

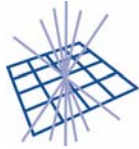
# GridPP

## GRID SECURITY VULNERABILITY DETECTION AND REDUCTION

Document Identifier	<b>XXXXX</b>
Date	<b>16/02/2005</b>
Version	<b>0.0</b>
Document status:	<b>DRAFT</b>
Author	<b>Linda Cornwall, CCLRC (RAL)</b>
Document link:	

---

**Abstract:** This document describes what we mean by vulnerabilities, describes which principals should be protected from such vulnerabilities, and describes what these principals need protecting from. It contains strategies for detecting and reducing vulnerabilities, including checklists to reduce the likelihood of there being vulnerabilities in both the middleware and deployment, logging of known vulnerabilities, and ‘anti use cases’ actions which should not be allowed by the system.



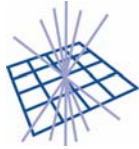
XXX

### Document Log

Issue	Date	Comment	Author/Partner
0-0	16 <sup>th</sup> Feb 2004	First draft	Linda Cornwall

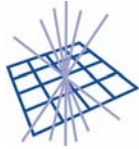
### Document Change Record

Issue	Item	Reason for Change



## CONTENT

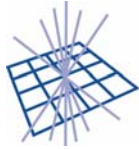
<b>1. INTRODUCTION.....</b>	<b>6</b>
1.1. PURPOSE .....	6
1.2. APPLICATION AREA .....	6
1.3. REFERENCES .....	7
1.4. DOCUMENT EVOLUTION PROCEDURE.....	7
1.5. TERMINOLOGY.....	8
<b>2. SECURITY VULNERABILITY DESCRIPTION.....</b>	<b>9</b>
2.1. WHAT SHOULD WE BE PROTECTING? .....	9
2.1.1. <i>Protecting the system</i> .....	9
2.1.2. <i>Protecting data and information</i> .....	9
2.1.3. <i>Protecting other systems from our Grid system</i> .....	9
2.1.4. <i>Protecting the user from the system</i> .....	9
2.2. WHAT ARE WE PROTECTING FROM? .....	10
2.2.1. <i>An Untrustworthy system</i> .....	10
2.2.2. <i>An untrustworthy administrator</i> .....	10
2.2.3. <i>An untrustworthy user</i> .....	10
2.2.4. <i>An untrustworthy machine</i> .....	10
2.2.5. <i>A hacker</i> .....	10
2.2.6. <i>Untrustworthy software</i> .....	10
2.3. WAYS OF PROTECTING FROM VULNERABILITIES .....	10
2.3.1. <i>Middleware</i> .....	10
2.3.1.2. <i>General Grid Middleware</i> .....	11
2.3.2. <i>Configuration and Deployment</i> .....	11
2.4. GOOD PRACTICES .....	12
2.4.1. <i>Checking principals</i> .....	12
2.4.2. <i>Care over combination of roles</i> .....	12
2.4.3. <i>CA and VO procedures</i> .....	12
2.4.4. <i>Management of Secrets</i> .....	12
2.5. MECHANISMS FOR PREVENTING OR DETECTING MISUSE .....	12
2.5.1. <i>Logging and processing of logs</i> .....	12
2.5.2. <i>Usage tools</i> .....	12
2.5.3. <i>Service version checking</i> .....	12
2.5.4. <i>Incident Response</i> .....	13
2.6. VULNERABILITY DETECTION THROUGH USE OF CHECKLISTS .....	13
2.6.1. <i>Principle</i> .....	13
2.6.2. <i>What does satisfying each check give us?</i> .....	13
2.7. DISCOVERY OR AWARENESS OF SPECIFIC VULNERABILITIES.....	13
2.8. RELATION TO RISK ASSESSMENT.....	14
<b>3. MIDDLEWARE VULNERABILITY CHECKLIST.....</b>	<b>15</b>
3.1. DESIGN .....	15
3.2. CODING PRACTICE.....	16
3.3. COMMUNICATIONS .....	17
3.4. INPUT CHECKING .....	18
3.5. BUFFER OVERFLOWS.....	20
3.6. UNWELCOME CODE BEHAVIOUR .....	21
3.7. CONFIDENTIAL DATA.....	23
3.8. BACKDOORS .....	24
3.9. MIDDLEWARE ACCESS TO THE SYSTEM.....	24
3.10. EXPOSING INFORMATION ABOUT THE SYSTEM .....	25
3.11. FILE HANDLING .....	25
3.12. AUTHORIZATION.....	27



