Secure Development Lifecycle

Secure Development Lifecycle

❌ Identifying errors in late lifecycle phases makes them more expensive to fix or mitigate. [^1]

❌ Having an unpublished or informal Secure Development Lifecycle will not be successful. [^2]

✓ Security must be embedded into all stages of the Software Development Lifecycle to be effective. [^2]

✓ A close connection with the right expert and management drive from the beginning are both mandatory. [^3]

Spaghetti Analogy

Sprinkling security on insecurely written software is equivalent to sprinkling salt on spaghetti 🍝 after cooking them in unsalted water 🚰.

Estimates of Relative Cost Factors of Correcting Errors

Introduction of Error	Requirements / Design	Coding / Unit Test	Integration / System Test	Early Access / Beta Test	Post-Release
Requirements / Design	x1	x5	x10	x15	x30
Coding / Unit Test		x1	x10	x20	x30
Integration / System Test			x1	x10	x20

NIST: The Economic Impacts of Inadequate Infrastructure for Software Testing, 2002

Example: Microsoft SDL

Phase	Practice
Training	Core Security Training
Requirements	Establish Security Requirements, Create Quality Gates/Bug Bars, Perform Security and Privacy Risk Assessments
Design	Establish Design Requirements, Perform Attack Surface Analysis/Reduction, Use Threat Modelling
Implementation	Use Approved Tools, Deprecate Unsafe Functions, Perform Static Analysis

Phase	Practice
Verification	Perform Dynamic Analysis, Perform Fuzz Testing, Conduct Attack Surface Review
Release	Create an Incident Response Plan, Conduct Final Security Review, Certify Release and Archive
Response	Execute Incident Response Plan

Security Requirements

Derive Security Requirements from Business Functionality

- Gather and review functional requirements
- For each functional requirement derive relevant security requirements
  - Lead stakeholders through explicitly noting security expectations
    - e.g. data security, access control, transaction integrity, criticality of business function, separation of duties, uptime etc.
  - Follow the same principles for writing good requirements in general
    - i.e. they should be specific, measurable, and reasonable

Security and Compliance Guidance for Requirements

- Determine industry best-practices that project teams should treat as requirements
  - e.g. publicly available guidelines, internal or external guidelines/standards/policies, or established compliance requirements
- Do not attempt to bring in too many best-practice requirements into each development iteration
- Slowly add best-practices over successive development cycles

Protection Requirements ("Schutzbedarf" 🇩🇪) Calculator

- Provides an idea of the expected effort for security topics
- Serves as a starting point for detailed requirements analysis
- Formalizes the "gut-feeling" of business and IT stakeholders
- Covers all **CIA** triad aspects in a high-level fashion
  - **C**onfidentiality: Information classification, Compliance requirements
  - **I**ntegrity: Authentication mechanism, Compliance requirements
  - **A**vailability: Business criticality, Exposure to threats

Requirements Score Table

Aspect	🔴 (=5)	🔶 (=2)	🔸 (=1)	💚 (=0)
Business criticality	Mission Critical	Business Critical	Business Operational	Administrative Service
Information classification	Secret	Confidential	Internal	Public
Compliance requirements	Legal	Industry	Customer	None
Exposure to threats	Internet-facing		Internal Web	Desktop / Batch
Authentication mechanism		🔹 (+/-0) None	🔷 (-1) Proprietary	💙 (-2) Centralized

Protection Requirements Rating Evaluation

$$TotalScore = Min(0, (BusinessCriticality + InformationClassification + ComplianceRequirement) + ExposureToThreats + AuthenticationMechansim))$$

Total Score	PR Group
10 - 20	High
5 - 9	/ Medium
0 - 4	Low

Exercise 9.1

1. Calculate the Total Score and Rating for the applications of fictive *Juice Shop Inc.* (Fill any gaps with reasonable assumptions)
2. Repeat for at least one additional system from your own company

