Increasing Automated Vulnerability Assessment Accuracy on Cloud and Grid Middleware *

Jairo Serrano[1], Eduardo Cesar[1], Elisa Heymman[1], and Barton Miller[2]

[1] Computer Architecture & Operating Systems
Universitat Autònoma de Barcelona
Barcelona, Spain
{jairodavid.serrano,eduardo.cesar,elisa.heymann}@uab.es
[2] Computer Sciences Department
University of Wisconsin-Madison
Madison, WI, USA
bart@cs.wisc.edu

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 (water, power, electricity, oil, gas, etc.) distributed systems, i.e. SCADA 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 bases their security on mechanisms such as authentication, authorization, and delegation. While these mechanisms have been studied in depth and can control 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 approach, called Attack Vector Analyzer (AvA), which is able to automatically hint which middleware components should be assessed and why. AvA 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. AvA'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 middleware, and corroborate which middleware components should be assessed and why.

Keywords: Vulnerability Assessment, Security, Weakness, Attack Vector, Cloud, Grid, Middleware

* This research has been supported by the MEC-MICINN Spain under contract TIN2007-64974 and by Department of Homeland Security grant FA8750-10-2-0030

1 Introduction

Vulnerability assessment is a security task that is insufficiently addressed in most existing Grid and Cloud projects, and even in "Supervisory Control and Data Acquisition (SCADA)" [1] systems it is an afterthought. Such projects use middleware software which usually bases its security on mechanisms such as authentication, authorization, and delegation. These mechanisms have been studied in depth and carry out control of key resources, but they are not enough to assure that all middleware resources, nor that the project's data running on them are safe. However, middleware systems usually do not undergo a thorough vulnerability assessment during their life cycle or after deployment, whereby security flaws may be overlooked. One possible solution would be to use existing automated tools such as Coverity Prevent [2] or HP Fortify SCA [3] to parse source code for previously known threats, but even the best of these tools find only a small percentage of the serious critical and complex vulnerabilities [4].

Nowadays security is one of the most desirable features of the computational Grid, Cloud, and SCADA systems because of the increasing number of threats and cybercriminal groups on the Internet, and more recently on industrial systems, that could lead to millionary costs not only repairing damages but also on investigating and unraveling the theft of personal and classified information. Therefore, a thorough vulnerability assessment requires a systematic approach that focuses on the key resources to be protected and allows for a detailed analysis of those parts of the code related to those resources and their trust relationships. Consistently, *First Principles Vulnerability Assessment (FPVA)* [5] answers these requirements. FPVA have been successfully applied to several large and widely-used middleware systems, such as Condor [6], a high-throughput scheduling system; Storage Resource Broker (SRB) [7], a data grid management system; Crossbroker [8], a Grid resource management for interactive and parallel applications, among others [9].

FPVA starts with an architectural analysis. This step identifies key structural components in a middleware system, including modules, threads, processes, and hosts. It then goes for a resources analysis, which identifies the key resources accessed by each component, and the operations supported on those resources. Privilege analysis is the next step, it identifies the trust assumptions about each component, answering such questions as how are they protected or who can access them. A complex but crucial part of trust and privilege analysis is evaluating trust delegation. By combining the information from the first two steps, we determine what operations a component will execute on behalf of another. The results of these steps are documented in diagrams that provide a roadmap for the last stage of the analysis, the manual middleware source code inspection. This top-down, architecture-driven analysis, can also help to identify more complex vulnerabilities that are based on the interaction of multiple system components and are not amenable to local code analysis.

For all the FPVA-analyzed systems we have noticed 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 last stage of the analysis (e.g. depending on the language used the analyst should look for different kind of vulnerabilities) and be creative 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. We have realized that security practitioner knowledge is similar to the one recorded on several available vulnerability classifications, such as The Common Weakness Enumeration (CWE) [10], The Seven Pernicious Kingdoms (McGraw-Fortify) [11], The OWASP Top Ten [12], and The Microsoft SDL [13], and that it can be codified in the form of rules, metrics, and scores to be applied automatically. The proposed *Attack Vector Analyzer (AvA)* presented in this paper implements an automated hinting of Grid and Cloud middleware vulnerabilities based on codified expert knowledge. AvA's results include a prioritized alert list of likely weaknesses, which can lead to exploitable vulnerabilities. Results are based on a detailed joint analysis of complex relationship between middleware components, which could be potentially attractive targets for the attackers. Furthermore, results provide particular guidance to the security practitioner to avoid overlooking security flaws, and false positives. This paper discusses how to address the FPVA "gap" without looking at the middleware source code, or parsing it, and the automatic tool AvA for systematically indicating which middleware components should be assessed and why, before the analyst shift to the source code inspection.

