Universitat Autònoma de Barcelona

Escola d'Enginyeria
Departament d'Arquitectura de
Computadors i Sistemes Operatius

Vulnerability Assessment for Complex Middleware Interrelationships in Distributed Systems

Tesis doctoral presentada por Jairo D. Serrano Latorre para optar al grado de Doctor por la Universitat Autònoma de Barcelona, bajo la dirección de Dr. Elisa Heymann, y Dr. Eduardo César.

Bellaterra, September 30, 2013

Vulnerability Assessment for Complex Middleware Interrelationships in Distributed Systems

Tesis doctoral presentada por Jairo David Serrano Latorre para optar al grado de Doctor por la Universitat Autònoma de Barcelona. Trabajo realizado en el Departament d Arquitectura de Computadors i Sistemes Operatius de la Escola d Enginyeria de la Universitat Autònoma de Barcelona, dentro del Programa de Computación de Altas Prestaciones , bajo la dirección de Dr. Elisa Heymann, y Dr. Eduardo César.

Bellaterra, September 30, 2013

Directores: Autor:

Dr. Elisa Heymann, Dr. Eduardo César Jairo D. Serrano Latorre

Aknowledgment

To my Mom, grandparents, and beloved relatives, friends, and some foes. To my advisors, Elisa and Eduardo, for their tolerance and patience. To Emilio and Lola for their confidence and teaching.

"M. Csikszentmihalyi explica en su libro "Fluir" que no importa la actividad que elijamos para disfrutar ni tampoco determina el nivel de placer de la actividad, el sexo, la cultura, la clase social o la edad. Lo que realmente importa es cómo nos sentimos mientras hacemos esa actividad. Parece que la capacidad de disfrutar requiere de las mismas características psicológicas para todo el mundo. Algunos requisitos de la actividad para que nos haga disfrutar son que tengamos posibilidades de éxito al realizar la tarea, ser capaces de concentrarnos en ella, tener unas metas claras y obtener información inmediata de si las conseguimos o no, actuar con tanta involucración que nos olvidemos de lo cotidiano, ser capaces de olvidarnos hasta de nosotros mismos mientras la realizamos, y que el sentido del tiempo se vea alterado, las horas pasan volando y los minutos parecen horas mientras estamos inmersos en ella"

Abstract

The fast adaptation of Cloud computing has led to an increased speedy rate of novel information technology threats. The targets of these new threats involve from large scale distributed system, such as the Large Hadron Collider by the CERN, up to industrial (nuclear, electricity, oil, etc.) distributed systems, i.e. SCADA networked systems.

The use of automated tools for vulnerability assessment is quite attractive, but while these tools can find common problems in a program s source code, they miss a significant number of critical and complex vulnerabilities. In addition, frequently middleware systems of distributed systems base their security on mechanisms such as authentication, authorization, and delegation. While these mechanisms have been studied in depth and might have control over key resources, they are not enough to assure that all application s resources are safe. Therefore, security of distributed systems have been placed under the watchful eye of security practitioners in government, academia, and industry. To tackle the problem of assessing the security of critical middleware systems, we propose a new automated vulnerability assessment methodology, called *Attack Vector Analyzer for Complex Middleware Interrelationships(AvA4cmi)*, which is able to automatically hint which middleware components should be assessed and why.

AvA4cmi is based on automatizing part of the First Principles Vulnerability Assessment, an innovative analystic-centric (manual) methodology, which has been used successfully to evaluate several known middleware systems. AvA4cmi s results are language-independent, provide a comprehensive assessment of every possible attack vector in the middleware, and it is based on the Common Weakness Enumeration (CWE) system, a formal list for describing security weaknesses. Our results are contrasted against previous manual vulnerability assessment of the CrossBroker and gLite WMS middleware, and corroborate which middleware components should be assessed and why.

Contents

List of Figures

List of Tables

CHAPTER 1

Overview

The rapidly changing information society has led to an increasing rate of novel information technology threats, even more with the fast adaptation of Cloud computing and the hundreds of thousands gigabytes of sensitive information being shared and distributed over computer networks. The targets of these new threats range from large scale distributed system, such as the Large Hadron Collider by the CERN, to industrial (water, power, electricity, oil, gas, etc.) distributed systems, i.e. SCADA systems. Consequently, security of distributed systems have been placed under the watchful eye of security practitioners in government, academia, and industry. This is because, on one hand, automated tools can find basic security errors in source code, but overlook a significant number of critical and complex vulnerabilities, and, on the other hand, current security mechanisms of distributed systems - authentication, authorization, certification, and delegation usually are not in accordance with a systematic vulnerability assessment process, which is an afterthough even during the software development life cycle (SDLC).

In addition, conducting a comprehensive vulnerability assessment can be very expensive and complicated, due to several factors such as the complexity and size of the systems to be evaluated. Consequently, it would be very helpful to develop techniques and tools that help the analyst focus on the systems high value assets.

The consequences of this lack of vulnerability assessment are distributed systems that can be compromised and exploited cleverly. This work focuses on the guidance towards a more comprehensive and accurate vulnerability assessment, characterized by considering the high interoperability between middleware components for systematically hinting where and why to deploy an assessment. In vulnerability assessment, security practitioners goals are both to uncover as quickly as possible the most likely vulnerabilities, i.e., to be one step ahead of the attackers, and to provide the developers full details of vulnerabilities found, i.e., vulnerability reports, including suggested fixes.

The deployment of comprehensive and accurate vulnerability assessment of critical middleware systems involves a great number of issues. In particular, this work deals with four of them:

- Selecting and applying a vulnerability assessment methodology, from the various existing methodologies, needed for focusing the analyst s attention on the parts of the middleware and its resources that are mostvaluable, and which are the vulnerabilities they are more likely suffer.
- Selecting and applying a security error classification system, from the different existing systems, needed for improving analyst s vulnerability assessment obtaining quality accomplishments during source code inspection, following the previously selected methodology.
- Reducing the false positive and false negative rates, produced when an automated vulnerability assessment of an application is performed, such as when using automated tools, so security analysts have to face hundreds or thousands of individual bug reports for weaknesses that were discovered or not really. Forcing the analysts into a situation in which they must prioritize which issues they should investigate and fix first.
- The absence of a formal method that attempts to both systematically use and connect the information gathered by the vulnerability assessment methodologies, and the knowledge found on the security error classification systems.

Throughout this work, workload management systems have been assessed because most common and frequently used high-end service fit into this kind of critical middleware. In these middlewares, there are several components with complex interrelationships across different hosts machines, providing lots of large distributed systems, which mean a significant number of computing resources, which make them attractive targets for attackers.

In order to perform a comprehensive and accurate vulnerability assessment for critical middlewares, a methodology is needed. Thus, we considered several vulnerability assessment approaches and security error classification systems, from which the First Principle Vulnerability Assessment *(FPVA)* [42] was choosen along with the Common Weakness Enumeration *(CWE)* system [77]. This was decided, on the one hand, not only because FPVA has been successfully applied to several large and widely-used middleware systems, but also because a thorough vulnerability assessment requires a systematic approach that focuses on the key resources to be protected, and allowing detailed analysis of those parts of the code related to those resources and their trust relationships.

While FPVA was designed to address these requirements, other approaches are based on lists of known threats, developer team interactions, single component code analysis, implementation during the whole development lifecycle; and could lead to a biased analysis, where known vulnerabilities may be detected only, but may not refer to high value assets, and may result in critical threats going undetected.

On the other hand, CWE is a widely-used labeling system of security errors, and because CWE was created to serve as a common language for describing software security errors; serve as a standard measuring stick for software security tools targeting these erros; and to provide a common baseline standard for weakness identification, mitigation, and prevention efforts.

The methodology presented in this work do vulnerability assessment for distributed systems, which uses knowledge about the system, consisting on components and resources obtained with FPVA, and the expertise codified in CWE for focusing the analyst attention on the most likely attacks on the highest value system assets. It does not need either access the middleware source code or meet the developer teams, and does not have to be deployed during the whole lifecycle. It works by automating part of FPVA, using CWE for systematically codified security practitioner s expertise in order to be able to automatically hint which middleware components should be assessed, and why.

In this work we:

- Use the information gathered at the initial FPVA analysis stages, that is, the outcome information (the FPVA diagrams) on the middleware architecture, resources, and privileges analysis.
- Use the impact surface definition derived from several vulnerability assessments conducted by MIST research group [51], along with the well known attack vector and attack surface definitions.
- Use codified rules derived from CWE to connect initial analysis with the middleware component analysis through our customized version of the CWE scoring system.

And we obtain a list of security alerts for every attack vector traversed during the middleware assessment, which are clustered in accordance to the CWE hierarchical structure chosen, and arranged by the maximum score obtained.

Therefore from the initial stages of FPVA: the architectural analysis stage produces a document that diagrams the structure of the system and the interactions amongst the different components and with the end users. With this diagrams, the Attack Surface of the system was derived, the Attack Surface is the set of coordinates from which an attack (interaction user-system) might start.

Then, the resource analysis stage produces a document that describes for each resource its value as an end target (such as a database with personnel or proprietary information) or as an intermediate target (such as a file that stores access-permissions). These resources are the target of an exploit. From this stage, we derived the Impact Surface, as the set of coordinates where exploits or vulnerabilities might be possible.

Finally, the diagrams produced by the privilege analysis step is a further labeling of the previous documents with trust levels and labeling of interactions with delegation information. In this diagrams the Attack Vectors were derived. An Attack Vector is the sequence of transformations that allows control flow-data to go from a point in the attack surface to a point in the impact surface. At a later stage of this work we will introduce in depth the derived elements, how they are depicted in a graph, and we will propose an algorithm to use all of them for generating the security hints automatically.

We apply *AvA4cmi (Attack vector analyzer for complex middleware interrelationships) methodology* to two different workload management systems, namely, CrossBroker [18] and gLite WMS [8], those systems were previously assessed manually using FPVA. The AvA4cmi methodology was implemented and evaluated on both middlewares, with results that showed that the proposed methodology correlates previous manually found vulnerabilities with several attack vectors, and corroborates which middleware components should be assessed and why. These results are not sensitive to source code analysis, which make they independent from the middleware language programming.

This thesis is organized as follows:

Chapter 2: presents a general introduction to distributed systems, the architecture and security requirements of grid, cloud, and SCADA paradigms are outlined, and a review of relevant related work in vulnerability assessment topic is given.

Chapter 3: describes the First Principles Vulnerability Assessment, which corresponds to the vulnerability assessment methodology considered throughout this work. Later, it summarizes the Common Weakness Enumeration, and its weakness scoring framework used in this work.

Chapter 4: contains the proposed *AvA4cmi* methodology, its main elements (i.e., attack vector graphs, knowledge base, system attributes, rules), the algorithm to support and connect the codified expert knowledge within *AvA4cmi*, the results produced by the methodology, and its implementation details.

Chapter 5: first introduces the middleware CrossBroker, and gLite WMS. Then, it shows the assessment outcomes of both middleware with FPVA. Finally, a deep comparision is made of the results obtained when these two middlewares were assessed with the AvA4cmi methodology.

Chapter 6: summarizes the main conclusions derived from this thesis, outlining, in addition, current and future work.

CHAPTER 2

Introduction

Distributed systems belong to the rapidly changing field of computer sciences, nowadays, they have become more popular given the emergence of on demand computing paradigm Cloud, along with the computing paradigm Grid, as well as due to the SCADA industrial systems. However, although governmental, commercial, and academic research organizations might have collaborative or economical reasons to let users share CPU cycles, data, and resources across geographical and organizational boundaries, still they are unlikely to fully rely and trust such a distributed infrastructure until they can rely on the confidentiality, the integrity, the availability, and the privacy of its communications, data, resources, and the user information.

Traditional security measures concentrate on isolating systems and protecting resources with restrictive user policies. For instance, most organizations today deploy firewalls, intrusion detection systems, antivirus, and so on, around their computer networks to protect their sensitive and proprietary data. But security challenges in distributed systems   authentication, authorization, certification, and delegation   have not been exempt of issues at all, because distributed systems security is a multidimensional problem, and most existing projects in grid and cloud either lack or insufficiently addressed vulnerability assessment tasks, and usually it is an afterthought, even for well funded projects.

In Section 2.1 we introduce most important Grid concepts. Following this, Section 2.2 presents some Cloud concepts. In Section 2.3, we present a brief introduction to industrial distributed systems (SCADA) concepts. In section 2.4 we give an overview of the current status of the vulnerability asssessment projects. Finally, we will review the main contributions of this thesis in Section 2.5.

2.1 Grid

In the middle of 1990s the Grid computing [25] term was proposed for an analogous infrastructure of wide-area parallel and distributed computing, inspired by the electrical power grid s pervasiveness, ease of use, and reliability. A grid enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources, and specialized devices owned by different organizations for solving large-scale resource intensive problems in science, engineering, and commerce. Generally, for a system to be considered as a grid, it must meet the following criteria [22]:

(1) *"A grid coordinates resources that are not subject to centralized control and at the same time addresses the issues of security, policy, payment, membership, and so forth that arise in these settings."*

(2) *"A grid must use standard, open, general-purpose protocols and interfaces. These protocols address fundamental issues such as authentication, authorization, resource discovery, and resource access."*

(3) *"A grid delivers nontrivial quality service, i.e. it is able to meet complex user demands (e.g. response time, throughput, availability, security, etc.)."*

The development of the grid infrastructure has become the focus of a large community of researchers. The grid systems need to solve several challenges originating from inherent features of the grid [25]:

- *"Multiple administrative domains and autonomy. Grid resources are geographically distributed across multiple administrative domains and* *owned by different organizations. The autonomy of resource owners needs to be honored along with their local resource management and usage policies."*

- *"Heterogeneity. A grid involves a multiplicity of resources that are heterogeneous in nature and will encompass a vast range of technologies."*

- *"Scalability. A grid might grow from a few integrated resources to millions. This raises the problem of potential performance degradation as the size of the grid increases. Consequently, applications that require a large number of geographically located resources must be designed to be latency and bandwidth tolerant."*

- *"Dynamicity or adaptability. In a grid, resource failure is the rule rather than the exception. In fact, with so many resources in a grid, the probability of some resource failing is high. Resource managers or applications must tailor their behavior dynamically to use the available resources and services efficiently and effectively."*

The grid goes further than simply sharing resources and data. A grid enables new scientific collaboration methods, termed e-Science [34], that tackle large scale scientific problems. e-Science enables massively distributed computation, the sharing of huge data sets almost immediately, and cooperative scientific work to gather new results.

2.1.1 Grid Architecture

Typically, grid architectures are arranged into layers [3], where each layer builds on the services offered by the lower layer, in addition to interacting and co-operating with components at the same level. Figure 2.1 shows the architecture stack of a grid proposed by Foster et al. in The Anatomy of the Grid . It consists of five layers, fabric, connectivity, resource, collective, and application [27]:

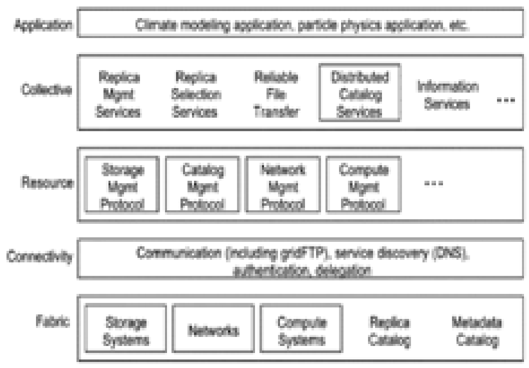


Figure 2.1: Grid Architecture[27]

- *"Fabric: The fabric layer defines the interface to local resources, which may be shared. These resources include computational resources, data storage, networks, catalogs, software modules, and other system resources."*

- *"Connectivity: The connectivity layer defines the basic communication and authentication protocols required for grid–specific networking–service transactions."*

- *"Resource: This layer uses the communication and security protocols (defined by the connectivity layer) to control secure negotiation, initiation, monitoring, accounting, and payment for the sharing of functions among individual resources. The resource layer calls the fabric layer functions to access and control local resources. This layer only handles individual resources, ignoring global states and atomic actions across the resource collection pool, which are the responsibility of the collective layer."*

- *"Collective: While the resource layer manages an individual resource, the collective layer is responsible for all global resource management and interaction with collections of resources. This protocol layer implements a wide variety of sharing behaviors using a small number of resource-layer and connectivity-layer protocols."*

- *"Application: The application layer enables the use of resources in a grid environment through various collaboration and resource access protocols."*

Grid Middleware

In order to take advantage from all the grid architecture, a Grid middleware is required, which provides the services needed to support a common set of applications in a grid ecosystem such as the one in the Figure 2.2. It hides the underlying infrastructure details and offers transparent access to the distributed resources, allowing collaborative efforts between organizations.

The middleware sits between the fabric and application layers of the grid architecture, keeping them loosely-coupled with a set of interfaces and protocols. In the bottom of the middleware is the myriad of underlying resources upon which the services are built (local operating systems, networks, file systems, etc.), and the top is where the applications are located.

Between the most commonly used grid middleware [15] are the Globus Toolkit [24], CONDOR [29], and gLite [70], which are in continuous development together with the most known grid infrastructure projects, EGI-EMI [40], OSG [55], and so on.

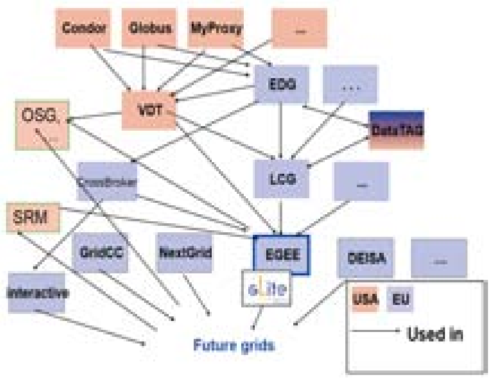


Figure 2.2: Grid Ecosystem [16]

2.1.2 Grid Security

Independently from the grid middleware or grid architecture implemented, one of the grid computing goals is making virtual organizations across one or more physical organizations (or administrative domains). These virtual organizations require common solutions for resource management, data management and access, application development environments, and information services.

One of the most significant challenge for Grid computing is to develop a comprehensive set of mechanisms and policies for securing the Grid; where users need to know if they are interacting with the right piece of software or human, and that their messages will not be modified or stolen as they traverse the virtual organization. Also, where users will often require the ability to prevent others from reading data that they have stored in the virtual organization.

In addition, while the virtual organization certainly could define some common security mechanisms and policies across the entire virtual organization, a site that provides resources to a Grid might now have to consider opening some access points that were closed in the process of securing the host or site so that the resource can be utilized by non-local members of the virtual organization. In short, users must trust the software infrastructure of the Grid to sufficiently prevent malicious activities, and viceversa.

The Grid security requirements [36, 35, 30, 23] can be grouped into several broad categories each with its own challenges:

Naming and authentication

The assignment of some identifier to a unique person, resource or other entity is called Naming, hence it can be used for authorization and auditing. Hence, it is taken some thought for defining a global Grid identifier. The X.509 names, derived from the X.500 standard, are commonly used by Grid software to provide global names for users and hosts [82]. Authentication in a computer environment is the process of associating a real world identity with a request to a machine. Authentication takes place when the connecting entity provides a unique-id and the authenticating agent can verify that the id legitimately represents the connecting entity.

In Grids, authentication across many domains is possible due to a loosely coupled approach, the Public Key Infrastructure (PKI) technology. The most common public key authentication protocol in use in Grids today is the Transport Layer Security (TLS) protocol [14], that was derived from the Secure Sockets Layer (SSL) v3 protocol [28]. TLS uses an X.509 public key certificate [82], which binds a multi-component meaningful name, called a Distinguished Name (DN), to a public key.

Communication

A secure communication requires the ability of two or more entities to establish a conversation with confidentiality, and integrity. To address this the parties in the communication have to authenticate each other, and the corresponding communication channel have to support integrity checking, and of course the confidentiality. So, integrity is typically provided using standard hash algorithms, and confidentiality is provided using encryption.

Different mechanisms exist for securing communication channels. Today, the most common way is the usage of asymmetric cryptography, instead of sharing a secret key. If the parties wish to communicate, they can publish their public keys in a place accessible for the other parties, so they can retrieve the public keys for using in asymmetric cryptography. In the Grid systems, the only public key published is from the Certification Authority (CA) that signed the individual public keys. This becomes a root of trust for identity in the PKI. Any public key signed by the trusted CA is trusted to represent a unique individual or entity. These keys can be used by protocols such as TLS or IPSec [31] to establish a symmetric session key that is known by both ends of the authenticated connection and used in all subsequent communications. On top of the TLS protocol is built the Grid Security Infrastructure (GSI), the principles of GSI are described in [26, 7].

Trust management

To have confidence that a party will behave in an expected manner even in the absence of ability to monitor or control it, is named Trust. The management of trust is the process of deciding what parties are to be trusted to do what actions. Trust management consists of defining the sources of authority for user identification, attribute assignment and possibly policy creation.

According by the GGF Grid Policy research group [30], a policy is a set of principals or rules that regulate the behavior of a system. External representations of a policy are highly desirable, in order to achieve flexibility, transparency and scalability of a system. Policies related to security may regulate trust, including delegation of trust, authentication, authorization, and levels of message or data integrity and confidentiality.

Delegation

In the different Grid usage scenarios is required that an agent impersonates a principal. When a user ask to a service to perform some operation on her behalf, the conventional approach is to grant unlimited delegation, which is to unconditionally grant the service the ability to act on behalf of the user.

Into general-purpose computational Grids, the services can not be wholly trusted by the users who wish to invoke them. This is a not reasonable approach for general-purpose Grids, and it is clearly not scalable.

The crucial issue for Grid delegation, is the determination of those privileges that should be granted by the user to the service and the circumstances under which those privileges are valid. It is clear that delegating too many privileges could lead to abuse, while delegating too few privileges could prevent a task to be completed.

Grid security requirements make it clear that vulnerability assessment does not belong to their potential. Despite being numerous articles in the literature on mechanisms and protocols (i.e., PKI, X.509, etc.) extensively studied, which implement Grid security requirements, they are not enough to fully protect Grid resources and services from both malicious users and actions. Proof of this are the reports of vulnerabilities found by the MIST group [51], in different well-known Grid middlewares.

2.2 Cloud

Security practitioners and researchers are continuously aware with news, and literature about the trends of information technologies. Many of these trends are not arriving alone, not to say all of them, so new trends also imply novel information technology threats. Cloud computing is one of these new trends. Cloud is *"a catch-all term that describes the evolutionary development of many existing technologies and approaches to computing that, at its most basic, separates application and information resources from the underlying infrastructure and mechanisms used to deliver them with the addition of elastic scale and the utility model of allocation"* [2].

Cloud enhances collaboration, agility, scale, availability and provides the potential for cost reduction through optimized and efficient computing. More specifically, cloud describes the use of a collection of distributed services, applications, information and infrastructure comprised of pools of compute, network, information and storage resources. These components can be rapidly orchestrated, provisioned, implemented and decommissioned using an on-demand utility-like model of allocation and consumption [54]. Cloud services are most often, but not always, utilized in conjunction with and enabled by virtualization technologies to provide dynamic integration, provisioning, orchestration, mobility and scale.

This model is architecturally similar to grid computing, but where grids are used for loosely coupled technical research computing applications, this new cloud model has been applied to Internet services [31]. Understanding how Cloud Computing architecture impacts security requires an understanding of cloud s principal characteristics, the manner in which cloud providers deliver and deploy services, how they are consumed and ultimately how they need to be safeguarded.

2.2.1 Cloud Architecture

Cloud services are based upon five principal characteristics that demonstrate their relation to, and differences from, traditional computing approaches [2]:

- *"Abstraction of Infrastructure: The compute, network and storage infrastructure resources are abstracted from the application and information resources as a function of service delivery. Where and by what physical resource that data is processed, transmitted, and stored on becomes largely opaque from the perspective of an application or services ability to deliver it. Infrastructure resources are generally pooled in order to deliver service regardless of the tenancy model employed – shared or dedicated. This abstraction is generally provided by means of high levels of virtualization at the chipset and operating system levels or enabled at the higher levels by heavily customized file systems, operating systems or communication protocols."*

- *"Resource Democratization: The abstraction of infrastructure yields the notion of resource democratization – whether infrastructure, applications, or information – and provides the capability for pooled resources to be* *made available and accessible to anyone or anything authorized to utilize them using standardized methods for doing so."*

- *"Services Oriented Architecture: As the abstraction of infrastructure from application and information yields well-defined and loosely-coupled resource democratization, the notion of utilizing these components in whole or part, alone or with integration, provides a services oriented architecture where resources may be accessed and utilized in a standard way. In this model, the focus is on the delivery of service and not the management of infrastructure."*

- *"Elasticity – Dynamism: The on-demand model of Cloud provisioning coupled with high levels of automation, virtualization, and ubiquitous, reliable and high-speed connectivity provides for the capability to rapidly expand or contract resource allocation to service definition and requirements using a self-service model that scales to as-needed capacity. Since resources are pooled, better utilization and service levels can be achieved."*

- *"Utility Model of Consumption & Allocation: The abstracted, democratized, service-oriented and elastic nature of Cloud combined with tight automation, orchestration, provisioning and self-service allows for dynamic allocation of resources based on any number of governing input parameters. Given the visibility at an atomic level, the consumption of resources can then be used to provide an "all-you-can-eat" but "pay-by-the-bite" metered utility-cost and usage model. This facilitates greater cost efficiencies and scale as well as manageable and predictive costs."*

Cloud service delivery models:

Three archetypal models and the derivative combinations thereof generally describe cloud service delivery. The three individual models are often referred to as the SPI Model , where SPI refers to Software, Platform and Infrastructure (as a service) respectively and are defined thusly [46]:

Software as a Service (SaaS): *"The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure and* *accessible from various client devices through a thin client interface such as a Web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure, network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings".*

Platform as a Service (PaaS): *"The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created applications using programming languages and tools supported by the provider (e.g., java, python, .Net). The consumer does not manage or control the underlying cloud infrastructure, network, servers, operating systems, or storage, but the consumer has control over the deployed applications and possibly application hosting environment configurations".*

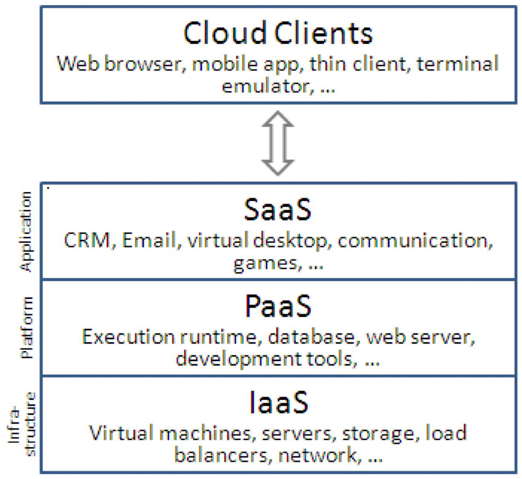

Cloud Clients
Web browser, mobile app, thin client, terminal emulator, ...
Application
SaaS
CRM, Email, virtual desktop, communication, games, ...
Platform
PaaS
Execution runtime, database, web server, development tools, ...
Infra-structure
IaaS
Virtual machines, servers, storage, load balancers, network, ...

Figure 2.3: Cloud Architecture [84]

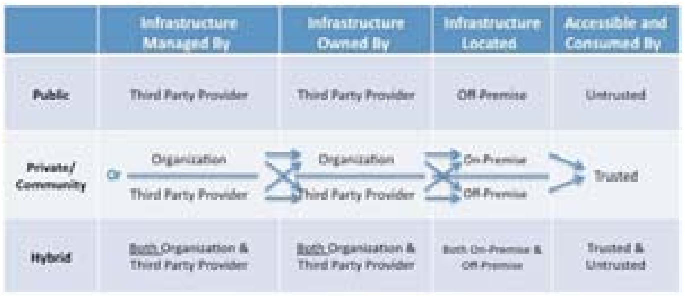
Infrastructure as a Service (IaaS): *"The capability provided to the consumer is to rent processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly select networking components (e.g., firewalls, load balancers)".*

Understanding the relationship and dependencies between these models is critical. IaaS is the foundation of all Cloud services with PaaS building upon IaaS, and SaaS – in turn – building upon PaaS (see Figure 2.3). Narrowing the scope or specific capabilities and functionality within each of the *aaS offerings or employing the functional coupling of services and capabilities across them may yield derivative classifications. For example "Storage as a Service" is a specific sub-offering within the IaaS family, "Database as a Service" may be seen as a derivative of PaaS, etc.

2.2.2 Cloud Security

According to a recent IDC cloud research [37], it shows that worldwide revenue from public IT cloud services exceeded \$21.5 billion in 2010 and will reach \$72.9 billion in 2015, representing a compound annual growth rate of 27.6%. This rapid growth rate is over four times the projected growth for the worldwide IT market as a whole (6.7%). Also, because of the need to reduce costs and enable IT responsiveness to business change is driving more and more applications, including critical ones, such as in the CERN [20], to various types of cloud platforms. From a security point of view, the cloud's economies of scale and flexibility are both a friend and a foe, due that massive concentrations of resources and data, so them present a more attractive target for the attackers.

Whether adopted in public, private or hybrid form, or delivered as IaaS, PaaS or SaaS, the cloud imposes unique and stringent security requirements. These security requirements are responsible for describing: Who manages it?, Who owns it?, Where it's located?, Who has access to it?, and How it's accessed? [44]. Figure 2.4 illustrates the summarization of these points.



	Infrastructure Managed By	Infrastructure Owned By	Infrastructure Located	Accessible and Consumed By
Public	Third Party Provider	Third Party Provider	Off-Premise	Untrusted
Private/ Community	Organization Or Third Party Provider	Organization Third Party Provider	On-Premise Off-Premise	Trusted
Hybrid	Both Organization & Third Party Provider	Both Organization & Third Party Provider	Both On-Premise & Off-Premise	Trusted & Untrusted

Figure 2.4: Cloud Service Management [44]

In accordance with [5, 33, 39, 2, 48], Cloud security requirements can be grouped into several broad categories each with its own challenges.

Robust security

Providing robust security means moving beyond a traditional perimeter-based approach to a layered model that ensures the proper isolation of data, even in a shared, multitenant cloud. This includes content protection at different layers in the cloud infrastructure, such as at the storage, hypervisor, virtual machine and database layers. It also requires mechanisms to provide confidentiality and access control. These may include encryption, obfuscation and key management, as well as isolation and containment, robust log management and an audit infrastructure.

Trust and assurance

To provide trust and assurance, the user (or company) needs to have confidence in the integrity of the complete cloud environment. This includes the physical data centers, hardware, software, people and processes employed by the provider. The service provider needs to establish an evidence-based trust architecture and control of the cloud environment, through adequate monitoring and reporting capabilities to ensure the customer of transparency around security events. This should include audit trails that help the customer meet internal and external demands for provable security, as well as automated notification and alerts that support the user s existing problem or incident management protocols so it can manage its total security profile. Collectively, these capabilities can assure the user of the operational quality and security of the cloud provider. Companies also need to take an active role in governing their cloud implementations and taking action on the information delivered by the provider.

Monitoring and governance

Monitoring and governance provide utilities that allow users to monitor the environment for security, as well as ensure compliance with other KPIs, such as performance and reliability. Using these utilities, customers should be able to perform these activities almost as well as they could in their own data centers. Just as importantly, these utilities allow users to take appropriate action based on the security information received from the provider. These actions might include shutting down an application that appears to be under attack or forcing the provider to tighten its procedures if critical updates or patches are not being applied on time.

Isolation

To ensure isolation within a multitenant environment, service providers often employ multiple virtual data centers, each on its own virtual LAN, to maintain customer data separation. For further security, each virtual data center can be configured into one or more trust clusters (each including, for example, separate Web servers, application servers and database zones), separated by demilitarized zones (DMZs) and virtual firewalls to ensure multitenancy security.

Confidentiality

Confidentiality is provided by encryption and/or obfuscation based on business requirements. Encryption might seem like the most complete and foolproof protection, but by completely obscuring the characteristics of the data, it can defeat indexing and search capabilities and increase the expense of filtering, querying or consolidation. Obfuscation retains enough properties of the data to allow these operations, as well as any other that relies on the semantics of data, while obscuring the data sufficiently to destroy its value if compromised. While obfuscation has traditionally been used as a one-way (nonreversible) masking technology, using obfuscation in the cloud to protect data requires the use of new architectures and approaches (such as tokenization) that enables access to the original non-obfuscated data as needed under tight security control.

