Internet Applications Design and Implementation

(Lecture 6 - on Security)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt),
contributions from Beatriz Moreira, APDC-INV 2017/2018)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Software Security (Overview)

- Security is a hot topic in CS in present days!
- **Cryptography** can only protect data **outside of systems**, when properly applied and if there are no “internal” information leaks.
- **Security breaches** are often caused by internal errors that cause: system crashes or erroneous behaviour due to unexpected inputs
- **Security breaches** are often caused by programming mistakes.

Security of Applications (Overview)

- Good engineering practices and correct usage of methods and tools ensures that all specified data confidentiality rules are properly enforced.
- There are two elective courses in MIEI only about this topic.
  - Software Security: fundamental concepts and technologies
  - Network and Computer Systems Security: system-level security
- This lecture is about models and frameworks that implement software security.

Outline

- Security Concepts Review
- Foundations of Computer Security
- Security Models
- Kotlin and Spring Security
- Model-Based Access Control in Spring

Internet Applications Design and Implementation

(Lecture 6 - Part 1 - Security Concepts)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Security of Internet Applications

- Layers of internet application security
  - Network Level (network security and system identification)
  - System Level (firewalls, VPNs, SSL, DMZ)
  - Application Level (our focus)
    - Authentication
    - Access control
    - Information flow

The Big Picture - Organisation Industry Standards (ISO 27002)

- Definition of General Security policies
- Organization of information security
- Asset management
- Human resources security
- Physical and environmental security
- Communications and operations management
- Access control (data, operations, and other resources)
- Acquisition, development, and maintenance of Software Systems
- Incident management
- Compliance with regulations

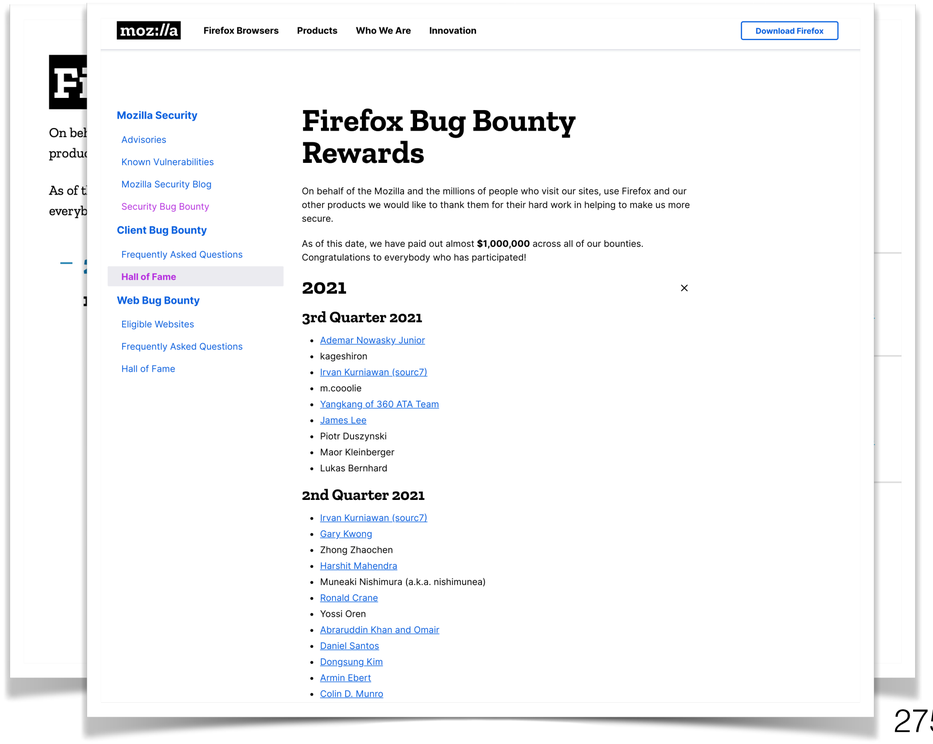
About software

Security Policies in Software Systems

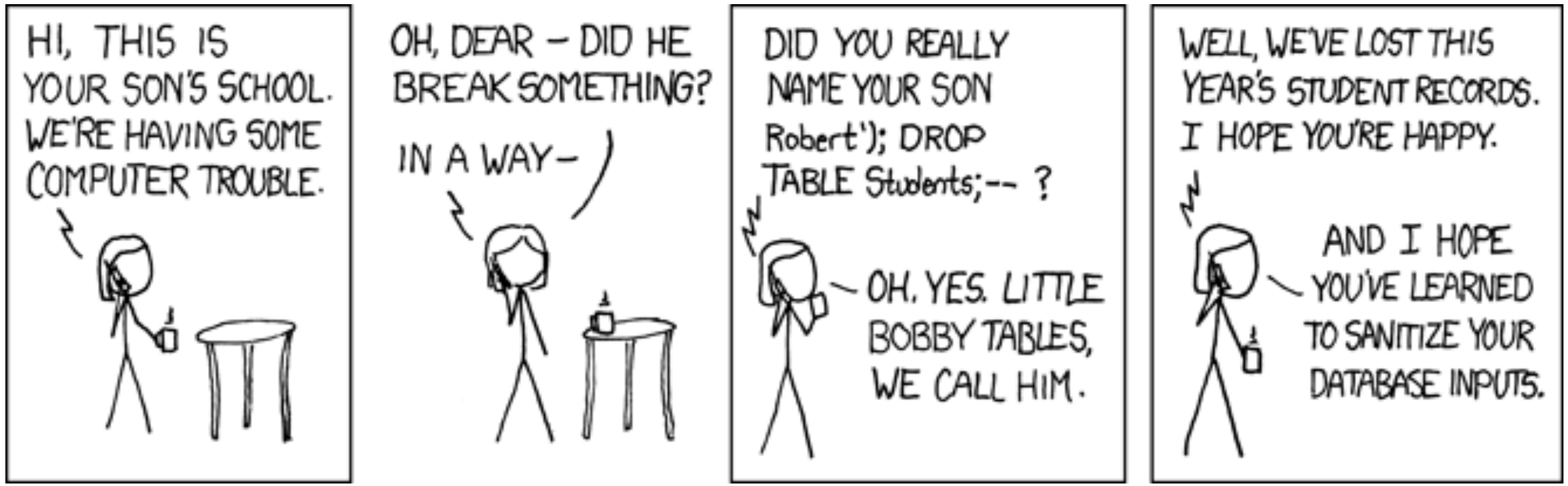
- within an software system:
  - “The set of restrictions and properties that specify how a computing system prevents information and computing resources from being used to violate an organizational security policy” in Computer Security, Dieter Gollmann 2011
- Policies define who has permission to access or operate on, a given resource.
  - Access control lists,
  - firewall settings,
  - services that may be run on user devices,
  - security protocols for protecting network traffic,
  - …

Attacks and Attackers

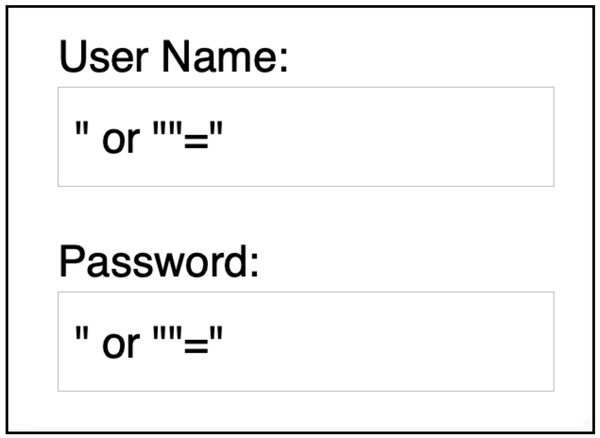
- Security can be defined and challenged based on the definition of attacks and attackers. What does an attacker know and what operations can they perform on a system.
- The correct functioning of a system is based on assumptions on the environment.
- Attacks are designed to challenge system assumptions with illegal inputs, causing illegal states, erroneous behaviour or crashes.



Classic Example: SQL injection



Classic Example: SQL injection




```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ="'
+ uName + '" AND Pass ="' + uPass + '"'
```

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

https://www.w3schools.com/sql/sql_injection.asp

Security flaws in software

OWASP

OWASP Top 10 - 2017

The Ten Most Critical Web Application Security Risks

OWASP Top 10 - 2017

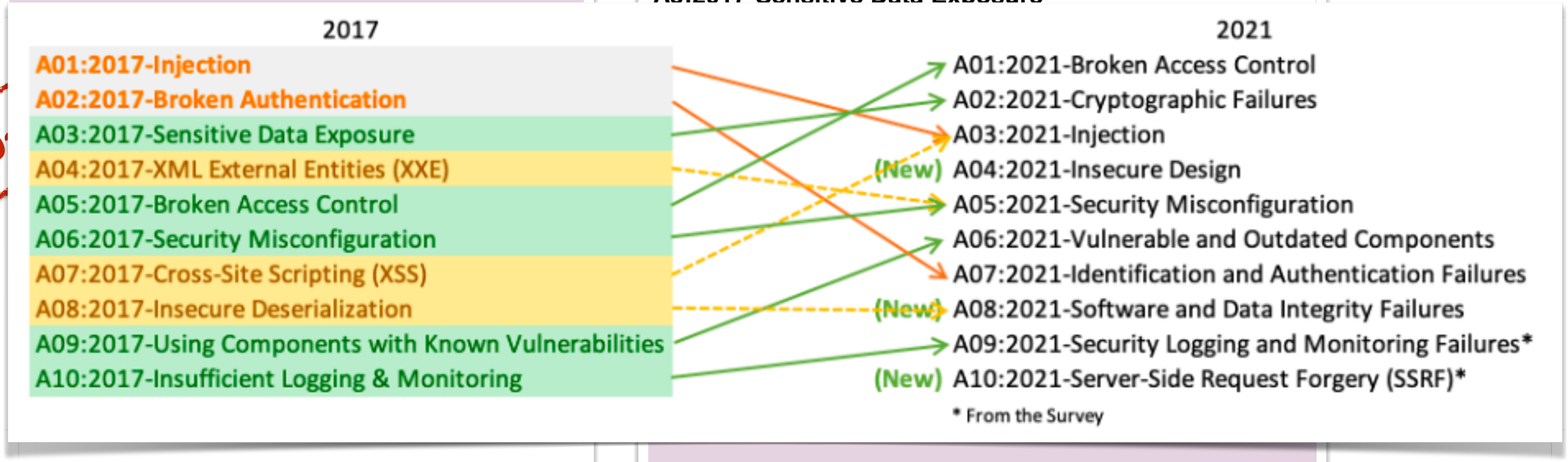
A1:2017-Injection

A2:2017-Broken Authentication

A3:2017-Sensitive Data Exposure

A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Still Up-to-D



https://www.owasp.org/

Security flaws in software

- **A01:2021-Broken Access Control** moves up from the fifth position; 94% of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category.
- **A02:2021-Cryptographic Failures** shifts up one position to #2, previously known as Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed focus here is on failures related to cryptography which often leads to sensitive data exposure or system compromise.
- **A03:2021-Injection** slides down to the third position. 94% of the applications were tested for some form of injection, and the 33 CWEs mapped into this category have the second most occurrences in applications. Cross-site Scripting is now part of this category in this edition.
- **A04:2021-Insecure Design** is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to “move left” as an industry, it calls for more use of threat modeling, secure design patterns and principles, and reference architectures.
- **A05:2021-Security Misconfiguration** moves up from #6 in the previous edition; 90% of applications were tested for some form of misconfiguration. With more shifts into highly configurable software, it’s not surprising to see this category move up. The former category for XML External Entities (XXE) is now part of this category.
- **A06:2021-Vulnerable and Outdated Components** was previously titled Using Components with Known Vulnerabilities and is #2 in the Top 10 community survey, but also had enough data to make the Top 10 via data analysis. This category moves up from #9 in 2017 and is a known issue that we struggle to test and assess risk. It is the only category not to have any Common Vulnerability and Exposures (CVEs) mapped to the included CWEs, so a default exploit and impact weights of 5.0 are factored into their scores.
- **A07:2021-Identification and Authentication Failures** was previously Broken Authentication and is sliding down from the second position, and now includes CWEs that are more related to identification failures. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks

Still Up-to-D

trol
ures
uration
tdated Components
Authentication Failures
Integrity Failures
nd Monitoring Failures*
t Forgery (SSRF)*
m.]
org/

In a nutshell

Most common attacks are at the application level and can be avoided by properly using well tested and tested frameworks and well established methods.