The remainder of this paper is structured as follows. Section 2 introduces the Attack Vector Analyzer approach and its components. Section 3 discusses a case study: AvA applied to CrossBroker, and its results compared to the ones obtained by a manual assessment of the same middleware. The related work is introduced in Section 4. Finally, conclusions and future work are shown in Section 5.

2 The Attack Vector Analyzer

Before proceeding with the description of the AvA approach it is needed to stand out relevant aspects from the FPVA methodology that helped us to derive needful concepts. One of these aspects is the FPVA aim of concentrating the analyst's attention on the (components and resources) assets of the middleware system that are most likely to have critical vulnerabilities. Thus, on each stage FPVA remains focusing in analyzing the data and control flows among the high-value system assets looking for insecure features and/or attributes. Other aspect is the artifacts produced on initial steps of FPVA: the architectural analysis step produces a document that diagrams the structure of the system and the interactions amongst the different components and with the end users. In this diagram the Attack Surface of the system can be defined.

The **Attack Surface** is the set of coordinates from which an attack (interaction user $\rightarrow$ system) might start, indeed it tells security practitioners where to start looking for the attacker's initial behavior. Then, the resource analysis step 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 step, we define other useful concept: **Impact Surface** as the set of coordinates where exploits or vulnerabilities might be possible. Finally, the artifact 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 diagram the Attack Vectors can be identified. An **Attack Vector** is the sequence of transformations that allows control flow to go from a point in the attack surface to a point in the impact surface.

Despite all the information gathered on the FPVA initial artifacts, finding actual vulnerabilities in the selected system during the component analysis depends on the implementation details of each component and the analyst's knowledge expertise. We have already claimed that this knowledge can be found in several existing vulnerability classifications [2,13,23,28], and that, in consequence, can be systematically codified in order to be able to automatically indicate which attack vectors of a given system should be analyzed and why. We have developed a complete methodology following this approach, and we have also implemented this methodology in a prototype tool called Attack Vector Analyzer (AvA) for demostrating how the FPVA gap can be sistematically reduced.

In the following subsections we describe this automatic approach, which is depicted in figure 1. It shows the components of the AvA architecture, composed

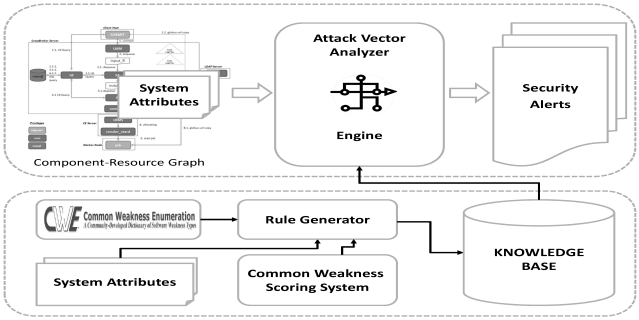

Fig. 1. Attack Vector Analyzer Architecture

by the (attack vector) analyzer engine, who receives as inputs: a single modified version of the FPVA diagrams called the component-resource graph, and a knowledge base (KB) with codified rules. The KB is based on three elements: the most current knowledge about vulnerabilities the CWE, the CWSS scoring system , and the system attributes. The outcome of the analyzer engine will be security alerts concerning to use this inputs knowledge on a particular system.

2.1 Building the Knowledge Base

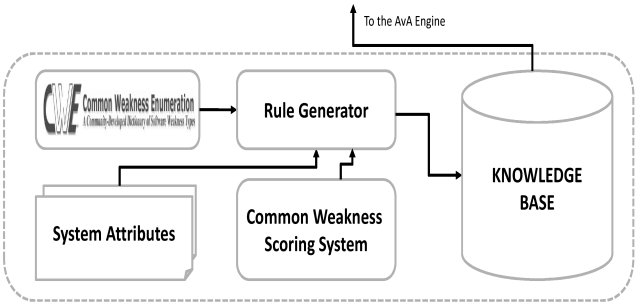

Fig. 2. AvA's Knowledge Base

In this section we describe the process followed for defining the propositions included in the AvA knowledge base (KB) from the information provided by the

Common Weakness Enumeration [10] system, the Common Weakness Scoring [14] system, and the middleware system attributes extracted from the experience using FPVA. All this information depicted in figure 2 has been used for building the set of rules that will guide the analysis of a target system.

