Universitat Autònoma de Barcelona

Master Thesis

Vulnerability Assessment of gLite's CREAM Grid Computing Middleware Using the FPVA Methodology

Author:
Maxime Frydman

Supervisor:
Dr. Elisa Heymann

A thesis submitted in fulfilment of the requirements
for the degree of Master of High Performance Computing and Security

in the

Computer Architecture & Operating Systems Department (CAOS)

July 2013

Declaration of Authorship

I, Maxime FRYDMAN, declare that this thesis titled, 'Vulnerability Assessment of gLite's CREAM Grid Computing Middleware Using the FPVA Methodology ' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a Master degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"It's easy to cry "bug" when the truth is that you've got a complex system and sometimes it takes a while to get all the components to co-exist peacefully."

Doug Vargas

"When you catch bugs early, you also get fewer compound bugs. Compound bugs are two separate bugs that interact: you trip going downstairs, and when you reach for the handrail it comes off in your hand."

Paul Graham

UAB

Abstract

Computer Architecture & Operating Systems Department (CAOS)

Master of High Performance Computing and Security

Vulnerability Assessment of gLite's CREAM Grid Computing Middleware Using the FPVA Methodology

by Maxime Frydman

The aim of this thesis is to assess the security of the CREAM component which belongs to the gLite Grid middleware. Security is an ongoing problem with large software projects like gLite and traditional vulnerability assessment techniques have proven to be inadequate due to the complex nature of such software which limit the use of automated assessment tools while making manual assessment cost and time prohibitive.

To alleviate these issues the assessment was performed using the First Principles Vulnerability Assessment methodology. This analyst-centric methodology focuses on guiding a human analyst in his assessment by identifying critical assets most likely to lead to vulnerabilities in order of their potential damage.

The first part of this work focuses on how this methodology was applied through the production of a series of diagrams and the information the analyst can extract from them to guide his assessment. The second part of this work shows how this information was used to search for vulnerabilities and presents the results of this search.

The results of this work present a number of severe vulnerabilities found within the CREAM component, the suggestions made to the developers to fix them and presents the response from the developers.

UAB

Abstract

Computer Architecture & Operating Systems Department (CAOS)

Master of High Performance Computing and Security

Vulnerability Assessment of gLite's CREAM Grid Computing Middleware Using the FPVA Methodology

by Maxime Frydman

El objetivo de esta tesis es evaluar la seguridad del componente CREAM que pertenece a gLite Grid middleware.

La seguridad es un problema constante en grandes proyectos de software como gLite, y las técnicas tradicionales de evaluación de vulnerabilidad han demostrado ser inadecuadas debido a la naturaleza compleja de este software, que limita el uso de herramientas de evaluación automatizadas, aumentando así su coste, debido requerir una evaluación manual e implicar un tiempo prohibitivo.

Para disminuir estos problemas, la evaluación ha sido realizada usando la metodología de los Primeros Principios de Evaluación de Vulnerabilidad. Esta metodología centrada en el análisis se centra en guiar a la persona analista durante el proceso de evaluación, identificando los aspectos críticos con más probabilidades de provocar vulnerabilidades, ordenándolos según la cantidad de daños que implicarian.

La primera parte de este trabajo se centra en como se aplicó esta metodología, gracias a la utilización de una serie de diagramas y a la información que el analista puede extraer de estos para guiar su evaluación. La segunda parte muestra cómo fue usada esta información para buscar vulnerabilidades, y presenta los resultados de esta búsqueda.

En los resultados se presentan una serie de graves vulnerabilidades que se encuentran en el componente CREAM, las sugerencias realizadas a los desarrolladores para que sean corregidas, y presenta respuesta de los desarrolladores.

UAB

Abstract

Computer Architecture & Operating Systems Department (CAOS)

Master of High Performance Computing and Security

Vulnerability Assessment of gLite's CREAM Grid Computing Middleware Using the FPVA Methodology

by Maxime Frydman

L'objectiu d'aquesta tesi és avaluar la seguretat del component CREAM que pertany a gLite Grid middleware.

La seguretat és un problema constant en els grans projectes de sofware com gLite, i les tècniques tradicionals d'avaluació de vulnerabilitat han demostrat ser inadequades a causa de la naturalesa complexa d'aquest software, que limita l'ús d'eines d'avaluació automatitzades, augmentant així el seu cost, pel fet de requerir una avaluació manual i implicar un temps prohibitiu.

Per disminuir aquests problemes, l'avaluació s'ha realitzat usant la metodologia Primers Principis d'Avaluació de Vulnerabilitat. Aquesta metodologia centrada en l'anàlisi es centra en guiar a la persona durant el procés d'avaluació, identificant els aspectes crtics amb més probabilitats de provocar vulnerabilitats, ordenant-los segons la quantitat de danys que implicarien.

La primera part d'aquest treball es centra en com es va aplicar aquesta metodologia, grácies a la utilització d'una sèrie de diagrames i a la informació que l'analista pot extreure d'aquests per guiar la seva avaluació. La segona part mostra com va ser usada aquesta informació per buscar vulnerabilitats, i presenta els resultats d'aquesta recerca.

En els resultats es presenten una sèrie de greus vulnerabilitats que es troben en el component CREAM, els suggeriments realitzats als desenvolupadors perquè siguin corregits, i presenta la resposta dels desenvolupadors.

Acknowledgements

Foremost I would like to thank my advisor Prof. Elisa Heymann for her advice and mentoring during this project, her deep knowledge and insight into the field of security have improved my knowledge of the field and prepared me for future work.

Beside my advisor I would like to thank my colleagues Dr. Manuel Brugnoli, Prof. Barton Miller, James Kupsch and Joe Carrion who advised and supported me throughout this work.

I would also like to thank the friends with whom I shared the Master, particularly Gemma Sanjuan, Guillem Nicolau, César Acevedo, Roger Ten and Viola Saveta for their companionship that turned this Master into an amazing experience.

Not forgotten are my friends back in Belgium, Denis Coupez, Pierre Marot, Julie Neven and Hatoul Mitjons for all their help in getting me where I am.

Last but not least I would like to thank my family who has always believed in me and provided me with love and support beyond measure, without them none of this would have been possible.

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
BLAH	**B**atch **L**anguage **A**SCII **H**elper
CE	**C**omputing **E**lement
CREAM	**C**omputing **R**esource **E**xecution **a**nd **M**anagement
DoS	**D**enial **o**f **S**ervice
EGEE	**E**nabling **G**rids for **E**-scienc**E**
EMI	**E**uropean **M**iddleware **I**nitiative
FPVA	**F**irst **P**rinciples **V**ulnerability **A**ssessment
IS	**I**nformation **S**ervices
JM	**J**ournal **M**anager
MIST	**Mi**ddleware **S**ecurity and **T**esting
MitM	**M**an **i**n **t**he **M**iddle
MTM	**M**icrosoft **T**hreat **M**odeling
LRMS	**L**ocal **R**esource **M**anager
POC	**P**roof **O**f **C**oncept
SE	**S**torage **E**lement
OWASP	The **O**pen **W**eb **A**pplication **S**ecurity **P**roject
VO	**V**irtual **O**rganization
VOMS	**V**irtual **O**rganisation **M**anagement **S**ystem
WMS	**W**orker **M**anagement **S**ystem
WN	**W**orker **N**ode

Dedicated to my parents for their unconditional encouragement, love and support.

Chapter 1

Introduction

This chapter introduces the work that was undertaken during the Master. It starts with a general introduction to the field of software security, followed by the motivations to perform vulnerability assessment and its objective, and concludes with an outline of this thesis.

1.1 Overview

Over the last few decades, the world has rapidly evolved through informatisation. Technology has invaded almost every aspect of society to the point where most daily actions involve the combination of dozens of complex systems. Digital infrastructures have become the backbone of economy, government, business and our daily communications.

This rapid change has brought countless innovations and ameliorations but also comes with new challenges and concerns. In a world where virtually every system is interconnected and holds valuable information, security has become a key concern of businesses [4] and government alike [5].

To alleviate these issues the industry has developed a number of standards and guidelines to aid in the production of secure software and introduced security assessment to identify vulnerabilities during later phases of production or after release. While no software can be proven secure, these techniques aim to reduce the possibility of attack by malicious agents and mitigate their consequences.

A vast majority of attacks on software use simple techniques exploiting recurring issues in software that are both known and well understood [6]. Security assessment takes advantage of that fact by using automated tools as part of their assessment process to

identify these vulnerabilities. However these techniques have proven to be ineffective when assessing large and complex software while non automated techniques show to be cost prohibitive due to the scale of such software which makes the analysis time consuming.

Grid computing middleware are typically extremely large and complex software projects. They serves a variety of purposes both in the scientific community and private sector to optimally use internal resources to perform large calculations and resource intensive computations. They are used by organization like CERN [7] in their LHC computing grid to support LHC experiments or NASA [8] for the NAS (NASA Advanced Supercomputing facility) but are becoming increasingly used by businesses to provide for rising needs in computation.

Security is a very important aspect of such software; grid-computing systems are used by multiple users or organizations simultaneously, have a very high installation and upkeep cost, contain sensitive data and execute long running applications that can suffer greatly from disruption. Combining a large amount of disparate users interacting on the same system with the need for a reliable and secure environment is a complex challenge to address.

With new applications like cloud computing, grid adoption is likely to gain even more momentum however concerns over security are a factor limiting expansion [9]. Amongst others, security in the grid environment is becoming a relevant research topic necessary to provide for the needs of this growing industry.

1.2 Problem Statement

Grid computing systems have a growing need for security assessment but current techniques, presented in Chapter 2, are either ill adapted or prohibitively expensive to perform.

The software used to build a grid computing system is generally composed of a combination of general-purpose middleware components, interconnected to support a wide variety of environments and applications. These large interconnected units have proven difficult to audit using current automated vulnerability assessment tools.

Vulnerabilities that arise through the interconnection of multiple independent middleware components are much more complex in nature and automated vulnerability assessment tools are ill suited for such applications [10].

Complete unguided code review with an analyst centric approach however is unthinkable due to the scale of such projects, which would drive assessment costs to exorbitant rates.

New methodologies like First Principles Vulnerability Assessment (FPVA) [11] were created for the analysis of middleware components and provide a feasible manner to make an analyst-centric vulnerability assessment. The FPVA methodology gives the analyst the tools required to identify the most critical portions of the system in order to guide his review over critical portions of the code. These sections is where it is most likely that vulnerabilities will be encountered and where those vulnerabilities would cause the most damage.

The purpose of this work is to apply the FPVA methodology to perform the vulnerability assessment of the CREAM component, which is a part of gLite middleware.

1.3 Motivation and past works

There are numerous motivations for this line of research, foremost there is currently an increasing need for security in software independently of the type of software considered. In the field of security, middleware components are currently posing serious challenges to assessment work and this research aims to better the quality of such audits while reducing costs.

On a personal note, this work is the continuation of previous academic undertakings started during the bachelor degree concerning securing servers for e-commerce applications and mitigating risk from a hosting perspective.

Past work in the field of middleware vulnerability assessment is limited as this is a novel approach to solving this problem. This research group has in the past assessed a number of middleware components including some found within the gLite project (VOMS, ARGUS) and this is a unique opportunity to further this work.

FPVA shares most commonalities with the Microsoft threat modeling methodology [12] however there are key differences which will be further detailed in Section 2.4.

1.4 Objectives

The primary objective of this work is to perform the vulnerability assessment of the CREAM middleware component using the FPVA methodology (detailed in Section 2.3).

The objective of a security assessment is to identify security vulnerabilities and report them to the developers so that they may be fixed thus increasing the security of the software.

More specifically the steps involved in discovering vulnerability can be summarized as such:

- Identify a weakness within the software.
- Produce an exploit to prove that the vulnerability exists.
- Propose a fix to the developers so that the vulnerability can be addressed.
- Write a vulnerability report on the found vulnerability so that the developers can reproduce and fix it.

The motivation behind these steps will be further explained in Chapter 2 detailing the assessment methodology. The culmination of the assessment work will lead to the production of an unknown amount of reports, depending of the vulnerabilities present in the software, and a final report summarizing the work done in the assessment. A secondary objective is to identify possible refinements that can be brought to the methodology. Each new project audited using the FPVA methodology offers new challenges and insight on how to evaluate middleware components. The methodology is an ongoing project, which is being refined as more audits are performed. From the experience gained some possible ameliorations will be presented for review and if they are relevant be adopted into the methodology.

1.5 Bibliography Disclaimer

It is in general considered to be the best practice to only include within a bibliography material originating from peer reviewed scientific journals and reference books. This work follows this practice when possible however within the field of security it is standard practice for material to be published solely over the internet. This material is commonly used both when producing software to set security guidelines and when performing security assessment work. Also the field of security evolves extremely rapidly following software trends and practices, it was therefore considered to be in the best interest of this document to include the most up to date information as this is the information available and used by the security analyst.

As such selected articles and reports which have only been published over the internet were included in this work when all other resources were exhausted taking care to only

use those produced by organizations considered reliable by the security community and taking care to verify, to the best of our ability, the quality of that information.

1.6 Confidentiality and Ethical Considerations

During the writing of this thesis great care has been taken to respect the confidentiality agreements that bind this research group and the software producers and ethical considerations of vulnerability reporting as defined in Section 2.5. The work achieved over these past several months revealed a number of real vulnerabilities that could have serious implications if they were to be divulged. These vulnerabilities have however not yet been addressed nor publicly announced by the developers.

For this reason the information disclosed in Chapter 5 concerning the results of this work is presented in a heavily redacted format. Providing enough information and context to show the results that were obtained without revealing critical details that could damage gLite's user community. This decision was taken considering the public nature of this Master thesis that would make any other decision irresponsible.

1.7 Thesis Outline

This Master thesis is organized in the following way:

Chapter 1 offers an overview of the general context of this work, the problem that is being studied and clear objective, and deliverables.

Chapter 2 presents concepts of vulnerability assessment and details the FPVA methodology used during this work.

Chapter 3 gives an introduction to the Grid, gLite and the middleware component that was assessed, CREAM, detailing its main functionalities and mechanics.

Chapter 4 shows the results of the Architectural, Resource and Privilege Analysis and presents proposed refinements to the Resource diagrams.

Chapter 5 presents the component analysis by describing the vulnerabilities discovered and the response from the developers.

Chapter 6 offers the conclusions derived from this work and suggest open lines of research that can be explored in the future.

Chapter 2

Vulnerability Assessment and the FPVA Methodology

This chapter defines some key concepts of vulnerability assessment, introduces vulnerability assessment methodologies related to our work and then gives an overview of the FPVA methodology that was used during this project. It concludes with certain ethical considerations that have to be taken into account when performing vulnerability assessment.

2.1 Vulnerability assessment

Let's start by defining security assessment in software as an effort to detect the largest possible number of design and implementation flaws that would allow a malicious attacker to gain access to resources they are not allowed to use. Note the use of "the largest possible number " as it is rigorously impossible to formally prove that a software component (excluding trivial cases) is at any one point free of bugs [13] or vulnerabilities (the distinction between bugs and vulnerabilities is discussed in Section 2.2). Even if the former were possible it would not be a guarantee that it would stay secure in the light of new methods of attack.

Let us also define the scope of vulnerability assessment as the assessment of the software component and not the platform on which it runs. This is an important restriction on the scope of the work, without it vulnerability assessment projects would become unending enterprises.

In its current form, software assessment and code reviews for security purposes exists in a number of different forms of varying formality. It is to be expected that

methodologies[14], although sharing certain fundamental aspects differ depending on the context in which they are to be applied. The environment, in which the software runs, stakes involved and budgets allocated define this context.

There are two distinct categories of code-reviewing techniques:

Automated

Software based solutions which attempt to automate the discovery of vulnerabilities in software through either analysis of the source code or external means.

Analyst-Centric

Human based reviewing techniques where analysts will verify to a point the robustness of a component through careful analysis.

2.1.1 Automated Tools for Vulnerability Assessment

Automated tools propose attractive solutions for vulnerability discovery. In its purest form a code-reviewing tool should be able to point out all vulnerabilities present in the analyzed code while avoiding to report non-existent problems. The reporting of a perceived issue which is not in fact usable as a vector of attack is defined as a false positive, while omitting to report a vulnerability present in the system a false negative. The quality of such a tool can therefore be defined by the amount of false positive and false negatives produced in comparison to the number of actual real problems reported.

Producing a large amount of false positives is an issue that is costly, in man-hours, for the auditors, as they have to analyze portions of code in order to discard the reported vulnerability. However the production of a large amount of false negatives has an even greater impact, luring the auditors into a false sense of security.

Another issue with automated tools is that they are based on preexisting libraries of vulnerabilities and cannot detect new attacks previously unknown. These are the problematic aspects of current automated code reviewing tools and while they are of use in discovering implementation issues, their weakness lies in more complex interactions between systems when vulnerabilities emerge from trust boundaries or design flaws. This limits the usefulness of these tools for this research in particular, this has been shown in previous work by this research group [10].

