BUILDING APPLICATIONS SECURELY

JAX London 2019

Eoin Woods
@eoinwoodz
Endava

INTRODUCTIONS

Eoin Woods

- CTO at Endava
- Career has spanned products and applications
  - Architecture and software engineering
  - Bull, Sybase, InterTrust
  - BGI (Barclays) and UBS
- Long time security dabbler
- Increasingly concerned at cyber threat for “normal” systems

CONTEXT OF THIS TALK

CONTENT

- The Threat
- Software Security is Mitigation
- Principles for Secure Implementation
- Implementation Guidelines
- Summary

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

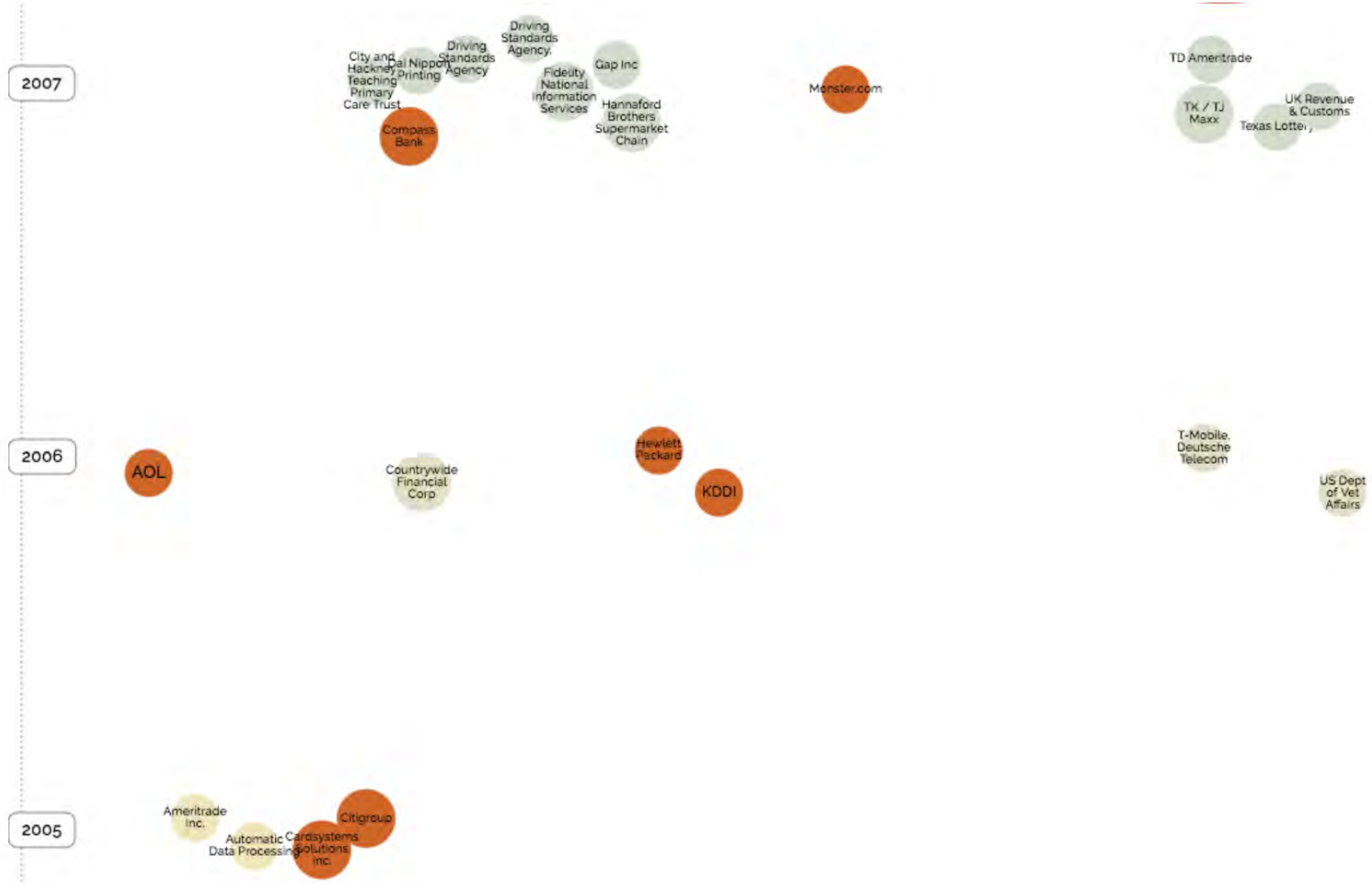
Today's **internal application** is tomorrow's "**digital channel**"

System **interfaces on the Internet**

Introspection of **APIs**

Attacks being "**weaponized**"

