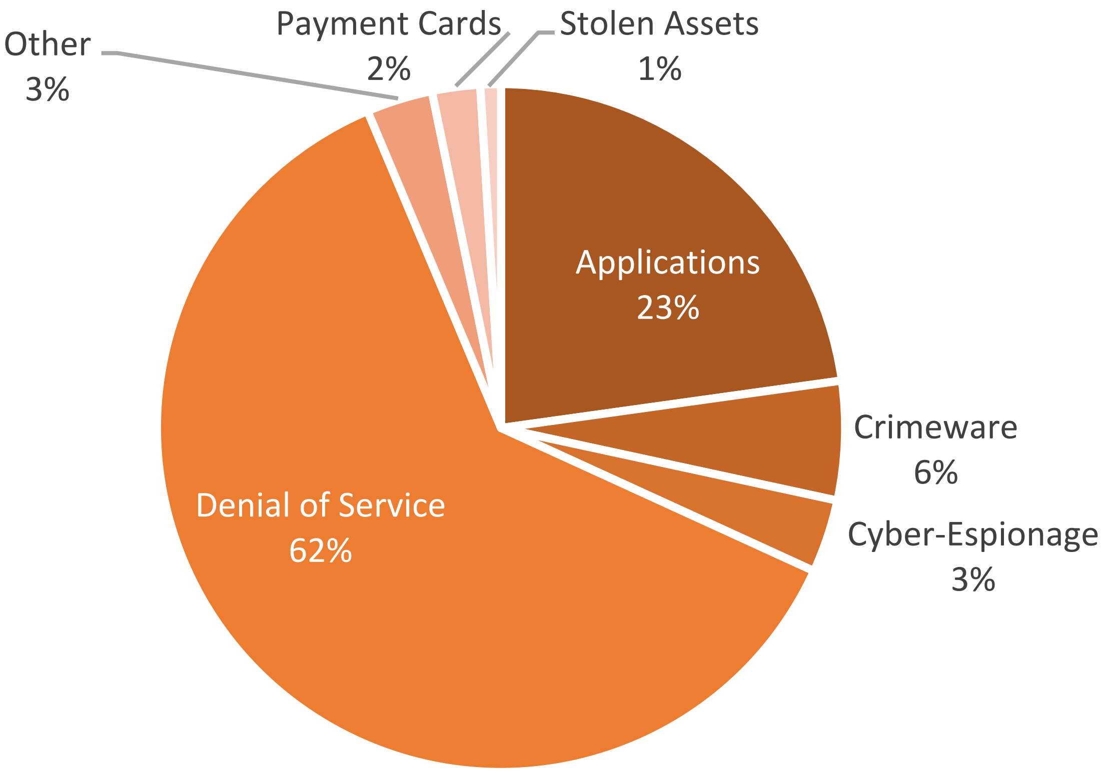


DATA BREACHES: 2019 – 2021



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



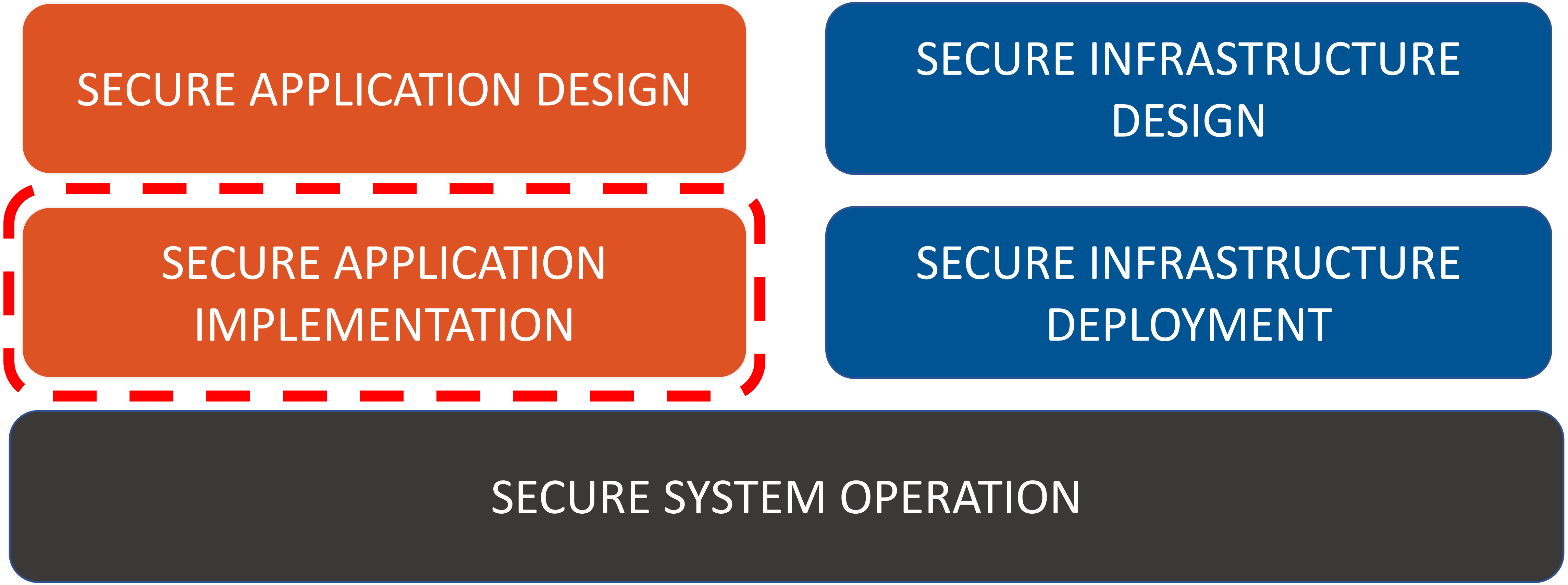
https://enterprise.verizon.com/resources/reports/dbir

