oTN-2010-01 Openlab Summer Student Report

TRoIE (Test-bench for Robustness of Industrial Equipments) Reporting System

Gazmend Bajrami

Supervisor: Filippo Tilaro

27 August 2010

Version 2.1

Contents

Abstract

The aim of this project at CERN was to design and implement the database which contains information about PLC's vulnerabilities collected by the TRoIE test bench. Scanning different computers, networks and PLCs is a very important process and a way to find out possible vulnerabilities on these devices and to understand the weaknesses and the risks that they convey with their operation on different environment. It is also an important step before taking any preventative measures. Through the scanning we can identify the weaknesses of the system under analysis and the information collected are very helpful to fix any issues in order to make the system more secure. This could avoid that any possible attack can exploit these vulnerabilities to get access to system and to the information as well. In this report we will describe all the steps taken from the beginning of database design phase to the final implementation.

Key words: Vulnerabilities Report System, Spring Roo

Introduction

Over the last ten years the number of attacks is much higher than it was before as well as the number of vulnerabilities is increased as the Figure 1 shows. Within this period the technology has advanced very much and has changed the way of organizing the work. If from one hand it has made it easier, on the other hand these technology improvements have introduced new vulnerabilities and security issues.

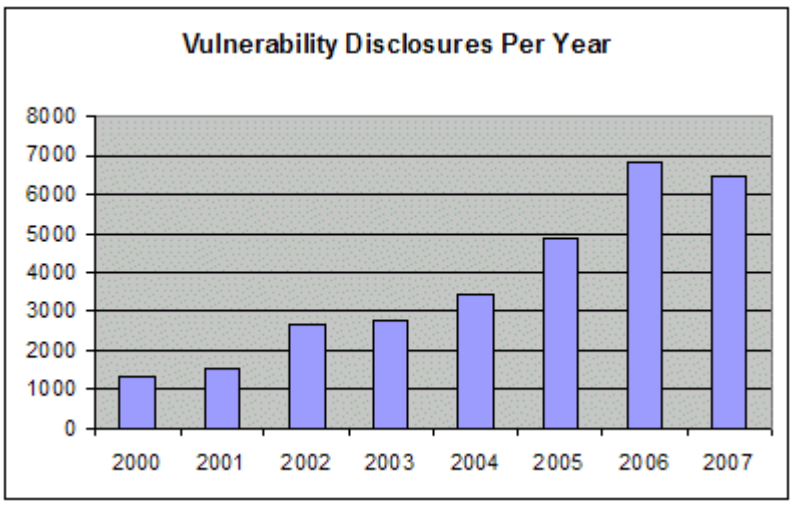


Figure 1: Number of vulnerabilities per year [5].

The security holes can exist on: Operating systems, applications, and network protocols. A vulnerability can be a programming error or misconfigurations that an attacker or an intruder can exploit to gain unauthorized access. A scanning vulnerability is important to discover any vulnerability which could compromise the stability of the entire system. In a second phase a patching activity is necessary to resolve these security issues. In our specific case the TRoIE test bench is finalized to test both the individual devices - before any deployment - and all the components that are already part of the system.

Vulnerabilities scanning tools check for any hole on the particular target system and some of the tools give information for potential solution. Scanning tool collect all the information during the scanning process and create a list of vulnerabilities for the particular devices which is presented in a final report. So far each security tool (for example Nessus, OpenVas etc.) produces its own report presenting all the information in a specific structure. As a result all this amount of information is difficult to read and manage. This is why in our project we decided to create a database which will store and organize all the information taken from different scanning tools in a uniform and homogeneous way. Moreover it will allow us to query the database for specific information filtering by a particular device, model, test and so on.
After an initial analysis we have chosen to implement the database using the Spring Roo framework which is a quite new and open source technology.

Databases Overview

A **database** is an organized collection of associated data that is stored for a specific purpose, typically in digital form. This collection is arranged in a fixed structure, in such way that they easily be accessed, managed, and updated. One way of classifying databases is according to types of content: bibliographic, full-text, numeric, and images.
In computing, databases are sometimes classified according to their organizational approach. The most used approach is the relational database, a tabular database. A distributed database is one that can be dispersed or replicated among different points in a network. An object-oriented programming database is one that is congruent with the data defined in object classes and subclasses. In our case we have combined the object oriented programming database with relational database using Spring Roo framework in combination with Hibernate technology. Moreover the mentioned products let us to choose in a totally transparent way the specific Database Management Systems (Oracle, MySQL, MSSQL, DB2 etc...).

Database Design Process

The database design process consists of some specific and ordered steps, which are mostly based on three phases: analysis, design and implementation.

Analysis phase

In the analysis phase of the database design process we started to collect the necessary information to store:

- All the data coming from Security analyzer applications.
- Be in line with the CRT requirements [8]
- Configuration environment used for the test

- Exploited Attack Patterns
- Specific technical specifications related to PLCs/Industrial devices

After an initial comparison with the Nessus and OpenVas reports we established to organize the database into these tables: *Devices (PLCs), Tests, Vulnerabilities, Configuration I/O, Connection Information, Application that are running on the device, Attacks, Monitoring System, Scanning tool, Scanner*. All these data are related to each other and necessary for the database.

Database Tables

Each of database tables holds specific information about the devices, specific performed tests, network configuration and possible related vulnerabilities. In the following we will give a brief description for each table.

- **Device** Table**:** This table contains the information about the scanned devices

Field Name	Data Type	Description
Device ID	INT Auto Increment	This filed is Primary Key field the contain IDs of different devices that have been scanned
Manufacturer	Varchar	Information about the Manufacture of the Device
Device Name	Varchar	The name of the Device
Device Type	Varchar	Describes the type of device
Order Number	Varchar	This field contain the Order Number for the device
Serial Number	Varchar	The Serial Number of the device
Firmware Version	Varchar	Firmware version of the Device
Operating System	Varchar	This field contain Operating System used by device

- **Test** Table**:** This table is used to holds information for each specific test against particular devices.

Field Name	Data Type	Description
Test ID	INT Auto Increment	Is the ID for each different Test and it is a Primary key field.
Test Name	Varchar	Name of the Test
Test-Start	TimeDate	The time and date where the Test start
Test-End	TimeDate	The time and date where the Test has finished
Termination Status	Enum	
Communication load	Double	Percentage of the scan cycle time left to the communication

- **Vulnerability** Table: this table describes the discovered vulnerabilities. The table contains the information for the risk related to the specific threat which affects the device.

Name of the Field	Data Type	Description
Vulnerability ID	INT Auto Increment	The ID for each Vulnerability
Impact Severity level	Enum	High, Low and Medium
Risk Factor	Text	This field describe the risk that threatens the device
Port Affected	INT	Communication Port
Service affected	Varchar(30)	Name of the service running on that specific port
Protocol affected	Varchar(30)	Name of the protocol which is affected
Synopsis	Text	General and quite short summary of the issue
Description	Text	Description of the issue
Possible Solution	Text	Possible solution to the specific issue
Packet Capture	Blob	Sequence Packets capture file
Plugin output	Text	Output of the plug-in used to perform the test

- **Configuration I/O** table: the Configuration I/O table contains the devices configuration related to the I/O process.