Common weakness enumeration: CWE is a community initiative of security practitioners, including individual researchers and representatives from industry, academia, and government, interested in actively reducing and managing common software weaknesses that can occur in software's architecture, design, code or implementation, and that can lead to exploitable security vulnerabilities. CWE could be roughly described as a three tiered approach, 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. The Attack Vector Analyzer is based on the CWE Research View approach.

Common weakness scoring system: CWSS is a part of the CWE project, it provides a scoring mechanism for weaknesses in a flexible, open, and consistent manner. CWSS works scoring CWE's with 18 different factors in three metric groups: (1) the Base Finding group, which captures the inherent risk of the weakness, confidence in the accuracy of the finding, and strength of controls; (2) the Attack Surface group, which captures the barriers that an attacker must cross in order to exploit the weakness; and (3) the Environmental group, which includes factors that may be specific to a particular operational context, such as business impact, likelihood of exploit, and existence of external controls.

Name	Description
Owner	The owner's component
User	The user's component
User-Admin Interface	Is the component part of the user or admin interface
Sanitize	Determines if the component performs data sanitizing operations
Transform Data	Determines if the component performs data transforming operations
Transfer Data	Determines if the component performs data transfering operations
Trust	Determines if the component performs trustworthy operations
Server Interaction	Determines if the component performs a DB, LDAP, etc., server operations
Timeout	Determines if the component performs timeout operations
Max/Min	Determines if the component performs data restriction operations
Third-party	Determines if the component performs local/remote third-party operations
Spoofing	Determines if the component performs operations against spoofing
Tampering	Determines if the component performs operations against tampering
Encryption	Determines if the component performs encryption operations
Attachment	Determines if the component performs attachment operations
Error Handling	Determines if the component performs operations against unexpected error
Client-Server	Determines if the component is installed on client or server host
Web	Determines if the component is a web service or application
Log-Backup	Determines if the component performs Log and/or Backup operations

Table 1. AvA's System Attributes

System attributes: The system attributes have been defined in a large research and refining process, and it has been reflected in our initial works [15,

16]. System attributes are based on the information provided by several FPVA artifacts, developer team interviews, and user, admin, and API documentation. The attributes included in AvA, shown in table 1, have been derived from the different middleware systems assessed and characterized using FPVA, such as Condor, SRB, MyProxy [17], gLExec [18], VOMS-admin [19], and CrossBroker.

Rule generation: We use CWE, CWSS, and system attributes information for defining assessment rules (propositions). First, we gathered the most useful information from the CWE source, which is represented in the AvA analyzer by twelve different elements with relevant details acquired through a public XML file provided by the CWE community. These elements are:

1. The weakness identificator
2. The weakness name
3. The weakness description
4. The weakness extended description
5. The programming language in which the weakness may occur
6. The consequence scope, which identifies an individual consequence that may be associated to the weakness
7. The consequence technical impact, which describes the technical impacts that can arise if an attacker attempts to exploit the weakness
8. The consequence notes, which provides additional commentary about its consequence
9. The mitigation description, which contains a single method for mitigating the weakness
10. The mapped node names, which identifies the name of the entry to which this weakness is being mapped in other taxonomies or classifications
11. The relationship, which contains a note regarding the relationships between CWE entries.
12. The observed example description, which presents an unambiguous correlation between the example being described and the weakness which it is meant to exemplify.

Secondly, a comprehensive (word, synonyms) search over the twelve mentioned elements have been conducted on the 682 CWE weaknesses to find the set of weaknesses related to each system attribute. For example, for the system attribute "Owner", 150 different related weaknesses were found, among these we have the "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". Similarly, for each of the remaining system attributes we found between 20 to 150 relationships.

Once the system attributes have been related to the weaknesses, we use a customized version of CWSS for producing the final set of rules. It consists essentially in defining logical propositions for each value of each system attribute, to obtain a quantitative measurement of how the attribute contributes to each of its related weaknesses. More precisely, we verify it through the three metrics groups of the customized CWSS scoring system, linking up the system attributes with its relevant factors, and scoring accordingly to them. The relevant factors for the customized CWSS metrics groups are shown in table 2. Thus, each factor has a list of possible values, and its corresponding score, e.g., the technical impact (TI) factor is shown in the table 3. Finally, we illustrate the kind of propositions in the KB with the next examples.