2

BUILDING APPLICATIONS SECURELY

Mitigation via Software Security

DIMENSIONS OF SECURITY PRACTICE



SECURE APPLICATION DESIGN

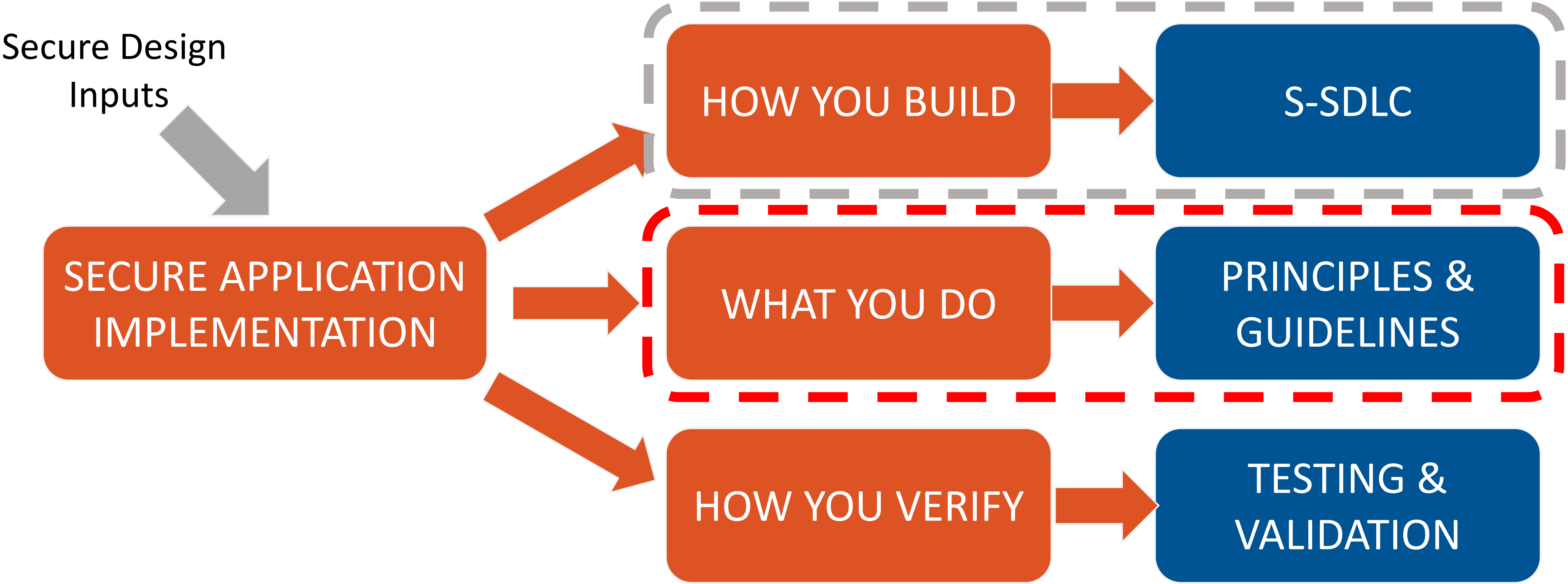
SECURE INFRASTRUCTURE DESIGN

SECURE APPLICATION IMPLEMENTATION

SECURE INFRASTRUCTURE DEPLOYMENT

SECURE SYSTEM OPERATION

SECURE APPLICATION IMPLEMENTATION


Secure Design Inputs

SECURE APPLICATION IMPLEMENTATION

HOW YOU BUILD

S-SDLC

WHAT YOU DO

PRINCIPLES & GUIDELINES

HOW YOU VERIFY

TESTING & VALIDATION

SECURITY IN THE DEVELOPMENT LIFECYCLE

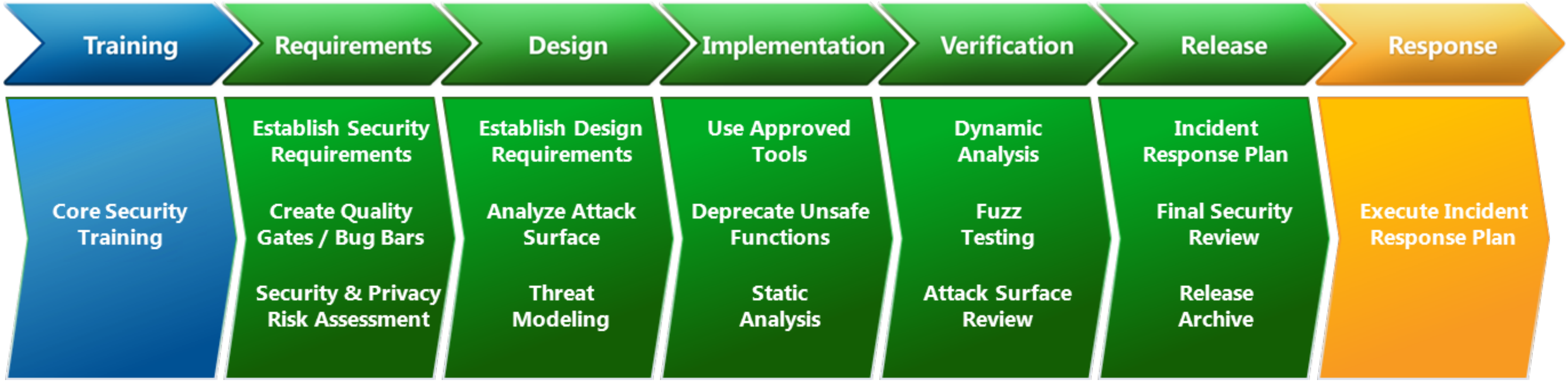
Microsoft SDL

OWASP SAMM

Building Security In Maturity Model

SAFECode Fundamental Practices

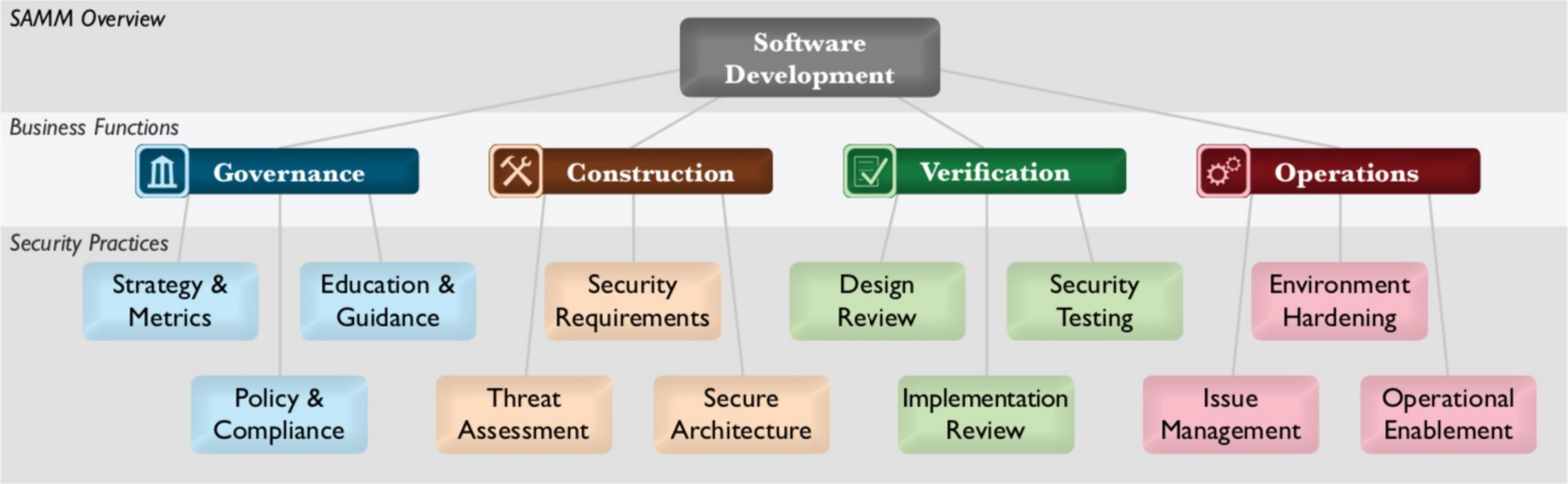
MICROSOFT SECURE DEVELOPMENT LIFECYCLE