Name of the Field	Data Type	Description
Configuration ID	INT Auto Increment	
Number of Used Input	INT	Digital/Analog ports used as INPUT
Number of Used Output	INT	Digital/Analog ports used as OUTPUT
Input signal frequency	Double	Frequency of the signal in INPUT
Output signal frequency	Double	Frequency of the signal in OUTPUT

- **Connection Info** table: it will provide information about the network communication during the tests.

Name of the Field	Data Type	Description
Connection ID	INT Auto Increment	
Local port open	INT	Number of the open port in the target device for the specific connection
Remote port open	INT	Number of the open port in the partner device for the specific connection
Type of communication (protocol)	ENUM	Name of the protocol used for the specific connection
IP of Target	Varchar	IP address of the target device
IP of Partner	Varchar	IP address of the partner device

Device partner ID	INT	ID of the partner
Active/Passive	Enum	Active if the target device will establish the communication. Passive otherwise
Bit rate input	Double/Float	INPUT Data Bit/rate
Bit rate output	Double/Float	OUTPUT Data Bit/rate

- **Application running** table: this table will keep track of information about the application that are running in the device

Name of the Field	**Data Type**	**Description**
Application running ID	INT Auto Increment	
Instruction set used	Enum	Type of the instruction used in the target Data Blocks
Block Type in execution	Enum	Numbers of the Block Type used during the tests (OB1, 0B80, etc…)
Scan Cycle	Time	Duration of the scan cycle in msec
Protection	Text	Type of the protection
Description of protection	Text	Description of the used protection
Startup mode	Enum	Warm /Cold/ Hot Restart
CPU usage	Double/decimal	Percentage of the CPU usage during a normal execution

- **Attack pattern** table: represents the possible attack that can exploit various different vulnerabilities to gain unauthorized access.

Name of the Field	**Data Type**	**Description**
ID	INT Auto Increment	
Parent ID	INT	ID of the father Attack pattern
Name	Varchar	Name of the attack pattern
Description	Text	Description of the attack pattern

- **Monitoring System** table: it will keep track of Monitoring System used when the vulnerabilities are detected.

Name of the Field	**Data Type**	**Description**
Monitoring ID	INT Auto Increment	
Type of monitoring	Enum	I/O Process, Communication, PLC status Monitoring
Description	Text	Description of the specific monitoring system

- **Scanning tool** table: this table provides the information for possible plug-in that some scanner provides, for example Nessus consists of different sub-plug-ins.

Field Name	Data type	Description
ID	INT	ID of the scanner the plug-in belongs to
Tool ID	INT Auto Increment	
Tool Name	Text	Name of the plug-in
Tool Family	Text	Family of the specific plug-in (General, Service Detection...)
Tool Version	Text	Version of the plug-in
Description	Text	Description of the plug-in

- **Scanner** table: will provide information about the Security analyzers application.

Field Name	Data type	Description
ID	INT Auto Increment	
Name	Text	Name of the scanner
Version	Text	Version of the scanner
Description	Text	Description of the scanner tool

Design phase

The design phase brings up the concept of 'data models'. Data models are data flow diagrams and system flow charts or schemas, which are used to present the data requirements at different levels of abstraction. In this phase we have produced a conceptual model for the database, using Entity Relationship Diagrams or E R diagrams. The E-R diagram for our database is presented in the figure 2.

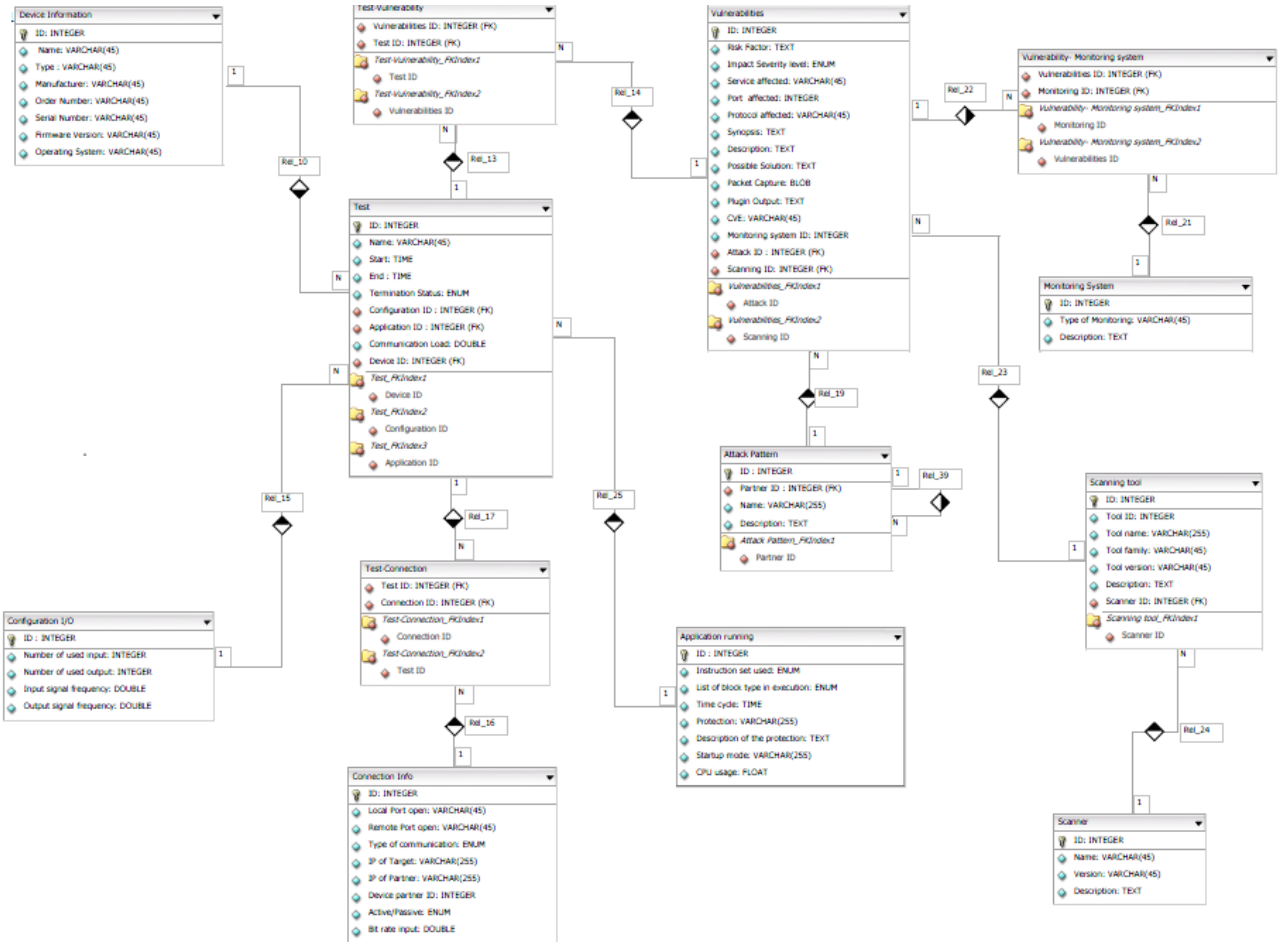


Figure 2: The E-R diagram for database

The E-R diagram in figure 2 shows the database tables and the relationships between the tables. In this database we have used two types of relationships: One-To-Many and Many-To-Many.