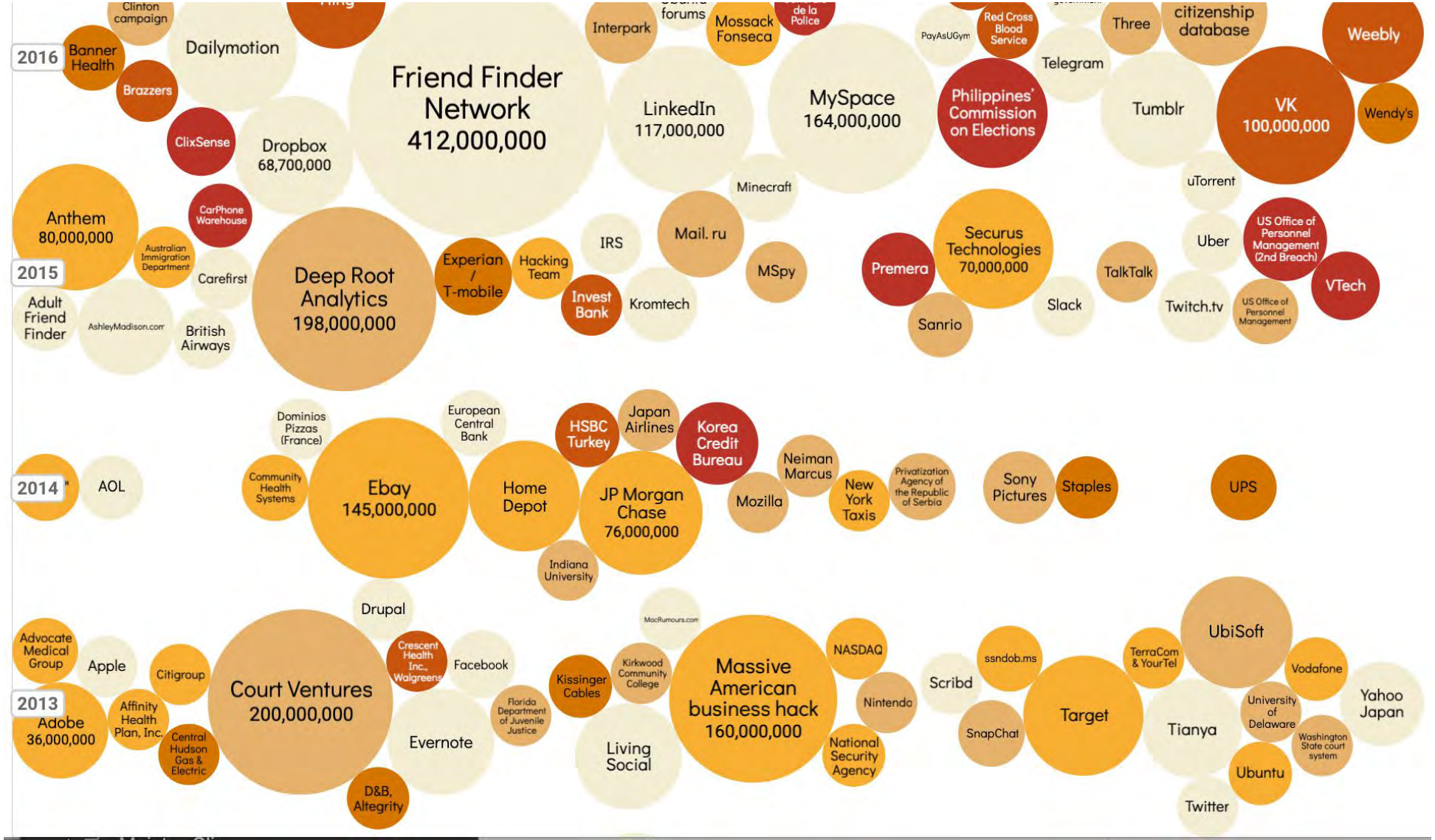
DATA BREACHES: 2005 - 2007



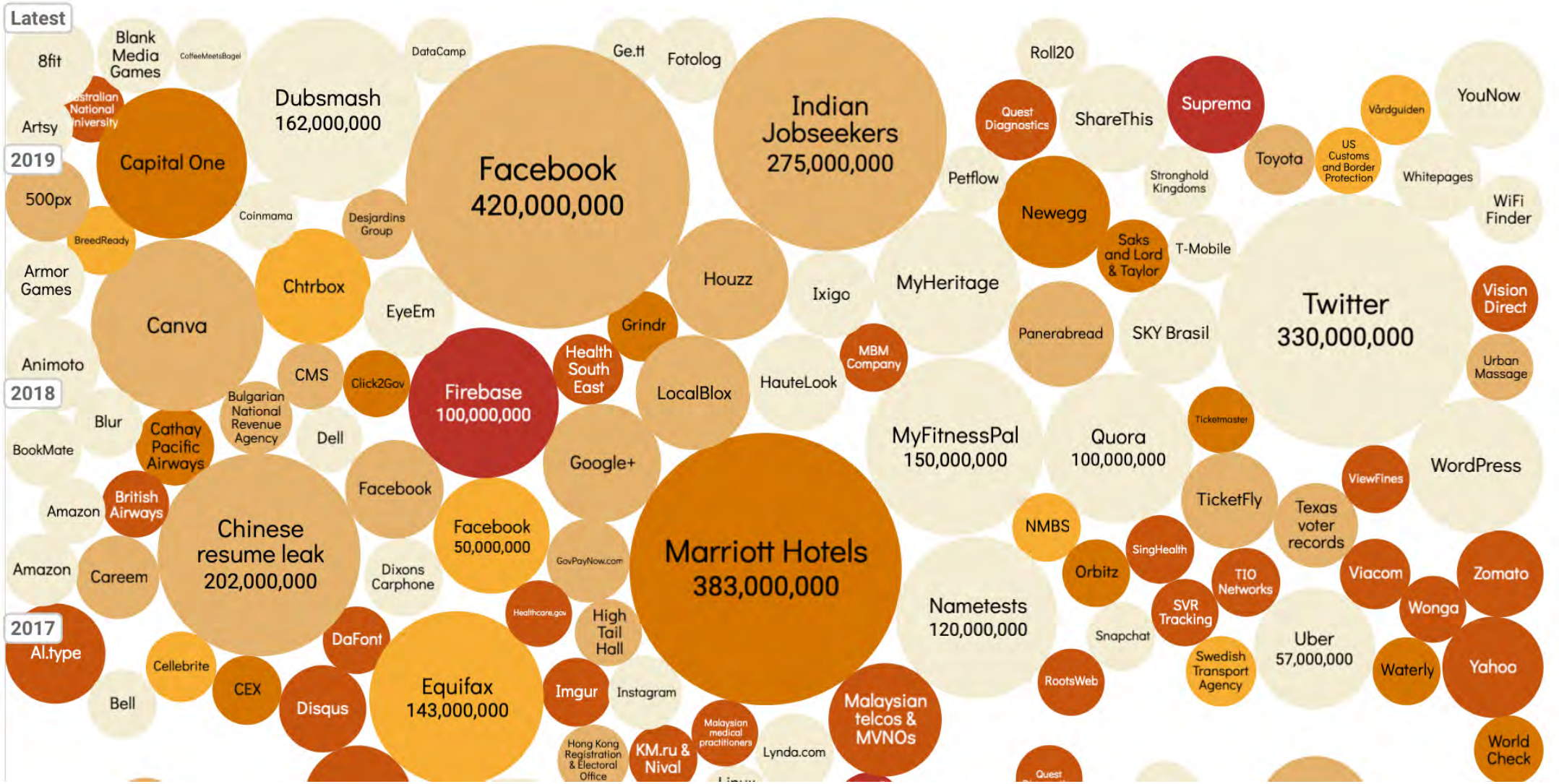
http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