Automated tools however are relatively low cost in comparison to the analyst-centric approach and as such remain useful in their specific domain.

2.1.2 Analyst-Centric Vulnerability Assessment

Analyst centric approaches rely on human assessment of software components, by reviewing code and probing for vulnerabilities. This approach relies heavily on the knowledge and creativity of the analyst that allows him to detect vulnerabilities.

This approach has the advantage of allowing the discovery of new methods of attack, as the analyst does not rely on previously known vulnerabilities. It does rely on the capacity of the analyst to spot vulnerabilities which is a time consuming effort, particularly for large code bases like middleware components, and this requires the analyst to have a good understanding of how the component works.

To alleviate these issues methodologies are used to guide the analyst's efforts by helping to identify critical sections where vulnerabilities are most likely to be encountered and have higher chances to be of grave consequences.

These methodologies serve to increase the understanding of the component, the resources of which it is comprised and the interactions between components. When done properly this increases productivity, reducing the work to a manageable size, while maximizing the potential for vulnerability identification. A few examples of such methodologies are the Microsoft threat modeling [12] and FPVA.

2.2 Vulnerabilities

For this work we will use the following definition of a vulnerability:

> "Vulnerabilities are specific flaws or oversights in a piece of software that allows attackers to do something malicious - expose or alter sensitive information, disrupt or destroy a system, or take control of a computer system or program." - *The art of software security assessment: Identifying and preventing software vulnerabilities*

This definition can lead to the impression that vulnerabilities and bugs are indistinguishable. It is true that vulnerabilities often originates from bugs but not all bugs lead to vulnerabilities.

When looking for vulnerabilities the security expectations of the software also has to be taken into account to define what is considered to be a vulnerability [15]. These expectations are set by the developers but can be classified into three components:

Availability

Availability is the expectation that the system be available to its users and resist disruptions. While service disruption can occur for a variety of reasons it is expected that it resists attacks that attempt to render it unavailable to a reasonable degree.

Confidentiality

Confidentiality is the expectation that certain information should not be divulged. This information can be the internal state and data of the system which should only be privy to administrators of the system but also the information of each user of the system which is not expected to be accessible by other parties unless specified by the user.

Integrity

Integrity is the expectation that the data within the system has not been altered outside of the normal functions of the system. Whether this information is internal to the system or belongs to users, it is expected of the data to correctly reflect the actions of the software and not be alterable outside these boundaries.

With this classification in mind the analyst can determine whether an issue he has discovered with the software can be classified as a vulnerability or a bug. The distinction is very important to the analyst. Finding vulnerabilities is the prime objective, mandates a vulnerability report that will be addressed by the developers and is considered a significant issue with the software until it has been fixed. Bugs however are not systematically reported, it is left either to the analyst's discretion or to pre-established convention with the software producer whether to report them and in which fashion.

2.2.1 Types of Vulnerabilities

The Common Vulnerability Exposure standard [16] is an effort to allow the communication between organization of encountered vulnerabilities. They offer a standard for naming and classifying vulnerabilities and maintain a dictionary of publicly known vulnerabilities and exposures.

The following is a non exhaustive list of the main categories of vulnerabilities, a more complete list can be consulted at [17].

Denial of Service

A denial of service (DoS) is an attack targeting the availability of a system. They are often performed by stressing a system to the point where it is incapable of delivering its service.

Privilege Escalation

Privilege escalation are vulnerabilities allowing a user to gain additional privileges, typically administrator privileges. When successful they usually lead to an attacker gaining control of the system and having access to its resources.

Overflows

Overflows occur in processes that do not have proper bounds set on input data that lead to the corruption of memory. A typical overflow attack is the buffer overflow which can force the execution of another process. This can be used to gain additional privileges on a system, crash application or expose new services.

SQL Injections

SQL injections allow the modification of SQL statements, allowing arbitrary queries to be executed. This can be used to bypass restrictions like authentication methods, perform DoS attacks and modify database entries.

Information Distribution

Information distribution vulnerabilities come from the exposure of information that should not be accessible. They can be used to extract confidential information from a system.

Restriction Bypass

Restriction bypass comes from the incorrect validation of security measures. Applications typically have a number of authentication pre-requisites before certain functionalities can be used. However if a system does not verify that the authentications has been performed before allowing access to its functionalities the restriction has been avoided allowing unrestricted access.

Man in the Middle attacks

Man in the Middle (MitM) is a category of vulnerabilities that is caused by the presence of an attacker along the network path between two communicating hosts. Depending on the type of network the attacker can have the opportunity to capture and possibly alter traffic. When capturing traffic the confidentiality of the communication is at risk and when it is altered the integrity is compromised. There exists many variants of MitM attacks that go from simple eavesdropping to complex host redirection and traffic manipulation.

2.2.2 Attack Surface

The attack surface in software are the areas that are exposed to users and where vulnerabilities can occur. An attacker only has access to a certain number of points of interaction with a software component, for example a server's API or submission forms on a web page. Other areas are considered to be off limit to the attacker, for example when assessing a server's security it is assumed that the attacker does not have administrator privileges on that system, all the information that could be gathered with such access are not considered to be part of a vulnerability.

Reducing the attack surface is considered to be a good security practice [18], although it does not prevent vulnerabilities or the damage which can be done if they are found, they do limit the point of entries. This makes it easier to assess the software by limiting the amount of code that has to be reviewed and which offer the possibility of attack.

When performing an assessment, one of the first steps is to identify attack surfaces as those are the points where vulnerabilities can be exploited. Different attack surfaces are vulnerable to different types of vulnerabilities and attack surfaces differ depending on the software and the technology used but usually fall into the following categories:

API

APIs are the interfaces that are offered by a component to allow interactions from other software. For example a server typically has an API that define how clients should interact with it. These can be susceptible to injections, DoS or erroneous data flow.

Communications

Communications are the channels of dialog between users and the software or between components, usually networks. These can be susceptible to MITM and DoS attacks.

Data Input

Software generally offers methods of input for data supplied by its users, for example web forms. These can be vulnerable to injection attacks.

Human Factor

Humans are often included as an attack surface to describe social engineering exploits where human can be tricked into performing malicious actions. These attacks can rely solely on human error like password divulgation but can also be combined with software vulnerabilities. For example XSS attacks, where the user

performs a normal task unaware that he is using a modified version of the software sent by the attacker.

2.2.3 Attack Vector

The attack vector is the complete path that is used by an attacker to exploit a vulnerability. While a software component might be riddled with vulnerabilities if there is no way to trigger them the vulnerability cannot be exploited. Finding a way to reach and trigger a vulnerability is the attack vector.

When identifying and reporting vulnerabilities it is important to understand the attack vector to understand all the components that are traversed and why this path is vulnerable, at the end of the attack vector lies the specific vulnerability.

A single vulnerability can sometimes be exploited through multiple attack vectors, this might give the impression that each vector is a separate vulnerability. For example multiple fields sent from a client to a server might lead to an injection attack on a vulnerable function, this single function is the core vulnerability and is what should be reported to the developers. If an attack vector is confused with a vulnerability it can lead to multiple reports that each describe a seemingly different issue when in fact it stems from a single vulnerability.

2.3 First Principles Vulnerability Assessment

The FPVA methodology [11], first published in 2010 is the result of research on the topic of vulnerability assessment for grid and cloud middleware by our research group It is an effort to provide a solution to the issues of securing such middleware components.

It has been used successfully to assess gLite's VOMS, Google Chrome, MyProxy, Condor and a number of other components, figure 2.1 shows some results. Through each of these projects, incremental refinements have been made as new challenges were encountered.

Project	Root acct	Admin acct	Other acct	DoS	Total
Condor	1	13	1		15
SRB		5			5
MyProxy		1	1	3	5
glExec	3	1	1		5
Gratia Condor Probe	2			1	3
CondorQuill				6	6
CrossBroker			2	1	3
Total	6	20	5	11	42

TABLE 2.1: Results obtained by applying FPVA, taken from the FPVA MIST Project Technical Report [1]

The methodology has been developed taking into account the size and complexity of middleware components while keeping an analyst centric approach. It focuses on guiding the analyst towards the critical sections of the components where vulnerabilities are most likely to be encountered and vulnerabilities would have the largest impact. It does so by separating the assessment into two distinct steps. The first step serves to increase the understanding of the system through the elaboration of diagrams representing the architecture, interactions and assets of the middleware. With this information at hand the auditor can guide his assessment in a much more efficient manner, limiting the scope of the code that has to be reviewed and ordered by matter of priority in terms of security risks.

The second step is the component analysis where the analyst reviews critical sections of the code, identified in the previous step, looking for vulnerabilities that can be abused. When such vulnerabilities are found they are reported to the developers through a vulnerability report. This report contains all the information required by the developers to understand, repeat and fix the issue. This includes: the components affected, severity of the attack, working POC code, suggestions on how to fix the issue and any other material the analyst deems necessary.

To guide his assessment the analyst uses the information gathered during the elaboration of the diagrams to identify attack surfaces. Once the the attack surfaces are clearly identified the analyst can perform a more thorough investigation on each one to identify the types of vulnerabilities that are possible. From there the analyst has a clear road map of what has to be reviewed and knows how to prioritize his work starting with what is likely to be the most damaging.

While these steps give the analyst a general idea of how the assessment will be performed, he does not know whether he will find any vulnerabilities and how long the review of each section will take. Depending on the size and complexity of the code the assessment of a single attack surface can be a significant task without any guarantees to find vulnerabilities. However the ordering of the analysis, with regard to risk, increases the chance that the analyst will find significant issues sooner rather than later.

Vulnerabilities, as seen in Section 2.2.1 and 2.2.2, are inherently conditioned by the type of resources used by the system, databases, configuration files, working folders etc., and the technology on which component is built, programming languages, communication methods, component interaction etc.

This work is further aided by the work of organizations like the OWASP, which is an open source application security project that produces guidelines and metrics on vulnerabilities. These projects are invaluable to the security analyst both to increase his knowledge of common vulnerabilities but also providing him with statistics of the most common vulnerabilities specific to each programming language.

2.3.1 Steps of the analysis and deliverables

The goal of the first three steps, respectively the architectural, resource identification and trust and privileges analysis, are to gain knowledge over the software component under assessment. The results of this comes under the form of a series of diagrams that give an analyst an overview of the components mechanics. They are quite different from typical software development diagrams, inner details of process are abstracted to reveal the flow of information. The focus is centered on information that is valuable to the vulnerability analyst. They emphasize inter component communications, resources permissions and other details that help elaborate attacks.

Each step of the analysis is aimed to aid the analyst to achieve his goal, we will now see how each of these steps is carried out and the information it offers to the analyst.

Architectural diagrams

The architectural focuses on giving the analyst an overview by separating a system into its components . This step focuses on identifying major structural components of the system, this includes executable files, the interactions between components and with users and communication methods. 2.1.

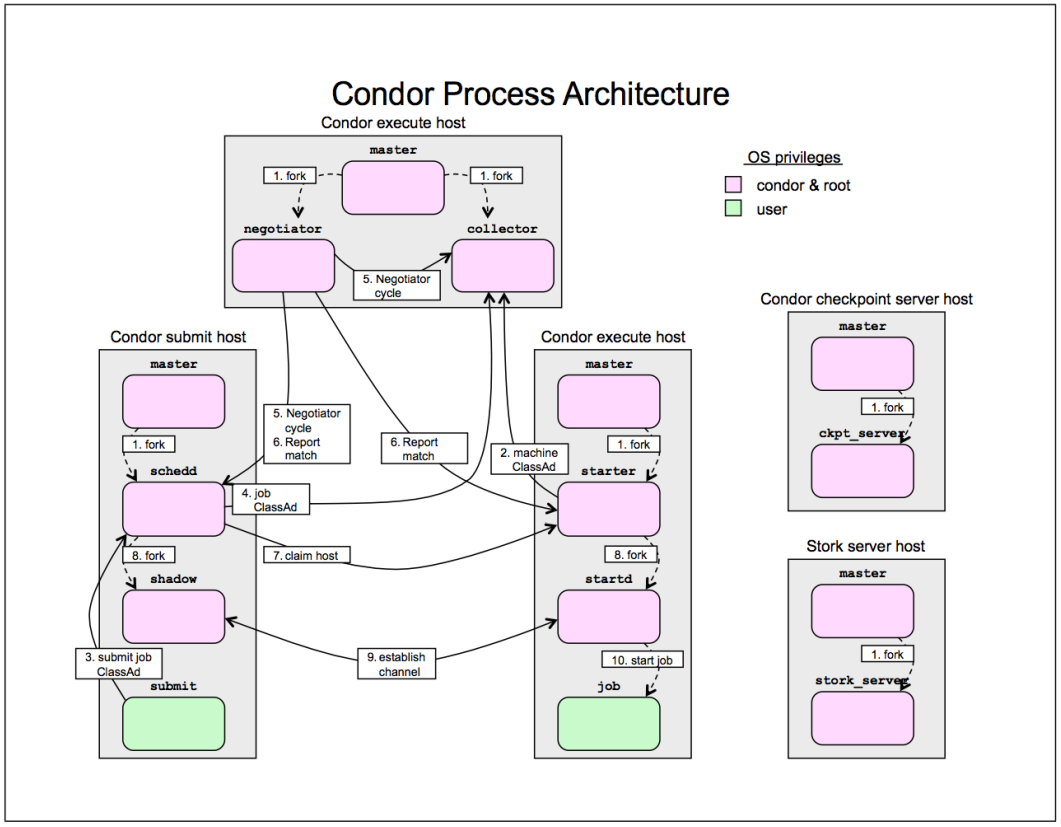


FIGURE 2.1: Condor Architecture diagram taken from the FPVA MIST Project Technical Report [1]

These diagrams are used to understand how a user's request travels through the system, showing all the components involved in a transaction. This is a fundamental step in understanding how everything fits together and identifying attack vectors.

Resource diagrams

The second part of the analysis focuses on resources. Typical software components access a great number of resources like configuration files, databases and logs 2.2. Resources play a fundamental role in security, they are one of the prime targets of the analysts, the resource diagram gives insight into which components have access to each resources and the type of access they have (consultation, modification or append).

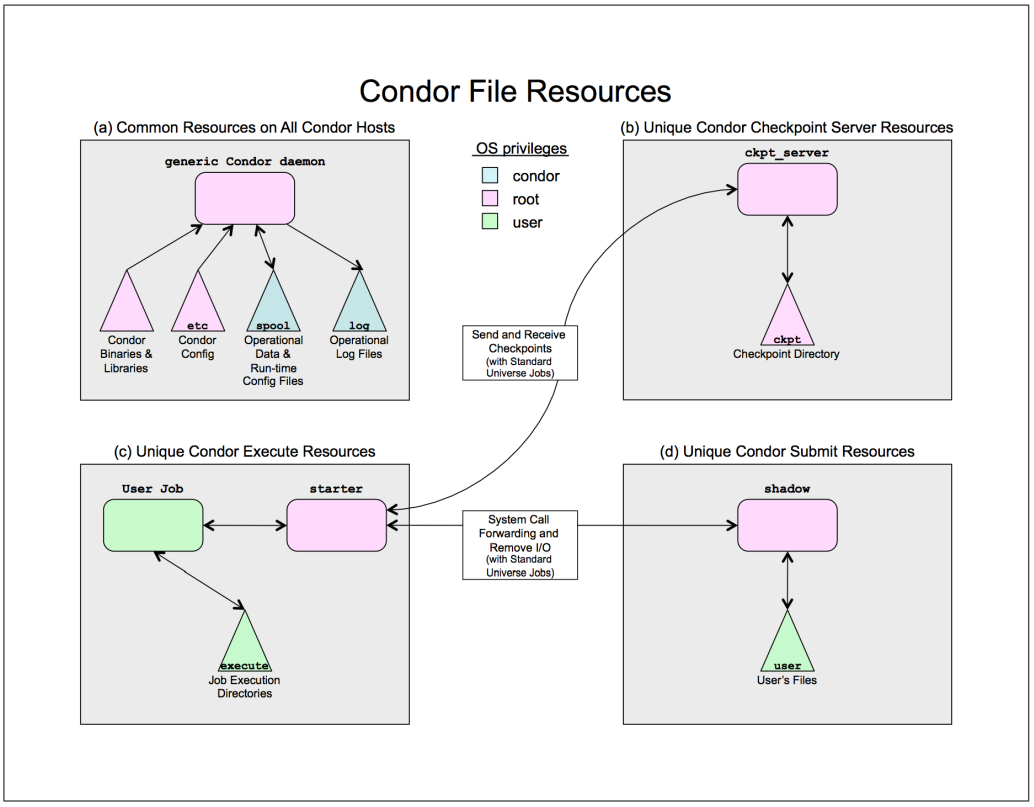


FIGURE 2.2: Condor Ressource diagram taken from the FPVA MIST Project Technical Report [1]

Combining the information gathered from the resource and the architectural diagram critical processes can be identified. The analyst can then focus on breaching the confidentiality or the integrity of these resources having identified the vector and goal of the attack.