OneToMany relationships in database's E-R diagram are between:

- **Device** and **Test**- this relationship is defined because one device can be tested more than one time with different configuration.
- **Configuration I/O** and **Test** - one test can have only one configuration I/O, but one configuration of Input and Output can be used in different tests.
- **Application running** and **Test**-
- **Attack pattern** and **Vulnerabilities –** An attack can exploit different vulnerabilities.
- **Scanning Tool** and **Vulnerabilities** - As we mention before the table Scanning tool provide information for the plug-in that some scanner offer. This relationship is One-To-Many because different vulnerabilities can be related to one plug-in

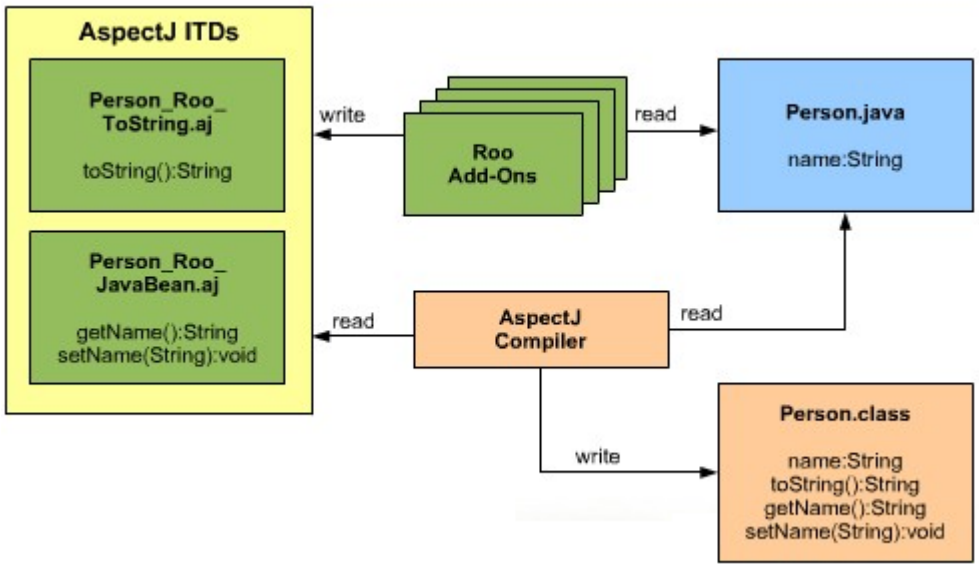
- **Scanner** and **Scanning tool -** A scanner provide more than one plug-in.

Many-To-Many relationships in database's E-R diagram are between:

- **Test** and **Vulnerabilities –** under the test we can detect more vulnerabilities, and one vulnerability can be detected with different tests.
- **Test** and **Connection Info -**

Implementation Phase

We have chosen Spring Roo to implement our database: it is an open source software and easy-to-use tool for building application in Java. Spring Roo uses some useful technologies (such as Spring Framework, Spring Security and Spring Web Flow), Maven, Java Server Pages (JSP), Java Persistence API (JPA, such as Hibernate), Tiles and AspectJ [Roo]. It is a high productivity and efficient tool: with Roo we can build sophisticated enterprise applications. It let us choose we can choose various different databases for our project. Spring Roo is very flexible tool, it means that if we want to change something like, deleting, editing a file or removing the Roo it is simple, just do it.



Installing Roo

Roo as we already know is an open source tool and a standard Java application, everyone can download it on the Web: http://www.springsource.org/roo. First we download Spring Roo for the specific operating system but before we installed Spring Roo, we have to download and install

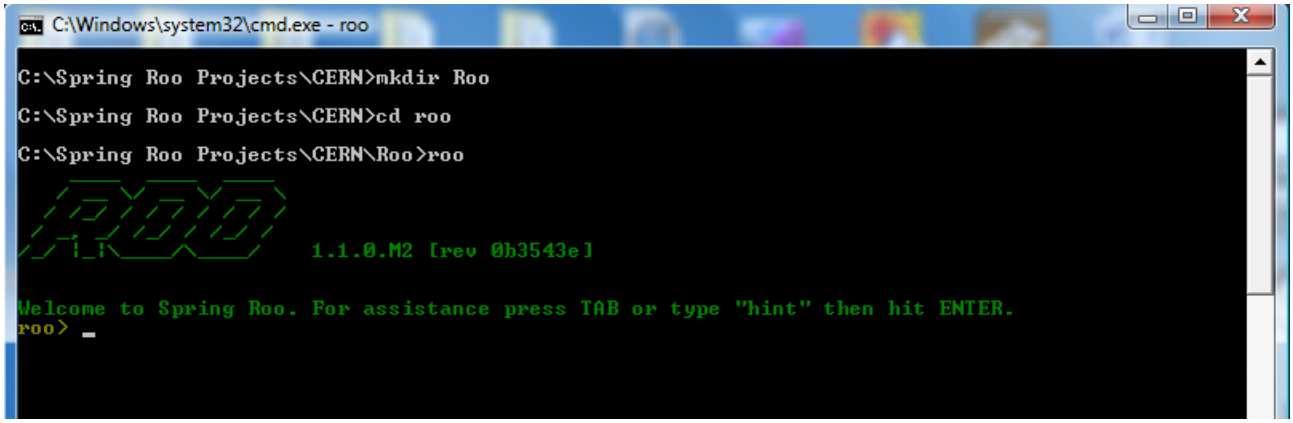
- Java 6 JKD
- Apache Maven

Roo requires JDK and Maven, it is recommended to download and install the latest versions. We also installed Eclipse because it is easy to import the code generated by Spring Roo.
Spring Roo is easy and simple to install, we just Unzip the Roo installation ZIP to a directory we choose.

Then:

- For Windows users: Windows, add $ROO_HOME\bin to your %PATH% environment variable
- For Linux or Apple: create a symbolic link using a command such as sudo ln -s $ROO_HOME/bin/roo.sh /usr/bin/roo