DATA BREACHES: 2009 - 2012

DATA BREACHES: 2013 - 2016

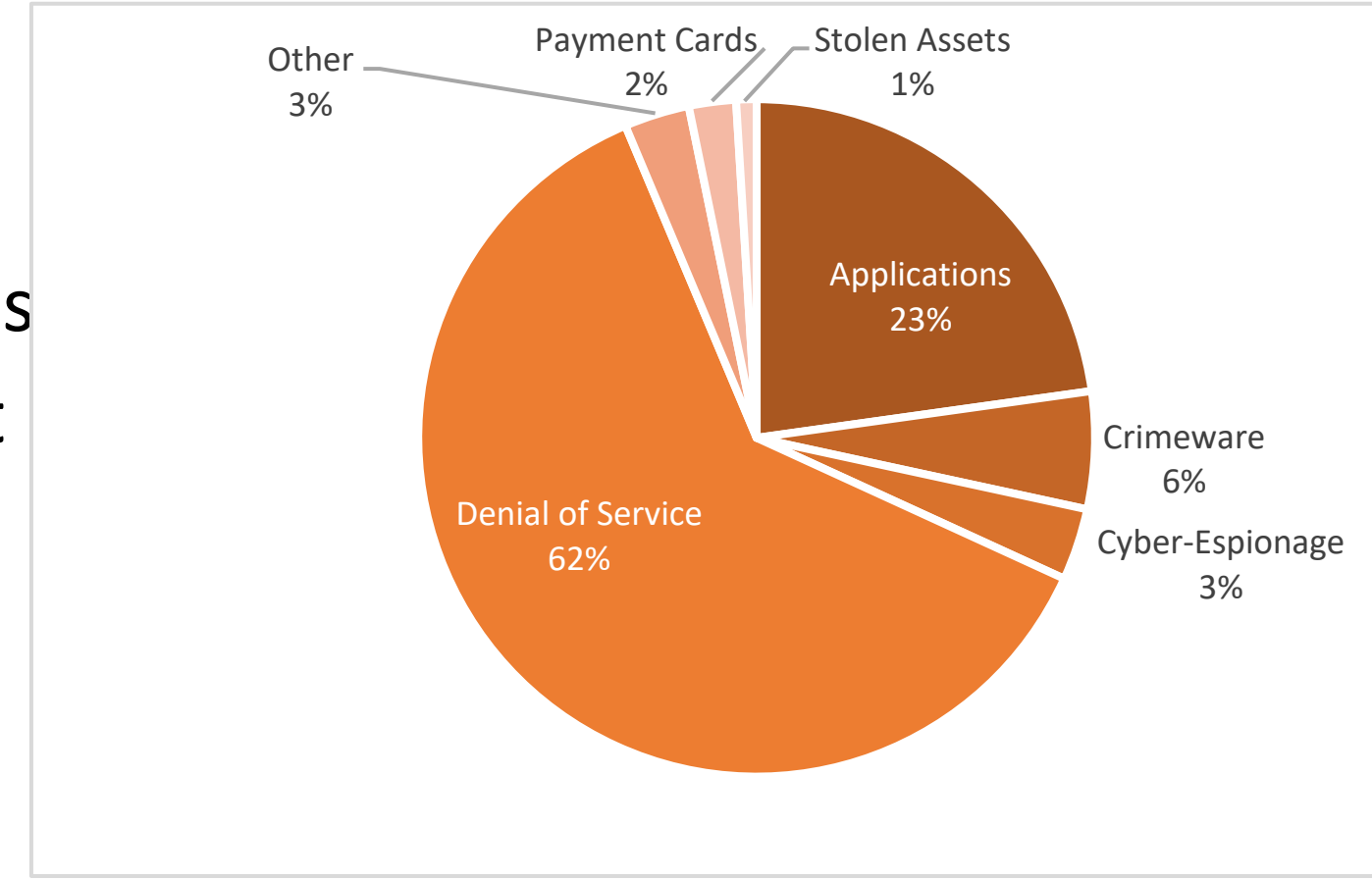


DATA BREACHES: 2017 - 2019



THE IMPORTANCE OF APPLICATION SECURITY

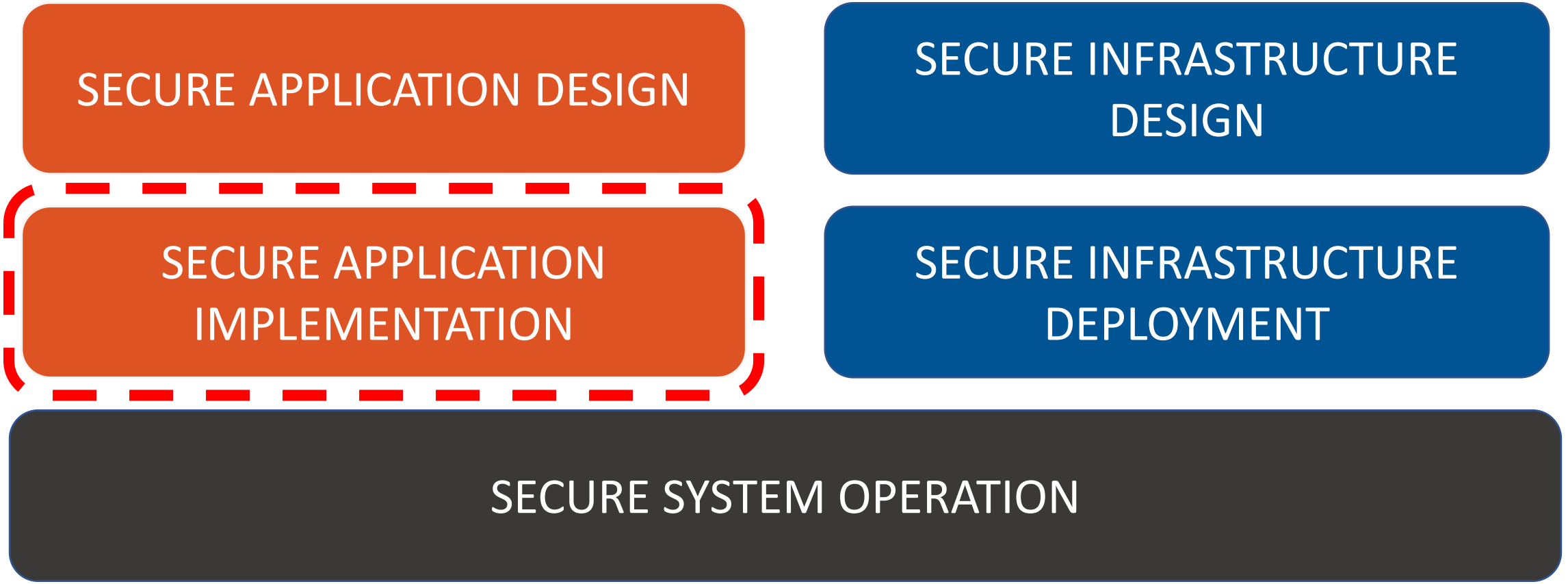
- Verizon research security incidents annually