Access control

Identity management and provisioning platforms ensure that only authorized users can see the appropriate applications and data. This needs to be backed by compliance and audit and log management, so that companies have a record of which users accessed (or tried to access) which resources, and when. In a cloud environment, access and identity management (which proves users are who they claim to be) is often provided through federated identity management that allows users to use their existing IT management systems in the cloud. Authentication, authorization and validation processes also help to ensure access and identity control. Providers may also need to ensure the integrity of data and messages (whether in transit or resident in the cloud) through strong authentication or other means to make sure data has not been compromised in transit.

So far, despite Cloud security requirements appear to be improved over Grid security requirements, they are not still fully aware about a thorough vulnerability assessment. Moreover, both Cloud and Grid share similiar security features, such as the PKI framework, and so on.

2.3 Supervisory Control And Data Acquisition

Nowadays, Supervisory Control And Data Acquisition (SCADA) systems, such as energy, oil, gas, water, chemical, and nuclear facilities has become very important, not only because they are vital for operating these critical infrastructures, but also because they are in the mirror of specialized attackers, which have been hired for some governments, due to the interest for fighting in most cases against countries which suppose are supporting worldwide terrorism groups. As a consequence, distributed and networked generation of SCADA systems (figure 2.5) are now being exposed to threats and vulnerabilities they have never been exposed to before, and to a much greater extent than earlier, despite the fact SCADA have been in use more than 40 years. SCADA development started before the widespread use of Internet, when the need for computer security mostly consisted of protecting the physical access to the system. In the last ten years, the number of connections to SCADA systems and the use of internet-based techniques have increased rapidly. SCADA systems have also moved from using proprietary protocols and software to using the same standards and solutions as administrative IT systems.

It is sometimes stated that while the prioritization in traditional information security is CIA (Confidently, Integrity, and Availability) the prioritization for SCADA systems is typically AIC (Availability, Integrity, Confidently) [73]. Moreover, many times is argued that SCADA system security is different from traditional IT security because of the environment the SCADA systems are used

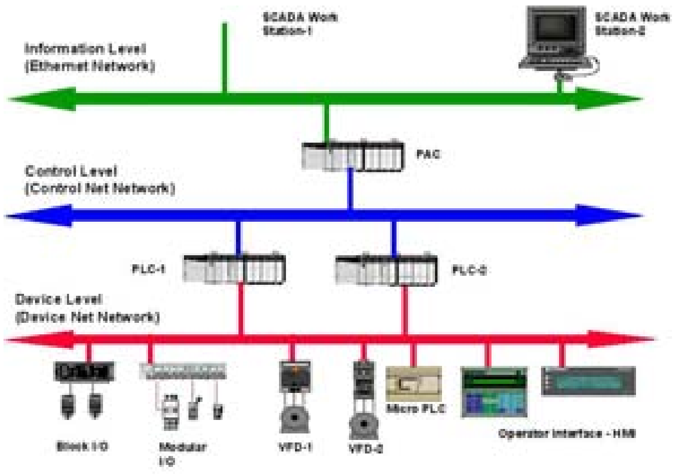


Figure 2.5: SCADA architecture[19]

within, and the requirements placed on them. As a consequence, a large number of recommendations, guidelines and regulations [6, 50, 53] that describe matters specifically related to SCADA security has been developed.

These standards (i.e. recommendations, guidelines and regulations) provide elaborate descriptions in terms of requirements that are placed on the system. For example, performance requirements: control systems are time-critical and real-time whereas information technology systems only require consistent response times and are not real-time [17]. Therefore, these general purpose security standards are used as basis for policies and practices applied to SCADA systems. And moreover, there has been no comprehensive analysis of how recommendations given in SCADA standards can encourage a vulnerability assessment process, and thus avoid new threats and attacks.

2.4 Vulnerability Assessment

As security has become almost imperative requirement today, with many computing projects deploying its own security features (but low standard compliance rates), which range from smartphones bank applications, to complex LHC Computing Grid middleware, vulnerability assessment is needed to protect most valuable assets of a system.

In traditional IT security, many definitions about what a vulnerability is can be found on [71, 38, 72], this thesis follows the NIST definition [72]:

> *A flaw or weakness in system security procedures, design, implementation, or internal controls that could be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system's security policy.*

Therefore, following the NIST criteria, a vulnerability assessment is:

> *The process by which we evaluate the susceptibility or risk factor of a component or system as a whole, to be totally or partially damaged by the impact of a threat.*

On the one hand, this process can be achieved by automated tools, which is quite attractive, due to vulnerability assessment is a repetitive, expensive, and time consuming task. But also, it requires qualified practitioner s knowledge to know how interpreting the tool s outcomes, because *"a fool with a tool is still a fool"* (Booch, Jacobson and Rumbaugh, developers of the Unified Modeling Language).

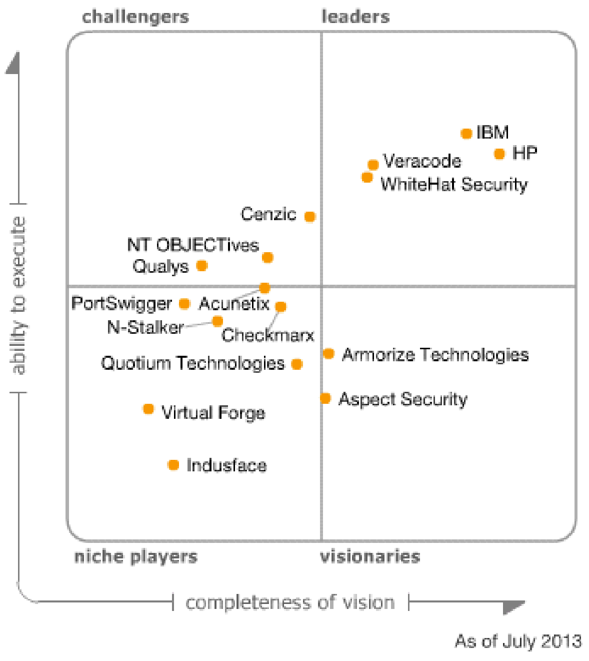


Figure 2.6: Gartner SAST Magic Quadrant 2013

The behaviour of the tools is to analyze an application from the inside out in a nonrunning state, finding common problems, analyzing application source code, byte code and binaries for coding and design conditions that are indicative of security vulnerabilities, matching it against either a list or a database of well known software security errors (vulnerability taxonomies will be discussed later in this section).

Eventually, these tools can add a sense of security during the software development life cycle (SDLC), but at the end overlook critical and complex vulnerabilities, even the best of these tools (figure 2.6) find only a small percentage of the serious critical and complex vulnerabilities [41]. That means, both open source and commercial automated tools for vulnerability assessment suffer from false negatives and false positives. A false positive is, a reported security vulnerability in a program that is not indeed a vulnerability. The problem with the false positives is that a great number can induce to many issues, e.g. if a developer faced with a long list of false positives he could miss out important data hidden in the list. Also, other problem with false positives is that project managers and team developers are more bowed to think that the tools are not useful and, therefore, they will not use them.

In any case, false negatives are worse, as in many other disciplines related with uncertainty. A false negative is defined as a vulnerability in the code which is not detected by the tool. The penalization associated with a false positive is the quantity of time used to check the results of the tool. The penalization associated with a false negative is much greater because, in this case, the vulnerability will remain in the code. If a tool does not find a concrete vulnerability (false negative), finding it by any other method is more difficult than discarding false positives.

On the other hand, vulnerability assessment also can be achieved throughout manual approach methodologies, which in turn can be used to reduce false negatives, and avoid false positives. Between the common methodologies are the Microsoft s threat modeling [74], the operationally critical threat, assess, and vulnerability evaluation (OCTAVE) [1] by the Carnegie Mellon University, the open source security testing methodology from the ISECOM institute [21], and the first principles vulnerability assessment (FPVA) [42] jointly developed by the University of Wisconsin-Madison & Universidad Autónoma de Barcelona. Below we present a brief description of the mentioned methodologies.

2.4.1 Microsoft Threat Modeling

Threat Modeling is a core element of the Microsoft security development lifecycle (SSDL)[74]. As part of the design phase of the SDL, it allows software architects to identify and mitigate potential security issues early, when they are relatively easy and cost-effective to resolve. Therefore, it helps reduce the total cost of development. Basically, the threat modeling allows you to systematically identify and rate the threats that are most likely to affect your system. By identifying and rating threats based on a solid understanding of the architecture and implementation of your application, you can address threats with appropriate countermeasures in a logical order, starting with the threats that present the greatest risk.

The Microsoft threat modeling process is composed by the following steps:

(1) Identify the valuable assets that your systems must protect.

(2) Use simple diagrams and tables to document the architecture of your application, including subsystems, trust boundaries, and data flow.

(3) Decompose the architecture of your application, including the underlying network and host infrastructure design, to create a security profile for the application. The aim of the security profile is to uncover vulnerabilities in the design, implementation, or deployment configuration of your application.

(4) Keeping the goals of an attacker in mind, and with knowledge of the architecture and potential vulnerabilities of your application, identify the threats that could affect the application. Two approaches are proposed for that: (a) STRIDE is the Microsoft acronym to categorize different threat types. STRIDE stands for Spoofing (using a false identity to gain access to the system), Tampering (unauthorized modification of data), Repudiation (denial of performed operations), Information disclosure (unwanted exposure of private data), Denial of service (make the application unavailable), and Elevation of privilege (gain privileged access to the application). (b) To use categorized threat lists for network, host and application threats.

(5) Document each threat using a common threat template that defines a core set of attributes to capture for each threat.

(6) Rate the threats to prioritize and address the most significant threats first. These threats present the biggest risk. The rating process weighs the probability of the threat against damage that could result should an attack occur. It might turn out that certain threats do not warrant any action when you compare the risk posed by the threat with the resulting mitigation costs.

There are some issues with Microsoft s methodology, after developing the architectural overview of the application, it applies a list of pre-defined and known possible threats, and tries to see if the application is vulnerable to these threats. As a consequence only known vulnerabilities may be detected, and the vulnerabilities detected may not refer to high value assets. And, additionaly Microsoft s threats identification suggest a brainstorming with the developers and test teams. These interactions could lead to a biased analysis and may result in threats going undetected.

2.4.2 OCTAVE

The Operationally Critical Threat, Asset, and Vulnerability Evaluation, define the essential components of a comprehensive, systematic, context-driven information security risk evaluation [1]. By following the OCTAVE Method, an organization can make information protection decisions based on risks to the CIA (Confidently, Integrity, and Availability) of critical information technology assets. The operational or business units and the IT department work together to address the information security needs of the enterprise. Using a three-phase approach, OCTAVE examines organizational and technology issues to assemble a comprehensive picture of the information security needs of the enterprise.

The Phases of OCTAVE are:

- Build Asset-Based Threat Profiles: This is an organizational evaluation. Key areas of expertise within the organization are examined to identify important information assets, the threats to those assets, the security requirements of the assets, what the organization is currently doing to protect its information assets (protection strategy practices), and weaknesses in organizational policies and practice (organizational vulnerabilities).

- Identify Infrastructure Vulnerabilities: This is an evaluation of the information infrastructure. The key operational components of the information technology infrastructure are examined for weaknesses (technology vulnerabilities) that can lead to unauthorized action.

- Develop Security Strategy and Plans: Risks are analyzed in this phase. The information generated by the organizational and information infrastructure evaluations (Phases 1 and 2) are analyzed to identify risks to the enterprise and to evaluate the risks based on their impact to the organization s mission. In addition, a protection strategy for the organization and mitigation plans addressing the highest priority risks is developed.

Like Microsoft, OCTAVE integrates people from different areas to the analysis, and thus the results may be biased too. And being a methodology focused on the assets that are important for an organization, like information, systems, software, hardware and people, makes it more general methodology in scope.

2.4.3 OSSTM

The Open Source Security Testing Methodology is provided by the ISECOM institute [21], it is an open source methodology, and a peer-reviewed methodology for performing security tests and metrics. OSSTMM encompasses any interaction, with any asset within the whole operating security environment, including the physical components of security measures as well. The methodology mandates that all the threats must be considered possible, even if not probable. The methodology scope is organized in three channels: COMSEC (communications security), PHYSSEC (physical security), and SPECSEC (spectrum security). Channels are the means of interacting with assets, where an asset is defined as anything of value to the owner. The three main channels are split into 5 sub-channels: human, physical, wireless communication, data networks, and telecommunications. The OSSTM follows a four point process to perform security tests:

(1) Induction: What can we tell about the target from its environment? How does it behave in that environment? If the target is not influenced by its environment, that s interesting too.

(2) Inquest: What signals (emanations) does the target give off? Investigate any tracks or indicators of those emanations. A system or process generally leaves a signature of interactions with its environment.

(3) Interaction: What happens when you poke it? This point calls for echo tests, including expected and unexpected interactions with the target, to trigger responses.

(4) Intervention: How far will it bend before it breaks? Intervene with the resources the target needs, like electricity, or meddle with its interactions with other systems to understand the extremes under which it can continue operating.

OSSTMM describes 17 modules to analyze each of the sub-channels. Consequently, the tester has to perform 17*5 = 85 analyses before being able to write the final report. And being considered not a corporative but an operational methodology, it does not explain how security tests should be performed. It its more focused on which items need to be tested, what to do before, during, and after a security test, and how to measure the results.

2.4.4 FPVA

First Principles Vulnerability Assessment [42] is a primarily analyst-centric (manual) approach to assessment, whose aim is to focus the analyst s attention on the parts of the software system and its resources that are mostly likely to contain vulnerabilities that would provide access to high-value assets. FPVA proceeds in five steps:

(1) Architecture analysis: identifies key structural components in a middleware system, including modules, threads, processes, and hosts.

(2) Resource analysis: identifies the key resources accessed by each component, and the operations supported on those resources.

(3) Privilege and Trust analysis: identifies the trust assumptions about each component, answering such questions as how are they protected or who can access them.

(4) Component analysis: examines each component in depth. This step is guided by information obtained in the first three steps, helping to prioritize the work.

(5) Results Dissemination: includes reporting the vulnerabilities to the development team. After the vulnerabilities are fixed FPVA suggests that notice of vulnerabilities be released in an abbreviated form when the new version of the software is released.

This approach has a couple of advantages. First, it allows us to find new vulnerabilities, not just exploits based on those that were previously discovered. Second, when a vulnerability is discovered, it is likely to be a serious one whose remediation is of high priority. The above are a twofold reason quite significant, whereby we have considered FPVA to develop this work, and it will be detailed later in the next chapter.

Revised both the tools and the methodologies, we introduce the vulnerability taxonomies. The aim of defining a vulnerability taxonomy is to organize sets of security rules that can be used to help software developers and security practitioners understand the kinds of errors that have an impact on security. Because today developers are not yet half aware of the several ways they can introduce security problems into their work, thus vulnerability taxonomies should provide tangible benefits to the computer security community. In the following subsections we reviewed the most used taxonomies in vulnerability assessment, in order to achieve the goals of this research work.

2.4.5 The 7 Pernicious Kingdoms

McGraw et al. [45] developed a taxonomy based on specific type of coding errors (i.e. a phylum that is for example a illegal pointer value) and a collection of phyla called a kingdom that share a common theme (for example input validation). The 7 Pernicious Kingdoms taxonomy has the following classification:

- Input Validation and Representation
- Api Abuse
- Security Features

- Time and State
- Errors
- Code Quality
- Encapsulation
- Environment

For example under input validation and representation you might have buffer overflows, cross site scripting (XSS), SQL injection and many others, under API abuse you might have unsafe string manipulation APIs, under time and state you might have TOCTOU (time to check time to execute), etc. The first seven kingdoms are associated with security defects in source code, while the last one describes security issues outside the actual code.

The McGraw classification does not aim to be stringent from the classification point of view but rather to be useful. According to McGraw the classification is not theoretically complete and can change but also points out that can serve well a classification for errors (i.e. security flaws) to be part of both a manual and automatic code review (i.e. static code parsers).

2.4.6 CLASP

The Comprehensive, Lightweight Application Security Process [83] is a root cause and a high-level classification, it identifies 104 underlying problem types that form the basis of security vulnerabilities in application source code, providing a well organized and structured approach for moving security concerns into the early stages of the software development life cycle (SDLC), whenever possible. This taxonomy has three perspectives: How(genesis) the problem entered the system, When(time of introduction) the problem entered the system, and Where (location) the problem manifested, and depending the point of introduction into the SDLC, CLASP can be classified in a hierarchical view:

- Level 1: Identify Range and Type of Errors. Part of level 1 you might have buffer overflows (introduced during Requirements, Design and Implementation), Command Injection (design and implementation), and Double Free (implementation).

Level 1.1: Identify Environment Problems. Part of level 1.1 you might have resource exhaustion (design and implementation).

Level 1.2: Identify Synchronization and timing errors, TOCTOU, and race conditions (design and implementation).

Level 1.3: Identify Protocol Errors. Misuse of cryptography (design).

Level 1.4: Identify Generic logic Errors. Performing a chroot without a chdir (implementation).

CLASP is effective in determining root causes of security flaws with emphasis when in the SDLC is actually might originate helping architects and developers to build security into the SDLC. CLASP taxonomy struggles to provide a reliable lexicon for security flaws classification but some of the issues in the taxonomy cannot be actually classified as security problems.

2.4.7 OWASP Top 10

The OWASP [68] Top 10 is determined by the Open Web Application Security Project (OWASP), it represents a broad consensus on the most critical web application security flaws. The errors on the top 10 occur frequently in web applications, are often easy to find, and easy to exploit. The current 2013 top 10 is as follows:

- A1 Injection [58].
- A2 Broken Authentication and Session Management [60].
- A3 Cross-Site Scripting (XSS) [61].
- A4 Insecure Direct Object References [62].
- A5 Security Misconfiguration [63].
- A6 Sensitive Data Exposure [64].
- A7 Missing Function Level Access Control [65].
- A8 Cross-Site Request Forgery (CSRF) [66].
- A9 Using Components with Known Vulnerabilities [67].

- A10 Unvalidated Redirects and Forwards [59].

The latest top 10 (2013) is based on 8 datasets from 7 firms specialized in application security (4 consulting companies and 3 tool/SaaS vendors). This datasets span over 500,000 vulnerabilities across hundreds of organizations and thousands of applications. The top 10 items are selected and prioritized according to this prevalence data, in combination with consensus estimates of exploitability, detectability, and impact estimates.

Such as the other taxonomies, the OWASP Top 10 has never looked substantially different. There is not much difference between the 2003 and 2013 list, usually one category is added, one category is removed, and a few categories are repositioned. Thus, despite web applications are becoming more common components from middlewares, a top 10 classification is not enough for the purposes of this work.

2.4.8 Common Weakness Enumeration (CWE)

The CWE [77] is a list of common software weaknesses. In this context security weaknesses are flaws, faults, bugs, vulnerabilities, and other errors in any stage of SDLC that if left unaddressed could result in computers and networks being vulnerable to attack. CWE could be roughly described as a three tiered taxonomy, the tier one consist of the full CWE List (hundreds of nodes); the tier two consists of descriptive affinity groupings of individual CWEs, which is called the Development View; and the tier three consists of high level groupings (pillars) of the intermediate nodes to define strategic classes of vulnerabilities, which is called the Research View.

CWE encompass a large portion of the CVE [76] list s (15,000 CVE names), it also includes detail, and classification structure from a diverse set of other industry and academic clasifications including the McGraw/Fortify 7 Kingdoms taxonomy; OWASP Top 10 ; and Secure Software s CLASP project, among others. Due to which gives CWE a substantial weight to be chosen as our taxonomy, more specifically the *CWE Research View approach* is used in this research work. To further support our decision, we quote the words of McGraw:

"Christey's and Martin's put vulnerability taxonomy at the other end

of the ambiguity spectrum – the vulnerability categories are much more specific than in any of the other taxonomies. An attempt to draw parallels between theoretical attacks and vulnerabilities known in practice is an important contribution and a big step forward from most of the earlier taxonomies."

In addition, considering when a vulnerability assessment is performed over a system and because of the high volume of weaknesses reported, a method for identifying which of these dangerous weaknesses would be most harmful to a particular organization is needed, given the intended use of a specific piece of the system within that organization. To address this need, in this work we use a modified version of the Common Weakness Scoring System (CWSS) [78], which is provided as part of the CWE project.

CWE and CWSS will be described in detail in the next chapter.

2.5 Contributions

The developing of a vulnerability assessment in a distributed system arises from a set of challenges that have not been addressed completely before. Most of the vulnerability assessment tools and techniques are devoted to source code review and/or application during the development life cycle. The high interoperability between middleware components is not addressed by current tools or techniques for systematically hinting where, and why to deploy an assessment. In this work we propose a comprehensive solution that deals with both the high interoperability between middleware components and focusing the analyst attention on the most likely attacks on the highest value middleware assets. The main contributions of this work are the following:

(1) The definition of a methodology *AvA4cmi* focusing on the guidance towards a more comprehensive and accurate vulnerability assessment. This methodology deals with the high interoperability between middleware components for sistematically hinting where, and why to deploy an assessment.

(2) The definition and design of attack vector graphs that allow both the insertion and use of the derived element by MIST research group (i.e.,

impact surface), as well as the well known terms attack surface and attack vector, conforming to the proposed methodology. The attack vectors graphs are depicted as an extension of GraphML, an XML-based file format for graphs.

(3) An algorithm (formal method) that traverse attack vectors using knowledge about the middleware, generated with the application of FPVA, and the expertise codified in CWE for focusing the analyst attention on the most likely attacks on the highest value system assets.

(4) An experimental study where we compare the benefits of the proposed methodology against previous manual vulnerability assessment in distributed systems. We also deliver the outcomes of the study as prioritized and hierarchized list of likely weaknesses, which can lead to exploitable vulnerabilities.

2.6 Conclusions

In this chapter we have introduced the background material which is required to read the remainder of this work. Cloud and Grid computing are important platforms for the next generation of either scientific or commercial applications, where new threats arise due to the large number of highest value assets. Additionally, we introduced the distributed and networked generation of SCADA systems, where these critical infrastructures are in the mirror of specialized attackers. The vulnerability assessment is an essential part for the security of distributed systems, which is not a de-facto standar for their current security mechanisms.

Vulnerability assessment has become almost imperative requirement today with most recent projects deploying its own security features. We outlined some of the methodologies in vulnerability assessment and we reviewed the different approaches for vulnerability classification. Among these we found FPVA and CWE, a methodology and clasiffication approach on which this work is based.

CHAPTER 3

FPVA & CWE: Foundations

In this chapter, we make a depth presentation of the vulnerability assessment methodology, and the vulnerability classification system followed in work. In Section 3.1., we present *First Principles Vulnerability Assessment (FPVA)*, the step by step methodology to perform a vulnerability assessment, a summary of results achieved by FPVA, and its key accomplishments. Then, Section 3.2 presents the *Common Weakness Enumeration (CWE)*, and its scoring framework, the *Common Weakness Scoring System (CWSS)*.

3.1 First Principles Vulnerability Assessment

The approach called First Principles Vulnerability Assessment (FPVA) was developed by the *Middleware Security and Testing (MIST)* [51] research group, conformed by the Computer Sciences Departments of the University of Wisconsin and the Universidad Autónoma de Barcelona.

FPVA evaluates the security of a system in depth. Rather than working from common vulnerabilities, the starting point for FPVA is to identify high value assets in a system, such as components (e.g., processes or parts of processes) and resources (e.g., configuration files, databases, connections, and devices) whose exploitation offer the greatest potential for damage by an attacker. From these components and resources, FPVA works outwards to discover execution paths through the code that might exploit them. As we have mentioned before, this approach has a couple of advantages. First, it allows us to find new vulnerabilities, not just vulnerabilities based on those that were previously discovered. Second, when a vulnerability is discovered, it is likely to be a serious one whose remediation is of high priority.

As it is explained in [42], FPVA starts with an architectural analysis of the source code, identifying the key components in a distributed system. It then goes on to identify the resources associated with each component, the privilege level of each component, the value of each resource, how the components interact, and how trust is delegated between components. The results of these steps are documented in clear diagrams that provide a roadmap for the last stage of the analysis, the manual code inspection. In addition, the results of this step can also form the basis for a risk assessment of the system, identifying which parts of the system are most immediately in need of evaluation. After these steps, FPVA uses code inspection techniques on the critical parts of the code. The FPVA analysis strategy targets the high value assets in a system and focuses attention on the parts of the system that are vulnerable to not only unauthorized entry, but unauthorized entry that can be exploited. In addition to the analysis task, there needs to be a way to integrate the detection and repair of security flaws into the SDLC and release process. The SDLC process must now include vulnerability reporting, release integration, and a policy on public dissemination of the vulnerabilities.

FPVA have been successfully applied to several large and widely-used middleware systems, such as Condor [52], a high-throughput scheduling system; Storage Resource Broker (SRB) [9], a data grid management system; Crossbroker [18], a Grid resource management for interactive and parallel applications; the gLite WMS [8], a workload management system, among others. In these analyses, significant vulnerabilities were found, software development teams were educated to the key issues, and the resulting software was made more resistant to attacks.

When evaluating a methodology such as FPVA, the big question arises: Why not use automated vulnerability assessment tools? To addres the question, in

2009, the MIST group surveyed security practitioners in academia, industry, and government laboratories to identify which tools were consider best of breed in automated vulnerability assessment. Repeatedly, two commercials software packages were named, Coverity Prevent [12] and Fortify Source Code Analyzer [11].

To evaluate the power of these tools and better understand the FPVA approach, MIST group compared the results of the largest assessment activity (on Condor) to the results gathered from applying these automated tools [41]. As a result of these studies, MIST group demonstrated that the automated tools found few of the vulnerabilities that FPVA had identified in Condor (i.e., had significant false negatives) and identified many problems that were either not exploitable vulnerabilities (i.e., had many false positives).

Current FPVA and MIST group effort is just the beginning of a longer term research to develop more effective assessment techniques. MIST is using the experiences with these techniques to help design tools that will simplify the task of manual assessment. In addition, MIST is working to develop a formal characterization of the vulnerabilities FPVA have found in an attempt to develop improved automated detection algorithms that would include most of these vulnerabilities.

3.1.1 FPVA Methodology Steps

First Principles Vulnerability Assessment can characterize the steps of architectural, resource, privilege, and interaction analysis as a narrowing processing that produces a focused code analysis. These analysis steps can help to identify more complex vulnerabilities that are based on the interaction of multiple system components and are not amenable to local code analysis.

FPVA have found several vulnerabilities that are caused because a component (process) is allowed to write a file that will be later read by another component. The read or write operation by itself does not appear harmful, but how the data was created or used in another part of the code could allow an exploit to occur. Without a more global view of the analysis, such problems are difficult to find. There was nothing to indicate that this was a problem at the time the file was created, but it became dangerous at the time (and place) it was used.

FPVA is divided in five main steps:

Architectural Analysis

The first step is to identify the major structural components of the system, including modules, threads, processes, and hosts. For each of these components, it then identifies the way in which they interact, both with each other and with users. Interactions are particularly important as they can provide a basis for understand how trust is delegated through the system. The artifact produced at this stage is a document that diagrams the structure of the system and the interactions amongst the different components, and with the end users. Figure 3.1 and Figure 3.2 show the architectural analysis diagrams for the middlewares CrossBroker and gLite WMS.

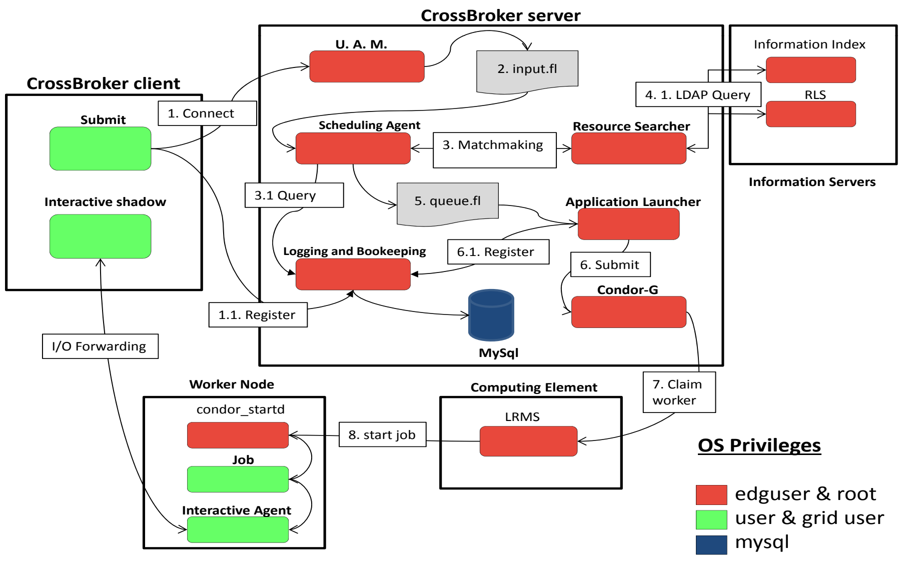


Figure 3.1: FPVA Architecture Diagram of CrossBroker

Resource Analysis

The second step is to identify the key resources accessed by each component, and the operations supported on those resources. Resources include elements such as hosts, files, databases, logs, and devices. These resources are often the target of an exploit. For each resource, it describes its value as an end target (such as a database with personnel or proprietary information) or as an intermediate

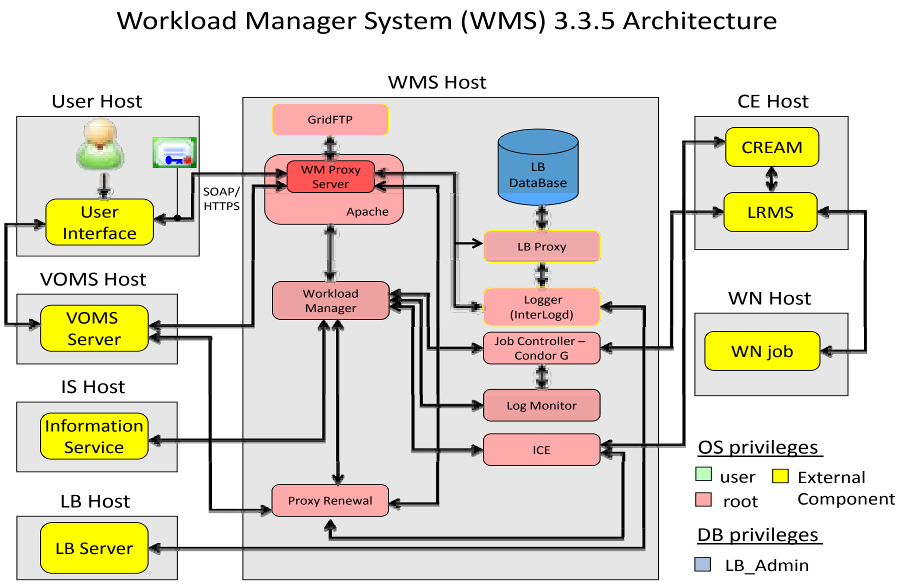


Figure 3.2: FPVA Architecture Diagram of gLite WMS

target (such as a file that stores access-permissions). The artifact produced at this stage is an annotation of the architectural diagrams with resource descriptions. Figure 3.3 and Figure 3.4 show the resource analysis diagrams for the middlewares CrossBroker and gLite WMS.