Trust and Privileges

The last step in the production of the diagrams focuses on trust and privileges. Each system makes assumptions on the level of trust given to components, generally internal components that are protected have a higher level of trust for resources than external components like clients that are open to modification.

Each of the previous diagrams is labeled with execution privileges and resources with their permissions. With this step the diagrams are complete and the analyst has gained a great deal of information about how to perform his assessment. Components with high levels of trust and privileges and access to critical resources have much more potential for damage. The analyst now has the information over the vectors, the goals and the components having the means to perform his attack successfully.

With the first step of the analysis complete the analyst is now able to plan his assessment, from this point on the assessment will focus on the second step of the analysis, the component assessment. Most components will not be assessed entirely which would be unfeasible, however with the guidance of the previous analysis the reviewing process can be focused on the most high value targets. This step repeats itself until the end of the assessment and consists of the following:

Code review

The code is reviewed by the analyst paying special attention to how information flows through the component and how data is being handled. This step can be aided by automated tools to facilitate the discovery of particular types of issues, however for reasons mentioned previously 2.1.1 the main method of review remains analyst centric. When the analyst spots the possibility to derail that flow outside of its pre-established path and can lead to a vulnerability he then moves on to the second part of the process.

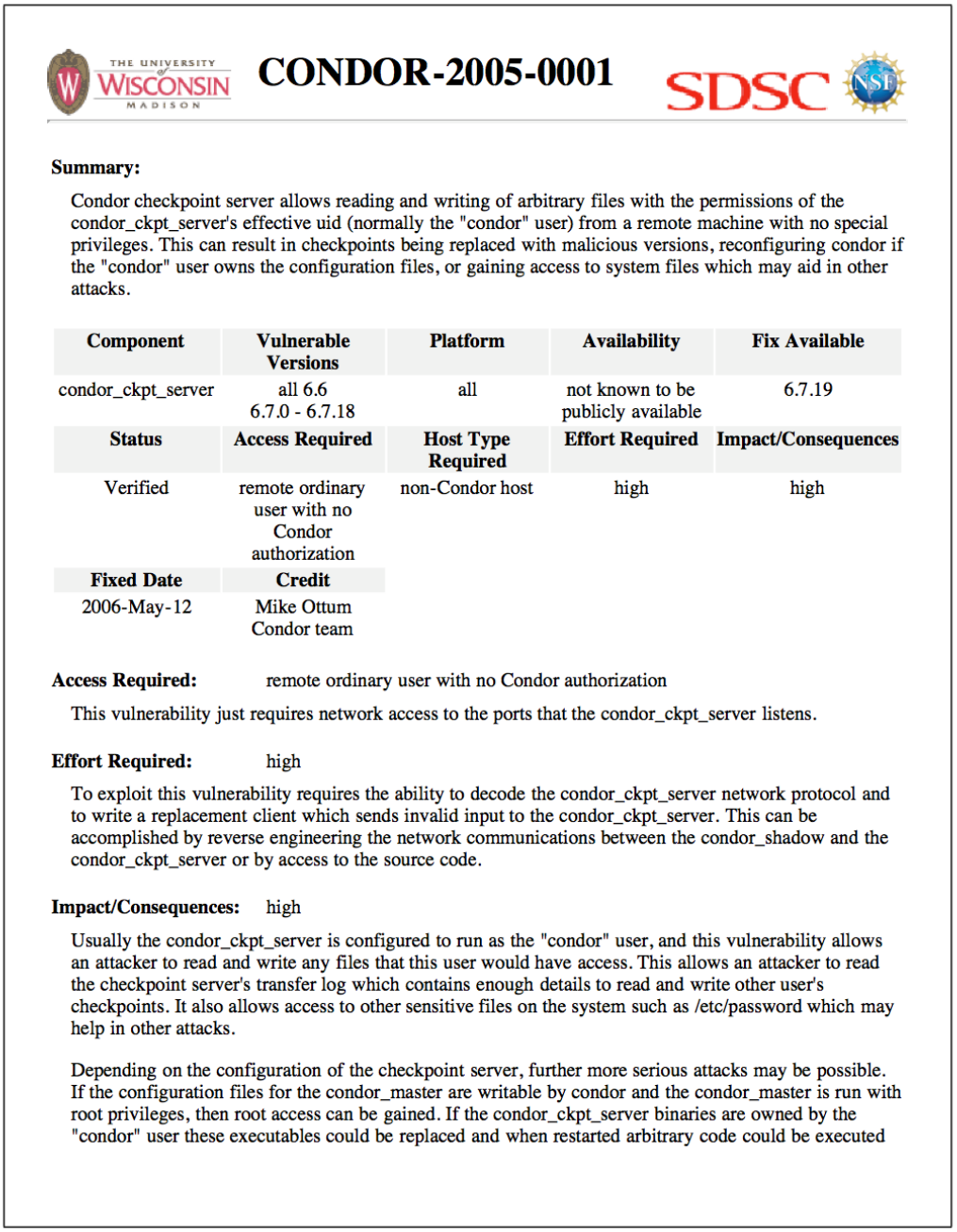
Proof of concept

Having spotted the possibility to manipulate the system outside of its normal functioning the analyst will focus on producing an actual exploit which proves that the attack is feasible. This is a critical step within the methodology, no vulnerability can reported if there is no practical way to abuse it and cause real damage to the system.

If the analyst produces a successful attack on the system he can then move on to the next step in the process, otherwise the suspected vulnerability is not exploitable and he returns to code reviewing.

Reporting

Having identified the source of the vulnerability and produced a successful attack the analyst then writes a report for the developers, this report will include his findings the POC. Including information that allows the reproduction of the vulnerability is one of the most important aspects of a report as this will allow developers to verify the issue and test their corrections, an example report is shown in Figure 2.3.



THE UNIVERSITY of WISCONSIN MADISON

CONDOR-2005-0001

SDSC

Summary:

Condor checkpoint server allows reading and writing of arbitrary files with the permissions of the condor_ckpt_server's effective uid (normally the "condor" user) from a remote machine with no special privileges. This can result in checkpoints being replaced with malicious versions, reconfiguring condor if the "condor" user owns the configuration files, or gaining access to system files which may aid in other attacks.

Component	Vulnerable Versions	Platform	Availability	Fix Available
condor_ckpt_server	all 6.6 6.7.0 - 6.7.18	all	not known to be publicly available	6.7.19
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	remote ordinary user with no Condor authorization	non-Condor host	high	high
Fixed Date	**Credit**			
2006-May-12	Mike Ottum Condor team			

Access Required: remote ordinary user with no Condor authorization

This vulnerability just requires network access to the ports that the condor_ckpt_server listens.

Effort Required: high

To exploit this vulnerability requires the ability to decode the condor_ckpt_server network protocol and to write a replacement client which sends invalid input to the condor_ckpt_server. This can be accomplished by reverse engineering the network communications between the condor_shadow and the condor_ckpt_server or by access to the source code.

Impact/Consequences: high

Usually the condor_ckpt_server is configured to run as the "condor" user, and this vulnerability allows an attacker to read and write any files that this user would have access. This allows an attacker to read the checkpoint server's transfer log which contains enough details to read and write other user's checkpoints. It also allows access to other sensitive files on the system such as /etc/password which may help in other attacks.

Depending on the configuration of the checkpoint server, further more serious attacks may be possible. If the configuration files for the condor_master are writable by condor and the condor_master is run with root privileges, then root access can be gained. If the condor_ckpt_server binaries are owned by the "condor" user these executables could be replaced and when restarted arbitrary code could be executed

FIGURE 2.3: Condor Report taken from the FPVA MIST Project Technical Report [1]

These reports generally include as much information possible to aid the developers in identifying the root cause of a vulnerability and can include recommendations by the analyst on how to fix it. These reports use metrics to indicate the severity of a vulnerability taking into account the ease of exploitation, the impact and the required access of a vulnerability.

There are certain constraints set by the methodology on reporting and communications between the assessment team and the developers.

First of all a specific vulnerability can only be reported once, once a report has been delivered it is considered that the same issue cannot be reused to perform another attack as it is in the process of being fixed. Reporting the same vulnerability multiple times would yield low value to the assessment effort, as such when a vulnerability is found the analyst is to present the attack that produces the largest impact.

Second the methodology takes care to isolate the developers from the analysts in order to minimize bias, allowing the analyst to have a fresh view on the components without being exposed to assumptions made by the developers. Not all methodologies take this approach, Microsoft's threat modeling methodology cited previously for example advocates close collaboration between the two teams. This discussion is however outside of the scope and field of this work.

2.4 Related Work

There are a number of existing vulnerability assessment methodologies but few which focus on manual vulnerability identification and even fewer which are dedicated to very large software projects.

The one that shares the most similarity with FPVA is the Microsoft Threat Modeling (MTM) methodology [12], it proposes an approach based on viewing the system from an attackers point of view. This is done by identifying high values assets which represent what the application wants to protect, integrating this information with architecture diagrams to understand how the assets are used, decomposing the application to create a security profile and then identifying and documenting the vulnerabilities found.

This is similar to how FPVA models resources and processes to identify high value assets. However there some key differences between the two methodologies. First of all in MTM the search for vulnerabilities is guided by lists of known threats, this is fundamentally different from FPVA which emphasizes the discovery of new vulnerabilities.

Second MTM focuses on assessing software during development from an early stage and encourages communications and interactions between the developers and assessment team where FPVA can be applied during late development just prior to release but is commonly applied after release and takes care to isolate the assessment team from the developers to avoid bias.

Other methodologies exist, for example the OWASP testing framework [19] combines multiple methodologies including MTM to introduce vulnerability assessment at various stages of the software development. However they are more generic and do not specialize in the assessment of large software projects.

2.5 Ethical Considerations of Vulnerability Reporting

Ethical considerations in security assessment arise from the disclosure of encountered vulnerabilities. As vulnerability reports are produced they are made available to the developers, however these reports should not be made public immediately. Depending on the release cycle of the project results will be disseminated in the appropriate fashion. This stems from the consequences that such vulnerabilities can have on the users of the system, particularly if you consider that the full reports contain sample exploit code. Releasing a vulnerability to the public before the issue has been addressed puts both the user base and the developers at risk. This is contrary to the goals of vulnerability assessment and it is the responsibility of the analyst to ensure that this is avoided.

One method, proposed by the FPVA methodology, is to release vulnerability reports incrementally. In a first phase the results are submitted to the developers, once time has allowed for the release of a security update (whether imminent or in the planned security release cycle) then a first publication of the vulnerability in a redacted form can be published publicly. Later the full reports can be released once time has allowed clients to update their software (although this step might be omitted).

Naturally this can vary depending on the agreement that ties the software producers and the assessors. For analysis of closed source components it is possible for the reports not to be publicly disclosed. The argument has been made though that both users and developers benefit from the public divulgation of security vulnerabilities.

The conditions under which this work was conducted are ideal for vulnerability reporting as it is an official evaluation made at the request of the software producer. In other situations were vulnerabilities are found in the wild, whether in commercial or open source software, contact has to be made with the stake holders to reach an agreement before considering publication, again leaving time for the software producer to publish a fix.

The issue can become more problematic when these fixes do not appear after reasonable amount of time or when there is limited motivation in fixing the issues. In that case vulnerability disclosure before the publication of a fix can be discussed, at least in a heavily redacted format. This is however a complex issue and there are different opinions within

the community [20] but in any case the problem should always be approached carefully with the best interest of the user base in mind as there can be grave consequences to releasing an unfixed vulnerability in the wild.

Chapter 3

gLite in the Grid and the role of CREAM

This chapter presents the general concepts of Grid computing, how the gLite middleware implements those concepts and the role of CREAM within that middleware. It then goes on to define CREAM's features, architecture and security.

3.1 Grid Computing Overview

Grid computing is a term that was introduced in the late 90's [21] to define a type of distributed computing architecture. It distinguishes itself through its loose coupling, heterogeneous nature and geographical dispersion.

This definition was refined in 2000 [22] as:

> "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations."
>
> "The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem- solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization. "

The purpose of the Grid is to allow the sharing of high-end computation capabilities through resources that are scattered and controlled by a variety of organizations while offering reliability, consistency and low economic costs. This is not a trivial task and justifies the need for hierarchical architectures like the ones proposed by Foster.

Figure 3.1 shows the main hierarchical and architectural components of the Grid, let us define its components:

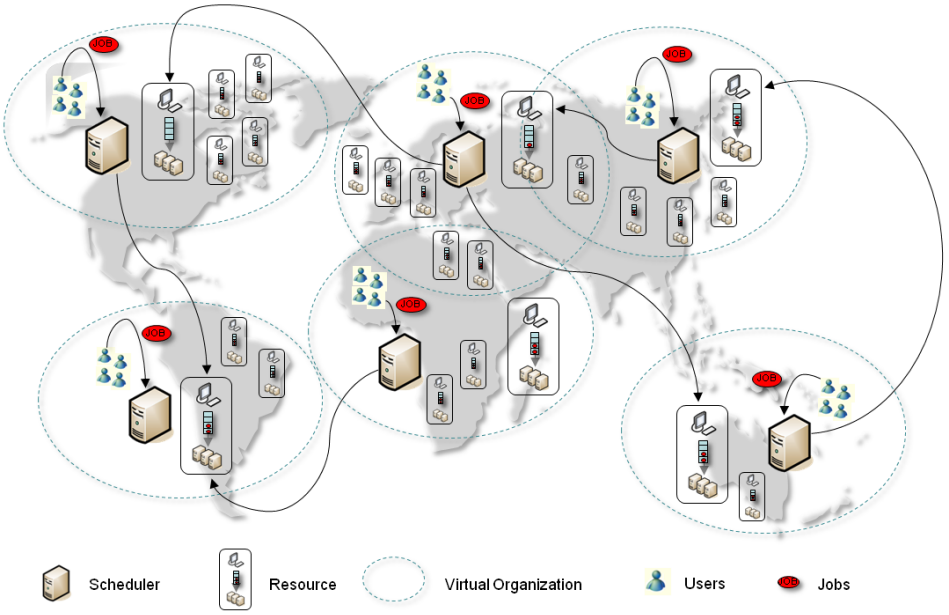


FIGURE 3.1: Grid computing architecture taken from [2]

Virtual Organization

Virtual organizations (VO) were introduced to resolve the sharing of resources across organizations. A VO is composed of all the organizations collaborating at one point, this can dynamically change over time as actors enter and leave projects.

Each organization participating in such a project can offer a number of resources to other organizations included in the project and expect access to certain others. A VO can contain a number of different rules to administer resources, for example the allowed execution time, the maximum number of resources that can be used concurrently or access to specific resources. These sets of rules can be specific to a user or an organization and depend on the agreements made with the project.

VOs serves as a concept to describe those complex inter-organizational relationships and regulate the use of resources while keeping flexibility to accommodate change for the purpose of achieving collaborative goals.

Scheduler

The scheduler is a structural component of the Grid that allows the distribution of tasks amongst resources. It serves as an aggregated view of the whole or a particular subset of resources in order to distribute jobs across them taking into account the rules of the VO. It often offers features like load balancing, dealing with sequential jobs and data placement. The scheduler is not required within the grid, in which case jobs are sent directly to a resource, but serve as a centralizing mechanism that can facilitate the use of resource and the overall performance.

Resource

A resource is what is being shared on the Grid. Resources are generally computers available for computations, storage or data. The purpose of the grid is to allow the aggregation of these resources so that they can be used collaboratively for the purpose of problem solving.

Computational resources are grouped in one or multiple Computing Elements (CE) built around a specific purpose, for example a CE can be a number of machines equipped with accelerators while another unit can be machines that only have regular processors. Storage facilities are grouped in one or multiple Storage Elements (SE).

There is no requirement on the homogeneity of resources within the Grid as long as those that comprise a CE can be made to function together.

User

Users are the actors of the system. They are the ones who have jobs that require execution on the resources available in the Grid. They are part of one or multiple VOs and have specific privileges on each of them.

Job

Job execution is the actual goal of the Grid, they are sent by users and organization and in general are resource intensive tasks that would not be executable in a reasonable amount of time without resources contained in the Grid. There is no specific definition of the characteristics of jobs as the purpose of the Grid is to adapt to many different use cases, but it can be broadly defined as one or a combination of executables that receive certain inputs and produce outputs needed by the user.

This gives us a general idea of the components of the Grid, however this can still lead to confusion with other types of architectures. Worried that the term might lose its original purpose, the original authors of the definition produced an additional three requirements for a system to truly be a Grid [23].

