Using Design Structure Matrices (DSM) as security controls for software architectures

Pierre Parrend, Timothé Mazzucotelli, Florent Colin

RESEARCH REPORT N°1

Saturday, 20th May, 2017

4P-Factory E-Laboratory: the Factory of the Future

United Nations
Educational, Scientific and
Cultural Organization

COMPLEX SYSTEMS
DIGITAL CAMPUS

ARK UNITWIN-CS.ORG/ARK:/69427/03

Laboratoire des sciences de l'ingénieur
de l'informatique et de l'imagerie

UNIVERSITÉ DE STRASBOURG

Using Design Structure Matrices (DSM) as security controls for software architectures

Pierre Parrend[1] Timothé Mazzucotelli[2] Florent Colin[3]

Saturday, 20th May, 2017

[1]ECAM Strasbourg-Europe, ICube Laboratory Strasbourg, Complex System Digital Campus
[2]IGBMC, Illkirch-Graffenstaden
[3]IGBMC, Illkirch-Graffenstaden

Abstract

Building secure software is often seen as a task mainly focused on development and penetration testing. However, it requires ensuring that the system embeds robust architecture principles, on which security features and proper code can rely. Few solutions for creating and monitoring such architectures exist, and those existing are dedicated to mission- and life-critical systems. Mainstream technologies, such as web platforms, host ever increasing critical application and data, and need suitable tools for enforcing relevant architecture-level monitoring. We propose to apply the Design Structure Matrix (DSM) model to represent and analyze the structure of complex applications. DSM is both an efficient analysis tool and a convenient tool for visualising complex systems. It supports the translation of software architectures into graphs, which prove to be efficient tools for structural analysis. Guidelines for secure architectures are expressed, first qualitatively, then in a quantitative manner, as constraints on these graphs. The Archan tool for supporting DSM-based architecture monitoring is presented. Our approach is illustrated for pedagogical stakes using two toy examples, and validated on a middle-scale project for managing sensitive medical data. Archan thus enforces sound architectural principles, which are a pre-requisite for building secure systems.

Key-words

Design Structure Matrices; Security Information Systems Architecture; Software Security; Secure Software Development Methodologies; Security and privacy in Complex Systems; Security Metrics and Measurement

Contents

List of Figures

1 Introduction

Security monitoring is a key process of Security Operations, *i.e.* the day-to-day enforcement of security at the system level by administrators of systems and applications. However, very few tools are available beyond user creation and database-level grant management. This lets the designer or developer of systems and applications left to his own development tools, experience, intuition, and will. In a world where applications are ever more heterogeneous - running on servers, PCs, smartphone, and now every kind of connected fridge or watch - ever more connected, and ever more evolutive, failing to provide the development community with efficient, simple to use solutions for ensuring security of software architectures will inevitably lead to major data leaks or targeted thefts.

To be fair, secure software architecture are ensured by two complementary elements: hardened system design, and secure components. Secure components, on the first hand, require a never-relaxing attention to potential threats and actual attacks against the platform. The use of professional, wide-spread environments having a sustainable team in charge of their evolution is a key issue here. It provides a reasonable expectation that security issues will be corrected and communicated quickly when and if they occur. However, their is no guarantee: for instance, two majors service frameworks, Apache CXF and the Spring web framework, have been shown in the past years to embed major vulnerabilities. One can guess than some of the 22 yearly millions downloads from these software may have led to actual damage in the vulnerable systems. Nonetheless, in the case of the medical application used for evaluation in this paper, the framework, in this case Django, prove to bring a quite complete protection against mainstream security issues.

Hardened system design, on the other hand, requires to be able to provide specific architecture analysis for each individual application. So far, very few concrete tools are available. If security monitoring tools are scarce, monitoring tools at the architecture level are still in their infancy. Architectural design tools are too often conceptual patterns or blueprints focusing on requirement elicitation rather than on system building blocks: Microsoft[1], the OWASP[2] (Open Web Application Security Project), IBM[3] all propose processes and checklists for building secure architecture. Very few propose to automatize these processes for tailored made solutions. One of the main reason for this is of course the high heterogeneity and dynamicity of the systems.

We therefore claims that a control tool capable of supporting the design, management and monitoring of complex, heterogeneous, and dynamic systems is required for building secure architecture. One such tool is the Design Structure Matrix (DSM) model [Yas04], which originates in the manufacturing and product development community, and already proved to be capable of giving accurate insight into software architecture [MRB06]. DSMs visualise the dependencies between software modules, and make formalization of architectural properties a straightforward thing. By expressing the core properties of secure computing systems as given by Saltzer and

[1] https://msdn.microsoft.com/en-us/library/ee658084.aspx

[2] https://www.owasp.org/index.php/Application_Security _Architecture_Cheat_Sheet

[3] https://www.opengroup.org/architecture/0310wash/ presents/Jim_Whitmore-Enterprise_Security_Architceture.pdf

Schroeder [SS75], they prove to bring rapid feedback on the actual security status of systems, thus supporting corresponding rapid improvement of the overall security status. The DSM-based process for architecture monitoring is defined and evaluated in this paper. The Archan (Architecture Analysis) tool implements, in the context of a Django application, this process.

The challenges addressed by Archan are therefore the following ones:

- Standardised security controls are not defined for web application frameworks, beyond widely available login and database features
- Technical controls for ensuring architecture level software security issues are only scarcely available
- Software suite suffer from a very poor auditability and monitorability support in the domain of security issues

Section 2 discusses the role played by secure software architectures in IT security. Section 3 introduces the concept of Design Structure Matrix (DSM) and identify their potential benefit for security controls. Section 4 provides a qualitative characterization of Saltzer and Schroeder security principles using DSMs. Section 5 presents the Archan tool we developed for performing this analysis, and section 6 defines metrics for quantitative evaluation of security principles for secure architectures. Section 7 evaluates the proposed approach of DSM-based monitoring of software architectures, based on a real-life application, and section 8 discusses its relevance with regards to the expected benefits. Section 9 concludes this work.

Enforcing secure software architectures is the cornerstone of building sustainably secure software systems. For this reason, it builds a core element of supervision as well as system security properties.

2.1 Information security, Software security

Architecture is a pre-requisite for information security: numerous maurity models for security assurance, such as the OpenSAMM[1] or the 'Building Security In Maturity Model'[2] (BSI-MM) clearly states its importance for sound development of secure architecture.

Figure 2.1 shows the 7 touchpoints for software security, which highlight key security activities in the lifecycle of software [McG06], which can be considered as the 'developer's view' on the BSI-MM process. Architecture builds the first brick of technical analysis, in particular through architectural risk analysis [PHL]. Security patterns [Hal+08] are a tool of choice here.

