Deploying Security Testing Practice

by Artem Vasiuk

About me

In testing since 2004

Test Manager in Danish company Scalepoint

From Ukraine. Live in Denmark

Love snowboarding

Why Security?

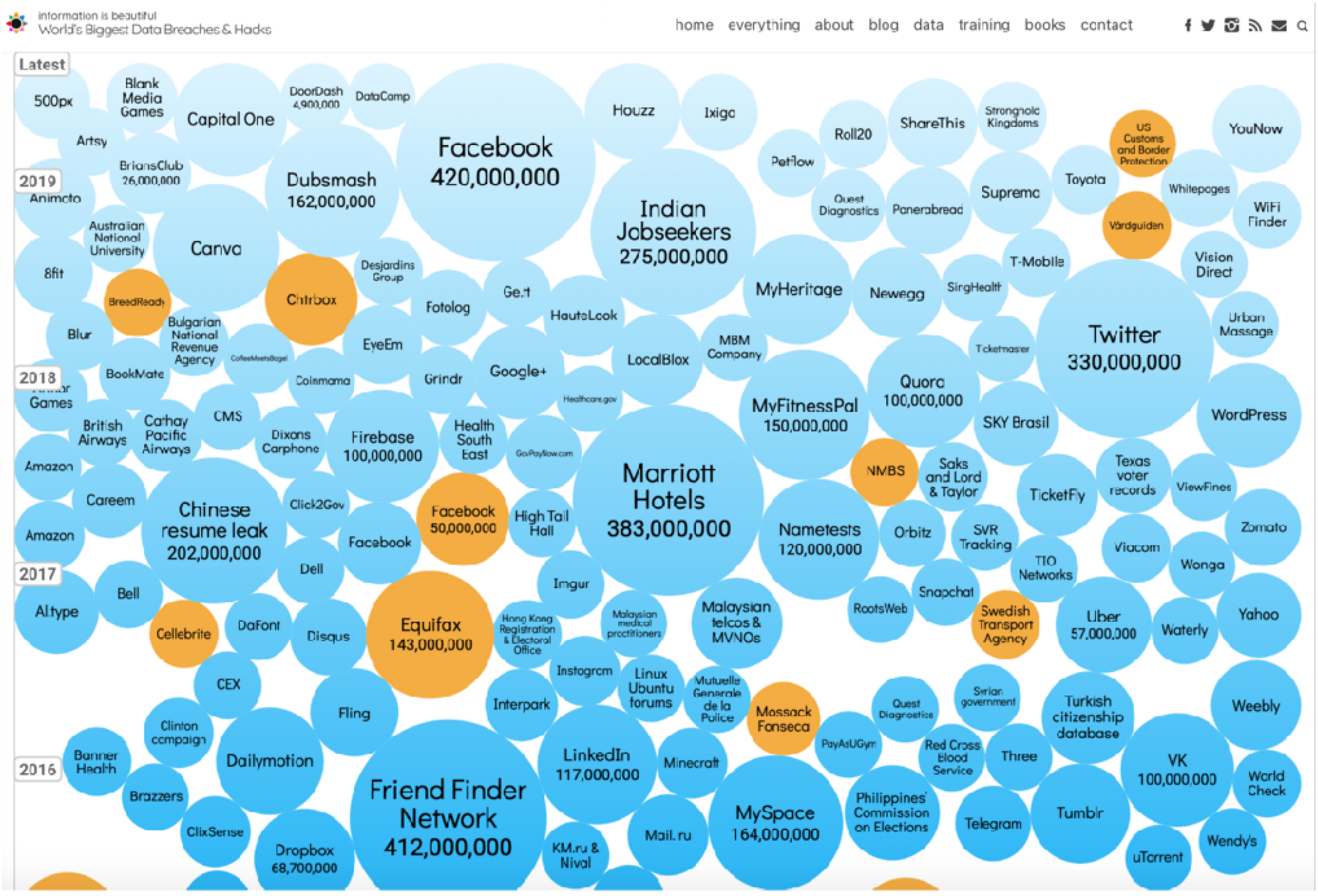
Increasing number of breaches

Numerous tools for detection and attacks

New area for personal development

High stakes due to GDPR

Source: https://informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/



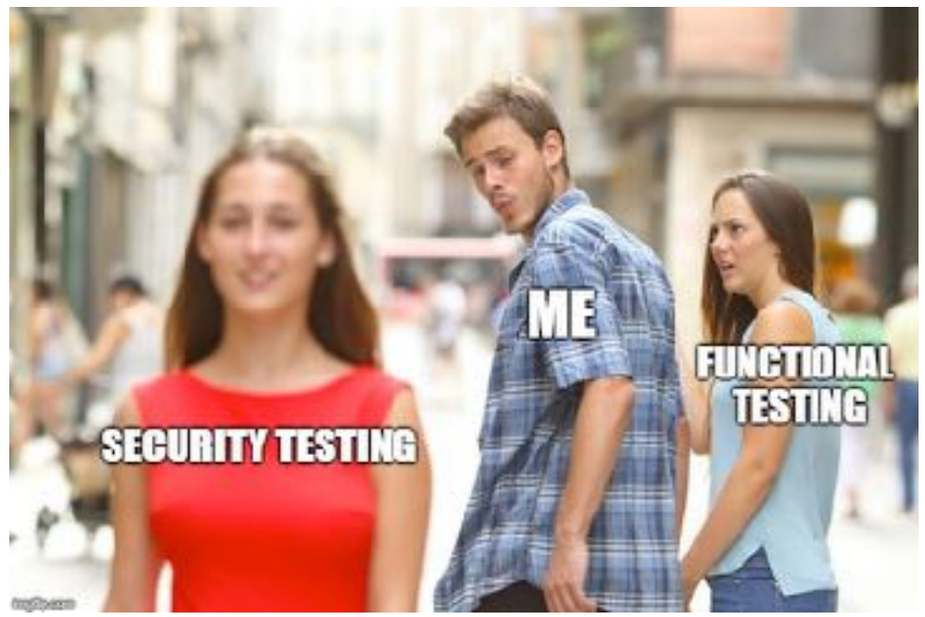
The Beginning

Let’s do it! But what’s next step?

Should we do it or delegate to professionals?

How do we get time for it?

Can robots do the stuff for us?



The Situation

What is the driver?

- Management Decision
- Need Driven
- Personal Initiative
- Career Opportunity

The Situation

How do you organise the process?

- One-man battle
- Team Work
- Corporate Goal

The Situation

Where do you focus?

- Secure Frameworks and Components
- Automated Testing Tools
- Professional Consultants

The Situation

When do you act?

- Non-Functional Testing of Features
- Design Review and Code Review
- Penetration Testing per Release
- Definition of Done

Can I help you?

OWASP.org

- Top 10 Vulnerabilities
- Testing Checklist
- App Sec Verification Standard (aka ASVS)
- Software Assurance Maturity Model (aka SAMM)

Top 10 Vulnerabilities

T10

OWASP Top 10 Application Security Risks – 2017

6

A1:2017-Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5:2017-Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6:2017-Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.
A7:2017-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not

Testing Checklist

Testing for Reflected Cross site scripting (OTG-INPVAL-001)

This article is part of the new OWASP Testing Guide v4.

Back to the OWASP Testing Guide v4 ToC:
https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents
Back to the OWASP Testing Guide Project:
https://www.owasp.org/index.php/OWASP_Testing_Project

Summary

Reflected Cross-site Scripting (XSS) occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users who open a maliciously crafted li
string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.

Reflected XSS are the most frequent type of XSS attacks found in the wild. Reflected XSS attacks are also known as non-persistent XSS attacks and, since the attack payload is delivered and executed via a single request and response, they are also referred to

When a web application is vulnerable to this type of attack, it will pass unvalidated input sent through requests back to the client. The common modus operandi of the attack includes a design step, in which the attacker creates and tests an offending URI, a social
her victims to load this URI on their browsers, and the eventual execution of the offending code using the victim's browser.

Commonly the attacker's code is written in the Javascript language, but other scripting languages are also used, e.g., ActionScript and VBScript. Attackers typically leverage these vulnerabilities to install key loggers, steal victim cookies, perform clipboard theft, a
download links).

One of the primary difficulties in preventing XSS vulnerabilities is proper character encoding. In some cases, the web server or the web application could not be filtering some encodings of characters, so, for example, the web application might filter out "<script>",
simply includes another encoding of tags.