- There are many root causes
- Applications are significant
- This study suggests that about a quarter are application related



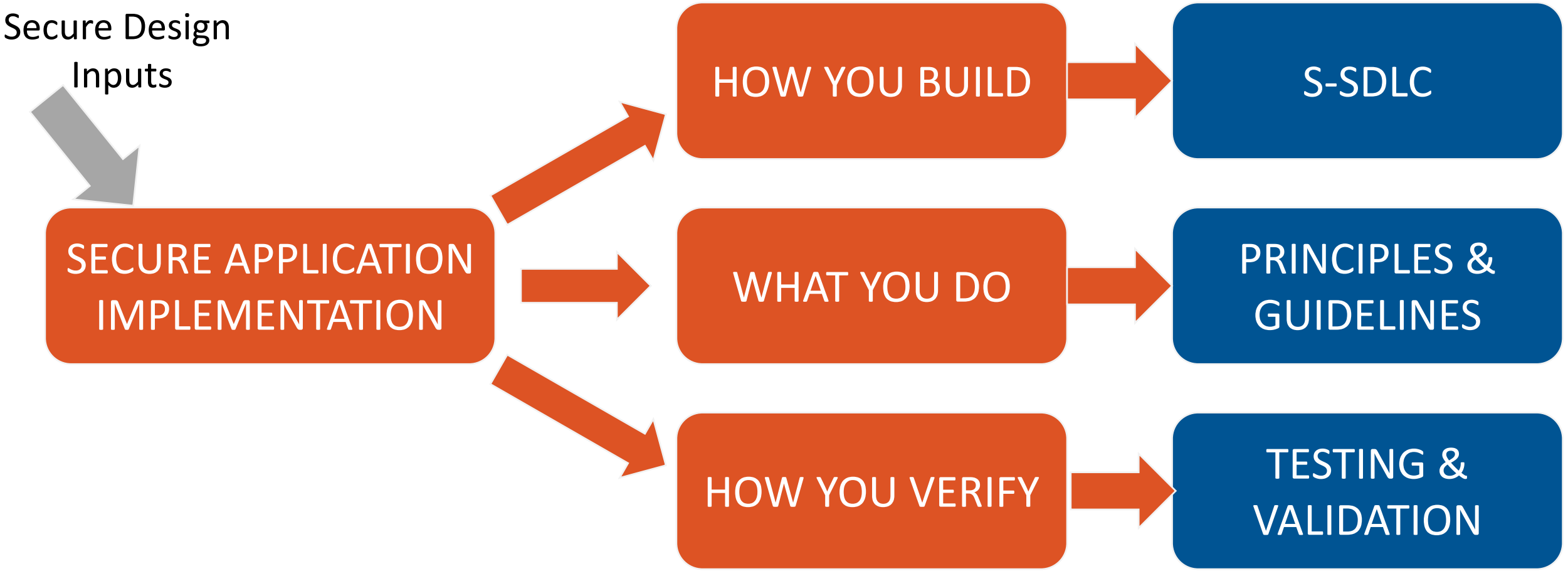
https://enterprise.verizon.com/resources/reports/dbir