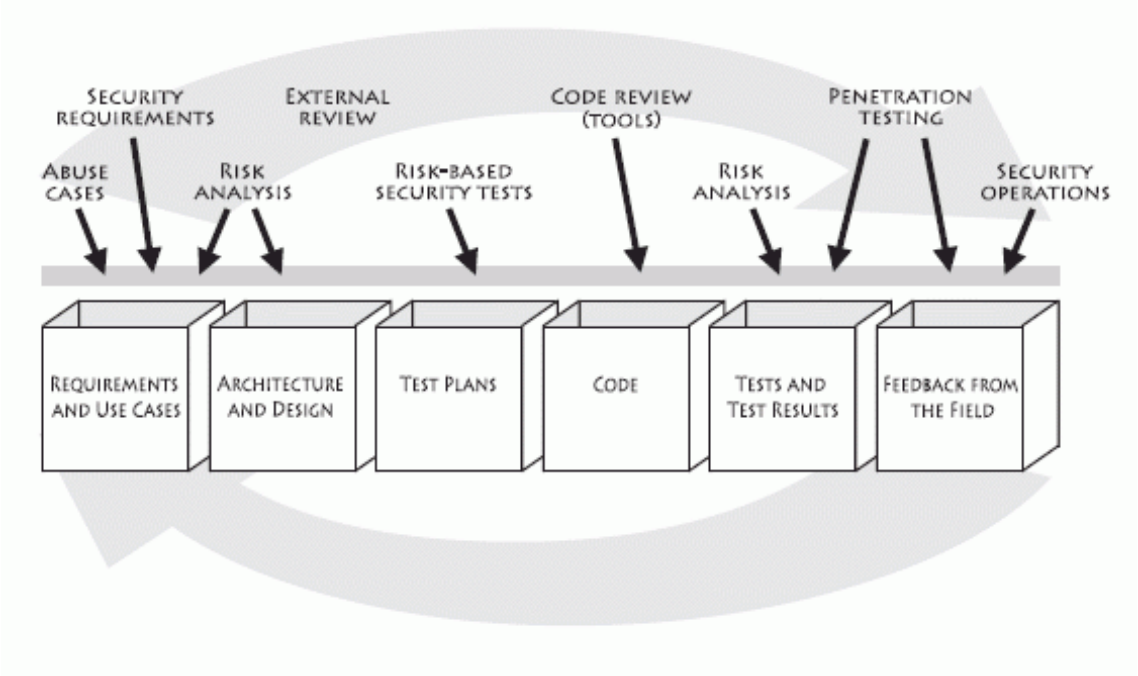

Figure 2.1: The 7 touchpoints for software security throughout the development life-cycle

However, a common critic to security patterns is their lack of pragmatism when it comes to letting an application actually run. In particular, it is often very hard to ensure that these patterns, when carefully implemented in a software platform, are complied with on the long run throughout the evolution of the application. This matter of fact leads us to consider that actual technical controls

[1] http://www.opensamm.org/
[2] https://www.bsimm.com/

build longer term solutions to software security, provided that they are monitorable, *i.e.* they come with a suitable visualisation environment.

2.2 Software architecture in security analyses

Whereas software architectures are often neglected in formal validation processes such as the Common Criteria [Her02], they are identified as a prerequisite in technical processes such as the 'Application Security Verification Standard'[3] (ASVS), which is defined by the Open Web Application Security Project[4] (OWASP), the Open Security Assessment Maturity Model[5] (OpenSAMM), as well as the 'Building Security in Maturity Model'[6] (BSIMM).

The ASVS provide numerous resources for performing qualitative security analysis of the security of software architectures: an Application Security Architecture Cheat Sheet[7], as well as guidelines for compliance with level 1[8] and level 2[9]. These recommendations are useful best practices, but miss both a systematic technological approach and an academic evaluation.

The OpenSAMM identifies 'Secure Architectures' as one of its 12 core activities, in the 'construction' domain. It defines three levels of maturity: 'Insert consideration of proactive security guidance into the software design process'; 'Direct the software design process toward known-secure services and secure-by-default designs'; 'Formally control the software design process and validate the utilisation of secure components'.

The BSIMM has a similar treatment of what it calls 'Architecture analysis' (AA), as one if its 9 core activities. The three levels are identified as 'Get started with AA'; 'Model objects'; 'Build capabilities organization-wide'.

These approaches keep being technology-agnostic, which prevent them to recommend specific actual solution, and limit their scope at usefull but abstract best practices.

2.3 State of the art properties of secure architectures

Even though numerous architectural solutions have been proposed in various domain of computing (networks, computing machines, middlewares, and so on), very few academic work deals with the subject of characterizing what actually is a secure architecture. The most prominent reference is this domain is the work done by Saltzer and Schroeder on the Multics systems, which defines the core principles for protecting information in computer systems [SS75]. These principles are:

[3] https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

[4] https://www.owasp.org/

[5] http://www.opensamm.org/

[6] http://www.bsimm.com/

[7] https://www.owasp.org/index.php/Application_Security_Architecture_Cheat_Sheet

[8] https://www.owasp.org/index.php/How_to_perform_a_security_architecture_review_at_Level_1

[9] https://www.owasp.org/index.php/How_to_perform_a_security_architecture_review_at_Level_2

- **Economy of mechanism** (1): Keep the design as simple and small as possible
- **Fail-safe defaults** (2): Base access decisions on permission (white-list) rather than exclusion (black-list)
- **Complete mediation** (3): Every access to every object must be checked for authority
- **Open design** (4): The design should not be secret
- **Separation of privileges** (5): where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key
- **Least privilege** (6): Every program and every user of the system should operate using the least set of privileges necessary to complete the job
- **Least common mechanism** (7): Minimize the amount of mechanism common to more than one user and depended on by all users
- **Psychological acceptability** (8): It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

These principles reach far beyond architecture issues, and can be classified in three categories: architecture (1,3,4,7); implementation issues for authorizations (2,5,6); ergonomy (8). We will focus on architectural issues in this paper.

The authors also mention following best practices, even though they do not consider them as core security properties:

- **Work factor**
- **Compromise recording**

To adapt the reference principles, which date back to 1975, to the evolution of networks and systems, Saltzer and Schroeder extend their principle list with two kind of principles: security principles, and non-security specific, software architecture principles.

Extended security principles by the authors entail [SK09]:

- **Minimize secret**

Non-security specific principles advocated by the authors entail [SK09]:

- **Least astonishment** : rewriting of 'psychological acceptability'
- **Adopt sweeping simplifications**
- **Design for iteration** : for incorporating continuous improvement as a design principle

The 'Least astonishment' principle is common to security and ergonomy issues. The two latter principles shall not be considered for security analysis, since they are more focused on system design than on system security.

A second set of principles devised at the beginning of the 2000's focus on the implementation of security protections rather than on overall design issues [PP02]:

- **Easiest penetration**
- **Weakest link**
- **Adequate protection**
- **Effectiveness**

The first two items address the 'work factor' concern.

Smith provides an extensive review and discussion of computer security principles [Smi12],

and proposes a focus on 8 security principles as defined in [Smi15]:

- **Continuous improvement**
- **Least privilege**
- **Defense in depth**
- **Open design**
- **Chain of control**
- **Deny by default**
- **Transitive trust**
- **Separation of duty**
- **Trust, but verify** : a 9th statement
- **'minimize secrets', 'least astonishment'** : recommended by the author to complete the list of state of the art principle

The author consider that the principles of: 'Least privilege', 'open design', 'fail-safe defaults', 'separation of privilege', and 'least common mechanism' are still relevant, but formulate critics against several of the original 7 Saltzer and Schroeder principles [Smi12], namely:

- **Simplicity** : the author consider that simplicity is not a security property,
- **Complete mediation** : the author consider that complete mediation is of no relevance in a distributed world,
- **Psychological acceptability** : the author consider that simplicity is not a security property.

