Western Norway
University of
Applied Sciences

DAT152 – Advanced Web Applications

Web Security - Introduction

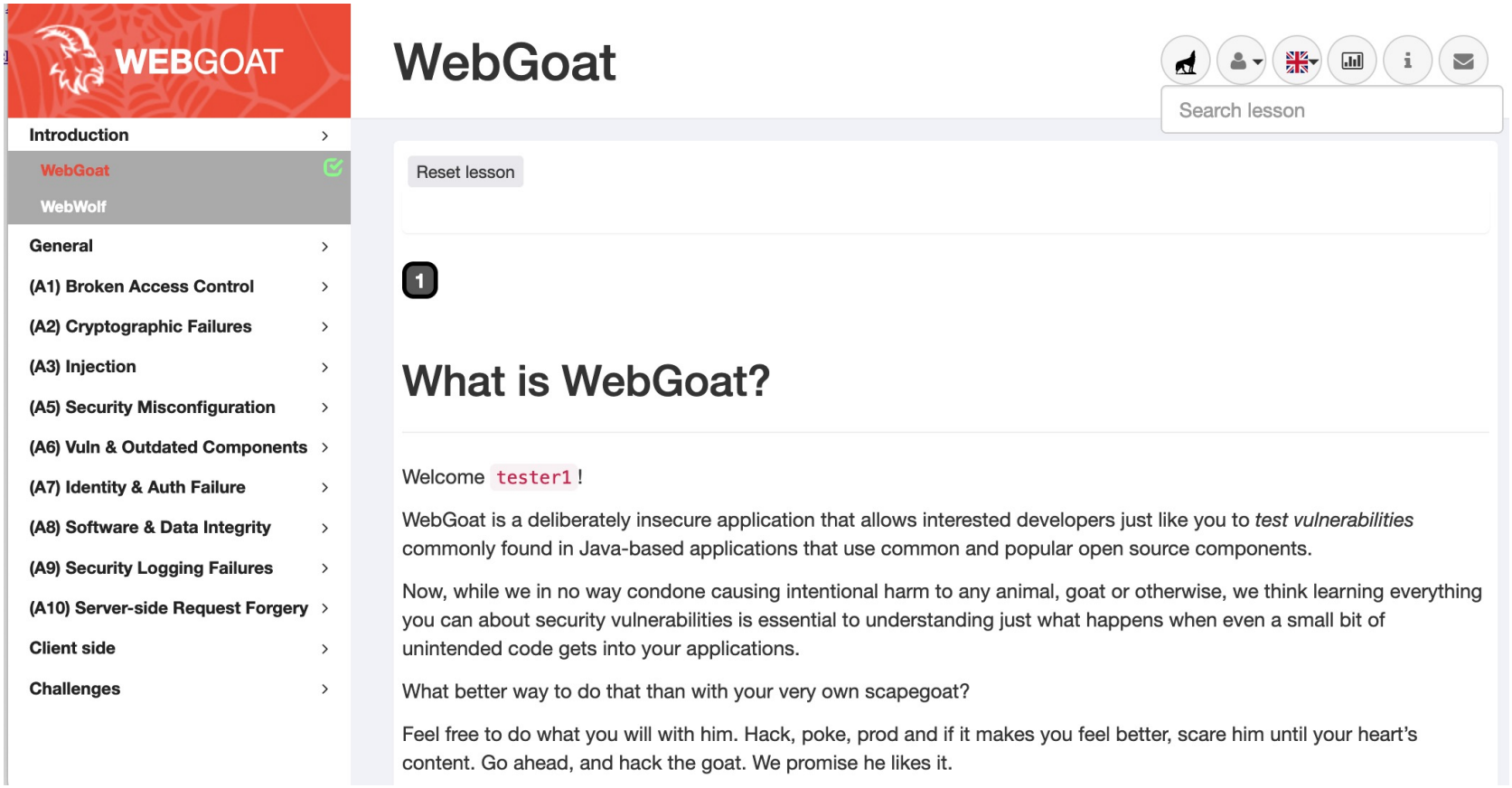
Main Goal

- To be aware of web application top security risks and how to mitigate them
- You should know the correct approach to secure design, coding & testing
- You should know the approaches to identify vulnerability at different SSDLC stages
  - Abuse/Misuse case/Protection Poker (requirement)
  - Threat Modelling (Design)
  - Code review/Static Analysis (implementation)
  - Pen/Dynamic Analysis (runtime)

Practical information

- Lectures (Mainly): OWASP Top 10 Risks
- Language: Java
- IDE: Eclipse (you are free to use other IDEs)
- Exercises & Labs: WebGoat Lessons +/- PortSwigger
  - Visit https://github.com/WebGoat/WebGoat
- Oblig. 4: Testing Vulnerable web applications (Manual, static, dynamic) with a structured report
  - More to come later

Webgoat - Exercises



https://portswigger.net/web-security/all-topics

Tools & Resources

- **WebGoat Lessons**
  - https://github.com/WebGoat/WebGoat
  - There is Docker option. Follow the instructions to start it up
- **PortSwigger Lessons (Server-side)**
  - https://portswigger.net/web-security/all-topics
- **OWASP Testing Guide**
  - https://owasp.org/www-project-web-security-testing-guide/stable/
  - https://www.owasp.org/index.php/OWASP_Testing_Project
- **SpotBugs/FindSecBugs**
  - IDE configured
  - Instructions @ https://find-sec-bugs.github.io/
- **OWASP ZAP**
  - Instructions @ https://www.zaproxy.org/
- **Burp suite**
  - https://portswigger.net/burp/communitydownload
- **Mitigations & APIs**
  - https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java
  - https://cheatsheetseries.owasp.org/Glossary.html
  - https://owasp.org/www-project-java-encoder/
- **Dependency-check (optional)**
  - https://github.com/google/osv-scanner
  - https://www.owasp.org/index.php/OWASP_Dependency_Check
- **Threat Modelling Tool (optional)**
  - **https://www.threatdragon.com/#/**

Tentative schedule

- **Lecture F28 (20.10)**: Overview and Introduction to Web Security
- ***Exercise F29 (20.10)***: Lab/Practice on WebGoat - Setup tools, environments, complete introduction exercises (+/- PortSwigger exercises)
- **Lecture F30 (24.10)**: Injection Attacks (SQL Injection, Command Injection, XXE)
- **Lecture F31 (27.10):** Security Tools – Static and Penetration Testing I
- ***Exercise F32 (27.10)***: Lab/Practice on WebGoat - Injection
- **Lecture F33 (31.10)**: Broken Authentication I - Session Management
- ***Oblig-4 F38***: Early introduction
- **Lecture F34 (03.11)**: Broken Authentication II - SSO Session/Token Management
- ***Exercise F35 (03.11)***: Lab/Practice on WebGoat - Broken Authentication
- **Lecture F36 (07.11)**: Broken Authentication III - Password Security
- ***Oblig-4 F38 (07.11)*:** Vulnerable web application testing (Manual, static, & dynamic security testing)–Written report []
- **Lecture F39 (10.11)**: Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF)
- **Lecture F39 (14.11):** Security Tools – Static and Penetration Testing II (LLM)

Security Repositories

- CWE – Common Weaknesses Enumeration
  - https://cwe.mitre.org/
- CVE – Common Vulnerability
  - https://cve.mitre.org/
- Exploit Database
  - https://www.exploit-db.com/
- ATT&CK (Attacker tactics)
  - https://attack.mitre.org/
- CAPEC (Attack patterns)
  - https://capec.mitre.org/

Agenda

- Introduction
- Security properties
- OWASP Top-10 security risks (Web Applications)
- Correct approach to security of web applications
  - Walk through an example of Threat Modeling