Internet Applications Design and Implementation

(Lecture 6 - Part 2 - Foundations)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Foundations of Computer Security

- Fundamental Properties:
  - **confidentiality** - prevention of unauthorised disclosure of information,
  - **integrity** - prevention of unauthorised modification of information,
  - **availability** - prevention of unauthorised withholding of information or resources
  - … more… accountability; non-repudiation; reliability/dependability; …

"Computer security deals with the prevention and detection of unauthorized actions by users of a computer system" - Dieter Gollmann,

See also the wikipedia page

Foundations of Software Security

- Main concepts used in software security
  - **Principal** (the active subject)
  - **Resourse** (the passive object)
  - **Authentication** - the principal provides proof that they are who they say they are
  - **Authorisation** - the principal is trusted to access or perform an operation on a resource
  - **Delegation** - can you operate on behalf of other principal?

- Declassification operations in confidentiality analysis
- Endorsement operations in integrity analysis

Principal/Authentication

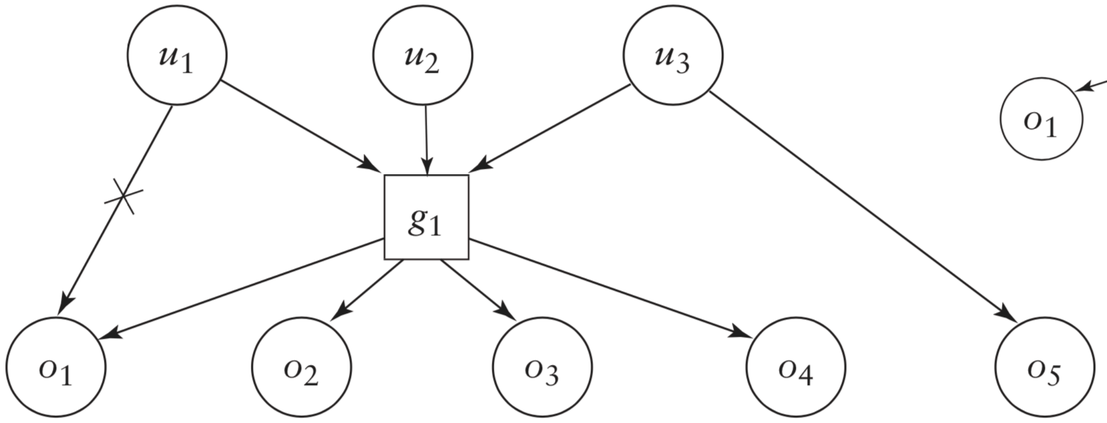
- An authenticated entity (or group) that uses the system
- The active subject on system operations
- **Authentication**:
  - The certification process of determining the identity of an external entity that uses the system, invokes actions, produces and consumes data.

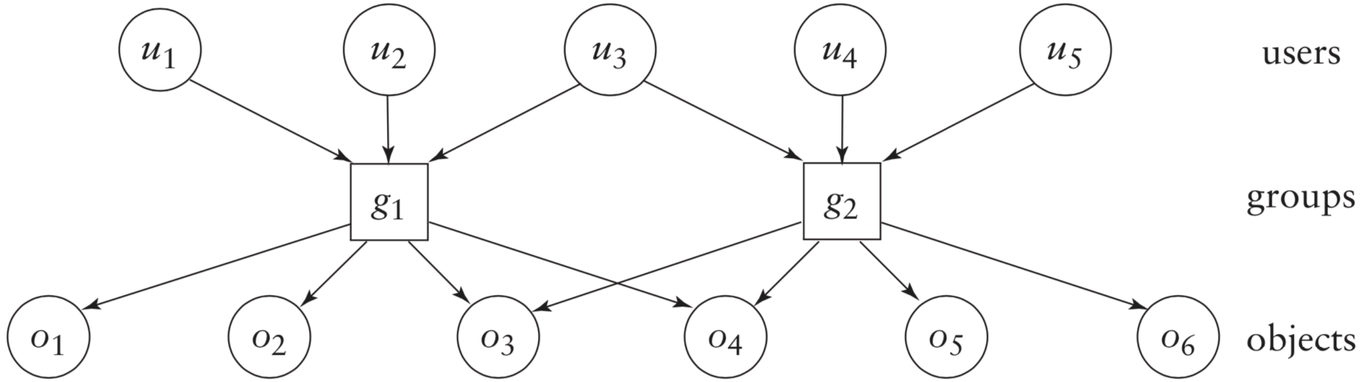
Resource/Authorisation

- The process of checking what resources a principal is allowed to access and manipulate, and what operations is it allowed to execute at a given time and system state (data, parameters, context, etc.).
- **Authorisation** (to observe or change)
  - read, execute, write, append, delete, change access, change ownership, …
  - Classic access control models
    - Access Control Matrix
    - Capabilities
    - Access Control Lists

Authorisation and access control models

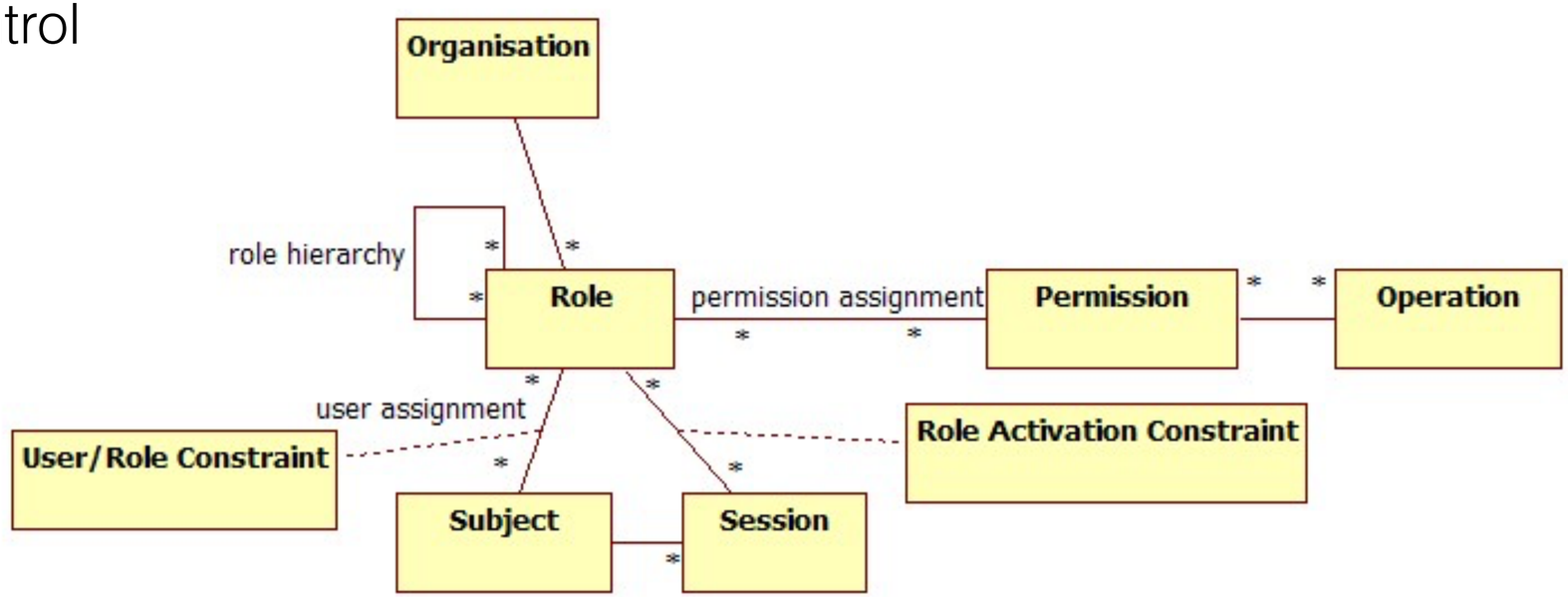
- Base models
  - Access Control Matrix; Capabilities; Access Control Lists
- Intermediate control structures
  - Groups and negative permissions





Authorisation and access control models

- Base models
  - Access Control Matrix; Capabilities; Access Control Lists
- Intermediate control structures
  - Role-based access control
    - Principal
    - Roles
    - Permissions
    - Operations
    - Resources



Internet Applications Design and Implementation

(Lecture 6 - Part 3 - Security Models)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

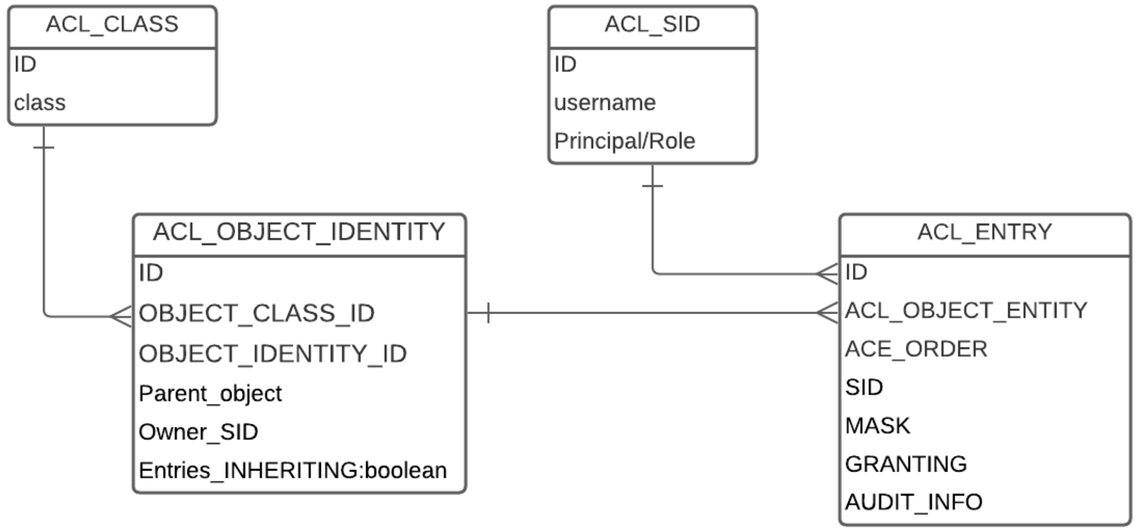
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Security Models

- Access Control List
- Capability-based access control
- Role-based access control
- Bell–LaPadula model for confidentiality
- Biba Integrity model for data integrity
- Model-based access control