The effort of the author for focusing on implementable principles, which can each be bound with specific tools, is valuable to the community. It highlights the difference between security functions, which they include in the principles, and design issues, which they consider not to be relevant for the creation of secure systems.

Nonetheless, the considered restriction can be very quickly questionable: for instance, though complete mediation has necessarily a different implementation in a mainframe environment and in distributed systems, the concept is far from being obsolete. The Active Directory, which can be considered as a de facto standard for access control in corporate environments, enforces complete mediation over all resources in an organisation, through centralised control of logins and dynamic GPO group policies. The GPOs control user, file system, applications or network services access. Complete mediation on mobile devices set the user as ower of the device resources, by systematically performing requests in case of modification of data access or application execution rights. Cryptographic certificates and Certificate Revocation Lists provide another example of fully mediated process, in this case for ensuring the traceability of identities of certificate owners. These three examples show that, even though the concept has strongly evolved since the 1970's, complete mediation continues to be a strong standard in various domains of the IT field.

Simplicity and psychological acceptability will not be discussed in detail here, since ergonomy aspects are beyond the scope of this report. However, our experience in the development of security application for banking systems, especially authentication scheme, encourage us to consider that the adequacy of proposed tools and user or client acceptation is a must for ensuring use of these security measure - and their inadequacy a powerful deterrent and call for bypass.

2.4 Ontology of core properties of secure architectures

The systematic and reproduceable analysis of security properties require that the set of such properties is properly defined and standardised. This standardisation step is best supported by ontologies: defined as 'an explicit specification of a conceptualization' [Gru95] they provide a structured definition of a knowledge domain.

We consider as reference for the following of this analysis the selected properties from Saltzer and Schroeder, extended with proposed security specific properties updates:

- **Economy of mechanism** (1): Keep the design as simple and small as possible
- **Fail-safe defaults** (2): Base access decisions on permission (white-list) rather than exclusion (black-list)
- **Complete mediation** (3): Every access to every object must be checked for authority
- **Open design** (4): The design should not be secret
- **Separation of privileges** (5): where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key
- **Least privilege** (6): Every program and every user of the system should operate using the least set of privileges necessary to complete the job
- **Least common mechanism** (7): Minimize the amount of mechanism common to more than one user and depended on by all users
- **Psychological acceptability** (8): It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.
- **Minimize secret** (9): each hidden token, used especially for authentication, is another door for abuse and an opportunity of secret theft. As few as possible should therefore be necessary to operate the system.

Figure 2.2 shows the ontology for core properties of secure architectures.

Figure 2.2: Ontology for core properties of secure architectures

The ontology is designed to match the expected properties for ontologies as defined by Gruber [Gru95]:

- **Clarity:** The ontology should effectively communicate the intended meaning of defined terms;

here, reference definitions for the security literature are used, which should be a significant factor for consensus among experts and users

- **Coherence:** The ontology and related axioms should be coherent with informal, natural languages examples; here, the ontology is coherent with the properties under analysis
- **Extendibility:** The representation should be crafted so that one can extend and specialize the ontology; new principles can in fact be added if the core properties are completed by the community
- **Minimal encoding bias:** The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding. No context-specific encoding is expected to be used here
- **Minimal ontological commitment:** An ontology should make as few claims as possible about the world being modeled; the extraction of the principles from the literature ensures that expert consensus apply over the set of concepts considered here.

This ontology is designed to be a local rather than a global ontology [Usc00], which means it intends to be a properly defined ontology for solving a local problem, without any claim of providing an overall view which would be valid across several knowledge domains.

2.5 Advanced properties of secure architectures

Design of secure architecture is a domain where few formalism have been defined, but lots of solutions proposed.

Available formalisms include UML-based patterns [Fer04], or the application of the SAM Software architecture model [WHD99; Yu+04], which is based on petri-nets and temporal logic. Computational intelligence techniques have also been introduced for validating corporate security policies [Mor+14]. Other solutions, such as tools for requirements elicitation, are beyond the scope of this paper.

Some of the specific architectural solutions worth mentioning for computer and distributed systems are for a single machine: the trusted computing base [AFS97], secure computer system architectures [DK06]; for centralized systems: dedicated architecture for pseudonymisation, of medical data for instance [Rie+07]; for dynamic and heterogeneous systems: middleware architecture for secure cloud like Mimosecco [AGH11].

The diversity of existing solutions underlines the high complexity of software and system architecture. A powerful tool capable of dealing with heterogeneous, dynamic systems, appears to be a strong requirement for performing trustworthy architecture monitoring.

Design Structure Matrix[1] (DSM) represent the dependency structure of complex systems. As such, their are an efficient tool for managing complex IT systems, and, as we claim, security issues of these complex IT systems.

3.1 What are DSM?

DSM is a formalism widely used in the design and production engineering community. It aims at modelling, visualising and analyzing complex systems made up of numerous interacting parts. It aims at compact representation of systems, and provides a capture method for the interdependencies between the elements of the system. It typically enables deriving potential improvements for the system being analyzed.

DSM have been first introduced by Don Steward in 1981, including use of dependency levels instead of binary dependencies [Ste81]. Their use in problem solving issues for complex systems is defined in [Won03].

DSM supports the expression of several complexity level: homogeneous systems, systems with interactions among various domains, with Domain Mapping Matrix (DMM) [Bar+07; DB07], and highly heterogeneous systems, with Multiple-Domain Matrix (MSM) [ML+07]. DSM relates entities of the same type together. DMM performs a mapping between different views of a system, or relates together interacting entities of different types. MSM integrate DSM and DMM, and is commonly considered the level where complete systems can be represented and managed.

DSM are also used as numerical DSM [BE02], for instance expressing the probability of impact from one entity to the other, or as timed matrix for analyzing system evolution [Lan01].

Several operations are typically defined on DSMs [Yas04]: tearing, banding, and clustering. Tearing is the simplification of a system for removing coupled entities. Banding consists in identifying independent activities by grouping dependent one together and highlighting them by light and dark colour bands. Clustering consists in identifying groups of entities having a high dependency degree. Clusters are typically depicted using coloured squares.

One of the additional core benefit of DSMs is their ability to scale up, and easily supports systems built up by more than 1000 components [EB12].

3.2 DSM and IT systems

The use of DSM in IT systems is up to now still relatively limited. It mainly focuses on system exploration and analysis, and on compliance with architecture best practices.

Visualising the internal structure of complex software systems proved to be a benefit in itself, since it enables to extract and share information about the system structure and about

[1] http://www.dsmweb.org/

interdependencies between modules [MRB06]. Hidden knowledge, hardly available to experienced developers being actively involved in a project, thus become available to the team, and to the management. DSM has therefore a huge potential for quality improvement of software systems.

In particular, the effects of certain architectures can be observed and analysed. Modularity, which is known to be critical for maintainability and evolutivity of systems, is in particular made visible [Sul+01]. The compliance of system with other software engineering best practices such as the proper use of layers and subsystems [San+05] is made easy to control, and thus to improve. Similarly, integration issues between systems can be identified and solved [Bro01].

DSM proves to bring a radical insight under the hood of software systems, easing project management, quality tracking, software evolution, and maintenance. However the tools already developed mainly limit themselves to observe the modularity of the system and still misses to support development quality guidelines or architectural choices.