Decentralized Control of the Resources

The resources that constitute a Grid and its users must not be under the control of a single entity, they must be controlled by various control domains. This distinguishes the Grid from local management systems. The Grid deals with the issues of such an environment like security, policy, payment or membership.

Standard and Open General-Purpose Protocols and Interfaces

The Grid must be built around multi-purpose protocols and interfaces that deal with authentication, authorization, and resource discovery and access. This distinguishes the Grid from application specific systems. It keeps the Grid as an open concept with homogeneity in the way it functions that compensates for the heterogeneity of its resources. By keeping the protocols and interfaces standard and open the Grid keeps its original general-purpose nature.

Non Trivial Quality of Service

The grid allows the use of the resources that it comprises and adds values to these resources. The quality offered come form coordination for example security, availability co-allocation of multiple resource types. This gives a true added benefit from using the Grid, to allow the users to focus on the problem they are solving and not unrelated issues.

With these criterions in mind it is now possible to identify whether a system is truly a Grid, the purpose of most of its components and the problems they are meant to solve.

3.2 The gLite Middleware

gLite [24] is a middleware which includes many components with the purpose of building Grids. It was produced in 2004 as part of the Enabling Grids for E-Science (EGEE) [25] initiative and is currently maintained by the European Middleware Initiative (EMI) [26].

It is renowned for being the Grid middleware used by the CERN [7] for its Large Hadron Collider (LHC) experiments and has been adopted by 250 computing centers around the world for their daily operations. It is a large project that was originally the fruit of the collaboration between more than 80 programmers and has been in constant evolution since its creation. It is actively maintained and new features are introduced regularly with teams of developers assigned to specific components.

To give and idea of the scale gLite, each software component runs in the tens to hundreds of thousands lines of code for a total in the millions. Each component also has multiple

and complex interactions with other components. This is necessary to offer all the features required from a modern and flexible Grid middleware but is also what makes this type of software complex to assess from a security standpoint. There are many different mechanisms to be taken into account, a good overview of the system is required to understand what each features does and more importantly what it's not supposed to do.

gLite regroups a large number of software components and integrates with preexisting external components for the purpose of a building a Grid. The main components that were developed for gLite are those specific to the Grid like the scheduler and the VO management system. Services specific to resources like resource managers and file transfer systems were however integrated from external components by offering unified communication interfaces independent of the specific resources.

Figure 3.2 shows the main architectural components of gLite that are relevant to this project. Let us give a brief introduction to the purpose of each of these components.

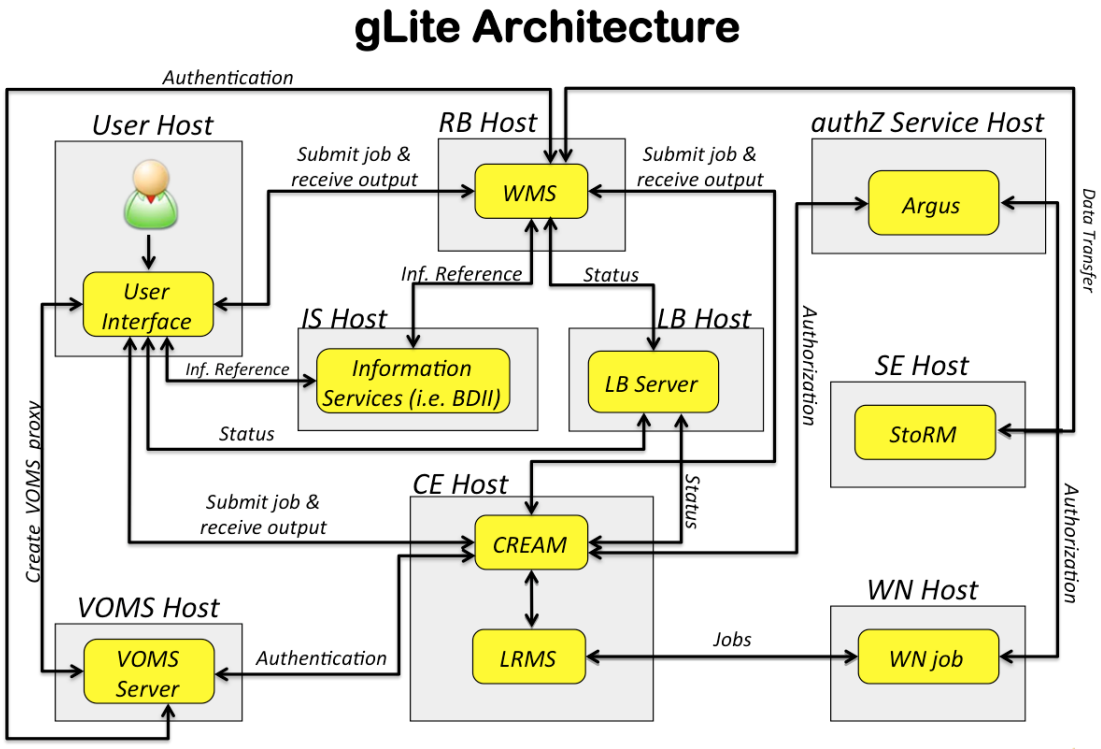


FIGURE 3.2: gLite architecture diagram

User Interface

The user interface is used by the user to interact with the system primarily for job submission, data transfer and authentication.

WMS: Worker Management System

The WMS is the scheduler of the system. It receives jobs and assigns them to the most appropriate CE, it also deals with the submissions and retrieval of data.

CE: Computing Element

The purpose of the CE is to allow seamless integration with the LRMS. The CE offers a unified interface for job submission and data transfers to and from the LRMS independently of the specific LRMS used. The CE also allows the WMS to access information like the number of jobs currently running, the number of available WNs and the state of jobs.

WN: Worked Node

The WN is the computational resource, they execute the jobs. There can be any number of WNs from one to thousands and there is no specific requirement on their homogeneity as long as all of the WNs in a CE use the same LRMS. A job sent to the CE will be sent to one or multiple working nodes, depending on configuration, where it will be executed.

LRMS: Local Resource Management System

The purpose of an LRMS or batch system is to dispatch the work from the LRMS server to LRMS client or WNs. The LRMS handles the transfer, execution and retrieval of jobs but at the level of a local organization unlike the CE which dispatches jobs across VOs.

There are a number of LRMS systems supported by gLite, for example Torque, OpenPBS, CONDOR and sun Grid engine. Each of these is developed and maintained independently of gLite by other actors and as such are not part of the gLite middleware.

LB: Logging and Bookkeeping Service

The LB collects information about jobs that were submitted by users and the events that occurred through the job's life cycle. This tracking combines information from jobs submission to the WMS onto the CE and finally the WNs.

SE: Storage Element

The SE provides access to data required by the jobs submitted to the CE and are often massive storage units. The CE will access the SE to retrieve files that are necessary for a jobs execution and later store the results of these jobs for the user.

ARGUS: Authorization Framework

ARGUS serves as a unified authorization system for distributed services. It allows all the CE's to receive consistent authorization decisions based on centralized authorization policies. This serves to ensure that all the components of the gLite Grid use the same set of policies. These policies do not regulate users but systems and their interactions within gLite.

VOMS: Virtual Organization Membership Service

VOMS manages users privileges within a VO, this includes information about the user, the resources it can access. It serves as a centralized source for user authorization information. It is used by the user to generate temporary certificates that are sent alongside jobs to authenticate the user as well as by the CE to verify the validity of these certificates. VOMS also offers administration interfaces that allow adding, modifying and removing users of VOs.

IS: Information Services

The IS holds information of user accounts and privileges inside the gLite system. These user accounts are those used to run jobs and access information across systems. It uses Berkeley Databases to provide uniform accounts across Linux systems.

The gLite architecture diagram gives a general overview of how gLite can be used and the interactions between components. However no two systems are alike depending on their goals and specific usage scenario. Other configurations are possible and certain components can be replaced or removed. Some components are mandatory like the CE but others like the scheduler, WMS or the SE are not.

As we can see gLite contains a large number of components to provide the services required for Grid computing described in Section 3.1. If we compare Figures 3.1 and 3.2 we can see that all the architectural components of the Grid are present within gLite. It offers all the facilities required to run and maintain large scaled Grids.

Our research group, MIST, has been actively involved in securing these components. Previous work on gLite includes the assessment of VOMS and ARGUS. This thesis focuses on the latest assessment within gLite, the CREAM CE.

3.3 CREAM

The CE is one of the cornerstones of gLite, without the CE there is no way to interact with computing resources. gLite can work with two different CEs, the first is CREAM

which has been developed within the gLite middleware and the second is CONDOR [27] which is developed by the university of Wisconsin.

The Computing Resource Execution and Management (CREAM) was released in 2006 [3]. It is a component of the gLite middleware whose purpose is to offer a generic interface between the gLite middleware and the LRMS for the execution of jobs. It also extends the functionalities the LRMS by offering features required in a Grid environment like authentication methods based on VOs and load information for schedulers.

The responsibilities of CREAM are centered on jobs; each of its functionalities is related to an action necessary to carry out the execution of a job. The job lifecycle starts when the job is submitted, through its execution and until the user recuperates the results. The interface offers features that are specific to the current state of a job.

CREAM does not require all the components of gLite to operate. The only mandatory component is VOMS for authentication purposes. This makes it useful for building Grids but also as a wrapper around an LRMS. It allows the creation of jobs using the generic job description language (JDL) regardless of the specific syntax used by the LRMS. This facilitates the submission of jobs for users as they do not need to deal with the specificities of the system.

Interactions with CREAM are handled through SOAP web services. There are a number of web services running each offering a specific functionality. The advantage of web services is that it allows a great deal of freedom in the implementation of components that interact with CREAM, whether inside the gLite package or from outside sources, as almost every programming language has mechanics to interact with web services. To communicate with CREAM all an external component has to know is the interface definition of the service. This is in accordance with the principle of open and unified interface and protocol seen in the definition of the Grid.

3.3.1 Architecture

CREAM is entirely built upon Tomcat as a Java-Axis servlet as such all of its processes are contained within Tomcat. Requests from users and components come solely through SOAP web services. This offers a single unified interface offering all the features supported by CREAM.

CREAM has three different web service interfaces running concurrently by different Java threads as shown in Figure 3.3. The first one, labeled as CREAM, offers job management features. The second one, labeled delegation, handles the delegation of proxies which involves particular communications with the client and VOMS. The third

one, labeled CEMON, offers features required by the WMS, it serves information relevant to scheduling.

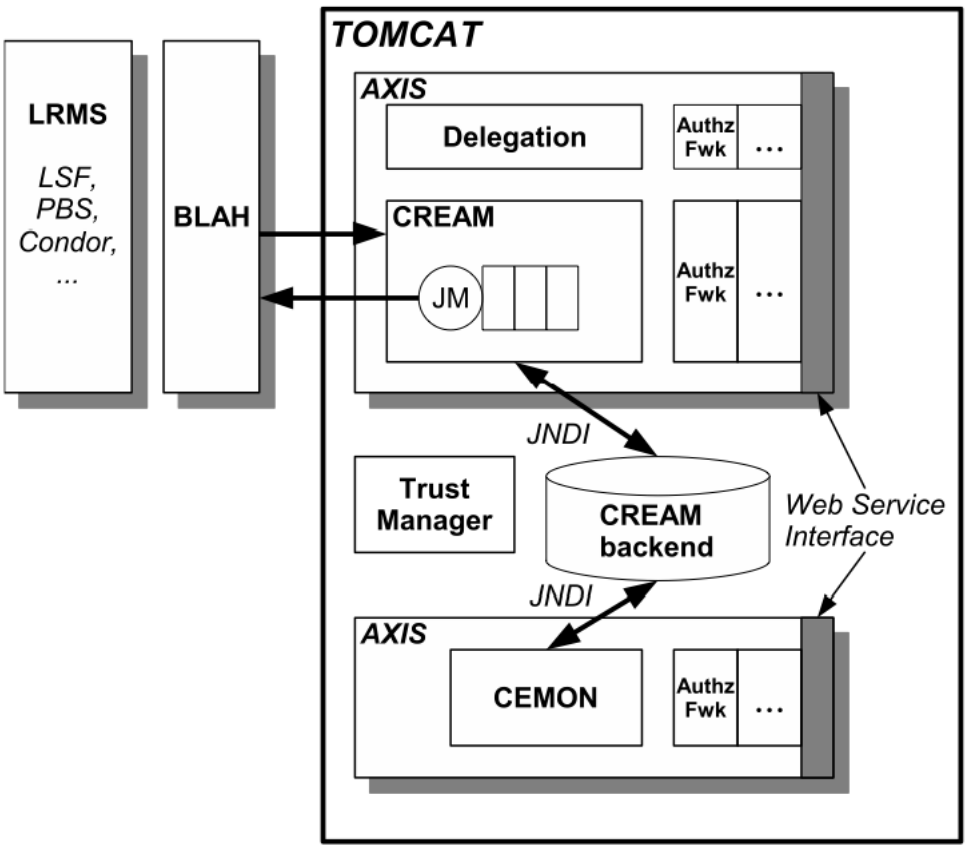


FIGURE 3.3: CREAM architecture from [3]

All requests sent to CREAM are first authorized before being processed, when a specific user sends a request the authorization framework will contact VOMS to verify whether the user has sufficient privileges to carry out that request. This mechanism serves both for regular users and administrators of the system.

CREAM interface

This process handles requests from external component relating to job management, it offers the main features (see Section 3.3.2 for the complete list) of CREAM from the standpoint of users. When a request is received it will create a new thread of execution to handle that interaction while the main process stays available to receive new requests.

Requests can fall in one of two categories, those that are contained within CREAM and those that require interaction with the LRMS.

Functionalities like the status of jobs or the status of the CE can be directly answered by CREAM, all the necessary information is contained inside its database. This information is returned through the SOAP interface.

Other requests like job creation or job cancellation require communication with the LRMS. In that case the process will insert the required commands inside the database into a queue, initiate possible file transfers and then confirm the success of the operation. At this point however the LRMS has not yet been contacted and the command is still contained within the CE.

Journal Manager

The Journal Manager (JM) is a pool of threads that carry out commands that were queued inside the database. This serves as failsafe mechanism in case of interruptions as the command queue is in persistent storage and only cleared when a command has been successfully communicated to the LRMS.

When a thread is attributed to execute a command, it retrieves the information from the queue and sends it to another component of the middleware. The interaction with the LRMS is not handled by CREAM but is abstracted through the Batch Local ASCII Helper (BLAH) component. BLAH offers a unified interface to LRMS commands independent of the specific LRMS in use. The JM thread will contact the BLAH who will convert the generic command into its LRMS specific equivalent and execute it. At this point the request has traveled all the way from outside the CE to its destination, the resources.

The BLAH process can also communicate with CREAM in order to update the status of jobs and initiate the transfer of results from the worker nodes to the CE for retrieval by the users. The information sent to CREAM by BLAH is stored inside the database.

Delegation interface

The delegation interface handles the delegation of proxy certificate that are used as a temporary authentication method for jobs submitted by users. Before a user can use a CE it has to request a proxy certificate from the VO user manager (VOMS), this certificate is then sent to CREAM and will be stored locally by the CE until the job is completed.

This certificate is used to authenticate the user within the CE but also to allow jobs to communicate with other resources of the Grid like storage elements.

CEMON interface

The CEMON interface is used by the scheduler (WMS), the scheduler will periodically query information from all the CEs that are under his control for information

required by its operations. The CEMON use the information stored within its database by the CREAM interface to provide information over the status of its resources.

While the architecture of CREAM might appear complex for its purpose and has many levels of indirection, it is necessary to provide the features expected from a component of the Grid, i.e., security, quality of service and flexibility of each component. The database serves as a centralized communication method within CREAM, this facilitates the interaction of its various processes while increasing reliability and being thread safe.

3.3.2 Functionalities

CREAM offers a number of different functionalities for each use case, for the user and the administrator these come in the forms of programs bundled in the CREAM client. These programs are implemented in C++ as lightweight implementations of the API, they contain very little logic and their main purpose is to pass the information from the client to the CE using the web service interface.

CREAM offers a well-defined set of functionalities. For users it offers commands related to job management:

- Proxy delegation: glite-ce-delegate-proxy submits a proxy certificate generated by VOMS to the CE, if valid this proxy can then be used to submit jobs.
- Proxy renewal: glite-ce-proxy-renew allows to extend the lifetime of a proxy previously submitted to avoid its expiration, it is used when jobs are still running to avoid their automated cancellation.
- Job submission: glite-ce-job-submit is used to submit a JDL file to the CE and associate that job to a previously delegated proxy, if the job is accepted the required files will be transferred using GLOBUS to the CE.
- Job status querying: glite-ce-job-status returns the status of one or multiple jobs that belong to the user, this status contains a full manifest of all what has happened to the job during its lifetime as well as its current state.
- Job cancellation: glite-ce-job-cancel allows to cancel a job that is currently running or pending execution.
- Job output recuperation: glite-ce-job-output is used to transfer all the files that were outputed as a result of a jobs execution from the CE to the client.