Access Control Lists

- ACLs are lists of permissions (identity, operation) attached to objects stored within the system.
- It must explicitly assign individual identities to operations on resources.
- Spring Security ACL implements ACL on top of explicitly managed database tables




```
@PostFilter("hasPermission(filterObject, 'READ')")
List<NoticeMessage> findAll();

@PostAuthorize("hasPermission(returnObject, 'READ')")
NoticeMessage findById(Integer id);

@PreAuthorize("hasPermission(#noticeMessage, 'WRITE')")
NoticeMessage save(@Param("noticeMessage")NoticeMessage noticeMessage);
```

https://www.baeldung.com/spring-security-acl

Capability-based access control

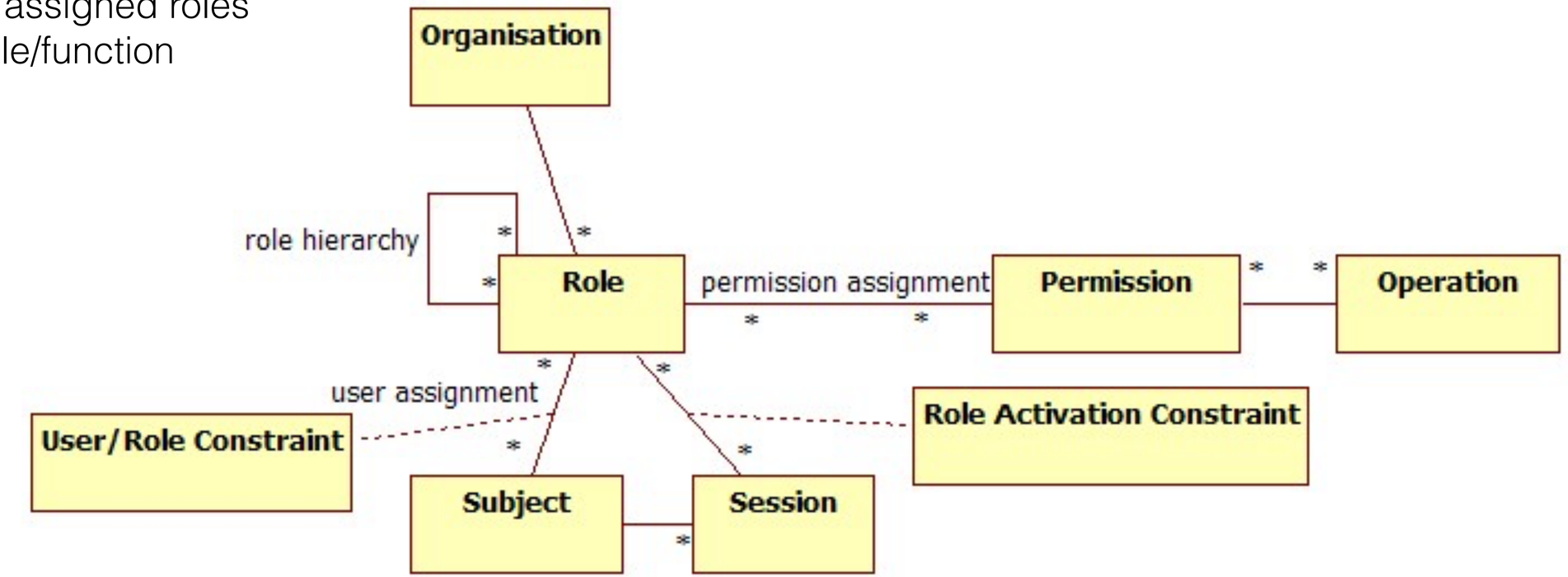
- A decentralised method that provides a scalable approach to access control
  - No need for centralised access control lists and mechanisms;
- A capability refers to an object and a set of rights, the user that owns it can perform the operation described in the capability
  - Example: a pair (“/etc/pass”, O_RDWR) gives readwrite access
- Common in micro-service based architectures and IoT scenarios

Capability-based access control

- A capability is transmitted through unforgeable (and ephemeral) tokens
- Capabilities can be explicitly and dynamically managed
  - Good support for authorisation and delegation
- To perform an operation on a particular object, the principal has first to acquire the associated capability via some dynamic authorisation method.

Role-Base Access Control

- Standard approach to authorise users to perform operations in software systems.
  - Roles are specified for system related tasks
  - Permissions are assigned to specific roles
  - Users are dynamically assigned roles according to their profile/function

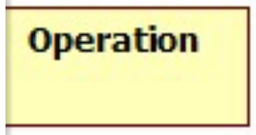


https://upload.wikimedia.org/wikipedia/en/c/c3/RBAC.jpg

Role-Base Access Control

- Standard approach to authorise users to perform operations in software systems.
  - Roles are specified for system related tasks
  - Permissions are assigned to specific roles
  - Users are dynam
    according to thei

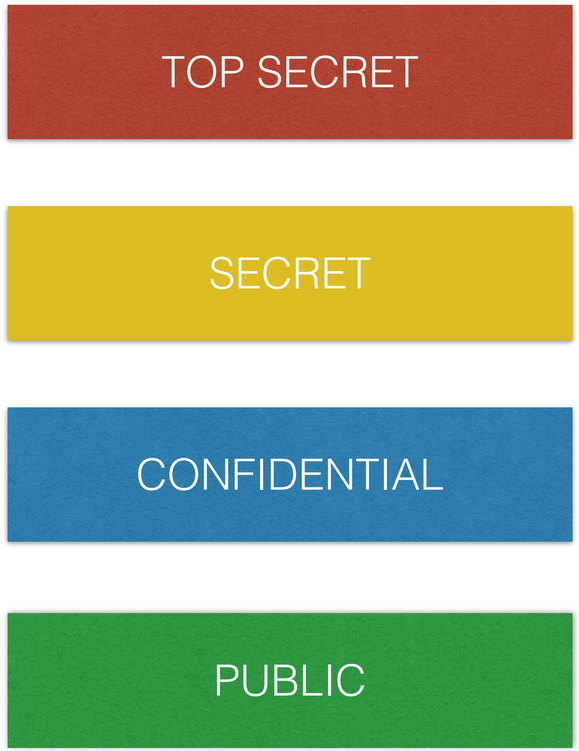
- **Roles can also be combined in hierarchies or lattices**
- **Delegation can be integrated in the general model**
- **RBAC provides a framework for a verification based on interceptors and filters**
- **Roles can also be parameterised and depend on actual data.**



https://upload.wikimedia.org/wikipedia/en/c/c3/RBAC.jpg

Bell–LaPadula model - Confidentiality

- Formalises the U.S. Department of Defense multilevel security policies.
- The security level of a subject is compared to the classification of the object (and the security compartment where it is stored)
- The access control rules state that:
  - A subject cannot read up
  - A subject cannot write down
  - + an access control matrix



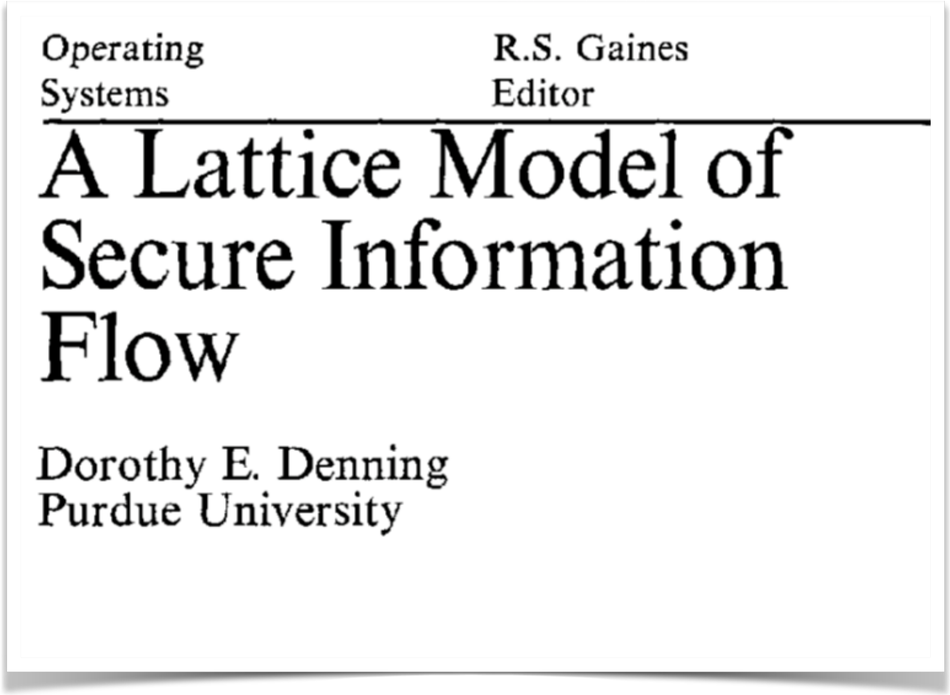
No Write Down

No Read Up

Information Flow Control - Confidentiality

- The process of ensuring that data is only seen by trusted principals. This involves assigning a level of trust to data and principals.
- General technique to check it is called **information flow control**
- **Declassification**: The process of lowering the level of trust needed to access a given data item.

```
var high = 1
var low = … 10 + high …

var low = 2
if ( high > 0 ) low else 3
```


Operating Systems

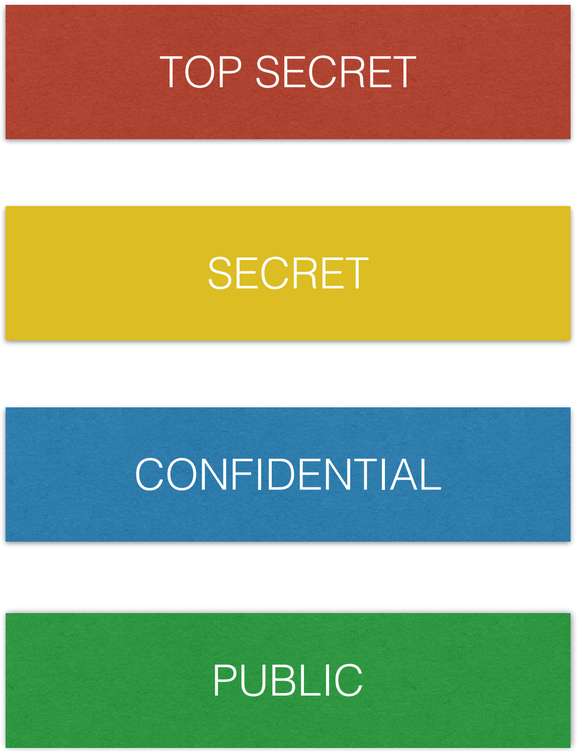
R.S. Gaines Editor

A Lattice Model of Secure Information Flow

Dorothy E. Denning
Purdue University

Biba Integrity model - Data Integrity

- Data integrity has three goals:
  - Prevent data modification by unauthorised principals
  - Prevent unauthorised data modification by authorised principals
  - Maintain internal and external data consistency
- The access control rules state that:
  - A subject cannot read up
  - A subject cannot write down
  - + an access control matrix



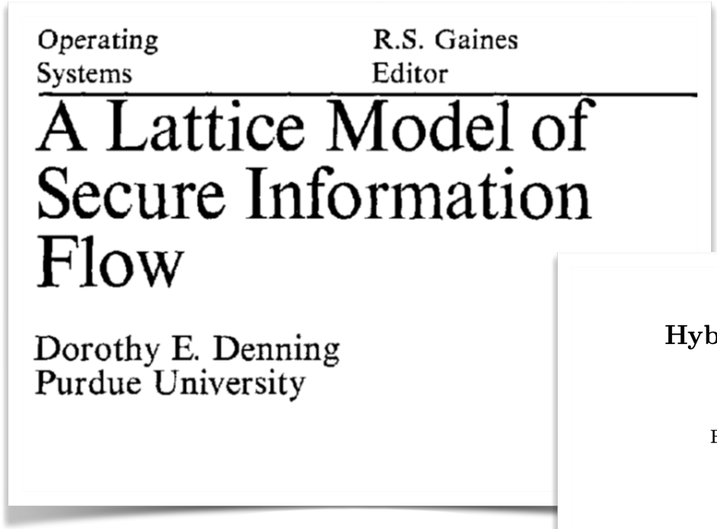
No Read Down

No Write Up

Information Flow Control - Data Integrity

- The process of ensuring that data is only provided by trusted principals.
- Also an information flow control property (dual of confidentiality)
- **Endorsement**: the process of increasing the level of trust o a given data item.