Intro – Web security

WORLD NEWS JULY 16, 2019 / 10:04 AM / 3 MONTHS AGO

In systemic breach, hackers steal millions of Bulgarians' financial data

PRIVACY AND SECURITY

885 Million Records Exposed Online: Bank Transactions, Social Security Numbers, and More

Entity	Year	Records	Organization type	Method
Evide data breach	2023	1,000	computer services for charities	ransomware hacked
Philippine law enforcement agencies (Philippine National Police, National Bureau of Investigation, Bureau of Internal Revenue)	2023	1,279,437	government	poor security
Consumer Financial Protection Bureau	2023	256,000	bureau	poor security
Directorate General of Population and Civil Registration (Dukcapil)	2023	337.225.463	Government	leaked and published
Tesla	2023	75,000	transport	inside job
Directorate General of Immigration of Indonesia	2023	34,900,867	Government	hacked and published
Duolingo	2023	2,676,696	educational services	hacked

October 11

Air Europa Data Breach: Spanish airline carrier Air Europa has told their customers to cancel all of their credit cards after hackers managed to access their financial information during a

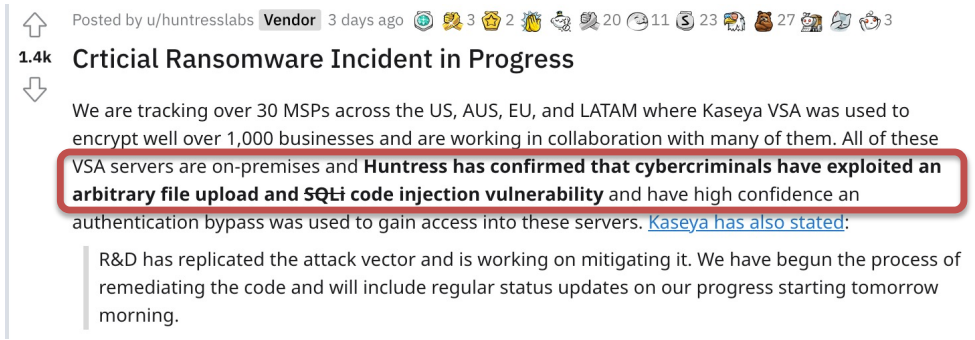
https://tech.co/news/data-breaches-updated-list

Hundreds of millions of Facebook user records were exposed on Amazon cloud server

https://en.wikipedia.org/wiki/List_of_data_breaches

Intro – Web security

- Weaknesses
- Vulnerabilities
- Exploits



Posted by u/huntresslabs **Vendor** 3 days ago 3 2 20 11 23 27 3

1.4k

Crticial Ransomware Incident in Progress

We are tracking over 30 MSPs across the US, AUS, EU, and LATAM where Kaseya VSA was used to encrypt well over 1,000 businesses and are working in collaboration with many of them. All of these VSA servers are on-premises and **Huntress has confirmed that cybercriminals have exploited an arbitrary file upload and ~~SQLi~~ code injection vulnerability** and have high confidence an authentication bypass was used to gain access into these servers. Kaseya has also stated:

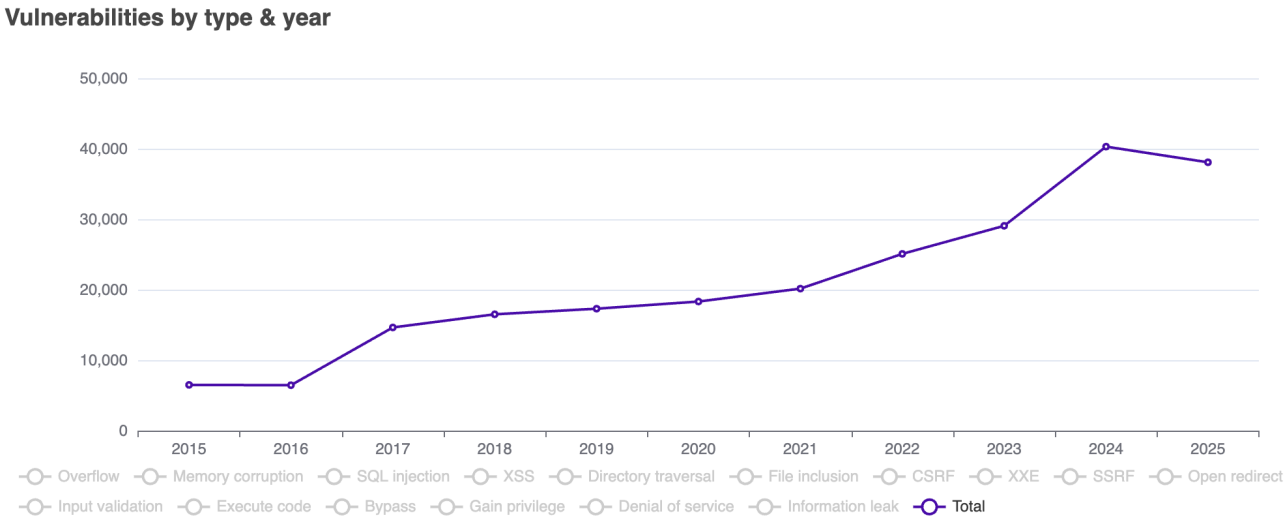
> R&D has replicated the attack vector and is working on mitigating it. We have begun the process of remediating the code and will include regular status updates on our progress starting tomorrow morning.

the attack was triggered via an authentication bypass vulnerability in the Kaseya VSA web interface.

Intro – Web security

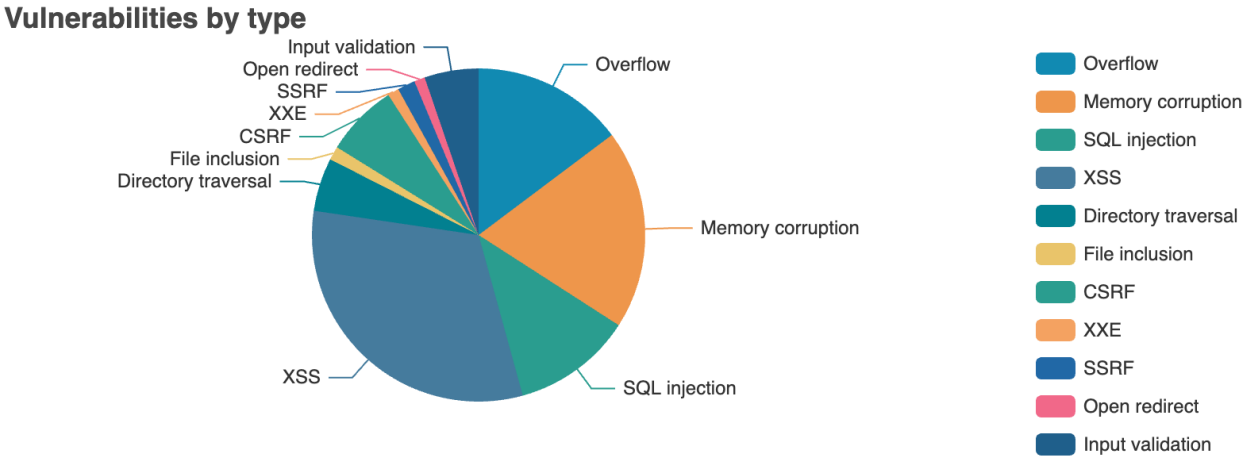
- Software weaknesses are errors in software implementation, code, design, or architecture that if left unaddressed could result in systems and networks being vulnerable to attack (CWE)
  - Examples include buffer overflows, format strings, etc., structure and validity problems, code evaluation and injection, authentication errors, resource management errors, insufficient verification of data, etc.
- A software vulnerability, such as those enumerated on the Common Vulnerabilities and Exposures (CVE) List, is a mistake/weakness in software that can be directly used by an attacker to gain access to a system or network
- An Exploit is a piece of software containing attack vectors that could be directly used to take advantage of a vulnerability in a system (see Metasploit)