Privilege and Trust Analysis

The third step identifies the trust assumptions about each component, answering such questions as how are they protected and who can access them? For example, a code component running on a client s computer is completely open to modification, while a component running in a locked computer room has a higher degree of trust. Trust evaluation is also based on the hardware and software security surrounding the component. Associated with trust is describing the privilege level at which each executable component runs. The privilege levels control the extent of access for each component and, in the case of exploitation, the extent of damage that it can accomplish directly. A complex but crucial part of trust and privilege analysis is evaluating trust delegation. By combining the information from the first two steps, FPVA determines what

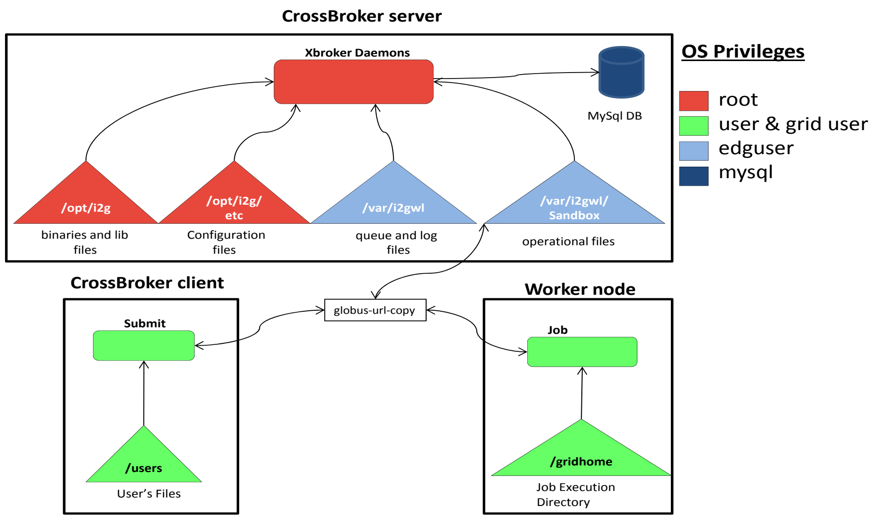


Figure 3.3: FPVA Resources Diagram of CrossBroker

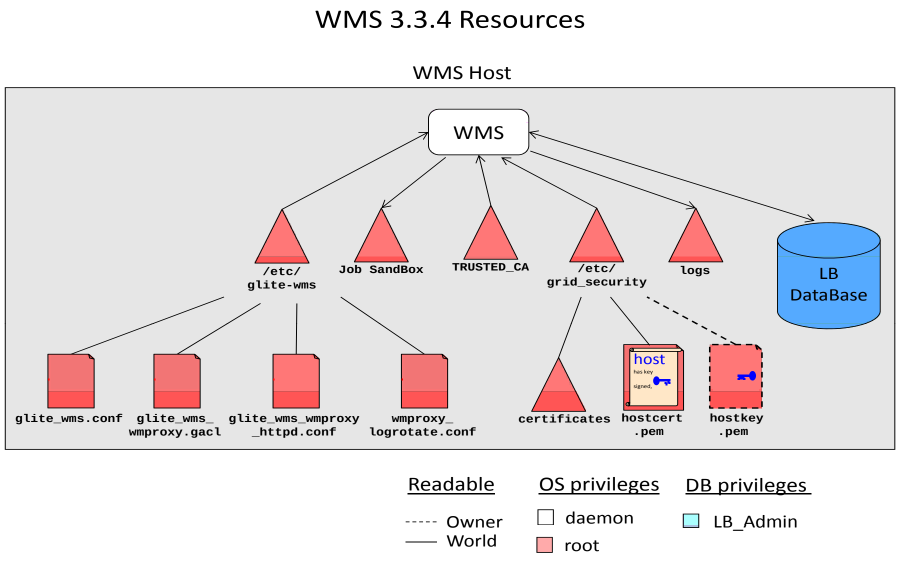


Figure 3.4: FPVA Resources Diagram of gLite WMS

operations a component will execute on behalf of another component. The artifact produced at this stage is a further labeling of the basic diagrams with trust levels and labeling of interactions with delegation information. For the sake of simplicity, the Privilege and Trust diagrams for CrossBroker and gLite WMS coincide with the diagrams in the first two steps.

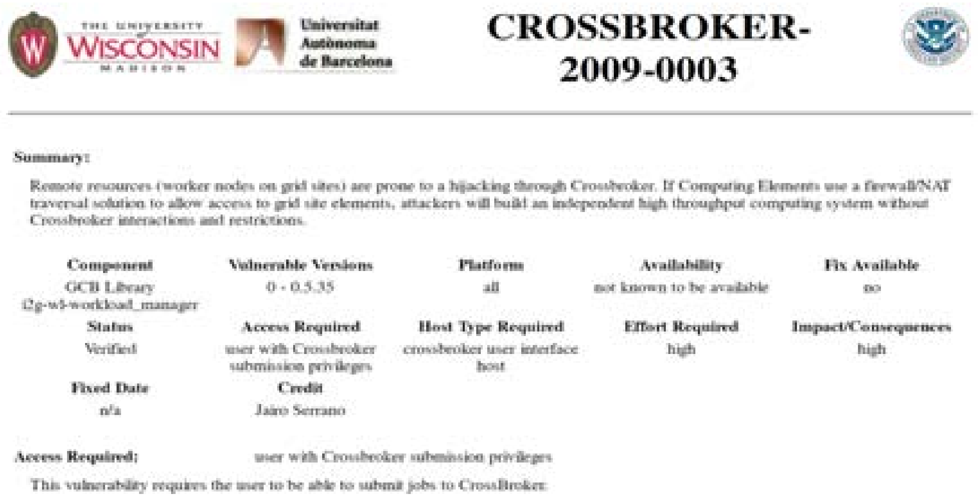
Component Analysis

The fourth step is to examine each component in depth. For large systems, a line by line manual examination of the code is infeasible, even for a well-funded effort. A key aspect of FPVA technique is that this step is guided by information obtained in the first three steps, helping to prioritize the work so that high value assets are evaluated first. The work in this step can be accelerated by automated scanning tools. While these tools can provide valuable information, they are subject to false positives, and even when they indicate real flaws, they often cannot tell whether the flaw is exploitable and, even if it is, whether it will allow serious damage. In addition, these tools typically work most effectively on a local basis, so flaws based on inappropriate trust boundaries or delegation of authority are not likely to be found. Therefore, these tools work best in the context of a vulnerability analysis process. The artifacts produced by this step are vulnerability reports, perhaps with suggested fixes, to be provided to the software developers. Figure 3.5 and Figure 3.6 show examples of the vulnerability reports for the middlewares CrossBroker and gLite CREAM.

Dissemination of Results

Once vulnerabilities are reported, the obvious next step is for the developers to fix them. However, once fixed, they are confronted with some questions that can be difficult to answer in a collaborative and often open source world. Some questions are: How do we integrate the update into our release stream? (Do we put out a special release or part of an upcoming one?) When do we announce the existence of the vulnerability? How much detail do we provide initially? If the project is open source, how do we deal with groups that are slow to update? Should there be some community-wide mechanism to time announcements and releases?

In the following Table 3.1 derived from [69], we present the entire analisys by the MIST group, and their corresponding expertise and results with FPVA:



THE UNIVERSITY of WISCONSIN MADISON Universitat Autònoma de Barcelona

CROSSBROKER-2009-0003

Summary:

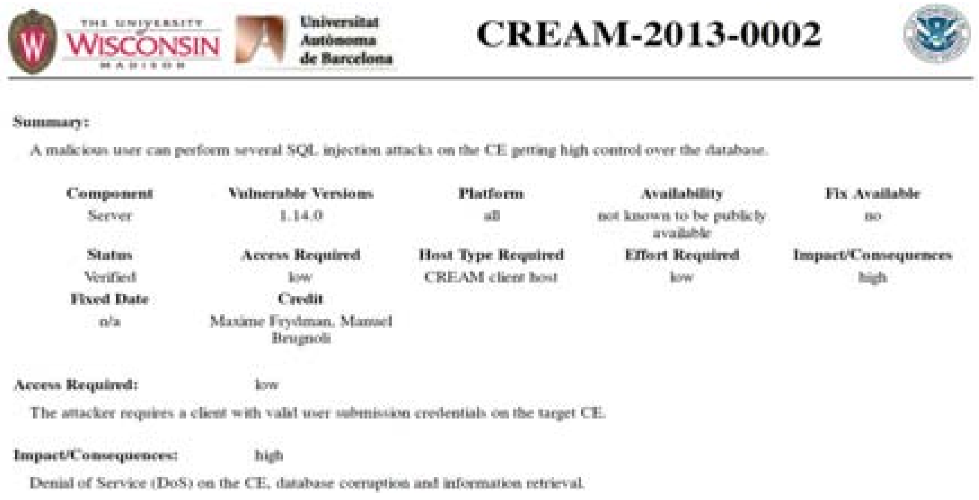
Remote resources (worker nodes on grid sites) are prone to a hijacking through Crossbroker. If Computing Elements use a firewall/NAT traversal solution to allow access to grid site elements, attackers will build an independent high throughput computing system without Crossbroker interactions and restrictions.

Component	**Vulnerable Versions**	**Platform**	**Availability**	**Fix Available**
GCB Library i2g-wl-workload_manager	0 - 0.5.35	all	not known to be available	no
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	user with Crossbroker submission privileges	crossbroker user interface host	high	high
Fixed Date	**Credit**			
n/a	Jairo Serrano			

Access Required: user with Crossbroker submission privileges

This vulnerability requires the user to be able to submit jobs to CrossBroker.

Figure 3.5: FPVA Vulnerability Report of CrossBroker



THE UNIVERSITY of WISCONSIN MADISON Universitat Autònoma de Barcelona

CREAM-2013-0002

Summary:

A malicious user can perform several SQL injection attacks on the CE getting high control over the database.

Component	**Vulnerable Versions**	**Platform**	**Availability**	**Fix Available**
Server	1.14.0	all	not known to be publicly available	no
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	low	CREAM client host	low	high
Fixed Date	**Credit**			
n/a	Maxime Frydman, Manuel Brugnoli			

Access Required: low

The attacker requires a client with valid user submission credentials on the target CE.

Impact/Consequences: high

Denial of Service (DoS) on the CE, database corruption and information retrieval.

Figure 3.6: FPVA Vulnerability Report of gLite CREAM

Condor	University of Wisconsin Batch queuing workload management system **15 Vulnerabilities, 600KLOC of C/C++**
SRB	San Diego Supercomputing Center Storage Resource Broker - data grid **5 Vulnerabilities, 280KLOC of C**
MyProxy	National Center for Supercomputing Applications Credential management system **5 Vulnerabilities, 25KLOC of C**
glExec	Nikhef Identity mapping service **5 Vulnerabilities, 48KLOC of C**
Gratia Condor Probe	FNAL and Open Science Grid Feeds Condor usage into Gratia accounting system **3 Vulnerabilities, 1.7KLOC of Perl/Bash**
Condor Quill	University of Wisconsin DBMS Storage of Condor operational and historical data **6 Vulnerabilities, 7.9KLOC of C/C++**
Wireshark	Wireshark.org Network protocol analyzer **2 Vulnerabilities, 2400KLOC of C**
Condor PrivSep	University of Wisconsin Restricted identity switching module **2 Vulnerabilities, 21KLOC of C/C++**
gLite CREAM	INFN Computing Resource Execution And Management **5 Vulnerabilities, 35KLOC of C/C++**
CrossBroker	Universidad Autónoma de Barcelona Workload management for parallel and interactive jobs **4 Vulnerabilities, 97KLOC of C/C++/Python**
gLite WMS	INFN gLite Workload management system **0 Vulnerabilities,728KLOC of C/C++/Python**
Argus	HIP, INFN, Nikhef, SWITCH gLite authorization service **0 Vulnerabilities, 42KLOC of Java/C**
VOMS Core	INFN Virtual organization management system **1 Vulnerability, 161KLOC of C/C++**
iRODS	DICE Data management system **9 Vulnerabilities, 285KLOC of C/C++**
Google Chrome	Google Web browser **In progess, 2396KLOC of C/C++**

Table 3.1: FPVA Experience.

3.1.2 FPVA Accomplishments

In summary, the key accomplishments for FPVA include:

- An architecture and resource based analysis that is not dependent on known vulnerabilities.

- A clear reminder that assessment must be an independent activity, done by analysts separate from the software development team. In addition, assessment must be part of the normal software development lifecycle.

- A demonstration of the strengths and weaknesses of automated assessment tools by comparing their results against those of a thorough FPVA study (and not just comparing the automated tools against each other).

- Several assessment activities of key middleware systems, resulting in significant improvements in the security of these systems, and similar improvements by creating a security aware atmosphere among the software developers.

- A foundation and new approach for future research into improved automated vulnerability assessment tools.

Despite all the experience gathered and the vulnerabilities found with FPVA, we realized for all FPVA analyzed middlewares that there is a gap between the three initial steps and the manual source code inspection. The security practitioner should provide certain expertise about the kind of security problems that the systems may present during the component analysis step (e.g. depending on the language used the analyst should look for different kind of vulnerabilities), and be creative enough as to discover new vulnerabilities. Hence, this gap directly affects the quality of the vulnerability assessment, because security flaws may be overlooked due to either incomplete analyst knowledge or insufficient time for an in-depth analysis. This gap has been addressed by the *AvA4cmi* methodology proposed in this work.

After presenting FPVA, we proceed with the presentation of CWE.

3.2 Common Weakness Enumeration

The Common Weakness Enumeration (CWE) [77], developed by Christey and Martin, supported by multiple stakeholders, and currently maintained by the MITRE Corporation under the sponsorship of the National Cyber Security Division of the Department of Homeland Security (DHS USA), provides a standard language for discussing, finding and dealing with the causes of software security vulnerabilities as they are found in code, design, or system architecture.

Also, CWE can be defined as a community developed formal list of common software weaknesses, which serves as a common language for describing software security weaknesses, a standard measuring stick for software security tools targeting these vulnerabilities, and as a baseline standard for weakness identification, mitigation, and prevention efforts. Leveraging the diverse thinking on this topic from academia, the commercial sector, and government, CWE unites the most valuable breadth and depth of content and structure to serve as a unified standard.

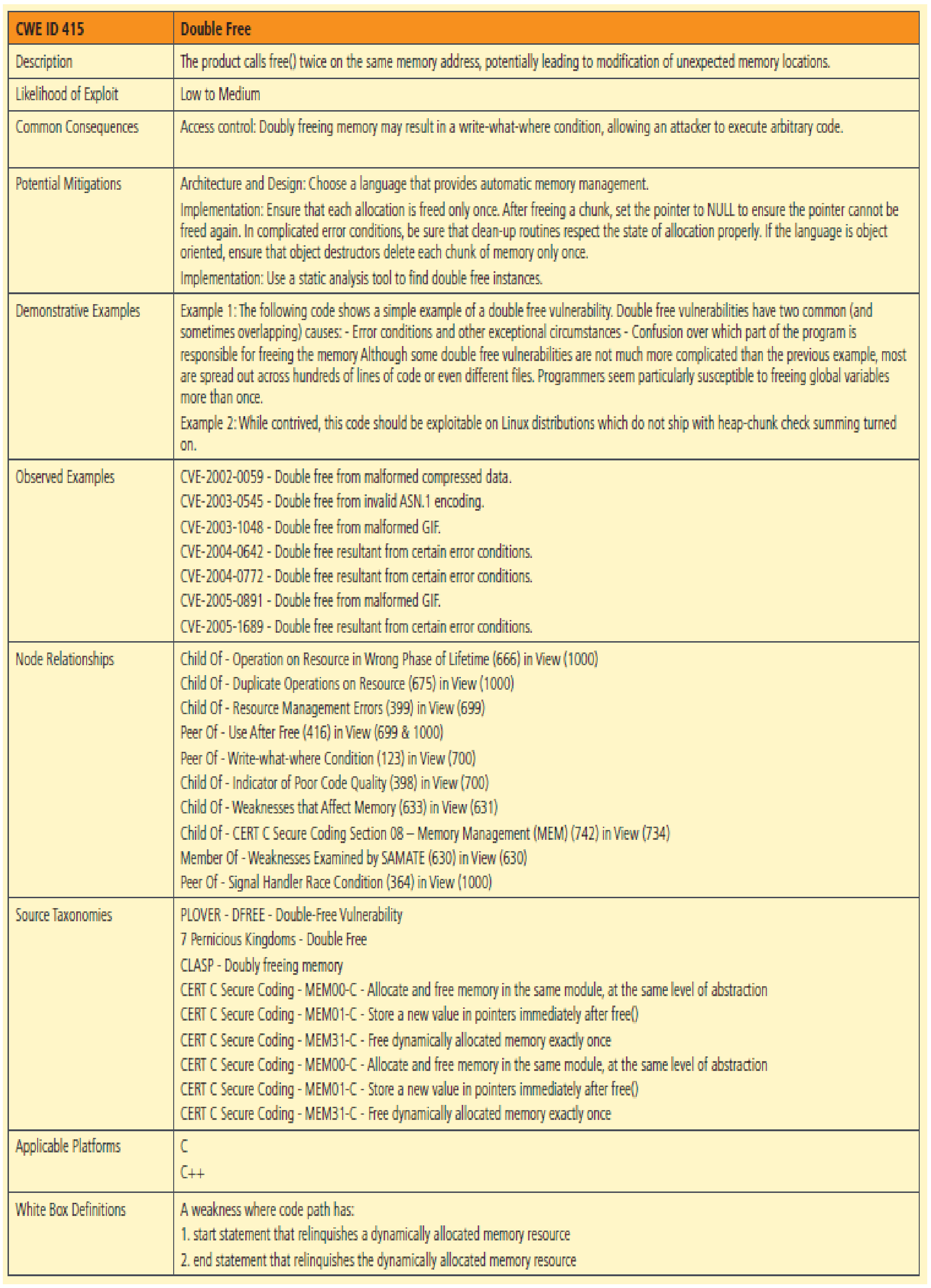
As it is explained in [77], each individual CWE represents a single vulnerability type, and includes a list of associated details. An example of a single CWE weakness and its details are depicted in Figure 3.7.

From these CWE details, and other not included in the example, such as an extended description, references, consequence scope, consequence notes, and time of introduction, we gathered the most useful information to build our methodology, which will be discussed in the next chapter.

All individual CWEs (over 714 separate weaknesses) are held within a hierarchical structure from two main organizational views that allows for multiple levels of abstraction. CWEs located at higher levels of the structure (i.e. Configuration) provide a broad overview of a vulnerability type and can have many children CWEs associated with them. CWEs at deeper levels in the structure (i.e. Cross Site Scripting) provide a finer granularity and usually have fewer or no children CWEs.

CWE has a glossary of terms, in order to better understand the organizational views we introduce some of them:

- ***View***: a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) and Graphs



CWE ID 415	**Double Free**
Description	The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.
Likelihood of Exploit	Low to Medium
Common Consequences	Access control: Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.
Potential Mitigations	Architecture and Design: Choose a language that provides automatic memory management. Implementation: Ensure that each allocation is freed only once. After freeing a chunk, set the pointer to NULL to ensure the pointer cannot be freed again. In complicated error conditions, be sure that clean-up routines respect the state of allocation properly. If the language is object oriented, ensure that object destructors delete each chunk of memory only once. Implementation: Use a static analysis tool to find double free instances.
Demonstrative Examples	Example 1: The following code shows a simple example of a double free vulnerability. Double free vulnerabilities have two common (and sometimes overlapping) causes: - Error conditions and other exceptional circumstances - Confusion over which part of the program is responsible for freeing the memory Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once. Example 2: While contrived, this code should be exploitable on Linux distributions which do not ship with heap-chunk check summing turned on.
Observed Examples	CVE-2002-0059 - Double free from malformed compressed data. CVE-2003-0545 - Double free from invalid ASN.1 encoding. CVE-2003-1048 - Double free from malformed GIF. CVE-2004-0642 - Double free resultant from certain error conditions. CVE-2004-0772 - Double free resultant from certain error conditions. CVE-2005-0891 - Double free from malformed GIF. CVE-2005-1689 - Double free resultant from certain error conditions.
Node Relationships	Child Of - Operation on Resource in Wrong Phase of Lifetime (666) in View (1000) Child Of - Duplicate Operations on Resource (675) in View (1000) Child Of - Resource Management Errors (399) in View (699) Peer Of - Use After Free (416) in View (699 & 1000) Peer Of - Write-what-where Condition (123) in View (700) Child Of - Indicator of Poor Code Quality (398) in View (700) Child Of - Weaknesses that Affect Memory (633) in View (631) Child Of - CERT C Secure Coding Section 08 – Memory Management (MEM) (742) in View (734) Member Of - Weaknesses Examined by SAMATE (630) in View (630) Peer Of - Signal Handler Race Condition (364) in View (1000)
Source Taxonomies	PLOVER - DFREE - Double-Free Vulnerability 7 Pernicious Kingdoms - Double Free CLASP - Doubly freeing memory CERT C Secure Coding - MEM00-C - Allocate and free memory in the same module, at the same level of abstraction CERT C Secure Coding - MEM01-C - Store a new value in pointers immediately after free() CERT C Secure Coding - MEM31-C - Free dynamically allocated memory exactly once CERT C Secure Coding - MEM00-C - Allocate and free memory in the same module, at the same level of abstraction CERT C Secure Coding - MEM01-C - Store a new value in pointers immediately after free() CERT C Secure Coding - MEM31-C - Free dynamically allocated memory exactly once
Applicable Platforms	C C++
White Box Definitions	A weakness where code path has: 1. start statement that relinquishes a dynamically allocated memory resource 2. end statement that relinquishes the dynamically allocated memory resource

Figure 3.7: CWE Weakness details example, taken from [81]

(containing relationships between entries).

- ***Category***: a CWE entry that contains a set of other entries that share a common characteristic.
- ***Weakness***: a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC.
- ***Compound Element***: an Entry that closely associates two or more CWE entries. The CWE team s research has shown that vulnerabilities often can be described in terms of the interaction or co-occurrence of two or more weaknesses.
- ***Chain***: a Compound Element that is a sequence of two or more separate weaknesses that can be closely linked together within software. One weakness, X, can directly create the conditions that are necessary to cause another weakness, Y, to enter a vulnerable condition.
- ***Composite***: a Compound Element that consists of two or more distinct weaknesses, in which all weaknesses must be present at the same time in order for a potential vulnerability to arise.
- ***Pillar***: a top-level entry, equivalent to kingdoms in Seven Pernicious Kingdoms [45].

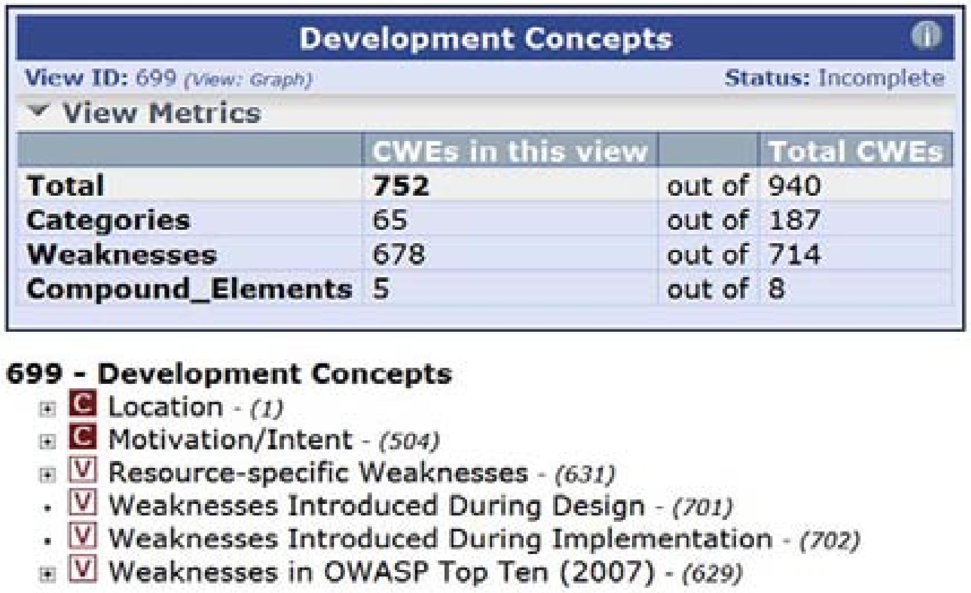
The two main organizational views of CWE are:

Development View

The Development View organizes weaknesses around concepts that are frequently used or encountered in software development. Thus, this view can align closely with the perspectives of developers, educators, and assessment vendors. Some goals for the Development View include flexible navigation (useful categories), familiarity (similarity to other taxonomies), and coverage (identifying all low-level CWE weaknesses).

The higher level nodes in the Development View (Figure 3.8) borrow heavily from the structure used by Seven Pernicious Kingdoms, the categories of errors in

CLASP, etc., in order to achieve familiarity. As a result, the Development view can be readily understood by users who are already familiar with any of these taxonomies.


Development Concepts

View ID: 699 (View: Graph) Status: Incomplete

View Metrics

	CWEs in this view		Total CWEs
Total	752	out of	940
Categories	65	out of	187
Weaknesses	678	out of	714
Compound_Elements	5	out of	8

699 - Development Concepts
- Location - (1)
- Motivation/Intent - (504)
- Resource-specific Weaknesses - (631)
- Weaknesses Introduced During Design - (701)
- Weaknesses Introduced During Implementation - (702)
- Weaknesses in OWASP Top Ten (2007) - (629)

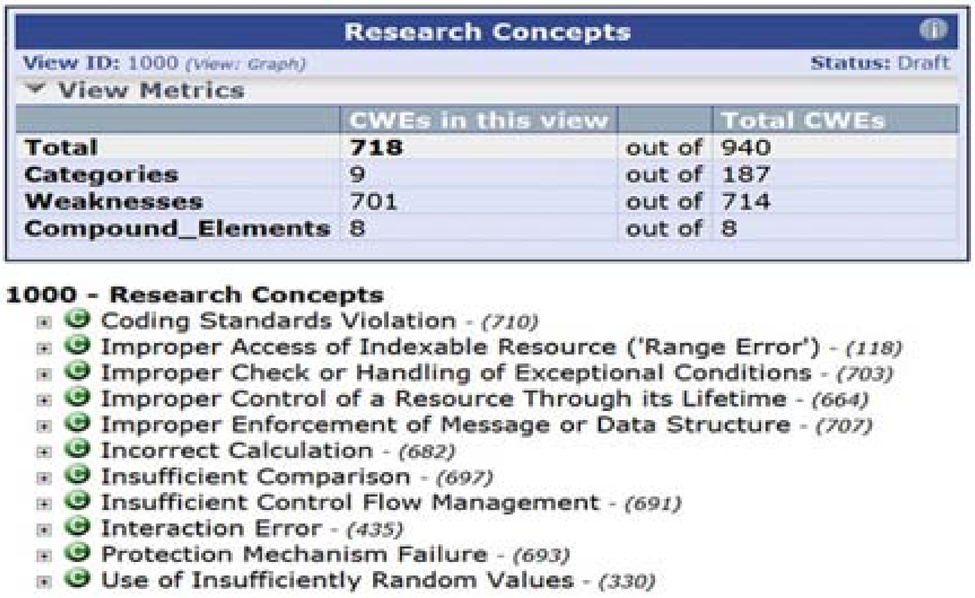
Figure 3.8: CWE Development View Hierarchy, taken from [80]

Regarding to navigation, these taxonomies and ongoing CWE have introduced a variety of different categories for weaknesses. This provides a mechanism for CWE users to navigate through the large number of weaknesses that are covered, from a variety of perspectives. Many of these categories define groups of weaknesses based on common attributes such as language, resource, or consequence. Categories include pointer issues, mobile code issues, error handling, data handling, time and state, temporary file problems, weaknesses in J2EE and ASP environments, web problems, and so on.

Resarch View

The Research View was developed to address the need of a view based solely on weaknesses and their abstractions, to provide a more formal mechanism for classifying weaknesses, and performing vulnerability research. It classifies weaknesses in a way that largely ignores how they can be detected, where they appear in code, and when they are introduced in the SDLC. Accordingly, it avoids capturing relationships based on specific language, environment, technology, framework, frequency of occurrence, impact, and mitigation; since these relationships are convenient for many users, they are captured in Development View. By doing so, Research View has been able to concentrate on canonical factors that make each weakness unique.

The Research view is mainly organized (Figure 3.9) accordingly to abstractions of software behaviors and the resources that are manipulated by those behaviors, which aligns with MITRE s research into vulnerability theory. In addition to classification, the Research view explicitly models the interdependencies between weaknesses, which have not been a formal part of past classification efforts. The main examples are chains and composites.



Research Concepts

View ID: 1000 (View: Graph) Status: Draft

View Metrics

	CWEs in this view		Total CWEs
Total	718	out of	940
Categories	9	out of	187
Weaknesses	701	out of	714
Compound_Elements	8	out of	8

1000 - Research Concepts
- Coding Standards Violation - (710)
- Improper Access of Indexable Resource ('Range Error') - (118)
- Improper Check or Handling of Exceptional Conditions - (703)
- Improper Control of a Resource Through its Lifetime - (664)
- Improper Enforcement of Message or Data Structure - (707)
- Incorrect Calculation - (682)
- Insufficient Comparison - (697)
- Insufficient Control Flow Management - (691)
- Interaction Error - (435)
- Protection Mechanism Failure - (693)
- Use of Insufficiently Random Values - (330)

Figure 3.9: CWE Research View Hierarchy, taken from [80]

The Research View uses multiple (11 to be precise) deep hierarchies as its organization structure, with more levels of abstraction than other classification schemes. Ideally, the abstraction is only on weakness to weakness relationships, with minimal overlap and no categories. Thus, weaknesses would be presented from the lowest levels all the way to roots of the tree, and each member weakness would cover a single error. The top-level entries of each of the hierarchies are called Pillars.

Because of their relevance to this work, as it is explained in [79], the eleven (11) pillars of the Research view, and their details are depicted below:

CWE–118: Improper Access of Indexable Resource (Range Error)

- Description: The software does not restrict or incorrectly restricts operations within the boundaries of a resource that is accessed using an index or pointer, such as memory or files.
- Time of Introduction: Architecture and Design, Implementation, and Operation.
- Languages: All
- Common Consequences

  Scope: Other.

  Technical Impact: Varies by context.
- Relationship: Parent of CWE 119

CWE–330: Use of Insufficiently Random Values

- Description: The software may use insufficiently random numbers or values in a security context that depends on unpredictable numbers
- Extended Description: When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information
- Time of Introduction: Architecture and Design, and Implementation
- Languages: All
- Common Consequences

  Scope: Confidentiality, Access Control

  Technical Impact: Bypass protection mechanism, Gain privileges/assume identity

- Likelihood of Exploit: Medium → High
- Observed Examples: CVE-2009-3278, CVE-2009-3238, CVE-2009-2367
- Relationship: Parent of CWE 329, CWE 331, CWE 334, etc.

CWE–435: Interaction Error

- Description: An interaction error occurs when two entities work correctly when running independently, but they interact in unexpected ways when they are run together.
- Extended Description: This could apply to products, systems, components, etc.
- Time of Introduction: Architecture and Design, Implementation, and Operation.
- Languages: All
- Common Consequences

  Scope: Integrity

  Technical Impact: Unexpected state, Varies by context
- Relationship: Parent of CWE 188, CWE 436, CWE 439, etc.

CWE–664: Improper Control of a Resource Through its Lifetime

- Description: The software does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release.
- Extended Description: Resources often have explicit instructions on how to be created, used and destroyed. When software does not follow these instructions, it can lead to unexpected behaviors and potentially exploitable states.
- Time of Introduction: Implementation

- Common Consequences

  Scope: Other

  Technical Impact: Other

- Relationship Parent of CWE 221, CWE 284, CWE 400, etc.

CWE–682: Incorrect Calculation

- Description: The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.
- Extended Description: When software performs a security-critical calculation incorrectly, it might lead to incorrect resource allocations, incorrect privilege assignments, or failed comparisons among other things. Many of the direct results of an incorrect calculation can lead to even larger problems such as failed protection mechanisms or even arbitrary code execution.
- Time of Introduction: Architecture, Design, and Implementation.
- Languages: All
- Common Consequences

  Scope: Availability, Integrity, Confidentiality, Access Control

  Technical Impact: DoS: crash / exit / restart, Bypass protection mechanism, Gain privileges / assume identity

- Likelihood of Exploit: High.
- Relationship: Parent of CWE 128, CWE 131, CWE 190, etc.

CWE–691: Insufficient Control Flow Management

- Description: The code does not sufficiently manage its control flow during execution, creating conditions in which the control flow can be modified in unexpected ways.
- Time of Introduction: Architecture, Design, and Implementation.

- Languages: All
- Common Consequences

  Scope: Other

  Technical Impact: Alter execution logic.
- Relationship: Parent of CWE 94, CWE 362, CWE 430, etc.

CWE–693: Protection Mechanism Failure

- Description: The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.
- Extended Description: This weakness covers three distinct situations. A missing protection mechanism occurs when the application does not define any mechanism against a certain class of attack. An insufficient protection mechanism might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended. Finally, an ignored mechanism occurs when a mechanism is available and in active use within the product, but the developer has not applied it in some code path.
- Time of Introduction: Architecture, Design, Implementation, and Operation.
- Languages: All
- Common Consequences

  Scope: Access Control

  Technical Impact: Bypass protection mechanism.
- Relationship: Parent of CWE 20, CWE 183, CWE 345, etc.

CWE–697: Insufficient Comparison

- Description: The software compares two entities in a security-relevant context, but the comparison is insufficient, which may lead to resultant weaknesses.

- Extended Description: This weakness class covers several possibilities, the comparison checks one factor incorrectly; the comparison should consider multiple factors, but it does not check some of those factors at all.
- Time of Introduction: Implementation
- Common Consequences

  Scope: Other

  Technical Impact: Other
- Relationship: Parent of CWE 185, CWE 478, CWE 184, etc.

CWE–703: Improper Check or Handling of Exceptional Conditions

- Description: The software does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of the software.
- Time of Introduction: Architecture, Design, Implementation, and Operation.
- Language: All
- Common Consequences

  Scope: Confidentiality, Availability, Integrity

  Technical Impact: Read application data; DoS: crash / exit / restart; Unexpected state
- Relationship: Parent of CWE 393, CWE 274, CWE 167, etc.