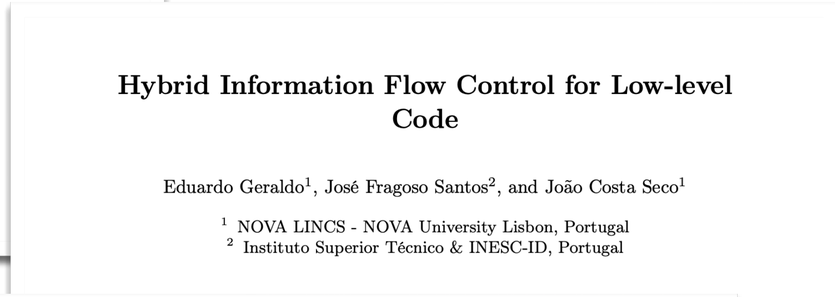
```
var untrusted = 1
var trusted = … 10 + untrusted …

var trusted = 2
if ( untrusted > 0 ) trusted else 3
```



Operating Systems
R.S. Gaines Editor

A Lattice Model of Secure Information Flow

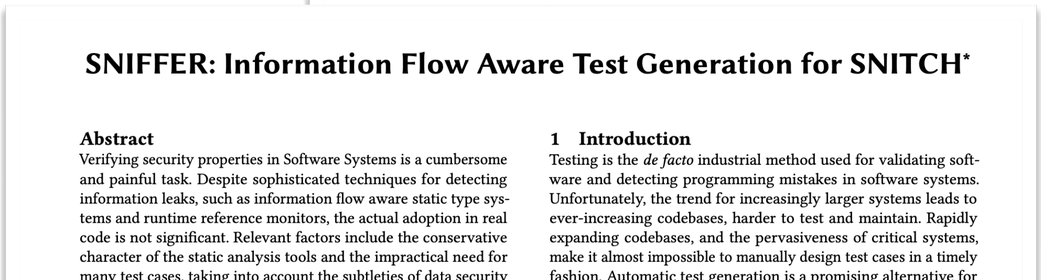
Dorothy E. Denning
Purdue University



Hybrid Information Flow Control for Low-level Code

Eduardo Geraldo[1], José Fragoso Santos[2], and João Costa Seco[1]

[1] NOVA LINCS - NOVA University Lisbon, Portugal
[2] Instituto Superior Técnico & INESC-ID, Portugal



SNIFFER: Information Flow Aware Test Generation for SNITCH*

Abstract

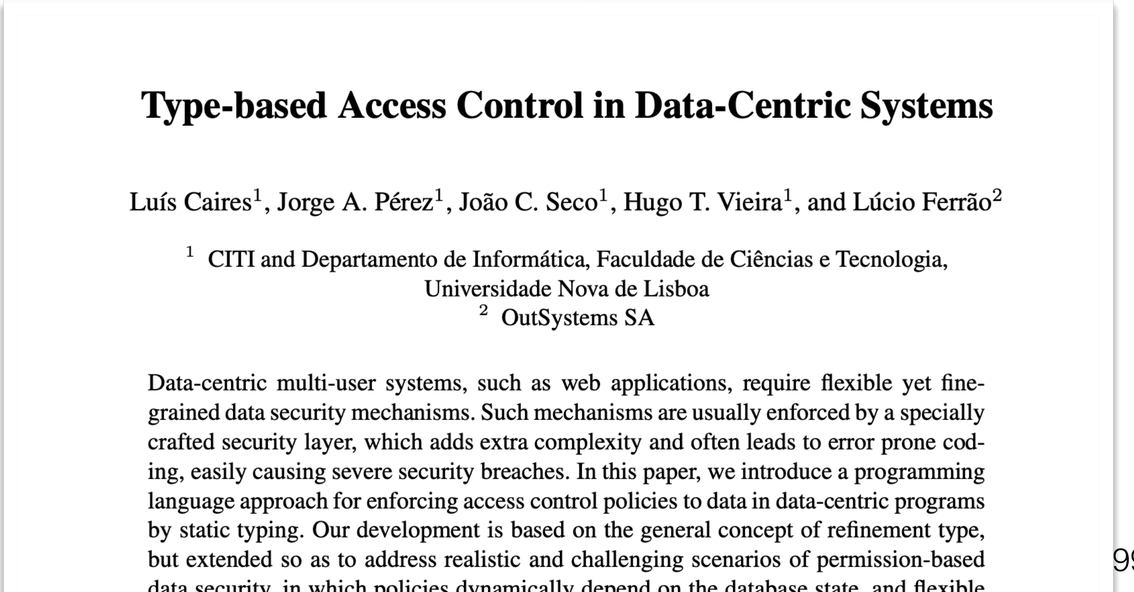
Verifying security properties in Software Systems is a cumbersome and painful task. Despite sophisticated techniques for detecting information leaks, such as information flow aware static type systems and runtime reference monitors, the actual adoption in real code is not significant. Relevant factors include the conservative character of the static analysis tools and the impractical need for

1 Introduction

Testing is the *de facto* industrial method used for validating software and detecting programming mistakes in software systems. Unfortunately, the trend for increasingly larger systems leads to ever-increasing codebases, harder to test and maintain. Rapidly expanding codebases, and the pervasiveness of critical systems, make it almost impossible to manually design test cases in a timely

Model-Based Access Control

- Security conditions are many times hidden in query filters and program conditions. These policies are very difficult to get right, maintain, and modify.
- Developer-defined roles usually depend on the current state of the application. (e.g. ModeratorOf(…)). Role-based models usually ignore this and require the application logic to validate it.
- Example: authorisations usually depend on the state of the target entity (status == “submitted”)
- Capabilities defined by the developer can extend the standard read/write/delete (not hardwired to the basic programming elements)



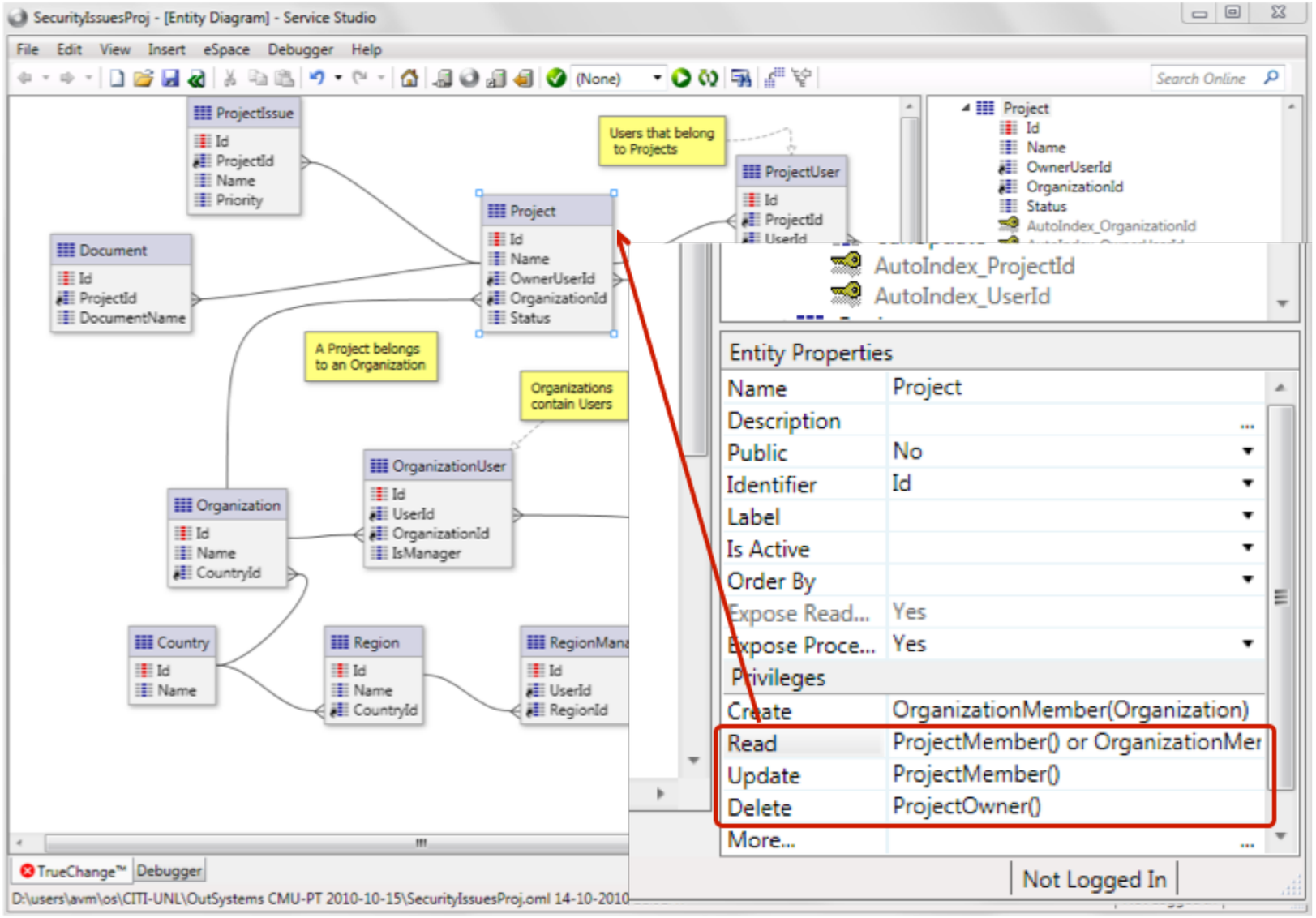
Type-based Access Control in Data-Centric Systems