ASVS

Category:OWASP Application Security Verification Standard Project

Help

ASVS V2 Authentication

V2: Authentication Verification Requirements

Control Objective

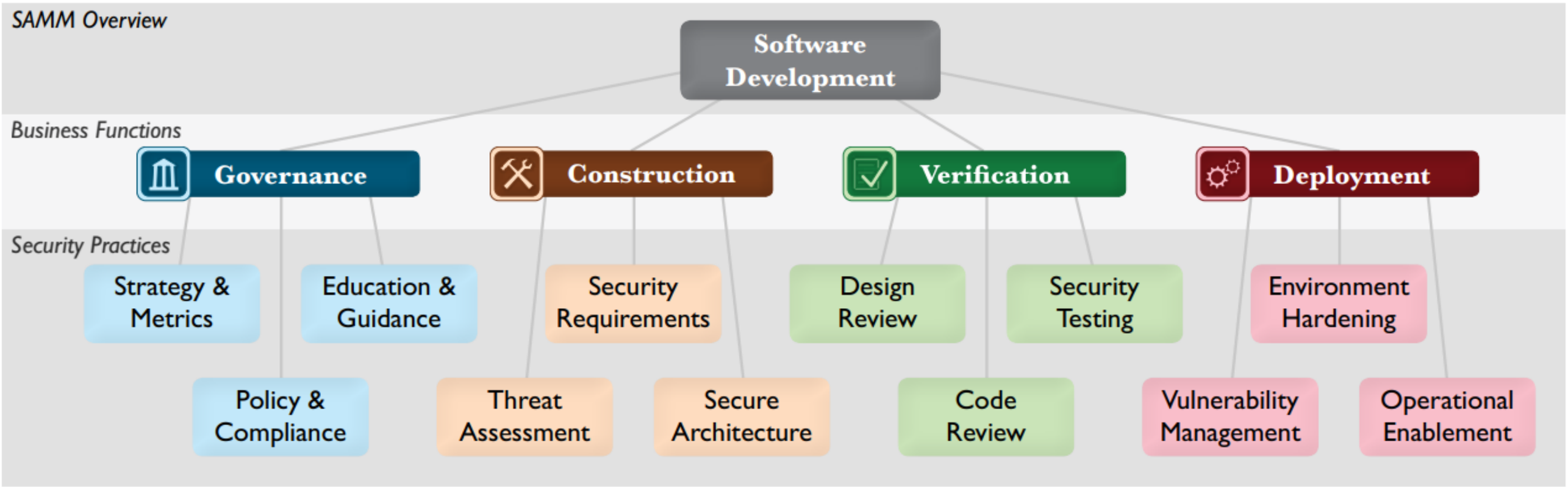
Authentication is the act of establishing, or confirming, something (or someone) as authentic, that is, that claims made by or about the thing are true. Ensure that a verified application satisfies the following high level requirements:

- Verifies the digital identity of the sender of a communication.
- Ensures that only those authorised are able to authenticate and credentials are transported in a secure manner.

Security Verification Requirements

#	Description	L1	L2	L3
2.1	Verify all pages and resources are protected by server-side authentication, except those specifically intended to be public.	✓	✓	✓
2.2	Verify that the application does not automatically fill in credentials – either as hidden fields, URL arguments, Ajax requests, or in forms, as this implies plain text, reversible or de-cryptable password storage. Random time limited nonces are acceptable as stand ins, such as to protect change password forms or forgot password forms.	✓	✓	✓
2.6	Verify all authentication controls fail securely to ensure attackers cannot log in.	✓	✓	✓
2.7	Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases or highly complex passwords being entered.	✓	✓	✓
2.8	Verify all identity functions (e.g. forgot password, change password, change email, manage 2FA token, etc.) have the security controls, as the primary authentication mechanism (e.g. login form).	✓	✓	✓
2.9	Verify that the changing password functionality includes the old password, the new password, and a password confirmation.	✓	✓	✓
2.12	Verify that all authentication decisions can be logged, without storing sensitive session identifiers or passwords. This should include requests with relevant metadata needed for security investigations.		✓	✓
2.13	Verify that account passwords are one way hashed with a salt, and there is sufficient work factor to defeat brute force and password hash recovery attacks.		✓	✓
2.16	Verify that all application data is transmitted over an encrypted channel (e.g. TLS).	✓	✓	✓
2.17	Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user. A one time password reset link should be used instead.	✓	✓	✓
2.18	Verify that information enumeration is not possible via login, password reset, or forgot account functionality.		✓	✓

SAMM


SAMM Overview
Software Development
Business Functions
Governance
Construction
Verification
Deployment
Security Practices
Strategy & Metrics
Education & Guidance
Security Requirements
Design Review
Security Testing
Environment Hardening
Policy & Compliance
Threat Assessment
Secure Architecture
Code Review
Vulnerability Management
Operational Enablement