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 2. CWSS Metric groups

Technical Impact	Critical	High	Medium	Low	None	Default	Unknown	Not Applicable
Score	1.0	0.9	0.6	0.3	0	0.6	0.5	0.3

Table 3. TI Scoring (CWSS adaptation)

– 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-Admin Interface" == "Yes" then:
"TI" == "High", "AV"=="Local", "IN"=="Automated"
"DI"=="High", "EX"=="High", "AS"=="Moderate"

It can be appreciated in both examples that the "Owner" and "User-Admin Interface" attributes are related with particular CWSS factors such as the TI factor, which are influenced by their corresponding values. In the first example, the TI factor answers with the "Critical" value when the "Owner" attribute corresponds with the "Administrator" value. On the other hand, 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. Then, with the knowledge base of rules stated, we proceed to describe the analysis process of the attack vectors of a middleware system.

2.2 Analyzing Attack Vectors

This section describes how the AvA engine analysis works on the attack vectors, the inputs required, and the security alerts produced. Below, we present the component-resource graph, which is the input of the AvA engine, and then we proceed with the AvA engine, as we can see depicted in figure 3.

Component-Resource graph: with the stated objective of reducing the gap between the outcomes from the initial FPVA stages, and the manual FPVA component code analysis, we have defined a structure called *Component-Resource graph* [20] for representing the results of these initial stages in a more suitable and readable format, which also includes relevant information about middleware components such as the system attributes. A component-resource

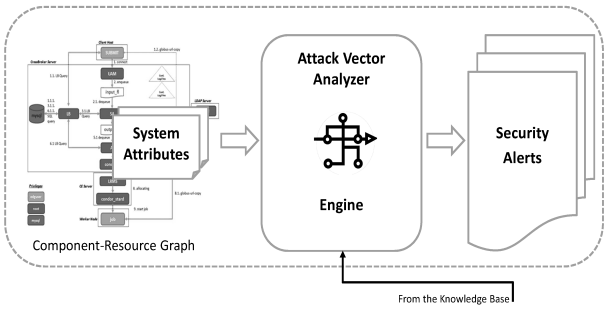

Fig. 3. AvA Engine

graph is aimed to depict all attack vectors between middleware components, given that most of the generated FPVA diagrams describe particular operations of the middleware, such as submitting a job in a workload management system. Thereby, the order in which an attack vector is built is also quite clear because every edge in the 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.

To represent a component-resource graph we have chosen the GraphML format [21]. Basically, a component-resource graph is an XML file composed by the information gathered from the FPVA diagrams. Once the component-resource graph is correctly depicted in the graphml format, we proceed to apply the analysis process on it.

The AvA engine analysis: algorithm 1 shows the AvA engine analysis process. It begins reading the component-resource graph, this step allows the AvA engine to identify and to load the attack vectors of the middleware been assessed. Then, the AvA knowledge base is read, in order to load both the rules and the weaknesses - system attributes relationships. The next step of the engine is to start traversing each attack vector, component by component. For each component, its system attributes are fetched, and then assessed accordingly to the KB-rules, taking into account our customized CWSS system. Hence, the weaknesses related to the system attributes are assessed too. As a result of this step, a mark is obtained for each attribute associated to every weakness in every component of all identified attack vectors in the component-resource graph. For example, for a weakness CWE-X with associated attributes X_0, X_1, and X_2 (such as: owner, user, tampering, etc.) and an attack vector composed by components C_0, C_1, C_2, the results shown in table 4 can be obtained. After this

CWE-X			
Component \ Attribute	X_0	X_1	X_2
C_0	SC_{00}	SC_{01}	SC_{02}
C_1	SC_{10}	SC_{11}	SC_{12}
C_2	SC_{20}	SC_{21}	SC_{22}

Table 4. Individual marks

first assessment, with the objective of obtaining a single score for each weakness associated to the attack vector, the algorithm applies the following steps:

(1) assign to each attribute the minimum score obtained by any component of the attack vector (line 12). Using the same example of table 4, the score of attribute X_0 for the weakness CWE-X will be $min(SC_{00},SC_{10},SC_{20})$. We have choosen the minimun value because it means that in that component at least, a weakness mitigation has been implemented; (2) once the minimum scores are computed then proceed to weigh them accordingly to the CWE research view (line 13) because the system attributes might have a different impact 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; at last, (3) the maximum weighed score for the weakness is computed, bearing in mind how was the score of its child weaknesses (line 14). It means that child weaknesses provide more information to the top-level weaknesses. Once all the weaknesses are processed, the last step is to sort them into the eleven pillars of the CWE research view accordingly to their maximum weighed score. Finally,

Algorithm 1

```
1.  Read the Component-Resource graph
2.      Load the Attack Vectors
3.  Read the Knowledge Base
4.      Load the Rules
5.      Load the Weaknesses
6.  For each Attack Vector
7.      For each Component
8.          Fetch the system attributes
9.          Parse the system attributes with KB-rules
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 max weighed score based on CWE
15.     For each Pillar at the CWE research view
16.         Sort weaknesses in order of max weighed score
17.         Write the sorted security alert lists
```

the security alerts for the assessed attack vector are delivered as a hierarchical list of weighed weaknesses for each CWE pillar. Thus, we are systematically providing comprehensive information to the security practitioner, pointing out not only which vulnerabilities should be analyzed, but also why we should pay attention to them in the assessed attack vector.

3 Case Study

This case study shows the benefits of AvA approach by using it on a grid middleware system and then verifying the results against the previous FPVA assessment on the same middleware. CrossBroker is a Grid resource management system for interactive and parallel appplications used in various european projects, including Crossgrid [22] the Interactive European Grid [23], and it is being used by the Spanish Grid Initiative. CrossBroker was built by extending the functionality provided in LCG [24] and gLite WMS [25].

3.1 FPVA applied to CrossBroker

The manual vulnerability assessment following the FPVA guidelines on CrossBroker identified serious and complex vulnerabilities affecting high value

assets. A completed and detailed information about them can be found on previous work [15], which is out of scope for this paper. Below, we introduce a summarized description of the CrossBroker FPVA found vulnerabilities:

Vulnerability 1: 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. ***Cause:*** *Code injection, Improper data validation, Incorrect authorization.* ***Component:*** *submit, network server.*

Vulnerability 2: Certain types of user's job submitted to CrossBroker are not protected from manipulation from other user's jobs. ***Cause:*** *Incorrect privileges, Missing authentication, Multiple unique privilege domains.* ***Component:*** *scheduling agent.*

Vulnerability 3: Remote resources 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. ***Cause:*** *Missing authorization, Misconfiguration.* ***Component:*** *scheduling agent.*

Vulnerability 4: 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. ***Cause:*** *Improper error handling, Inability to handle missing/invalid field or value.* ***Component:*** *submit, logging and bookkeeping, mysql.*

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 artifacts, without further guidance or information. The goal of the following subsection is to demonstrate that the AvA concepts can be used to increase the effectiveness of the FPVA component analysis, with a more comprehensive guidance to automatically indicate where and why the components should be assessed.

3.2 AvA applied to CrossBroker

The validation consists of applying the AvA assessment process to the CrossBroker component-resource graph (figure 4), and look for those weaknesses in the security alerts that are related with the vulnerabilities found manually, and those weaknesses that can depict likely new vulnerabilities that were not found initially. CrossBroker has several attack vectors that can be observed from its graph, and for the sake of simplicity we just introduce two different attack vectors with completely different impact surfaces. It is worth to state that the number of attack vectors depends on the number of attack and impact surfaces components we have identified using a graph editor tool.

Attack vector I: The attack vector I is composed by ten components starting with the *submit component* which is the attack surface component in this case, passing through the *network server, input queue, scheduling agent, output queue, application launcher, condor daemon I, local resource manager, condor daemon II*, until achieve its impact surface the *job component.* To visualize the results of asssessing the attack vector I, the AvA security alerts were clustered in accordance to every hierarchical pillar of the CWE research