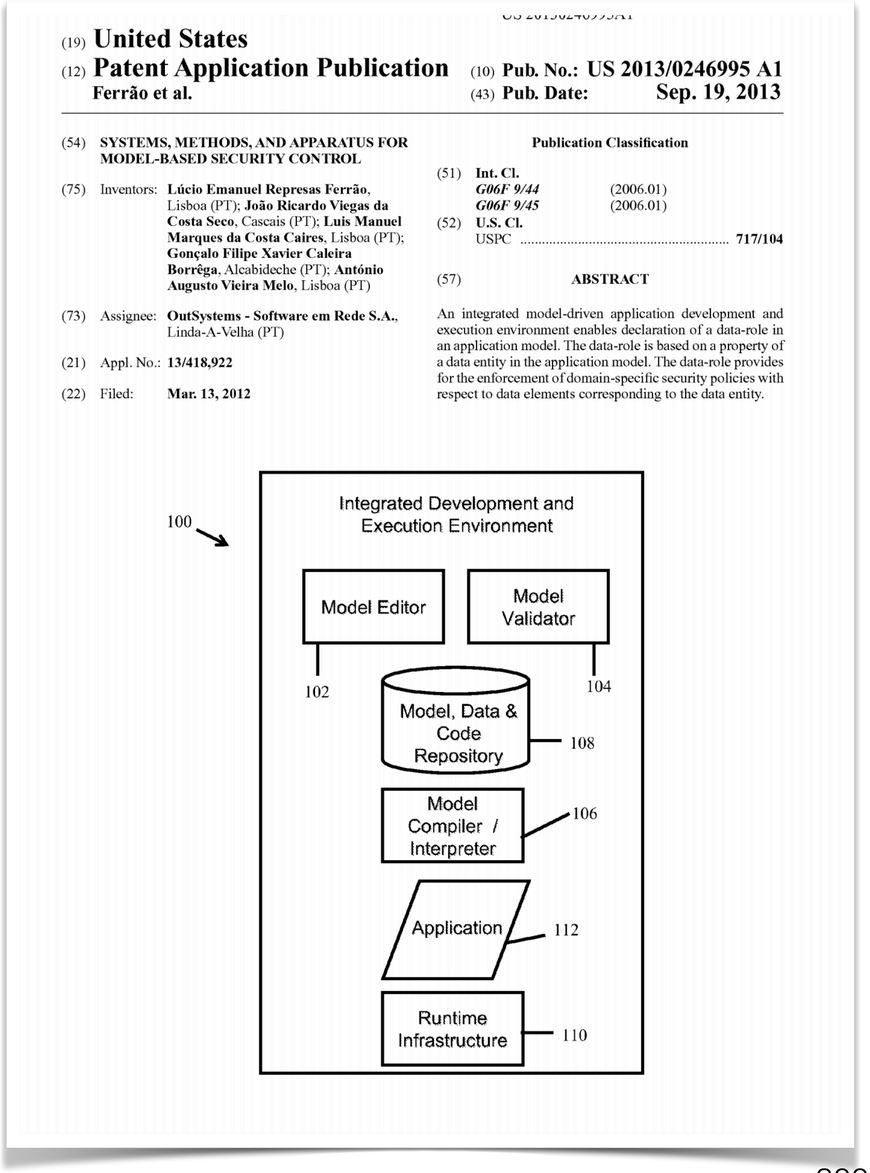
Luís Caires[1], Jorge A. Pérez[1], João C. Seco[1], Hugo T. Vieira[1], and Lúcio Ferrão[2]

[1] CITI and Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
[2] OutSystems SA

Data-centric multi-user systems, such as web applications, require flexible yet fine-grained data security mechanisms. Such mechanisms are usually enforced by a specially crafted security layer, which adds extra complexity and often leads to error prone coding, easily causing severe security breaches. In this paper, we introduce a programming language approach for enforcing access control policies to data in data-centric programs by static typing. Our development is based on the general concept of refinement type, but extended so as to address realistic and challenging scenarios of permission-based data security, in which policies dynamically depend on the database state, and flexible

Model-Based Access Control





(19) **United States**

(12) **Patent Application Publication**

Ferrão et al.

(10) **Pub. No.: US 2013/0246995 A1**

(43) **Pub. Date: Sep. 19, 2013**

(54) **SYSTEMS, METHODS, AND APPARATUS FOR MODEL-BASED SECURITY CONTROL**

(75) Inventors: **Lúcio Emanuel Represas Ferrão**, Lisboa (PT); **João Ricardo Viegas da Costa Seco**, Cascais (PT); **Luis Manuel Marques da Costa Caires**, Lisboa (PT); **Gonçalo Filipe Xavier Caleira Borrêga**, Alcabideche (PT); **António Augusto Vieira Melo**, Lisboa (PT)

(73) Assignee: **OutSystems - Software em Rede S.A.**, Linda-A-Velha (PT)

(21) Appl. No.: **13/418,922**

(22) Filed: **Mar. 13, 2012**

Publication Classification

(51) **Int. Cl.**
G06F 9/44 (2006.01)
G06F 9/45 (2006.01)

(52) **U.S. Cl.**
USPC **717/104**

(57) **ABSTRACT**

An integrated model-driven application development and execution environment enables declaration of a data-role in an application model. The data-role is based on a property of a data entity in the application model. The data-role provides for the enforcement of domain-specific security policies with respect to data elements corresponding to the data entity.

Further Reading

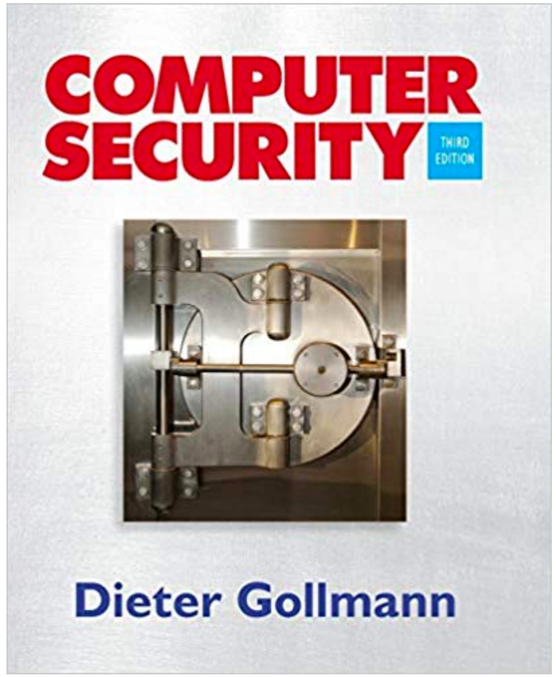
Computer Security. Dieter Gollmann. 3rd edition. Wiley, 2011.
OWASP Testing Guide V4. Owasp foundation, 2016. https://www.owasp.org/images/1/19/OTGv4.pdf

– CIA triad: confidentiality, integrity, and availability (A1, A6)
– Authentication, authorization, non-repudiation (A1, A2, A6)
– Risk, threats, vulnerabilities, and attack vectors (A3, A6)
– Concept of trust and trustworthiness (A4)
– Threat and attacker modelling (A3, A5, B1)

– Identification and authentication (A2, B3)
– Authorization and access control models (B2, B4, B5)
– Defensive programming (C2, D1)
– Software security testing (A5, C2, D2)

– Secure design basic principles (B2, B4, C1)
– Security best practices and standards (B3, B4, B5, B6, C2)
– Techniques for preserving security across modules and trust maintenance (A4, B1, B6, D1, D2, D3)

– Web security model (B6)
– Session management, authentication (B3, C3)
– Web application vulnerabilities and defenses (A5, B1, C3, C4)
– Client-side security (C4, D3, D4)
– Server-side security tools (C4, D3, D5)



Internet Applications Design and Implementation

(Lecture 6 - Part 4 - Using Kotlin & Spring)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

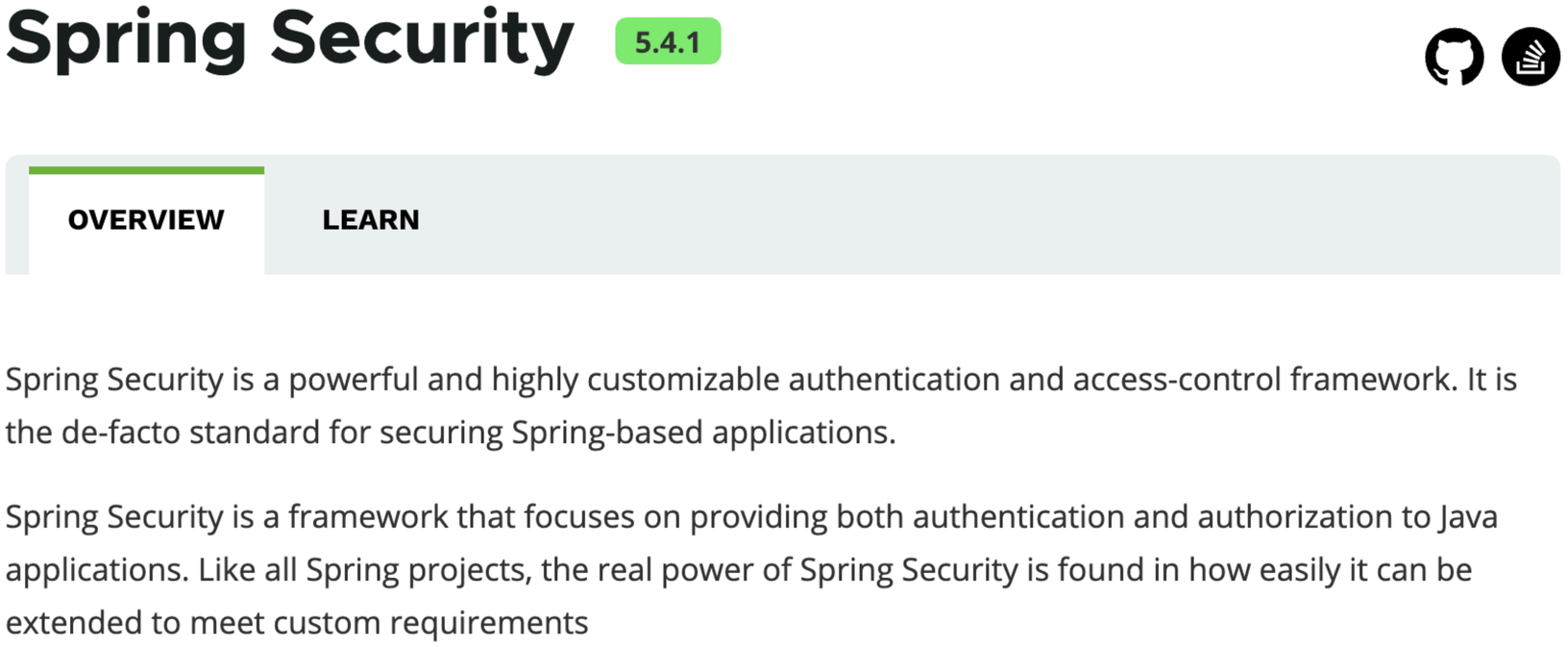
João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Using frameworks for security

- Frameworks such as Spring, with Spring Security project promote the reuse of (correct) code, good practices, and great number of base features.



https://spring.io/projects/spring-security

Using frameworks for security

- Frameworks such as Spring Security promote the reuse of (correct) code, good practices, and great number of base features.

  - HTTP BASIC authentication headers (an IETF RFC-based standard)
  - HTTP Digest authentication headers (an IETF RFC-based standard)
  - HTTP X.509 client certificate exchange (an IETF RFC-based standard)
  - LDAP (a very common approach to cross-platform authentication needs, especially in large environments)
  - Form-based authentication (for simple user interface needs)
  - OpenID authentication
  - Authentication based on pre-established request headers (such as Computer Associates Siteminder)
  - JA-SIG Central Authentication Service (otherwise known as CAS, which is a popular open source single sign-on system)
  - Transparent authentication context propagation for Remote Method Invocation (RMI) and HttpInvoker (a Spring remoting protocol)
  - Automatic “remember me" authentication (so you can tick a box to avoid re-authentication for a predetermined period of time)
  - Anonymous authentication (allowing every unauthenticated call to automatically assume a particular security identity)
  - Run-as authentication (which is useful if one call should proceed with a different security identity)
  - Java Authentication and Authorization Service (JAAS)
  - JEE container autentication (so you can still use Container Managed Authentication if desired)
  - Kerberos

Including Spring Security

- By just including the dependency in the application, security is automatically enabled.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Configuration is then possible by declaring basic access properties and basic user information.