For administrators it offers a few management and maintenance commands:

- Removing old or stale jobs: glite-ce-job-purge allows the administrator to purge all the jobs whose execution completed to free space on the CE. This command is also available to the user but is then limited to his jobs.
- Enabling or disabling job submission on the CE: glite-ce-enable/disable-submission allows the administrator to disable submissions without affecting currently running jobs. This can be used to allow the job queue to empty itself before maintenance work on the CE and without disturbing previously submitted jobs.

And for schedulers it offers data polling commands:

- Job status notification subscription: Automated notification of job status changes for the scheduler to avoid active polling.
- Job status polling: Active polling of the status of a job.
- CE status polling: Polling of the status of the CE provides the scheduler with information like the number of jobs queued and the available resources on the CE for scheduling purposes.

Chapter 4

CREAM Architectural, Resource and Privilege Analysis

This chapters presents the first the steps of the analysis, the architectural, resource and privilege analysis, that are required before the component analysis can start. First it introduces the preparations and the environment of this analysis, then presents each diagrams produced and details various key aspects of CREAM that were discovered during the analysis.

4.1 General Setting and Methodology of the Analysis

The first step defined by the FPVA methodology is the elaboration of the architectural and resource diagrams, but there are some pre-requirements that have to be fulfilled before this analysis can begin. The assessment process requires a functioning test environment (testbed) as the diagrams are based either from observations that come from the code or the software's execution.

Some methodologies make use of existing diagrams that were made by the software producer, while the FPVA analyst might consult these diagrams at an early stage to understand the general architecture of the application but will not assume this information to be correct. There are two main motivations for this. First diagrams within a project are often outdated, diagrams are produced during the analysis and design phase of development but there is no guarantee that they accurately reflect the current state of the software. Second diagrams that are useful for developers tend not to contain the information required by the analyst. An analyst will focus on component

interaction, communication methods and resources involved but will abstract most of the implementation details and business logic.

This explains the need for a functional testbed from the very beginning of the assessment. The testbed is where analysts will trace executions, analyze logs, review the database but also where they will test exploits (further explained in Chapter 5). Naturally this should never be a production system with real users or valuable data. It is not necessary for the testbed to have the scale of an actual Grid but rather a small-scale functional system that best mimics real conditions. In this case the limit is set on the number of resources (worker nodes). This strikes a balance between properly reproducing real world conditions and the budget and infrastructures available for the assessment.

The installation of the testbed is also the first contact the analyst has with software. Having to build the system from the ground up greatly increases the knowledge of the various components of the system and its configurations. CREAM was installed following the official documentation focusing on adhering to the default configurations. It is important to install the system properly to avoid introducing vulnerabilities due to improper configuration. Vulnerability assessment focuses on finding vulnerabilities within the software when it is properly installed and not on issues that arise from improper system administration.

The testbed only contains components which are required by CREAM, i.e. the CREAM CE, VOMS, two worker nodes and a CREAM client each running on an individual host. Installing the whole of gLite would introduce unnecessary complexity and is not required for the assessment as the absences of components like the storage elements or the scheduler does not limit the functionalities that can be tested.

The installation was made on a single host using virtual machines to emulate the five hosts of this setup. Using virtual machines offers the advantage of limiting hardware requirements while offering flexibility when modeling the architecture and a solid backup mechanism to restore components to their initial state after exploits are tested.

Once the setup has been installed some time is spent to familiarize with the functionalities and mechanisms of each component. This is done in two steps, first the analyst focuses on the role of the user and then the role of the administrator.

As a user of CREAM this includes the steps of job submission, using the Job Description Language (JDL) to define jobs, retrieving output and testing various types of jobs. This serves to give a general understanding of how a user interacts with the system, what he expects at various stages and the general requirements to perform specific tasks.

Once the analyst is familiar with the interactions of users he can then explore the possibilities offered to the administrator. Within CREAM this includes disabling submissions, purging old jobs and solving problems with the various components.

Having setup, used and administered the system the analyst has gathered enough knowledge to start studying the internal mechanisms of each component and produce the diagrams. These steps are not outlined by the methodology but are important to give the analyst the means to perform his work and give meaning to the information gathered.

4.2 Architecture Diagrams

The first series of diagrams presented are the architectural diagrams. Architectural diagrams represent hosts, their process and their interactions.

The first diagram shows the overview, this is a high level view showing just the processes and their communication protocols. The second diagram shows the client server interactions which depicts all the interactions involved when submitting a job. The third and final diagram focuses on the components of which cream is composed CREAM internally, while these are not processes in the classical sense it shows how responsibilities are delegated within the code.

The architectural diagrams provide the analyst with a great deal of information on the attack surface, attack vector and the general structure of the software.

4.2.1 Architectural Diagram Legend

The architectural diagrams are meant to be simple to read and comprehend but certain core elements have to be defined. The legend shown in Figure 4.1 presents each item present on the architectural diagrams.

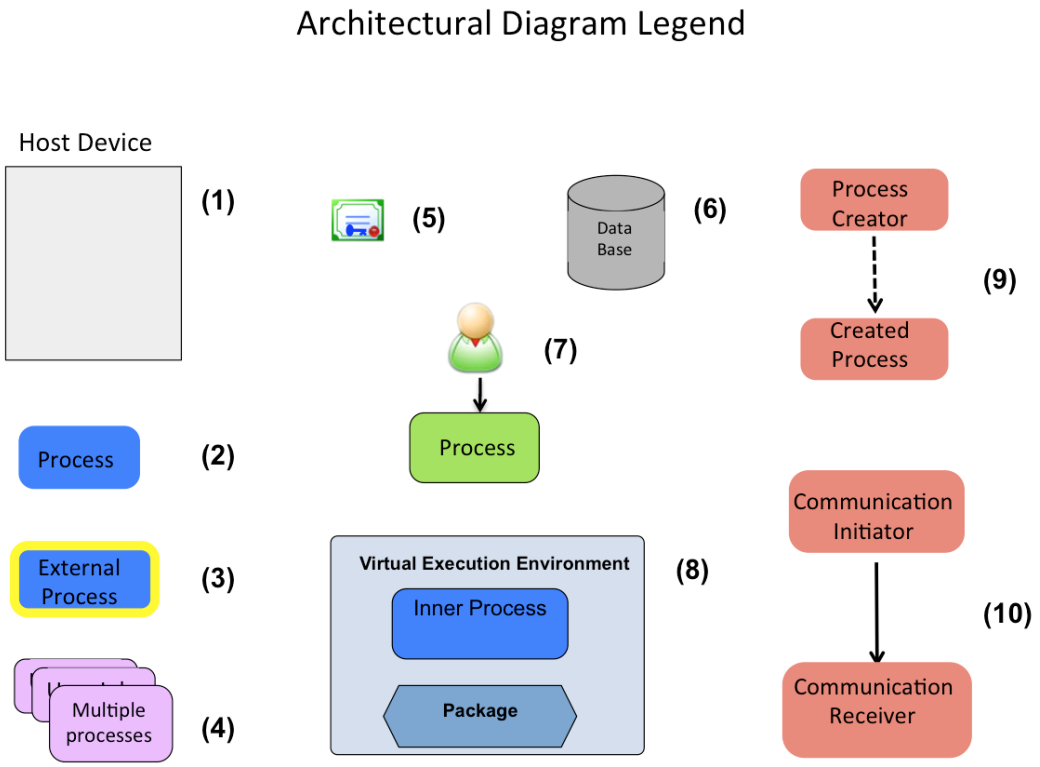


FIGURE 4.1: Architectural Diagram Legend

- 1 : Host device. Each host is represented as a separate entity, a host is a separate system that runs specific process and contains resources.
- 2: Process. Each process running on a host that is relevant to the assessment is represented and named, its color represents the execution privileges under which it runs. Each process that is identified is part of the assessment and will be reviewed.
- 3: External Process. Similar to a process however external processes are identified as they are not part part of the assessment. This means they will not be assessed for vulnerabilities but they might still be part of an exploit.
- 4: Multiple Processes. When representing the creation of processes the number of processes created is not always defined and can vary. In that case the multiple process representation is used.
- 5: Certificate. A certificate used to identify the user within the system.
- 6: Database. The database is represented as it is a running process, the database is also a resource but in this case it seen as a process which accepts communication.
- 7: User. A user and his interaction with a component.

- 8: Virtual Execution Environment. For code running under in a virtual machine like Java the process is represented as running within the confines of this virtual machine.
- 9: Process Creation. The dotted arrow represents the creation of a process by another. The direction of the arrow shows the parent process creating the child.
- 10: Inter Process Communication. The continuous arrow represents a communication between two processes, this arrow can be annotated with the communication protocol and the direction shows the flow of information.

4.2.2 Architectural Overview Diagram

The first diagram produced was the architectural overview, Figure 4.2, this diagram shows the various components in a very abstract form focusing mainly on communications and the location of each component.

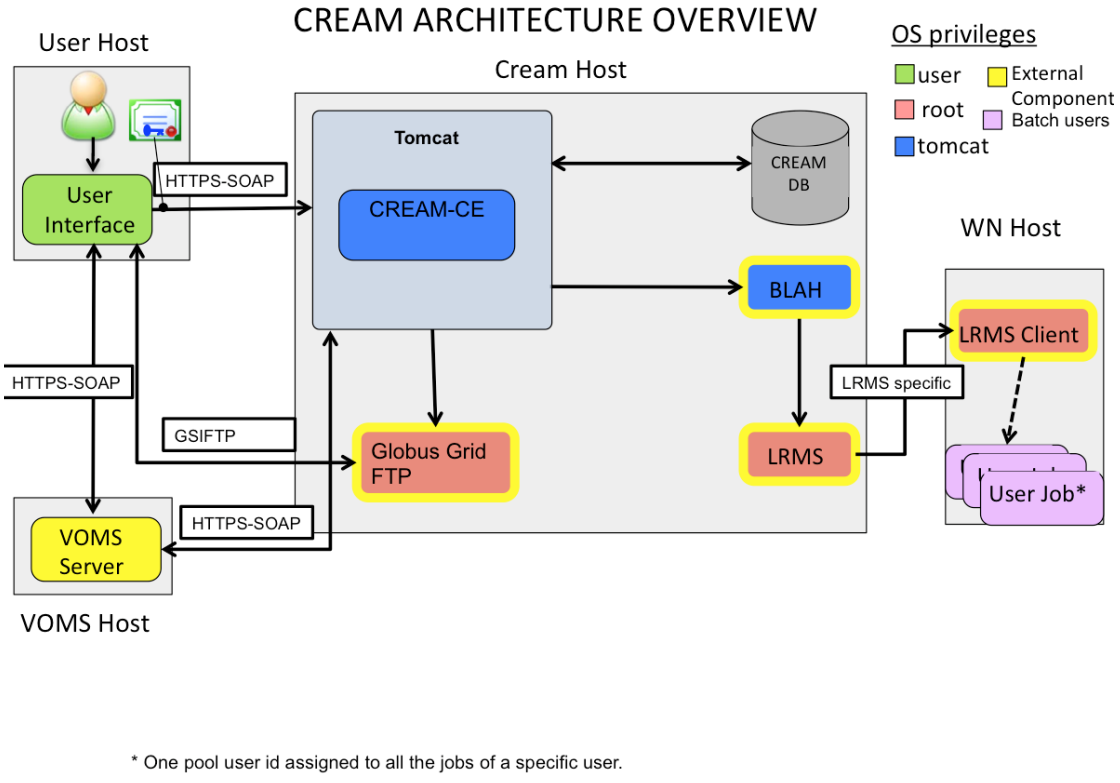


FIGURE 4.2: CREAM Architecture Overview Diagram

The hosts shown are the client, CE, VOMS and the Worker Nodes (WN). Hosts detail the processes running (these are the components of the middleware), whether a component belongs to CREAM and its privileges. Components that are external to CREAM are

not part of the assessment and are not reviewed for vulnerabilities, they can however be used to perform attacks if they can be manipulated to perform malicious actions.

Each communication between hosts is labeled with its protocol. This offers a first insight into attack surfaces, for example the CE communicates with the client using HTTPS encrypted SOAP web services and the GLOBUS FTP protocol. The first is used to offer the functionalities of the CE to the client (as seen in Section 3.3.1) while the second is used for the transfer of job files. This gives an initial feeling for possible attacks, particularly communications and API attacks.

4.2.3 Interaction Diagram

The second diagram is the client-server interactions, Figure 4.3, it details all the steps involved in the submission of a job on the CE.

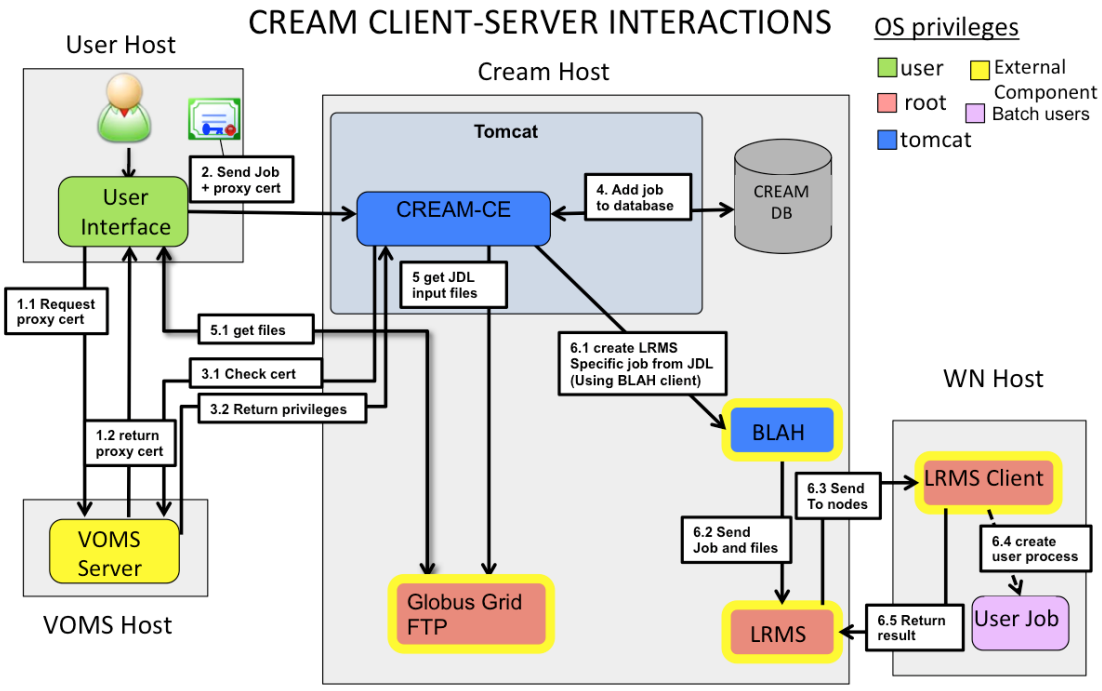


FIGURE 4.3: CREAM Client-Server Interaction Diagram

Job submission is one of the core services offered by CREAM and involves the interaction of all of its components. This is why the job submission was used to present the CE's interactions as it gives a general idea of how data flows within CREAM. Other services offered by the CE follow the same general pattern of interactions.

The interactions diagram shows the order in which tasks are handled and the responsibilities of each component. It also details the general steps of the job submission process. The whole process is not a single user action, for example step 1 which acquires the proxy certificate is handled by the VOMS client while step 2 is carried out by the CREAM client and are separate actions by the user.

The steps involved in the submission of a job are the following.

Certificate Acquisition

To submit jobs on the CE a proxy certificate is required. This certificate is generated by the VOMS component and serves to authorize the user on the CE. Proxy certificates are a temporary form of authentication, by default they last 24 hours, and are used to mitigate the security risk of sending the real users certificate to the CE. If the CE is compromised or the proxy certificate is intercepted it is only of very limited value due to its short-lived nature.

Certificate Delegation and Job submission from the client to the CE

The certificate and the job description are sent to the CE. The CE will store the certificate associated to a delegation identifier, this identifier can then be used to send multiple jobs using the same certificate. The information over the job submitted to the CE is only the JDL file that describes the various parameters of the job like the location of the executable, the location of the data and the files that have to be retrieved as output.

User Authentication and Authorization

The CE will immediately contact VOMS before storing any of the information over the jobs to confirm the users identity and establish whether the user has the privileges required to create jobs. The authentication is done using the users real certificate signing the data with his private key on the client side. The CE will then verify that this data is correct by comparing it to the information sent by VOMS. If everything checks out the job creation will proceed.