CWE–707: Improper Enforcement of Message or Data Structure

- Description: The software does not enforce or incorrectly enforces that structured messages or data are well-formed before being read from an upstream component or sent to a downstream component.
- Extended Description: If a message is malformed it may cause the message to be incorrectly interpreted. This weakness typically applies in cases where the product prepares a control message that another process must act on,

such as a command or query, and malicious input that was intended as data, can enter the control plane instead. However, this weakness also applies to more general cases where there are not always control implications.

- Time of Introduction: Architecture, Design, and Implementation.
- Language: All
- Common Consequences

  Scope: Other

  Technical Impact: Other
- Relationship: Parent of CWE 116, CWE 138, CWE 170, etc.

CWE–710: Coding Standards Violation

- Description: The software does not follow certain coding rules for development, which can lead to resultant weaknesses or increase the severity of the associated vulnerabilities.
- Time of Introduction: Architecture, Design, and Implementation.
- Language: All
- Common Consequences

  Scope: Other

  Technical Impact: Other
- Relationship: Parent of CWE 657, CWE 912, CWE 758, etc.

With the taxonomy described, and due to the high volume of reported weaknesses when a vulnerability assessment is performed, it is needed to provide a standard method for identifying which of these dangerous weaknesses would be most harmful to a particular organization, given the intended use of a specific piece of software within that organization. To address this need, the CWE project provided the Common Weakness Scoring System.

3.2.1 Common Weakness Scoring System (CWSS)

CWSS [78] is the mechanism provided by the CWE project for scoring weaknesses *"in a particular piece of software in a consistent, flexible, open manner while incorporating knowledge of the context of the software's use in the particular business and reflecting the impacts of weaknesses against that context"*. CWSS is being developed as a collaborative, community-based effort so that it addresses the needs of stakeholders across government, academia, and industry. It is aimed at providing a consistent approach for tools and services prioritizing their static and dynamic analysis findings

In the current version of the CWSS, as it is explained in [78], a weakness score can be calculated across three metric groups, along with their 18 different factors, each one with different values and weight:

The Base Finding group

It captures the inherent risk of the weakness, confidence in the accuracy of the finding, and strength of controls. The Base Finding metric group consists of the following factors:

- Technical Impact (TI)
- Acquired Privilege (AP)
- Acquired Privilege Layer (AL)
- Internal Control Effectiveness (IC)
- Finding Confidence (FC)

The Technical Impact is the potential result that can be produced by the weakness, assuming that the weakness can be successfully reached and exploited. This is expressed in terms that are more fine-grained than confidentiality, integrity, and availability.

Its corresponding values and weights are described in Table 3.2:

Value	**Weight**	**Value**	**Weight**
Critical	1.0	High	0.9
Medium	0.6	Low	0.3
None	0.0	Default	0.6
Unknown	0.5	Not Applicable	1.0
Quantified			

Table 3.2: CWSS Technical Impact Set of Values

For example, a Critical value for Technical Impact factor means complete control over the software, the data it processes, and the environment in which it runs (e.g. the host system), to the point where operations cannot take place. Default is the median of the weights for Critical, High, Medium, Low, and None.

The Acquired Privilege The Acquired Privilege identifies the type of privileges that are obtained by an entity who can successfully exploit the weakness. In some cases, the acquired privileges may be the same as the required privileges, which implies either horizontal privilege escalation (e.g. from one unprivileged user to another), or privilege escalation within a sandbox, such as an virtual machine user who can escape to the host machine.

Its corresponding values and weights are described in Table 3.3:

Value	**Weight**	**Value**	**Weight**
Administrator	1.0	Partially Privileged User	0.9
Regular User	0.7	Guest	0.6
None	0.1	Default	0.7
Unknown	1.0	Not Applicable	1.0

Table 3.3: CWSS Acquired Privilege Set of Values.

For example, an Administrator value for the Acquired Privilege factor means the entity has administrator, root, SYSTEM, or equivalent privileges that imply full control over the software or the underlying OS.

The Acquired Privilege Layer identifies the operational layer to which the entity gains access if the weakness can be successfully exploited.

Its corresponding values and weights are described in Table 3.4:

Value	**Weight**	**Value**	**Weight**
Application	1.0	System	0.9
Network	0.7	Enterprise	1.0
None	0.1	Default	0.9
Unknown	0.5	Not Applicable	1.0

Table 3.4: CWSS Acquired Privilege Layer Set of Values.

For example, an Application value for the Acquired Privilege Layer factor means the entity must be able to have access to an affected application.

The Internal Control Effectiveness is a control, protection mechanism, or mitigation that has been explicitly built into the software (whether through architecture, design, or implementation). Internal Control Effectiveness measures the ability of the control to render the weakness unable to be exploited by an attacker. For example, an input validation routine that restricts input length to 15 characters might be moderately effective against XSS attacks by reducing the size of the XSS exploit that can be attempted.

Its corresponding values and weights are described in Table 3.5:

Value	**Weight**	**Value**	**Weight**
None	1.0	Limited	0.9
Moderate	0.7	Indirect	0.5
Best Available	0.3	Complete	0.0
Default	0.6	Unknown	0.5
Not Applicable	1.0		

Table 3.5: CWSS Internal Control Effectiveness Set of Values.

For example, a None value for the Internal Control Effectiveness factor means No controls exist.

The Finding Confidence is the confidence that the reported issue: (1) is a weakness, and (2) can be triggered or utilized by an attacker.

Its corresponding values and weights are described in Table 3.6:

Value	**Weight**	**Value**	**Weight**
Proven True	1.0	Proven Locally True	0.8
Proven False	0.0	Default	0.8
Unknown	0.5	Not Applicable	1.0
Quantified			

Table 3.6: CWSS Finding Confidence Set of Values.

For example, a Proven True value for the Finding Confidence factor means the weakness is reachable by the attacker.

The Attack Surface group

It captures the barriers that an attacker must cross in order to exploit the weakness. The Attack Surface metric group consists of the following factors:

- Required Privilege (RP)
- Required Privilege Layer (RL)
- Access Vector (AV)
- Authentication Strength (AS)
- Authentication Instances (AI)
- Level of Interaction (IN)
- Deployment Scope (SC)

The Required Privilege identifies the type of privileges required for an entity to reach the code/functionality that contains the weakness.

Its corresponding values and weights are described in Table 3.7:

Value	Weight	Value	Weight
None	1.0	Guest	0.9
Regular User	0.7	Partially-Privileged User	0.6
Administrator	0.1	Default	0.7
Unknown	0.5	Not Applicable	1.0

Table 3.7: CWSS Required Privilege Set of Values

For example, a None value for the Required Privilege factor means No privileges are required, i.e., a web-based search engine may not require any privileges for an entity to enter a search term and view results.

The Required Privilege Layer identifies the type of privileges required for an entity to reach the code/functionality that contains the weakness.

Its corresponding values and weights are described in Table 3.8:

Value	Weight	Value	Weight
System	0.9	Application	1.0
Network	0.7	Enterprise	1.0
Default	0.9	Unknown	0.5
Not Applicable	1.0		

Table 3.8: CWSS Required Privilege Layer Set of Values

For example, a System value for the Required Privilege Layer factor means the entity must have access to, or control of, a system or physical host.

The Access Vector identifies the channel through which an entity must communicate to reach the code or functionality that contains the weakness.

Its corresponding values and weights are described in Table 3.9:

Value	**Weight**	**Value**	**Weight**
Internet	1.0	Intranet	0.8
Private Network	0.8	Adjacent Network	0.7
Local	0.5	Physical	0.2
Default	0.75	Unknown	0.5

Table 3.9: CWSS Attack Vector Set of Values

For example, an Internet value for the Attack Vector factor means an attacker must have access to the Internet to reach the weakness.

The Authentication Strength overs the strength of the authentication routine that protects the code/functionality that contains the weakness.

Its corresponding values and weights are described in Table 3.10:

Value	**Weight**	**Value**	**Weight**
Strong	0.7	Moderate	0.8
Weak	0.9	None	1.0
Default	0.85	Unknown	0.5
Not Applicable	1.0		

Table 3.10: CWSS Authentication Strength Set of Values

For example, a Strong value for the Authentication Strength factor means the weakness requires strongest-available methods to tie the entity to a real-world identity, such as hardward-based tokens, and/or multi-factor authentication.

The Authentication Instances covers the number of distinct instances of authentication that an entity must perform to reach the weakness.

Its corresponding values and weights are described in Table 3.11:

Value	**Weight**	**Value**	**Weight**
None	1.0	Single	0.8
Multiple	0.5	Default	0.8
Unknown	0.5	Not Applicable	1.0

Table 3.11: CWSS Authentication Instances Set of Values

For example, a None value for the Authentication Instances factor means No authentication is required.

The Level of Interaction covers the actions that are required by the human victim(s) to enable a successful attack to take place.

Its corresponding values and weights are described in Table 3.12:

Value	**Weight**	**Value**	**Weight**
Automated	1.0	Limited/Typical	0.9
Moderate	0.8	Opportunistic	0.3
High	0.1	No interaction	0.0
Default	0.55	Unknown	0.5
Not Applicable	1.0		

Table 3.12: CWSS Level of Interaction Set of Values

For example, an Automated value for the Level of Interaction factor means No human interaction is required.

The Deployment Scope identifies whether the weakness is present in all deployable instances of the software, or if it is limited to a subset of platforms and/or configurations.

Its corresponding values and weights are described in Table 3.13:

Value	**Weight**	**Value**	**Weight**
All	1.0	Moderate	0.9
Rare	0.5	Potentially Reachable	0.1
Default	0.7	Unknown	0.5
Not Applicable	1.0	Quantified	

Table 3.13: CWSS Deployment Scope Set of Values

For example, an All value for the Deployment Scope factor means it is Present in all platforms or configurations.

The Environmental group

It includes factors that may be specific to a particular operational context, such as business impact, likelihood of exploit, and existence of external controls. The Environmental metric group consistes of the following factors:

- Business Impact (BI)
- Likelihood of Discovery (DI)
- Likelihood of Exploit (EX)
- External Control Effectiveness (EC)
- Remediation Effort (RE)
- Prevalence (P)

The Business Impact describes the potential impact to the business or mission if the weakness can be successfully exploited.

Its corresponding values and weights are described in Table 3.14:

Value	**Weight**	**Value**	**Weight**
Critical	1.0	High	0.9
Medium	0.6	Low	0.3
None	0.0	Default	0.6
Unknown	0.5	Not Applicable	1.0
Quantified			

Table 3.14: CWSS Business Impact Set of Values

For example, a Medium value for the Business Impact factor means the business/mission would be affected, but without extensive damage to regular operations.

The Likelihood of Discovery is the likelihood that an attacker can discover the weakness.

Its corresponding values and weights are described in Table 3.15:

Value	**Weight**	**Value**	**Weight**
High	1.0	Medium	0.6
Low	0.2	Default	0.6
Unknown	0.5	Not Applicable	1.0
Quantified			

Table 3.15: CWSS Likelihood of Discovery Set of Values

For example, a High value for the Likelihood of Discovery factor means it is very likely that an attacker can discover the weakness quickly and with little effort using simple techniques, without access to source code or other artifacts that simplify weakness detection.

The Likelihood of Exploit is the likelihood that, if the weakness is discovered, an attacker with the required privileges/authentication/access would be able to successfully exploit it.

Its corresponding values and weights are described in Table 3.16:

Value	**Weight**	**Value**	**Weight**
High	1.0	Medium	0.6
Low	0.2	None	0.0
Default	0.6	Unknown	0.5
Not Applicable	1.0	Quantified	

Table 3.16: CWSS Likelihood of Exploit Set of Values

For example, a High value for the Likelihood of Exploit factor means it is highly likely that an attacker would target this weakness successfully, with a reliable exploit that is easy to develop.

The External Control Effectiveness is the capability of controls or mitigations outside of the software that may render the weakness unable to be reached or triggered by an attacker.

Its corresponding values and weights are described in Table 3.17:

Value	Weight	Value	Weight
None	1.0	Limited	0.9
Moderate	0.7	Indirect	0.5
Best-Available	0.3	Complete	0.1
Default	0.6	Unknown	0.5
Not Applicable	1.0		

Table 3.17: CWSS External Control Effectiveness Set of Values

For example, a Moderate value for the External Control Effectiveness factor means the protection mechanism is commonly used but has known limitations that might be bypassed with some effort by a knowledgeable attacker.

The Remediation Effort is the amount of effort required to remediate the weakness so that it no longer poses a security risk to the software.

Its corresponding values and weights are described in Table 3.18:

Value	Weight	Value	Weight
Extensive	1.0	Moderate	0.9
Limited	0.8	Default	0.9
Unknown	0.5	Not Applicable	1.0
Quantified			

Table 3.18: CWSS Remediation Effort Set of Values

For example, an Extensive value for the Remediation Effort factor means it Requires significant labor or time to address, possibly requiring modifications to design or architecture; available remediations will otherwise break legitimate functionality; etc.

The Prevalence of a finding identifies how frequently this type of weakness appears in software.

Its corresponding values and weights are described in Table 3.19:

Value	**Weight**	**Value**	**Weight**
Widespread	1.0	High	0.9
Common	0.8	Limited	0.7
Default	0.85	Unknown	0.5
Not Applicable	1.0	Quantified	

Table 3.19: CWSS Prevalence Set of Values

For example, a Widespread value for the Prevalence factor means the weakness is found in most or all software in the associated environment, and may occur multiple times within the same software package.

Unknown	The use of Unknown emphasizes that the score is incomplete or estimated, and further analysis may be necessary. This makes it easier to model incomplete information, and for the Business Value Context to influence final scores that were generated using incomplete information. The weight for this value is 0.5 for all factors, which generally produces a lower score; the addition of new information (i.e., changing some factors from Unknown to another value) will then adjust the score upward or downward based on the new information.
Not Applicable	The factor is being explicitly ignored in the score calculation. This effectively allows the Business Value Context to dictate whether a factor is relevant to the final score. For example, a customer-focused CWSS scoring method might ignore the remediation effort, and a high assurance environment might require investigation of all reported findings, even if there is low confidence in their accuracy.
Quantified	The factor can be weighted using a quantified, continuous range of 0.0 through 1.0, instead of the factor s defined set of discrete values. Not all factors are quantifiable in this way, but it allows for additional customization of the metric.
Default	The factor s weight can be set to a default value. Labeling the factor as a default allows for investigation and possible modification at a later time.

Table 3.20: CWSS factors common values.

CWSS can be used in cases where there is not enough information of the weakness at first, but the quality of information can improve over time. It is expected that in many use cases, the CWSS score for an individual weakness finding may change, as more information is discovered. Most factors have three values in common, unknown, not applicable, and default, which are described in

Table 3.20, but also can be quantified.

In order to calculate the scoring S of a weakness a formula is applied using the 18 factors and their values, which are represented by the Formula (I). The formula S is calculated as the multiplication between the individual scores of each metric group.

$$\boxed{S = Base\ Finding\ *\ Attack\ Surface\ *\ Environmental} \tag{I}$$

The Base Finding BF score is calculated as the Formula (II):

$$\boxed{\begin{aligned} BF = [(10\ *\ Technical\ Impact\ +\ 5\ *\ (Acquired\ Privilege +\\ Acquired\ Privilege\ Layer)\ +\ 5\ *\ Finding\ Confidence) *\\ f(Technical\ Impact)\ *\ Internal\ Control\ Effectiveness]\ *\ 4.0 \end{aligned}} \tag{II}$$

The function f(Technical Impact) will be 0 if Technical Impact equal 0; otherwise f(Technical Impact) equal 1.

The Attack Surface AS score is calculated as the Formula (III):

$$\boxed{\begin{aligned} AS = [20\ *\ (Required\ Privilege\ +\ Required\ Privilege\ Layer\ +\ Access\ Vector)\\ +\ 20\ *\ Deployment\ Scope\ +\ 10\ *\ Level\ Interaction +\\ 5\ *\ (Authentication\ Strength\ +\ Authentication\ Instances)]\ /\ 100 \end{aligned}} \tag{III}$$

The Environmental ENV score is calculated as the Formula (IV):

$$\boxed{\begin{aligned} ENV = [(10\ *\ Business\ Impact\ +\ 3\ *\ (Likelihood\ Of\ Discovery\ +\ Likelihood\\ Of\ Exploit)\ +\ 3\ *\ Prevalence\ +\ Remediation\ Effort) *\\ f(Business\ Impact)\ *\ External\ Control\ Effectiveness]\ /\ 20.0 \end{aligned}} \tag{IV}$$

The function f(Bussiness Impact) will be 0 if Bussiness Impact equal 0; otherwise f(Bussiness Impact) equal 1.

3.3 Conclusions

A thorough presentation of the First Principal Vulnerability Assessment for distributed systems has been done in this chapter. The methodology steps for this approach were presented next. This methodology defines five steps: Architecture analysis, Resources Analysis, Trust and Privileges analysis, Component Analysis, and Dissemination of results. A set of key accomplishments were depicted, derived from the application of FPVA to several large and widely-used middleware systems. For all the middlewares analyzed with FPVA, we have realized that there is a gap between the three initial steps and the component code analysis. This gap directly affects the quality of the vulnerability assessment, because security flaws may be overlooked due to insufficient either analyst knowledge or time for an in depth analysis. Then, the Common Weakness Enumeration for vulnerability classification was presented. A list of CWE details allow the actual classification of such vulnerability types. We have presented the two CWE organizational views, the Development View (CWE-699) and the Research View (CWE-1000), which held within hierarchical structures over 714 weaknesses, from which we chosen the Research View because explicitly models the interdependencies between weaknesses, allowing the matching to FPVA information.

The Common Weakness Scoring System is responsible of prioritizing static and dynamic analysis findings, providing a mechanism for scoring weaknesses in a particular piece of software in a consistent, and flexible manner, through three metrics groups (Base Finding, Attack Surface, and Environmental) with 18 different factors. The combination of some elements from FPVA, CWE, and CWSS allowed to develop a methodology, *AvA4cmi*, which constitutes the essence of this work, and becomes a way to systematically fill the gap found in FPVA, connecting the information gathered by the initial analysis, and knowledge found on the classification of weaknesses.

The details of design and implementation of *AvA4cmi* methodology will be given in the next chapter.

CHAPTER 4

Vulnerability Assessment for Complex Middleware Interrelationships

In this chapter we propose *"AvA4cmi"*, a vulnerability assessment methodology for complex middleware interrelationships in distributed systems. In the first section we describe the attack vectors graph, a key element for the proposed methodology. In Section 4.2, we present the system attributes, our customized version of the CWSS scoring system, and how they are used to build the set of rules that constitute the knowledge base of the methodology. In addition, we introduce the way how it generates a relationship between system attributes and CWE weaknesses, in order to assist rules applicability. In Section 4.3, we describe the algorithm that allows the assessment of such complex middleware interrelationships (depicted in the attack vector graphs) using the codified knowledge within *"AvA4cmi"*. Finally, Section 4.4 presents the security alerts generated by the assessment process.

Introduction to AvA4cmi

The methodology called Attack vector analyzer for complex middleware interrelationships (*AvA4cmi*) focuses on the guidance towards a more comprehensive and accurate vulnerability assessment, characterized by considering the high interoperability between middleware components for systematically hinting where and why to deploy code assessment. The starting point for *AvA4cmi* is to build most likely attack vectors. From these attack vectors, *AvA4cmi* works applying codified knowledge through an algorithm, which is in charge of evaluates and scores the components and their attributes-weaknesses relationship, and how they influence other components that belong to the same attack vector being assessed. Lastly, *AvA4cmi* generates for each attack vector, security alerts containing hierarchized lists of weaknesses (following the CWE taxonomy, and sorted by maximum score), with which the methodology provides to the security practitioner enough information to decide where and why deploy code assessment.

4.1 Attack Vectors Graphs

The first step of (*"AvA4cmi"*) is to automatically determine what to look for at each component in the FPVA diagrams derived from the architectural, resource, and privilege analysis; and then develop a suitable data representation for all this gathered information, to support the asessment process in the methodology. Currently, the security practitioner decides which middleware components are critical, based on his experience.

Hence, the diagrams generated from the first FPVA analysis steps frequently describe particular functionalities of the application (e.g., submitting and canceling jobs, or retrieving job outcomes), such as in the architecture diagrams of CrossBroker and WMS (introduced in the previous chapter), and given that starting (an attack surface point) and ending (impact surface point) components of the diagrams can be clearly identified, they can be viewed as graphs describing how the middleware being assessed works.

In order to represent all these initial FPVA outcomes (diagrams) in a suitable, useful, and unified diagram, we defined a structure called *Attack Vector Graph.*

An attack vector graph is aimed to depict the sequence of transformations that allows control flow to go from a point in the attack surface to a point in the impact surface.

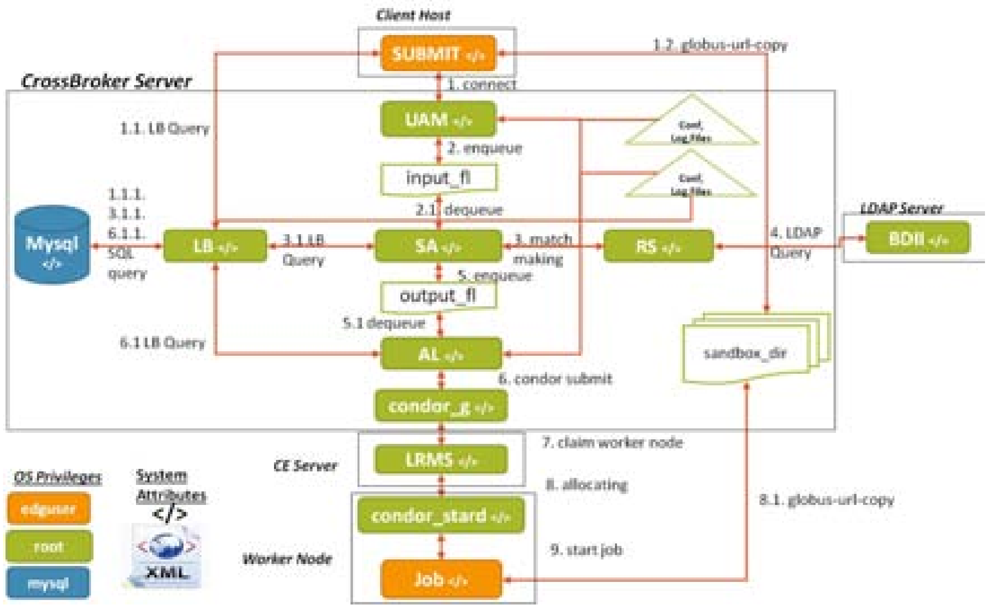


Figure 4.1: Attack Vector graph: CrossBroker example

With this structure, *AvA4cmi* systematically determines all the attack vectors (complex interrelationships) between middleware components and resources, such as those can be found in the figure 4.1 The order in which an attack vector graph is built is also quite clear because every edge in the FPVA diagrams is labeled with a number indicating when the interaction represented by the edge takes place, and also indicating if a node in the diagrams belong either to the attack or the impact surface.

We decided to represent the attack vector graphs with the GraphML format [32], for further use in *Ava4cmi*. GraphML has the advantage over other graph representations, that provides a mechanism to add data to the structural elements (e.g. graph s, node s, edge s, etc.), which help parsers to process a document more efficiently.

Basically, an attack vector graph represented in GraphML format (e.g., Code 4.1, is an XML file composed by the information gathered from the FPVA diagrams, which also includes the relevant system attributes information about the middleware components. Systems attributes will be introduced in subsection 4.2.1.

```
<?xml version="1.0"?>
<graphml xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://graphml.graphdrawing.org/xmlns">
<!--System Attributes Definition-->
<key id="CSI" for="node" attr.type="string"
     attr.name="Client-Server Installation"/>
<key id="Timeout" for="node" attr.type="boolean"
     attr.name="Timeout Operations"/>
<key id="id" for="node" attr.type="int" attr.name="id"/>
...
<!-- Graph instantiation-->
<graph id="CrossBroker" edgedefault="directed">
<!-- Attack vectors statement-->
<data key="attack_vectors">
1,2,3,4,5,6,11,12;1,2,3,4,11,12;1,11,12;1,2,3,4,5,6,7,8,9,10;
     1,2,3,4,11,6,7,8,9,10;1,15,10;1,11,4,5,6,7,8,9,10;
     1,11,6,7,8,9,10</data>
<!-- Component attributes statement-->
<node id="10">
<data key="predecessors">9</data>
<data key="sucessors">11</data>
<data key="id">12</data>
<data key="name">MySQL</data>
<data key="owner">Partially-Privileged User</data>
<data key="language">C++</data>
<data key="timeoutOperations">Yes</data>
<data key="clientServer">Server</data>
<data key="attackSurface">no</data>
<data key="impactSurface">Yes</data> </node>
...
<!-- Component or Resource relationships-->
<edge id="10" target="47" source="12" directed="true"/>
<edge id="20" target="19" source="17" directed="true"/>
<edge id="22" target="10" source="19" directed="true"/>
...
</graph>
</graphml>
```

Code 4.1: Attack Vector GraphML example

Currently, the attack vector graphs are built with an FPVA Graph Editor tool, which belong to the work developed within the MIST group to support the proposed methodology.

4.2 Knowledge Base

We understand that security practitioner knowledge is similar to the one recorded on several available vulnerability classifications, such as the Common Weakness Enumeration [77], The Seven Pernicious Kingdoms [45], the OWASP Top Ten [68], the Comprehensive, Lightweight Application Security Process [83], and the Microsoft SDL [74], and that it can be codified in the form of rules, metrics, and scores.

Thus, the following steps in the *AvA4cmi* methodology is to build a Knowledge Base (KB) based on rules, which allow to be applied systematically when traversing attack vector graphs.

Our KB is based on three elements:

- The most updated knowledge about weaknesses, the CWE community effort.
- A customized weakness scoring system (metrics and scores) based on the CWSS community effort.
- And the most common middleware System Attributes.

All this elements, the CWE weaknesses, the CWSS scoring system, and the system attributes are depicted in figure 4.2, among them there is a synapsis (explained in the following subsections), that is used for building the set of rules that will guide the vulnerability assessment of a target system.

In the next subsection, we introduce the system attributes.

4.2.1 System Attributes

The System Attributes (SA) have been defined in a large research and refining process, extracted from the experience using FPVA, and where the MIST group survey to security practitioners was extended to identify those attributes from middleware components that are and should be remarkable when performing a vulnerability assessment.

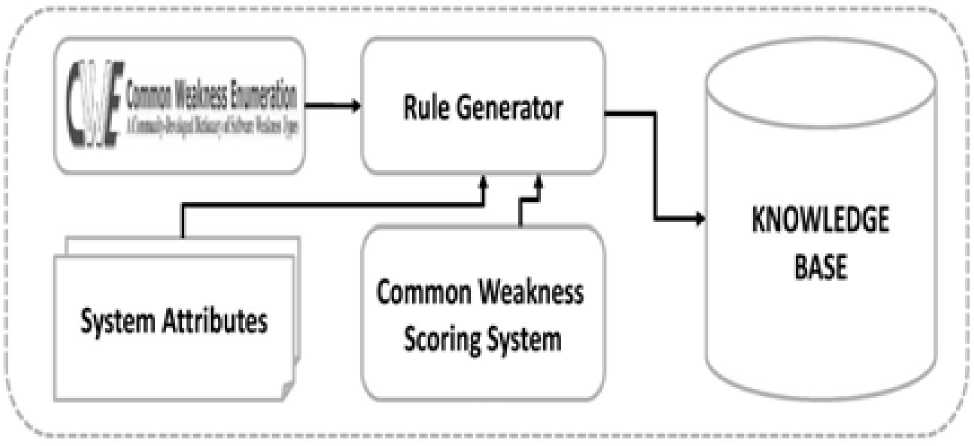

Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types
Rule Generator
KNOWLEDGE BASE
System Attributes
Common Weakness Scoring System

Figure 4.2: Knowledge Base architecture in *AvA4cmi*

System attributes are built with the information provided by several FPVA diagrams, developer team interviews, user and admin documentation, as well as the API documentation. Below, we describe each one of the system attributes we consider.

Owner: The owner attribute is unique, it is an abstraction that denotes a logical entity for assignment of ownership and operation privileges over the system. Due that the owner of the component is capable to assign privileges levels to users who want to interact with it, this attribute allows to know how compromised the component and the system could be in case of a vulnerability exists.

User: The user attribute may correspond to a real world person, but also a type of system operation, which have some operation privileges over the system, such as read, write, and execution. The user attribute allows to know the impact of that component being attacked.

UAI: This attribute shows if the component belong to the user or the administrator interface. Thus, the conditions for the attacker to achieve its goal would increase or not, if the component belong to the user or administrator interface (UAI).

Sanitize: Sanitizing operations are one of the most critical tasks in security. If the component performs some sanitizing operation over the data flow, then the sanitize attribute determine if the exploitability of a vulnerability may be mitigated or not. That means that the conditions are more or less favorable for the attacker to achieve its goal.

Transform: This attribute shows if the component performs transforming operations over the dataflow, whereby an attacker could expect the dataflow is converted or behaves in an inappropriate manner. A transforming operation is for example, when a job description file is then packaged into a wrapper script to be execute later.

Transfer: This attribute shows if the component performs transfering operations with the dataflow, whereby an attacker could expect the dataflow behaves in an inappropriate manner when arrives to other components. A transfering operation is for example, when the wrapper script is submitted from the resource manager to an available compute node, to finally be execute.

Trust: The trust attribute shows if the component performs trustworthy operations, such as authentication and authorization of user and server credentials, when users or communication process try to interact to the component. Usually, this operations often happen in the first components of a middleware, so insider attackers can have more favorable conditions to achieve its goal.

Server Interaction: This attribute shows if a component performs or not server operations, such as queries against a database, LDAP, NFS, Web server or any other type of server query. If the middleware components perform several different queries, attackers can have more favorable conditions to achieve its goal. For query we mean read, write, and execute operations.

Timeout: A critical task in security is to control timeout answers. If a component does not implement timeout answer for its operations, an attacker is able to perform brute force attacks, or worst denial of service attacks. Thus, this attribute shows if the component controls or not timeout operations.

Max-Min: The operations to control length, size, volume, or bounds are a great concern for security practitioners, due that they are one of the most used points by attackers to trigger and execute arbitrary code. The max-min attribute determines if the component performs or not control operations over dataflow length, size, volume, or bounds.

Third-party: This attribute determines if the component performs or not operations with third parties, such as interact locally o remotelly with services or components outside of the middleware scope.

Spoofing: This attribute shows if the component performs or not operations against spoofing attacks, a situation in which one person or program act on behalf of another by falsifying data and thereby gaining an illegitimate advantage. This is especially attractive for attackers in distributed systems, e.g., once they have sniffed certificates for single sign on, they can impersonate another to gain delegation of rights to other entities.

Tampering: Tampering involves the deliberate altering or adulteration of dataflow, a component, or system. Some attacks just may want to alter or destroy sensitive data, in order to avoid exposing or using them in critical business operations. The tampering attribute shows if the component provides or not tamper proof operations.

Encryption: This attribute shows if the component has or not an abstract or concrete protocol that performs a security related function and applies encryption methods, which are designed to provide communication security. But today attackers may not be too worried about it[1].

Attachment: The capability to attach any file type becomes a potential threat for the whole middleware, no matter the environment circumstances. It is difficult to detect if the attachment is per se the real threat. This attribute shows if the component provides or not attachment operations.

EEH: This attribute shows if the component performs or not error and exception handling strategies to deal with anomalous, exceptional events, or processing, that often change the normal flow of program execution. An attacker may take advantage of lack of error and exception handling to trigger an event that is controlled at will.

Client-Server: The client-server (CSI) attribute determines if the component is installed on the client side or the server side of the middleware. Similarly to UAI attribute, to be installed at the client or server side makes more easy or difficult the task for an attacker to achieve its goal. But, it is worth to state that the client side will be always under control of the user, and anything can happen there.

Web: Web technologies have ventured strongly in every computing area, and nowadays middleware systems are not the exception. Along with them specific threats arise, which allow attackers to have bigger attack surfaces. This attribute shows if the component is using or not a Web technology.

Logging-Backup: The logging and backup (LogBak) attribute determines if the component performs loggin and/or backup tasks. They seem to be two simple tasks without any risk, on the other hand these tasks may contain too much information that an attacker can exploit.

[1] `http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security`

Attack surface: This attribute determine if the component is a point in the attack surface. It is included to support the building of the attack vectors, and it is only reflected in the *graphml* file when traversing the attack vector graphs.

Impact surface: This attribute determine if the component is a point in the impact surface. As well as the attack surface attribute, the impact surface attribute is included to support the building of the attack vectors, and it is only reflected in the graphml file when traversing the attack vector graphs.