---

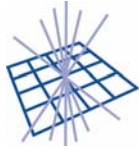
3.13. USAGE .....	29
3.14. DEPENDENCIES .....	30
3.15. ERROR HANDLING.....	30
3.16. FAILURE .....	32
3.17. LOGGING MECHANISMS .....	32
3.18. SPECIFIC CHECKS ON SECURITY MIDDLEWARE.....	33
3.19. INTEGRATION OF SECURITY MIDDLEWARE WITH OTHER MIDDLEWARE .....	34
3.20. TESTING.....	34
3.21. OTHER MIDDLEWARE CHECKS.....	37
3.22. DOCUMENTATION .....	38
<b>4. DEPLOYMENT AND CONFIGURATION CHECKLIST .....</b>	<b>39</b>
4.1. SOFTWARE VERSIONS .....	39
4.2. DEPLOYMENT METHODS .....	39
4.3. DEPLOYMENT AUTHENTICATION AND AUTHORIZATION .....	40
4.4. FIREWALLS .....	41
4.5. USE OF CREDENTIALS .....	42
4.6. HANDLING CONFIDENTIAL DATA.....	44
4.7. LOGGING STRATEGY .....	45
4.8. STAFF .....	46
4.9. INCIDENT RESPONSE AND DETECTION .....	46
<b>5. SPECIFIC KNOWN VULNERABILITY LOGGING .....</b>	<b>48</b>
5.1. WHAT TO LOG.....	48
5.1.1. Location.....	48
5.1.2. Vulnerability.....	48
5.1.3. Exploitation .....	48
5.1.4. Analysis .....	48
5.1.5. Risk analysis.....	48
5.1.6. Proposed solution.....	48
5.1.7. Bugs submitted .....	48
5.1.8. Checklist reference.....	48
5.1.9. Checklist proposal.....	48
5.2. RESTRICT ACCESS TO LOG .....	48
<b>6. VULNERABILITY ‘USE CASES’ .....</b>	<b>49</b>
6.1. USE CASE CHECKING.....	49
6.1.1. How would you achieve it?.....	49
6.1.2. What Specific Mechanism(s) prevent this?.....	49
6.1.3. Try it - possibly.....	49
6.1.4. Is the attempt successful? .....	49
6.1.5. Is the attempt logged? .....	49
6.1.6. Is the attempt detected? .....	49
6.1.7. How is the attempt handled? .....	49
6.2. USE CASE EXAMPLES .....	49
6.2.1. Storage of illegal material for distribution.....	49
6.2.2. Certificate Cracking .....	49
6.2.3. Credentials theft .....	49
6.2.4. Launch of a denial of service attack on the Grid.....	50
6.2.5. Launch of a denial of service attack from the Grid.....	50
6.2.6. Theft of confidential information.....	50
6.2.7. Addition of an un-trusted host.....	50
6.3. DISCUSSION .....	50
<b>7. CHECKLIST TABLES .....</b>	<b>51</b>

---



---

7.1. MIDDLEWARE CHECKLIST .....	51
7.2. DEPLOYMENT CHECKLIST .....	55



## 1. INTRODUCTION

### 1.1. PURPOSE

Many people who develop or deploy grids are aware that there may be some vulnerability in the currently deployed grid systems. The purpose of the document is to describe some strategies for detecting and reducing Security vulnerabilities on the Grid, both associated with Grid middleware, and with grid deployment.

Security requirements in a Grid environment were looked at as part of the DataGrid project (D7.5) Ref – [1], a design was presented (D7.6) Ref – [2], various security middleware utilities were written and an assessment of progress towards satisfying these requirements was produced (D7.7) Ref – [3]. Work is continuing in this area in the EGEE project, requirements have been revised Ref - [4] and an architectural design Ref – [5] along with various other documents have been produced.

However, these all look mainly at security functionality, particularly the security functionality needed to satisfy the Authentication and Authorization Requirements. Little has been done to ask, let alone answer the question ‘is the grid secure’.

Vulnerability can be considered to be where a system may behave in a way that is not intended, with the possibility of causing damage to the system itself, damage or unauthorized access to data or information on the system, or damage to a third party such as a user or external system.

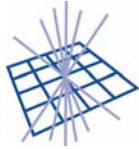
Here we attempt to look at some of the areas where we should guard against vulnerabilities. We do not consider general security vulnerabilities related to non grid specific software – these are addressed in many other places. We consider two main areas:--

- 1) Vulnerabilities associated with a Grid environment
- 2) Vulnerabilities associated with Grid Specific Middleware.