SOFTWARE SECURITY IS MITIGATION

ASPECTS OF SECURITY PRACTICE



SECURE APPLICATION IMPLEMENTATION



SECURITY IN THE DEVELOPMENT LIFECYCLE

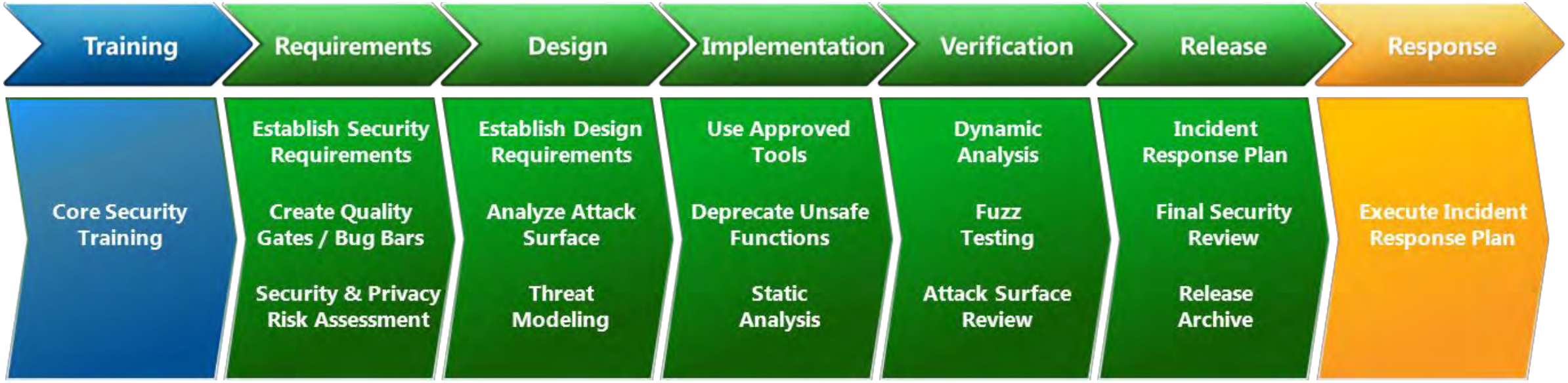
Microsoft SDL

OWASP SAMM

SAFECode Fundamental Practices

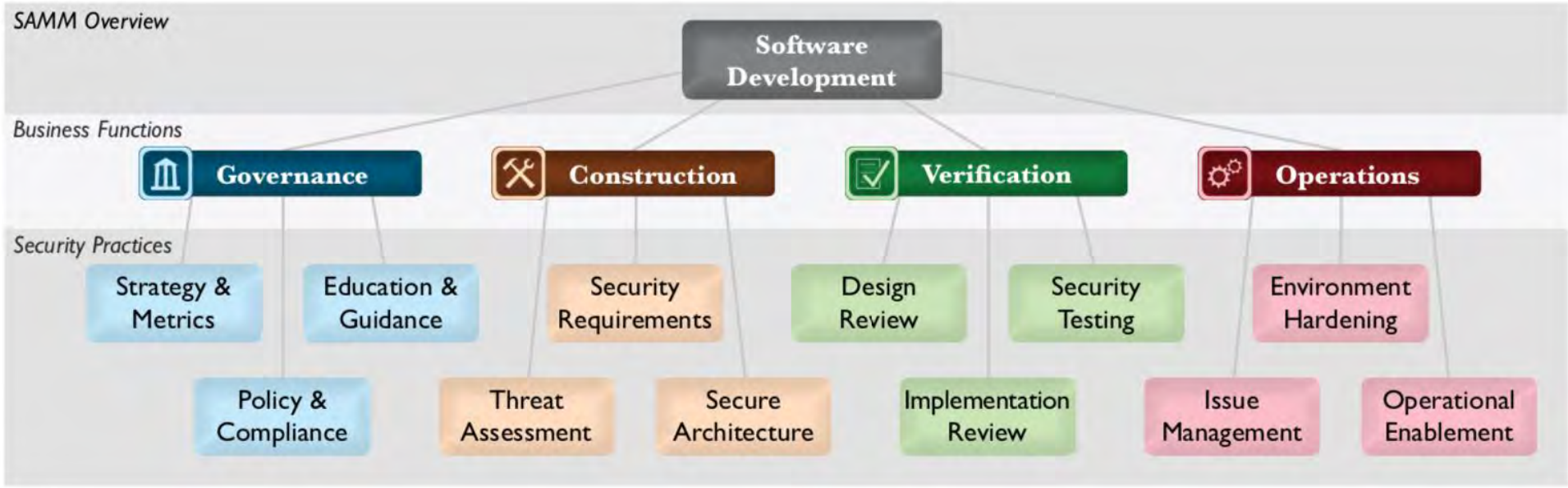
Building Security In Maturity Model

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

ADOPTING SECURITY PRACTICE

EXPERT APPLICATION SECURITY TEAM

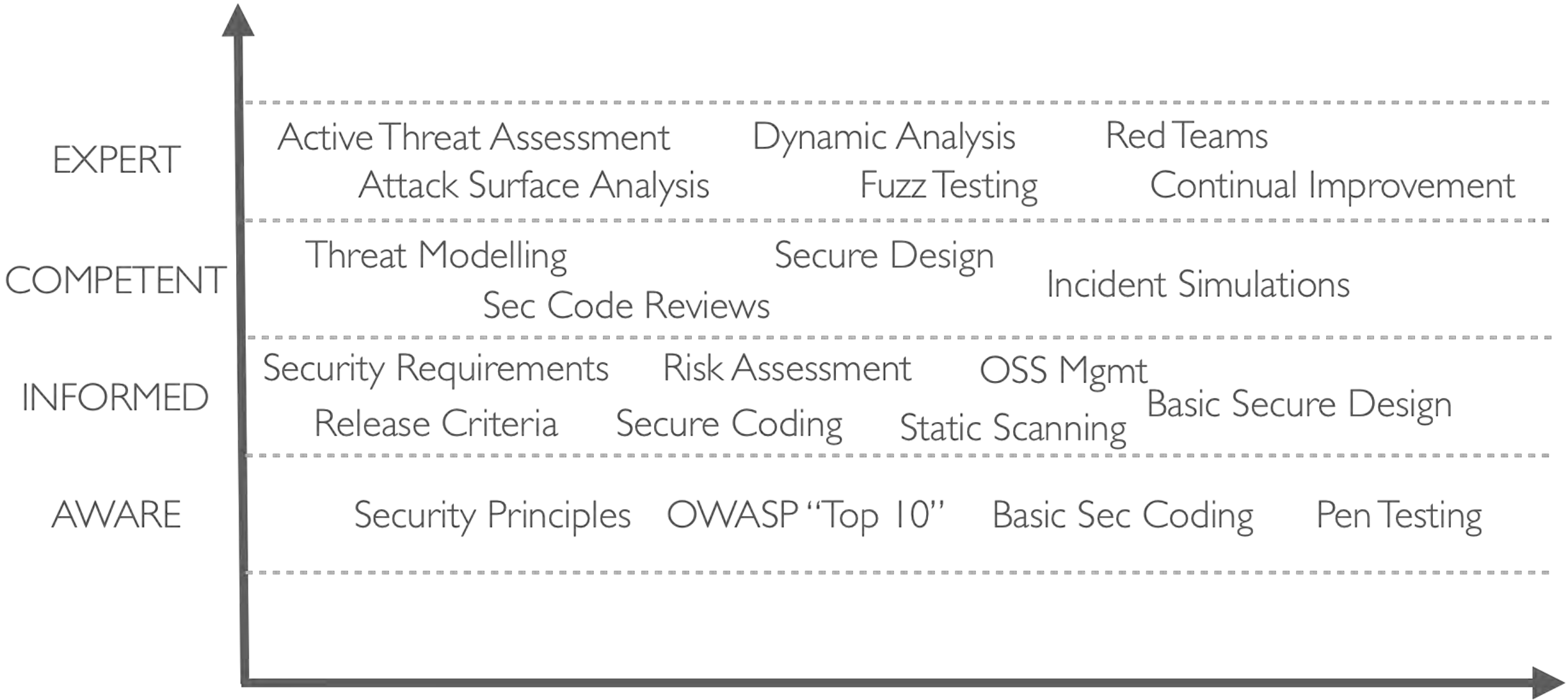
CAPABLE APPLICATION SECURITY TEAM

INFORMED APPLICATION SECURITY TEAM

SECURITY AWARE TEAM

NO SECURITY PRACTICE

EXAMPLE CAPABILITY PLAN