Intro – Web security



Statistics

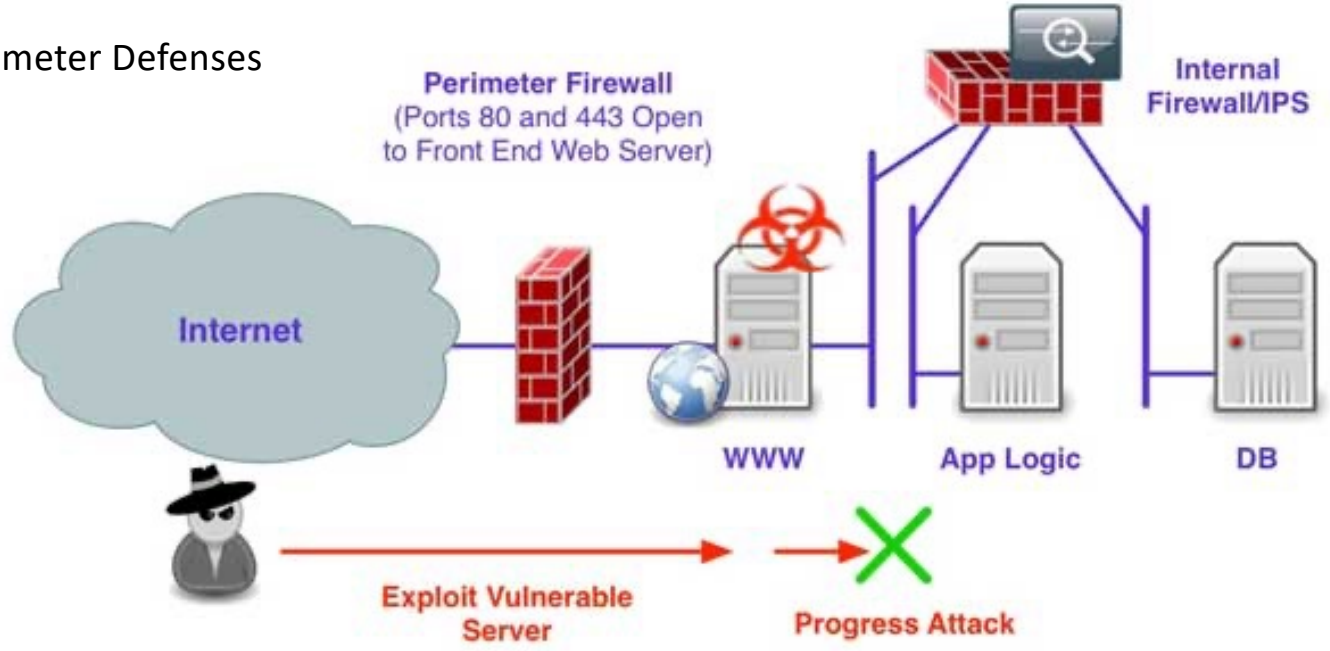
Source: CVE Details



Intro – Web security

Why web application is a game changer?

Network Perimeter Defenses



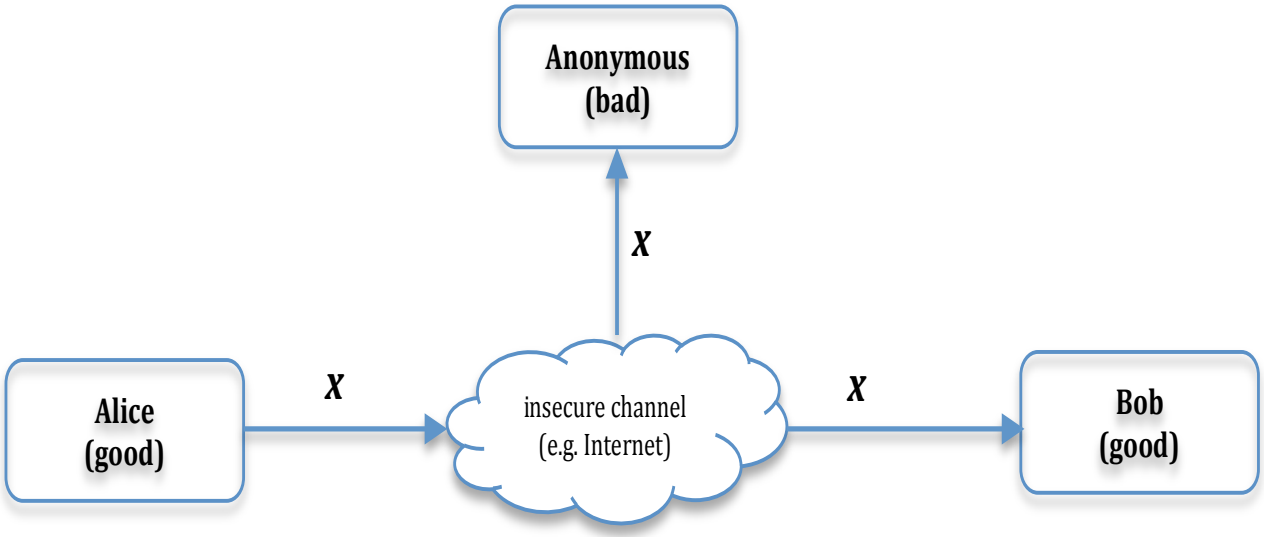
We have a secure network firewall?

But, usual ports (80, 443) have to be open for web applications – We cannot filter them
At best, we can use Web Application Firewall (WAF) to detect anomalies in web packets

Security Properties (What we want)

- Security properties (How to build in security)
  - Confidentiality
  - Integrity
  - Availability
  - Authentication
  - Authorization
  - Auditing/Non-repudiation
- Security Threats
  - STRIDE (will come back to this later)
- Mitigations/Countermeasures

CONFIDENTIALITY



Communication over an insecure channel

Confidentiality: dictates that data and information are not disclosed to unauthorized entity (human, process/system)
Threat: Information Disclosure

INTEGRITY

- The principle that information should be protected from intentional, unauthorized or accidental modification
  - Transit
  - Processing
  - Rest
- **Threat**: Tampering

AVAILABILITY

- The principle that data/service is available and accessible to authorized users when needed
- **Threat**: Denial of Service (DOS)

AUTHENTICATION

- The property that deals with identification and authorized access to a system
- **Threat**: Spoofing

AUTHORIZATION

- The principle that concerns access and control to system resources and operations
- **Threat**: Elevation of Privilege

NON-REPUDIATION

- The principle that actions cannot be denied by the performer
- **Threat**: Repudiation

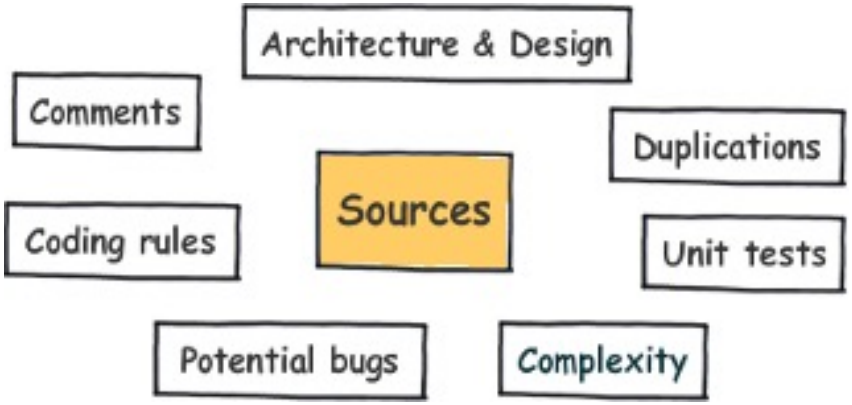
Security functionality vs. Secure Software

- **Security functionalities != secure software**
  - Authentication does not by default = implementation of strong authentication mechanism
  - Authorization does not by default = complete mediation or avoiding security by obscurity or broken access control