Aspect / Application	Website	VCS	Webshop	B2B API
Business criticality				
Information classification				
Compliance requirements				
Exposure to threats				
Authentication mechanism				

Secure Design Principles

Minimize Attack Surface Area	Don't trust Services
Establish Secure Defaults	Separation of Duties
Principle of Least Privilege	Avoid Security by Obscurity
Principle of Defense in Depth	Keep Security simple
Fail securely	Fix Security Issues correctly

Minimize Attack Surface Area

- **Every feature** that is added to an application **adds a certain amount of risk** to the overall application
- The aim for secure development is to **reduce** the overall risk by reducing **the attack surface area**

Establish Secure Defaults

- The **"out-of-box"** experience for the user **should be secure**
- It should be up to the user to reduce their security Ã¢â‚¬â€œ if they are allowed

Principle of Least Privilege

- Accounts have the **least amount of privilege required** to perform their business processes
- This encompasses user rights and resource permissions, e.g.
  - CPU limits
  - memory
  - network
  - file system

Principle of Defense in Depth

- Where one control would be reasonable, **more controls that approach risks in different fashions** are better
- In-depth-controls can make severe vulnerabilities extraordinarily difficult to exploit

Fail securely

- Whenever a transaction fails or code execution throws an exception it should **always "fail closed"** and never "fail open"

Don't trust Services

- Third party partners more than likely have differing security policies and posture
- Implicit **trust of externally run systems is not warranted**
- All external systems should be treated in a similar fashion

Separation of Duties

- Separation of duties is a key **fraud control**
- **Administrators should not also be users** of an application they are responsible for

Avoid Security by Obscurity

- Security through obscurity is a weak security control, and nearly always fails when it is the only control
- The **security** of key systems **should not be reliant upon keeping details hidden**

Keep Security simple

- **Attack surface and simplicity go hand in hand**
- Prefer straightforward and simple code over complex and over-engineered approaches
- Avoid the use of double negatives and complex architectures when a simpler approach would be faster and simpler

Fix Security Issues correctly

- Once a security issue has been identified, it is important to develop a test for it, and to **understand the root cause** of the issue
- It is likely that the security issue is widespread amongst all code bases, so **developing the right fix without introducing regressions** is essential

Secure Coding Guidelines

🔧 TODO

Security Testing

(SAST, DAST)

🔧 **TODO**

Security Logging & Monitoring

Insufficient Logging & Monitoring

- Exploitation of **insufficient logging and monitoring is the bedrock of nearly every major incident**
- **Attackers** rely on the lack of monitoring and timely response to **achieve their goals without being detected**
  - Most successful attacks start with vulnerability probing
  - Allowing such probes to continue can raise the likelihood of successful exploit to nearly 100%

Examples of Insufficiencies

- Auditable events, such as logins, failed logins, and high-value transactions are not logged
- Warnings and errors generate no, inadequate, or unclear log messages
- Logs of applications and APIs are not monitored for suspicious activity
- Logs are only stored locally
- Appropriate alerting thresholds and response escalation processes are not in place or effective