PRINCIPLES FOR SECURE IMPLEMENTATION

SECURE IMPLEMENTATION 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

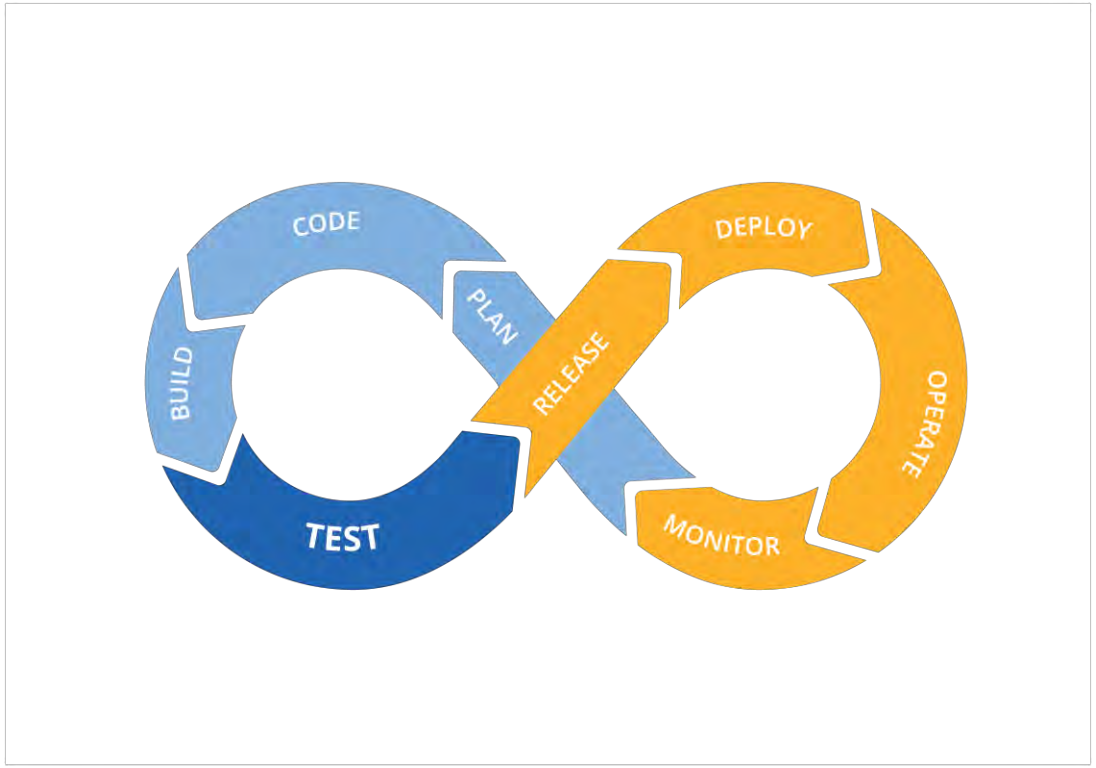
- A "concern" not a "feature"
- Needs team-wide awareness
- Avoid security being a "specialist" problem
- Integrate security awareness into normal dev tasks

SECURITY CHAMPIONS

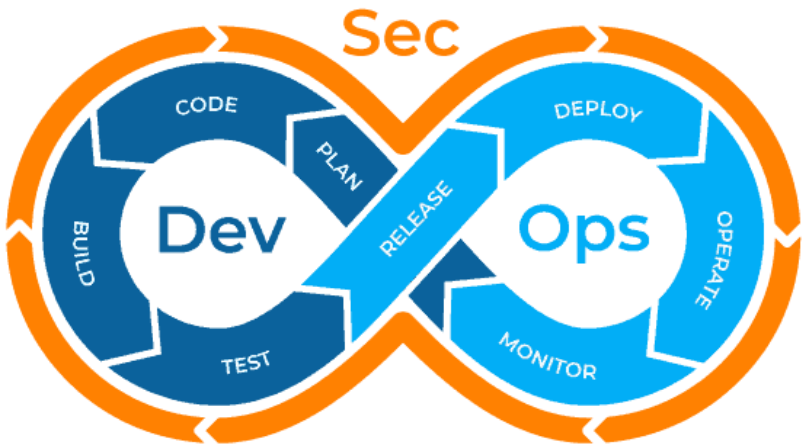
- Security is everyone's problem …
  but always someone else's
- You need enthusiastic advocates
  - People who will take ownership
- Self-selecting "security champions"
- Invest, involve, promote, support
  - don't isolate them!