HTTPS/TLS vs. Secure Software

- TLS only protects data in-transit
- TLS does not prevent unsafe and unfiltered input data
  - XSS
  - SQLi
  - CSRF
  - etc

Static Analysis vs. Secure Software



Warning!!!:

- Running a tool doesn't make you "secure"
- Static analysis may and may not be used for security audit
- An elegant code ≠ a "secure" code

OWASP Top Ten - 2021
https://owasp.org/www-project-top-ten/

OWASP Top 10 API Security Risks – 2023
https://owasp.org/API-Security/editions/2023/en/0x11-t10/

OWASP Top-10 Risks

Risk	Description
A01 Broken Access Control	Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.
A02 Cryptographic Failures	focus is on failures related to cryptography (or lack thereof). Which often lead to exposure of sensitive data.
A03 Injection	Injection of untrusted data to be processed by interpreters (SQL, NoSQL, OS command, LDAP, etc)
A04 Insecure Design	Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.”
A05 Security Misconfiguration	insecure default configurations -incomplete or ad hoc configurations
A06 Vulnerable and Outdated Components	libraries, frameworks, and other software modules, run with the same privileges as the application

https://owasp.org/Top10/

OWASP Top-10 Risks

Risk	Description
A07 Identification and Authentication Failures	Incorrect implementation of authentication and session management -compromise passwords, keys, or session tokens
A08 Software and Data Integrity Failures	relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs)
A09 Security Logging and Monitoring Failures	ineffective integration with incident response. Most breach studies show time to detect a breach is over 200 days. Detected by external parties
A10 Server Side Request Forgery (SSRF)	SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL

A01 Broken Access Control

- Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone.
- Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.
- Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)
- Accessing API with missing access controls for POST, PUT and DELETE.
- Elevation of privilege. Acting as a user without being logged in or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, or a cookie or hidden field manipulated to elevate privileges or abusing JWT invalidation.
- CORS misconfiguration allows API access from unauthorized/untrusted origins.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

A01 Broken Access Control - Example

- CORS misconfiguration

http://localhost:3001/

```
app.use((req, res, next) => {
    res.setHeader('Access-Control-Allow-Origin', 'http://localhost:5173’)
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE’)
    res.setHeader('Access-Control-Allow-Headers', 'Content-Type’)
    next()
})
```

```
app.use((req, res, next) => {
    res.setHeader('Access-Control-Allow-Origin’,’*’)
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE’)
    res.setHeader('Access-Control-Allow-Headers', 'Content-Type’)
    next()
})
```

A01 Broken Access Control - Example

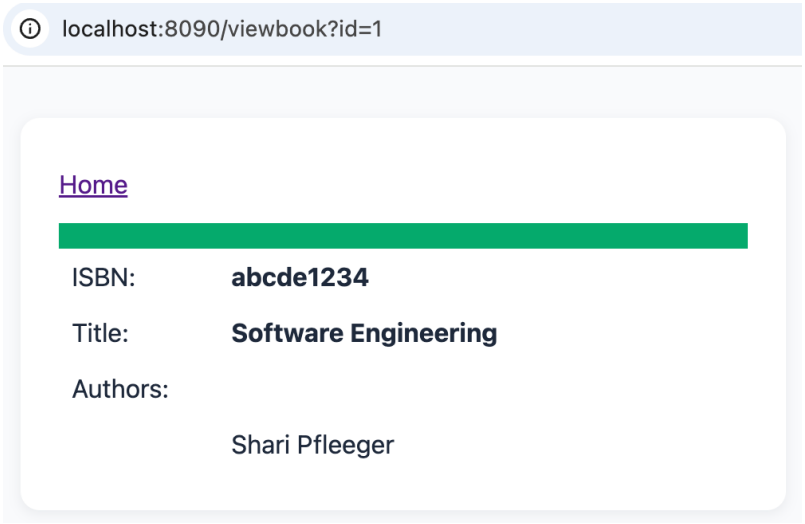
- Accessing API with missing access controls for POST, PUT and DELETE & GET

```
// Missing access controls
@PutMapping("/users/{id}")
public ResponseEntity<Object> updateUser(@RequestBody User user,
    @PathVariable Long id) throws UserNotFoundException{
    user = userService.updateUser(user, id);
    return new ResponseEntity<>(user, HttpStatus.OK);
}
```

A01 Broken Access Control - Example

- Bypassing access control checks by modifying the URL (parameter tampering or force browsing),

/viewbook?id=1 -> unauthorize access through parameter tampering
/viewbook?id=2



/deletebook?id=1 **?**

A01 Broken Access Control - Mitigations

- Except for public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g., repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- Stateful session identifiers should be invalidated on the server after logout. Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized. For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.

OWASP Top 10 **API** Security Risks

- API1:2023 Broken Object Level Authorization
  - /{product}/{id} -> unauthorize access to different objects (product or id)??
- API2:2023 Broken Authentication
  - e.g., Accepts unsigned/weakly signed JWT tokens ({"alg":"none"})
- API3:2023 Broken Object Property Level Authorization
  - Internal object property {"customer":"1", "price":"20000", "paid":"true"}. Unauthorize access to internal properties? Can be read, manipulated?

https://owasp.org/API-Security/editions/2023/en/0x00-header/

https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

API1: Broken Object Level Authorization

- API1:2023 BOLA - Example

```
// user 3 token trying to access user 1 orders
Response response = RestAssured.given()
    .header("Authorization", "Bearer "+ USER3_TOKEN)
    .get(API_ROOT+"/users/1/orders");
```

GET /api/v1/users/3/orders HTTP/1.1 **Authorization**: Bearer yJhbGciOiJSUzI1N	**GET** /api/v1/users/1/orders HTTP/1.1 **Authorization**: Bearer yJhbGciOiJSUzI1N

{
"id": 1,
"isbn": "qabfde1230",
"expiry": "2025-11-08",
}
	{
	"id": 2,
	"isbn": "ghijk1234",
	"expiry": "2025-11-01",
	}