3.3 DSMs for qualitative and quantitative evaluation of dependencies

DSM support multi-level dependency analysis for complex structures, and are typically used to handle up to several thousand components. As such, they support both qualitative and quantitative analysis:

- **Qualitative analysis:** DSM enable to identify the presence of direct and indirect dependency paths between components. DSMs leaning on ontologies therefore enable multi-domain analysis, and help refining dependency understanding both at component and at domain level.
- **Quantitative analysis:** DSM enable to enumerate the quantity of dependencies, as well as the probability thereof. They thus provide a solid basis for obtaining fine quantitative data about complex structures.

3.4 Why DSMs are relevant for IT security

On the one hand, the domain of architecture security is missing a technology-independent tool for supporting the monitoring of software systems, and on the other hand, DSM already proved, although with deployment limited to rare academic projects, to provide a relevant solution for visualisation and analysis of complex software systems.

We therefore believe that Design Structure Matrices (DSM):

- enable the visualisation of the internal of complex systems, which is a pre-requisite for architecture security monitoring
- enable to characterize security-relevant interactions between system entities, such as software modules, in a qualitative manner (which entity is in interaction with which one?)
- enable to characterize security-relevant interactions between system entities in a quantitative manner (how much does an entity interact with another?)

- and therefore build an efficient tool for monitoring the actual system state, which is a strong prerequisite for building secure systems.

The following of the paper will be dedicated to the definition of a DSM-based model for architecture monitoring, and its validation.

The proposed methodology for building and using the DSM-based model for performing a qualitative architecture monitoring based on Saltzer and Schroeder security principles is presented in this section. It is illustrated by a toy example, a 'ticket reservation system'. This chapter presents the qualitative evaluation of Design structure Matrices (DSM) for enforcing security principles.

4.1 The 'ticket reservation' toy example

Let's consider a toy application for ticket reservation. This application is built of three tiers, Presentation, Business, and Data. A clean layered architecture is enforced, *i.e.* a layer can only access to the layer immediately below it. This application will be used for illustrating the process of DSM elicitation and of characterizing architecture security properties.

Figure 4.1 shows the software stack for the 'ticket reservation' web application.

Figure 4.1: Software stack for 'ticket reservation' web application

Figure 4.2 shows the DSM for the 'ticket reservation' web application: it highlights the dependencies between application modules, in our case between the three tiers of the application.

	Presentation	Business Tier	Data Tier
Presentation Tier	X	X	
Business Tier		X	X
Data Tier			X

Figure 4.2: DSM for the 'ticket reservation' web application

4.2 Ontology for the components of a multi-tier web application

The automation of the extraction of the DSM requires a sound standardisation of the concepts handled by the matrix, again provided under the form of an ontology. Analysing the properties of dependencies between application modules requires that these modules are classified in a suitable and reproduceable manner.

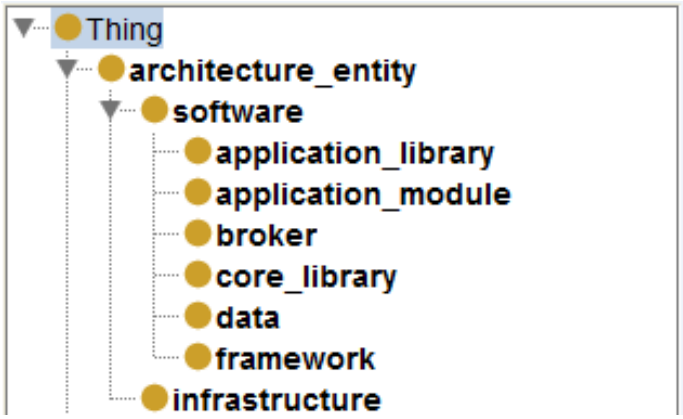

Figure 4.3: Ontology for the components of a multi-tier web application

Figure 2.2 shows the ontology for the components of a multi-tier web application.

The ontology is designed to comply with the properties of ontology definition according to Gruber [Gru95]: clarity, coherence, extendibility, minimal encoding bias, minimal ontological commitment.

4.3 The role of the service broker

Controlling the actual implementation of a security policy requires a centralised broker, which defines and enforces the roles, access rights and resources. A typical market implementation of such a broker is the Active Directory service of Microsoft Windows servers. Its function is to centralise all interactions. Its role is to enable the enforcement of the complete mediation property in the system under control.

Brokers can be either centralised or distributed. In a web application located in one single place, centralisation make full sense. In distributed systems, such as ecosystems of mobiles and embedded devices, distributed control will be preferred, since it ensures a continue overview of all enforced policies. However, system developers and administrators will need to be of great care in this case to ensure that all devices actually enforce access control policies, and that they have access to their current version - including potential revocations as in certificate revocation lists.

4.4 Qualitative expression of Saltzer and Schroeder's principles

The expression of Saltzer and Schroeder's principles as applied to a Design Structure Matrix representation of a software architecture is now given. Relying on an ontology of these principles ensures the repeatability of analyses based on our framework.

- **Economy of mechanism** : Figure 4.4 shows the Design Structure Matrix for economy of mechanism; Figure 4.5 shows the case where access control is performed in the code rather than through a centralized broker. The overall number of dependencies should be reduced; L,

M, N, denote the number of dependencies from a module to another. A broker is a specific module which references and support access to other application modules.

	Presentation Tier	Business Tier	Data Tier	Broker
Presentation Tier	-			M
Business Tier		L		N
Data Tier			-	
Broker		M	N	-
L, M, N, minimal				

Figure 4.4: Design principle: economy of mechanism

	Presentation Tier	Business Tier	Data Tier	Broker
Presentation Tier	-	M		
Business Tier		L	N	
Data Tier			-	
Broker				-
M, N, minimal				

Figure 4.5: Design principle: economy of mechanism (with no broker)

- **Fail-safe defaults** : this property should be enforced at implementation level.
- **Complete mediation** : Figure 4.6 shows the Design Structure Matrix for complete mediation, where all access to functional services are performed over a dedicated mediator, thus enabling the enforcement of systematic access control policies.

	Presentation Tier	Business Tier	Data Tier	Broker
Presentation Tier	-			X
Business Tier		-		X
Data Tier			-	
Broker		X	X	-

Figure 4.6: Design principle: complete mediation

- **Open design** : this principle is matched if Security DSM for the architecture of an application are given.
- **Separation of privileges** : Figure 4.7 shows the Design Structure Matrix for separation of privilege, where complex operations are parted in several smaller operations each bound with specific, restricted privileges.
  F represents full access to a resource.
  Several F's on a single column represent several ways to access a resource. This is especially to be avoided when access is performed in inconsistent ways across the layers. This is an obvious way of lacking separation of privileges)
  P1, P2 represent several accesses to a single resource with individualized partial access rights, which is an example of separation of privileges.

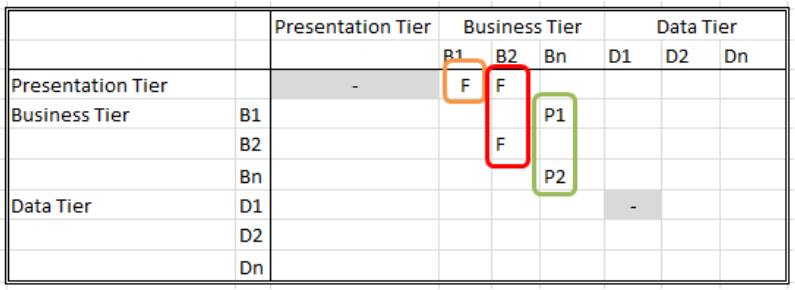

Figure 4.7: Design principle: separation of privileges

- **Least privilege** : in Figure 4.7, the case with one module having unlimited access rights to a given service is a counterexample of least privilege.
- **Least common mechanism** : An excessive number of dependencies from several services to a single one could be a source of weakness and should be limited when possible.
- **Psychological acceptability** : This is an ergonomy issue and can't be reduced to architectural or implementation properties of the application.
- **Minimize secret** : Identification tokens are not consider further here.