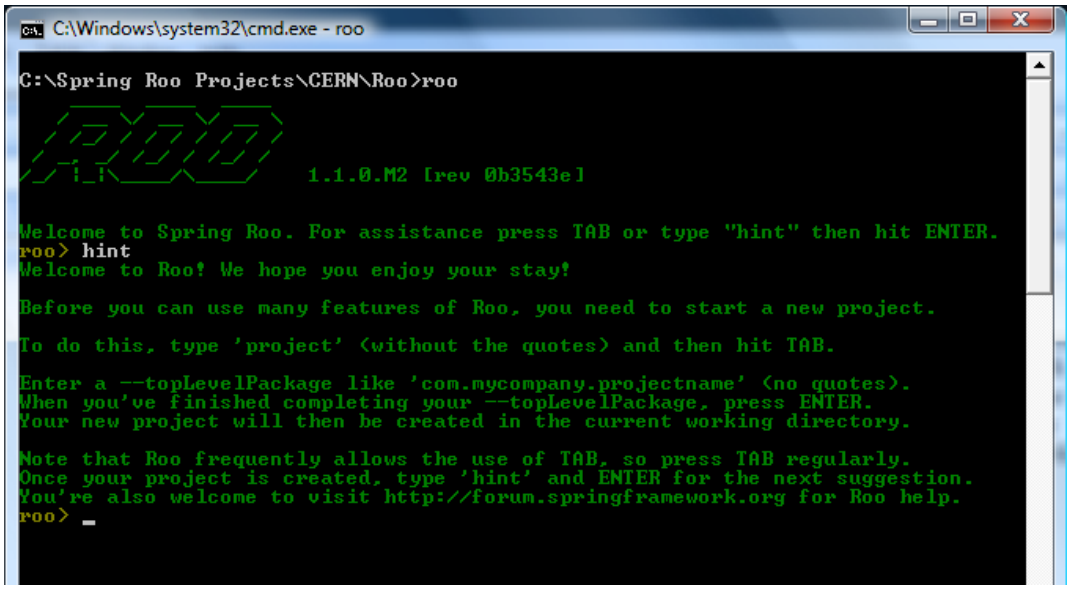
Spring Roo's main user interface is a command-line shell. Next we verified if the Roo has been installed correctly. To verify it we have to go to Windows command line cmd and type the following commands:



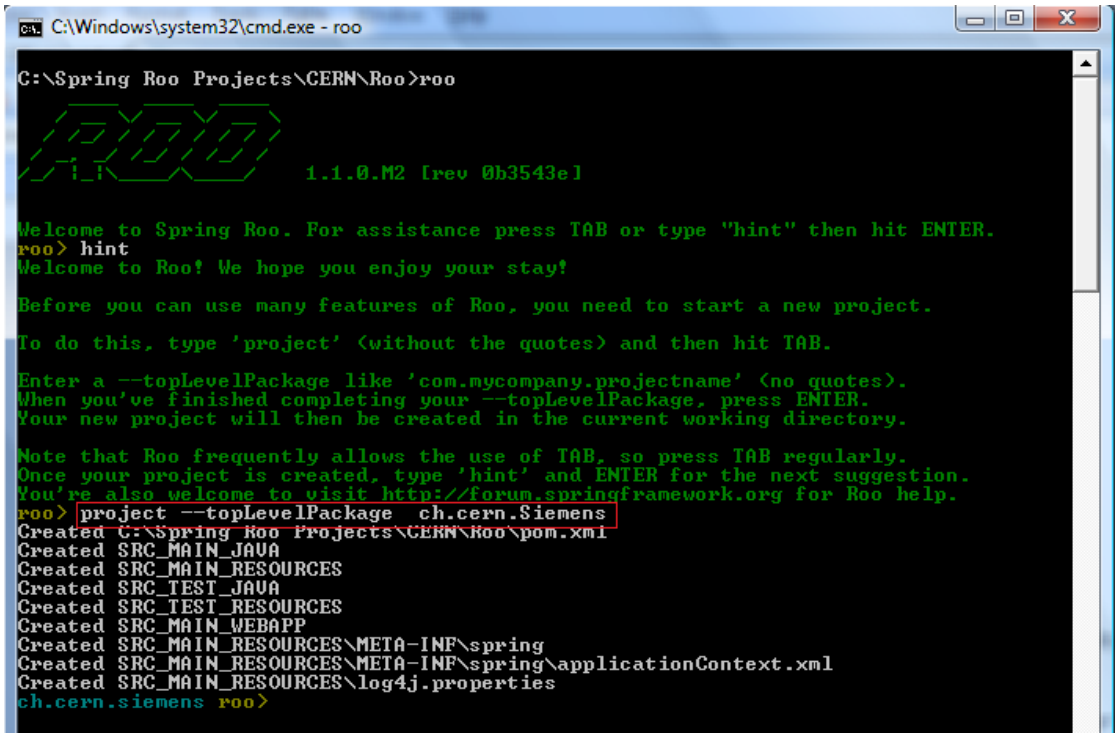
The logo of Roo shows that it has been installed correctly.

Spring Roo starting the project

Once we have installed Roo and test if it has been installed correctly, than we can start to create the database. First as we saw on the figure above we created a directory called “Roo” in which Spring Roo will store the projects. Spring roo provides very useful features: for examples the TAB key is a command line completion and the command "hint" provides information for the step-by-step commands we have to type. If we type “hint” and then ENTER we will see this:



The “hint” command describes the steps we have go through and what to do next. Now we simple start creating the project by typing this command on the Roo shell:



When we used the command *project* it creates a new project.

project --topLevelPackage ch.cern.Siemens

--topLevelPackage

The uppermost package name (this becomes the <groupId> in Maven and also the '~' value when using Roo's shell); no default value (mandatory) [roo]

--projectName

The name of the project (last segment of package name used as default); no default value [roo].

Once the project structure has been created by Roo we continue by installing a persistence configuration for our application. Roo uses the Java Persistence API (JPA) which provides an appropriate abstraction to achieve object-relational level. JPA takes care of mappings between our persistent domain objects (entities) and their underlying database tables. To install or change the persistence configuration in our project we can use the **persistence setup** command [Roo].

```
roo> project --topLevelPackage  ch.cern.Siemens
Created C:\Spring Roo Projects\CERN\Roo\pom.xml
Created SRC_MAIN_JAVA
Created SRC_MAIN_RESOURCES
Created SRC_TEST_JAVA
Created SRC_TEST_RESOURCES
Created SRC_MAIN_WEBAPP
Created SRC_MAIN_RESOURCES\META-INF\spring
Created SRC_MAIN_RESOURCES\META-INF\spring\applicationContext.xml
Created SRC_MAIN_RESOURCES\log4j.properties
ch.cern.siemens roo> persistence setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY
Managed SRC_MAIN_RESOURCES\META-INF\spring\applicationContext.xml
Created SRC_MAIN_RESOURCES\META-INF\persistence.xml
Created SRC_MAIN_RESOURCES\META-INF\spring\database.properties
Managed ROOT\pom.xml [Added dependency org.hsqldb:hsqldb:2.0.0]
Managed ROOT\pom.xml [Added dependency org.hibernate:hibernate-core:3.5.1-Final]
Managed ROOT\pom.xml [Added dependency org.hibernate:hibernate-entitymanager:3.5.1-Final]
Managed ROOT\pom.xml [Added dependency org.hibernate.javax.persistence:hibernate-jpa-2.0-api:1.0.0.F
Managed ROOT\pom.xml [Added dependency org.hibernate:hibernate-validator:4.0.2.GA]
Managed ROOT\pom.xml [Added dependency javax.validation:validation-api:1.0.0.GA]
Managed ROOT\pom.xml [Added dependency cglib:cglib-nodep:2.2]
Managed ROOT\pom.xml [Added dependency javax.transaction:jta:1.1]
Managed ROOT\pom.xml [Added dependency org.springframework:spring-jdbc:${spring.version}]
Managed ROOT\pom.xml [Added dependency org.springframework:spring-orm:${spring.version}]
Managed ROOT\pom.xml [Added dependency commons-pool:commons-pool:1.5.4]
Managed ROOT\pom.xml [Added dependency commons-dbcp:commons-dbcp:1.3]
Managed ROOT\pom.xml
ch.cern.siemens roo>
```