API3: Broken Object Property Level Authorization

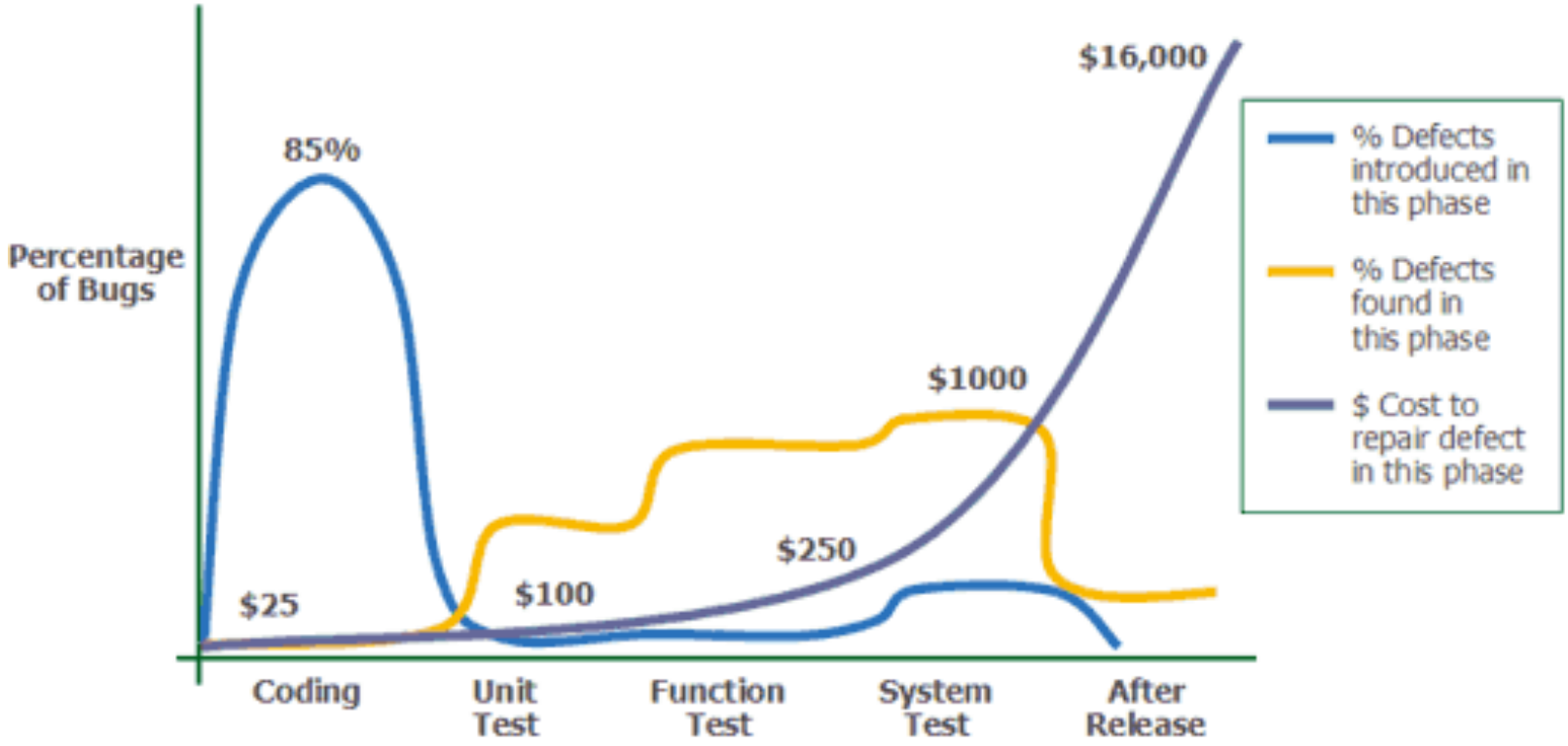
- API3:2023 BOPLA - Example

Normal request	Hacked request
POST /api/v1/users/1/orders HTTP/1.1 **Authorization**: Bearer yJhbGciOiJSUzI1N	**POST** /api/v1/users/1/orders HTTP/1.1 **Authorization**: Bearer yJhbGciOiJSUzI1N

Request
{
"id": 1,
"isbn": "qabfde1230",
"price": "1000",
}
	Request
	{
	"id": 1,
	"isbn": "qabfde1230",
	"price": "1000",
	"**paid": "true**"
	}

Correct Approach to Security

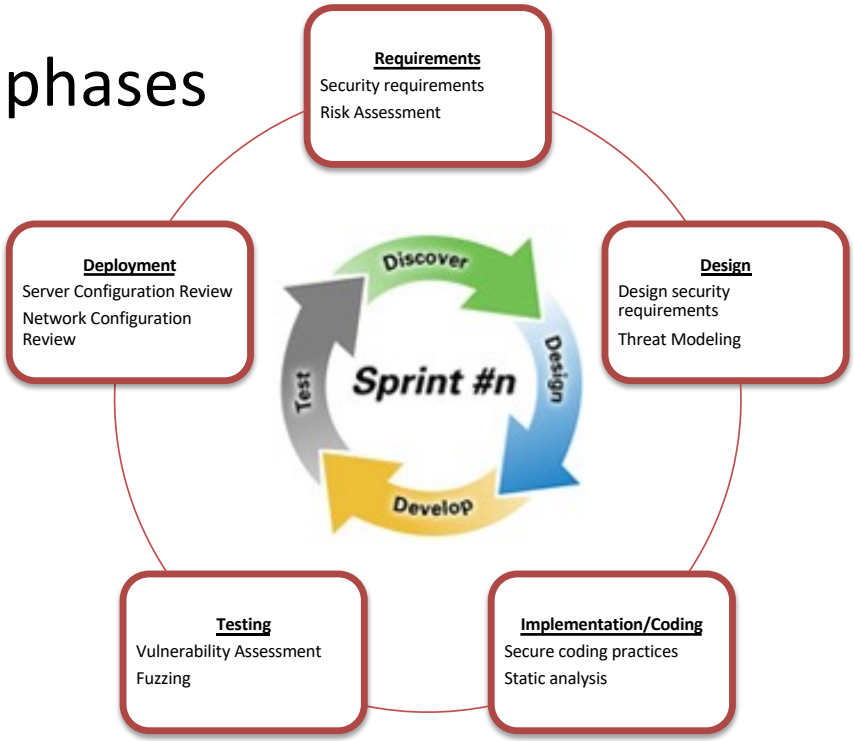
The cost of fixing a bug in Production is 100x bigger than during coding/design



Source: Applied Software Measurement, Capers Jones, 1996

Building-in security

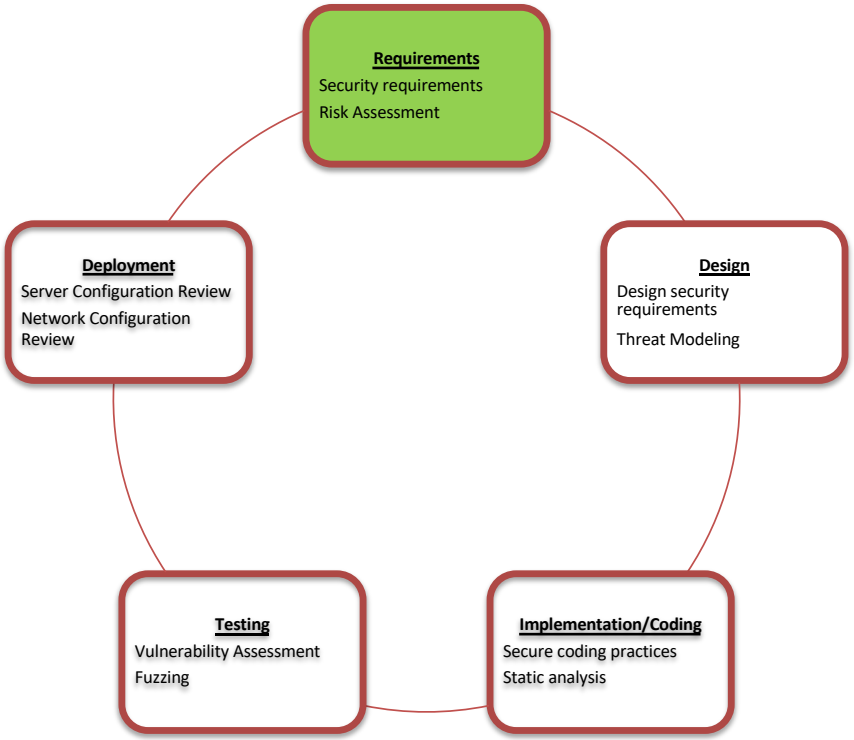
- Building any software involves different phases
  - Requirement
  - Design
  - Implementation
  - Testing
  - Deployment
  - Maintenance