Different types of analysis will be required to enforce all these properties: architectural analysis need to be performed for verifying the following properties: economy of mechanism, complete mediation, least common mechanism, layered architecture (which is a criterion for separation of privileges); access right analysis need to be performed for verifying following properties: separation of privileges (requires layered architecture property to be enforced), least privilege; code review needs to be performed for verifying following properties: fail-safe default; user testing needs to be performed for verifying following properties: psychological acceptability. Note that the open design property requires both architectural and access right analysis.

4.5 The role of the service broker

From previous definition it is obvious that the 'complete mediation' property is not enforced in the 'ticket reservation' toy application. For enforcing it, a dedicated service broker, which will provide access to each module of the system and validate authorizations for each call, is required. Of course, as we will show in section 6, the systematic use of a service broker for all access module is only an approximation. Nonetheless, it underlines the necessity to refactor the application which emerge from the DSM-based monitoring.

Figure 4.8 shows the software stack for the 'ticket reservation' web application with a service broker, which is typically located at the business tier.

Figure 4.9 shows the DSM for the 'ticket reservation' web application. It highlights the compliance with the 'complete mediation' principle, since the presence of the broker is clearly visible.

Using the 'ticket reservation' toy application, we show how Design Structure Matrix can be based upon to perform a qualitative security evaluation of an application.

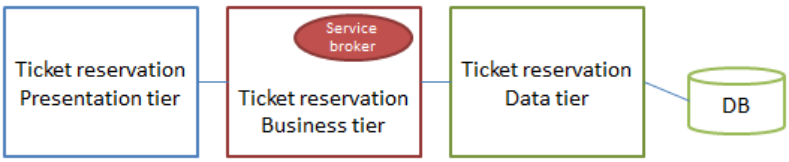

Figure 4.8: Software stack for 'ticket reservation' web application

	Presentation	Business Tier	Data Tier	Broker
Presentation Tier	-			X
Business Tier		-		X
Data Tier			-	
Broker		X	X	-

Figure 4.9: DSM for the 'ticket reservation' web application

The reification of the proposed process for software architecture monitoring requires to automatically extract the Design Structure Matrix expressing the dependencies between modules. It is therefore possible to build a tool for performing this evaluation. Our implementation, in Python, is named Archan (Architecture Analysis), and is integrated in the management interface of Django-Suite web management framework. It is a new implementation, dedicated to security issues, of the HeatMatrix tool [Lut+14] for the design and management of complex computational ecosystems.

In this first step of our analysis, Archan is used for qualitative analysis of the system dependencies.

5.1 Archan controls

The current implementation of Archan controls supports the monitoring of both code quality standards and of the structure of software dependency.

Archan controls are:

- Automatic and monitorable controls, *i.e.* they support the escalation to manual enforcement for unsupported issues
- Preventive controls, since their enforcement prevents unauthorised accesses
- detective controls, since monitoring enables a quick visualisation of abnormal evolution of the system, and especially its architecture.

5.2 Archan Module Dependency tracking

Figure 5.1 shows the DSM visualisation using Archan of the early versions of a medical web site project, Genida[1], we use for evaluating the approach we propose (see section 7).

Archan depicts module dependencies according to module groups (framework - here Django, core libraries - here Python libraries, application libraries part of the application, application modules), according to name, and according to frequency of dependencies. These groups are specified in the ontology for component of multi-tier applications defined in paragraph 4.2. The reliance on an ontology enables the reusability of the framework and its use for similar application architectures. Direct and indirect dependencies with depth 2 to 5 are supported, as well as item ordering by type of module or by alphabetical order. Archan also enable to extract the data as CSV file, and to access previous versions of the DSMs.

Three categories of modules are considered: modules from Django framework, in green, third party libraries, in red, and Genida modules, in purple. Django framework typically show no dependencies between the different modules. Several libraries are extension of Django, and thus have dependencies to Django modules. Genida modules are highly coupled, and rely both on

[1] https://genida.unistra.fr/

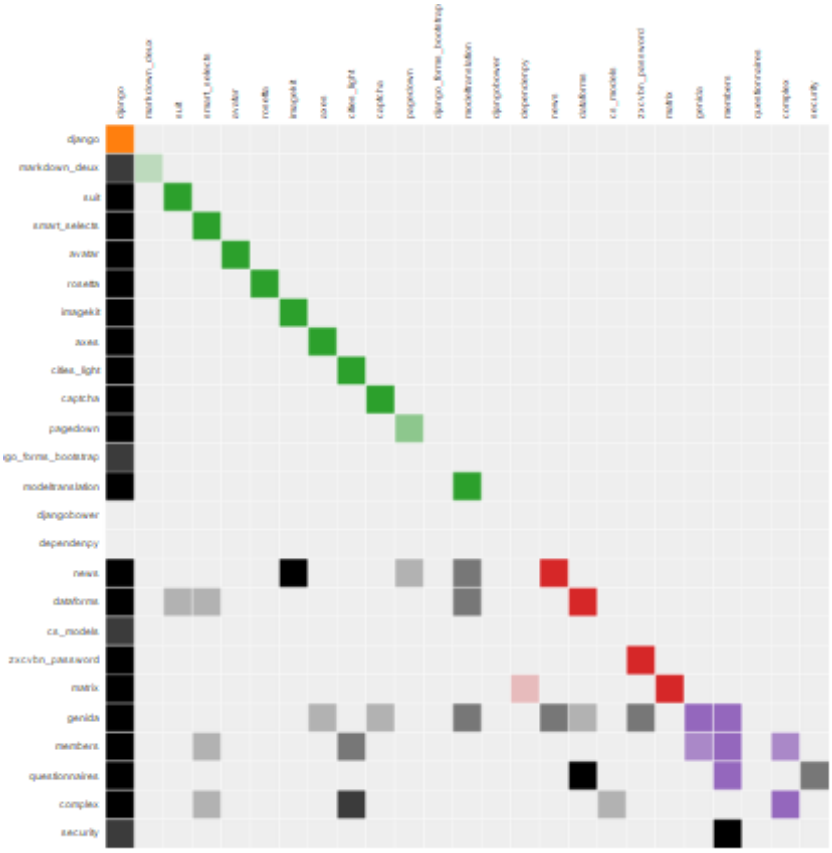

Figure 5.1: Visualisation of module dependencies using Archan (Architecture Analysis) tool

libraries which support them and on the Django framework itself, which provides standard classes to be implemented for graphical interface and form development.

The evaluation of the Archan DSM is given in section 7, and its relevance for architecture monitoring is discussed in section 8.