Improve Efficiency

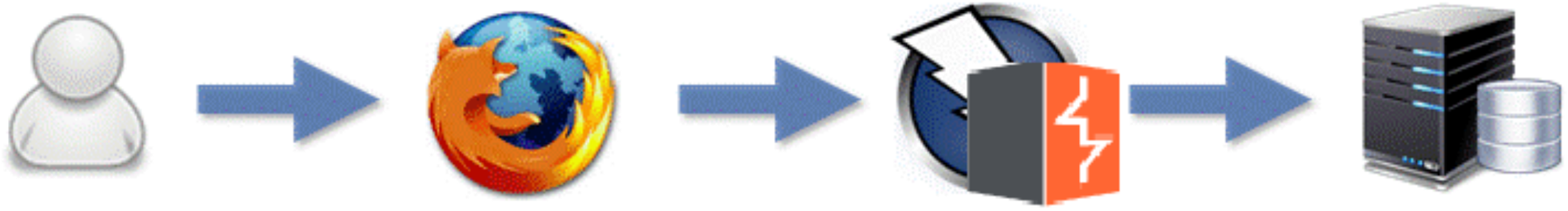
Automated Sec Scanning tools

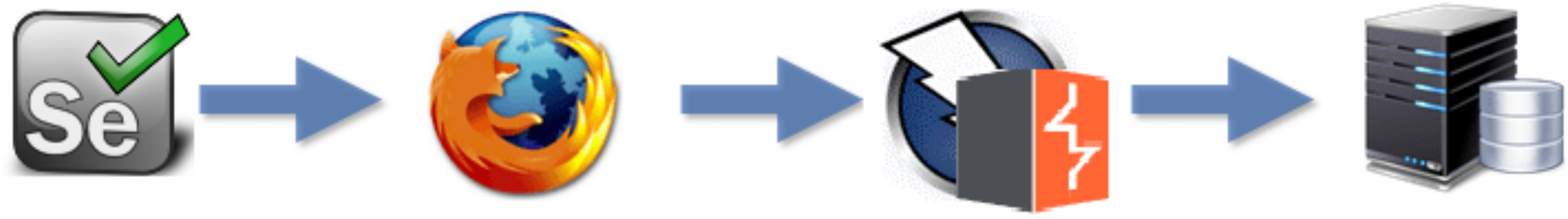
Burp Suite, Zed Attack Proxy (ZAP), AppScan

SonarQube (Java, C#, JS... +DependencyChecker)

Security test data in Automated tests

Scanning Setup





Challenges

"Who would even hack us? People are nice!"

"We have firewall (https, kaspersky, _____) to protect us!"

"We have Michael. He is responsible for Security"

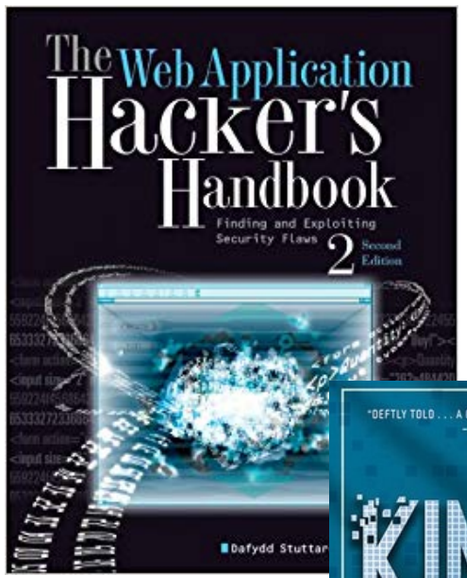
"I have no time for learning"

"Security is technical, so do it in your Technical backlog"

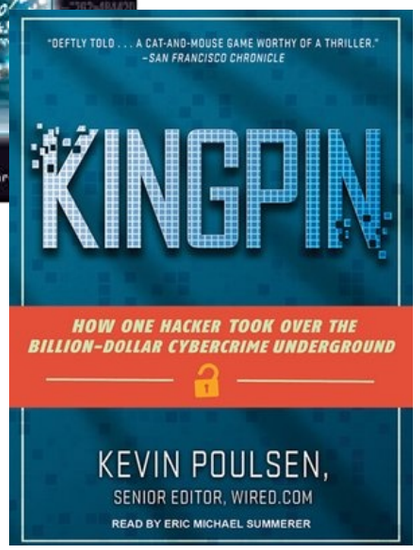
"Ok, ok…But let’s do release now and improve Security later"

Must read

- The Web Application Hacker's Handbook



- Kingpin: How One Hacker Took Over the Billion-Dollar Cybercrime Underground



Must see

Pluralsight course

- "Hack yourself first" by Troy Hunt

- "WebApp Penetration Testing" by Sunny Wear

The End

Thank you!