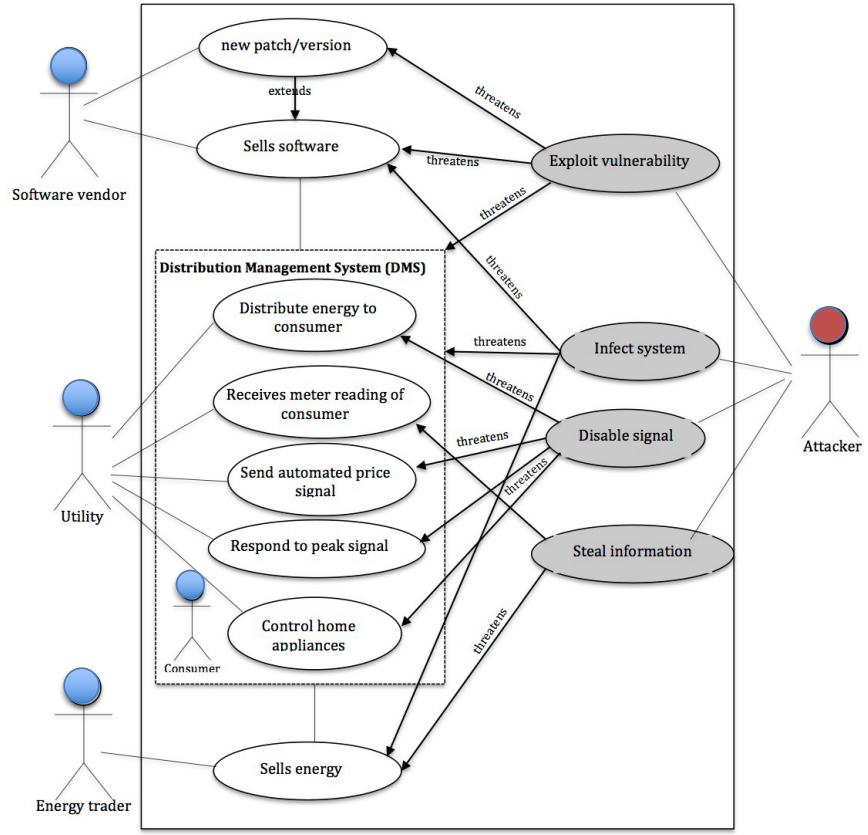
- Security is not an add-on, it must be built in
- It relates to all aspects and phases of software development lifecycle
- Thereby leading to a Secure Software Development Lifecycle (SSDLC)

Security in Requirement/Features

- Security requirements
  - Misuse case
  - Abuse stories
  - Protection Poker Game



Misuse case



Capturing security requirements using misuse case model

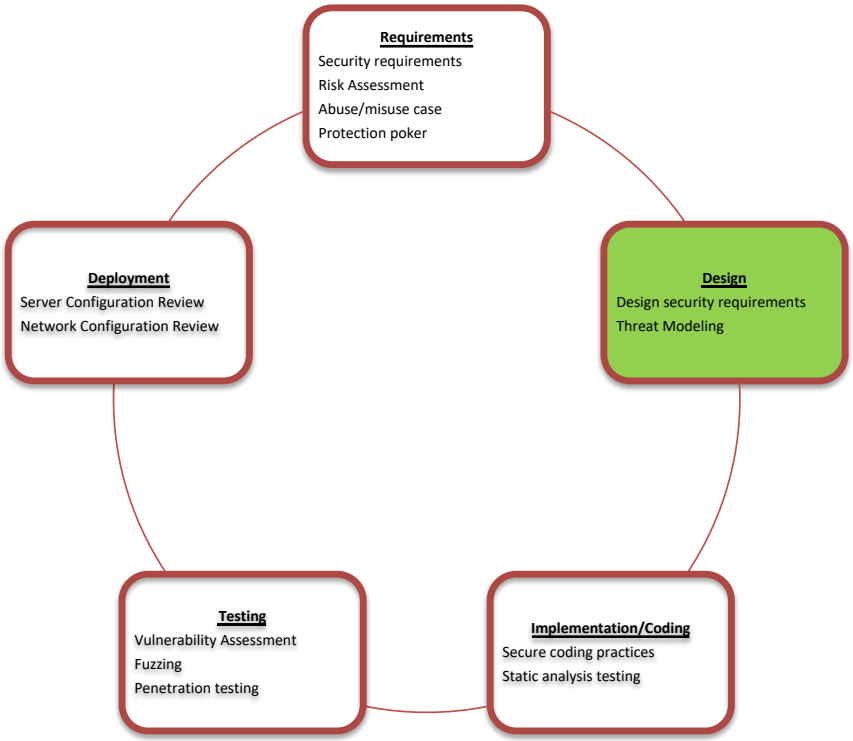
A simple misuse cases for energy sales and distribution systems

Abuser/Evil user stories

- “hacking the backlog” by crafting Evil User Stories,
  - negative cases that team needs to consider when they implement other stories.
  - Example #1. "As an attacker, I can send bad data in URLs, so I can access data and functions for which I'm not authorized."
  - Example #2. "As an attacker, 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 an attacker, I can send bad data in HTTP headers, so I can access data and functions for which I'm not authorized."
  - Example #4. "As an attacker, I can read and even modify all data that is input and output by your application."

Security in Design

- We'll look more into threat modelling using STRIDE



Shostack, A. (2014). *Threat modeling: Designing for security*. John Wiley & Sons.

Methodology – STRIDE (Software-centric)

Model system

Find threats

S Spoofing
T Tampering
R Repudiation
I Information Disclosure
D Denial of Service
E Elevation of Privilege

Mitigate threats

Mitigation toolbox

Encryption
Hashing
Signature
Least privilege
Sandboxing
Validation
ACLs
Resource Quota
Firewall
Logging

Evaluate & repeat the process

Methodology

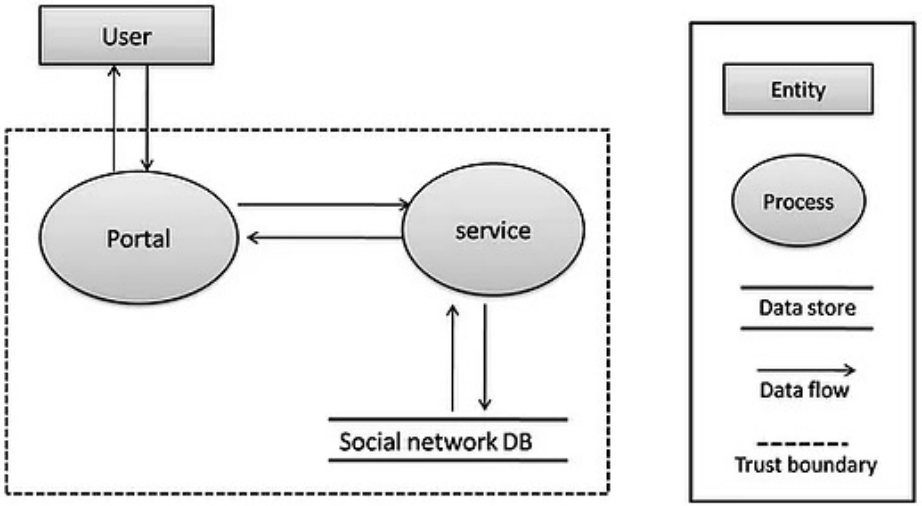
	Type of Threat	What Was Violated
S	Spoofing	Authentication
T	Tampering	Integrity
R	Repudiation	Non-repudiation
I	Information Disclosure	Confidentiality
D	Denial of Service (DoS)	Availability
E	Elevation of Privilege	Authorization

DFD element	S	T	R	I	D	E
External Entity	X		X			
Process	X	X	X	X	X	X
Data Flow		X		X	X	
Data Store		X	?	X	X	

Threats
Mitigations

Data flow diagram (DFD)

What are we working on?



The data flow diagram (DFD) of a simple Social Network application

An **External entity** represents an entity that exists outside the system being modeled and which interacts with the system at an entry point

A **Process** represents a task in the system that processes data or performs some action based on the data.

A **DataStore** represents a repository where data is saved or retrieved, but not changed. Examples of data stores include a database, a file, or the Registry — a database of configuration settings in Windows operating systems.