In chapter 2 we describe some of the problems that may result from a vulnerability, who we are protecting from, some general information on vulnerability protection. Chapter 3 contains a checklist of things to take care of to prevent or reduce middleware vulnerability. Chapter 4 contains a checklist for the deployment of a grid system. Chapter 5 suggests how to describe and handle specific vulnerabilities which are found in the middleware or the deployment. Chapter 6 lists some ‘anti use cases’, things that must be prevented. Chapter 7 presents tables for checking the middleware and deployment described in chapters 3 and 4.

### 1.2. APPLICATION AREA

This is lead by the GridPP project, and is relevant to all grid middleware and deployment.

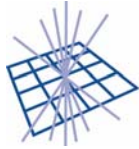


### 1.3. REFERENCES

[R1]	DataGrid Security Requirements and Testbed-1 Security Implementation, D7.5, <a href="https://edms.cern.ch/document/340234/">https://edms.cern.ch/document/340234/</a>
[R2]	DataGrid Security Design, D7.6 <a href="https://edms.cern.ch/document/344562/">https://edms.cern.ch/document/344562/</a>
[R3]	DataGrid Final Security report, D7.7 <a href="https://edms.cern.ch/document/414762/">https://edms.cern.ch/document/414762/</a>
[R4]	EGEE JRA3 (Security) User Requirements Survey <a href="https://edms.cern.ch/document/485295/">https://edms.cern.ch/document/485295/</a>
[R5]	EGEE Global Security Architecture DJRA3.1 <a href="https://edms.cern.ch/document/487004/">https://edms.cern.ch/document/487004/</a>
[R6]	The European Policy Management Authority for Grids (PMA) <a href="http://www.eugridpma.org/">http://www.eugridpma.org/</a>
[R7]	Guide to LCG Application, Middleware & Network Security Ian Neilson. CERN edms document 452128 <a href="https://edms.cern.ch/document/452128/">https://edms.cern.ch/document/452128/</a>
[R8]	Secure Coding – Principles and Practice. Mark G. Graff and Kenneth R.van Wyk 2003 O'Reilly & Associates Inc. ISBN 0-596-00242-4
[R9]	Secure Programming Cookbook for C and C++ John Viega & Matt Messier. 2003 O'Reilly & Associates Inc. ISBN 0-596-00394-3
[R10]	EGEE Activity Quality Plan – JRA1 <a href="https://edms.cern.ch/document/475557/">https://edms.cern.ch/document/475557/</a>
[R11]	LCG incident response <a href="https://edms.cern.ch/document/428035/">https://edms.cern.ch/document/428035/</a>

### 1.4. DOCUMENT EVOLUTION PROCEDURE

This document is designed to be a tool to help reduce the number of vulnerabilities in both the middleware and the deployment. It is a technical document, which is not a formal delivery, which is expected to evolve over time as we better understand how to detect and handle Grid security Vulnerabilities. The editor will amend the document to improve it taking input from the GridPP project as well as the various Grid security interest groups, such as the Joint Security Policy Group (JSPG), EGEE Middleware Security Working Group, and the EGEE JRA3 Security Workpackage.

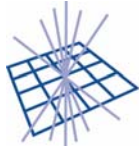


## 1.5. TERMINOLOGY

### Glossary

EGEE	Enabling Grids for E-Science in Europe (EU funded project) <a href="http://egee-intranet.web.cern.ch/egee-intranet/gateway.html">http://egee-intranet.web.cern.ch/egee-intranet/gateway.html</a>
JRA3	The EGEE Joint Research Area 3, Security
JSPG	Joint Security Policy Group (Security Group for the LCG project)
LCG	Large Hadron Collider Grid Project
MWSG	EGEE Middleware Security Group

### Definitions

---

## 2. SECURITY VULNERABILITY DESCRIPTION

### 2.1. WHAT SHOULD WE BE PROTECTING?

Here we briefly describe the main areas that need protecting.

#### 2.1.1. Protecting the system

This is about ensuring the system is available to those who have legitimate access. This tends to be the first thing people consider, it is what conventional computer security is generally about. This is particularly true of systems run by large organisations, they wish to protect the system from:--

1. Intrusion by an unauthorized person. In particular a person who may have malicious intent or wishes to use facilities for unlawful activities.
2. From someone with legitimate access gaining access beyond their right.
3. From someone with legitimate access doing something which causes damage to the system, whether intentionally or not.