Training	Requirements	Design	Implementation	Verification	Release	Response
Core Security Training	Establish Security Requirements Create Quality Gates / Bug Bars Security & Privacy Risk Assessment	Establish Design Requirements Analyze Attack Surface Threat Modeling	Use Approved Tools Deprecate Unsafe Functions Static Analysis	Dynamic Analysis Fuzz Testing Attack Surface Review	Incident Response Plan Final Security Review Release Archive	Execute Incident Response Plan

- 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



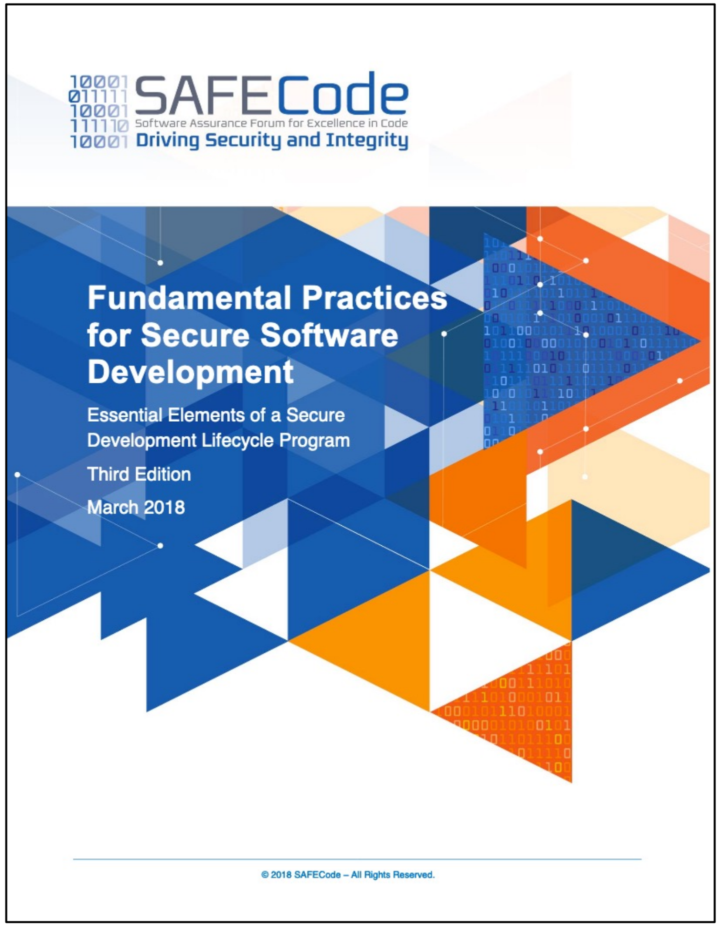
- Project from OWASP volunteers since 2008
- Governance, Construction, Verification & Operation
- Three key practice areas for each
- Maturity model rather than an SDLC