Database Storage

When the user has been authenticated the CE will then store the information over the job and the proxy certificate inside its database and call GLOBUS to launch file transfers.

File Transfer

GLOBUS is required to transfer the executable and input files from the client to the CE. The authentication of GLOBUS uses the certificate previously delegated and then transfers the requested files for storage on the CE. The files will reside on the CE until the job is sent to the worker nodes.

Job Submission from the CE to the Resources

When the CE is notified that there are free resources on the worker nodes the JDL is sent to BLAH. BLAH will translate the generic commands described in the JDL file into commands specific to the LRMS. BLAH will also generate a wrapper executable that will be executed on the worker nodes. The purpose of the wrapper is to launch the executable specified in the JDL with all the parameters requested by the user. This deals with site specific configurations like the location of the input data for an executable and authentication on storage resources, it can be modified if needed by the administrator to adapt to the specifics architectural details of a CE.

The submission of the files from the CE to the worker nodes is left to the LRMS. When all the files have been transferred to the resources the LRMS will launch the wrapper execution. When the execution is finished the LRMS will transfer the results back to the CE so that the user can retrieve the results.

The interaction diagram provides the analyst with information over which component accesses the data sent to the CE and how that data is used. Where the overview diagram gave information about possible attack surfaces the interaction diagram gives information on attack vectors. Knowing where specific input sent to the CE will be used and what actions are triggered guides the analyst in the assessment of a component to evaluate if it is vulnerable to an attack.

For example in the second step of the submission, the user sends the certificate and the JDL to the CE, some of this information is then stored inside the database. This indicates the possibility of SQL injection attacks from these inputs.

4.2.4 Component Diagram

The last diagram that is part of the architectural analysis is the component diagram as shown in Figure 4.4. This diagram is not always included in the analysis but is useful when analyzing software which runs within a virtual environment like java's Tomcat used by CREAM. The purpose of the component diagram is to show the various packages and their responsibilities in treating requests.

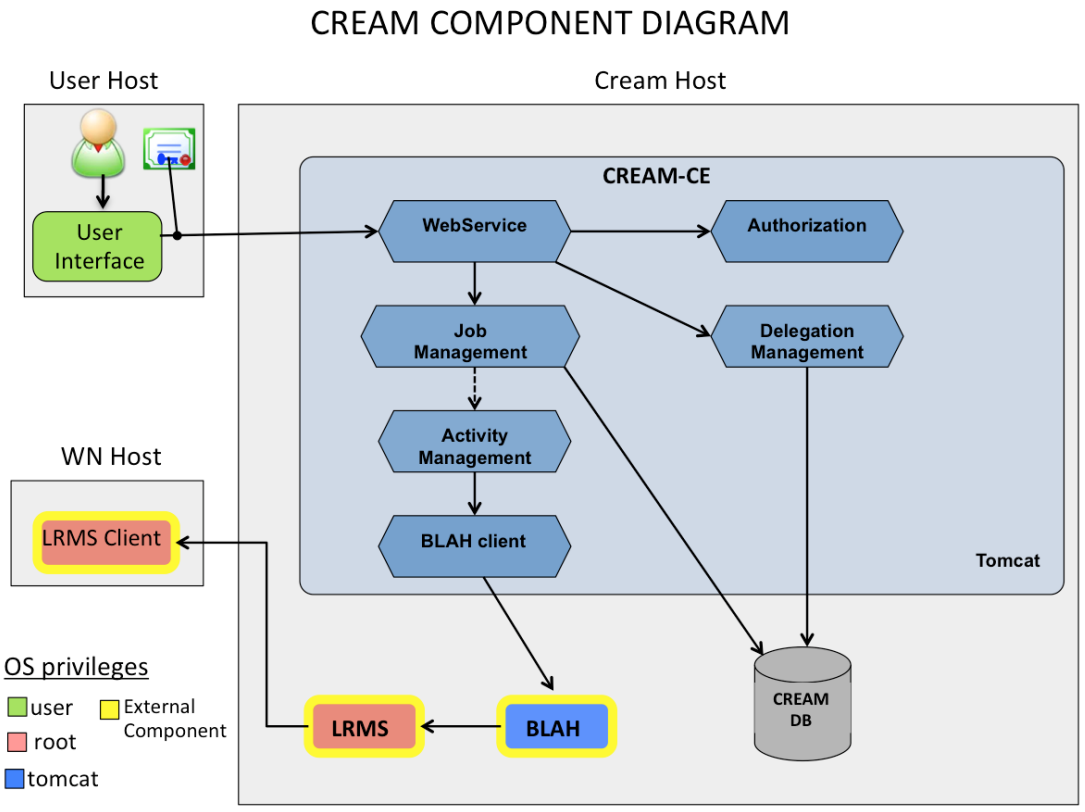


FIGURE 4.4: CREAM Component Diagram

WebService

The webservice package is the front end of the CREAM application, it offers the web services of the two underlying packages that represent the core functionalities of the CE. It is also the only direct means of communication with the CE for users, administrators and schedulers alike.

It offers SOAP web services over the encrypted HTTPS protocol, each web service takes a number of arguments depending on the precise service and additionally a proxy certificate for authentication.

Authorization

The authorization package is the first one called after a request is sent to the CE. It serves to authentify the request by communicating with VOMS to ensure that the identity submitted is legitimate and that the right privileges are met to execute a specific command. If it is successful then the CE will transmit the request further to the appropriate package while if it fails the operation is terminated and the user notified.

This package also facilitates further operations for the CE, since all request have obligatorily passed through authentication and were carried on only if the requierementts were met no further verification is required in other packages.

Job Management

Job management is the first package containing the core functionalities of the CE, it deals with all the commands that interact with jobs and includes job creation, job cancellation and job output retrieval. The package is called by the web service package when job related commands are called after the authorization process.

The package includes all the logic for in memory and database storage for the tables that are related to jobs. Instructions that will be transmitted to the LRMS are stored inside the database before creating a new activity which will invoke the BLAH client. Also included in this package are the operations for automated job cancellation due to certificate expiration.

Delegation Management

Delegation Management is the second package containing the core functionalities of the CE, it handles all the operations involving proxy certificates like proxy delegation and renewal. The package is called by the web service package when certificate operations are used.

The package includes database operations to store and retrieve certificates, when a certificate is delegated it is stored inside the database to be used when new jobs are created by associating the job with its corresponding proxy certificate.

Activity Management

The activity management package handles a pool of Java threads used to communicate with processes that are external to CREAM like BLAH. When a job management command requires interaction with BLAH a new activity is created, this activity runs in its own thread and will create a new BLAH client.

This mechanism is used so that the request of a client can be directly answered while operations that can incur delays are processed independently. Since the BLAH component is external to CREAM and therefore information over its state is not always accurate the package also handles multiple retries of a command until it is successfully processed.

BLAH Client

The BLAH client is a package used by the activity management package to communicate with BLAH. BLAH is an external process independent of CREAM and all the logic to interface with it is held within this package. Commands to be

transmitted to BLAH are stored within the database and then passed on to the BLAH client which passes this information on to the actual BLAH process that will handle the conversion of the generic command to one that is specific to the LRMS.

The component diagram provides a great deal of information over the structure of the code, this gives the analyst a precise idea of where each section of code is located and what type of information is passed on from one package to the other. This information is useful when the code is reviewed and serves as a road map when specific functionalities are assessed to locate the relevant sections of code.

4.3 Resource Diagrams

The second series of diagrams produced are the resource diagrams. Resource diagrams focus on identifying all the resources used by a specific component and how they are accessed. This includes configuration files, logs and databases. Resources are an important part of vulnerability assessment and gaining access to them is often what leads to successful attacks.

This section first presents the rating scale used to identify the value of assets that is used throughout this analysis and how it relates with the rating of vulnerabilities used during the component analysis followed by the resource analysis of the CE and the client.

4.3.1 Asset Value Rating

During the resource analysis the term asset value is used to refer to the potential damage that can be achieved by gaining access to a resource.

The FPVA methodology rates vulnerabilities on a four level scale.

- Level 0: False alarm, the vulnerability does not actually allow unauthorized access.
- Level 1: Zero-value vulnerability, the vulnerability allows unauthorized access but no assets of any value can be accessed
- Level 2: Low-value vulnerability, the vulnerability allows access to an asset but this asset holds little value to an attacker.
- Level 3: High-value vulnerability, the vulnerability allows access to an asset of high value which would have a great impact on the system.

The goal of the analysis is to help identify high-value assets (and to a lesser extent low value assets) so that the analyst can focus on finding vulnerabilities that have the highest impact.

During the resource analysis, resources are labeled accordingly. For example log files which if accessed reveal information of little value to an attacker are labeled low-value assets, while the database or host certificates are labeled high-value.

4.3.2 Resource Diagram Legend

The resource diagram have similarities with the architectural diagrams but new symbols are introduced specific to resources as shown in Figure 4.5.

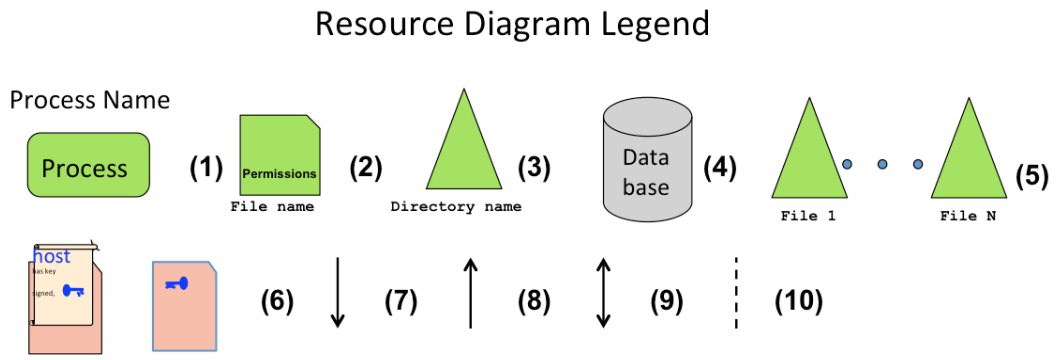


FIGURE 4.5: Resource Diagram Legend

- 1 : Process. Each resource diagram depicts the assets of a specific process. Similarly to the architectural diagrams, colors represent privileges.
- 2: File. The most basic resource type is files, a file has a name, a color to represent its owner and full permissions indicated.
- 3: Directory. To be able to represent the arborescence of a file system directories are used. Each directory has an absolute path to indicate its location and a color to indicate ownership.
- 4: Database. Databases are represented on the resource diagrams as they contain key information, they are named and their privileges are represented. In the architecture diagrams they were represented as processes while in the resource diagrams they are seen as resources.
- 5: Multiple Files. When an unknown number of files or folders are present this representation is used, it indicates the presence of one to many elements.

- 6: Host Certificates. The public and private pair of certificates used to identify the identity of a host.
- 7: Writing. An arrow pointing from a process to a resource indicates that the process accesses the resource with writing permissions.
- 8: Reading. An arrow pointing from a resource to a process indicates that the process accesses the resource with reading permissions
- 9: Read and Write. A bidirectional arrow indicates both reading and writing.
- 10: Resource Ownership. A dotted line represents that a file is owned by the user under which the process is executed.

4.3.3 CE Resource Analysis

The first diagram, Figure 4.6, focuses on the resources of the CE. As seen previously the CE runs within the Tomcat environment, the resource shown are all those accessed by the Tomcat process relating to CREAM during the operations of the CE.

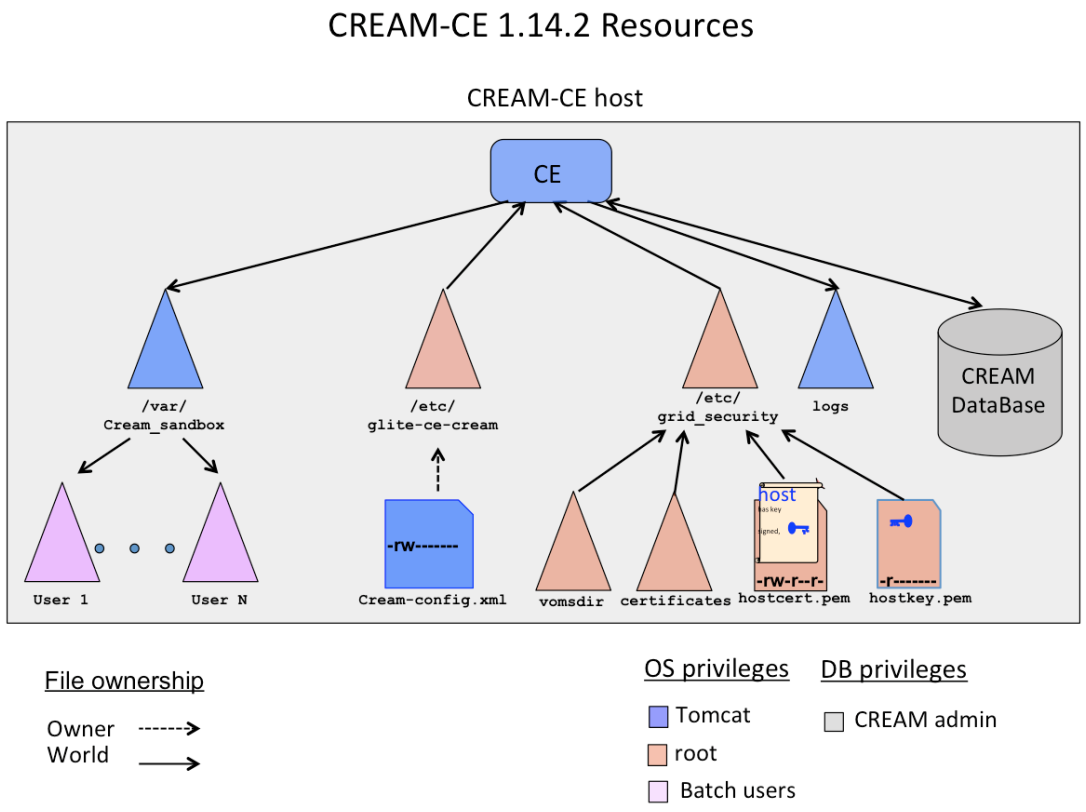


FIGURE 4.6: CREAM CE Resource Diagram

The CE uses relatively few resources and relies heavily on the database to store content with the exception of the CREAM configuration, logs, certificates and job files.

CE Configuration

The CE configuration file holds all the information necessary to launch the CE, it holds information on the location and credentials of the database, information over the VOMS server and other configuration settings required to launch the CE. Only the CE process reads this file, this is reflected in its permissions by being owned and only accessible by the TOMCAT user.

This is a high-value asset, if an attacker was to consult this file he would gain information that could facilitate an attack like the database user name and password. If an attacker were able to modify this file he would gain high control over how the CE performs and be able to modify most configuration parameter. By modifying the configuration file an attacker could for example force the CE to connect to an external database controlled by the attacker.

Job Sandbox

The job sandbox contains all the files submitted by the user for a job and after execution the outputs of those jobs. Each user of the VO has his own folder containing the data from all the jobs he has submitted.

On the CE each user is assigned a CE batch user from a pool of users created for that purpose, these users also exist on all the worker nodes and serve for execution. Each sandbox is owned by its corresponding batch user, which was assigned to a VO user.

The sandboxes are high-value assets, jobs are expected to stay confidential within a CE, input and output files can contain valuable data a should not be accessible by other users.

Logs

There are a number of log files on the CE that describe the operations that were executed. These files contain some information about the jobs submitted but do not hold sensitive information. Gaining access to them would be of little value to an attacker. These are low-value assets.

Host Certificates

The CE holds a number of public certificates to authenticate the hosts it communicates with and its own public and private certificate. These certificates are used to guarantee the identity of hosts, when a client connects to the CE it will use the public certificate of the CE (which is publicly available) to ensure the data was signed using the private certificate (which is only available to the CE).

Gaining read access to the public certificates holds no value to an attacker as these are meant to be shared. However being able to modify the public keys could allow certain attacks like host replacement. The private key of the CE however is an extremely valuable asset, if this key is compromised then attackers could replace the CE with a custom host without alerting the clients. This makes the public certificates low-value assets while the private certificate of the CE is a high-value asset.

The Database

The CREAM-CE uses the database for most of its operations. The database is used to hold the state of the CE holding information like job information and proxy certificates but also for communication between the components of the CE like commands that will be sent to BLAH. The database has a single user for both reading and writing, if an attacker was to gain access to the database he would gain control over most aspects of the CE. This makes the database a high value-asset.

4.3.4 Client Resource Analysis

The second resource diagram produced, Figure 4.7, focuses on the resources of the client.

When analyzing the client resources the objective is twofold, the client can both be the target and the source of an attack. First, similarly to the CE, understanding what resources on a client can be compromised and the type of damage that would result highlights the most valuable resource to target when attacking the client. Second a number of the resources located on the client will be transmitted to the CE and are potential vectors of attack.