Most of system attributes have boolean values (i.e., yes or no), in order to avoid generate decision problems, or what is the same, uncertainty, when creating the attack vector graphs, and when applying the *AvA4cmi* methodology.

Only the owner and user attributes can have different values (i.e., Administrator, Partially-Privileged User, Regular User, Guest, None, Default, Unknown, or Not Applicable). If a system attribute can not be derived, the attribute is deemed not controlled, in other words a pessimistic approach is used.

In the next subsection, we introduce the relationship between system attributes and CWE weaknesses.

4.2.2 Relationship between System Attributes & Weaknesses

In order to get the synapsis between system attributes and CWE weaknesses that assist the rules applicability, we studied comprehensively the CWE taxonomy, and analyzed all the information it provides. From this study, we understand that not all the information provided by CWE is relevant for the purposes in the relationship we looked for; also, not all the weaknesses provide information for specific details, such as real examples of the weakness in the CVE database [76].

Therefore, we proposed a two-step approach to find out an accurate relationship:

(1) The first thing needed is to gather the most useful information from the CWE source; We proposed 12 different elements, which provide the most relevant details of a weakness. They are acquired through a public XML file supplied by the CWE community [75]. The 12 elements are: 1) a

weakness identificator, 2) a weakness name, 3) a description, 4) an extended description, 5) the programming language, 6) the consequence scope, 7) the consequence technical impact, 8) the consequence notes, 9) the mitigation description, 10) the mapped node names, 11) the relationships, and 12) the observed example description.

(2) For the second step, *AvA4cmi* runs a comprehensive searching of words and synonyms related to each of 19 system attributes over the twelve elements, and conducted carefully on the 714 weaknesses to find the accurate set of CWE weaknesses.

For the sake of understanding, we describe the 12 elements using a real example of a CWE weakness [10]:

(1) The weakness identificator: *CWE-20.*

(2) The weakness name: *Improper Input Validation.*

(3) The weakness description: *The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.*

(4) The weakness extended description: *When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.*

(5) The programming language in which the weakness may occur: *Language-independent.*

(6) The consequence scope, which identifies an individual consequence that may be associated to the weakness: *Availability, Confidentiality, Integrity.*

(7) The consequence technical impact, which describes the technical impacts that can arise if an attacker attempts to exploit the weakness: *DoS: crash / exit / restart; DoS: resource consumption (CPU); DoS: resource consumption (memory); Read memory; Read files or directories; Modify memory; Execute unauthorized code or commands.*

(8) The consequence notes, which provides additional commentary about its consequence: *An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU; An attacker could read confidential data if they are able to control resource references; An attacker could use malicious input to modify data or possibly alter control flow in unexpected ways, including arbitrary command execution.*

(9) The mitigation description, which contains a single method for mitigating the weakness: *Use an input validation framework such as Struts or the OWASP ESAPI Validation API; Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls; Assume all input is malicious.*

(10) The mapped node names, which identifies the name of the entry to which this weakness is being mapped in other taxonomies or classifications: *Input validation and representation; Unvalidated Input; Improper Input Handling.*

(11) The relationship, which contains a note regarding the relationships between CWE entries: *Child Of 19: Data Handling; Child Of 693: Protection Mechanism Failure; Parent Of 77: Improper Neutralization of Special Elements used in a Command ('Command Injection'); Parent Of 79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'); Parent Of 89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'); etc..*

(12) The observed example description, which presents an unambiguous correlation between the example being described and the weakness which it is meant to exemplify: *CVE-2008-3843: Insufficient validation enables XSS; CVE-2008-1303: Missing parameter leads to crash; CVE-2008-3494: Security bypass via an extra header, etc..*

For example, the synapsys for the system attribute *"User"* is produced after a search for the words:

- `root, admin, sysadmin, administrator(s), owner, ownership, actor(s), username, attacker(s), account(s), sudo, user(s), domain(s), superuser, and supervisor.`

As a result, 150 different related weaknesses were found. Among these search results, show up the weakness *"CWE–282: Improper Ownership Management"*. In a nutshell, it says *"The software assigns the wrong ownership, or does not properly verify the ownership, of an object or resource"*. Thus, for each of the remaining system attributes the step two of the appraoch is repeated, and show up between 20 to 150 relationships with CWE weaknesses.

With the system attributes and CWE weaknesses relationship stated, we present in the next subsection the rules.

4.2.3 Rules

The next step in the *AvA4cmi* methodology is define the logical propositions, or what is the same, the knowledge base of rules. The rules function consist of essentially to obtain a quantitative measurement for each one of the 19 system attributes of the middleware components.

More precisely, the rules evaluate the current value of each system attribute against metric groups and their factors from a customized version of the CWSS scoring system. Below we illustrate by way of seudo-code two different example of the rules:

- Example I. Rule for the Owner attribute

```
    If $Owner == Administrator$ then:
    $TI == Critical$, $AP == Administrator$
    $AL == Enterprise$, $AV == Private\ Network$
```

- Example II. Rule for the User-Admin Interface attribute

```
    If $User\text{-}Admin\ Interface == Yes$ then:
    $TI == High$, $AV == Local$, $IN == Automated$
    $DI == High$, $EX == High$, $AS == Moderate$
```

Legend: Technical Impact (TI), Acquired Privilege (AP), Acquired Privilege Layer (AL), Access Vector (AV), Level of Interaction (IN), Authentication Strength (AS), Likelihood of Discovery (DI), Likelihood of Exploit (EX).

It can be appreciated in both examples that the Owner and the User-Admin Interface attributes are related with particular CWSS factors such as the TI factor, which will be measured according to the corresponding values.

In the first example, the TI factor answers with the Critical value when the Owner attribute corresponds with the Administrator value.

For the User Admin Interface rule example, the TI factor assumes a High value, because the component being assessed is part of the attack surface, but not necessarily have to be considered harnessed by an attacker to get Critical .

All the rules are derived similarly to the two examples, taking into account the correlation between respective attributes and factors, and they are reflected in the implementation repository [43] of the methodology.

The Table 4.1 shows the metric groups and their factors for our own customized version of the CWSS system used to build the rules.

Base Finding	Attack Surface	Environment
Technical Impact (TI)	Required Privilege (RP)	Business Impact (BI)
Acquired Privilege (AP)	Required Privilege Layer (RL)	Likelihood of Discovery (DI)
Acquired Privilege Layer (AL)	Access Vector (AV)	Likelihood of Exploit (EX)
Internal Control Effectiveness (IC)	Authentication Strength (AS)	External Control Effectiveness (EC)
	Authentication Instances (AI)	Prevalence (P)
	Level of Interaction (IN)	
	Deployment Scope (SC)	

Table 4.1: Custom CWSS Metric groups

This customized version of CWSS is characterized by setup values in terms of the system attributes, e.g., the Internal Control (IC) factor have the same weight (1) for the Owner and User attributes, because there is no protection mechanism to prevent all possible attacks that either malicious user or owner could make. And, our customized CWSS systems is also characterized by removing some factors from the original Base Finding and Environment group metrics, such as the Finding Confidence, and the Remediation Effort. Because, on one hand, *AvA4cmi* does neither claim that a vulnerability exist nor that it does not exist, whereby we can not consider the finding confidence factor in the methodology, because this factor assumes a vulnerability exists, and can be triggered or utilized by an attacker.

On the other hand, the remediation effort factor reflects a bias that weaknesses that are more expensive to fix will have higher scores than the same types of weaknesses that are less expensive to fix, and due that the *AvA4cmi* methodology does not remediate issues, this factor is out of scope. Thus, the score formulas were correctly balanced, taking into account the elements removed.

In order to complete the rules building, we proposed a correlation between the system attributes and the factors, which is depicted in Table 4.2. Some of the factors are perfectly correlated attributes, scoreed with an X in Table 4.2. Others can be correlated to a lesser extent, which are labeled by default, using a D in the table. And other factors are not possible to correlate, which are scoreed NA in the table. The latter does not necessarily mean that provide zero score, on the contrary to not know the relationship may score higher.

With this correlation, we proposed different weights for the factors values, which are depicted in Tables 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, and 4.20.

Legend: X = Applicable, NA = Not applicable, D = Default

Attributes	Base Finding				Attack Surface							Environment				
	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
Owner	X	X	X	NA	NA	X	X	NA	NA	NA	NA	X	D	D	NA	NA
User	X	NA	X	NA	X	X	X	X	X	NA	NA	X	D	D	NA	NA
UAI	X	NA	NA	D	NA	X	X	X	X	X	X	X	X	X	NA	NA
Sanitize	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	X	X	NA	NA
Transform	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	D	D	NA	NA
Transfer	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	D	D	NA	NA
Trust	X	NA	NA	X	NA	X	NA	X	X	NA	NA	X	D	D	NA	NA
Server Int	X	NA	X	D	X	X	X	X	X	X	NA	X	X	X	D	NA
Timeout	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	X	X	D	NA
Max-Min	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	X	X	D	NA
3rd-Party	X	NA	NA	D	X	X	X	X	X	X	NA	X	NA	NA	X	NA
Spoofing	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	X	X	NA	NA
Tampering	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	X	X	NA	NA
Encryption	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	X	X	NA	NA
Attachment	X	NA	NA	X	NA	X	NA	NA	NA	X	NA	X	X	X	NA	NA
EEH	X	NA	NA	X	NA	X	NA	NA	NA	NA	NA	X	X	X	NA	NA
CSI	X	NA	X	NA	NA	X	X	NA	NA	NA	NA	X	X	X	NA	NA
Web	X	NA	X	NA	NA	X	X	NA	NA	NA	NA	X	X	X	NA	NA
LogBak	X	NA	X	NA	NA	X	X	NA	NA	NA	NA	X	X	X	NA	NA

Table 4.2: System attributes & CWSS factors correlation in *AvA4cmi*

Therefore, with the knowledge base of rules stated, and the obtained quantitative measurements, we proceed to use them further in the coming step of the methodology, when the complex middleware interrelationships come into play.

Owner	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
ADM	1	1	1	1	0,1	1	0,8	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5
PPU	0,9	0,9	0,9	1	0,1	1	0,8	0,5	0,5	0,1	0,1	0,9	0,6	0,6	0,1	0,5
RU	0,6	0,7	0,7	1	0,1	1	0,5	0,5	0,5	0,1	0,1	0,6	0,6	0,6	0,1	0,5
GU	0,3	0,6	0,5	1	0,1	1	0,5	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5
NU	0	0,1	0,5	1	0,1	1	0,5	0,5	0,5	0,1	0,1	0	0,6	0,6	0,1	0,5
DU	0,6	0,7	0,9	1	0,1	1	0,5	0,5	0,5	0,1	0,1	0,6	0,6	0,6	0,1	0,5
UU	0,5	1	0,5	1	0,1	1	0,5	0,5	0,5	0,1	0,1	0,5	0,6	0,6	0,1	0,5
NA	0,3	0,1	0,5	1	0,1	1	0,5	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.3: CWSS customized scores for Owner Attribute

User	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
ADM	1	0,1	1	1	0,1	1	0,8	0,8	0,8	0,1	0,1	1	0,6	0,6	0,1	0,5
PPU	0,9	0,1	0,9	1	0,6	1	0,8	0,8	0,8	0,1	0,1	0,9	0,6	0,6	0,1	0,5
RU	0,6	0,1	0,7	1	0,7	1	0,5	0,8	0,8	0,1	0,1	0,6	0,6	0,6	0,1	0,5
GU	0,3	0,1	0,5	1	0,9	1	0,5	0,8	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5
NU	0	0,1	0,5	1	1	1	0,5	0,8	0,5	0,1	0,1	0	0,6	0,6	0,1	0,5
DU	0,6	0,1	0,9	1	0,7	1	0,5	0,8	0,5	0,1	0,1	0,6	0,6	0,6	0,1	0,5
UU	0,5	0,1	0,5	1	0,5	1	0,5	0,8	0,5	0,1	0,1	0,5	0,6	0,6	0,1	0,5
NA	0,3	0,1	0,5	1	0,1	1	0,5	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.4: CWSS customized scores for User attribute

Legend: ADM = Administrator, PPU = Partially-Privileged user, RU = Regular user, GU = Guest user, NU = None, DU = Default user, UU = Unknown user, NA = Not applicable

UAI	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	0,5	0,6	0,1	1	0,5	0,8	0,8	1	0,7	1	0,6	0,6	0,1	0,5
NO	0,9	0,1	0,5	0,6	0,1	1	0,8	0,7	0,8	0,1	0,7	0,9	0,6	0,6	0,1	0,5

Table 4.5: CWSS customized scores for UAI attribute

Sanitize	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	0,6	0,1	0,5	0,7	0,1	1	0,75	0,5	0,5	0,1	0,1	0,6	0,6	0,6	0,1	0,5
NO	1	0,1	0,5	0,9	0,1	1	0,75	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5

Table 4.6: CWSS customized scores for Sanitize attribute

Transform	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	0,5	0,7	0,1	1	0,75	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,1	0,5	0,9	0,1	1	0,75	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.7: CWSS customized scores for Transform attribute

Transfer	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	0,5	0,7	0,1	1	0,75	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,1	0,5	0,9	0,1	1	0,75	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.8: CWSS customized scores for Transfer attribute

Trust	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	0,6	0,1	0,5	0,7	0,1	1	0,2	0,9	0,8	0,1	0,1	0,6	0,6	0,6	0,1	0,5
NO	1	0,1	0,5	0,9	0,1	1	0,2	0,8	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5

Table 4.9: CWSS customized scores for Trust attribute

Server	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,7	0,9	0,6	0,6	1	0,8	0,85	0,8	0,55	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,7	0,9	0,6	0,1	1	0,2	0,85	0,8	0,55	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.10: CWSS customized scores for Server attribute

Timeout	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	0,5	0,6	0,1	1	0,2	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,1	0,5	0,6	0,1	1	0,2	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.11: CWSS customized scores for Timeout attribute

Max-Min	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	0,5	0,6	0,1	1	0,2	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,1	0,5	0,6	0,1	1	0,2	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.12: CWSS customized scores for Max-Min attribute

3rd-Party	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	0,5	0,6	0,6	1	0,8	0,85	0,8	0,55	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,1	0,5	0,6	0,1	1	0,2	0,85	0,8	0,55	0,1	0,3	0,6	0,6	0,1	0,5

Table 4.13: CWSS customized scores for Third-party attribute

Spoofing	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	0,3	0,1	0,5	0,5	0,1	1	0,2	0,5	0,5	0,1	0,1	0,3	0,2	0,2	0,1	0,5
NO	1	0,1	0,5	0,9	0,1	1	0,2	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5

Table 4.14: CWSS customized scores for Spoofing attribute

Spoofing	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	0,3	0,1	0,5	0,5	0,1	1	0,2	0,5	0,5	0,1	0,1	0,3	0,2	0,2	0,1	0,5
NO	1	0,1	0,5	0,9	0,1	1	0,2	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5

Table 4.15: CWSS customized scores for Tampering attribute

Encryption	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	0,3	0,1	0,5	0,5	0,1	1	0,2	0,5	0,5	0,1	0,1	0,3	0,2	0,2	0,1	0,5
NO	1	0,1	0,5	0,9	0,1	1	0,2	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5

Table 4.16: CWSS customized scores for Encryption attribute

Encryption	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	0,5	0,7	0,1	1	0,2	0,5	0,5	1	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,1	0,5	0,5	0,1	1	0,2	0,5	0,5	0,1	0,1	0,3	0,2	0,2	0,1	0,5

Table 4.17: CWSS customized scores for Attachment attribute

EEH	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	0,3	0,1	0,5	0,5	0,1	1	0,2	0,5	0,5	0,1	0,1	0,3	0,6	0,6	0,1	0,5
NO	1	0,1	0,5	0,7	0,1	1	0,2	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5

Table 4.18: CWSS customized scores for Error & Exception handling attribute

CSI	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	0,6	0,1	1	0,3	0,1	1	0,5	0,5	0,5	0,1	0,1	0,6	0,6	0,6	0,1	0,5
NO	1	0,1	1	0,3	0,1	1	0,8	0,5	0,5	0,1	0,1	1	0,2	0,2	0,1	0,5

Table 4.19: CWSS customized scores for Client-Server Installation attribute

Web	TI	AP	AL	IC	RP	RL	AV	AS	AI	IN	SC	BI	DI	EX	EC	P
YES	1	0,1	1	0,3	0,1	1	1	0,5	0,5	0,1	0,1	1	0,6	0,6	0,1	0,5
NO	0,3	0,1	0,5	0,3	0,1	1	0,5	0,5	0,5	0,1	0,1	0,3	0,2	0,2	0,1	0,5

Table 4.20: CWSS customized scores for Web attribute

4.3 An Algorithm for the Evaluation of interrelationships in Attack Vector graphs

This step of the *AvA4cmi* methodology describes the algorithm (a formal method) that systematically apply the instantiated knowledge base when traversing the attack vector graphs, and assesses complex middleware interrelationships, in order to generate a list of hierarchized security alerts, which hint where and why middleware components should be assessed.

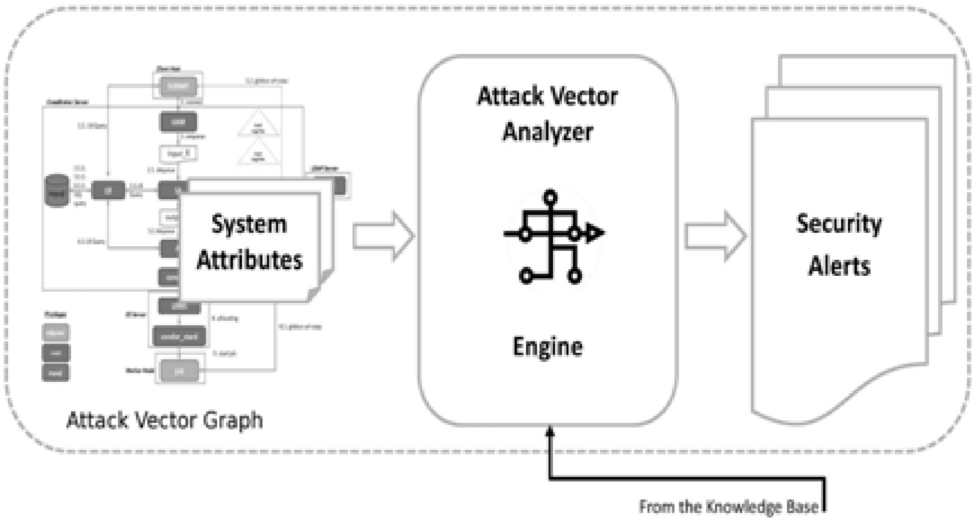


Figure 4.3: Algorithm architecture in *AvA4cmi*

Figure 4.3 describes the architecture proposed for the algorithm engine, in which the external black arrow represents the connection to the instantiated knowledge (Section 4.2); the left icon represents the attack vector graph (Section 4.1), and the right icon represents the outcomes produce by the algorithm, the security alerts.

The first element read is the attack vector graph, from which the algorithm identifies and loads the attack vectors, the components of the middleware being assessed, and their corresponding system attributes. Secondly, the algorithm read the knowledge base, in order to load both the stated rules and the stated relationships between the CWE weaknesses and the system attributes.

Once all the codified knowledge required is ready, the algorithm follows the next steps:

Step 0. The algorithm starts traversing each attack vector, component by component. For each component, its system attributes are fetched from the graph file, and then assessed according to the current stated rules. Hence, the weaknesses related with these system attributes are assessed too, according to the relationship proposed in 4.2.2. As a result, an individual score is obtained for each attribute associated with every weakness, for each component on the attack vector being traversed.

It is worth to state if the weakness belong to the Top 25 of CWE Weaknesses, we evaluated the Likelihood of Discovery (DI), and the Likelihood of Exploit (EX) factors with the High value.

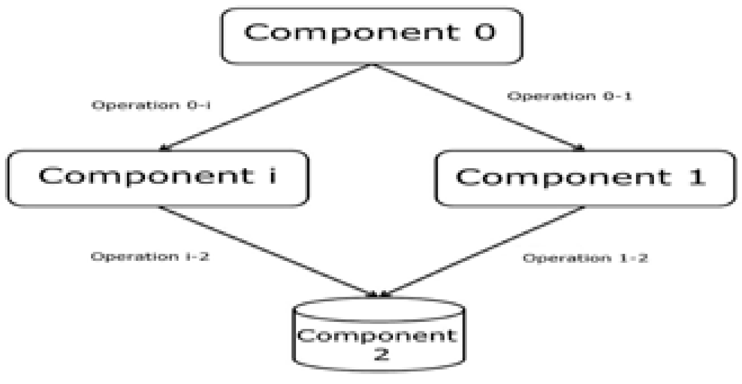


Figure 4.4: Simplified Attack Vector graph

CWE-X			
Attribute / Component	A_0	A_1	A_2
C_0	$0,1$	$0,2$	$0,3$
C_1	$0,4$	$0,5$	$0,6$
C_2	$0,7$	$0,8$	$0,9$

Table 4.21: Example of individual scores for A_0, A_1, and A_2

For example, in the Figure 4.4, after applying the stage zero of the algorithm, the attack vector comprising the components zero (C_0), one (C_1), and two (C_2), obtains the scores shown in Table 4.21, for their system attributes A_0, A_1, and A_2. And, therefore the weakness CWE-X with associated attributes A_0, A_1, and A_2 gets several scores.

Step 1. We realized the several scores (obtained in the previous stage) associated to the weaknesses can be summarized in a single score. We proposed to assign to each attribute the minimum score obtained by any component of the attack vector. We have choosen the minimun value because it means in that component at least, a weakness mitigation has been implemented.

Following with the example of Table 4.21, the minimum score of attribute A_0 related to the weakness CWE-X, will be the minimum value between $0,1$,$0,4$, and $0,7$, that is, $0,1$. Respectively, for the attributesA_1, and A_2, the minimun scores will be $0,2$ and $0,3$.

Step 2. After the minimum scores for the weaknesses are computed, then the algorithm proceeds to weigh them according to the relevance of the system attribute for every of the 11 top-level entries (pillars) in the taxonomy followed in this work (the CWE Research View). We proposed this weighing, because we realized the system attributes might impact different for each weakness depending of one pillar or another.

i.e., a system attribute such as owner has a high weight regarding the CWE 693 Protection Mechanism Failure pillar, while the encryption attribute has a low weight. On the contrary, for the CWE 330 Use of

Insufficiently Random Values pillar the encryption attribute has a high weight, while the owner has a low weight.

We defined three levels of relevance according to the top-level pillars, in level 1 the attributes are very relevant for the weakness, in level 2 the attributes are enough relevant, and level 3 the attributes are a bit relevant.

The different top-entries levels are depicted in Tables 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30, 4.31, and 4.32.

CWE-118 Improper Access of Indexable Resource (Range Error)	
Level 1.	Max-Min,EEH,Trust,UAI
Level 2.	Attachment,Timeout,Sanitize,LogBak,CSI,Owner,User,Server
Level 3.	Encryption,Spoofing,Tampering,Web,3rd-party,Transformation,Transfering

Table 4.22: Weighting levels for CWE-118

CWE-330 Use of Insufficiently Random Values	
Level 1.	Encryption,Timeout
Level 2.	Owner,User,UAI,Spoofing,Max-Min,Trust,EEH,Web
Level 3.	Tampering,Sanitize,Transformation,Transfering,Server,Attachment,3rd-party,CSI,LogBak

Table 4.23: Weighting levels for CWE-330

CWE-435 Interaction Error	
Level 1.	UAI,Sanitize,Attachment,EEH
Level 2.	Web,Trust,Owner,User,Transformation,Transfering,Server,3rd-party,TimeOut,Max-Min,CSI
Level 3.	Spoofing,Tampering,Encryption,LogBak

Table 4.24: Weighting levels for CWE-435

CWE-664 Improper Control of a Resource Through its Lifetime	
Level 1.	TimeOut,Transfering
Level 2.	LogBak,Owner,User,Attachment,Web,3rd-party,Server,EEh,Tampering,Max-Min,Trust
Level 3.	UAI,Spoofing,Encryption,Sanitize,Transformation,CSI

Table 4.25: Weighting levels for CWE-664

CWE-682 Incorrect Calculation	
Level 1.	Transformation,EEH,Trust
Level 2.	Max-Min,Transfering,Owner,User,TimeOut,Server,3rd-party, Sanitize,CSI,Attachment
Level 3.	UAI,Web,LogBak,Spoofing,Tampering,Encryption

Table 4.26: Weighting levels for CWE-682

CWE-691 Insufficient Control Flow Management	
Level 1.	UAI,Sanitize,Transfering
Level 2.	TimeOut,Max-Min,Owner,User,Transformation,Attachment,3rd-party,EEH,Trust,Server,CSI,Web
Level 3.	Spoofing,Tampering,Encryption,LogBak

Table 4.27: Weighting levels for CWE-691

CWE-693 Protection Mechanism Failure	
Level 1.	Owner,User,UAI,Sanitize,Spoofing,Tampering,Encryption,Trust
Level 2.	Transformation,Transfering,Attachment,3rd-party,Server,EEH,CSI,Web,LogBak
Level 3.	TimeOut,Max-Min

Table 4.28: Weighting levels for CWE-693

CWE-697 Insufficient Comparison	
Level 1.	Attachment
Level 2.	Transformation,3rd-party,Owner,User,TimeOut,Max-Min,UAI,Sanitize,Trust,Server
Level 3.	Spoofing,Tampering,Encryption,Transfering,CSI,Web,LogBak

Table 4.29: Weighting levels for CWE-697

CWE-703 Improper Check or Handling of Exceptional Conditions	
Level 1.	EEH,Owner,User
Level 2.	UAI,Sanitize,Trust,Server,Spoofing,Tampering,Encryption,Attachment, 3rd-party,Transformation,Transfering,TimeOut,Max-Min
Level 3.	CSI,Web,LogBak

Table 4.30: Weighting levels for CWE-703

CWE-707 Improper Enforcement of Message or Data Structure	
Level 1.	Sanitize,Web,Attachment
Level 2.	Owner,User,UAI,Spoofing,Tampering,Encryption,3rd-party, Trust,Server,Max-Min,EEH,CSI,Transformation,Transfering
Level 3.	TimeOut,LogBak

Table 4.31: Weighting levels for CWE-707

CWE-710 Coding Standards Violation	
Level 1.	Owner,User,Server,3rd-party
Level 2.	UAI,Transformation,Transfering,TimeOut,Max-Min,EEH,CSI,Encryption,Attachment,Trust
Level 3.	Spoofing,Tampering,Web,LogBak,Sanitize

Table 4.32: Weighting levels for CWE-710

Step 3. Once the scores are weighed, the algorithm computes a maximum score for the weakness. We considered that the maximum score have to take into account how was the score of its child weaknesses. It means that child weaknesses provide their weigh information to the top-level weaknesses.

For example, the weakness CWE-X got a weigh score A, and its only child weakness CWE-Y got a weigh score B, and score $A < B$, so the weakness parent CWE-X is more likely to be affected by its child weakness CWE-Y, hence the maximum score for the weakness CWE-X will be B instead of A.

4.4 Security Alerts

Once all the maximum scores for the weaknesses are processed, the last algorithm step is to present for each assessed attack vector, the security alerts as a hierarchical list of weighed weaknesses, for each one of the 11 top-levels entries (the pillars of the taxonomy followed in this work) the CWE Research View. The security alerts are sorted with respect to their maximum weighed scores, like in the Code 4.2., where the own score obtained for the weakness is represented by ***Sp***, and the score the weakness can inherited from a child weakness is represented by ***Max***.

```
CWE-PILLAR: A Top level entry weakness
__________________________________________

1. CWE-A: The weakness with the highest score: Sp= 0.9 Max= 0.9
     |
    ---> CWE-B: A weakness child of "A" with lower score:
      Sp= 0.7875 Max= 0.7875

2. CWE-C: The weakness with the second highest score inherited:
      Sp= 0.7875 Max= 0.8
     |
    --->CWE-D: A weakness child of "C" with higher score:
      Sp= 0.8 Max= 0.8
           |
          --->CWE-E: A weakness child of "D" with lower score:
      Sp= 0.6 Max= 0.6
```

```
3. CWE-F: Another weakness in the Pillar with the third highest score:
     Sp= 0.7 Max= 0.7
```

Code 4.2: Security alert example

With this type of security alerts proposed, *AvA4cmi* systematically provides comprehensive information to the security practitioner, pointing out not only which weaknesses should be analyzed, but also why we should pay attention to them in the assessed attack vector.

Below is depicted the whole algorithm of the *AvA4cmi* methodology:

Algorithm 1 Pseudo-code of *AvA4cmi* algorithm

```
1.  Read the Attack Vector graph
2.      Load the Attack Vector paths
3.  Read the Knowledge Base
4.      Load the Rules
5.      Load the Relationships
6.  For each Attack Vector
7.      For each Component
8.          Fetch the system attributes
9.          Apply Rules to the system attributes
10.         Assess the weaknesses related
11.     For each Weakness
12.         Compute the minimum score components
13.         Weigh the computed score for the weakness
14.         Compute the maximum score based on child weaknesses
15.     For each Pillar at the CWE research view
16.         Sort weaknesses in order of maximum score
17.         Write the sorted security alerts to plot graphical representation
```

4.5 Conclusions

A new methodology for guidance of vulnerability assessment of complex middleware interrelationships in distributed systems has been presented in this chapter. The **Attack Vector** graph for the methodology was presented. This graph defines the complex middleware interrelationships: the attack vectors and their components. Different elements allow the building of the current Knowledge Base of the methodology:

- The **System Attributes** identify those attributes from the middleware components that are remarkable when performing a vulnerability assessment, they were extracted from the experience using FPVA.
- The proposed approach to find out an accurate relationship between system attributes and weaknesses.
- The **Rules** are logical propositions to obtain a quantitative measurement for each one of the 19 system attributes of the middleware components, using a proposed customized version of the CWSS.

We also have presented an **Algorithm** that allows to systematically apply the instantiated codified knowledge when traversing the attack vector graphs, it is in charge of evaluates and scores the components and their attributes-weaknesses relationship. The results of traversing the attack vectors and applying the algorithm to them for hinting where and why to deploy code assessment were shown as the **Security Alerts**.

CHAPTER 5

AvA4cmi Experimental Evaluation

In this Chapter, we report on the different assessments performed in order to show the benefits of the *AvA4cmi* methodology, and the contrast against previous manual FPVA vulnerability assessment of CrossBroker and gLite WMS middlewares.

In Section 5.1 we present the manual results after apply FPVA vulnerability assessment guidelines on CrossBroker. Section 5.2 presents the security alerts for CrossBroker using the *AvA4cmi* methodology. Section 5.3 presents the manual results after apply FPVA vulnerability assessment guidelines on gLite WMS. Finally, Section 5.4 presents the security alerts for gLite WMS using the *AvA4cmi* methodology.

5.1 CrossBroker & FPVA

CrossBroker [18] is a Grid resource management system for interactive and parallel appplications. It was used in various european projects, including Crossgrid [56], the Interactive European Grid [57], and it was used also at the origins of the

Spanish Grid Initiative [13]. CrossBroker was built by extending the functionality provided in LCG [4] and gLite WMS [8] 3.0 version.

Being able to access the real production system and the human resources of CrossBroker, and being developed at the Universidad Autónoma de Barcelona, encouraged us and facilitated to perform the vulnerability assessment following the FPVA approach.

FPVA identified serious and complex vulnerabilities affecting high value assets in the selected 0.5.35 version of CrossBroker. Below, we describe the publicly available vulnerabilities for CrossBroker after applied FPVA:

5.1.1 CrossBroker-2009-0001

- Summary: If CrossBroker is used in an environment where the user can control certain attributes of the JDL submission file, but the executable to run must be selected from a white list of valid executables, then there exists a flaw that allows the user to run arbitrary code as the execute user beyond the white listed executables.

- Vulnerable Versions: 0 - 0.5.35

- Components: i2g-job-submit, i2g-wl-ns_daemon

- Access Required: This vulnerability requires the user to be able to submit jobs to CrossBroker.

- Effort Required: Low. Exploiting this vulnerability requires the user to be able to control certain attributes of the job submissions file and to specify carefully crafted values for these attributes.

- Impact/Consequences: Low. In a typical configuration the impact will be low since the code will be executed as an unprivileged execute user. If the batch system, uses common shared execute accounts for distinct submission users, then the consequences could be much higher as this could allow a malicious user to attack other user s jobs.

- Cause: Code injection, Improper data validation

5.1.2 CrossBroker-2009-0002

- Summary: Certain types of user s job submitted to CrossBroker are not protected from manipulation from other user s jobs.
- Vulnerable Versions: 0 - 0.5.35
- Components: i2g-wl-workload_manager
- Access Required: This vulnerability requires the user to be able to submit jobs to CrossBroker.
- Effort Required: Low.