DataFlow, represents data transferred between elements

Documenting & Mitigation

Documenting Threats

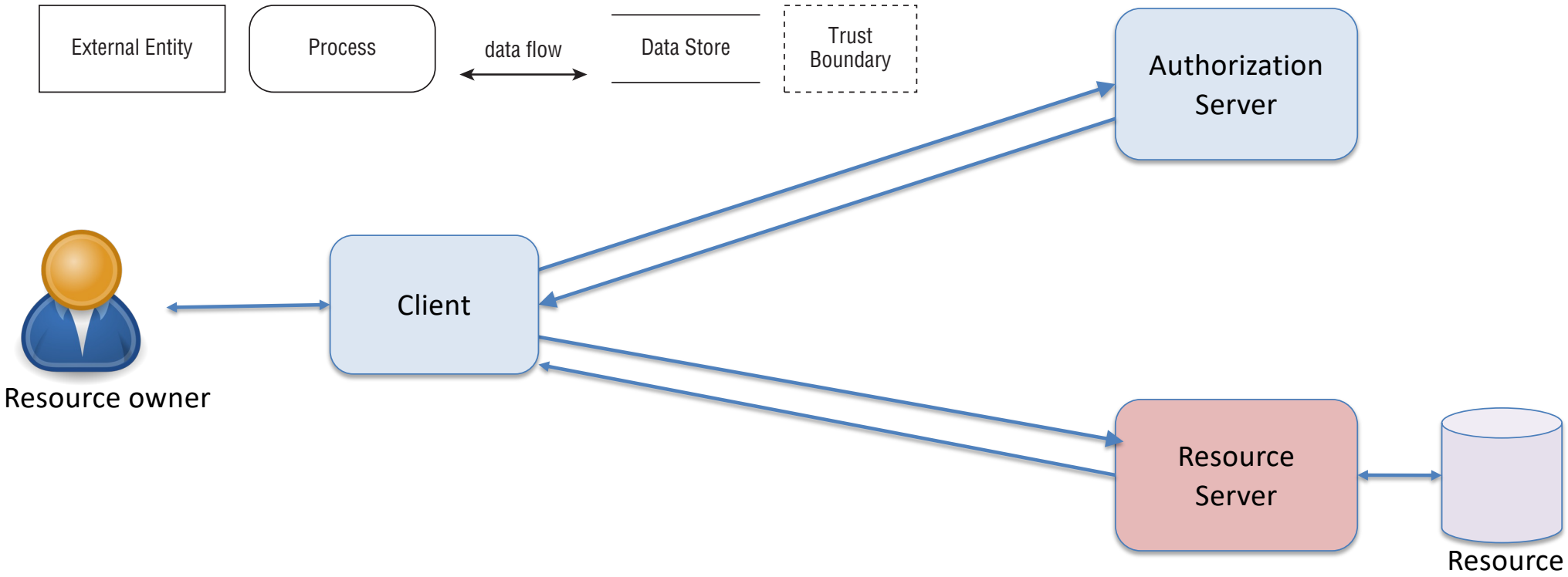
Threat Type	Diagram Element(s)	Threat	Bug ID
Tampering	Web browser	Attacker modifies our JavaScript order checking	4556 “Add order-checking logic to server”
	Data flow #2 from browser to server	Failure to authenticate	4557 “~~Add~~ enforce HTTPS everywhere”

What are we going to do about it?

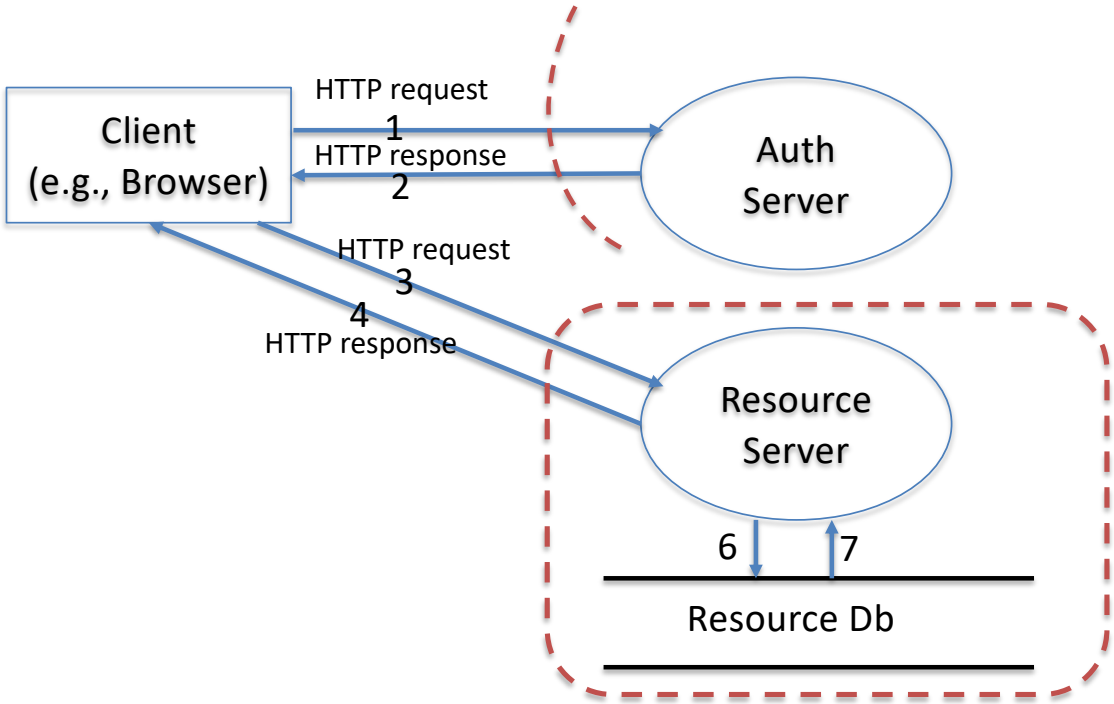
Threat	Mitigation Technology	Developer Example	Sysadmin Example
Spoofing	Authentication	Digital signatures, Active directory, LDAP	Passwords, crypto tunnels
Tampering	Integrity, permissions	Digital signatures	ACLs/permissions, crypto tunnels
Repudiation	Fraud prevention, logging, signatures	Customer history risk management	Logging
Information disclosure	Permissions, encryption	Permissions (local), PGP, SSL	Crypto tunnels
Denial of service	Availability	Elastic cloud design	Load balancers, more capacity
Elevation of privilege	Authorization, isolation	Roles, privileges, input validation for purpose,	Sandboxes, firewalls

An example with our REST API system

- We'll now make a STRIDE DFD model from this REST API system ☺

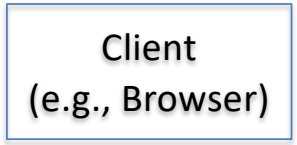


Create a DFD


Client
(e.g., Browser)
HTTP request
1
HTTP response
2
Auth
Server
HTTP request
3
4
HTTP response
Resource
Server
6
7
Resource Db


External Entity
Process
data flow
Data Store
Trust
Boundary

Find Threats



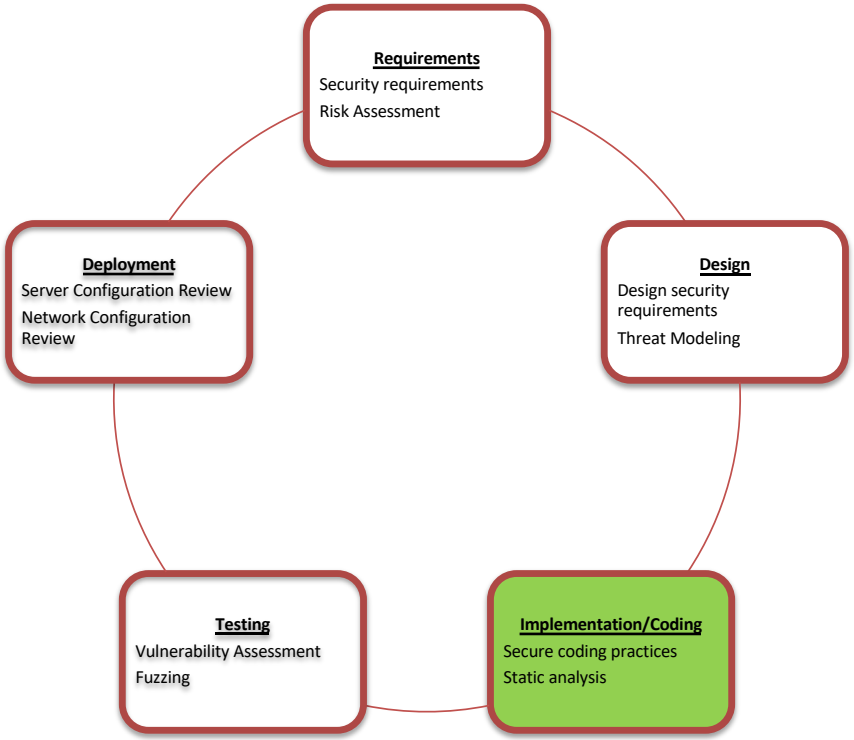
- Spoofing
- Repudiation

Can use different repo and gamification during this stage
e.g., EoP game, OWASP Cornucopia, and variants

https://eopgame.azurewebsites.net/

Threat Type	Diagram element	Threat	Bug ID
Spoofing	Client	Weak password can be brute-forced thereby leading to spoofed identity	#1 Enforce strong password policy
Repudiation	Client	Lack of logging can lead to client denying action	#2 Implement logging function for client requests
…	…	…	…