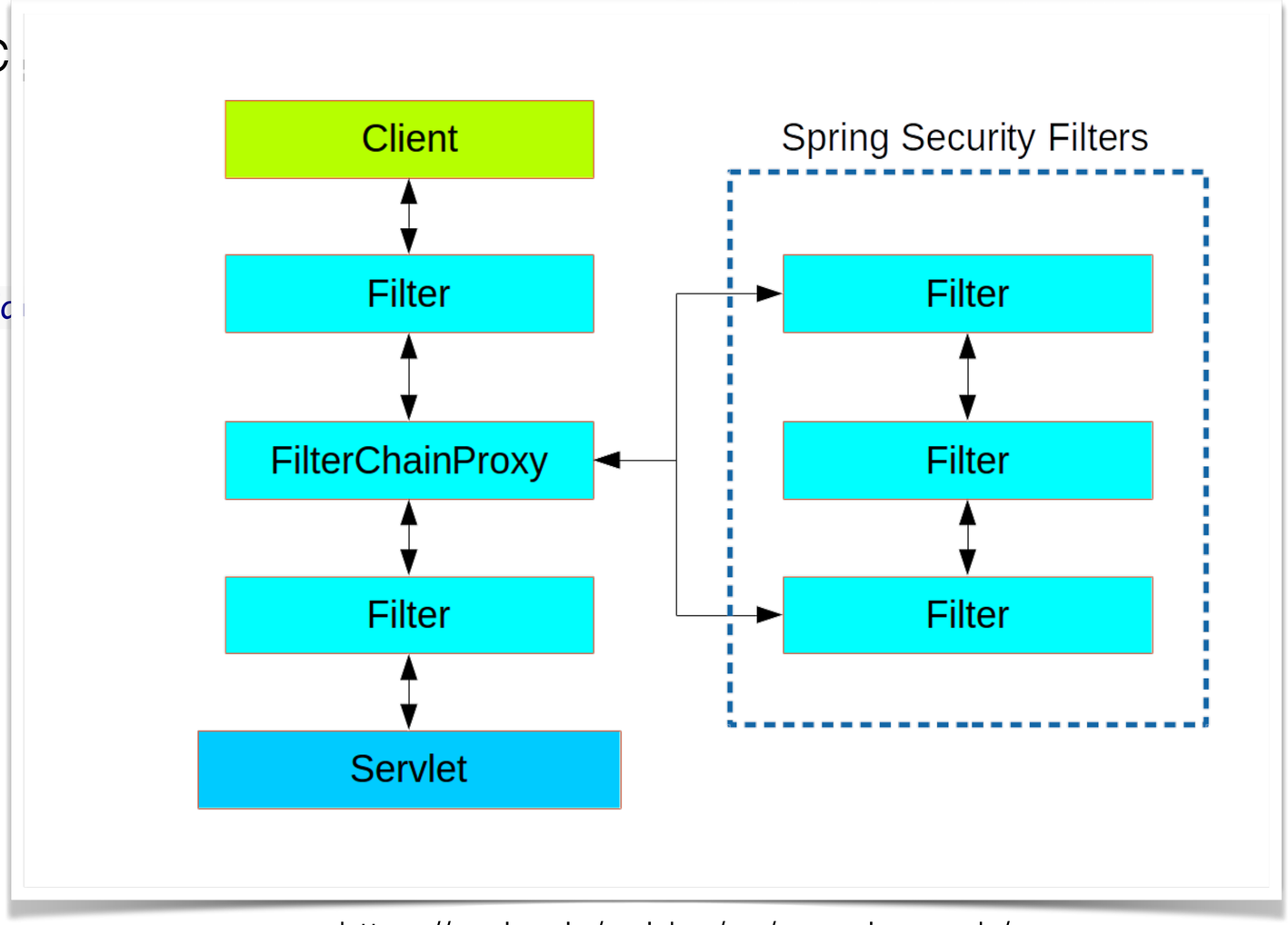
https://spring.io/guides/gs/securing-web/

Including Spring Security

- By just including the dependenc
  enabled.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifa
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Configuration is then possible by declaring basic access properties and basic user information.



https://spring.io/guides/gs/securing-web/

https://spring.io/guides/topicals/spring-security-architecture

Including Spring Security

- By just including the dependency i
  enabled.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactI
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

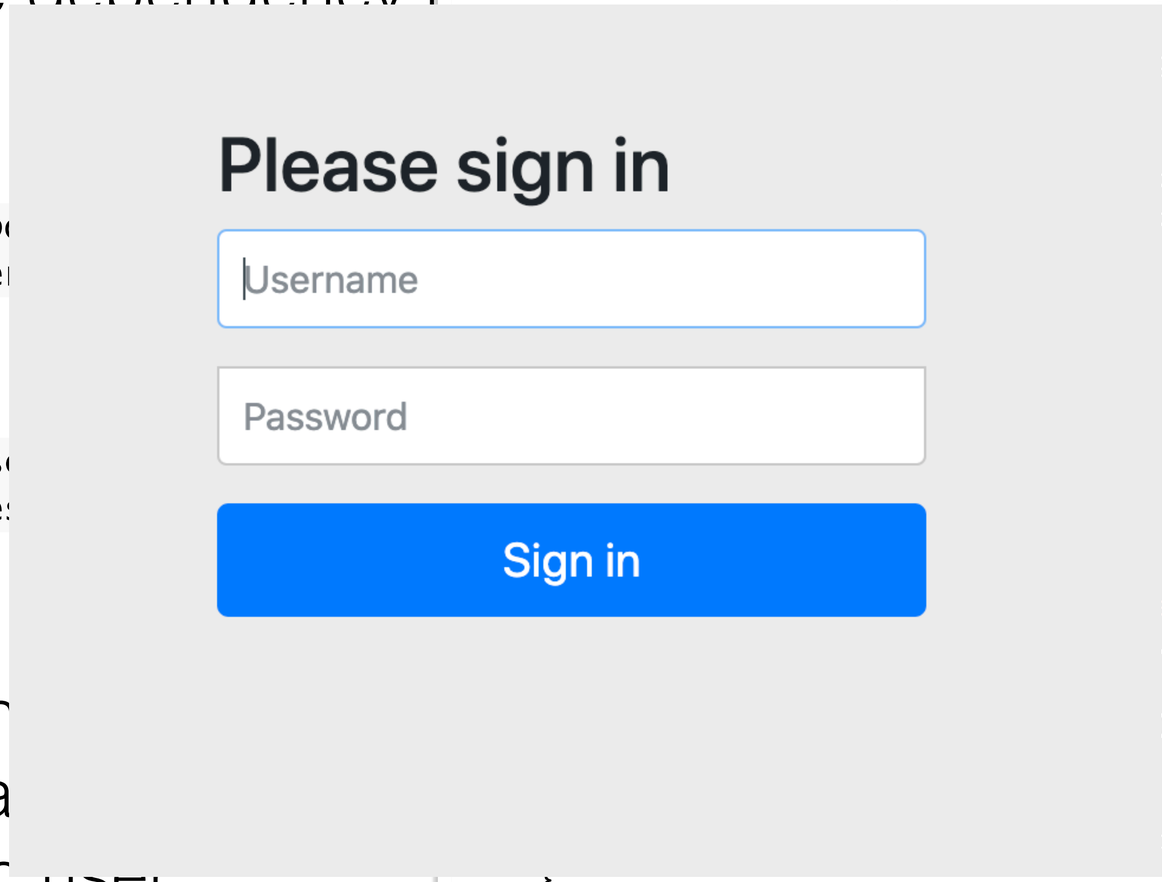
- Configuration is then possible by declaring basic access properties and basic user information.

```
@Configuration
@EnableWebSecurity
class WebSecurityConfig : WebSecurityConfigurerAdapter() {
    @Throws(Exception::class)
    override fun configure(http: HttpSecurity) {
        http
        .authorizeRequests()
            .antMatchers("/" ).permitAll()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .permitAll()
            .and()
        .logout()
            .permitAll()
    }

    @Bean
    public override fun userDetailsService(): UserDetailsService {
        val user: UserDetails = User.withDefaultPasswordEncoder()
                .username("user")
                .password("password")
                .roles("USER")
                .build()
        return InMemoryUserDetailsManager(user)
    }
}
```

Including Spring Security

- By just including the dependency i
  enabled.

```
<dependency>
  <groupId>org.springframework.bo
  <artifactId>spring-boot-starter
</dependency>

<dependency>
  <groupId>org.springframework.se
  <artifactId>spring-security-tes
  <scope>test</scope>
</dependency>
```




```
@Configuration
@EnableWebSecurity
class WebSecurityConfig : WebSecurityConfigurerAdapter() {

                                    oSecurity) {

                                    tAll()
                                    ted()

                                    ervice(): UserDetailsService {
                                    .withDefaultPasswordEncoder()

                                    )

                                    nager(user)
    }
}
```

- Configuration is ther
  by declaring basic a
  properties and basic user
  information.

Including Spring Security

- By just including the dependen enabled.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</arti
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Configuration is then possible by declaring basic access properties and basic user information.

```
@Configuration
@EnableWebSecurity
class WebSecurityConfig : WebSecurityConfigurerAdapter() {
    override fun configure(http: HttpSecurity) {
        http
        .csrf().disable()
        .authorizeRequests()
            .antMatchers("/" ).permitAll()
            .anyRequest().authenticated()
        .and().httpBasic()
    }

    override fun configure(auth: AuthenticationManagerBuilder) {
        auth.inMemoryAuthentication()
                .withUser("user")
                .password(BCryptPasswordEncoder().encode("password"))
                .authorities(emptyList())
            .and()
                .passwordEncoder(BCryptPasswordEncoder())
    }
}
```

https://spring.io/guides/gs/securing-web/

Including Spring Security

- By just including
  enabled.

```
<dependency>
  <groupId>org.springframe
  <artifactId>spring-boot-
</dependency>

<dependency>
  <groupId>org.springframe
  <artifactId>spring-secur
  <scope>test</scope>
</dependency>
```

- Configuration is
  by declaring ba
  properties and
  information.

jcs — -zsh — 80×24

```
jcs@Joaos-iMac ~ % http :8080/applications
HTTP/1.1 403
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Connection: keep-alive
Content-Type: application/json
Date: Mon, 19 Oct 2020 14:30:34 GMT
Expires: 0
Keep-Alive: timeout=60
Pragma: no-cache
Set-Cookie: JSESSIONID=2454FD5814FB3EEBF80E07CF1737C10D; Path=/; HttpOnly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

{
    "error": "Forbidden",
    "message": "",
    "path": "/applications",
    "status": 403,
    "timestamp": "2020-10-19T14:30:34.051+00:00"
}

jcs@Joaos-iMac ~ %
```

```
pter() {

agerBuilder) {

encode("password"))

oder())
```

https://spring.io/guides/gs/securing-web/

Including Spring Security

- By just including enabled.

```
<dependency>
  <groupId>org.springframe
  <artifactId>spring-boot-
</dependency>

<dependency>
  <groupId>org.springframe
  <artifactId>spring-secur
  <scope>test</scope>
</dependency>
```

- Configuration is by declaring bas properties and b information.

jcs — -zsh — 80×24

```
jcs@Joaos-iMac ~ % http :8080/applications --auth user:password
HTTP/1.1 200
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Connection: keep-alive
Content-Type: application/json
Date: Mon, 19 Oct 2020 14:57:40 GMT
Expires: 0
Keep-Alive: timeout=60
Pragma: no-cache
Set-Cookie: JSESSIONID=2361CEB5951A39C2967863030D2EBA48; Path=/; HttpOnly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

[]

jcs@Joaos-iMac ~ %
```

```
ter() {

gerBuilder) {

ncode("password"))

der())
```

https://spring.io/guides/gs/securing-web/

Authentication - Simple starter code

- The default security setting provides that all requests are protected, a login form is created, RESTful interface for login/logout…

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("user").password("password").roles("USER");
    }
}
```

Look into the documentation for the up-to-date API!!

Fundamental Concepts mapped to Spring Context

- Principal - Who is the entity behind a particular request

```
@RequestMapping("/messages/inbox")
public ModelAndView findMessagesForUser(@AuthenticationPrincipal CustomUser user) {

    // .. find messages for this user and return them ...
}
```

- Authentication: Certify that a given set of credentials identify one principal

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .jdbcAuthentication()
            .dataSource(dataSource)
            .withDefaultSchema()
            .withUser("user").password("password").roles("USER").and()
            .withUser("admin").password("password").roles("USER", "ADMIN");
}
```

Look into the documentation for the up-to-date API!!