“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



SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

Fundamental Practices for Secure Software Development

Essential Elements of a Secure Development Lifecycle Program

Third Edition

March 2018

3

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

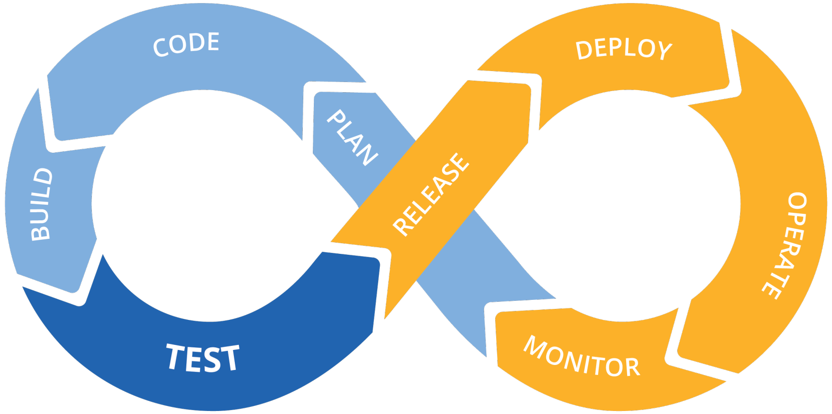
- A “concern“ not a ”feature”
- Needs team-wide awareness
- Avoid security being a ”specialist” problem
- Integrate security awareness into normal dev tasks

SECURITY CHAMPIONS

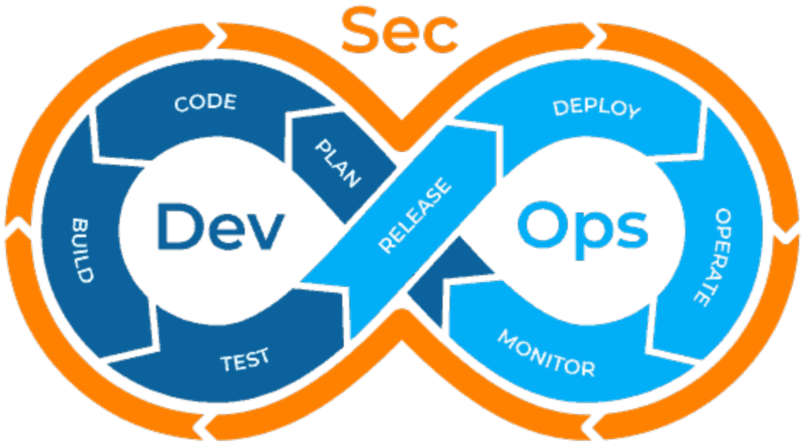
- Security is everyone’s problem …
  but always someone else’s
- You need enthusiastic advocates
  - People who will take ownership
- Self-selecting ”security champions”
- Invest, involve, promote, support
  - don’t isolate them!

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”