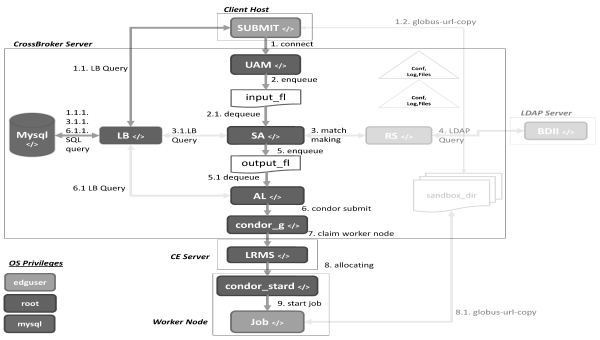

Fig. 4. CrossBroker component-resource graph with attack vectors I-II highlighted

view, and regarding to the max score of the weaknesses and the total number of them belonging to every pillar, as we can see in figure 5. Therefore, for the "CWE-664" pillar composed of 150 weaknesses, considering the characterization of the attack vector components, and the whole assessment process, we found that around 20 % of weaknesses in the whole pillar can derive into one of three vulnerabilities found with FPVA, which are distributed as follows: 11 weaknesses related to "Vulnerability 3", 17 weaknesses related to "Vulnerability 2", and one weakness related to "Vulnerability 1". In this attack vector there are no weaknesses matches for "Vulnerability 4", due that its underlying cause belong to other middleware components. Itemizing, from the 15 first

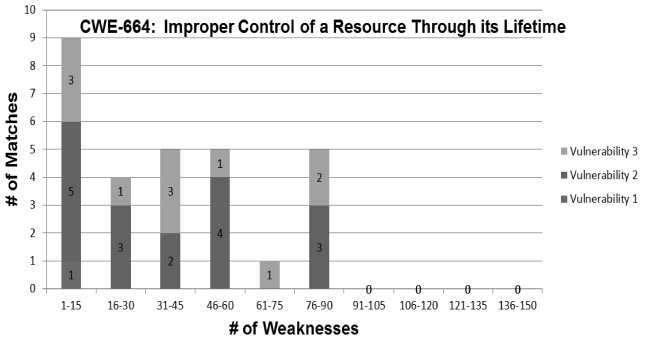

Fig. 5. Security Alerts for attack vector I clustered by CWE-664 pillar.

weaknesses with the highest scores, nine are related to three of the vulnerabilities found manually, as for example 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 summary along with the CWE-664 pillar description *"The software does not maintain or incorrectly maintains control over a resource throughout its lifetime of creation, use, and release"*. Thus, it can be seen the straight relationship between the weakness-pillar and the "Vulnerability 3" because although the AvA analysis has taken into account those system attributes related to authentication and authorization underlying

mechanisms for the attack surface, the attacker is able to handle at will the impact surface.

The AvA analisys corroborated this relationship, due that there is no more control found for the same system attributes on any other attack vector component after a job is submitted in CrossBroker. For some of the remaining CWE pillars their security alerts also hinted related weaknesses to the same vulnerabilities, which gives the attack vector I results consistency, and some others security alerts hinted a likely new vulnerability related to certificate issues.

Attack vector II: Since the attack vector II has only three components starting with the *submit component* which is again the attack surface component, the *logging and bookkeeping component*, and its impact surface the *mysql component*, then we put together all its AvA security alerts to visualize the results from a global perspective regarding the whole CWE research view, as we can see in figure 6, instead of an individual pillar visualization as in the attack vector I. By inspecting the security alerts for all the pillars, the assessment of the attack vector II with the AvA analyzer hinted 19 weaknesses, all of them related with the "Vulnerability 4" found with FPVA, which are distributed as follows: 8 weaknesses in "CWE-703 Improper Check or Handling of Exceptional Conditions" , 4 weaknesses in "CWE-664 Improper Control of a Resource Through its Lifetime", 3 weaknesses in "CWE-691 Insufficient Control Flow Management" , 1 weakness in "CWE-710 Coding Standards Violation" , 1 weakness in "CWE-682 Incorrect Calculation" , 1 weakness in "CWE-118 Improper Access of Indexable Resource", 1 weakness in "CWE-435 Interaction Error", and 0 weaknesses for the rest of pillars. Just like in attack vector I for "Vulnerability 4", it happens that there are no weaknesses matching for "Vulnerabilities 1, 2, and 3", due that their underlying causes belong to other middleware components. If we look inside the pillars with hinted weaknesses,

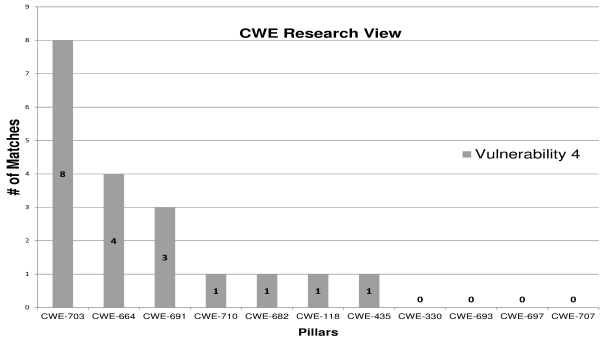

Fig. 6. Security Alerts for attack vector II clustered by entire CWE research view.