Linking to the application model

- Dynamic verification of user credentials is part of the application model

```
override fun configure(auth: AuthenticationManagerBuilder) {
    auth.inMemoryAuthentication()
            .withUser( username: "user")
            .password(BCryptPasswordEncoder().encode( rawPassword: "password"))
            .authorities(emptyList())
            .and()
            .passwordEncoder(BCryptPasswordEncoder())
            .and()
            .userDetailsService(customUserDetails)
}
```

Look into the documentation for the up-to-date API!!

Linking to the application model

- Dynamic verification of user credentials is part of the application model

```
override fun configure(auth: AuthenticationManagerBuilder) {
    auth.inMemoryAuthentication()
            .withUser( username: "
            .password(BCryptPassw
            .authorities(emptyLis
            .and()
            .passwordEncoder(BCry
            .and()
            .userDetailsService(
}
```

```
@Service
class CustomUserDetailsService(
        val users: UsersService
) : UserDetailsService {

    override fun loadUserByUsername(username: String?): UserDetails {

        username?.let { it: String
            val userDAO = users.findUser(it)
            if( userDAO.isPresent ) {
                return CustomUserDetails(userDAO.get().username, userDAO.get().password, mutableListOf())
            } else
                throw UsernameNotFoundException(username)
        }
        throw UsernameNotFoundException(username)
    }
}
```

Look into the documentation for the up-to-date API!!

Security of Internet Applications

- Authorisation - Check if a given principal has rights to access a given piece of information

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/resources/**", "/signup", "/about").permitAll()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/db/**").access("hasRole('ROLE_ADMIN') and hasRole('ROLE_DBA')")
            .anyRequest().authenticated()
            .and()
        // ...
        .formLogin();
}
```

- Delegation - to be able to temporarily assign one principals capabilities' to another principal (e.g. sudo)

org.springframework.security.web.authentication.switchuser

Class SwitchUserFilter

java.lang.Object
  org.springframework.web.filter.GenericFilterBean
    org.springframework.security.web.authentication.switchuser.SwitchUserFilte

All Implemented Interfaces:

javax.servlet.Filter, Aware, BeanNameAware, DisposableBean, InitializingBean, Ap

http://docs.spring.io/autorepo/docs/spring-security/3.2.1.RELEASE/apidocs/org/springframework/security/web/authentication/switchuser/SwitchUserFilter.html

Basic Spring Security Guaranties

There really isn't much to this configuration, but it does a lot. You can find a summary of the features below:

- Require authentication to every URL in your application
- Generate a login form for you
- Allow the user with the **Username** *user* and the **Password** *password* to authenticate with form based authentication
- Allow the user to logout
- CSRF attack prevention
- Session Fixation protection
- Security Header integration
  - HTTP Strict Transport Security for secure requests
  - X-Content-Type-Options integration
  - Cache Control (can be overridden later by your application to allow caching of your static resources)

From Spring Security docs

Basic Spring Security Guaranties

- Security Header integration
  - HTTP Strict Transport Security for secure requests
  - X-Content-Type-Options integration
  - Cache Control (can be overridden later by your application to allow caching of your static resources)
  - X-XSS-Protection integration
  - X-Frame-Options integration to help prevent Clickjacking
- Integrate with the following Servlet API methods
  - HttpServletRequest#getRemoteUser()
  - HttpServletRequest.html#getUserPrincipal()
  - HttpServletRequest.html#isUserInRole(java.lang.String)
  - HttpServletRequest.html#login(java.lang.String, java.lang.String)
  - HttpServletRequest.html#logout()

From Spring Security docs

CSRF - token issued by the server / recognised by the same server

6. Cross Site Request Forgery (CSRF)

This section discusses Spring Security's Cross Site Request Forgery (CSRF) support.

6.1. CSRF Attacks

Before we discuss how Spring Security can protect applications from CSRF attacks, we will explain what a CSRF attack is. Let's take a look at a concrete example to get a better understanding.

Assume that your bank's website provides a form that allows transferring money from the currently logged in user to another bank account. For example, the HTTP request might look like:

From Spring Security docs

```
POST /transfer HTTP/1.1
Host: bank.example.com
Cookie: JSESSIONID=randomid; Domain=bank.example.com; Secure; HttpOnly
Content-Type: application/x-www-form-urlencoded

amount=100.00&routingNumber=1234&account=9876
```

CSRF - token issued by the server / recognised by the same server

- Template example…

```
<div th:if="${#httpServletRequest.remoteUser}!=null">
  <label>User: </label><span th:text="${#httpServletRequest.remoteUser}"></span>
  <form style="display:inline-block" th:action="@{/logout}" method="post">
      <input type="submit" value="Sign Out"/>
  </form>
</div>

<div th:if="${#httpServletRequest.remoteUser} == null" >
  <form th:action="@{/}" method="post">
      <label> User : <input type="text" name="username"/> </label>
      <label> Password: <input type="password" name="password"/> </label>
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
      <input type="submit" value="Sign In"/>
  </form>

</div>
```

CSRF - token issued by the server / recognised by the same server

- Actual page example

```
<div class="container">
  <div>
    <form method="post" action="/">
      <label>
        " User : "
        <input type="text" name="username">
      </label>
      <label>
        " Password: "
        <input type="password" name="password">
      </label>
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
      <input type="submit" value="Sign In">
      <input type="hidden" name="_csrf" value="f067a25f-c6f5-4de0-b6ad-ea7416986b22">
    </form>
  </div>
</div>
```

Internet Applications Design and Implementation

(Lecture 7 - Model Based Access Control and Token-based Security)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Internet Applications Design and Implementation

(Lecture 7 - Part 1 - Model-based Authorisation in Spring)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

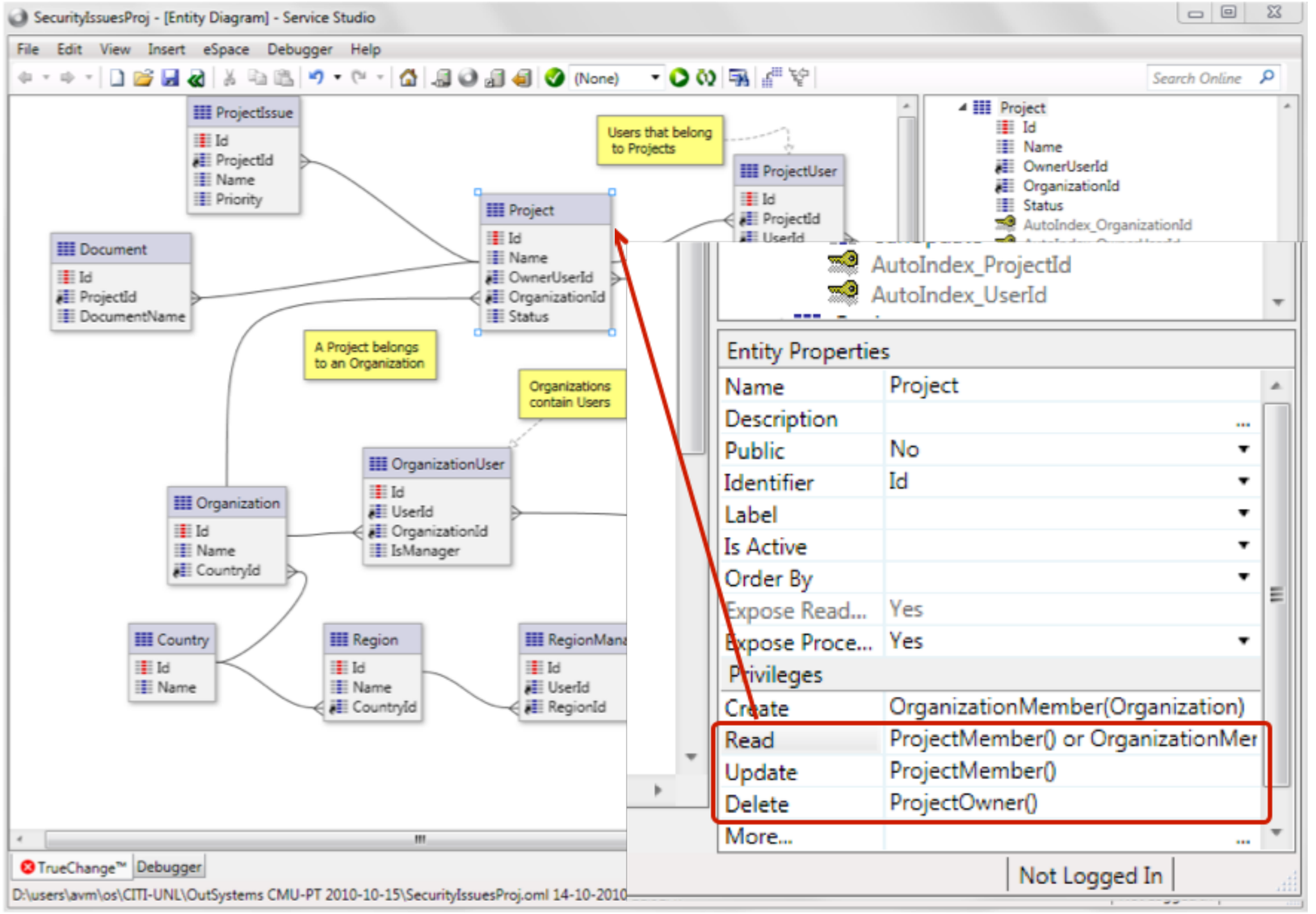
(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

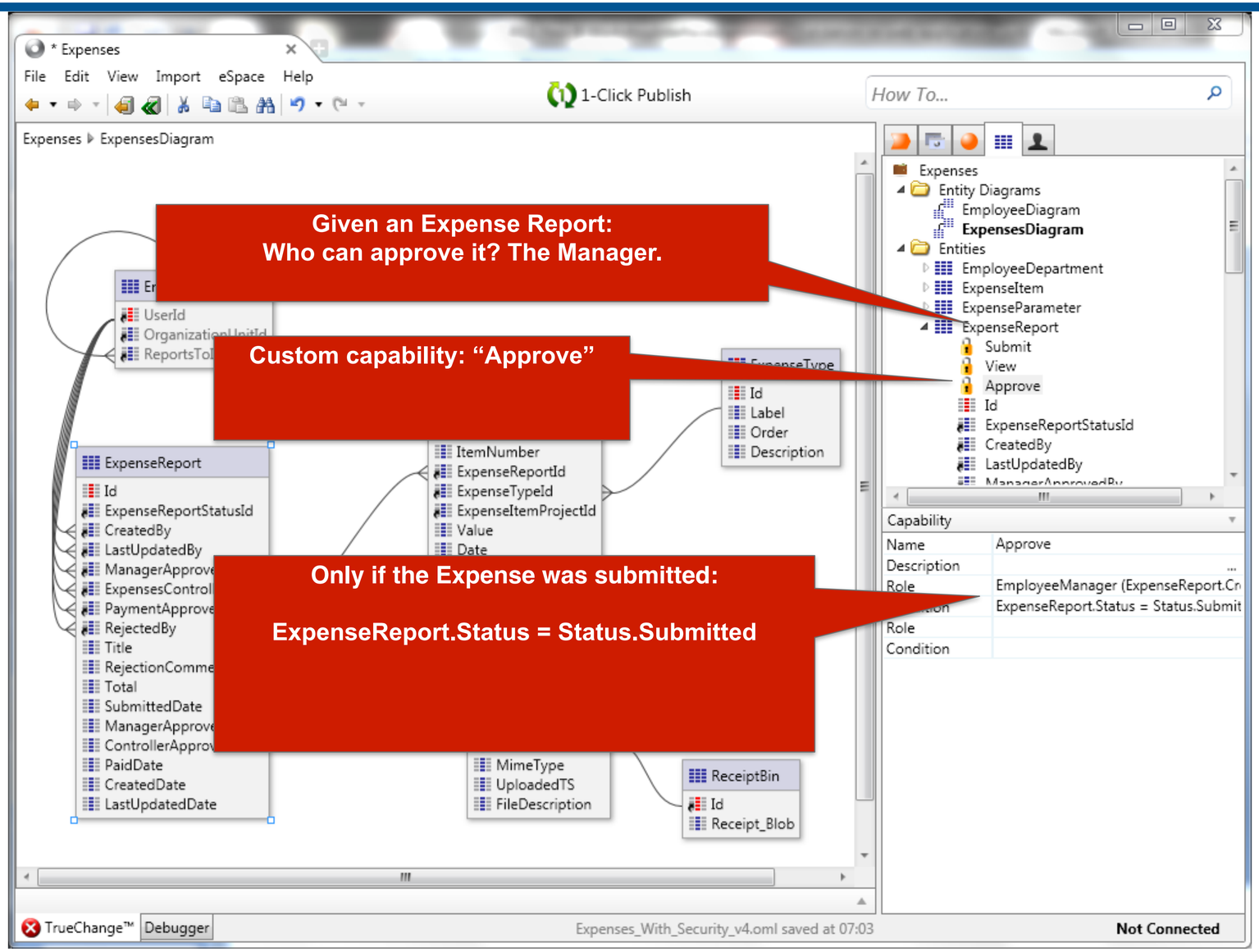
Model Based Access Control

- Security conditions are many times hidden in query filters and program conditions. They are very difficult to maintain/modify/get right.
- Developer defined roles usually depend on the state of the application. (e.g. ModeratorOf(…)). Role-based models ignore this.
- Authorisations usually depend on the state of the target entity (status == submitted)
- Capabilities defined by the developer extend read/write/delete (not hardwired to the basic programming elements)

Model Based Access Control



Model Based Access Control - Custom Cap.



Model Based Access Control - Spring?

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .antMatchers("/admin/hotels/{id}")
            .access(id => managerOfHotel(id))
            .access(id => hotels.findOne(id).isApproved())
//…
}
```