Performing a DSM-based monitoring requires to formalize several elements of the models. First, the entities of the model, as well as the domain they pertain to, need to be defined. This is done through an ontology. Next, the interactions between the modules are to be quantified. This is done by extracting the oriented, weighted graph which is implicitly represented by the DSM. Using graphs has the advantage to enable simple expression of the constraints on the interactions (the arcs), the incoming dependencies for a module (ingress nodes) and outgoing dependencies for a module (egress nodes). Arc direction represent direction of dependency. Arc weight represents the number of these dependencies.

Quantitative criteria are defined for architectural security principles. They are illustrated with a second toy application, the ‘online store’.

6.1 The online store

The online store is an application having a set of application modules, a set of application specific libraries, as well as data-access-object modules for the binding with the database. It is based on a web framework.

Figure 6.1 shows the software stack for Online Store application.

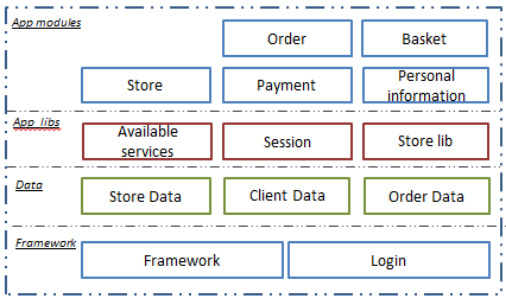

Figure 6.1: Software stack for Online Store applications

Figure 6.2 shows the Dependency Matrix for Online Store application.

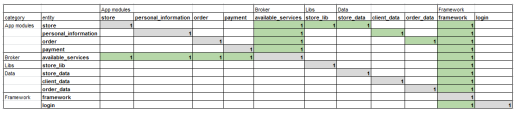

Figure 6.2: Dependency Matrix: Online Store modules

Figure 6.3 shows the Archan DSM model for the online store.

```
webAppCategories = ['app_module', 'app_module', 'app_module',
                    'app_module', 'broker',
                    'app_lib', 'data', 'data',
                    'data', 'framework', 'framework']
webAppEntities = ['store', 'personal_information', 'order',
                  'payment', 'available_services',
                  'store_lib', 'store_data', 'client_data',
                  'order_data', 'framework', 'login']
webAppDsm = [[1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0],
             [0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0],
             [0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0],
             [0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0],
             [1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0],
             [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0],
             [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
             [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
             [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
             [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
             [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]
self.webAppDependencyMatrix = DesignStructureMatrix(webAppCategories,
                                                    webAppEntities,
```

Figure 6.3: Archan DSM model for the online store

6.2 The model for the 'online store'

Figure 6.4 shows the ontology used for characterizing secure architectures. It contains both the architecture entities and the secure architecture principles. Here, only software architecture, not the infrastructure, is considered.

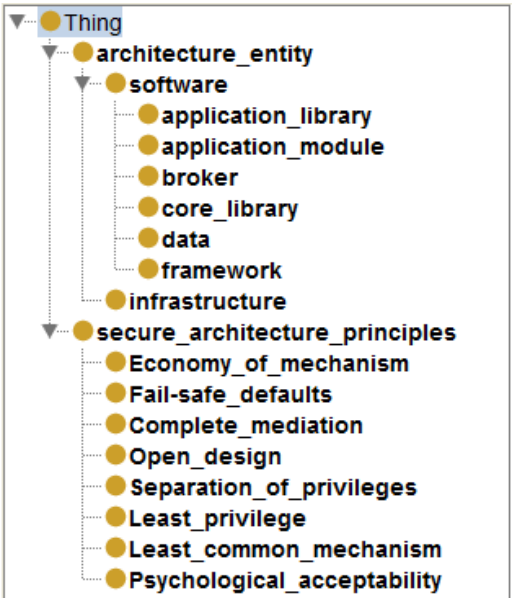

Figure 6.4: A pragmatic ontology for secure architectures

Figure 6.5 shows the oriented graph for the Online Store software architecture.

Ontology and graph enable to complete the available DSM model for supporting the subsequent analysis.

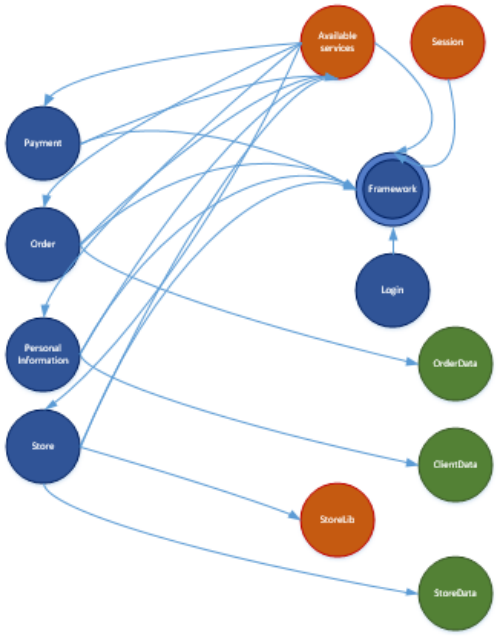

Figure 6.5: Online Store software architecture graph

6.3 Quantitative expression of Saltzer and Schroeder's principles

In this step of our analysis, the Archan tool is used for quantitative analysis of the system dependencies by extracting system specific metrics for the considered architecture.

The following principles are elected for architectural evaluation of software systems:

- **Economy of mechanism** , or simplicity: The number of edges in the module dependency graph (see Figure 6.5) can't exceed M times the number of vertices (see equation 6.2). Dependencies to (and if relevant from) the broker and the framework are not considered (as edges), since they can be considered as systematic and contribute to the system simplicity. Likewise, the broker and the framework are not considered in the module list (the vertices). We call M the simplicity factor.

$$e <= M * v \tag{6.1}$$

$$\begin{aligned} &with\ e\ the\ number\ of\ edges\ in\ the\ dependency\ graph \\ &v\ the\ number\ of\ vertices\ in\ the\ dependency\ graph \\ &M\ the\ simplicity\ factor \end{aligned} \tag{6.2}$$

  The smaller M, the simpler the system. In the 'online store' example, $M = e/v = 17/9 = 1,9$.

- **Complete mediation** : Only the broker can have access to all services; no dependency between application modules; no dependency between data modules; libraries can only have

dependencies to libraries

$$e_{am \to am} = 0 \tag{6.3}$$

$$e_{am \to b} = N_{am} \tag{6.4}$$

$$e_{b \to am} = N_{am} \tag{6.5}$$

$$\begin{aligned} &with\ e_{am \to am}\ the\ number\ of\ edges \\ &between\ application\ modules \\ &e_{am \to b}\ the\ number\ of\ edges\ between \\ &application\ modules\ and\ the\ broker \\ &e_{b \to am}\ the\ number\ of\ edges\ between \\ &the\ broker\ and\ application\ modules \\ &N_{am}\ the\ number\ of\ application\ modules \end{aligned}$$

- **Open design** : a criteria is checked for all 7 objective security criteria of Saltzer and Schroeder (evaluation of psychological acceptability can't be performed in an automated manner)
- **Least common mechanism,** or independence: the maximum number of incoming edges to a given vertex in the module dependency graph can't exceed a N^{th} of the total number of vertices (see equation 6.7). Again, broker and framework are not considered.

$$max(e_i) < v/N \tag{6.6}$$

$$\begin{aligned} &with\ e_i\ the\ number\ of\ ingress\ edges \\ &to\ a\ given\ node\ in\ the\ dependency\ graph \\ &v\ the\ total\ number\ of\ vertices\ in \\ &the\ dependency\ graph \\ &N\ independence factor. \end{aligned} \tag{6.7}$$

  The bigger N, the more independent the modules. In the Online Store example, $N = v/max(e_i) = 9/2 = 4,5$.
- **Minimize secret** : the use of secret tokens is not visible at this abstraction level.