we found that the sum of all the issues, such as improper check for unusual or exceptional conditions, unexpected status code or return value, return of wrong status code, missing report of error condition, uncontrolled resource comsuption, improper resource shutdown or release, insufficient control of network message volume, uncontrolled recursion, improper validation, or incorrect control flow scoping, among others are indeed strongly related to the "Vulnerability 4" causes. In summary, to focus the security alerts clearly, we visualized from either individually or globally point of view the hinted weaknesses by the AvA analyzer in both of the attack vectors illustrated, which are strongly related with the vulnerabilities found previously with FPVA.

4 Related Work

Vulnerability Assessment of middleware systems is a field that has attracted the interest of both research and commercial communities, due to the rapid growth of the use of distributed and high performance computing, as well as the increasingly rapid growth of threats. Accordingly, in this section we introduce those vulnerability assessment projects that are most related to the AvA approach, such as the Microsoft Threat Modeling [13], the Open Vulnerability and Assessment Language [26] project, and the vulnerability cause graphs [27].

4.1 Microsoft Threat Modeling

The methodology that has the most in common with the AvA approach is *Microsoft Threat Modeling*. It is aimed at identifying and rating the most likely threats affecting applications, based on understanding their architecture and implementation during the entire development life cycle. While Microsoft's methodology is the closest to AvA approach, there is a key difference: after developing the architectural overview of the application, the Microsoft methodology 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 on individual components may be detected, and the vulnerabilities detected may not refer to high value assets. With AvA, the component evaluation is performed only on the critical parts of the system, and we may be able to hint vulnerabilities based on a list of weaknesses, particularly those resulting from the interaction of complex relationships between attack vector components. In addition, Microsoft 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.

4.2 The Open Vulnerabilities and Assessment Language (OVAL)

OVAL is an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the spectrum of security tools and services. OVAL includes a language used to encode system details, and an assortment of content repositories held throughout the community. The repositories are collections of publicly available and open content that utilize the language. In short it means that OVAL is an open language to express checks for determining whether software vulnerabilities and configuration issues, programs, and patches exist on a system. It is based primarily on known vulnerabilities identified in Common Vulnerabilities and Exposures (CVE) [28], a dictionary of standardized names and descriptions for publicly known information security vulnerabilities and exposures developed by the MITRE Community and stakeholders. In contrast to OVAL, our effort is neither based on the specific CVE dictionary nor alleged vulnerabilities according to machine states, instead we claim that AvA approach works with CWE and CWSS systems, and with nonspecific software vulnerabilities, also AvA approach is based on FPVA stages, thereby AvA gathers more meaningful information for the assessment process.

4.3 Vulnerability Cause Graphs

It is based on a thorough analysis of vulnerabilities and their causes, similar to root cause analysis. The results are represented as a graph, which Byers et al. [27] called vulnerability cause graph. Vulnerability cause graphs provide the

basis for improving software development best practices in a structured manner. The structure of the vulnerability cause graph and the analysis of each individual cause are used to determine which activities need to be present in the software development process in order to prevent specific vulnerabilities. In a vulnerability cause graph, vertices with no successors are known as vulnerabilities, and represent classes of potential vulnerabilities in software being developed. Vertices with successors are known as causes, and represent conditions or events that may lead to vulnerabilities. In our case, the most noticeable difference is that we want to know whether a vulnerability may exist and why, instead Byers' work knows the vulnerabilities and looks for their causes.

5 Conclusions

In this paper we described a novel approach for hinting Grid and Cloud middleware vulnerabilities. The proposed methodology was implemented in a prototype tool, called the Attack Vector Analyzer (AvA). The absence of a formal method that attempts to systematically use the information gathered by the First Principles Vulnerability Assessment (FPVA) methodology, and the knowledge found on the Common Weakness Enumeration (CWE), motivated us to develop the AvA prototype, which validates the AvA approach. AvA demonstrates effectiveness for filling the gap between different steps of the FPVA methodology; and provides significant guidance for security practitioners to determine which middleware components should be assessed and why. We corroborated AvA's efectiveness with the assessment of the Grid middleware CrossBroker. In order to get comprehensive and accurate results, the CrossBroker's security alerts produced by AvA were analyzed from two perspectives, the first one, considering them through each CWE pillar, and the second one, considering them through the whole CWE research view. Thus, it was possible to correlate previous vulnerabilities found manually with several attack vector weaknesses, and automatically identify the most likely vulnerabilities.