As we ca see in our project we have used Hibernate as object-relational mapping (ORM)-provider. This is one of the three providers that Roo offers. We have chosen the Hypersonic in-memory database for testing purposes. Hibernate supports these databases in Roo:

```
roo> persistence setup --provider HIBERNATE --database

DB2                    DERBY                  GOOGLE_APP_ENGINE      H2_IN_MEMORY
HYPERSONIC_IN_MEMORY   HYPERSONIC_PERSISTENT  MSSQL                  MYSQL
ORACLE                 POSTGRES               SYBASE
```

Creating Entities and Fields

After we have created the project it is time to start creating the domain entities and fields form the E-R diagram. To create entities in Roo we can use *entity* command. One of the optional required attribute of entity command is `--class` and we have used also the attribute `-testAutomatically` which creates integrated tests for the entity.

```
roo> entity --class ~.domain.PLCTesting --testAutomatically
Created SRC_MAIN_JAVA\ch\cern\siemens\domain
Created SRC_MAIN_JAVA\ch\cern\siemens\domain\PLCTesting.java
Created SRC_TEST_JAVA\ch\cern\siemens\domain
Created SRC_TEST_JAVA\ch\cern\siemens\domain\PLCTestingDataOnDemand.java
Created SRC_TEST_JAVA\ch\cern\siemens\domain\PLCTestingIntegrationTest.java
Created SRC_MAIN_JAVA\ch\cern\siemens\domain\PLCTesting_Roo_Entity.aj
Created SRC_MAIN_JAVA\ch\cern\siemens\domain\PLCTesting_Roo_ToString.aj
Created SRC_MAIN_JAVA\ch\cern\siemens\domain\PLCTesting_Roo_Configurable.aj
Created SRC_TEST_JAVA\ch\cern\siemens\domain\PLCTestingIntegrationTest_Roo_Configurable.aj
Created SRC_TEST_JAVA\ch\cern\siemens\domain\PLCTestingDataOnDemand_Roo_DataOnDemand.aj
Created SRC_TEST_JAVA\ch\cern\siemens\domain\PLCTestingIntegrationTest_Roo_IntegrationTest.aj
Created SRC_TEST_JAVA\ch\cern\siemens\domain\PLCTestingDataOnDemand_Roo_Configurable.aj
~.domain.PLCTesting roo>
```

In this case we have created the entity called PLCTesting, this is the same table like in E-R diagram for Tests, but Roo does not allow using the word “Test” as a name for entity. Now that we have created the table we can continue by adding the fields in the table by typing the command *field* and the attributes for this field. After the command field we have to choose the data type for that field.

```
~.domain.PLCTesting roo> field

field boolean          field date             field email template   field enum
field number           field other            field reference        field set
field string
```

Roo offers this data types. The commands *Field set* and *Field reference* are used to create the relationships between tables. Based on the Spring Roo reference with *Field set* and *Field reference* we can define these relationships:

field reference

- Adds a private reference field to an existing Java source file (eg the 'many' side of a many-to-one)

field reference --fieldName –type

--fieldName
- The name of the field to add; no default value (mandatory)

--type
- The Java type of the entity to reference; no default value (mandatory)

--class
- The name of the class to receive this field; default if option not present: '*'

field set

- Adds a private Set field to an existing Java source file (eg the 'one' side of a many-to-one)

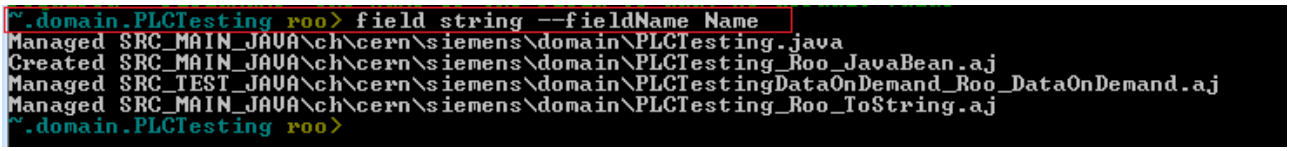
field set --fieldName --element

--fieldName
- The name of the field to add; no default value (mandatory)

--element
- The entity which will be contained within the Set; no default value (mandatory)

--class
- The name of the class to receive this field; default if option not present: '*'

With the command shows in the figure below we have created the field called “Name” and the data type for this field is *string*.



In same way we have created all the fields for our database. As we have seen our database contains some *enum fields*. How we created the enum fields? It is simple just write:

enum type --class ch.cern.Siemens.ImpactEnum

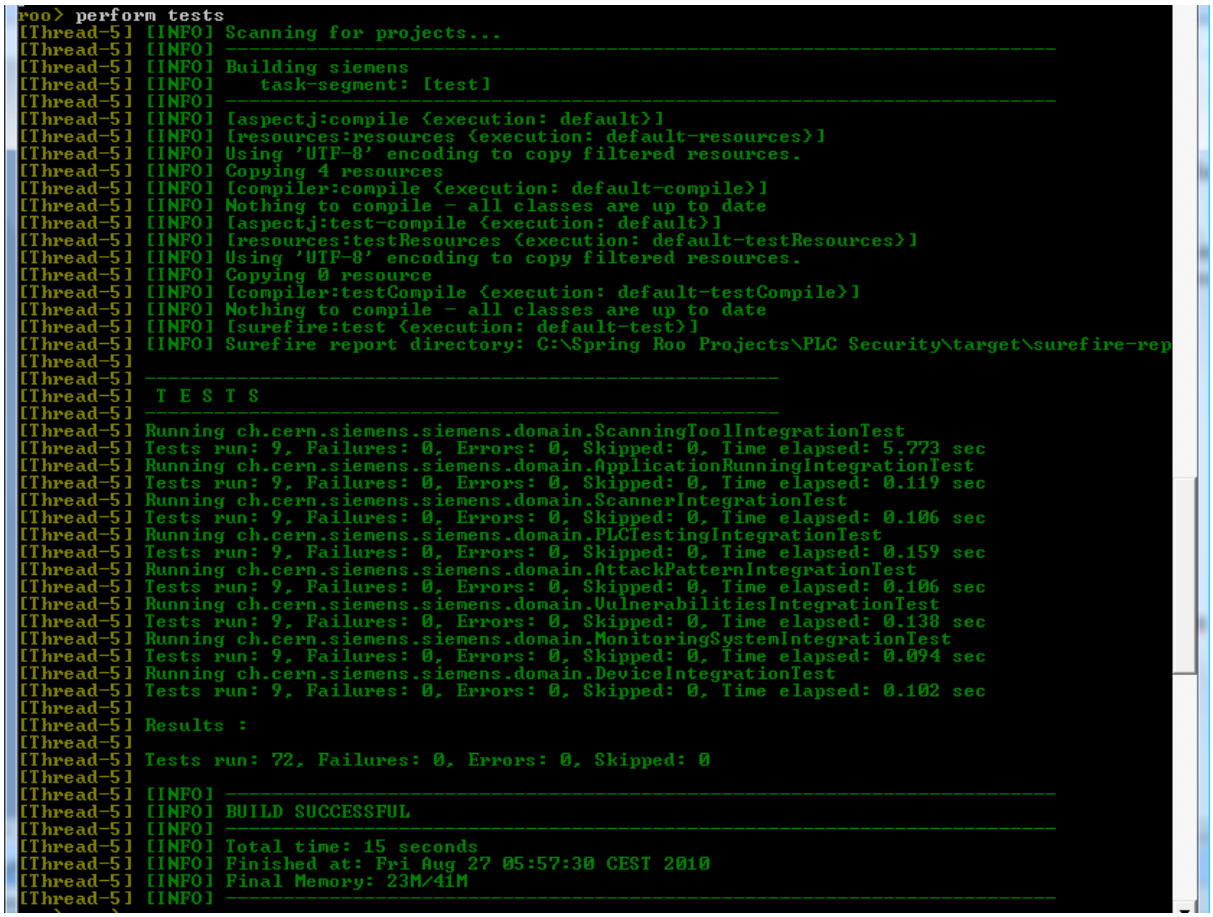
enum constant --name Value 1
enum constant --name Value 2