#### 2.1.2. Protecting data and information

This is about ensuring that material cannot be accessed or modified beyond the rights of any principal. It is necessary to be sure that data or information belonging to a VO or individual user is secure and correct. Functionally, in the past this has been considered by authorizing the service or use to handle such information, and encryption for sensitive information. We need to ensure this is effective and there are no ways that such information can be obtained by those who should not obtain it. This could include looking at how we could protect such information e.g. from being sold by an individual who has legitimate access.

#### 2.1.3. Protecting other systems from our Grid system

A Grid contains a lot of resources that could be used to enable another system to be attacked. There are two main areas for this

1. Use to initiate an attack on other services
2. Use to crack other organisations security.

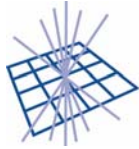
If someone has access to the Grid, whether legitimate or not, it is important to ensure they cannot use the Grid to launch e.g. a denial of service attack.

If the Grid were to be used to e.g. crack a banks' certificate to carry out a major fraud, we would be in serious trouble. This could include a legitimate user, who uses our system to carry out a fraud and then vanishes!

These are dangers we should take seriously, most other problems at present would only affect our own Grid and data. As soon as a grid is used as a tool to commit a crime we could be at best considered irresponsible for setting it up with inadequate protection. At worst...

#### 2.1.4. Protecting the user from the system

The user should be protected from being held responsible for something they either did not do or did not intend to do. While some of this is the user's responsibility, tools and recommendations need to be in place so that the user can use the system with confidence that they will neither unintentionally run up a large bill nor be held responsible if there is a breach of security which appears to have been done in their name.



## **2.2. WHAT ARE WE PROTECTING FROM?**

This describes the main untrustworthy entities we need to protect from.

### **2.2.1. An Untrustworthy system**

The software and deployment need to be such that they work reliably, don't contain flaws that allow any principal to carry out an action they should not, do not fail in such a way that provides unauthorized access to resources or data. The system needs to be such that it cannot easily be commanded in such a way that one person can deny access to another. In particular the system should not copy information to untrustworthy principals.

### **2.2.2. An untrustworthy administrator**

However good our technology is, an administrator of a CA, or a VO, or a Site could cause a lot of damage. Also, they may be able to steal and sell information, or be paid to destroy information.

### **2.2.3. An untrustworthy user**

Even though we may have granted access to a user, we cannot assume all users will use the system in the way it was intended. For example, a legitimate user may have access to a confidential database, and may be bribed to reveal information.

### **2.2.4. An untrustworthy machine**

It is important to protect against an untrustworthy machine added to the Grid.

A machine may be added to the Grid in order to attempt to gain access to information which is confidential.

### **2.2.5. A hacker**

Someone may intend to break into the system, whether to cause damage to the system, use the system to attack others, use the resources for their own (possibly unlawful) purposes) or steal information. We need to ensure the software does not contain weaknesses that allow such access.

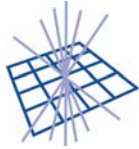
### **2.2.6. Untrustworthy software**

This is where software behaves in a way that wasn't intended, and it could be that no-one does anything with malicious intent yet the system becomes vulnerable. Such vulnerability could, of course, be exploited by a hacker.

## **2.3. WAYS OF PROTECTING FROM VULNERABILITIES**

The general criteria for dealing with vulnerabilities are: protect, deter, detect and react. Protect means making the system as resistant to attack as possible. Deter means having sanctions we can apply to those who are caught doing something they should not. Detect means detecting when anything occurs that should not. React means to deal with the situation, including fixing software vulnerabilities as they occur. We should aim to protect as much as we can, and remove as many vulnerabilities from the system as possible. Where this is not possible, we should aim to detect problems and react as quickly as possible. I

### **2.3.1. Middleware**



This is where we ask the question, does the Grid Middleware contain faults which allow non-compliance with the security requirements? Does the technology allow the software to be used in a way that exposes vulnerabilities? Are there flaws in the design, which allow the components to be used in a way that is not intended? Is the implementation such that it allows certain input to cause the system to crash or allow unintentional behaviour? It is important when designing and implementing any middleware that we consider unfriendly input and attempts to cause damage by overloading.

### **2.3.1.1. Security Components**

Security components are designed to allow access after appropriate authentication and authorization has taken place, but deny other access.