AvA will positively impact security practitioners empirical research during the source code inspection, and consequently its quality and accuracy of vulnerability assessment. Our methodology approach has produced several key accomplishments that distinguish it from formal related vulnerability assessment works. A list of our accomplishments include:

- Our assessment methodology has the important characteristic that it focuses on complex interrelationships among component, and not only on single components.
- The development of a well defined knowledge base based on 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 AvA's results provide significant guidance to security practitioner. These results are not sensitive to source code analysis, which makes results language independent.

Future work around AvA approach 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.

References

1. T. Sommestad, G. Ericsson, and J. Nordlander, "Scada system cyber security - a comparison of standards," in *Power and Energy Society General Meeting IEEE*, pp. 1 –8, july 2010.
2. "Coverity Prevent. http://www.coverity.com."
3. "Fortify Source Code Analyzer. http://www.fortify.com."
4. J. Kupsch and B. Miller, "Manual vs. automated vulnerability assessment: A case study," *International Workshop on Managing Insider Security Threats*, vol. 469, pp. 83–97, June 2009.
5. J. Kupsch, B. Miller, E. Heymann, and E. Cesar, "First principles vulnerability assessment, mist project," tech. rep., UAB & UW, September 2009.
6. "Condor Project. http://www.cs.wisc.edu/condor."
7. "Storage Resource Broker. http://www.sdsc.edu/srb/."
8. E. Fernandez del Castillo, *Scheduling for Interactive and Parallel Applications on Grid*. PhD thesis, Universitat Autònoma de Barcelona, 2008.
9. "MIST Group: Middleware security and testing web site. http://www.cs.wisc.edu/mist."
10. "The Common Weakness Enumeration. http://cwe.mitre.org/."
11. G. McGraw, K. Tsipenyuk, and B. Chess, "Seven pernicious kingdoms: A taxonomy of software security errors," *IEEE Security and Privacy*, vol. 3, pp. 81–84, 2005.
12. "The open web application security project (owasp). https://www.owasp.org/."
13. F. Swiderski and W. Snyder., *Threat Modeling*. Microsoft Press, 2004.
14. "The Common Weakness Scoring System. http://cwe.mitre.org/cwss/."
15. J. D. Serrano Latorre, E. Heymann, and E. Cesar, "Manual vs automated vulnerability assessment on grid middleware," in *III Congreso Espanol de Informatica - CEDI 2010.*, Sep 2010.
16. 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, August 2010.
17. "MyProxy. http://grid.ncsa.illinois.edu/myproxy."
18. "gLExec - Gluing grid computing jobs to the Unix world. https://www.nikhef.nl/."
19. "The virtual organization membership service (voms) - http://edg-wp2.web.cern.ch/edg-wp2/security/voms/voms.html."
20. J. D. Serrano Latorre, E. Heymann, E. Cesar, and B. Miller, "Vulnerability assessment enhancement for middleware," in *5th Iberian Grid Infrastructure Conference (IBERGRID)*, June 2011.
21. "The GraphML File Format. http://graphml.graphdrawing.org/."
22. "Crossgrid EU Project. http://www.eu-crossgrid.org/."
23. "Interactive European Grid Project. http://grid.ifca.es/inteugrid_ifca.htm."
24. J.-P. B. Baud, J. Caey, S. Lemaitre, C. Nicholson, D. Smith, and G. Stewart, "Lcg data management: From edg to egee," 2005.
25. P. Andreetto and et alter, "Practical approaches to grid workload and resource management in the egee project.," *Proceedings of the International Computing in High Energy and Nuclear Physics*, pp. 899–902, 2004.
26. "OVAL - Open Vulnerability and Assessment Language. http://oval.mitre.org/."
27. D. Byers, S. Ardi, N. Shahmehri, and C. Duma, "Modeling software vulnerabilities with vulnerability cause graphs," in *Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on*, pp. 411 –422, 2006.
28. "The Common Vulnerability and Exposures. http://cve.mitre.org/."