ImpactEnum is the name of the enum which is created on the topLevel domain, and the following commands added the Value 1 and Value 2 as members of enum.

Also in database we have a field in the table Vulnerabilities which is Blob type. Roo dose not support Blob data type, but the solution for this is that we can change the data type after we will import our application to Eclipse by adding these commands:

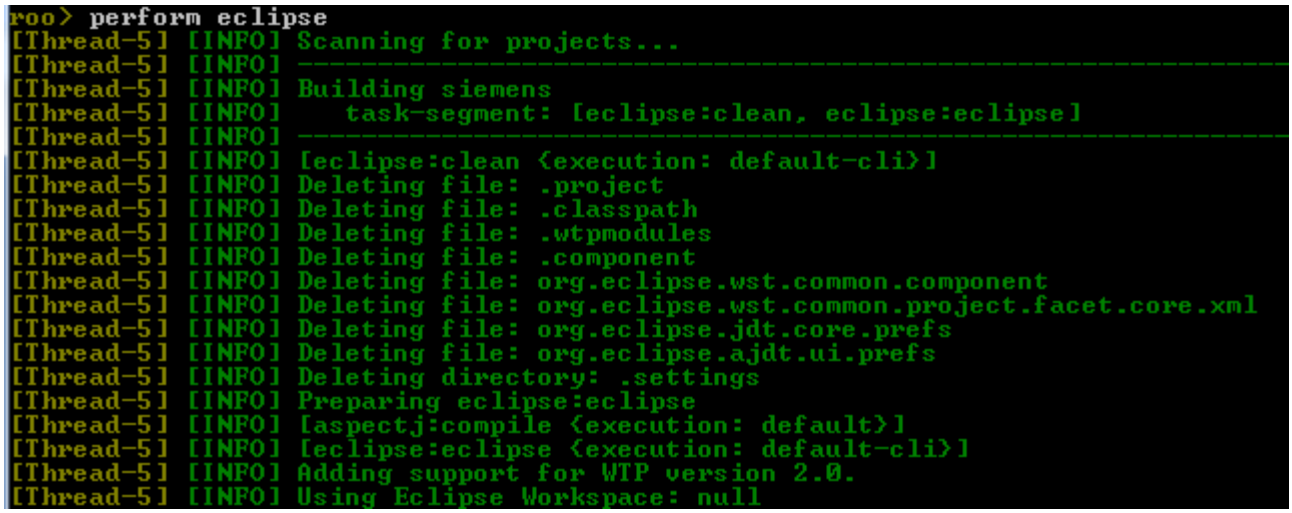
```
public class Vulnerabilities {


    @Lob
    @Column(name = "PacketCapture")
    private byte PacketCapture;

}
```

Finally after we created the project, database, all entities and fields we instruct Roo to perfom some integration tests by typing the command *perfom tests*. As we can see in the figure below Roo has issued a Maven command (equivalent to running '`mvn test`' outside the Roo shell) in order to execute the integration tests. All tests have passed, Roo has generated 9 integration tests per domain object resulting in a total of 36 integration tests for all 4 domain objects.