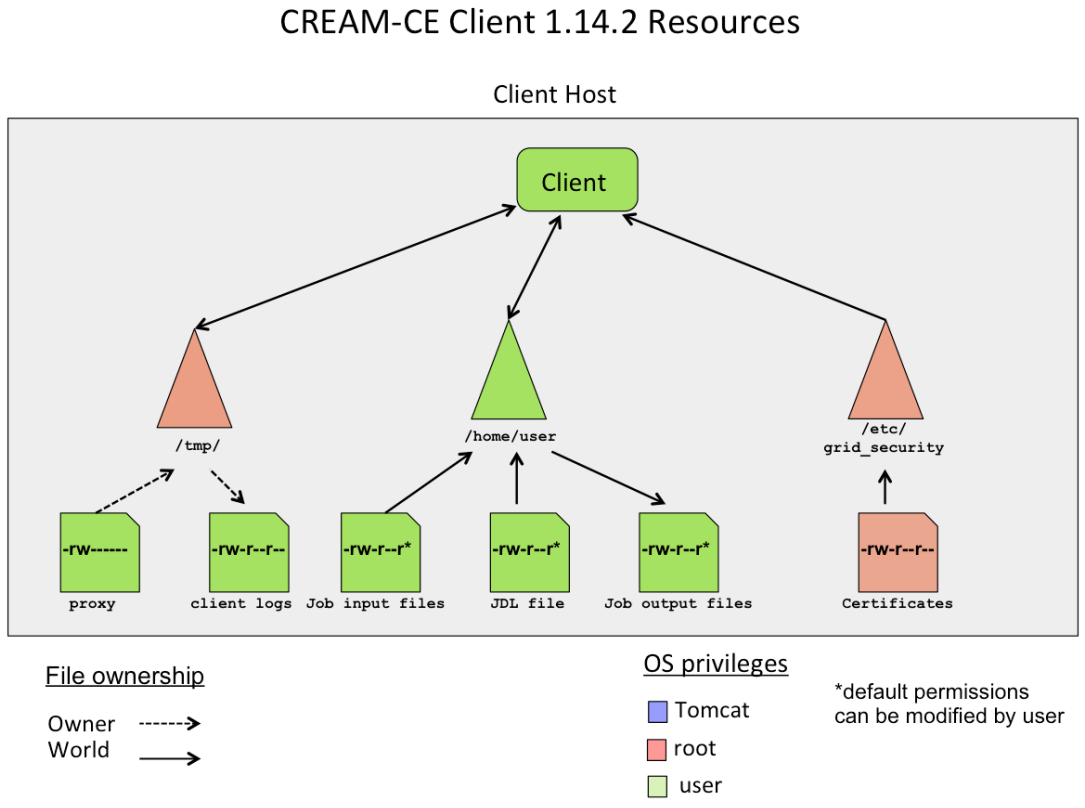


FIGURE 4.7: CREAM Client Resource Diagram

When assessing the client's security there are some specific use cases taken into account. If the client is totally isolated, having only one user that has total control over his machine and there no specific services running on the client host that would allow access to his resources then there isn't much that can be done with this information. However if there are situations where clients coexist on a single host or where access to the resources is possible then this information becomes relevant.

The CREAM client allows interactions with the CE, it does so by implementing simple commands that call one or multiple of the web services offered by the CE. The client is less complex than the CE but still uses a number of files that are required for its operations

The client has resources specific to the user like his certificate, job files and logs while certificates that identify the various hosts of the Grid are shared across a client host.

Proxy Certificate

The proxy certificate is required to authenticate with the CE, without it no commands can be executed. This certificate produced by the VOMS client is a requirement on the CREAM client host. The proxy is stored on the client host in

the temporary folder (/tmp), it is owned and can only be read by the user who has created it.

The proxy certificate is a temporary credential used to authenticate to any CE on which the user has privileges. If an attacker was to acquire it he could submit new jobs, control currently running jobs and retrieve job outputs that belong to the owner of the proxy. Although the proxy certificate mechanism mitigates the damage that can be done through its short lived nature it still holds value to an attacker if it can be recuperated within the lifetime of the proxy (by default a proxy is valid 24 hours). This makes the proxy certificate a high value asset.

Logs

Similarly to the logs contained on the CE, the client logs contain information over the transactions between the client and the CE. They hold no data of particular value to an attacker and make them low-value assets.

User Files

To submit jobs on the CE the user will have to place all the files and executables related to a job on the client and the output of jobs will be stored on the client when the user retrieves them. All these files are by default stored in the users home directory with the permissions of that user. These files are outside of the control of CREAM; they fall under the responsibility of the user and are not considered during the assessment. They are however relevant to the attacker as a vector of attack as these files will be submitted to the CE and could potentially be used to exploit the CE.

Host Certificates

Similarly to the CE the client holds a number of certificates to authenticate the hosts that he will communicate with. These include the public keys for VOMS, the CE and any other hosts configured in the client. They are stored in a special folder on the client owned by root but readable by all users. The client hosts does not require its own certificate as authentication is based on the certificates of the users.

Consulting the public certificates holds no value to the attacker, modifying them however would allow an attacker to replace a host with a machine under his control without the users knowledge. The certificates hold no value to the attacker as they are public information but if they can be modified they can lead to an attack making them high-value assets.

4.3.5 Refinements to the Resource Analysis

During this work two small additions were brought to the resource diagrams, these additions are not part of the FPVA methodology but were found to be useful to aid in the assessment work. The differences between the vanilla and modified diagrams is presented in Figure 4.8.

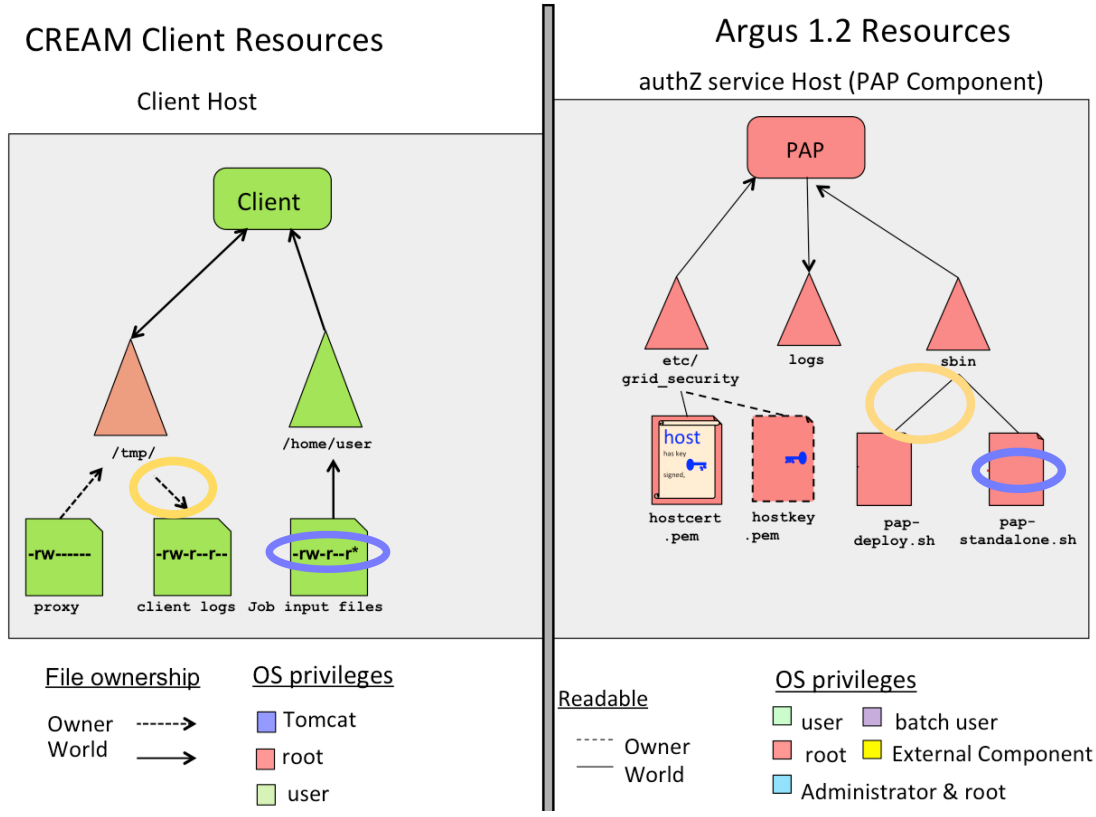


FIGURE 4.8: Resource Diagrams Differences Highlight

The first difference, highlighted in yellow, lowers the access identifier from the top level directories down to the files. This addition was made to allow a clearer representation of the access type of individual files without losing the expressivity of the diagram. This is useful when folders hold a number of files that are accessed in different ways.

The second difference, highlighted in blue, is the addition of individual file permissions to the diagram. During the assessment this information is used repeatedly when looking for the resources available to a specific user or process. This information is added at the file level where it is most relevant but the addition of the same information at the folder level is omitted as it is less pertinent and to avoid overloading the diagrams.

When looking for possible additions to the methodology it is important not to consider what was useful for this specific assessment but for assessment in general. The FPVA

methodology is meant to be generic and independent of the specific type of software under review and its platform. Also the diagrams are meant to stay concise and to easily convey information to the analyst and care has to be taken not to burden the diagrams with unnecessary details.

4.4 Privilege Analysis

The privilege analysis focuses on the users permissions that execute running processes and resource permissions. This information is gathered last and used to enrich the architectural and resource diagrams. The diagrams presented in Sections 4.2 and 4.3 are the final versions and already include this information.

First the privilege analysis adds a very important layer of information to the analysis, when elaborating attacks on a particular component knowing which assets can be accessed and in what way is critical. Where the architecture and resource diagrams highlights from which component and to what end an attack can be elaborated the privileges limit the possibilities by clearly defining whether it would be possible to interact with the resource.

Second it also serves to spot erroneous permissions which is a common type of vulnerability. The latest statistics on vulnerabilities by OWASP for 2013 [28] list security misconfiguration as the fifth most common type of vulnerability, these are simple errors that can lead to severe data exposure but are easily rooted out through the privilege analysis.

4.5 Moving Towards the Component Analysis

All the work presented up until this points is a foundation that is used to guide the analyst in his search of vulnerabilities. With all the information gathered a first guideline can be elaborated that indicate areas of the software that will have to be assessed for vulnerabilities as they handle high value assets that would make a vulnerability highly damaging to the system. This search for vulnerabilities is presented in the next chapter, the component analysis, and presents the findings that resulted using the work presented in this chapter.

Chapter 5

Component analysis

This Chapter presents the work and results of the components analysis. It starts by presenting how the work from the architectural, resources and privilege analysis was used to guide the component analysis. Followed by an explanation of the reports that were submitted to the developers. It then presents each vulnerability report produced during this work in details and concludes with an analysis of the results.

5.1 Guiding the Component Analysis

The work shown in chapter 4 produced a number of diagrams and detailed many functionalities of the CE but the motivation of the three analyses shown is to guide the component analysis. The component analysis is where vulnerabilities will be discovered which is the ultimate goal of the assessment project.

The information gathered at this point is sufficient to list some key areas of the components that will have to be verified and this will be where the component analysis will begin. This can be seen as a number of open questions that show the path to potential vulnerabilities.

- The client: Can the client be attacked directly whether from a remote host or from other users sharing the same client machine?
- Man in the Middle attacks: Can the communications between a client and the CE be intercepted? Is encryption properly implemented? Can the client be redirected to other hosts?
- SQL Injections: Are all inputs sent from the client properly sanitized in the CE?

- Jobs: Can jobs carry out malicious actions?
- Restriction Bypass: Do all the services from the API verify that they have been executed in the right order and that restrictions have been met? Are various users data isolated from each other?
- Privilege Escalation: Can a regular user use administrator functionalities? Are privileges always verified properly?
- Denial of Service: Does the CE limit the number of request per user? Can certain services be manipulated in ways that limit the responsiveness of the server?

This list only serves as a starting point for the analyst designating likely sources of vulnerabilities but does not offer any guarantees. The previous analysis revealed a number of mechanisms that are in place to prevent attacks, for example the SSL encrypted web interface should prevent communication interception, but it is paramount to answer these questions without bias and independently of how the system is supposed to function.

In certain cases it will be trivial to discard a particular type of attack due to specific implementation details or security measures. In other cases the analysis of the code will reveal other possible vulnerabilities that will have to be explored. However throughout the component analysis the analyst now has a structure to understand how each new potential attack fits within the whole architecture of the system and what could be obtained if it is successful.

5.2 Vulnerability Reports

The vulnerabilities found are presented in form of the reports sent to the developers, these reports are presented here in a redacted format meant to respect the confidentiality that binds the assessors and the software producer for the reasons enunciated in Sections 1.6 and 2.5.

The reports use a classification to rate the impact, complexity and access required of each vulnerability, this classification results from discussion between the assessment team. Each item can be classified as high, medium or low but this classification is only a recommendation and can be modified by the developers if they deem it necessary.

The impact represents the damage that can be caused by the attacker with the vulnerability and is important to the developers to prioritize the order in which vulnerabilities will be fixed and released.

The complexity describes the effort required by the attacker to successfully perform the attack, vulnerabilities can require specific conditions, equipment or knowledge to be performed. The more complex an attack the more skilled the attacker has to be and this reduces the likelihood that the vulnerability will be exploited.

The access required indicates the credentials and access required by the attacker, certain attacks require to have an account on certain hosts or systems. Requiring higher access limits the range of potential attackers.

Following are three vulnerability reports describing the findings produced during the assessment, each of these vulnerabilities are currently undergoing fixing.

5.2.1 Vulnerability Report 1: The Client Proxy Certificates Attack

The first vulnerability report sent to the developpers concerns the CREAM client. When a client host is shared amongst multiple users, certain conditions allows a malicious user to get complete control over jobs submitted by other user by tampering with certificate files. When successful this allows a malicious user to force the execution of other users jobs under the malicious user identity.

- Impact : Medium
- Complexity : Low
- Access Required : Low

The client has a very limited attack surface due to the CREAM client offering no services that are accessible from outside the host, this restricts the types of attack that can be attempted. For this reason the assessment was focused on identifying vulnerabilities than arise from multiple users sharing the same client host. While this is a specific use case it is common for groups or centers to share a unique machine that is configured to submit jobs to CE.

As explained in Chapter 4, the client uses proxy certificates to authenticate users on the CE. This certificate is the only requirement to interact with the CE and for this reason security on the client is centered around protecting the certificate. The main goal of security on a shared client is to protect the identities of each users from being stolen. This aspect of security is well implemented and as long as the general security configurations of the host is not compromised users can not access each others certificates.

The vulnerability discovered allows and attacker to replace a users proxy in order to gain control over jobs submitted as they are submitted using the attackers certificate.

If successful it allows the attacker to gain full control over all the jobs submitted with the malicious certificate.

This attack is possible due to a lack of verification on the part of the client. While every effort is made when certificates are created to ensure the security of the user, certificates have a lifetime of approximately 24 hours and during this time lapse they will be used multiple times. By not replicating the security checks performed during the creation of the certificate every time it is used an assumption is made by the system that they have not been tampered with. This opens a window of opportunity for the attacker to replace the original certificate with his own, from that point on and until a new certificate is generated the user has lost all control over the jobs he submits.

This vulnerability is based on the assumption by the developers that attackers would attempt to steal certificate files but in this case an attacker gains unauthorized access by sharing his certificate with users.

The developers have confirmed this vulnerability and are fixing it following the recommendations made in the report by using stricter restrictions on file permissions and alerting the user when these permissions are incorrect.

5.2.2 Vulnerability Report 2: SQL Injections and Denial of Service Attacks

The second vulnerability report concerns direct SQL injections from the client on the CE. A total of 17 vulnerable SQL queries were found on the CE, these findings were grouped in one report that provides examples for a selected few. These examples included listing the jobs of all users on the CE, DoS attack on the CE that requires a reboot to resume operations and a DoS that prevented any operations and persisted through reboots.

- Impact : High
- Complexity : Low
- Access Required : Low

The CREAM CE is built upon the database, it is used for all the communications between the various processes of the CE and to store the state of the system. This architecture implies that all user interactions with the CE require at least one database access. In the code there are over 300 SQL statements used for operations, within this set a subset of 17 queries that relied on improper sanitizing mechanisms were found. All of these vulnerabilities were found within the job management package of the CE.

Not all of the vulnerable queries found lead to valuable vulnerabilities, from the 17 there were 3 that lead to actual high value assets being compromised and these were used as examples for this report.