Risk Rating

Insufficient Logging & Monitoring

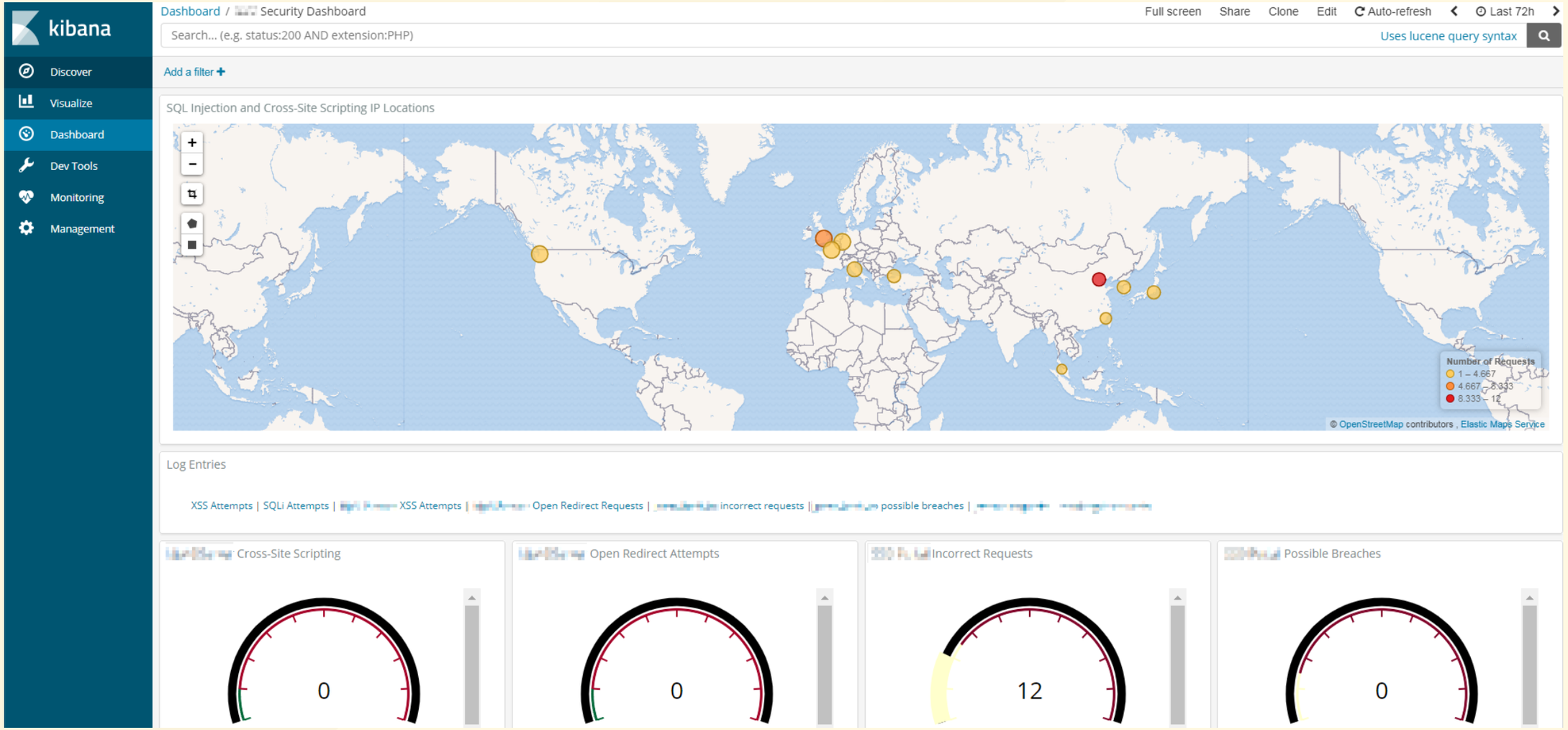
Exploitability	Prevalence	Detecability	Impact	Risk
🔶 Average	🔴 Widespread	🔸 Difficult	🔶 Moderate	A10
(**2**	+ **3**	+ **1**) / 3	* **2**	= **4.0**