It is important to ensure that these components cannot be forced to fail in such a way that unauthorized access can be allowed, that if they do fail they fail in a way that prevents access, and they cannot be commanded in such a way that causes access beyond a principals rights to take place.

### **2.3.1.2. General Grid Middleware**

It is important that any Grid middleware is robust, and checks all input for valid input.

It is important that no input can cause a problem. It is important that there are no back doors.

### **2.3.1.3. Integration of Security components with Grid Middleware**

Even if security middleware has been tested and is reliable it is important to ensure that the integration with the various other Grid Middleware components does not present security flaws. It is important that it is not integrated in such a way that allows the security middleware to be by-passed.

### **2.3.1.4. Principle of least privilege**

Software should run at the lowest level of privilege possible. If software runs as root, if there is a flaw that can be exploited, the potential for damage is much higher. This is a matter both for the design and implementation of the software itself, and the deployment.

### **2.3.1.5. Connections**

Every service that can be connected to presents the possibility of the existence of a security vulnerability, this includes connections to services that are designed to only be services to other Grid components, including other parts of the same service.

For example, a service may be written where the expected client is another instance of the same service or at least written by the authors of the service. This cannot be assumed.

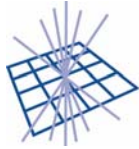
## **2.3.2. Configuration and Deployment**

### **2.3.2.1. Reliable configuration**

It is important that the software is configured securely. Tools need to be made available to allow sysadmins to configure the software reliably.

### **2.3.2.2. Level of logging**

There needs to be a balance between sufficient logging to enable adequate tracing of what people can do, and too much logging, which may cause the disk to become full, and allow a denial of service attack.



---

## **2.4. GOOD PRACTICES**

### **2.4.1. Checking principals**

Before allowing any principal to have a role it is important to check that they are suitable. It is important to check that CA admins, VO admins, and system admins can be trusted and are suitable for the role. It is important to check the identity and suitability of sites and users, before allowing them credentials that allow access to the system, especially those credentials which allow access to sensitive data or large quantities of resources.

### **2.4.2. Care over combination of roles**

Sometimes a combination of roles could lead to a principal having freedom to do a lot of damage. For example, if someone was both a CA admin and a VO admin then he could issue a false identity and VO credentials to himself and gain access to resources beyond what is intended.

### **2.4.3. CA and VO procedures**

Much has been discussed about the CA procedures, and ensuring that the identity of a user is checked. Ref – [6]

### **2.4.4. Management of Secrets**

Are security secrets such as passwords and private keys properly protected? Are well known user name and passwords used? This applies to both services themselves and to databases behind the services as well as users. In particular, is the host certificate secure?

## **2.5. MECHANISMS FOR PREVENTING OR DETECTING MISUSE**

It is important to have in place mechanisms for preventing or at least detecting misuse.

### **2.5.1. Logging and processing of logs**

It is important that adequate logging and processing of logs is carried out, as logging usage can help detect misuse, both by a hacker and by a legitimate user.

For example, if a legitimate user has access to confidential data, and they access beyond what is needed to do their job, then logging access and processing of such logs may allow this to be detected.

### **2.5.2. Usage tools**

If a user attempts to use the Grid to carry out processing that is not legitimate then we may be possible to prevent such usage. One way may be to only allow software signed by a VO to be used. If users wish to use their own software, maybe they should register a copy of the source code with the VO, and this should be built by the VO. This does not prevent usage for illegal purposes, but it could make such detectable. Another is to carefully control quota, and people who have a large quota which could be used to e.g. crack certificates, can only use software that is checked out by the VO.

### **2.5.3. Service version checking**

Can we be sure that the software used in the services is the intended version? Could it have been replaced e.g. by an unreliable sysadmin or hacker? A mechanism for ensuring unfriendly software is not on the system should be in place.

#### **2.5.4. Incident Response**

It is important to have an incident response plan in place, which allows fast and efficient dealing with any breach of security. The LCG project has such a plan, described in the LCG Incident response document. Ref – [11].

### **2.6. VULNERABILITY DETECTION THROUGH USE OF CHECKLISTS**

#### **2.6.1. Principle**

Much of the rest of this document consists of checklists for middleware and deployment. Vulnerability avoidance is a huge subject, and there are many books on the subject, e.g Ref [8]. We do not attempt to repeat everything that can possibly be checked from all sources, but provide a checklist of some of the more important things that we should check. Some vulnerabilities probably mean new middleware functionality is needed, or a new tool is needed to guard against such vulnerabilities. Some are just a matter of fixing a bug or fixing software to work in other ways. Some vulnerability is a matter of configuration and deployment.