Quantitative criteria based on oriented weighted graphs have been expressed for the architectural Saltzer and Schroeder properties. They have been illustrated with the 'online store' example.

To evaluate the proposed method, we perform an architecture security evaluation of a medical web platform for rare genetic diseases. The application is quite sensitive, since it contains information related to the genetic of patients and of their relatives, often bounded with intellectual deficiencies. It is therefore of high importance to be able to show that the application is developed according to the best practices.

7.1 Software Architecture

The software stack, as well as the dependency matrix generated by Archan, is given for our target application. Figure 7.1 shows the software stack for our medical application.

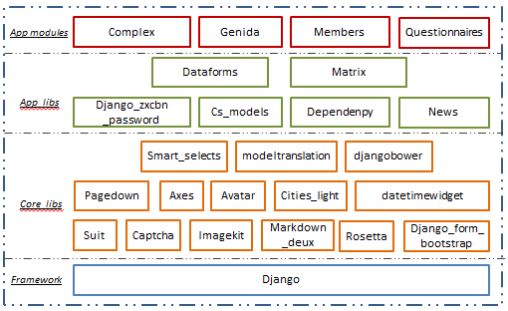

Figure 7.1: Software stack for our medical application

Figure 7.2 shows the visualisation of module dependencies using Archan. Note that this DSM represent a refactored version of the application, the older version having been shown in Figure 5.1.

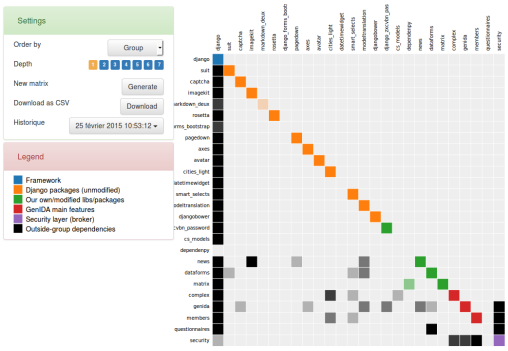

Figure 7.2: Visualisation of module dependencies using Archan

7.2 Compliance with Saltzer and Schroeder criteria

Archan enable to extract architecture security metrics as defined in section 6:

- **Economy of mechanism** : M, the simplicity factor, is $M = 2$. This is quite a good score for simplicity. Economy of mechanism principle is thus enforced.
- **Complete mediation** : After the first version of the project has been released (see Figure 5.1), the project has been refactored to comply with this principle. Complete mediation principle is thus enforced.
- **Open design** : All 3 defined metrics for secure architecture are enforced. However, 4 other principles are to be validated through complementary analysis, so it is not possible to claim that open design is enforced here.
- **Least common mechanism** : N, the independence factor, is $N = 6$. This is a satisfactory score for independance. Least common mechanism principle is thus enforced.

The extraction of relevant metrics and the visualisation thus prove to be an efficient solution for checking whether a given application complies with the identified security requirements. Moreover, it proves to be an efficient communication tool to urge teams to quickly correct obvious fails, even in the case of quite complex issues such as ‘complete mediation’.

In this section, we discuss the relevance of the proposed approach and tool, and explicit recommendations based on the performed evaluations for building secure architectures.

8.1 Relevance of Design Structure Matrices as architecture security controls

The objective of this work is to evaluate the relevance of Design Structure Matrix as a versatile tool for performing architecture security evaluations.

Based on the experimentation we drive, we can conclude that Design Structure Matrices (DSM):

- enable the visualisation of the internal of complex systems, and are an efficient communication tool and change management vector
- enable to characterize security-relevant interactions in a qualitative manner, through visualisation of the principles of simplicity, complete mediation, and independence
- enable to characterize security-relevant interactions in a quantitative manner, through quantitative expression of cited properties on the graphs that are implicitly defined by the DSMs

One of the core properties of DSM is its scalability. It is therefore necessary to evaluate the efficiency of Archan on bigger systems. Figure 8.1 shows the DSM for the Oodo-OpenERP software stack, which contains some hundreds of software components, using Archan. Oodo-OpenERP is an Open-Source Enterprise Resource Planning software developed in Python. We use it as an example of production software to validate the consistency and the scalability of our approach. Only significant dependencies are given.

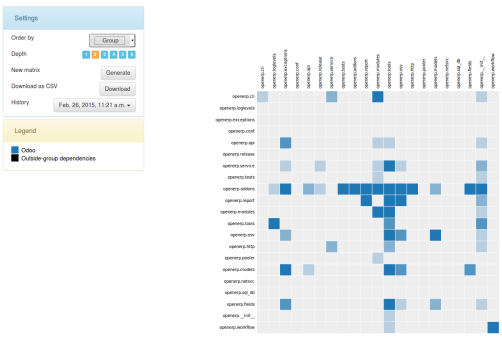

Figure 8.1: Visualisation of module dependencies for the Oodo-OpenERP stack using Archan

The Oodo-OpenERP DSM shows both that the tool has no issue in scaling for bigger applications, and for extracting relevant metrics. However, it also show that best architecture security practices, such as ‘Complete mediation’ must be enforced from the beginning of the development process to avoid huge refactoring efforts.

8.2 The Archan tool

The first version of the Archan tool is presented here. It proves to ensure rich features, ergonomy, as well as scalability. It is currently only a research prototype so far. Subsequent developments will be needed to make an actual evaluation tool out of this promissing proof-of-concept.

The proposed visualisation proves its efficiency for some tens of software components. The scalability of the approach is confirmed by the application of Archan to Oodo-ERP, which contains several hundreds of components. However, for bigger systems up to several thousands of components, specific strategies will need to be devised: representation of dependencies between component categories rather than between components, extraction of relevant dependencies only, focus on system structure metric rather than on the visualisation matrices. More experimentation are required to answer these questions.

The core contributions of the Archan tool are the following ones:

- Archan creates a monitorable technical control for key security properties of software architectures
- It automatises and ensures the enforcement of these properties in complex, rapidly evolving software frameworks such as web applications
- It supports the visualisation of the actual status of the system being monitored
- It bridges the gap between high-level security principles and the actual implementation of the system
- Lastly, it provides technical artefacts for secure software architectures, not only blueprints and principles

8.3 Recommendations for building secure software architectures

Based on this evaluation, we recommend following architectural choices for sensitive systems such as medical applications.

- simplicity: the total number of modules having interdependencies should be as low as possible.
- independence: modules acting as single points of failure for too many other modules should be avoided.
- the use of a broker for service access inside web applications is highly recommended, so as to enable transparent and simple enforcement of authorizations.

In service oriented framework like Spring, or JBoss, the ‘broker’ feature is part of the architecture, and should be used extensively. On other, like Django, this feature needs to be developed by the developer.

9 Conclusions and perspectives

In this work, we present a methodology for performing architecture evaluations for complex applications exploiting the power of DSM (Design Structure Matrices). Based on Saltzer and Schroeder principles for secure computing systems, we define qualitative as well as quantitative criteria for evaluating the security status of software architectures. A tool, Archan (Architecture Analysis) is developed to implement this methodology. The methodology is illustrated on two simple examples: the 'ticket reservation' and the 'online store' applications. It is evaluated using an actual web application from the medical domain handling very sensitive data like genetic information of patients and of their relatives.