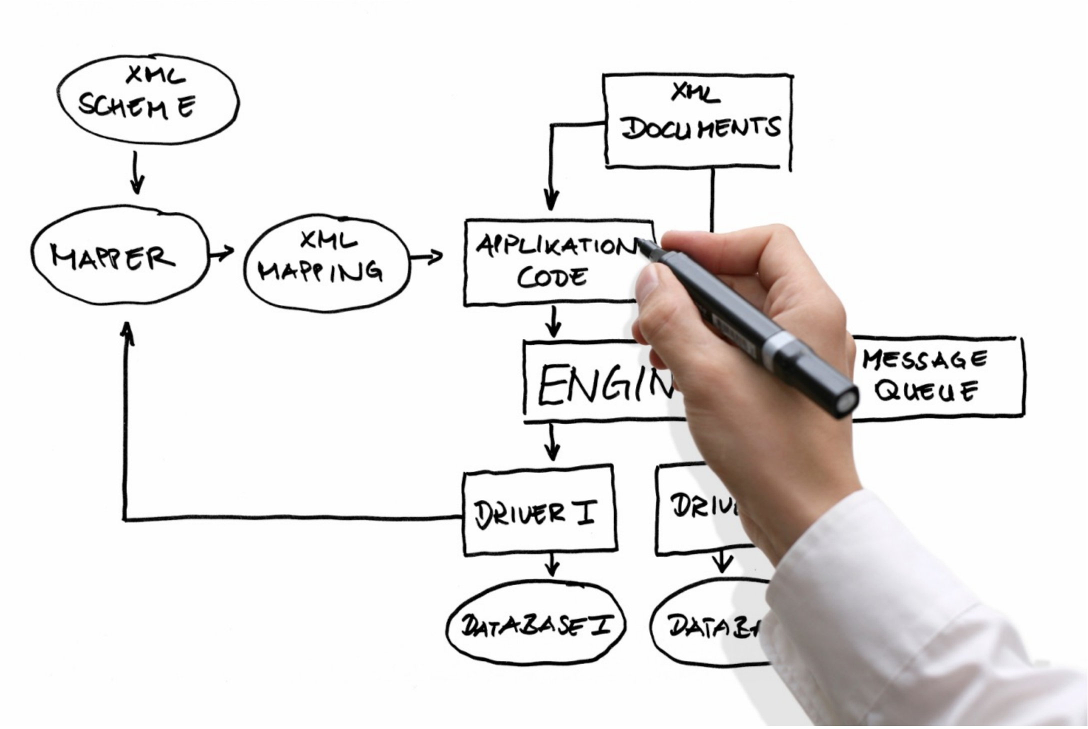
↓

We’re all security engineers now

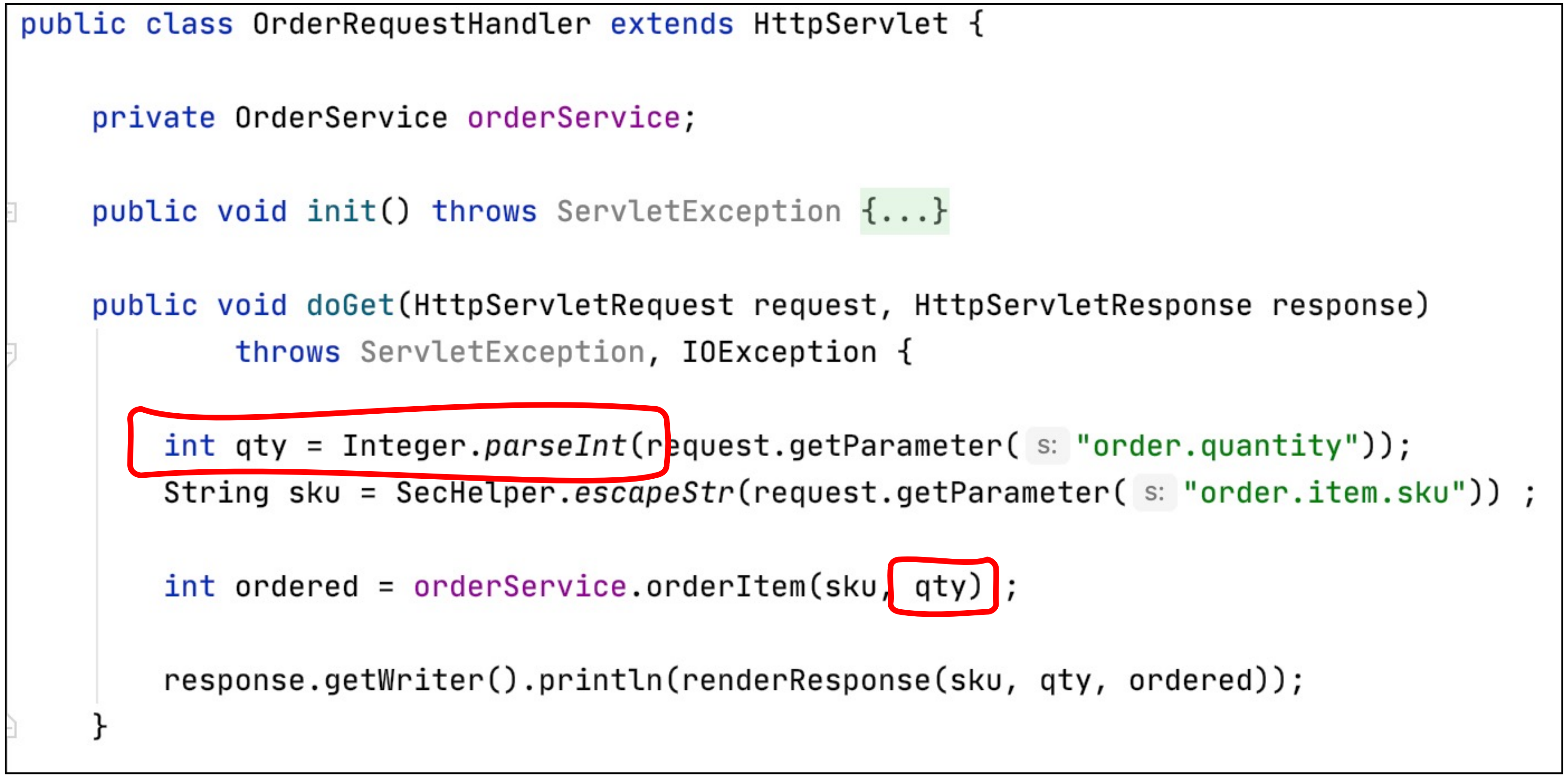
⇒ “Security” is another silo to integrate
into the cross-functional delivery team