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 RequestHandler extends HttpServlet {

    private OrderService orderService;

    public void init()

    public void doGet(
            throws Ser

        int qty = Inte
        String sku = r

        int ordered =

        response.getWr
    }
```

```
public class

    int quar

    public

        if (

        }

        this
    }

    // ...
}
```

```
public class RequestHandler extends HttpServlet {

    private OrderService orderService;

    public void init() throws ServletException {
        orderService = getOrderService() ;
    }

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

        int paramQty  = Integer.parseInt(request.getParameter( s: "order.quantity"));
        OrderQuantity qty = new OrderQuantity(paramQty) ;
        SkuValue sku = new SkuValue(request.getParameter( s: "order.item.sku")) ;

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

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

GOOD DESIGN IMPROVES SECURITY

```
public class RequestHandler 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 = 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 RequestHandler extends HttpServlet {

    private OrderService orderService;

    public void init() throws ServletException {
        orderService = getOrderService() ;
    }

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

        int paramQty  = Integer.par
        OrderQuantity qty = new Orc
        SkuValue sku = new SkuValue

        OrderQuantity ordered = orc

        response.getWriter().printl
    }
```

esponse)

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

    private 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
        }
        this.value = qty ;
    }
```

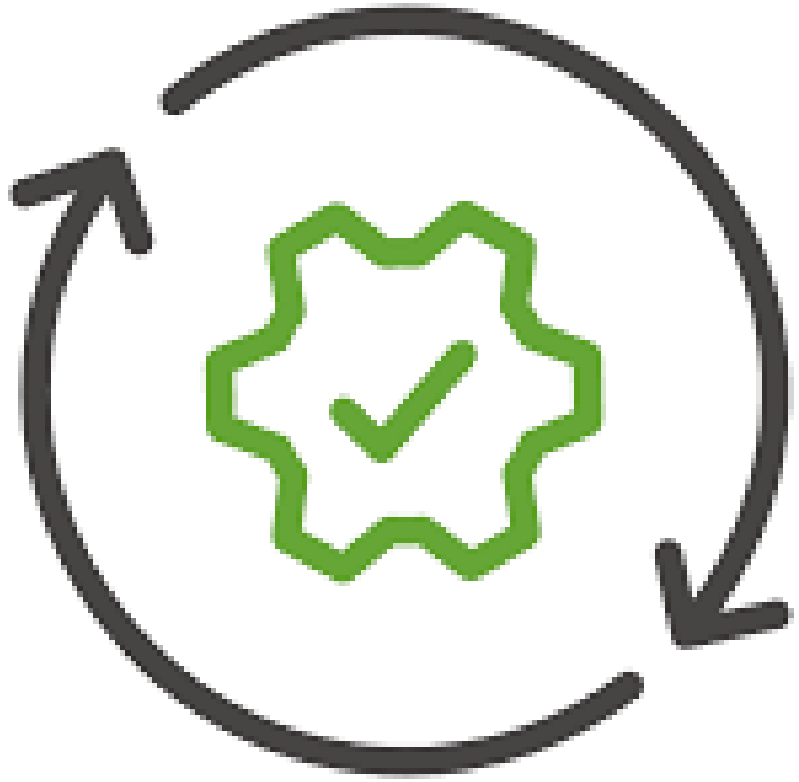
Example of DDD improving security ”for free”

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

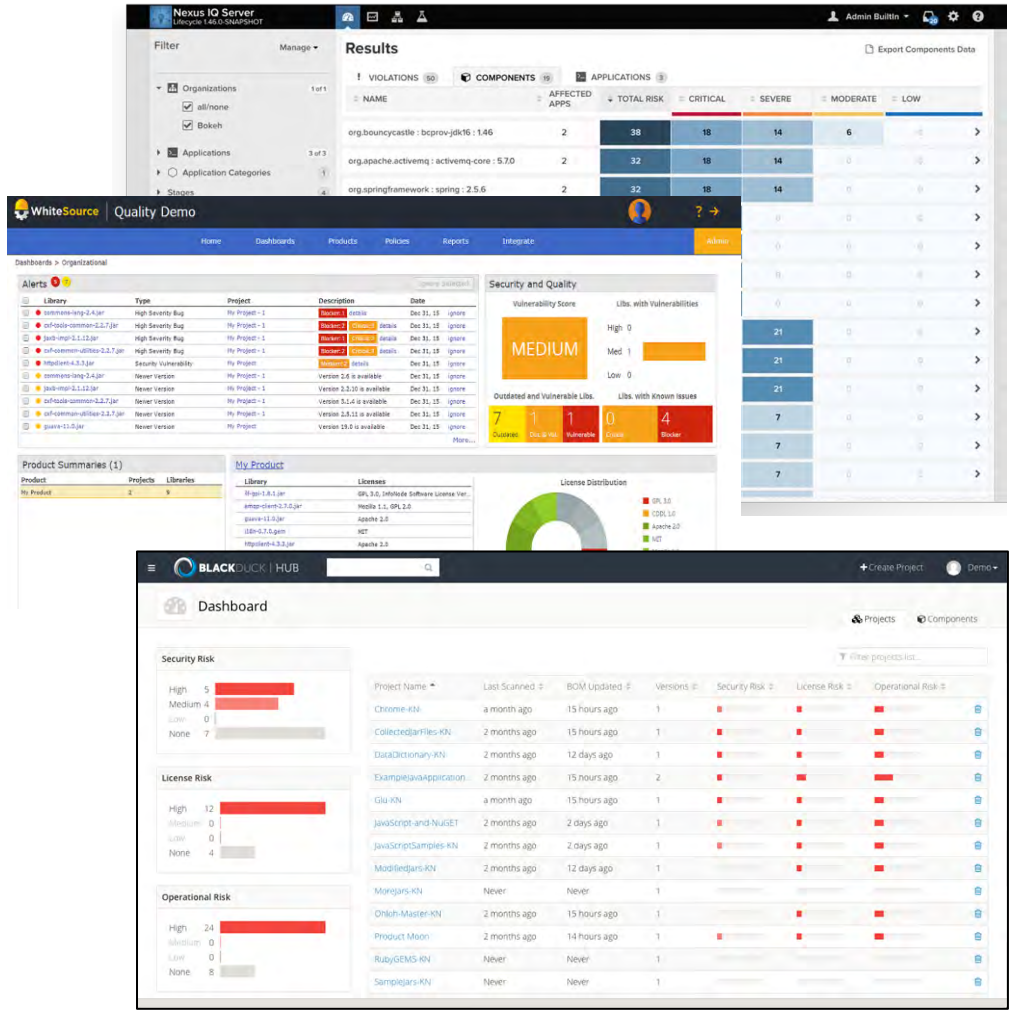
AUTOMATE SECURITY CHECKING

- 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



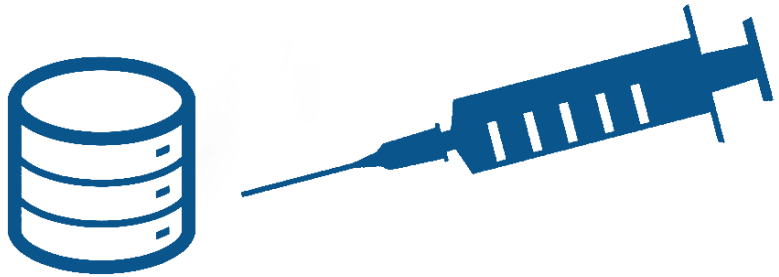
VERIFY YOUR SOFTWARE SUPPLY CHAIN

- 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 }
```