Security in Implementation (Secure Coding)



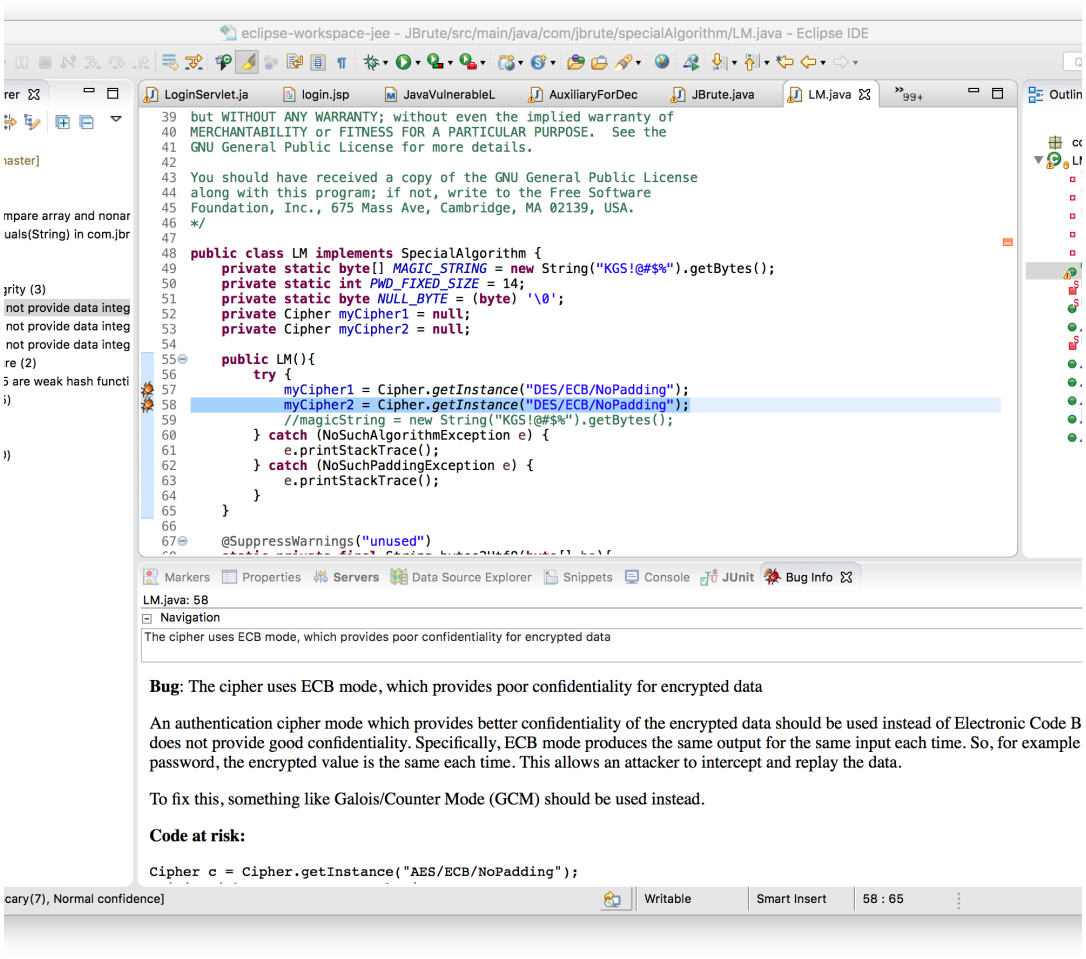
Secure Coding

- Input validation
- Output encoding
- Authentication/password management
- Session management
- Error/Exception handling
- Logging/Auditing
- Data Protection
- File Management
- Cryptography
- Configuration Management
- Concurrency
- Memory Management
- Safe APIs
- Type safety

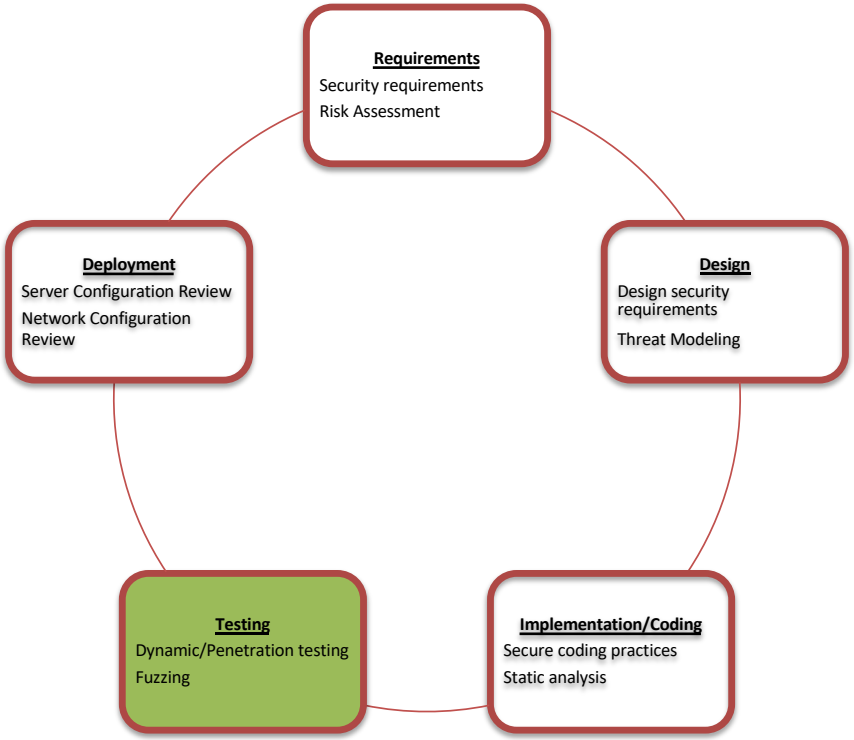
Static Application Security Testing Tools

Static Application Security Tools (SAST)

- FindSecBugs
  - IDE plugin
  - Requires SpotBug
  - Add FindSecBugs plugins
- LLM
  - Many options (Llama, DeepSeekCoder, etc)



Security Testing (Pre-Release)



Security Testing

- How does your web application perform when in production environment?
- Are there weaknesses that can be exploited by attackers?
- Some bugs can only be caught when the system is tested as a whole and in a different execution environment other than the developer's machine
  - **Penetration testing**: subject the web application to a rigorous security attack to find exploitable security bugs
    - **Fuzzing**

We'll come back to tools for performing penetration testing

Next lecture

- ***Lab/Exercise 1***: Practice on WebGoat – Setup / Create an account on PortSwigger
  - Get the WebGoat running on your machine.
  - Introduction to the labs and tools
- **Lecture 2**: Injection Attacks (SQL Injection, …)