The evaluation presented here show that DSMs are actually a powerful tool for performing the evaluation of complex systems, in particular in the security domain. Beside efficient analyses, they support visualisation, and therefore efficient communication of the results to stakeholders. This features has the potential to bring an important improvement in the process of architecture monitoring, which is so far often based on experience more than on factual, reproducible analyses.

This work builds a first step in the exploration of DSMs for security engineering. We believe they have the ability to adapt naturally to other systems such as network infrastructures, to support efficient authorization models, as well as to be able to adapt to rapidly changing systems, which is a huge challenge nowadays.

10 Scientific validation

This work has been reviewed and accepted for publication as a Research Report of the Complex System-Digital Campus by:

- Véronique Legrand, Professeur du Conservatoire National des Arts et Métiers (Paris, France)
- Anne Jeannin-Girardon, McF, Université de Strasbourg (France)

Bibliography

[Yas04] A Yassine. “An introduction to modeling and analyzing complex product development processes using the design structure matrix (DSM) method”. In: *Urbana* 51.9 (2004), pp. 1–17.

[MRB06] Alan MacCormack, John Rusnak, and Carliss Y Baldwin. “Exploring the structure of complex software designs: An empirical study of open source and proprietary code”. In: *Management Science* 52.7 (2006), pp. 1015–1030.

[SS75] Jerome H Saltzer and Michael D Schroeder. “The protection of information in computer systems”. In: *Proceedings of the IEEE* 63.9 (1975), pp. 1278–1308.

[McG06] Gary McGraw. *Software security: building security in*. Vol. 1. Addison-Wesley Professional, 2006.

[PHL] Gunnar Peterson, Paco Hope, and Steven Lavenhar. “Architectural Risk Analysis”. In: ().

[Hal+08] Spyros T Halkidis et al. “Architectural risk analysis of software systems based on security patterns”. In: *IEEE Transactions on Dependable and Secure Computing* 5.3 (2008), pp. 129–142.

[Her02] Debra S Herrmann. *Using the Common Criteria for IT security evaluation*. CRC Press, 2002.

[SK09] Jerome H Saltzer and M Frans Kaashoek. *Principles of computer system design: an introduction*. Morgan Kaufmann, 2009.

[PP02] Charles P Pfleeger and Shari Lawrence Pfleeger. *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

[Smi12] Richard E Smith. “A contemporary look at Saltzer and Schroeder’s 1975 design principles”. In: *IEEE Security & Privacy* 10.6 (2012), pp. 20–25.

[Smi15] Richard E Smith. *Elementary information security*. Jones & Bartlett Publishers, 2015.

[Gru95] Thomas R Gruber. “Toward principles for the design of ontologies used for knowledge sharing?” In: *International journal of human-computer studies* 43.5-6 (1995), pp. 907–928.

[Usc00] Mike Uschold. “Creating, integrating and maintaining local and global ontologies”. In: *Proceedings of the First Workshop on Ontology Learning (OL-2000) in conjunction with the 14th European Conference on Artificial Intelligence (ECAI-2000)*. Citeseer. 2000.

[Fer04] Eduardo B Fernandez. “A Methodology for Secure Software Design.” In: *Software Engineering Research and Practice*. 2004, pp. 130–136.

[WHD99] Jiacun Wang, Xudong He, and Yi Deng. “Introducing software architecture specification and analysis in SAM through an example”. In: *Information and Software Technology* 41.7 (1999), pp. 451–467.

[Yu+04] Huiqun Yu et al. “A formal approach to designing secure software architectures”. In: *High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on*. IEEE. 2004, pp. 289–290.

[Mor+14] Antonio M Mora et al. “Enforcing corporate security policies via computational intelligence techniques”. In: *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*. ACM. 2014, pp. 1245–1252.

[AFS97] William A Arbaugh, David J Farber, and Jonathan M Smith. “A secure and reliable bootstrap architecture”. In: *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. IEEE. 1997, pp. 65–71.

[DK06] Guillaume Duc and Ronan Keryell. “CryptoPage: an efficient secure architecture with memory encryption, integrity and information leakage protection”. In: *Computer Security Applications Conference, 2006. ACSAC’06. 22nd Annual*. IEEE. 2006, pp. 483–492.

[Rie+07] Bernhard Riedl et al. “A secure architecture for the pseudonymization of medical data”. In: *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*. IEEE. 2007, pp. 318–324.

[AGH11] Dirk Achenbach, Matthias Gabel, and Matthias Huber. “Mimosecco: A middleware for secure cloud storage”. In: *Improving Complex Systems Today*. Springer, 2011, pp. 175–181.

[Ste81] Donald V Steward. “The design structure system- A method for managing the design of complex systems”. In: *IEEE transactions on Engineering Management* 28.3 (1981), pp. 71–74.

[Won03] Allan Kai Tung Wong. “Before and Beyond Systems: An Empirical Modelling Approach”. PhD thesis. University of Warwick, 2003.

[Bar+07] J Bartolomei et al. “Analysis and applications of design structure matrix, domain mapping matrix, and engineering system matrix framework”. In: *Massachusetts Institute of Technology Engineering Systems Division* (2007), pp. 1–15.

[DB07] Mike Danilovic and Tyson R Browning. “Managing complex product development projects with design structure matrices and domain mapping matrices”. In: *International Journal of Project Management* 25.3 (2007), pp. 300–314.

[ML+07] Maik Maurer, Udo Lindemann, et al. “Structural awareness in complex product design–The Multiple-Domain Matrix”. In: *DSM 2007: Proceedings of the 9th International DSM Conference, Munich, Germany, 16.-18.10. 2007*. 2007.

[BE02] Tyson R Browning and Steven D Eppinger. “Modeling impacts of process architecture on cost and schedule risk in product development”. In: *Engineering Management, IEEE Transactions on* 49.4 (2002), pp. 428–442.

[Lan01] Michele Lanza. “The evolution matrix: Recovering software evolution using software visualization techniques”. In: *Proceedings of the 4th international workshop on principles of software evolution*. ACM. 2001, pp. 37–42.

[EB12] Steven D Eppinger and Tyson R Browning. *Design structure matrix methods and applications*. MIT press, 2012.

[Sul+01] Kevin J Sullivan et al. “The structure and value of modularity in software design”. In: *ACM SIGSOFT Software Engineering Notes* 26.5 (2001), pp. 99–108.

[San+05] Neeraj Sangal et al. “Using dependency models to manage complex software architecture”. In: *ACM Sigplan Notices*. Vol. 40. ACM. 2005, pp. 167–176.

[Bro01] Tyson R Browning. “Applying the design structure matrix to system decomposition and integration problems: a review and new directions”. In: *Engineering Management, IEEE Transactions on* 48.3 (2001), pp. 292–306.

[Lut+14] Evelyne Lutton et al. “GridVis: Visualisation of Island-Based Parallel Genetic Algorithms”. In: *Parallel Architectures and Distributed Infrastructures (EvoPar’2014)*. Apr. 2014.

Publisher
Complex System Digital Campus
http://unitwin-cs.org

ARK unitwin-cs.org/ark:/69427/03