  To be able to exploit this vulnerability only requires that an attacker be able to submit a job to CrossBroker of a certain type that contains a malicious payload.
- Impact/Consequences: Medium. The consequence of this vulnerability is that an attacker can manipulate the victim s process and data files, if the attacker and victim are scheduled on the same host concurrently. The attacker can only manipulate the victim s files that were placed on the host by CrossBroker as the victim is run under a different account than their own.
- Cause: Incorrect privileges, Multiple unique privilege domains

5.1.3 CrossBroker-2009-0003

- Summary: Remote resources (worker nodes on grid sites) are prone to a hijacking through Crossbroker. If Computing Elements use a firewall/NAT traversal solution to allow access to grid site elements, attackers will build an independent high throughput computing system without Crossbroker interactions and restrictions.
- Vulnerable Versions: 0 - 0.5.35
- Components: i2g-wl-workload_manager, LRMS (GCB) Library
- Access Required: This vulnerability requires the user to be able to submit jobs to CrossBroker.

- Effort Required: High. Exploiting this vulnerability requires the user to be able to write a shell script, to have knowledge of the condor_config configuration file, to pack the accurate condor daemons, to be able to build their own Condor system, and to know about firewall/NAT network parameters.
- Impact/Consequences: High. If the attacker is successful, any worker node on the grid site could be used infinitely without having to use CrossBroker to submit new jobs to these resources. Then worker nodes can be used at any time by the attacker. The worker nodes will belong to a new High Throughput Computing system controlled by the user. Then any program can be run on behalf of a local user on those worker nodes even if blacklist programs is defined. Finally impact/consequences severity depends on the specific grid sites security policies
- Cause: Incorrect privileges, Incorrect authorization

5.1.4 CrossBroker-2009-0004

- Summary: The Crossbroker is prone to a Denial of Service vulnerability. As a result of this attack, Crossbroker will not be able to process the submission of the user jobs, being necessary to stop and restart the Crossbroker host.
- Vulnerable Versions: 0 - 0.5.35
- Components: i2g-job-submit, i2g-wl-ns_daemon, i2g-wl-bkserverd
- Access Required: This vulnerability requires the user to be able to submit jobs to CrossBroker.
- Effort Required: Low.

  Exploiting this vulnerability requires the user to be able to write and submit a job with a big string ($\geq$ 64Kb) as the Arguments value in the JDL file.
- Impact/Consequences: High.

  If the attacker is successful, any other job who arrives to the Crossbroker will not be processed and therefore will not be submitted to the grid world.

Moreover the Crossbroker host should be stopped and restarted to gain the services work correctly, it means a lot of jobs will be stopped and may lost their results, and should be necessary that jobs submitted before the attack have to submit again.

- Cause: Improper error handling, Inability to handle missing field or value, Inability to handle invalid field or value

Up to now, the security practitioners who applied FPVA on several middleware provided their own expertise and creativity about different kind of security problems over the key structural components identified on FPVA diagrams, without further guidance or information.

5.2 CrossBroker & AvA4cmi

In this section, we present the security alerts produced for the attack vectors after the assessment process performed to CrossBroker following the *AvA4cmi* guidelines. Figure 5.1 shows the attack vector graph of CrossBroker, from which

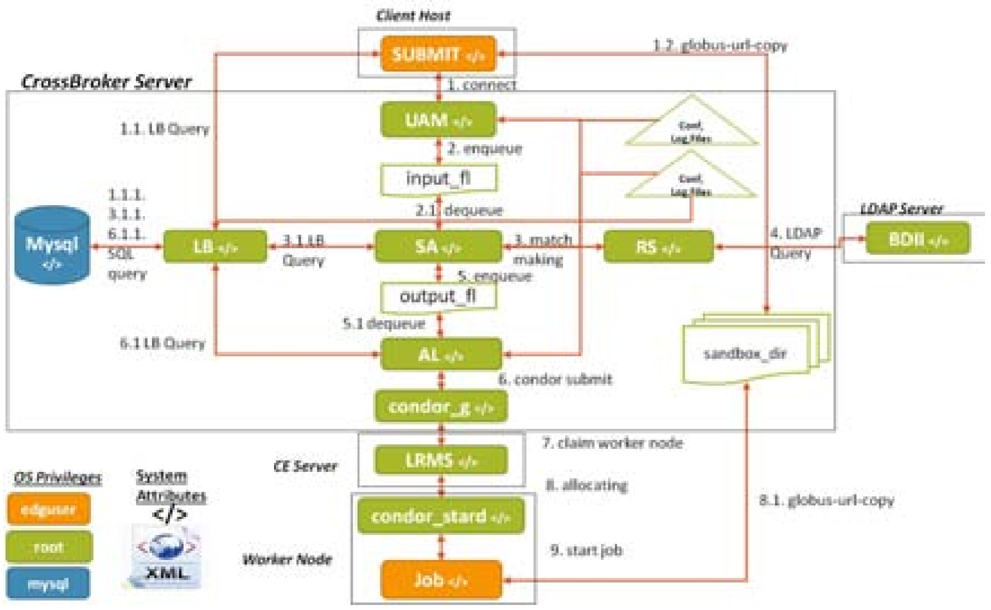


Figure 5.1: Attack Vector graph: CrossBroker

the most likely attack vectors were depicted in its corresponding *graphml* format.

It is worth to state that the number of attack vectors not only depends on the number of attack and impact surfaces components, but also in the security analyst conditions to consider strongly that a component is part of either attack or impact surface.

In order to carry out the assessment process, a prototype tool was implemented in Python, which is available in a GIT repository [43].

For the sake of simplicity, in this experimental evaluation, we introduce one attack vector of CrossBroker, with all the analysis of their security alerts. For the CrossBroker attack vectors II, III, IV, VI, VIII, IX, and X, we summarized their complete security alerts in a graphical representation, which are available too at the GIT repository.

For the attack vectors V and VII, the security alerts could not be derived into any of the known vulnerabilities found with FPVA, firstly, due that FPVA did not find vulnerabilities related to these attack vectors, and secondly, there are no relationships between the FPVA vulnerabilities and the impact surface of the attack vectors V and VII. But despite not exist relationship, security analyst must consider the security alerts reported, in the case of a new analysis will be performed, because some of them belong to the Top 25 of dangerous weaknesses, and they are scored the highest.

5.2.1 Attack vector I

The attack vector I is composed by 10 components (shown in Figure 5.2), starting with the ***submit*** component which belong to the attack surface, passing through the ***network server (UAM), input queue (input_fl), scheduling agent (SA), output queue (output_fl), application launcher (AL), condor daemon (Condor-G), local resource management system (LRMS), condor daemon (condor_startd)***, until achieve its impact surface the ***job*** component.

The security alerts for this attack vector are presented below.

Security Alerts for CWE-664

For the pillar CWE-664, we illustrate the first 15 high score security alerts in the Code 5.1.

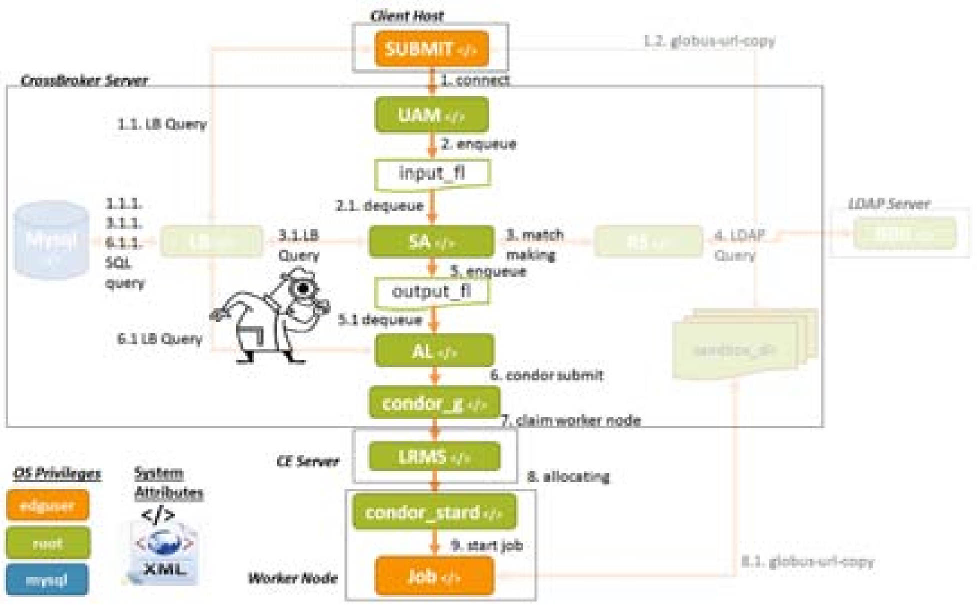


Figure 5.2: Attack Vector I for CrossBroker

```
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――
CWE-664: Improper Control of a Resource Through its Lifetime
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――

1. CWE-863 Incorrect Authorization:
      (Sp= 1.455; Max= 1.455;)
2. CWE-862 Missing Authorization:
      (Sp= 1.455; Max= 1.455)
   |
   -->CWE-638 Not Using Complete Mediation:
      (Sp= 0.555; Max= 0.7825)
        |
        -->CWE-424 Improper Protection of Alternate Path:
      (Sp= 0.7825; Max= 0.7825)
   |
   -->CWE-639 Authorization Bypass Through User-Controlled Key:
      (Sp= 0.65; Max= 0.65)
3. CWE-732 Incorrect Permission Assignment for Critical Resource:
      (Sp= 1.455; Max= 1.455)
   |
   -->CWE-281 Improper Preservation of Permissions:
      (Sp= 0.7825; Max= 0.7825)
   |
 -->CWE-276 Incorrect Default Permissions:
     (Sp= 0.5125; Max= 0.5125)
  |
 -->CWE-279 Incorrect Execution-Assigned Permissions:
     (Sp= 0.7825; Max= 0.7825)
  |
 -->CWE-278 Insecure Preserved Inherited Permissions:
     (Sp= 0.6366; Max= 0.6366)
  |
 -->CWE-277 Insecure Inherited Permissions:
     (Sp= 0.636; Max= 0.636)
4. CWE-306 Missing Authentication for Critical Function:
     (Sp= 1.455; Max= 1.455)
5. CWE-782 Exposed IOCTL with Insufficient Access Control:
     (Sp= 1.165; Max= 1.165)
6. CWE-307 Improper Restriction of Excessive Authentication Attempts:
     (Sp= 1.055; Max= 1.055)
7. CWE-250 Execution with Unnecessary Privileges:
     (Sp= 0.865625; Max= 0.865625)
8. CWE-558 Use of getlogin() in Multithreaded
     Application: (Sp= 0.7875; Max= 0.7875)
9. CWE-543 Use of Singleton Pattern Without Synchronization in a
     Multithreaded Context: (Sp= 0.7875; Max= 0.7875)
10. CWE-708 Incorrect Ownership Assignment:
     (Sp= 0.7825; Max= 0.7825)
11. CWE-648 Incorrect Use of Privileged APIs:
     (Sp= 0.7825; Max= 0.7825)
12. CWE-62 UNIX Hard Link:
     (Sp= 0.7825; Max= 0.7825)
13. CWE-424 Improper Protection of Alternate Path:
     (Sp= 0.7825; Max= 0.7825)
14. CWE-305 Authentication Bypass by Primary Weakness:
     (Sp= 0.7825; Max= 0.7825)
15. CWE-289 Authentication Bypass by Alternate Name:
     (Sp= 0.7825; Max= 0.7825)
```

Code 5.1: Partial list of CWE-664 s Security Alerts

From the whole security alerts, we understand that around the 20% of 150 weaknesses that compose the CWE-664 pillar, can derive into one vulnerability manually found with FPVA, and reviewing the first 15 weaknesses with the highest scores, we found the next relationships:

- For CrossBroker-2009-0001 : CWE-648.
- For CrossBroker-2009-0002 : CWE-863, CWE-732, CWE-306, CWE-250, and CWE-708.
- And, for CrossBroker-2009-0003 : CWE-862, CWE-424, and CWE-305.

For the group comprised between the 15th and 90th weakness, 20 weaknesses with lower scores can derive into one of the vulnerabilities found with FPVA, but their contributions are not enough to be considered remarkable; and for the rest of weaknesses no matches were found for this attack vector. The Figure 5.3 represents this distribution.

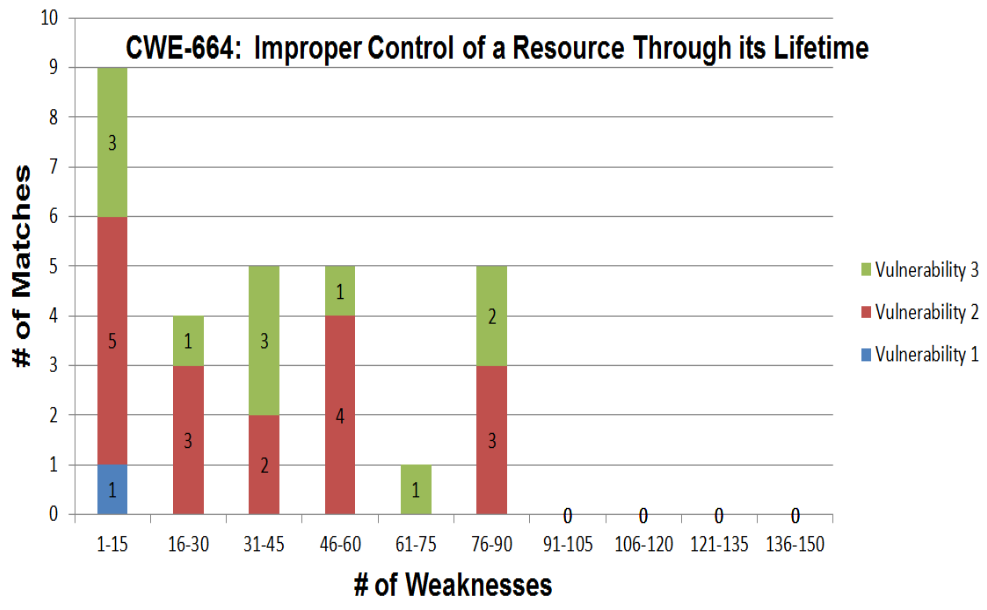


Figure 5.3: Distribution of weaknesses for CWE-664 Security Alerts in attack vector I.

For example, reviewing the weakness in first position with the highest score, the CWE-862: Missing Authorization , whose description is *"The software does not perform an authorization check when an actor attempts to access a resource or perform an action"*, and reviewing again the CrossBroker vulnerabilities manually found with FPVA, along with the description for the CWE-664 pillar, *"The software does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release"*, it can be seen the straight relationship between the weakness-pillar and the CrossBroker-2009-0003 vulnerability regarding to the authentication and authorization underlying mechanisms for the components in the attack vector.

In summary, from this CWE-664 security alerts, 11 weaknesses are related with the vulnerability CrossBroker-2009-0003 , 17 weaknesses are related with the vulnerability CrossBroker-2009-0002 , and one weakness is related with the vulnerability CrossBroker-2009-0001 . For these security alerts there are no matches for CrossBroker-2009-0004 in this attack vector, due that its underlying cause belong to other middleware components.

Security Alerts for CWE-693

For the pillar CWE-693, we illustrate the the whole security alerts in the Code 5.2.

```
CWE-693: Protection Mechanism Failure

1. CWE-863 Incorrect Authorization:
    (Sp= 1.9325; Max= 1.9325)
2. CWE-862 Missing Authorization:
    (Sp= 1.9325; Max= 1.9325)
  |
  -->CWE-638 Not Using Complete Mediation:
    (Sp= 0.849; Max= 1.173)
       |
       -->CWE-424 Improper Protection of Alternate Path:
    (Sp= 1.173; Max= 1.173)
  |
  -->CWE-639 Authorization Bypass Through User-Controlled
     Key: (Sp= 1.1675; Max= 1.1675)
3. CWE-732 Incorrect Permission Assignment for Critical
    Resource: (Sp= 1.9325; Max= 1.9325)
  |
  -->CWE-281 Improper Preservation of Permissions:
    (Sp= 1.173; Max= 1.173)
  |
  -->CWE-276 Incorrect Default Permissions:
    (Sp= 0.925; Max= 0.925)
  |
  —>CWE−279 Incorrect Execution−Assigned Permissions:
     (Sp= 1.173; Max= 1.173)
  |
  —>CWE−278 Insecure Preserved Inherited Permissions:
     (Sp= 0.955; Max= 0.955)
  |
  —>CWE−277 Insecure Inherited Permissions:
    (Sp= 0.955; Max= 0.955)
4. CWE−306 Missing Authentication for Critical Function:
    (Sp= 1.9325; Max= 1.9325)
5. CWE−250 Execution with Unnecessary Privileges:
    (Sp= 1.355; Max= 1.355)
6. CWE−307 Improper Restriction of Excessive Authentication
     Attempts: (Sp= 1.316; Max= 1.316)
7. CWE−708 Incorrect Ownership Assignment:
    (Sp= 1.17375; Max= 1.17375)
8. CWE−655 Insufficient Psychological Acceptability:
    (Sp= 1.17375; Max= 1.17375)
9. CWE−648 Incorrect Use of Privileged APIs:
    (Sp= 1.17375; Max= 1.17375)
10. CWE−424 Improper Protection of Alternate Path:
    (Sp= 1.17375; Max= 1.17375)
11. CWE−305 Authentication Bypass by Primary Weakness:
    (Sp= 1.17375; Max= 1.17375)
12. CWE−289 Authentication Bypass by Alternate Name:
    (Sp= 1.17375; Max= 1.17375)
13. CWE−281 Improper Preservation of Permissions:
    (Sp= 1.17375; Max= 1.17375)
14. CWE−279 Incorrect Execution−Assigned Permissions:
    (Sp= 1.17375; Max= 1.17375)
15. CWE−274 Improper Handling of Insufficient Privileges:
     (Sp= 1.17375; Max= 1.17375)
```

Code 5.2: List of CWE-693 s Security Alerts

From the whole security alerts, we understand that around the 17% of 97 weaknesses that compose the CWE-693 pillar, can derive into one vulnerability manually found with FPVA, and reviewing the first 15 weaknesses with the highest scores, we found the next relationships:

- For CrossBroker-2009-0001 : CWE-648.
- For CrossBroker-2009-0002 : CWE-863, CWE-732, CWE-306, CWE-250,

CWE-281, CWE-279, and CWE-708.

- And, for CrossBroker-2009-0003 : CWE-862, CWE-424, CWE-305, and CWE-655.

For the group comprised between the 15th and 97th weakness, 4 weaknesses with lower scores can derive into one of the vulnerabilities found with FPVA, but their contributions are not enough to be considered remarkable. The Figure 5.4 represents this distribution.

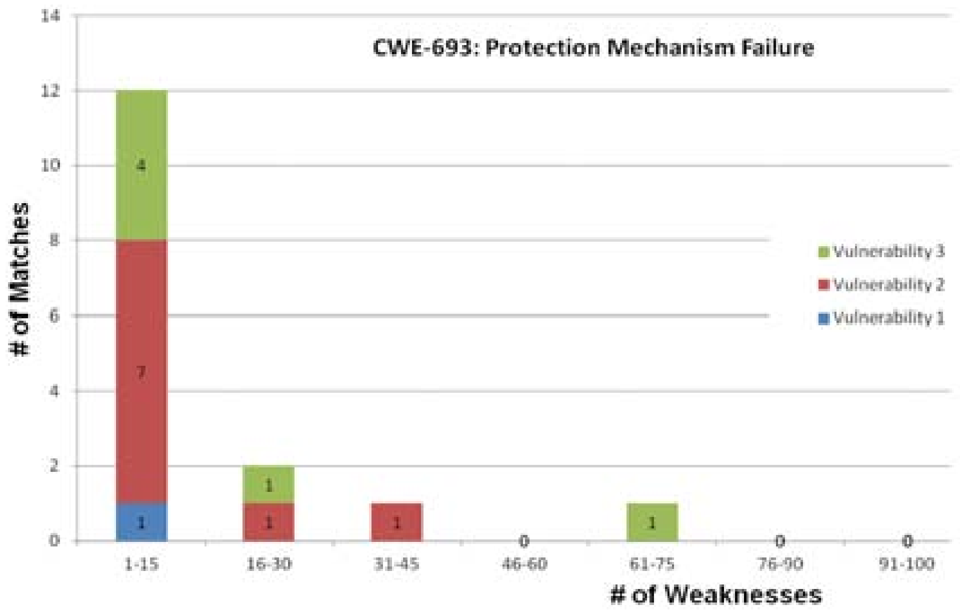


Figure 5.4: Distribution of weaknesses for CWE-693 Security Alerts in attack vector I.

For example, reviewing the weakness in first position with the highest score, the CWE-306: Missing Authentication for Critical Function , whose description is *“The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.”*, and reviewing again the CrossBroker vulnerabilities manually found with FPVA, along with the description for the CWE-693 pillar, *“The product does not use or incorrectly uses a protection mechanism that provides sufficient defense against directed attacks against the product.”*, it can be seen the straight relationship between the weakness-pillar and the

CrossBroker-2009-0002 vulnerability regarding to the missing, insufficient or ignored protection underlying mechanisms for the components in the attack vector.

In summary, from this CWE-693 security alerts, 6 weaknesses are related with the vulnerability CrossBroker-2009-0003 , 9 weaknesses are related with the vulnerability CrossBroker-2009-0002 , and one weakness is related with the vulnerability CrossBroker-2009-0001 . For these security alerts there are no matches for CrossBroker-2009-0004 in this attack vector, due that its underlying cause belong to other middleware components.

Security Alerts for CWE-118

For the pillar CWE-118, we illustrate the the whole security alerts in the Code 5.3.

```
CWE-118: Improper Access of Indexable Resource ('Range Error')

1. CWE-120 Buffer Copy without Checking Size of Input
     ('Classic Buffer Overflow'): (Sp= 1.563; Max= 1.563)
  |
  -->CWE-785 Use of Path Manipulation Function without
     Maximum-sized Buffer: (Sp= 0.7875; Max= 0.7875)
2. CWE-805 Buffer Access with Incorrect Length Value:
     (Sp= 0.7875; Max= 0.7875)
3. CWE-785 Use of Path Manipulation Function without
     Maximum-sized Buffer: (Sp= 0.7875; Max= 0.7875)
4. CWE-127 Buffer Under-read:
     (Sp= 0.7875; Max= 0.7875)
5. CWE-126 Buffer Over-read:
     (Sp= 0.7875; Max= 0.7875)
6. CWE-125 Out-of-bounds Read:
     (Sp= 0.7875; Max= 0.7875)
  |
  -->CWE-126 Buffer Over-read:
     (Sp= 0.7875; Max= 0.7875)
  |
  -->CWE-127 Buffer Under-read:
     (Sp= 0.7875; Max= 0.7875)
7. CWE-124 Buffer Underwrite ('Buffer Underflow'):
     (Sp= 0.7875; Max= 0.7875)
```

```
8. CWE−123 Write−what−where Condition:
     (Sp= 0.7875; Max= 0.7875)
9. CWE−122 Heap−based Buffer Overflow:
     (Sp= 0.7875; Max= 0.7875)
10. CWE−121 Stack−based Buffer Overflow:
     (Sp= 0.7875; Max= 0.7875)
11. CWE−119 Improper Restriction of Operations within the
     Bounds of a Memory Buffer: (Sp= 0.7875; Max= 1.56333333333)
  |
 −−>CWE−805 Buffer Access with Incorrect Length Value:
     (Sp= 0.7875; Max= 0.7875)
  |
 −−>CWE−120 Buffer Copy without Checking Size of Input
     ('Classic Buffer Overflow'): (Sp= 1.563; Max= 1.563)
       |
      −−>CWE−785 Use of Path Manipulation Function without
     Maximum−sized Buffer: (Sp= 0.7875; Max= 0.7875)
  |
 −−>CWE−123 Write−what−where Condition:
     (Sp= 0.7875; Max= 0.7875; Meg= 0.7875)
  |
 −−>CWE−125 Out−of−bounds Read:
     (Sp= 0.7875; Max= 0.7875; Meg= 0.7875)
```

Code 5.3: List of CWE-118 s Security Alerts

From the 11 weaknesses that compose the CWE-118 pillar, we understand that it is likely to exist problems in this attack vector regarding to Range Errors and more specifically to the classic buffer overflow, and despite of it can not be seen an straight relationship between the weaknesses-pillar and the vulnerabilities CrossBroker-2009-0001, 0002, and 0003 manually found with FPVA, the security analyst must take into account due that belong to the Top 25 of dangerous weaknesses, and it is scored the highest. On the contrary, these weaknesses-pillar shown a relationship with vulnerability CrossBroker-2009-0004 , but its underlying causes does not belong to the attack vector I.

Security Alerts for CWE-330

For the pillar CWE-330, we illustrate the the whole security alerts in the Code 5.4.

```
—————————————————————————————————————————————
CWE−330: Use of Insufficiently Random Values
—————————————————————————————————————————————
1. CWE−798 Use of Hard−coded Credentials:
    (Sp= 0.640; Max= 0.640)
  |
  −−>CWE−259 Use of Hard−coded Password:
    (Sp= 0.559; Max= 0.559)
  |
  −−>CWE−321 Use of Hard−coded Cryptographic
    Key: (Sp= 0.1125; Max= 0.1125)
2. CWE−259 Use of Hard−coded Password:
    (Sp= 0.559; Max= 0.559)
3. CWE−343 Predictable Value Range from Previous
    Values: (Sp= 0.1125; Max= 0.1125)
4. CWE−342 Predictable Exact Value from Previous
    Values: (Sp= 0.1125; Max= 0.1125)
5. CWE−339 Small Seed Space in PRNG:
    (Sp= 0.1125; Max= 0.1125)
6. CWE−338 Use of Cryptographically Weak PRNG:
    (Sp= 0.1125; Max= 0.1125)
7. CWE−337 Predictable Seed in PRNG:
    (Sp= 0.1125; Max= 0.1125)
8. CWE−336 Same Seed in PRNG:
    (Sp= 0.1125; Max= 0.1125)
9. CWE−335 PRNG Seed Error:
    (Sp= 0.1125; Max= 0.1125)
  |
  −−>CWE−339 Small Seed Space in PRNG:
    (Sp= 0.1125; Max= 0.1125)
  |
  −−>CWE−336 Same Seed in PRNG:
     (Sp= 0.1125; Max= 0.1125)
  |
  −−>CWE−337 Predictable Seed in PRNG:
    (Sp= 0.1125; Max= 0.1125)
10. CWE−334 Small Space of Random Values:
     (Sp= 0.1125; Max= 0.1125)
11. CWE−333 Improper Handling of Insufficient Entropy
     in TRNG: (Sp= 0.1125; Max= 0.1125)
12. CWE−332 Insufficient Entropy in PRNG:
     (Sp= 0.1125; Max= 0.1125)
```

```
13. CWE-331 Insufficient Entropy:
     (Sp= 0.1125; Max= 0.1125)
   |
  -->CWE-333 Improper Handling of Insufficient Entropy in TRNG:
      (Sp= 0.1125; Max= 0.1125)
   |
  -->CWE-332 Insufficient Entropy in PRNG:
      (Sp= 0.1125; Max= 0.1125)
14. CWE-329 Not Using a Random IV with CBC Mode:
      (Sp= 0.1125; Max= 0.1125)
15. CWE-323 Reusing a Nonce Key Pair in Encryption:
      (Sp= 0.1125; Max= 0.1125)
16. CWE-321 Use of Hard-coded Cryptographic Key:
      (Sp= 0.1125; Max= 0.1125)
```

Code 5.4: List of CWE-330 s Security Alerts

From the 16 weaknesses that compose the CWE-330 pillar, we understand that it is likely to exist problems in this attack vector regarding to Random Values and more specifically to the Use of Hard-coded Credentials. By the moment FPVA was applied to CrossBroker, vulnerabilities regarding to the use of credentials were not found. Since then, several vulnerabilities have been found caused by processing of X.509 certificates. If CrossBroker were used at the present time, it would be imperative to the security analyst review again the related code functionality for the use of Credentials.

Security Alerts for CWE-435

For the pillar CWE-435, we illustrate the the whole security alerts in the Code 5.5.

```
______________________________
CWE-435: Interaction Error
______________________________
1. CWE-115 Misinterpretation of Input:
     (Sp= 0.835; Max= 0.835)
2. CWE-436 Interpretation Conflict:
     (Sp= 0.75; Max= 0.835)
   |
  -->CWE-115 Misinterpretation of Input:
     (Sp= 0.835; Max= 0.835)
```

```
3. CWE-14 Compiler Removal of Code to Clear Buffers:
     (Sp= 0.26; Max= 0.26)
4. CWE-198 Use of Incorrect Byte Ordering:
    (Sp= 0.2575; Max= 0.2575)
5. CWE-733 Compiler Optimization Removal or Modification
     of Security-critical Code: (Sp= 0.16; Max= 0.26)
  |
  -->CWE-14 Compiler Removal of Code to Clear Buffers:
     (Sp= 0.26; Max= 0.26)
```

Code 5.5: List of CWE-435 s Security Alerts

From the whole security alerts that compose the CWE-435 pillar, we understand that the first two high score weaknesses CWE-115, and CWE-436, can derive into vulnerability CrossBroker-2009-0001 . Also, the weakness CWE-198 can derive into vulnerability CrossBroker-2009-0004 , but its underlying causes does not belong to the attack vector I; and for the rest of the weaknesses no matches were found.

Security Alerts for CWE-682

For the pillar CWE-682, we illustrate the the whole security alerts in the Code 5.6.

```
________________________________
CWE-682: Incorrect Calculation
________________________________
1. CWE-131 Incorrect Calculation of Buffer Size:
    (Sp= 1.61; Max= 1.61)
2. CWE-190 Integer Overflow or Wraparound:
    (Sp= 1.17; Max= 1.17)
3. CWE-191 Integer Underflow (Wrap or Wraparound):
    (Sp= 0.525; Max= 0.525)
4. CWE-135 Incorrect Calculation of Multi-Byte String
     Length: (Sp= 0.47; Max= 0.47)
5. CWE-128 Wrap-around Error:
    (Sp= 0.345; Max= 0.345)
6. CWE-193 Off-by-one Error:
     (Sp= 0.1475; Max= 0.1475)
```

Code 5.6: List of CWE-682 s Security Alerts

From the six weaknesses that compose the CWE-682 pillar, we understand that it is likely to exist problems in this attack vector regarding to Incorrect Calculation and more specifically to buffer size, and despite of it can not be seen an straight relationship between the weaknesses-pillar and the vulnerabilities CrossBroker-2009-0001, 0002, and 0003 manually found with FPVA, the security analyst must take into account due that belong to the Top 25 of dangerous weaknesses, and it is scored the highest. On the contrary, these weaknesses-pillar shown a relationship with vulnerability CrossBroker-2009-0004 , but its underlying causes does not belong to the attack vector I.

Security Alerts for CWE-691

For the pillar CWE-691, we illustrate the the whole security alerts in the Code 5.7.

```
_________________________________________________
CWE-691: Insufficient Control Flow Management
_________________________________________________
1. CWE-782 Exposed IOCTL with Insufficient Access Control:
     (Sp= 1.165; Max= 1.165)
2. CWE-307 Improper Restriction of Excessive Authentication
     Attempts: (Sp= 1.026; Max= 1.026)
3. CWE-837 Improper Enforcement of a Single Unique Action:
     (Sp= 0.7825; Max= 0.7825)
4. CWE-799 Improper Control of Interaction Frequency:
     (Sp= 0.565; Max= 1.026)
  |
 -->CWE-307 Improper Restriction of Excessive Authentication Attempts:
     (Sp= 1.026; Max= 1.026)
  |
 -->CWE-837 Improper Enforcement of a Single Unique Action:
      (Sp= 0.7825; Max= 0.7825)
5. CWE-421 Race Condition During Access to Alternate Channel:
     (Sp= 0.565; Max= 0.565)
6. CWE-558 Use of getlogin() in Multithreaded Application:
      (Sp= 0.525; Max= 0.525)
7. CWE-543 Use of Singleton Pattern Without Synchronization
    in a Multithreaded Context: (Sp= 0.525; Max= 0.525)
```

```
8. CWE-366 Race Condition within a Thread:
     (Sp= 0.525; Max= 0.525)
9. CWE-365 Race Condition in Switch:
     (Sp= 0.525; Max= 0.525)
10. CWE-364 Signal Handler Race Condition:
     (Sp= 0.525; Max= 0.525)
  |
  -->CWE-432 Dangerous Signal Handler not Disabled During Sensitive
      Operations: (Sp= 0.13; Max= 0.13)
11. CWE-96 Improper Neutralization of Directives in Statically
     Saved Code ('Static Code Injection'): (Sp= 0.42416; Max= 0.42416)
12. CWE-179 Incorrect Behavior Order: Early Validation:
     (Sp= 0.26; Max= 0.26)
13. CWE-705 Incorrect Control Flow Scoping:
     (Sp= 0.2; Max= 0.2)
  |
  -->CWE-455 Non-exit on Failed Initialization:
     (Sp= 0.11; Max= 0.11)
  |
  -->CWE-248 Uncaught Exception:
     (Sp= 0.155; Max= 0.155)
14. CWE-367 Time-of-check Time-of-use (TOCTOU) Race
     Condition: (Sp= 0.17083; Max= 0.525)
  |
  -->CWE-365 Race Condition in Switch:
     (Sp= 0.525; Max= 0.525)
  |
  -->CWE-363 Race Condition Enabling Link Following:
     (Sp= 0.13; Max= 0.13)
15. CWE-248 Uncaught Exception:
     (Sp= 0.155; Max= 0.155)
```

Code 5.7: List of CWE-691 s Security Alerts

From the whole weaknesses that compose the CWE-691 pillar, we understand that it is likely to exist problems in this attack vector regarding to Insufficient Control Flow Management , in particular the weaknesses CWE-837, and CWE-705, can derive into vulnerability CrossBroker-2009-0003 . Also, the weakness CWE-96 can derive into vulnerability CrossBroker-2009-0001 . On the contrary, the weaknesses CWE-248 can derive into vulnerability CrossBroker-2009-0004 , but its underlying causes does not belong to the attack vector I.