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

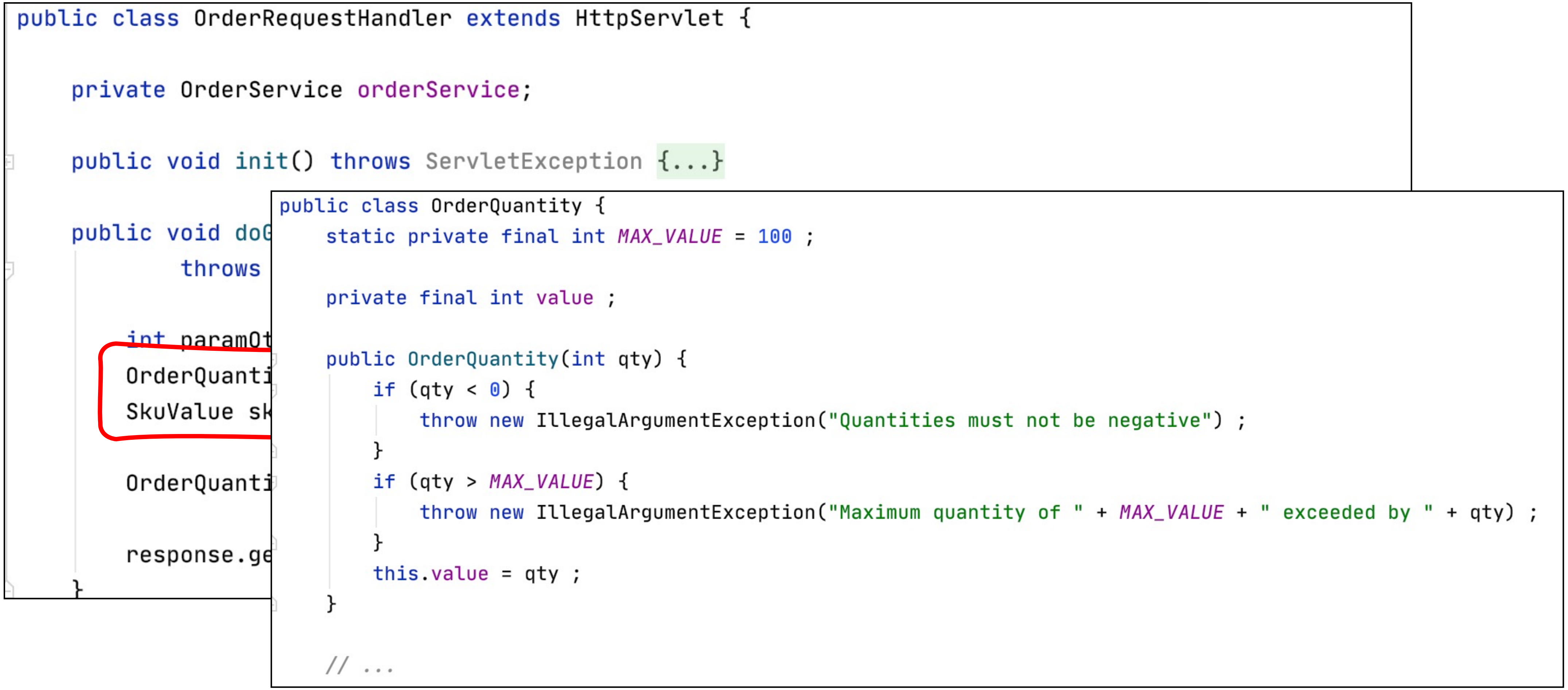


```
public class OrderRequestHandler extends HttpServlet {

    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



```
public class OrderRequestHandler extends HttpServlet {

    private OrderService orderService;

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

    public void doG
            throws

        int paramOt
        OrderQuanti
        SkuValue sk

        OrderQuanti

        response.ge
    }
```

```
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

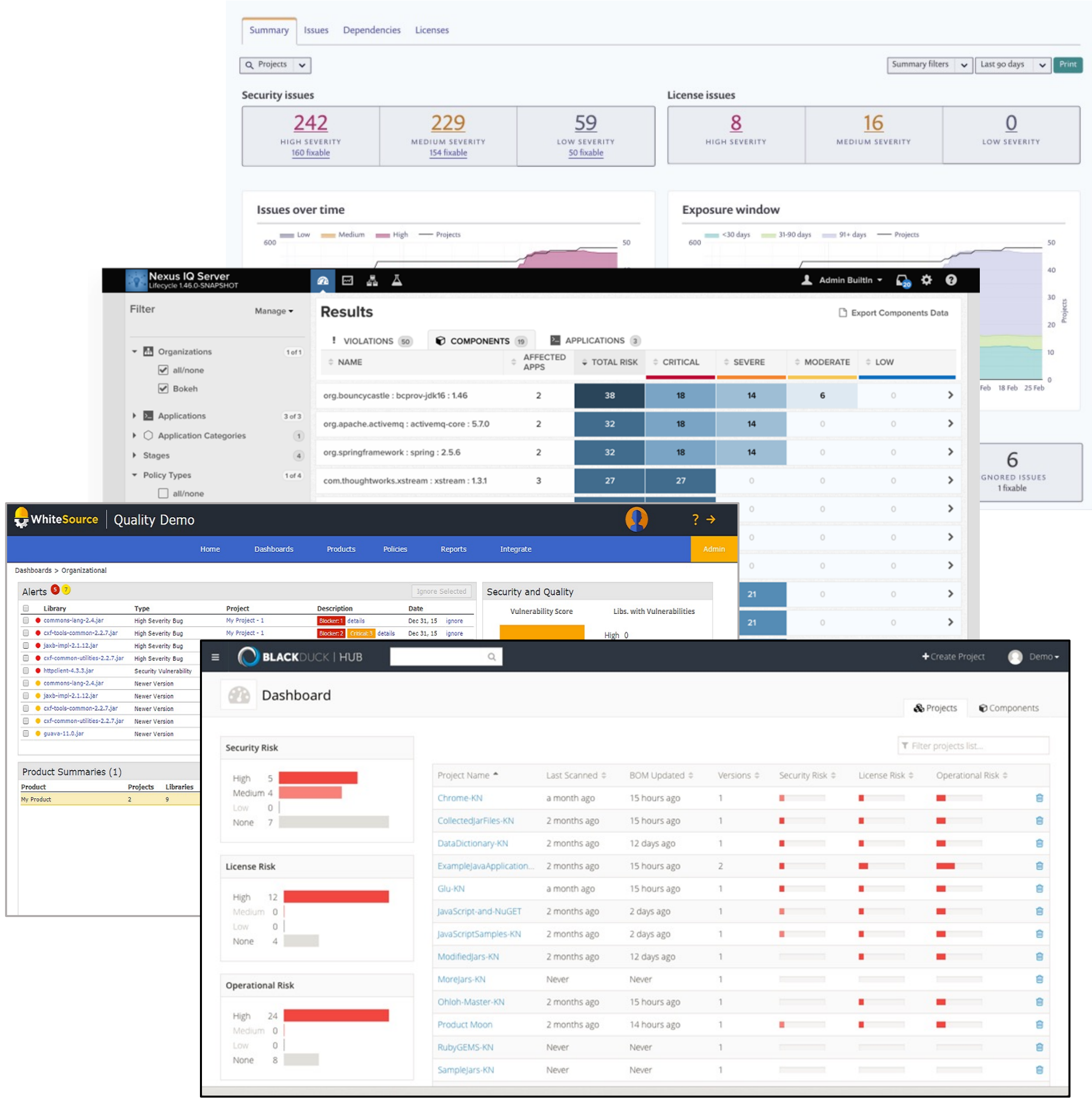
- Software is hard to secure
- Security software is very hard to secure
- Vulnerabilities emerge over time (from attacks)
- Proven tools & technology reduce production vulnerabilities