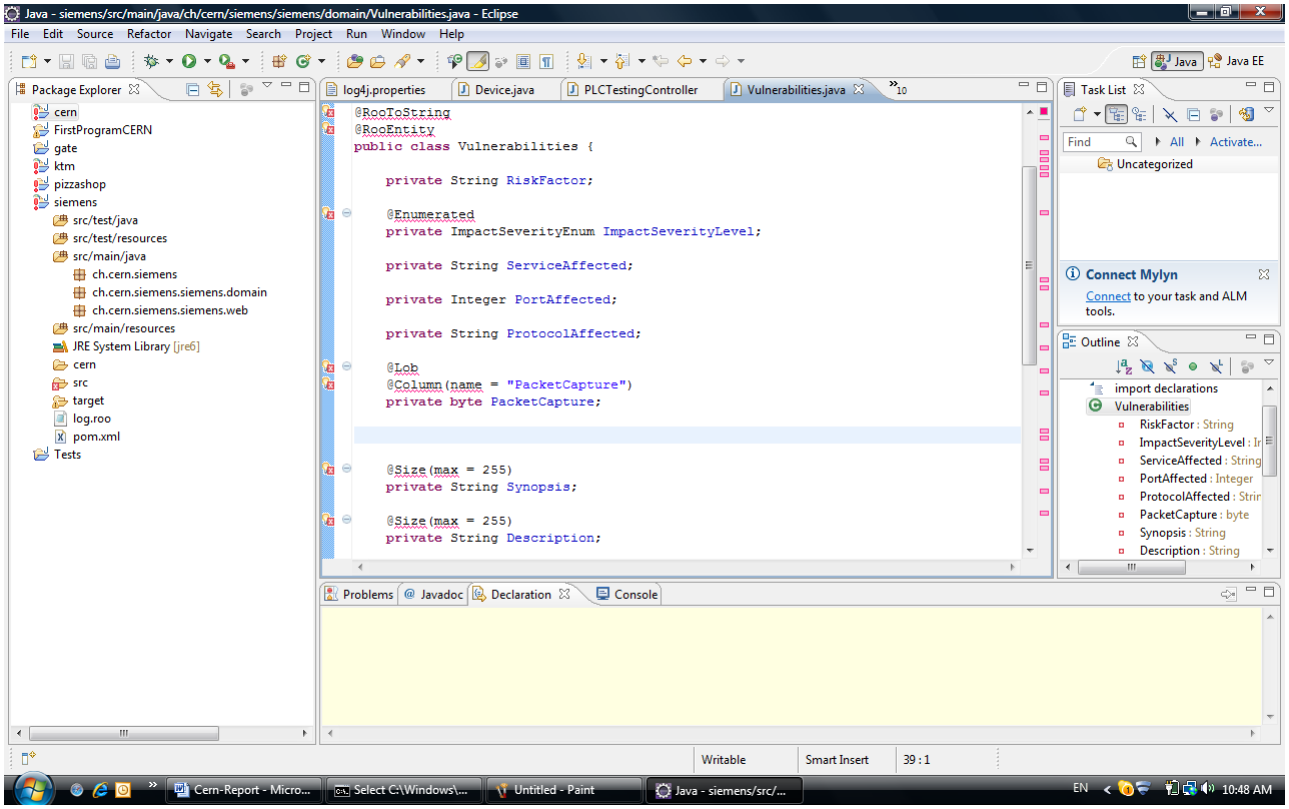
Using the IDE

Roo projects can be used in different IDE. All the Roo annotations start with the symbol @ and the word Roo like this *@Roo*. SpringSource offers a tool which is free of charge called SpingSource Tool Suite (STS), but as an IDE for our project we have chose Eclipse. Before importing the project we have to type the command *perform eclipse.*



```
[Thread-5] [INFO] ------------------------------------------------------------------------
[Thread-5] [INFO] BUILD SUCCESSFUL
[Thread-5] [INFO] ------------------------------------------------------------------------
[Thread-5] [INFO] Total time: 8 seconds
[Thread-5] [INFO] Finished at: Fri Aug 27 10:37:52 CEST 2010
[Thread-5] [INFO] Final Memory: 21M/38M
[Thread-5] [INFO] ------------------------------------------------------------------------
roo> roo>
```

As the next step after performing the eclipse command we can go to Eclipse and select File> Import> General > Existing Projects into Workspace; then we select the directory of our project and click Finish.



For example if we go to the table PLCTesting we will see this code:

```
import javax.persistence.Entity;
import org.springframework.roo.addon.javabean.RooJavaBean;
import org.springframework.roo.addon.tostring.RooToString;
import org.springframework.roo.addon.entity.RooEntity;
import java.util.Date;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import org.springframework.format.annotation.DateTimeFormat;
import ch.cern.siemens.TerminationStatusEnum;
import javax.persistence.Enumerated;
```

```
import ch.cern.siemens.siemens.domain.Device;
import javax.persistence.ManyToOne;
import javax.persistence.JoinColumn;
import ch.cern.siemens.siemens.domain.ConfigurationIO;
import ch.cern.siemens.siemens.domain.ApplicationRunning;
import java.util.Set;
import javax.persistence.ManyToMany;
import javax.persistence.CascadeType;

@Entity
@RooJavaBean
@RooToString
@RooEntity
public class PLCTesting {

    private String Name;

    @Temporal(TemporalType.TIMESTAMP)
    @DateTimeFormat(style = "S-")
    private Date TimeStrat;

    @Temporal(TemporalType.TIMESTAMP)
    @DateTimeFormat(style = "S-")
    private Date TimeEnd;

    @Enumerated
    private TerminationStatusEnum TerminationStatus;

    private Double CommunicationLoad;

    @ManyToOne(targetEntity = Device.class)
    @JoinColumn
    private Device Device;

    @ManyToOne(targetEntity = ConfigurationIO.class)
    @JoinColumn
    private ConfigurationIO ConfigurationIO;

    @ManyToOne(targetEntity = ApplicationRunning.class)
    @JoinColumn
    private ApplicationRunning ApplicationRunning;

    @ManyToMany(cascade = CascadeType.ALL)
    private               Set<ch.cern.siemens.siemens.domain.ConnectionInfo>
ConnectionInfo                                =                                  new
java.util.HashSet<ch.cern.siemens.siemens.domain.ConnectionInfo>();

    @ManyToMany(cascade = CascadeType.ALL)
    private              Set<ch.cern.siemens.siemens.domain.Vulnerabilities>
Vulnerability                                 =                                  new
java.util.HashSet<ch.cern.siemens.siemens.domain.Vulnerabilities>();
}
```

The code above describes each field of the PLCTesting table, and the fileds that are used as relationships with other tables. Every change we do on the eclipse - like adding or removing fields or changing data type- will be detected changed automatically.

Creating the web tier and loading the web server

As a next step for the project it is to scaffold a Web tier for our application. This can be accomplished via the **controller** command. The most appropriate way to generate controllers and all relevant Web artefacts is to use the **controller all** command:

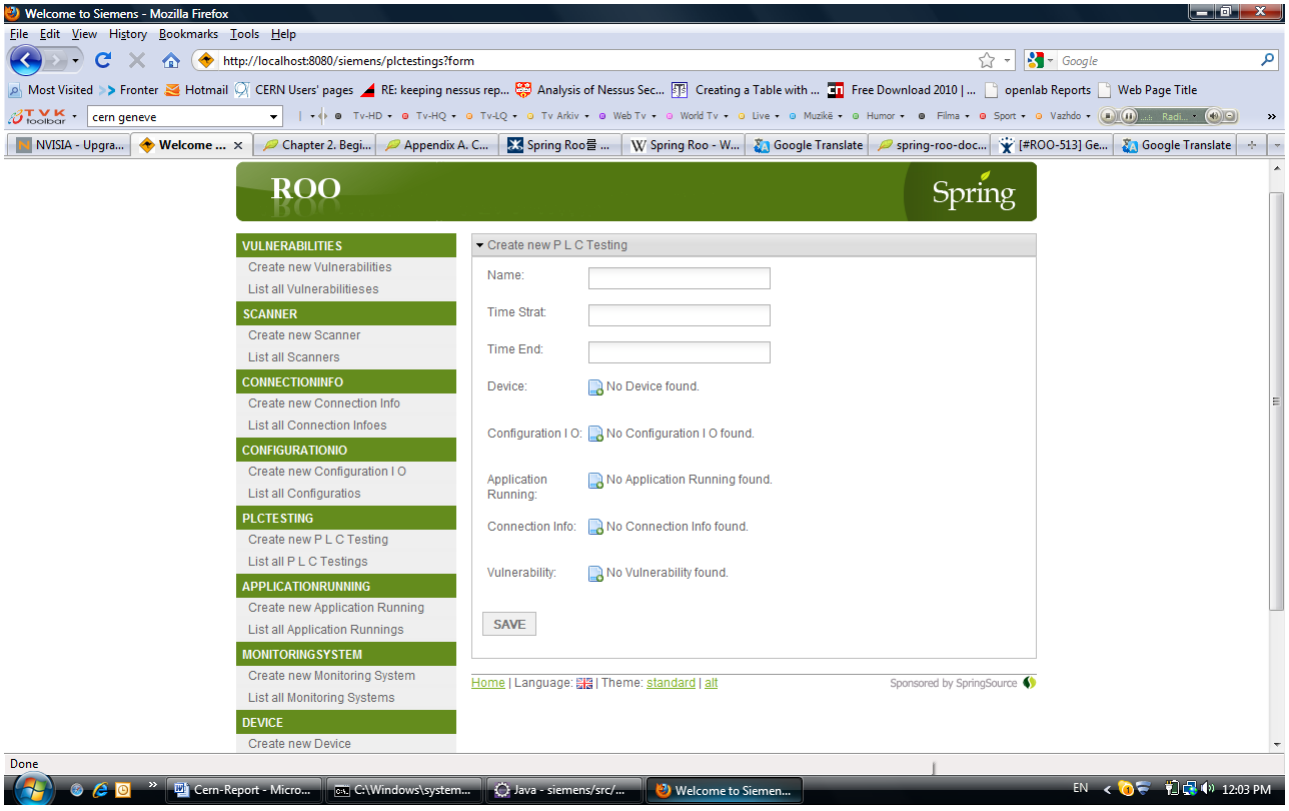
*roo> **controller all --package ~.web***

This command will scan our project for any domain entities, fields and scaffold a Spring MVC controller for each entity detected.