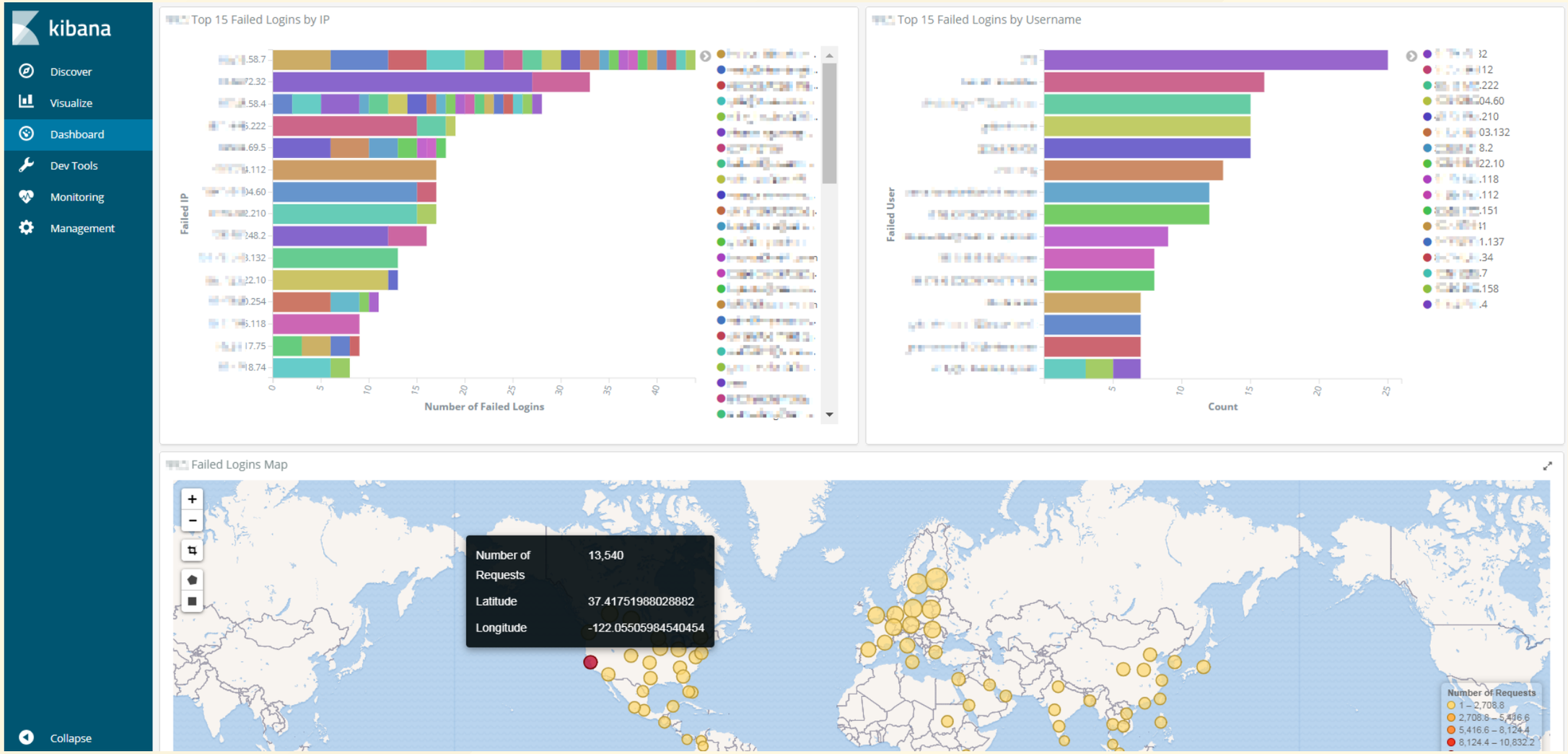
Prevention

- Ensure all login, access control failures, and server-side input validation **failures can be**
  - **logged with sufficient user context** to identify suspicious or malicious accounts
  - **held for sufficient time** to allow delayed forensic analysis
- Ensure that logs are generated in a format that can be easily consumed by a **centralized log management solution**
  - e.g. Elastic Stack (Kibana, Elasticsearch, Logstash & Beats)

- Ensure **high-value transactions have an audit trail** with integrity controls to prevent tampering or deletion
  - e.g. append-only database tables or similar
- Establish effective monitoring and alerting such that **suspicious activities are detected** and responded to **in a timely fashion**
- Establish or adopt an incident response and recovery plan

Example Kibana Security Dashboard





AppSec Pipeline

🔧 TODO