Security Alerts for CWE-697

For the pillar CWE-697, we illustrate the the whole security alerts in the Code 5.8.

```
_____________________________________________
CWE-697: Insufficient Comparison
_____________________________________________
1. CWE-184 Incomplete Blacklist:
   (Sp= 0.778; Max= 0.778)
2. CWE-183 Permissive Whitelist:
   (Sp= 0.778; Max= 0.778)
3. CWE-186 Overly Restrictive Regular Expression:
   (Sp= 0.415; Max= 0.415)
4. CWE-185 Incorrect Regular Expression:
   (Sp= 0.415; Max= 0.415)
  |
 -->CWE-186 Overly Restrictive Regular Expression:
    (Sp= 0.415; Max= 0.415)
```

Code 5.8: List of CWE-697 s Security Alerts

From the whole weaknesses that compose the CWE-697 pillar, we understand that it is likely to exist problems in this attack vector regarding to Insufficient Comparison , in particular the weaknesses CWE-184, and CWE-183, can derive into vulnerability CrossBroker-2009-0003.

Security Alerts for CWE-703

For the pillar CWE-703, we illustrate the the whole security alerts in the Code 5.9.

```
_____________________________________________
CWE-703: Improper Check or Handling of Exceptional Conditions
_____________________________________________
1. CWE-274 Improper Handling of Insufficient Privileges:
    (Sp= 1.173; Max= 1.173)
2. CWE-273 Improper Check for Dropped Privileges:
    (Sp= 1.17375; Max= 1.17375)
3. CWE-280 Improper Handling of Insufficient Permissions
    or Privileges : (Sp= 0.815; Max= 0.815)
```

```
4. CWE-236 Improper Handling of Undefined Parameters:
    (Sp= 0.415; Max= 0.415)
5. CWE-235 Improper Handling of Extra Parameters:
    (Sp= 0.415; Max= 0.415)
6. CWE-234 Failure to Handle Missing Parameter:
    (Sp= 0.415; Max= 0.415)
7. CWE-232 Improper Handling of Undefined Values:
    (Sp= 0.415; Max= 0.415)
8. CWE-231 Improper Handling of Extra Values:
    (Sp= 0.415; Max= 0.415)
9. CWE-168 Improper Handling of Inconsistent Special Elements:
     (Sp= 0.415; Max= 0.415)
10. CWE-167 Improper Handling of Additional Special Element:
     (Sp= 0.38; Max= 0.38)
11. CWE-166 Improper Handling of Missing Special Element:
    (Sp= 0.38; Max= 0.38)
12. CWE-370 Missing Check for Certificate Revocation after
     Initial Check: (Sp= 0.345; Max= 0.345)
13. CWE-299 Improper Check for Certificate Revocation:
    (Sp= 0.345; Max= 0.345)
  |
  -->CWE-370 Missing Check for Certificate Revocation
    after Initial Check: (Sp= 0.345; Max= 0.345)

14. CWE-298 Improper Validation of Certificate Expiration:
     (Sp= 0.345; Max= 0.345)
15. CWE-296 Improper Following of Chain of Trust for
    Certificate Validation: (Sp= 0.345; Max= 0.345)
```

Code 5.9: List of CWE-703 s Security Alerts

From the whole weaknesses that compose the CWE-703 pillar, we understand that around the 35% of 31 weaknesses that compose the CWE-703 pillar, can derive into one vulnerability manually found with FPVA , and reviewing the first 15 weaknesses with the highest scores, we found the next relationships:

- For CrossBroker-2009-0001 : CWE-236, CWE-235, CWE-234, CWE-232, CWE-231, CWE-168, CWE-167, CWE-166
- For CrossBroker-2009-0002 : CWE-274, CWE-273, and CWE-280
- And, for CrossBroker-2009-0003 no matches.

Security Alerts for CWE-707

For the pillar CWE-707, we illustrate the the whole security alerts in the Code 5.10.

```
________________________________________________________
CWE-707: Improper Enforcement of Message or Data Structure
________________________________________________________
1. CWE-134 Uncontrolled Format String:
    (Sp= 1.331; Max= 1.331)
2. CWE-78 Improper Neutralization of Special Elements used in an
    OS Command ('OS Command Injection'): (Sp= 0.882; Max= 0.882)
3. CWE-89 Improper Neutralization of Special Elements used in an
    SQL Command ('SQL Injection'): (Sp= 0.805; Max= 0.805)
4. CWE-641 Improper Restriction of Names for Files and Other
    Resources: (Sp= 0.778; Max= 0.778)
5. CWE-76 Improper Neutralization of Equivalent Special Elements:
     (Sp= 0.6225; Max= 0.6225)
6. CWE-75 Failure to Sanitize Special Elements into a Different
     Plane (Special Element Injection): (Sp= 0.6225; Max= 0.6225)
  |
 -->CWE-76 Improper Neutralization of Equivalent Special
    Elements: (Sp= 0.6225; Max= 0.6225)
7. CWE-56 Path Equivalence: 'filedir*' (Wildcard):
     (Sp= 0.6225; Max= 0.6225)
8. CWE-54 Path Equivalence: 'filedir' (Trailing Backslash):
     (Sp= 0.6225; Max= 0.6225)
9. CWE-53 Path Equivalence: 'multiple internal backslash':
     (Sp= 0.6225; Max= 0.6225)
10. CWE-52 Path Equivalence: '/multiple/trailing/slash//':
     (Sp= 0.6225; Max= 0.6225)
11. CWE-50 Path Equivalence: '//multiple/leading/slash':
    (Sp= 0.6225; Max= 0.6225)
12. CWE-49 Path Equivalence: 'filename/' (Trailing Slash):
    (Sp= 0.6225; Max= 0.6225)
13. CWE-46 Path Equivalence: 'filename ' (Trailing Space):
    (Sp= 0.6225; Max= 0.6225)
14. CWE-45 Path Equivalence: 'file...name' (Multiple Internal Dot):
     (Sp= 0.6225; Max= 0.6225)
15. CWE-43 Path Equivalence: 'filename....' (Multiple Trailing Dot):
     (Sp= 0.6225; Max= 0.6225)
```

Code 5.10: List of CWE-707 s Security Alerts

From the whole weaknesses that compose the CWE-707 pillar, we understand that it is likely to exist problems in this attack vector regarding to Improper Enforcement of Message or Data Structure , in particular the weaknesses CWE-75, CWE-76, CWE-78, and CWE-89, can derive into vulnerability CrossBroker-2009-0001. The security analyst must take them into account due that belong to the Top 25 of dangerous weaknesses, and they are scored the highest.

Security Alerts for CWE-710

For the pillar CWE-710, we illustrate the the whole security alerts in the Code 5.11.

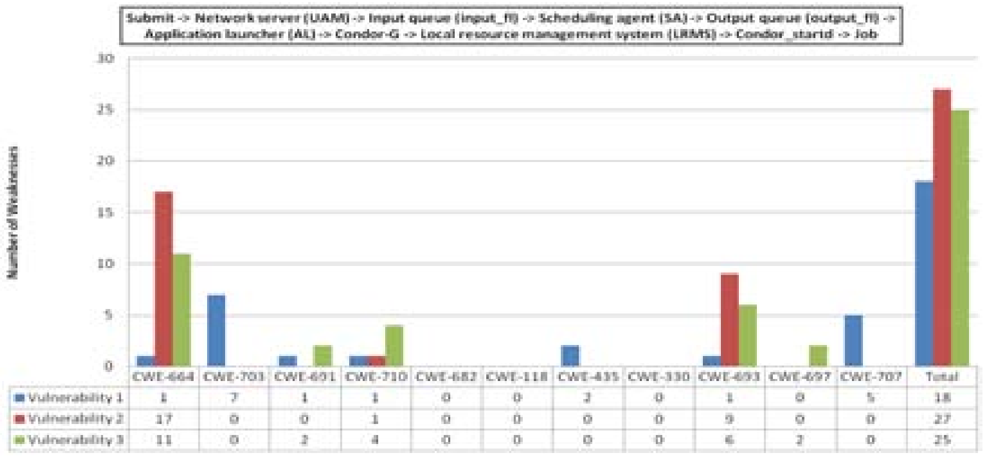
```
_________________________________________
CWE-710: Coding Standards Violation
_________________________________________
1. CWE-250 Execution with Unnecessary Privileges:
    (Sp= 1.2425; Max= 1.2425)
2. CWE-655 Insufficient Psychological Acceptability:
    (Sp= 1.173; Max= 1.173)
3. CWE-648 Incorrect Use of Privileged APIs:
    (Sp= 1.173; Max= 1.173)
4. CWE-637 Unnecessary Complexity in Protection Mechanism
     (Not Using 'Economy of Mechanism'): (Sp= 1.173; Max= 1.173)
5. CWE-424 Improper Protection of Alternate Path:
    (Sp= 1.173; Max= 1.173)
6. CWE-511 Logic/Time Bomb:
    (Sp= 1.039; Max= 1.039)
7. CWE-449 The UI Performs the Wrong Action:
     (Sp= 1.039; Max= 1.039)
8. CWE-448 Obsolete Feature in UI:
    (Sp= 1.039; Max= 1.039)
9. CWE-447 Unimplemented or Unsupported Feature in UI:
     (Sp= 1.039; Max= 1.039)
10. CWE-308 Use of Single-factor Authentication:
    (Sp= 0.89; Max= 0.8)
11. CWE-798 Use of Hard-coded Credentials:
    (Sp= 0.887; Max= 0.887)
   |
  -->CWE-259 Use of Hard-coded Password:
     (Sp= 0.8075; Max= 0.8075)
  |
  —>CWE—321 Use of Hard—coded Cryptographic Key:
    (Sp= 0.075; Max= 0.075)
12. CWE—446 UI Discrepancy for Security Feature:
    (Sp= 0.865; Max= 1.039)
  |
  —>CWE—449 The UI Performs the Wrong Action:
     (Sp= 1.039; Max= 1.039)
  |
  —>CWE—448 Obsolete Feature in UI:
    (Sp= 1.039; Max= 1.039)
  |
  —>CWE—447 Unimplemented or Unsupported Feature in UI:
     (Sp= 1.039; Max= 1.0391)
13. CWE—638 Not Using Complete Mediation:
    (Sp= 0.849; Max= 1.173)
  |
  —>CWE—424 Improper Protection of Alternate Path:
    (Sp= 1.173; Max= 1.173)
14. CWE—259 Use of Hard—coded Password:
    (Sp= 0.8075; Max= 0.8075)
15. CWE—309 Use of Password System for Primary
    Authentication: (Sp= 0.798; Max= 0.798)
```

Code 5.11: List of CWE-710 s Security Alerts

From the whole weaknesses that compose the CWE-710 pillar, we understand that it is likely to exist problems in this attack vector regarding to Coding Standards Violation , and reviewing the first 15 weaknesses with the highest scores, we found the next relationships:

- For CrossBroker-2009-0001 : CWE-648
- For CrossBroker-2009-0002 : CWE-250
- And, for CrossBroker-2009-0003 : CWE-655, CWE-424, and CWE-308.

In summary, by inspecting the whole security alerts for the attack vector I, the *AvA4cmi* assesment process hinted a total of 70 weaknesses strongly related with the vulnerabilities manually found with FPVA , which are distributed as follows: 25 weaknesses related with the vulnerability CrossBroker-2009-0003 , 27 weaknesses related with the vulnerability CrossBroker-2009-0002 , and 18



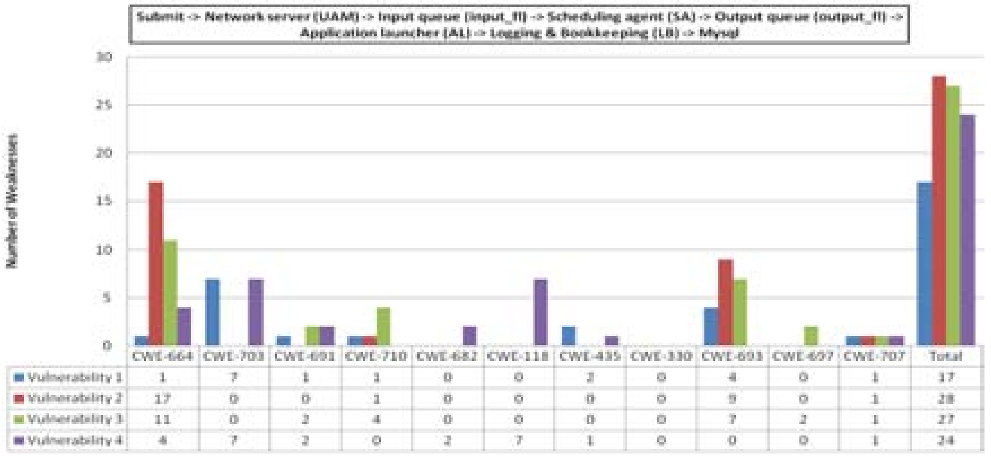
	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	1	0	5	18
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	0	27
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	6	2	0	25

Figure 5.5: Weaknesses-Vulnerabilities Relationship for Attack Vector I

weaknesses related with the vulnerability CrossBroker-2009-0001 . The Figure 5.5 represents this distribution.

5.2.2 Attack vector II

Just like in the attack vector I, the analysis process was carried out over the attack vector II, and it is summarized in the Figure 5.6.

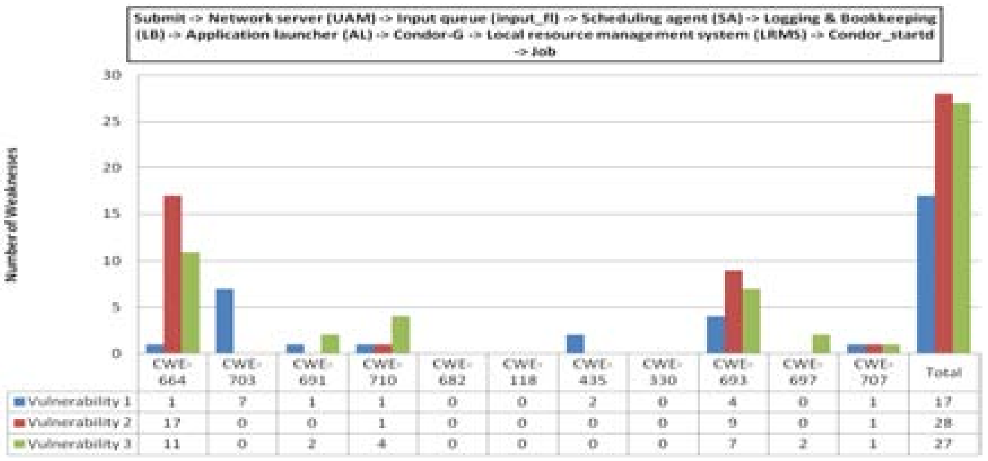


	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	4	0	1	17
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	1	28
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	7	2	1	27
🟪 Vulnerability 4	4	7	2	0	2	7	1	0	0	0	1	24

Figure 5.6: Weaknesses-Vulnerabilities Relationship for Attack Vector II

5.2.3 Attack vector III

Just like in the attack vector I, the analysis process was carried out over the attack vector III, and it is summarized in the Figure 5.7.

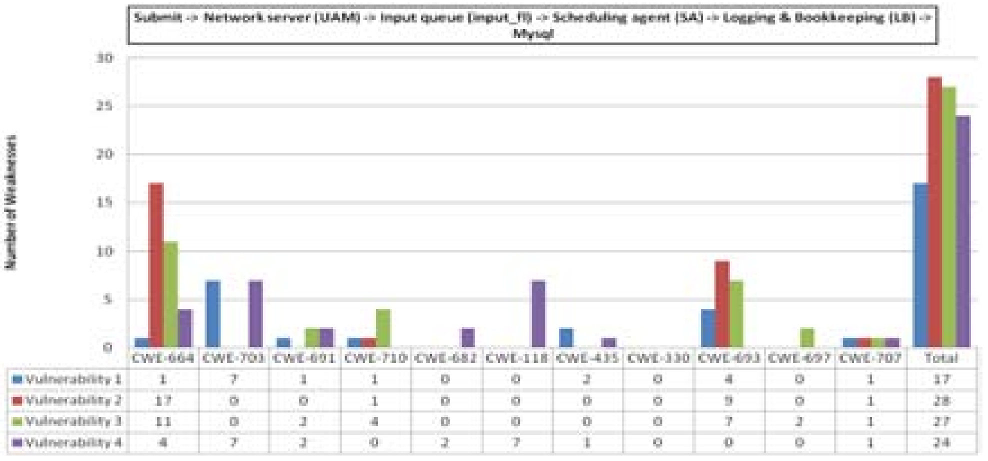


	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	4	0	1	17
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	1	28
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	7	2	1	27

Figure 5.7: Weaknesses-Vulnerabilities Relationship for Attack Vector III

5.2.4 Attack vector IV

Just like in the attack vector I, the analysis process was carried out over the attack vector IV, and it is summarized in the Figure 5.8.

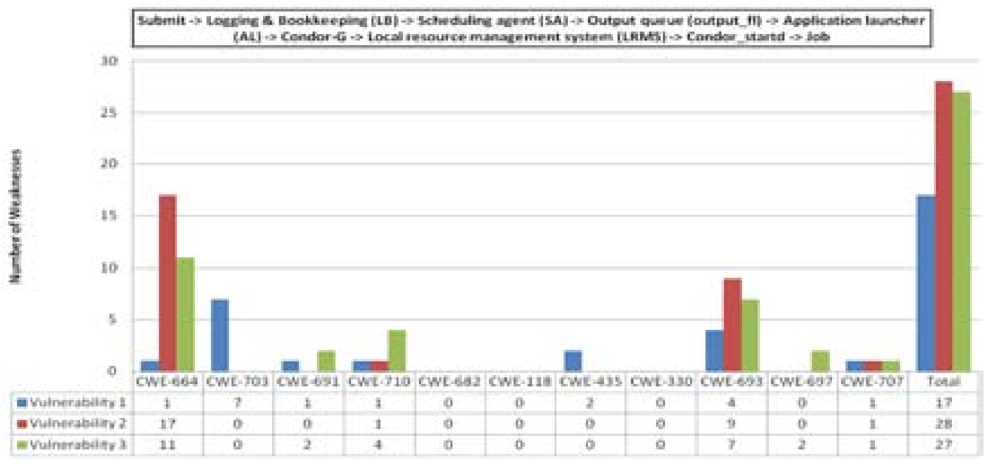


	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	4	0	1	17
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	1	28
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	7	2	1	27
🟪 Vulnerability 4	4	7	2	0	2	7	1	0	0	0	1	24

Figure 5.8: Weaknesses-Vulnerabilities Relationship for Attack Vector IV

5.2.5 Attack vector VI

Just like in the attack vector I, the analysis process was carried out over the attack vector VI, and it is summarized in the Figure 5.9.

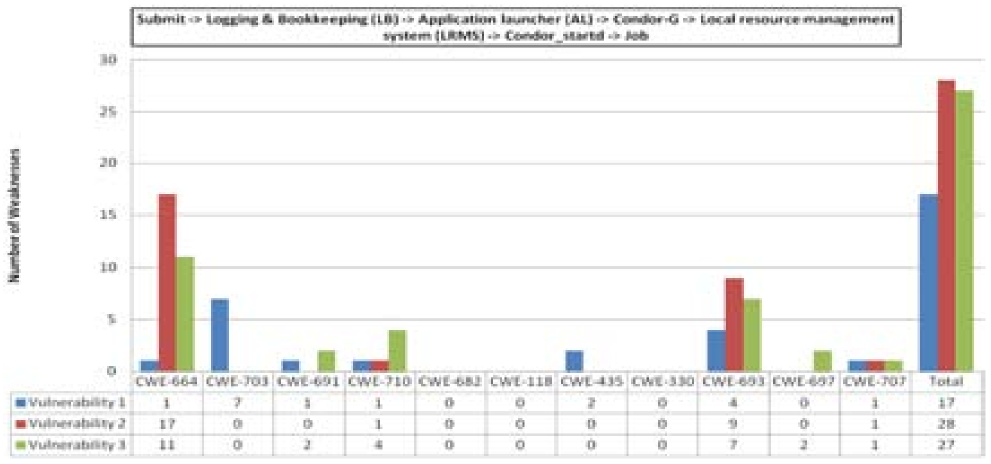


	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	4	0	1	17
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	1	28
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	7	2	1	27

Figure 5.9: Weaknesses-Vulnerabilities Relationship for Attack Vector VI

5.2.6 Attack vector VIII

Just like in the attack vector I, the analysis process was carried out over the attack vector VIII, and it is summarized in the Figure 5.10.

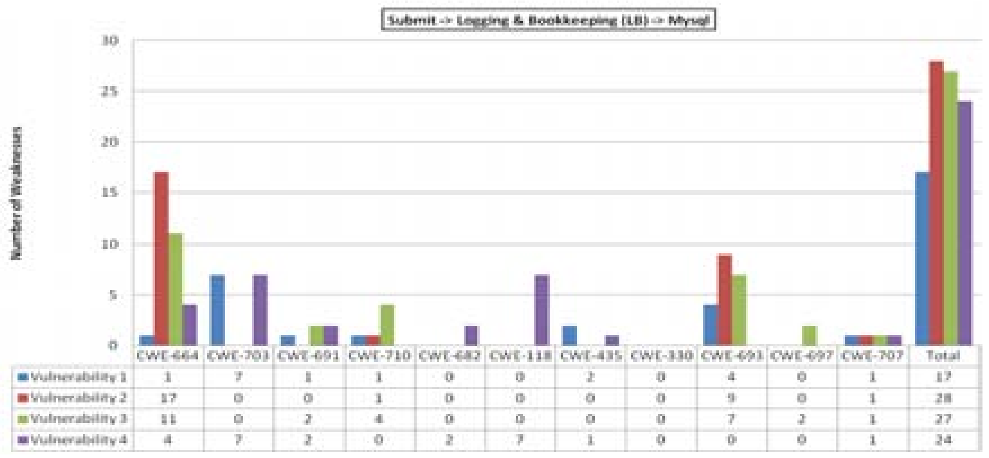


	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	4	0	1	17
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	1	28
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	7	2	1	27

Figure 5.10: Weaknesses-Vulnerabilities Relationship for Attack Vector VIII

5.2.7 Attack vector IX

Just like in the attack vector I, the analysis process was carried out over the attack vector IX, and it is summarized in the Figure 5.11.

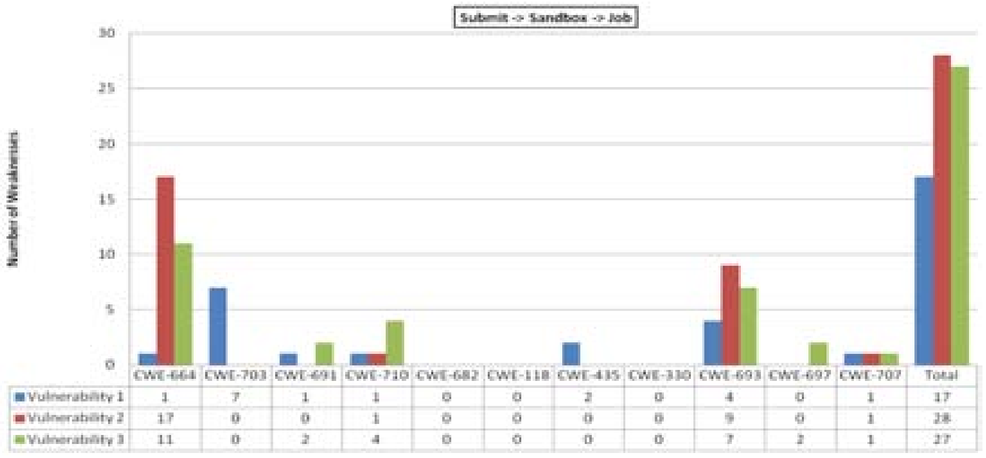


	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	4	0	1	17
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	1	28
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	7	2	1	27
🟪 Vulnerability 4	4	7	2	0	2	7	1	0	0	0	1	24

Figure 5.11: Weaknesses-Vulnerabilities Relationship for Attack Vector IX

5.2.8 Attack vector X

Finally, just like in the attack vector I, the analysis process was carried out over the attack vector X, and it is summarized in the Figure 5.12.



	CWE-664	CWE-703	CWE-691	CWE-710	CWE-682	CWE-118	CWE-435	CWE-330	CWE-693	CWE-697	CWE-707	Total
🟦 Vulnerability 1	1	7	1	1	0	0	2	0	4	0	1	17
🟥 Vulnerability 2	17	0	0	1	0	0	0	0	9	0	1	28
🟩 Vulnerability 3	11	0	2	4	0	0	0	0	7	2	1	27

Figure 5.12: Weaknesses-Vulnerabilities Relationship for Attack Vector X

5.3 gLite WMS & FPVA

The Workload Management System [8] is the gLite component that provides a service responsible for the distribution and management of task across Grid resources, in such a way that applications are conveniently, efficiently, and effectively executed. It is currently used by the European Grid Infrastructure (EGI) [49]. In order to perform the vulnerability assessment following the FPVA approach, a virtualized environment was built by MIST members, and therefore proceed to install the gLite WMS 3.3.5 version.

The MIST members who conducted the FPVA assessment did not identified any serious security problem in gLite WMS, nor did see any security problem with the architecture and implementation.

5.4 gLite WMS& AvA4cmi

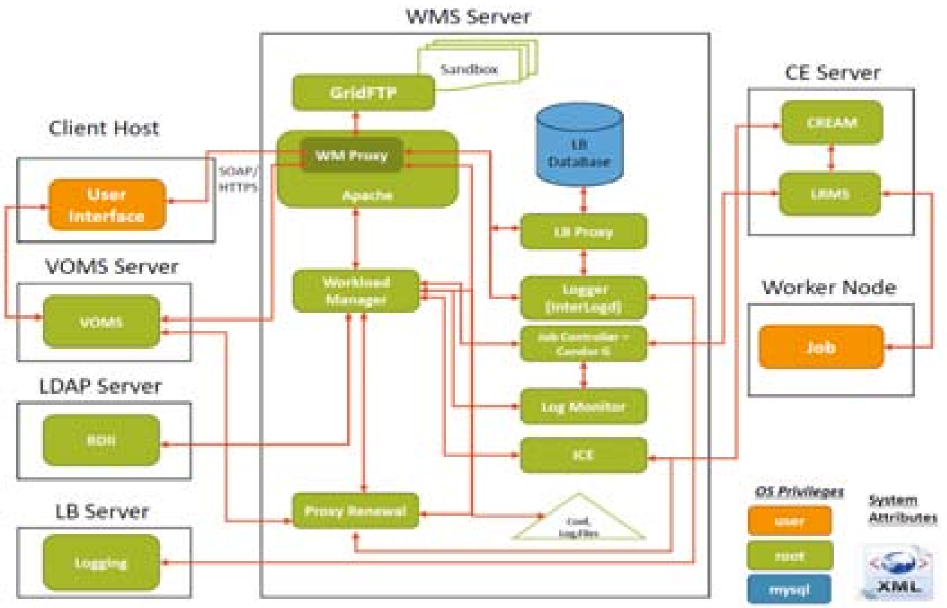


Figure 5.13: Attack Vector graph: gLite WMS

In this section, we present the security alerts produced for the attack vectors after the assessment process performed to gLite WMS following the *AvA4cmi* guidelines.

Indeed, during conversation with the members that conducted the FPVA analysis on gLite WMS, they identified the same attack vectors that the *AvA4cmi* methodology had identified, but without results.

Figure 5.13 shows the attack vector graph of gLite WMS, from which the most likely attack vectors were depicted in its corresponding *graphml* format. The total number of attack vectors depicted is 17.

As already have presented the whole analysis for an attack vector in CrossBroker, and due that there are no vulnerabilities to make contrast with our results, we summarize the most remarkable security alerts according to the CWE top-level entries (pillars) for the whole gLite WMS attack vectors in a graphical representation.