To deploy our application in a Web container we use the command below in the root of our project to start the Tomcat MVC front-end.

mvn tomcat:run

After we performed all the commands above, to see the Web container go to the following URL http://localhost:8080/siemens.



One way to find the data in the database with Sping Roo is through some *finders* that search for the information in the table and create a list of these information. We can perform a search Vulnerabilities' table with this command:

finder add –finderName FindVulnerabilitisByRiskFactor –class ~.domain.Vulnerabilities

Another way to query the database is by creating methods in java that will query the database.

Conclusion

Considering that the number of vulnerabilities is increasing, the data that security analyzer provide us are very important. The reports structure produced by the current security analyzers is difficult to read and manage. We consider that our database will make it much easier and in a more proficient way. The database we have defined is very flexible and the integrated Hibernate product is an open source and offers a wide choice of database management systems. We consider that this application will be very useful for the future of the TRoIE project.

Bibliography

[1] Spring Roo - Reference Documentation

[2] Rogers. Russ, Nessus Network Auditing, Second Edition, May 2008

[3] Rob. Peter. , Coronel. Carlos, Database Systems: Design, Implementation, and Management

[4] http://blogs.iss.net/archive/2007XFReport-Day1.html

[5] http://en.wikipedia.org/wiki/Database

[6] http://searchsqlserver.techtarget.com/definition/database

[7] http://www.springsource.org/roo

[8] ISA Security Compliance Institute- Embedded Device Security Assurance

Appendix: Database application code

```
project --topLevelPackage  ch.cern.Siemens

persistence setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY

enum type --class ch.cern.Siemens.TerminationStatusEnum
enum constant --name Val1
enum constant --name Val2

enum type --class ch.cern.Siemens.ImpactSeverityEnum
enum constant --name Val1
enum constant --name Val2
enum constant --name Val3

enum type --class ch.cern.Siemens.ActivePassiveEnum
enum constant --name Val1
enum constant --name Val2

enum type --class ch.cern.Siemens.TypeOfCommunicatioEnum
enum constant --name Val1
enum constant --name Val2

enum type --class ch.cern.Siemens.InstructionSetEnum
enum constant --name Val1
enum constant --name Val2

enum type --class ch.cern.Siemens.ListOfBlockEnum
enum constant --name Val1
enum constant --name Val2

entity --class ~.domain.PLCTesting --testAutomatically
field string --fieldName Name
field date --fieldName TimeStrat --type java.util.date --timeFormat
field date --fieldName TimeEnd --type java.util.date --timeFormat
field enum --fieldName TerminationStatus --type
ch.cern.Siemens.TerminationStatusEnum
field number --fieldName CommunicationLoad --type java.lang.Double

entity --class ~.domain.Vulnerabilities --testAutomatically
field string --fieldName RiskFactor
field enum --fieldName ImpactSeverityLevel --type
ch.cern.Siemens.ImpactSeverityEnum
field string --fieldName ServiceAffected
field number --fieldName PortAffected --type java.lang.Integer
field string --fieldName ProtocolAffected
field string --fieldName Synopsis --sizeMax 255
field string --fieldName Description --sizeMax 255
field string --fieldName PossibleSolution --sizeMax 255
field string --fieldName PluginOutput --sizeMax 255
```

```
field string --fieldName CVE



entity --class ~.domain.Device --testAutomatically
field string --fieldName Name
field string --fieldName Type
field string --fieldName Manufacture
field string --fieldName OrderNumber
field string --fieldName SerialNumber
field string --fieldName FirmwareVersion
field string --fieldName OperatingSystem

entity --class ~.domain.ConnectionInfo
field string --fieldName LocalPortOpen
field string --fieldName RemotePortOpen
field string --fieldName IpOfTarget
field string --fieldName IpOfPartner
field number --fieldName DevicePartnerID --type java.lang.Integer
field number --fieldName BitRateInput --type java.lang.Double
field number --fieldName BitRateOutput --type java.lang.Double
field enum --fieldName ActivePassive --type
ch.cern.Siemens.ActivePassiveEnum

entity --class ~.domain.MonitoringSystem --testAutomatically
field string --fieldName TypeOfMonitoring
field string --fieldName Description --sizeMax 255

entity --class ~.domain.Scanner --testAutomatically
field string --fieldName Name
field string --fieldName ScannerVersion
field string --fieldName Description --sizeMax 255

entity --class ~.domain.ScanningTool --testAutomatically
field number --fieldName ToolID --type java.lang.Integer
field string --fieldName ToolName
field string --fieldName ToolFamily
field string --fieldName ToolVersion
field string --fieldName Description --sizeMax 255

entity --class ~.domain.ApplicationRunning --testAutomatically
field date --fieldName TimeCycle --type java.util.Date --timeFormat
field string --fieldName Protection
field string --fieldName Description --sizeMax 255
field string --fieldName StartupMode
field number --fieldName CpuUsage --type java.lang.Double
field enum --fieldName InstructionSetUse --type
ch.cern.Siemens.InstructionSetEnum
field enum --fieldName ListOfBlockTypeInExecution --type
ch.cern.Siemens.ListOfBlockEnum

entity --class ~.domain.ConfigurationIO
```

```
field number --fieldName NumberOfUsedInput --type java.lang.Integer
field number --fieldName NumberOfUsedOutput --type java.lang.Integer
field number --fieldName InputSignalFrequency --type java.lang.Double
field number --fieldName OutputSignalFrequency --type java.lang.Double

entity --class ~.domain.AttackPattern --testAutomatically
field string --fieldName Name

field string --fieldName Description --sizeMax 255

field reference --fieldName PartnerID --type ~.domain.AttackPattern

field reference --class ~.domain.Device --fieldName Device --type
~.domain.PLCTesting

field reference --class ~.domain.PLCTesting --fieldName Device --type
~.domain.Device

field reference --class ~.domain.PLCTesting --fieldName ConfigurationIO
--type ~.domain.ConfigurationIO

field reference --class ~.domain.PLCTesting --fieldName
ApplicationRunning --type ~.domain.ApplicationRunning

field set --element ~.domain.ConnectionInfo --fieldName ConnectionInfo -
-class ~.domain.PLCTesting --cardinality MANY_TO_MANY

field set --element ~.domain.Vulnerabilities --fieldName Vulnerability -
-class ~.domain.PLCTesting --cardinality MANY_TO_MANY

field reference --class ~.domain.Vulnerabilities --fieldName Attack --
type ~.domain.AttackPattern

field reference --class ~.domain.Vulnerabilities --fieldName
ScanningTool --type ~.domain.ScanningTool

field set --element ~.domain.MonitoringSystem --fieldName
MonitoringSystem --class ~.domain.Vulnerabilities --cardinality
MANY_TO_MANY

field set --element ~.domain.PLCTesting --fieldName PlcTesting --class
~.domain.Vulnerabilities --cardinality MANY_TO_MANY

field reference --class ~.domain.ScanningTool --fieldName Scanner --type
~.domain.Scanner
perform
```