The first example used the glite-ce-job-status client program that allows a user to query the status of either a specific job or all of his jobs. By modifying the parameters passed to the CE, crafting an SQL injection, it became possible to bypass the authorization mechanisms. This resulted in the divulgation of confidential information regarding the jobs that were running on the CE. The vulnerability allows an attacker to list all the jobs of the CE since the last purge but can also be refined to show jobs that belong to specific users or that are currently in a specific state of execution. This breaks the confidentiality of the CE as this information should not be disclosed and has value to an attacker as will be shown in the third vulnerability presented in Section 5.2.3.

The second vulnerability showed how vulnerable queries could lead to DoS attacks on the CE. When assessing DoS attacks the scale and resources required for the DoS needs to be taken into account. Given unlimited resources virtually any system can be flooded with requests to the point where it becomes incapable of serving users reliably but this cannot be considered a vulnerability as it doesn't rely on a specific flaw in the component but rather a general limitation on communications and computational power.

For the purpose of the assessment a DoS is considered viable if it requires a very limited number of hosts having limited resources and relies on a specific vulnerability in the component. The DoS presented as the second example exploits the maximum number of connections that the database will accept concurrently. Typical database access require approximately a few milliseconds to complete, by crafting malicious queries that force the database connection to stay open longer, a few seconds or a few minutes, if enough queries are sent simultaneously this limit can be reached. When the limit of concurrent connections allowed is reached further requests of access are denied by the database.

Malicious queries were successfully tested that could hold a database connection open for a virtually unlimited amount of time (several days), from that point on it became trivial to saturate the CE with malicious requests that would never complete, which in turn denied access to any legitimate operation totally paralyzing the system. The resources required by an attacker to perform this attack are extremely limited, on a typical CE configuration this attack requires around 300 independent requests to be sent to the CE and each request only requires a few kilobytes of information to be sent from the client. This attack would be feasible using even the most modest of hardware configurations and connections and only requires one host.

Since the malicious database connection will never complete and therefore never be closed there is also no requirement for the attacker to continuously send request, once the CE has been flooded its operations can only be resumed when the CE host is restarted clearing the database connections. This attack exploits the SQL injection flaws found in the code but also the lack of timeout mechanisms for requests. If a timeout mechanism was employed on the CE this attack would be far less effective and require much more resources to perform as requests would have to be sent within a restricted time frame to keep enough database connections open to deny services to regular users.

The third example showed how a DoS attack could target the BLAH component of gLite by injecting a malicious query through the glite-ce-job-cancel program from a client. When a job is canceled on the CE, the cancel operation is stored within the database inside a queue and is later translated by the BLAH component into an LRMS specific command. The vulnerability uses a malicious query that can result in an invalid entry to be entered in the command queue that cannot be translated by BLAH. When this occurs the CE will keep the invalid entry inside the queue and induce a delay before invoking BLAH again, this comes from the assumption that all the entries stored within the queue table are correct and that errors from BLAH indicate an intermittent issue with the LRMS.

Once an invalid entry has been entered inside the queue table the CE will indefinitely loop trying to process the invalid command. From that point on any user request that requires interaction with the LRMS will be added to the queue table but will never be processed. This exploits combines the SQL injection with a vulnerability in how BLAH is invoked as there is no mechanism to clear invalid entries after a certain number of failures. Once the attack has occurred the CE will not be able to resume regular operations and since the invalid entry is persisted inside the database the DoS will remain in effect even after the host is restarted and persist until the database is manually cleansed.

These three examples were used in the report to show the diversity of attacks possible due to the presence of SQL injections and were those most damaging to the system. Other attacks were developed during the assessment that could retrieve any information contained within the database as well as configuration details. This included the name of the database and the username used in the connection, with this information an attacker could perform brute force attacks on the database to gain full access.

These attacks were also facilitated by the improper handling of error messages by the CE, which returns full SQL errors that occur inside the CE directly to the client. This makes injections on the CE much easier to identify and perfect as the error messages provided to the attacker divulge information on the database software used and highlights syntax errors in the crafted queries.

The second example provided to the developers lead to a discussion on whether this vulnerability was still present in the latest version of CREAM. An independent security researcher working on the BLAH component had disclosed a vulnerability three weeks prior to the publication of the vulnerability report that targeted BLAH through CREAM and allowed arbitrary code execution. Since this vulnerability had been fixed the question was whether the attack shown in the second example was still valid. Upon reviewing the changes in the latest version of gLite it was discovered that while the arbitrary code execution vulnerability of BLAH had been fixed there were no changes in CREAM that would have prevented this vulnerability. The developers had fixed the end results of the vulnerability but not the attack vector used to modify the input parameters of BLAH.

The developers have since confirmed the vulnerability is still present and are in the process of fixing the exploitable code, this fix will have the added benefit of protecting the BLAH component from further attacks by closing the attack vector.

5.2.3 Vulnerability Report 3: Arbitrary Job Cancellation Through Indirect Injection

The third vulnerability report submitted showed how a user could cancel any and all jobs on the CE regardless of his privileges. This attack allowed all jobs to be canceled at once or to target specific jobs and users. The report included a complete program that would automate this attack and use information obtained in the vulnerabilities shown in report 2 to get full control over which jobs to be cancel.

- Impact : Medium
- Complexity : Medium
- Access Required : Low

This attack focused on the lease concept used by CREAM, when a job is submitted through a scheduler a lease is generated and this lease defines the maximum amount of time a job associated to this lease can run. When the lease expires all its jobs that are still running on worker nodes are canceled as their permissions expired. This is similar to proxy certificates and is meant to increase security by limiting the time for which an authorization is valid and therefore limiting the damage when these credentials are stolen. By default the lease time is 24 hours but a user can set this to other values, the shortest possible lease time is 1 second.

The vulnerability discovered relies on how the CE handles expired leases, when this happens all the jobs that belong to a lease are listed and those still running or in queue

are canceled. This process occurs at regular intervals and the list of jobs associated with a lease are contained in the database. By maliciously crafting the lease when it is created the list of returned jobs that have to be canceled can be manipulated.

This malicious lease can be crafted with various intents, returning lists of jobs depending on various parameters. The most basic form of the attack returns all the jobs on the CE but specific jobs or all the jobs belonging to a specific user can be targeted. For this purpose an exploitation tool was written as a proof of concept that uses the information disclosed by the CE through the vulnerabilities shown in report 2 to give the attacker a simple interface that entirely automates the attack.

Once the malicious lease has been crafted it is then created on the CE with an expiration time of 10 seconds this ensures the lease will expire promptly. After it has been submitted the attacker has to wait on the automated cancellation process to execute for the exploit to be carried out. This vulnerability relies on an SQL injection but uses a different mechanism from those previously shown. In this case the injection is located inside the database and then retrieved by the vulnerable code, this is made possible by the assumption that the information contained inside the database is not malicious when in fact it can be.

This attack is indirect as the attacker will submit a lease but the actual vulnerability will only be exploited later when the automated process canceling its jobs is launched. This makes the attack much more complicated to detect and elaborate as the attacker does not receive outputs from the CE indicating whether the attack was successful or not. This is what makes this attack more complex than those reported in reports 1 and 2 as an attacker needs to have intimate knowledge of the inner working of CREAM to successfully carry it out.

This vulnerability was confirmed by the developers and is being fixed by correcting the vulnerable sections of the code.

5.3 Other Vulnerabilities and Work

In addition to the vulnerabilities presented one further vulnerability was discovered by another member of the assessment team. This was reported in the fourth vulnerability report. This vulnerability allowed the recuperation of the input and output files of any jobs submitted to the CE. This was achieved using a custom client program exploiting vulnerabilities in SQL statements to steal certificate credentials allowing access to other users sandboxes. It was classified as having medium impact but requiring high effort due the complexity of the attack and the need for a custom client.

One of the difficulties of presenting the work performed during a security assessment lies in presenting where vulnerabilities were not found. At the end of a project a report is sent to the developers which summarizes not only the vulnerabilities that were found but also what was reviewed and why the analyst is confident in his assessment. This is an important step in vulnerability reporting since this represents the largest amount of work that the analyst has performed, reviewing code and coming to the conclusion that it is safe and well implemented or that a vulnerability doesn't lead to any real damage.

At the beginning of the chapter a list of open questions that were to be answered during the analysis was presented, each one of these lead to the assessment of sections of CREAM and these were the results of this work.

- The client: Apart from the vulnerability presented in report 1 no other vulnerabilities were found. Certificates are well isolated between users and host certificates cannot be modified. On dedicated client machines there are no attack vectors allowing an attacker to communicate with CREAM.
- Man in the Middle attacks: Encryption between the client and the CE is well implemented and cannot easily be broken, this prevents eavesdropping on the information sent. Host spoofing techniques are thwarted by the authentication mechanisms using host certificates and without access to these certificates are not possible.
- SQL Injections: Although some vulnerabilities were found, the majority of SQL statements are secure, out of the 300+ SQL queries only 17 were vulnerable to injections.
- Jobs: Jobs are handled by the LRMS, in our setup this was handled by TORQUE which is a mature project and has been the subject security reviews. Jobs are well isolated from each other and data is well protected. No vulnerabilities were found in the communications between the LRMS and CREAM.
- Restriction Bypass: Each web service request passes through the authentication framework before being handled, the authentication mechanism that works in combination with the VOMS server is well implemented and prevented any unauthorized access to services and could no be bypassed.
- Privilege Escalation: Local administration permissions were secure and no way was found to escalate a user to an administrator on the system.
- Denial of Service: DoS attacks were found due to SQL vulnerabilities on the CE, as a results new security mechanisms were recommended to the developers that

limit the number of requests a user can perform concurrently to mitigate future attacks.

This list is not exhaustive but gives an idea of the various areas that were assessed. Certain types of vulnerabilities presented in Section 2.2.1 were not assessed due to the technology used to implement CREAM. For example buffer overflows which are extremely pertinent when assessing code written in C/C++ is not relevant to the assessment of a project like CREAM implemented mostly in Java.

In general the code was of good quality and many security measures had been used to prevent attacks. The consistent use of prepared statements and authorization, the use of standard Grid toolkits like GLOBUS and proper use of users and privileges on the Linux operating system made the system robust and more difficult to attack.

Chapter 6

Conclusions and Open Lines

6.1 Conclusions

The work of this Master thesis revolved around the vulnerability assessment of the CREAM middleware component, this was successfully achieved by following to the FPVA methodology to guide the assessment.

The primary objective was to find security vulnerabilities, to this end a number of sequential steps detailed within FPVA were followed. This started with the architectural, resource and privilege analysis. Each of these steps provided specific information over various areas of the software relating to its security and a general deep understanding of the inner workings of CREAM. This work was detailed in Chapter 4 where the value of each analysis in regards to the assessment was shown. This information was combined to guide the later component analysis which focused on the discovery of vulnerabilities.

The component analysis revealed a number of serious vulnerabilities within the software component, these vulnerabilities were documented and reported to the developers. Each of the vulnerabilities reported has been confirmed by the development team and are currently undergoing correction for the next candidate release. At the end of the assessment a list of the various aspects of the software that was reviewed was compiled indicating where vulnerabilities were and were not encountered to show the work that was performed and justifying the confidence in these findings. This work was presented in Chapter 5 and showed how the component analysis was guided by the work of the previous steps and the results produced.

Having thoroughly assessed CREAM the primary objective was attained and as a result the security of software has increased.

The secondary objective of this work was to gain experience with the FPVA methodology and possibly suggest refinements that will benefit further projects. Having worked thoroughly with every aspect of the methodology has allowed to understand all the steps that it contains and their purpose. This allowed small additions to be proposed to the methodology in the form of refinements to the resource diagrams that provide more precise information over specific files and their permissions.

The assessment project of CREAM has now successfully reached its end, the objectives stated were accomplished and new assessment projects will now be undertaken.

6.2 Open Lines

The first open line to explore as a continuation of this work is the assessment of further middleware components. Building upon the experience gained with the FPVA methodology and Grid software to continue identifying new vulnerabilities and work to refine the methodology in a long term effort to improve security in the Grid environment. There are still many software components including amongst gLite that have not been assessed for security and could benefit from this work.

The second line of work is to apply the knowledge gained by having applied the entire methodology by hand to elaborate new automated analysis tools. While analyst-centric assessment is currently the most successful path for large component assessment there is a possibility to build tools to aid the analyst in his assessment and reduce the time required to perform the analysis. More precisely not tools to automate the discovery of vulnerabilities but ones to automate the elaboration of the diagrams on which the analyst relies for his analysis. This has been explored previously in [29] and can be built upon to better automate specific diagrams particularly the resource diagrams which lends itself well to automated analysis.

This would improve on one of the key drawbacks of the manual process by reducing the time required to perform the analysis. By producing exhaustive information over all the resources accessed by system these tools could then be used to quickly elaborate the full diagrams allowing the analyst to spend more time focusing on the component analysis and the discovery of vulnerabilities.

Bibliography

[1] A. James Kupsch, P. Barton Miller, Cesar Eduardo, and Heymann Elisa. "first principles vulnerability assessment" (extended version). Technical report, University of Wisconsin, Universitat Autònoma de Barcelona, Sept. 2009.

[2] M. Brugnoli. *Decentralized Scheduling on Grid Environments*. PhD thesis, Univ. Autònoma de Barcelona, Spain, 2010.

[3] P Andreetto, SA Borgia, A Dorigo, A Gianelle, M Marzolla, M Mordacchini, M Sgaravatto, F Dvorák, D Kouril, A Krenek, et al. Cream: a simple, grid-accessible, job management system for local computational resources. *CHEP 2006, Mumbay, India*, 2006.

[4] Basie Von Solms and Rossouw von Solms. From information security to... business security? *Computers & Security*, 24(4):271–273, 2005.

[5] White House. Remarks by the president on securing our nation's cyber infrastructure, May 2009. URL http://www.whitehouse.gov/the_press_office/Remarks-by-the-President-on-Securing-Our-Nations-Cyber-Infrastructure.

[6] James Andrew Lewis. Raising the bar for cybersecurity, Feb 2013. URL http://csis.org/publication/raising-bar-cybersecurity.

[7] CERN. last checked july 2013. URL http://home.web.cern.ch/.

[8] NASA. last checked july 2013. URL http://www.nasa.gov/.

[9] Emmanuel E Udoh. The adoption of grid computing technology by organizations: A quantitative study using technology acceptance model. *ProQuest LLC*, page 124, 2010.

[10] J.A. Kupsch and B.P. Miller. Manual vs. automated vulnerability assessment: A case study. In *First International Workshop on Managing Insider Security Threats (MIST)*, pages 83–97, 2009.

[11] James A. Kupsch, Barton P. Miller, Elisa Heymann, and Eduardo César. First principles vulnerability assessment. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, pages 87–92, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0089-6. doi: 10.1145/1866835.1866852. URL http://doi.acm.org/10.1145/1866835.1866852.

[12] Frank Swiderski and Window Snyder. *Threat modeling*. O'Reilly Media, Inc., 2009.

[13] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.

[14] Jian Jim Hu, Qiaoyan Wen, and Ai Fen Sui. An efficient code audit method for accurately detecting security vulnerabilities in source codes. In *Communication Technology (ICCT), 2011 IEEE 13th International Conference on*, pages 698 –702, sept. 2011. doi: 10.1109/ICCT.2011.6157966.

[15] Mark Dowd, John McDonald, and Justin Schuh. *The art of software security assessment: Identifying and preventing software vulnerabilities*. Addison-Wesley Professional, 2006.

[16] CVE. Common vulnerabilty exposure website, last checked july 2013. URL http://cve.mitre.org/.

[17] CEW. last checked july 2013. URL http://cwe.mitre.org/.

[18] Michael Howard, Jon Pincus, and Jeannette M Wing. *Measuring relative attack surfaces*. Springer, 2005.

[19] OWASP Testing Framework. last checked july 2013. URL https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf.

[20] Adam Stone. Software flaws, to tell or not to tell? *Software, IEEE*, 20(1):70–73, 2003.

[21] Ian Foster and Carl Kesselman, editors. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-475-8.

[22] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001. ISSN 1094-3420. doi: 10.1177/109434200101500302. URL http://dx.doi.org/10.1177/109434200101500302.

[23] Ian Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002. URL http://dlib.cs.odu.edu/WhatIsTheGrid.pdf.

[24] gLite. last checked july 2013. URL http://glite.cern.ch.

[25] EGEE. last checked july 2013. URL http://www.eu-egee.org/.

[26] EMI. European middleware initiative, last checked july 2013. URL http://www.eu-emi.eu.

[27] CONDOR. last checked july 2013. URL http://research.cs.wisc.edu/htcondor/index.html.

[28] OWASP. Owasp list of top ten most common vulnerabilties in 2013. last checked july 2013. URL https://www.owasp.org/index.php/Top_10_2013-Top_10.

[29] Wenbin Fang, Barton P Miller, and James A Kupsch. Automated tracing and visualization of software security structure and properties. In *Proceedings of the Ninth International Symposium on Visualization for Cyber Security*, pages 9–16. ACM, 2012.