5.4.1 CWE-118 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-118 are summarized in Figure 5.14.

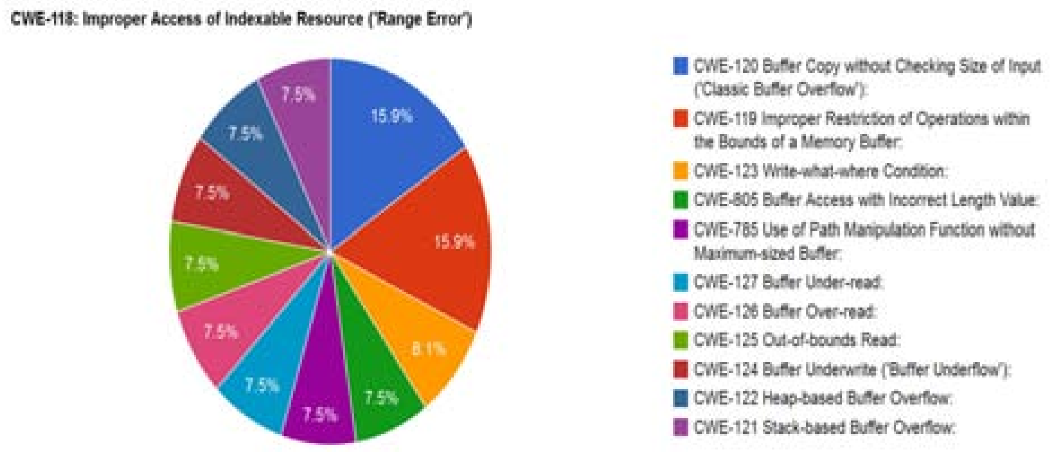


Figure 5.14: Distribution average of the CWE-118 security alerts for gLite WMS

5.4.2 CWE-330 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-330 are summarized in Figure 5.15.

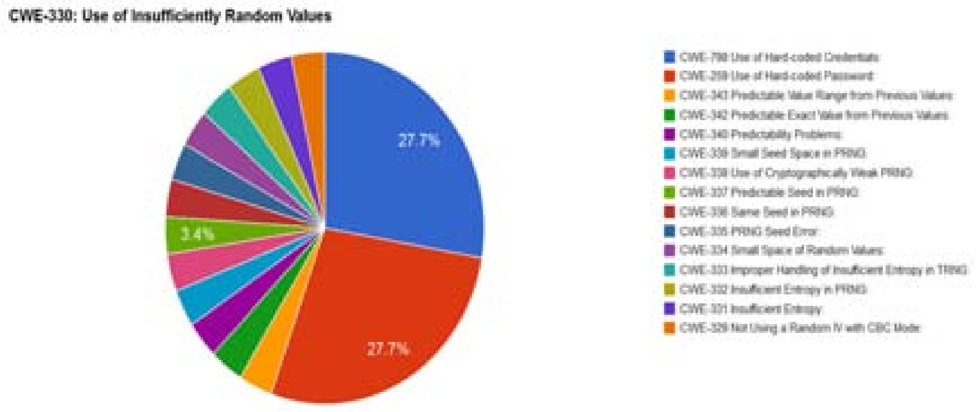


Figure 5.15: Distribution average of the CWE-330 security alerts for gLite WMS

5.4.3 CWE-435 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-435 are summarized in Figure 5.16.

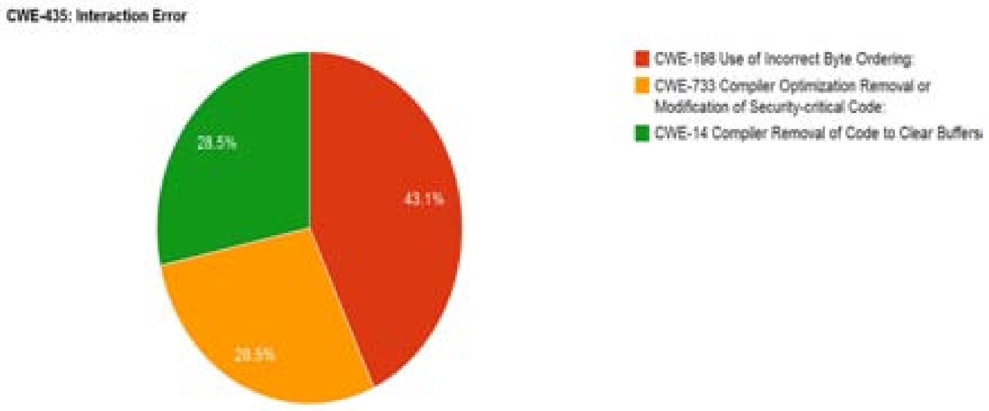


Figure 5.16: Distribution average of the CWE-435 security alerts for gLite WMS

5.4.4 CWE-664 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-664 are summarized in Figure 5.17.

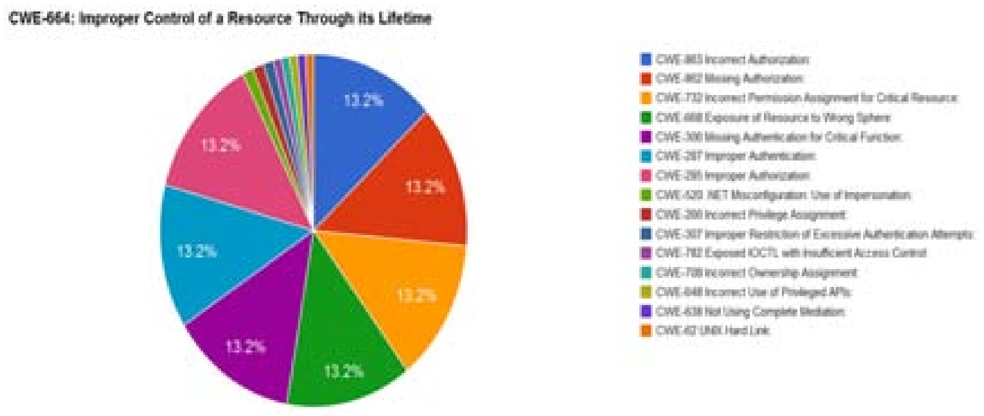


Figure 5.17: Distribution average of the CWE-664 security alerts for gLite WMS

5.4.5 CWE-682 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-682 are summarized in Figure 5.18.

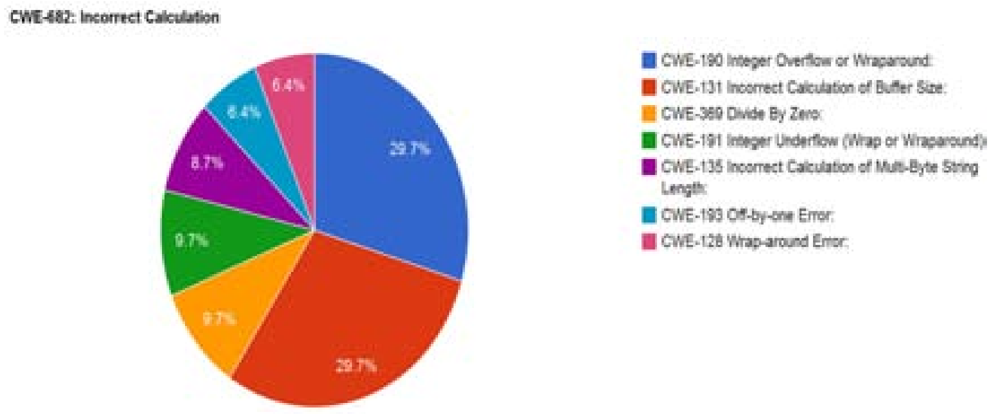


Figure 5.18: Distribution average of the CWE-682 security alerts for gLite WMS

5.4.6 CWE-691 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-691 are summarized in Figure 5.19.

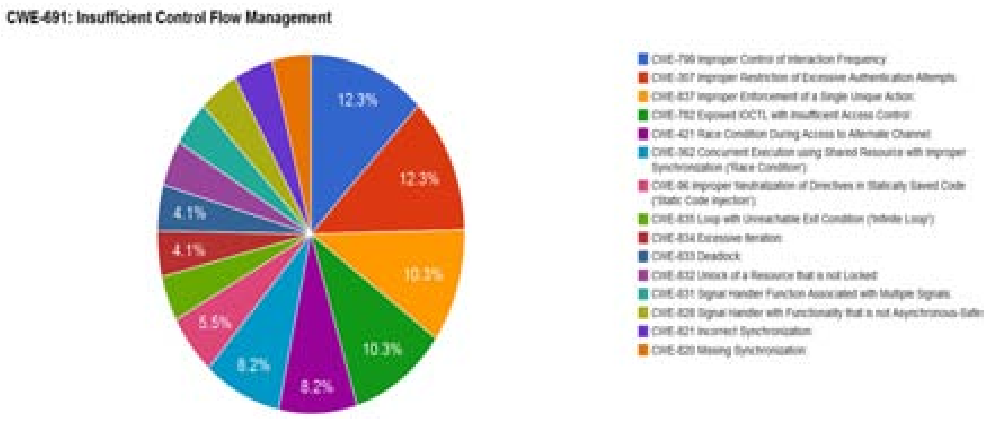


Figure 5.19: Distribution average of the CWE-691 security alerts for gLite WMS

5.4.7 CWE-693 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-693 are summarized in Figure 5.20.

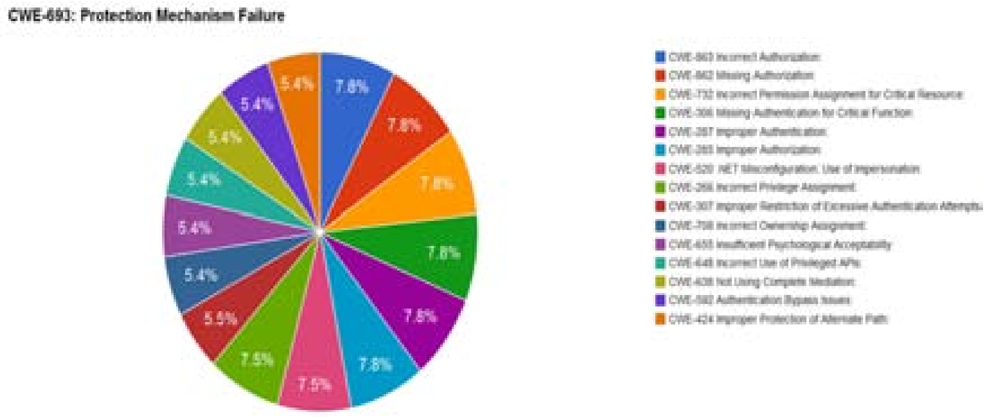


Figure 5.20: Distribution average of the CWE-693 security alerts for gLite WMS

5.4.8 CWE-697 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-697 are summarized in Figure 5.21.

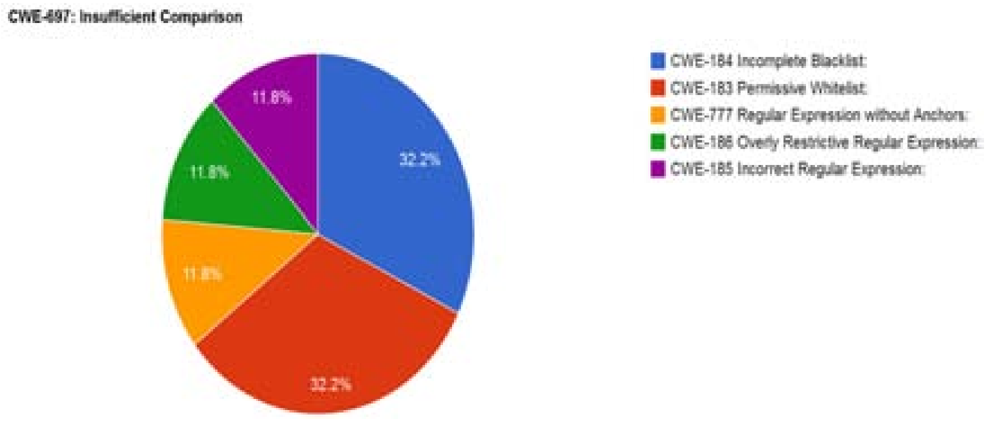


Figure 5.21: Distribution average of the CWE-697 security alerts for gLite WMS

5.4.9 CWE-703 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-703 are summarized in Figure 5.22.

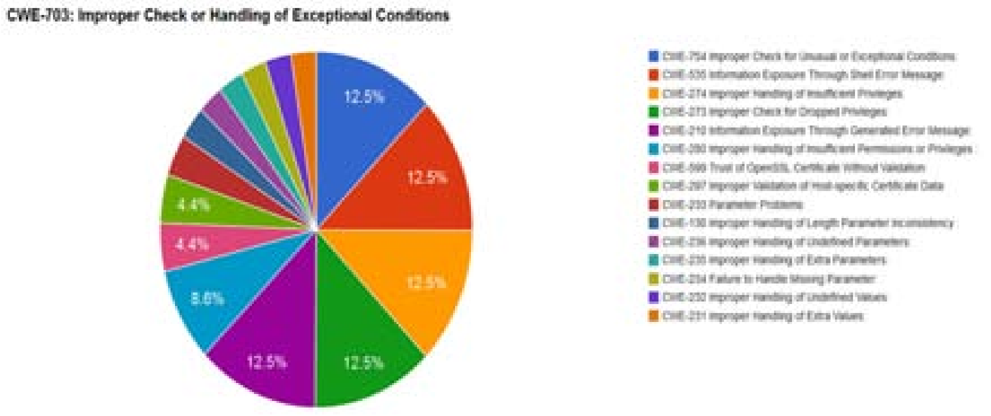


Figure 5.22: Distribution average of the CWE-703 security alerts for gLite WMS

5.4.10 CWE-707 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-707 are summarized in Figure 5.23.

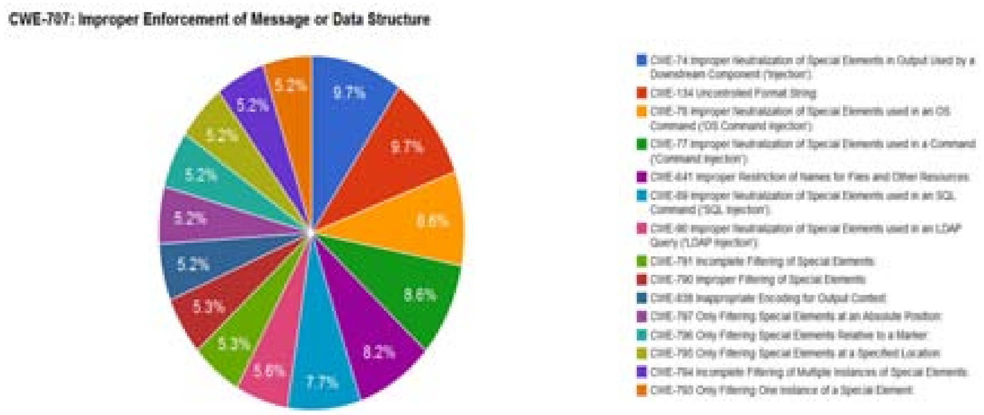


Figure 5.23: Distribution average of the CWE-707 security alerts for gLite WMS

5.4.11 CWE-710 Security Alerts

From the analysis of the attack vectors, the most remarkable security alerts for the top-level entry CWE-710 are summarized in Figure 5.24.

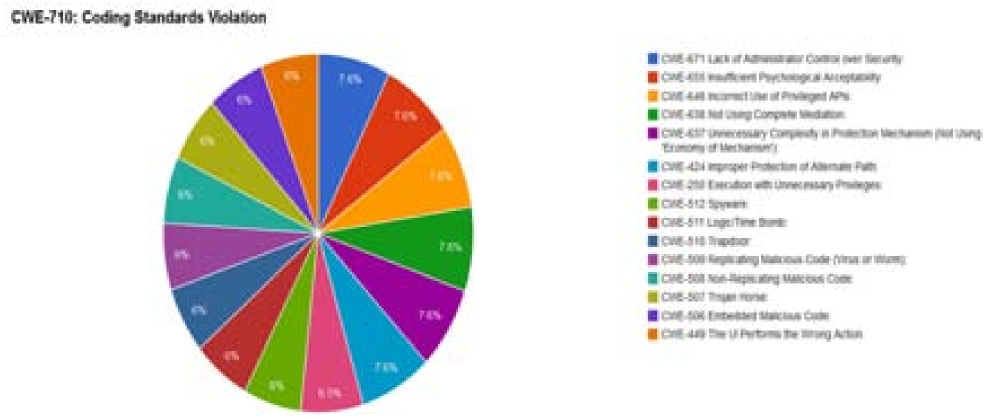


Figure 5.24: Distribution average of the CWE-710 security alerts for gLite WMS

5.5 Conclusions

In this Chapter we have presented an experimental evaluation of the *AvA4cmi* methodology. We have shown the security alerts derived by our prototype system.

In order to evaluate the benefits for the *AvA4cmi* methodology, we assessed the CrossBroker and the gLite WMS middlewares using the *AvA4cmi* prototype tool.

The security alerts derived by the assessment of CrossBroker hinted the weaknesses with highest scores, in the attack vectors where the vulnerabilities manually found with FPVA are strongly related. The most outstanding security alerts show that CrossBroker problems are more related to the pillars `Improper Control of a Resource Through its Lifetime , and `Protection Mechanism Failure , that is, for the corresponding attack vectors, the weaknesses with the highest scores can derive into the known vulnerabilities `CrossBroker-2009-0001 , `CrossBroker-2009-0002 , and `CrossBroker-2009-0003 . From the other pillars, it was possible to relate the weaknesses that would lead to the vulnerability `CrossBroker-2009-0004 in the corresponding attack vector.

We have also shown the security alerts for the assessment of gLite WMS, which present a close behaviour to CrossBroker, i.e., systems attributes and components are almost the same, and whereby, it is worth to state that gLite WMS problems share the same CrossBroker problems for the pillars `Improper Control of a Resource Through its Lifetime , and `Protection Mechanism Failure . The most outstanding security alerts were summarized for each pillar in a graphical representation, hinting which weaknesses for the attack vectors must be considered first by the security analyst during a code inspection.

CHAPTER 6

Conclusions and Future Work

In this work we have studied the problem of performing a vulnerability assessment of complex middleware interrelationships in distributed systems. In this Chapter we review the main conclusions and present the new lines of research opened by this work.

Security in distributed computing (e.g., Grid, Cloud, and SCADA networked systems) provides different protection mechanisms, ranging from the use of certificates issued by trusted certification authorities, to encrypted channels of communications. To date, current security mechanisms of distributed systems - authentication, authorization, certification, and delegation are not enough to assure that all application s resources are safe. Moreover, they are not in accordance with a systematic process for vulnerability assessment. In addition, novel threats have arisen, with the fast adaptation of cloud computing. This introduces the necessity of providing a guidance for comprehensive and more accurate vulnerability assessment, taking into account the complex middleware relationships.

We proposed a novel methodology that address the challenges for vulnerability assessment in distributed systems. It is called Attack Vector Analysis for Complex Middleware Interrelationships (*AvA4cmi*) . This methodology (*AvA4cmi*) defines a set of components to support and provide systematically guidance to where and why to deploy code assessment:

- The *Attack Vector Graphs* represent in a suitable, useful, and unified diagram all the initial FPVA analysis outcomes. An attack vector graph is aimed to depict the sequence of transformations that allows control flow to go from a point in the attack surface to a point in the impact surface. It allows the specification of the attack vectors conforming to the proposed methodology by extending the graphml language.
- The *Knowledge Base* is responsible for provide codified knowledge, which is used for building the set of rules that will guide the vulnerability assessment of a target system. The knowledge base is based on three elements: the most updated knowledge about weaknesses, the CWE community effort; a customized weakness scoring system based on the CWSS community effort; and the most common middleware System Attributes.
- The *AvA4cmi Algorithm* manages the stated knowledge base, when traversing the attack vector graphs, and assesses complex middleware interrelationships, in order to generate a list of hierarchized security alerts.

We also presented a synapsis between System Attributes and CWE weaknesses to allow the applicability of rules conforming to the proposed methodology by gathering the most useful information from the CWE taxonomy. The proposed methodology has been implemented in a prototype tool, developed using the programming language Python. It abstracts the low-level details and heterogeneity of middleware components from the security analyst responsibility.

We performed an experimental evaluation of our methodology. We assessed the middlewares CrossBroker, and gLite WMS. The assessment performed by our methodology demonstrates effectiveness to correlate vulnerabilities found manually using FPVA, with several security alerts (i.e., weaknesses) in the CrossBroker case, while in gLite WMS case, where FPVA did not find vulnerabilities, the remarkable security alerts hint where and why the security practitioner must consider to do a new code inspection.

Additionally, the *AvA4cmi* methodology demonstrates effectiveness for filling the gap between different steps of the FPVA methodology, provides significant guidance for security practitioners when performing code inspection, and positively impact the quality and accuracy of its vulnerability assessment.

This methodology (*AvA4cmi*) has produced several key accomplishments that distinguish it from formal related vulnerability assessment works:

- It has the important characteristic that it focuses on complex interrelationships among components, and not only on single components.
- The development of a well defined knowledge base of rules, which allows to match system attributes into multiple weaknesses, and quantifying attack vector weaknesses according to complex component interrelationships.
- A systematic guidance provided for the last FPVA analysis stage, automated by a software tool.
- The *AvA4cmi* results provide significant guidance to security practitioner. These results are not sensitive to source code analysis, which makes results language independent.

The main contributions of this work can be found in the following publications:

CEDI - Jornadas 2010 J. D. Serrano Latorre, E. Heymann, and E. Cesar, Manual vs automated vulnerability assessment on grid middleware , 2010.

CLCAR 2010 J. D. Serrano Latorre, E. Heymann, and E. Cesar, Developing new automatic vulnerability strategies for hpc systems in Latinamerican Conference on High Performance Computing - CLCAR, pp. 166-173, 2010.

IBERGRID 2011 J. D. Serrano Latorre, E. Heymann, E. Cesar, and B. Miller, Vulnerability assessment enhancement for middleware in 5th Iberian Grid Infrastructure Conference (IBERGRID), 2011.

COMPUTING AND INFORMATICS 2012 J. D. Serrano Latorre, E. Heymann, E. Cesar, and B. Miller, Vulnerability assessment enhancement for middleware for Computing and Informatics in Computing and Informatics Journal, ISSN: 1335-9150, Vol. 31, 2012, No. 1, pp. 103 118, 2012.

ISPEC 2013 J. D. Serrano Latorre, E. Heymann, E. Cesar, and B. Miller, Increasing Automated Vulnerability Assessment Accuracy on Cloud and Grid Middleware , 9th International Conference on Information Security Practice and Experience (ISPEC), LNCS, pp 278-294, 2013.

6.1 Future Work

Although the proposed methodology and the AvA4cmi implementation has been tested in well known middlewares, and offers a fully operational guidance for vulnerability assessment in distributed systems, open lines of research remain to be explored.

- Our Attack Vector graph implementation is static, where the system attributes of the middleware components used are the ones configured by the installation package. While there could be enough information of the system attributes, this information may differ during a real time execution. Creating attack vector graphs dynamically would avoid differences in the system attributes at the moment of assessing the middleware components. It will involve considering Self-Propelled Instrumentation [47], an execution monitoring technique that dynamically injects a fragment of code, the agent, into an application process on demand. The agent inserts instrumentation ahead of the control flow within the process and propagates into other processes, following communication events, crossing host boundaries, and collecting a distributed function-level trace of the execution.

- The Rules of the knowledge base are manually built, based on the experience gathered from several vulnerability assessment with FPVA, and the security practitioners experience. The update or creation of new rules can be cumbersome. The fine adjustment of the rules could be accomplished in an automatic way. It will involve considering the use of machine learning techniques for improvements of the whole knowledge base architecture, where rules could change in function of new acquired data at middleware runtime executions.

Bibliography

[1] C.J. Alberts and A.J. Dorofee. *Managing Information Security Risks: The Octave Approach*. SEI series. ADDISON WESLEY Publishing Company Incorporated, 2002.

[2] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing, version 3. https://cloudsecurityalliance.org/research/security-guidance/#_v3, November 2011.

[3] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *SOFTWARE: PRACTICE AND EXPERIENCE*, 32:1437 1466, 2002.

[4] Jean-Philippe B. Baud, James Caey, Sophie Lemaitre, Caitriana Nicholson, David Smith, and Graeme Stewart. Lcg data management: From edg to egee. http://ppewww.ph.gla.ac.uk/preprints/2005/06/, 2005.

[5] A. Behl. Emerging security challenges in cloud computing: An insight to cloud security challenges and their mitigation. In *Information and Communication Technologies (WICT), 2011 World Congress on*, pages 217 222, 2011.

[6] United States. President s Critical Infrastructure Protection Board and United States. Dept. of Energy. 21 steps to improve cyber security of scada networks, 2002.

[7] Randy Butler, Von Welch, Douglas Engert, Ian Foster, Steven Tuecke, John Volmer, and Carl Kesselman. A national-scale authentication infrastructure. *Computer*, 33(12):60 66, 2000.

[8] Marco Cecchi, Capannini Fabio, Alvise Dorigo, Antonia Ghiselli, Francesco Giacomini, Alessandro Maraschini, Moreno Marzolla, Salvatore Monforte, Fabrizio Pacini, Luca Petronzio, and Francesco Prelz. The glite workload management system. In *GPC*, volume 5529 of *Lecture Notes in Computer Science*, pages 256 268. Springer, 2009.

[9] San Diego Supercomputer Center. Storage Resource Broker. http://www.sdsc.edu/srb/, March 2013.

[10] The Common Weakness Enumeration. Cwe-20: Improper input validation. http://cwe.mitre.org/data/definitions/20, June 2013.

[11] Hewlett Packard Development Company. Fortify Source Code Analyzer. http://www8.hp.com/us/en/software-solutions/software.html?compURI=1338812, July 2013.

[12] Inc. Coverity. Coverity Prevent. http://www.coverity.com, July 2013.

[13] Ministerio de Ciencia e Innovación (MICINN), CSIC, and IFCA. Iniciativa grid nacional espanola (es-ngi). http://www.es-ngi.es/, 2013.

[14] T. Dierks and E. Rescorla. The transport layer security (tls) protocol. In *IETF RFC 4346*, 2006.

[15] e sciencecity.org. Grid cafe. http://www.gridcafe.org/, June 2013.

[16] Giorgio Emidio and EGEE-II. Middleware overview. http://agenda.ct.infn.it/materialDisplay.py?contribId=2&sessionId=0&materialId=slides&confId=48, April 2008.

[17] J. Falco, J. Gilsinn, and K. Stouffer. It security for industrial control systems: Requirements specification and performance testing. *NDIA Homeland Security Symposium & Exhibition*, 2004.

[18] E. Fernandez del Castillo. *Scheduling for Interactive and Parallel Applications on Grid*. PhD thesis, Universitat Autònoma de Barcelona, 2008.

[19] European Organization for Nuclear Research. Cern openlab welcomes siemens as latest partner. http://cerncourier.com/cws/article/cnl/36787, 2013.

[20] European Organization for Nuclear Research. Rackspace and cern openlab collaborate to deliver big bang with hybrid cloud. http://openlab.web.cern.ch/resources/press_release/rackspace-and-cern-openlab-collaborate-deliver-‘‘big-bang’’-hybrid-cloud, 2013.

[21] Institute for Security and Open Methodologies. Institute for security and open methodologies. http://http://www.isecom.org/, 2013.

[22] Ian Foster. What is the grid? a three point checklist. June 2002.

[23] Ian Foster, Carl Kessekan, Gene Tsudik, and Steven Tueckel. A Security Architecture for Computational Grids. *Proceedings of ACM Conference Computer and Communications Security*, pages 83 92, 1998.

[24] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115 128, 1996.

[25] Ian Foster and Carl Kesselman. The grid 2: Blueprint for a new computing infrastructure. 2003.

[26] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM conference on Computer and communications security*, CCS 98, pages 83 92, New York, NY, USA, 1998. ACM.

[27] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *CoRR*, cs.AR/0103025, 2001.

[28] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The ssl protocol version 3.0. Internet Draft, Transport Layer Security Working Group, November 1996.

[29] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, 5:7 9, 2001.

[30] Grid Policy Research Group (GGF). Grid policy. https://forge.gridforum.org/projects/policy-rg/, May 2013.

[31] George Gilder. The information factories. http://www.wired.com/wired/archive/14.10/cloudware.html, October 2006.

[32] graphdrawing.org. The GraphML File Format. http://graphml.graphdrawing.org, June 2013.

[33] Kevin Hemalen, Murat Kantarcioglu, Latifur Khan, and Bhavani Thuraisingham. Security issues for cloud computing. *International Journal of Information Security and Privacy.*, June 2010.

[34] Tony Hey and Anne E. Trefethen. The uk e-science core programme and the grid. *Future Generation Computer Systems*, 18:1017 1031, 2002.

[35] M. Humphrey and M.R. Thompson. Security implications of typical grid computing usage scenarios. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 95 103, 2001.

[36] Marty Humphrey and Mary R. Thompson. Security implications of typical grid computing usage scenarios. *Cluster Computing*, 5(3):257 264, July 2002.

[37] International Data Corporation (IDC). Idc cloud research. http://www.idc.com/prodserv/idc_cloud.jsp, March 2013.

[38] ISO/IEC. ISO/IEC 27005 Information technology - Security Techniques - Information security risk management. Technical report, jun 2008.

[39] Nagaraju Kilari and Rajagopal Sridaran. A survey on security threats for cloud computing. *International Journal of Engineering Research & Technology (IJERT)*, September 2012.

[40] B Kónya, C Aiftimiei, M Cecchi, L Field, P Fuhrmann, J K Nilsen, and J White. Consolidation and development roadmap of the emi middleware. *Journal of Physics: Conference Series*, 396(3):032062, 2012.

[41] J. Kupsch and B. Miller. Manual vs. automated vulnerability assessment: A case study. *International Workshop on Managing Insider Security Threats*, 469:83 97, June 2009.

[42] J. Kupsch, B. Miller, E. Heymann, and E. Cesar. First principles vulnerability assessment, mist project. Technical report, UAB & UW, September 2009.

[43] Jairo David Serrano Latorre. Ava4cmi git repository. https://github.com/ava4cmi/PythonApplication1.git, 2013.

[44] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance.* O Reilly Media, Inc., 2009.

[45] G. McGraw, K. Tsipenyuk, and B. Chess. Seven pernicious kingdoms: A taxonomy of software security errors. *IEEE Security and Privacy*, 3:81 84, 2005.

[46] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, June 2009.

[47] Alexander V. Mirgorodskiy and Barton P. Miller. Diagnosing distributed systems with self-propelled instrumentation. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware 08, pages 82 103, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[48] European Network and Information Security Agency (ENISA). Cloud computing: Benefits, risks and recommendations for information security. http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment/at_download/fullReport, November 2009.

[49] National Grid Initiatives (NGIs) and European International Research Organisations (EIROs). European grid infrastructure. http://www.egi.eu/, February 2013.

[50] National Institute of Standards & Technology. System protection profile industrial control systems, 2004.

[51] University of Wisconsin and Universitat Autonoma de Barcelona. The middleware security and testing group. http://www.cs.wisc.edu/mist, 2013.

[52] University of Wisconsin-Madison. Condor project. http://www.cs.wisc.edu/condor, March 2013.

[53] United States. General Accounting Office. Technology assessment: Cybersecurity for critical infrastructure protection, 2004.

[54] Cloud Standards Organization. Cloud standards organization. http://www.cloud-standards.org/, February 2013.

[55] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Warthwein, Ian Foster, Rob Gardner, Mike Wilde, Alan Blatecky, John McGee, and Rob Quick. The open science grid. *Journal of Physics: Conference Series*, 78(1):012057, 2007.

[56] European Union (EU) Information Society Technologies (IST) Programme. Crossgrid EU Project. https://www.cesga.es/es/investigacion/proyectos/Proyecto?id=39/, February 2009.

[57] European Union (EU) Information Society Technologies (IST) Programme. Interactive European Grid Project. https://www.cesga.es/es/investigacion/proyectos/Proyecto?id=108, February 2009.

[58] The Open Web Application Security Project. A1 injection. https://www.owasp.org/index.php/Top_10_2013-A1-Injection, July 2013.

[59] The Open Web Application Security Project. A10 unvalidated redirects and forwards. https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards, July 2013.

[60] The Open Web Application Security Project. A2 broken authentication and session management. https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management, July 2013.

[61] The Open Web Application Security Project. A3 cross-site scripting (xss). https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS), July 2013.

[62] The Open Web Application Security Project. A4 insecure direct object references. https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References, July 2013.

[63] The Open Web Application Security Project. A5 security misconfiguration. https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration, July 2013.

[64] The Open Web Application Security Project. A6 sensitive data exposure. https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure, July 2013.

[65] The Open Web Application Security Project. A7 missing function level access control. https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control, July 2013.

[66] The Open Web Application Security Project. A8 cross-site request forgery (csrf). https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF), July 2013.

[67] The Open Web Application Security Project. A9 using components with known vulnerabilities. https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities, July 2013.

[68] The Open Web Application Security Project. The open web application security project. https://www.owasp.org/, 2013.

[69] The Middleware Security and Testing Group. Secure coding practices for middleware. http://research.cs.wisc.edu/mist/presentations/CondorWeek-2012-Secure-Program-Tutorial.pdf, February 2012.

[70] Massimo Sgaravatto, P. Andreetto, S. Borgia, A. Dorigo, A. Gianelle, M. Mordacchini, L. Zangrando, S. Andreozzi, V. Ciaschini, C. Di Giusto, F. Giacomini, V. Medici, E. Ronchieri, V. Venturi, G. Avellino, S. Beco, A. Maraschini, F. Pacini, A. Guarise, G. Patania, D. Kouřil, A. Křenek, L. Matyska, M. Mulač, J. Pospíšil, Ruda, J. Sitera, J. škrabal, M. Voců, V. Martelli, M. Mezzadri, F. Prelz, D. Rebatto, S. Monforte, and M. Pappalardo. Practical approaches to grid workload and resource management in the egee project. *Proceedings of the International Computing in High Energy and Nuclear Physics*, pages 899 902, 2004.

[71] R. Shirey. RFC 2828 - Internet Security Glossary. may 2000.

[72] Gary Stoneburner, Alice Y. Goguen, and Alexis Feringa. Sp 800-30. risk management guide for information technology systems. Technical report, Gaithersburg, MD, United States, 2002.

[73] Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to Industrial Control Systems (ICS) Security Recommendations of the National Institute of Standards and Technology. Technical report, 2008.

[74] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft professional. Microsoft Press, 2004.

[75] The MITRE Community. Cwe research view xml file. http://cwe.mitre.org/data/xml/views/1000.xml.zip, March 2013.

[76] The MITRE Corporation. The Common Vulnerability and Exposure. http://cve.mitre.org/, 2013.

[77] The MITRE Corporation. The Common Weakness Enumeration. http://cwe.mitre.org/, 2013.

[78] The MITRE Corporation. The Common Weakness Scoring System. http://cwe.mitre.org/cwss/index.html, 2013.

[79] The MITRE Corporation. Cwe-1000: Research concepts. http://cwe.mitre.org/data/definitions/1000.html, June 2013.

[80] The MITRE Corporation. Cwe-699: Development concepts. http://cwe.mitre.org/data/definitions/699.html, June 2013.

[81] The MITRE Corporation. Cwe introductory brochure. http://makingsecuritymeasurable.mitre.org/docs/cwe-intro-handout.pdf, June 2013.

[82] International Telecommunication Union. Itu-t x.509 : Information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks, 1997.

[83] John Viega. Building security requirements with clasp. In *Proceedings of the 2005 workshop on Software engineering for secure systems–building trustworthy applications*, SESS 05, pages 1 7, New York, NY, USA, 2005. ACM.

[84] Wikipedia. Cloud computing. http://en.wikipedia.org/wiki/Cloud_computing, June 2013.