AUTOMATE SECURITY CHECKING

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



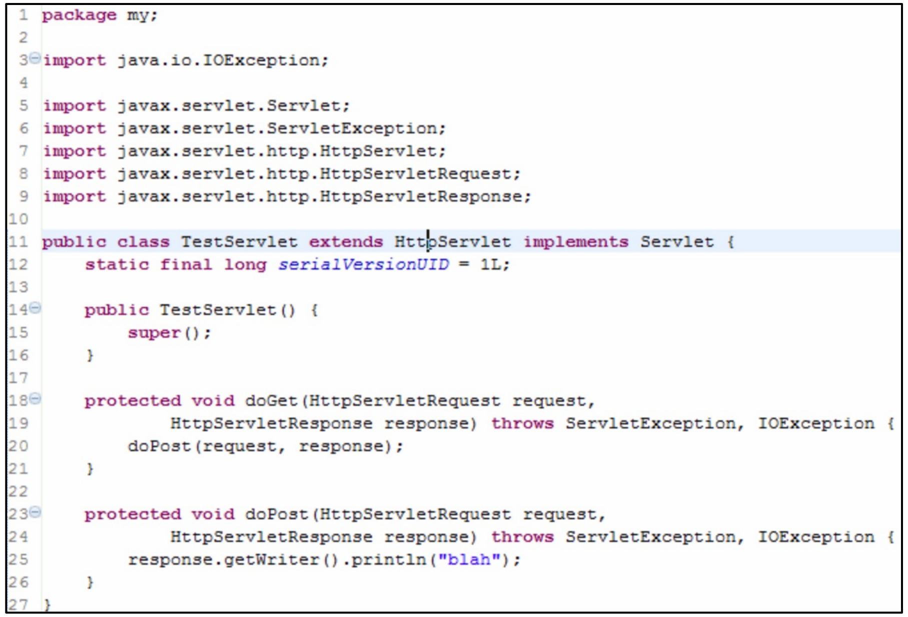
NAME	AFFECTED APPS	TOTAL RISK	CRITICAL	SEVERE	MODERATE	LOW
org.bouncycastle : bcprov-jdk16 : 1.46	2	38	18	14	6	0
org.apache.activemq : activemq-core : 5.7.0	2	32	18	14	0	0
org.springframework : spring : 2.5.6	2	32	18	14	0	0
com.thoughtworks.xstream : xstream : 1.3.1	3	27	27	0	0	0

Project Name	Last Scanned	BOM Updated	Versions
Chrome-KN	a month ago	15 hours ago	1
CollectedJarFiles-KN	2 months ago	15 hours ago	1
DataDictionary-KN	2 months ago	12 days ago	1
ExampleJavaApplication...	2 months ago	15 hours ago	2
Glu-KN	a month ago	15 hours ago	1
JavaScript-and-NuGET	2 months ago	2 days ago	1
JavaScriptSamples-KN	2 months ago	2 days ago	1
ModifiedJars-KN	2 months ago	12 days ago	1
MoreJars-KN	Never	Never	1
Ohloh-Master-KN	2 months ago	15 hours ago	1
Product Moon	2 months ago	14 hours ago	1
RubyGEMS-KN	Never	Never	1
SampleJars-KN	Never	Never	1

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



```
package my;

import java.io.IOException;

import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestServlet extends HttpServlet implements Servlet {
    static final long serialVersionUID = 1L;

    public TestServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        response.getWriter().println("blah");
    }
}
```