IMPLEMENTATION GUIDELINES

SECURE CODING GUIDELINES FOR JAVA

SAFE Code

OWASP Secure Coding Practices

CERT Coding Practices

Java ORACLE

SECURE CODING GUIDELINES FOR JAVA

- These 4 standards overlap significantly
- Contain both Java-specific and general advice
  - e.g. Java serialisation specifics and using “prepared” SQL statements
- Thorough documents needing 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

SECURE CODING GUIDELINES FOR JAVA

- Java is often thought of as ”secure” due to its memory model
- Some ”traditional” vulnerabilities are avoided
  - Stack buffer overflows (“smashing the stack”)
- Other common vulnerabilities are still a problem
  - XXE problems, SQL injection, pseudo–random numbers
- We’ll now look at a couple of Java-specific examples …

JAVA SPECIFIC EXAMPLES – 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
```

JAVA SPECIFIC EXAMPLES – XML ENTITY EXPANSION

Java has a lot of XML parsing, all of which expands XML entities by default – also watch out for embedded parsers (e.g. Spring MVC)

DocumentBuilderFactory, SAXParserFactory, DOM4J, TransformerFactory, Validator, SchemaFactory, XMLReader, …

For comprehensive advice check out the OWASP cheatsheet:

`https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html#java`

JAVA SPECIFIC EXAMPLES – XML ENTITY EXPANSION

Example: XMLReader (a SAX based parser)

```
public XMLReader createSafeXmlReader() throws SAXException {
    XMLReader reader = XMLReaderFactory.createXMLReader();
    reader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
    reader.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
    reader.setFeature("http://xml.org/sax/features/external-general-entities", false);
    reader.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
    return reader ;
}
```

Not a set of options most of us keep in our head!

JAVA SPECIFIC EXAMPLES – HTML SANITISATION

Example: Using OWASP Java HTML Sanitiser library

```
static final PolicyFactory POLICY = new HtmlPolicyBuilder()
        .allowStandardUrlProtocols()
        // Allow title="..." on any element.
        .allowAttributes("title").globally()
        // Allow href="..." on <a> elements.
        .allowAttributes("href").onElements("a")
        // These elements are allowed.
        .allowElements(
                "a", "p", "div", "i", "b", "em", "blockquote", "tt", "strong",
                "br", "ul", "ol", "li")
        // Custom slashdot tags.
        // These could be rewritten in the sanitizer using an ElementPolicy.
        .allowElements("quote", "ecode")
        .toFactory();

public void safeWriteHtml(Writer output, String rawHtml) {
    HtmlStreamRenderer renderer = HtmlStreamRenderer.create(output, Handler [redacted]
    HtmlSanitizer.sanitize(rawHtml, POLICY.apply(renderer));
}
```

Example of using proven 3rd party security software to solve a common problem: https://github.com/OWASP/java-html-sanitizer

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!

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 only 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 need to be addressed when using Java
  - A number of sets of guidelines exist … use them!
  - **SAFECode**, **OWASP** Secure Coding Practices, **Oracle** Secure Java 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

SECURITY RELATED ORGANISATIONS

- OWASP - The Open Web Application Security Project
  - Largely volunteer organisation, largely online
  - Exists to improve the state of software security
  - Research, tools, guidance, standards – “Top 10” lists
  - Runs local chapters for face to face meetings
- SAFE Code
  - Industry consortium of (mainly) software vendors
  - Aims to improve practice across the industry
  - Training and documentation
    - “Fundamental Practices for Secure Software Development” guide

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

SECURITY RELATED ORGANISATIONS

- MITRE Corporation
  - Common Vulnerabilities and Exposures (CVE)
  - Common Weaknesses Enumeration (CWE)

- BSIMM – Building Security In Maturity Model
  - Cigital (now Synopsys) initiative
  - Survey to reflect community practice
  - Led to report, conference and community
  - Useful baseline of practice across the industry
- There are a lot of others too (IIC, CPNI, CERT, CIS, ISSA, …)

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