Not really Spring… how could we do this?

Security and Frameworks (Spring)

- Web Development Frameworks provide basic mechanisms like:
  - Authentication
    - username, password, authorities (roles)
  - Authorization
    - hasRole, hasPermission

Role Based Access Control

- Application level pattern implemented on top of basic authentication mechanisms
  - Principal (The authenticated user)
  - Role (Job functions a user may have)
  - Capabilities (Systems operations a role may be associated with)
- Allows the dynamic assignment of capabilities to users
- Allows sophisticated assignment and delegation mechanisms (e.g. roles assigned in name of another principal, or by a limited time).

Model based security

- Security conditions are part of the application level
- Authentication and authorisation are system level

```
// POST /hotels/{id}         - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@PreAuthorize("hasRole('{ROLE_ADMIN,ROLE_MANAGER}')")
public String editSave(@PathVariable("id") long id, Hotel hotel,
                       Model model, Principal principal) {
    if( principal.getName().equals(hotel.getManager().getName())) {
        hotel.setTimestamp(new Date());
        hotels.save(hotel);
        return "redirect:/hotels/"+id;
    } else
        throw new AccessDeniedException("Cannot edit hotel "+hotel.getName());
}
```

- Security conditions are many times encoded into application logic (if-else-throw statements) and where conditions in queries.

Spring Security implementation

- Basic spring security is based on SPEL expressions using principal information and parameters…

```
// POST /hotels/{id}         - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@PreAuthorize("hasRole({'ROLE_ADMIN','ROLE_MANAGER'}) "
            + "and principal.name == #hotel.manager.name")
public String editSave(@PathVariable("id") long id, Hotel hotel,
                       Model model, Principal principal) {
    hotel.setTimestamp(new Date());
    hotels.save(hotel);
    return "redirect:/hotels/"+id;
}
```

- Rule: “a hotel can be edited by its manager or an administrator”
  - Problem: **`#hotel.manager.name`** is not loaded !!

Pros & Cons

- + May use model information
- + Provides row-level authorisation on resources
- - Has limited access to the model (database)
- - Bad reuse of condition expressions
- - Runtime parsing and execution of security expressions

Policies or security rules

- Isolate the implementation of security rules in services with full access to the model.

```
@Component("mySecurityService")
public class MySecurityService {

    @Autowired
    HotelRepository hotels;

    public boolean canEditHotel(User user, long hotelId) {

        Hotel hotel = hotels.findOne(hotelId);

        return hotel != null && hotel.getManager() != null &&
                user.getUsername().equals(hotel.getManager().getName());
    }
}
```

- Model access, Good reuse, auditing of rules.

Policies or security rules in “services”

- Isolate the implementation of security rules in services with full access to the model.

```
// POST /hotels/{id}        - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@PreAuthorize("hasRole({'ROLE_ADMIN'}) or "
              + "@mySecurityService.canEditHotel(principal,#id)")
public String editSave(@PathVariable("id") long id, Hotel hotel,
                       Model model, Principal principal) {
      hotel.setTimestamp(new Date());
      hotels.save(hotel);
      return "redirect:/hotels/"+id;
}
```

- Model access, Good reuse, auditing of rules.

Capabilities encoded in “annotations”

- Statically identify capabilities and associated rules in annotations, associated to controllers…

```
// POST /hotels/{id}          - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@AllowedForEditHotel
public String editSave(@PathVariable("id") long id, Hotel hotel, Model model) {
    hotel.setTimestamp(new Date());
    hotels.save(hotel);
    return "redirect:/hotels/"+id;
}
```

Capabilities encoded in “annotations”

```java
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
@PreAuthorize(AllowedForHotelCreation.Condition)
public @interface AllowedForHotelCreation {
    String Condition = "hasRole('ROLE_ADMIN')";
}
```

```java
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
@PreAuthorize(AllowedForEditHotel.Condition)
public @interface AllowedForEditHotel {
    String Condition = "@mySecurityService.canEditHotel(principal,#id) or "
                       + AllowedForHotelCreation.Condition;
}
```

http://blog.novoj.net/2012/03/27/combining-custom-annotations-for-securing-methods-with-spring-security/

View authorisations

- Authorization expressions can be also used in views (with an akward syntax, yes…)

```
<div class="container">
    <h1>Hotel <span th:text="${hotel.name}"></span></h1>
    <p th:text="${hotel.timestamp}"></p>

    <h2>Rooms</h2>
    <ul th:each="room : ${rooms}">
    <li th:text="${room.name}+ ' ' + ${room.type.description}"></li>
    </ul>

    <div sec:authorize="${#authorization.expression(T(booking.security.AllowedForEditHotel).Condition2)}">
        <p><a th:href="@{|/hotels/${hotel.id}/edit|}">Edit</a></p>
    </div>
    <div sec:authorize="${#authorization.expression(T(booking.security.AllowedForHotelCreation).Condition)}">
        <p><a href="/hotels/new">Add a new hotel</a></p>
    </div>
    <p><a href="/hotels">List all hotels</a></p>
</div>
```

- Binding is sometimes different and has to be adapted :(

```
String Condition = "@mySecurityService.canEditHotel(principal,#id) or "
                   + AllowedForHotelCreation.Condition;
String Condition2 = "@mySecurityService.canEditHotel(principal,#vars.hotel.id) or "
                   + AllowedForHotelCreation.Condition;
```

Pros & Cons

- + May use model information
- + Provides row-level authorisation on resources
- + Has full access to the model (database)
- + Good reuse of condition expressions (in services)
- - Runtime parsing and execution of security expressions
  (only better using static verification of code)

Further Reading

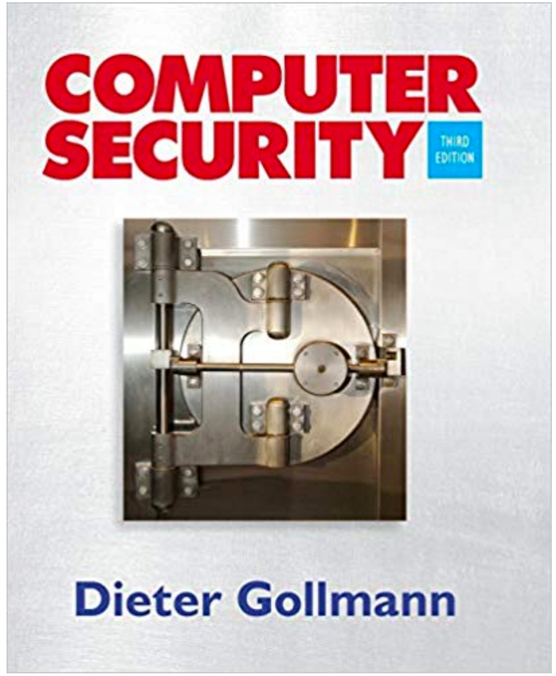
Computer Security. Dieter Gollmann. 3rd edition. Wiley, 2011.

OWASP Testing Guide V4. Owasp foundation, 2016.
https://www.owasp.org/images/1/19/OTGv4.pdf

Blogs:

https://heimdalsecurity.com/blog/best-internet-security-blogs/

https://www.schneier.com



OWASP
Open Web Application
Security Project

Overview: HTTP Authentication (Basic and Digest)

- Credentials (username/password) are repeated on each request
  - All requests are vulnerable to attacks (instead of only the login request)
  - Basic: username/password are passed in clear text and can be captured
  - Digest: digests can also be captured and guessed by brute force attacks

- Kind of ok under HTTPS, but…
  - Must have a centralised authority to control and manage principal capabilities
  - Does not easily support “logout” mechanisms (credentials are “always” valid)

Overview: Sessions to implement security

- Stateless APIs are good
- but not so good for:
  - ephemeral or distributed authentication,
  - capability based authorisation models
  - protocol management,
  - user preferences (in webapps)
  - …
- Hence, let's implement session management

Outline

- Sessions and cookies
- Token based authentication
- JSON Web Token (JWT)
- OAuth2