4

BUILDING APPLICATIONS SECURELY

Implementation Guidelines

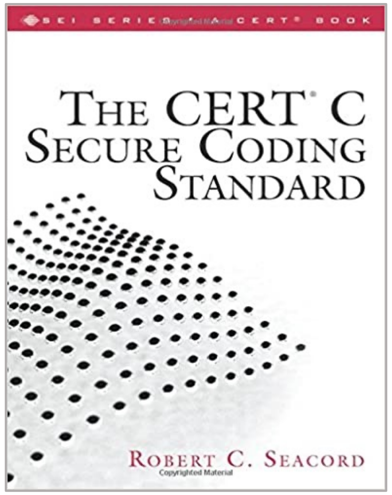
GENERIC SECURE CODING GUIDELINES

SAFECode Secure Coding Practices

OWASP Secure Coding Practices

Common Weaknesses Enumeration

TECHNOLOGY SPECIFIC GUIDELINES

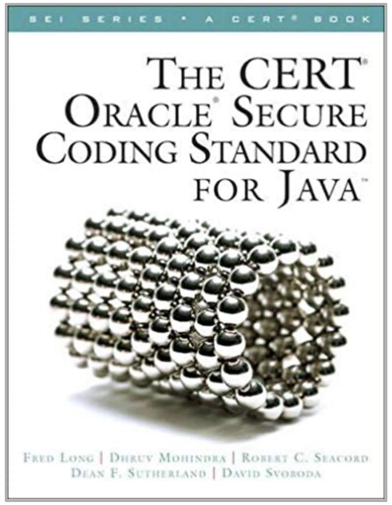


SEI SERIES • A CERT® BOOK

The CERT® C Secure Coding Standard

Robert C. Seacord

Copyrighted Material



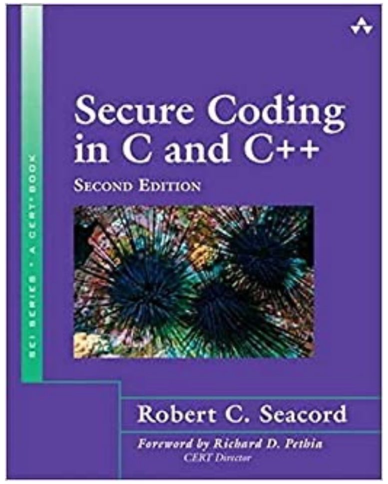
SEI SERIES • A CERT® BOOK

The CERT® Oracle® Secure Coding Standard for Java™

Fred Long | Dhruv Mohindra | Robert C. Seacord | Dean F. Sutherland | David Svoboda

Microsoft

.NET Secure Coding Guidelines



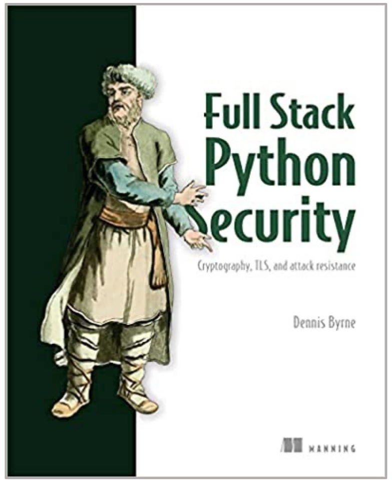
SEI SERIES • A CERT® BOOK

Secure Coding in C and C++

Second Edition

Robert C. Seacord

Foreword by Richard D. Pethia
CERT Director



Full Stack Python Security

Cryptography, TLS, and attack resistance

Dennis Byrne

MANNING

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 <u>your</u> 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:**

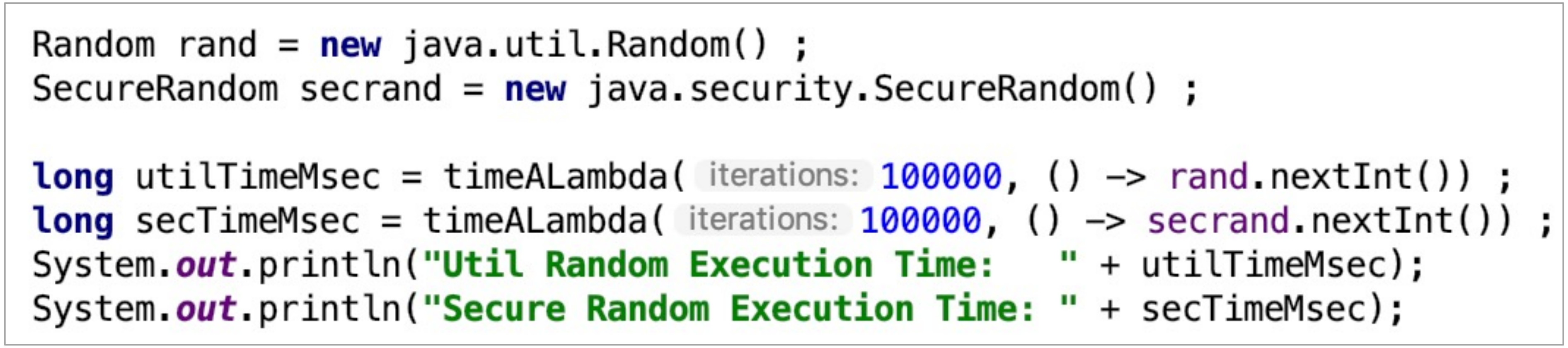
```
SELECT * FROM table1 WHERE name = ’’ OR 1=1 --
```

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?



```java
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);
```

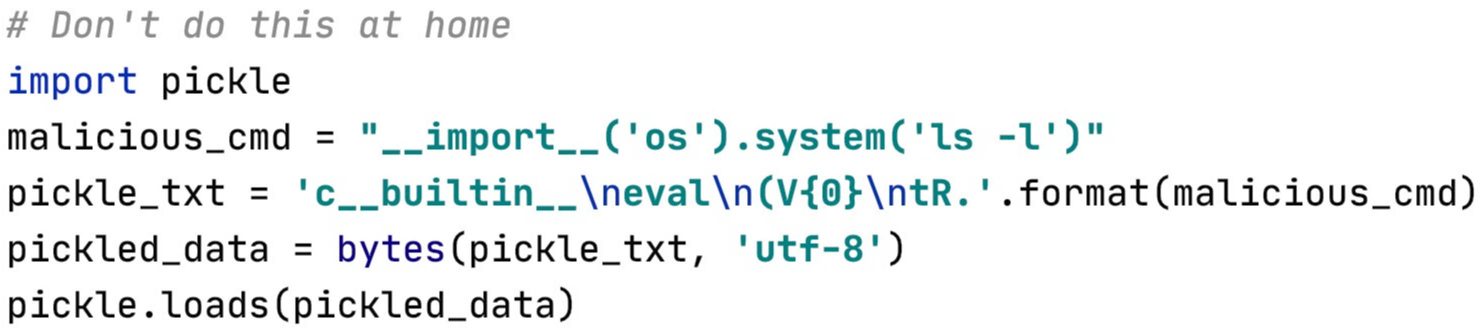
```
$> java com.artechra.RandomTest
Util Random Execution Time:   7
Secure Random Execution Time: 49
```

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!

5

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