#### **2.6.2. What does satisfying each check give us?**

Some checks are more specific than others. Some of the checks are to do with avoiding specific vulnerabilities that are well defined. Some of the checks are more to do with good practice to minimize the risk of introducing some sort of vulnerability, or to minimize the damage that can be done. Some of the checks are for the existence of tools that prevent or detect misuse. We indicate what we are checking for when by carrying out each check.

SV – Specific Vulnerability

VRR – Vulnerability Risk Reduction – a practice that if carried out is likely to reduce the risk of there being vulnerabilities.

VIR – Vulnerability Impact Reduction – a practice that if carried out is likely to reduce the impact of any vulnerability present.

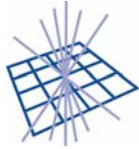
ADL – Activity Detection and Logging – this allows the detection of any exploitation of a vulnerability or an abuse of the system.

This should help avoid or reduce the risk from vulnerabilities, however we do not claim it is complete, and we do not claim that there is no way round the checklists – they are simply an aid to help reduce vulnerabilities.

### **2.7. DISCOVERY OR AWARENESS OF SPECIFIC VULNERABILITIES**

Some implementers and system administrators are well aware of the existence of various vulnerabilities, and have mentioned them in various discussions both person to person and via e-mail. Such vulnerability should obviously be fixed whenever possible. It should at least be noted in a systematic manner, and if it cannot be fixed immediately the risks associated with it should be assessed. Any note of specific vulnerabilities must be kept private, and access must be carefully controlled. Details are given in chapter 5.

When a specific vulnerability has been noted, those analysing it should consider if anything in one of the checklists used to minimize vulnerability would have picked it up or prevented it. If there isn't, then consider proposing an appropriate check. This may help prevent or detect other vulnerability in the future.



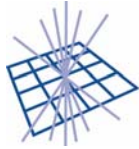
Care should also be taken concerning the publication of bugs, as these may represent vulnerability. The practice of logging bugs on a world readable database should be re-assessed.

Some Vulnerability may be the result of the combination of the way a multiple of packages or tools work, such possibilities need to be considered.

It is worth considering who could take advantage of a vulnerability. Someone without any credentials, such as a hacker? An authenticated user? A user with some Authorization? A system administrator?

## **2.8. RELATION TO RISK ASSESSMENT**

The finding and elimination of vulnerabilities is related to risk assessment, it is about reducing certain risks. But it is not a risk assessment in itself. Anyone who finds that a vulnerability is present, whether through discovery by the application of a checklist, or just being aware of it may want to consider what risk that poses, both in terms of the likelihood of exploitation and the impact if it is exploited.



### **3. MIDDLEWARE VULNERABILITY CHECKLIST**

#### **3.1. DESIGN**

This is where we look at ‘does the design itself logically introduce vulnerabilities’ and ‘does the design mean that if there is vulnerability can a lot of damage occur very rapidly?’

##### **DES-01**

###### **Carrying out a design process**

If the design is carried out using good practices, then it is less likely that vulnerabilities will be introduced. When carrying out the design it is important to ensure the design is clear and well documented. It is important to document all connections, especially those external to the component, or between different parts or instances of the middleware component.

Check that there is a good design in place, and that it describes the components and all interconnections.

VRR

##### **DES-02**

###### **Design for robustness**

Carry out a design such that if one part of the installation fails it does not cause the whole system to fail. Design such that if it is impossible to reach one part of the network, or if one part of the system is not working, compromised or incorrectly configured it does not mean the whole system fails.

Check this has been considered.

VIR

##### **DES-03**

###### **Principle of least privilege**

When carrying out the design it is important to consider what privileges the executing software should have, and make design decisions which limit these. This is also a deployment issue, the software should not run with privileges it does not need. In particular, not all software should run as root.

Check the privileges the software needs. Can these be reduced? Are they well documented? Is the default configuration just what it needs?

VIR

##### **DES-04**

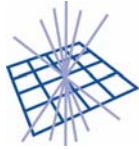
###### **Formal Analysis**

One method carried out by computer scientists is formal analysis of the design. This looks for logical vulnerabilities from a very neatly specified design.

This could be carried out in the future if appropriate expertise were to be made available.

SV

##### **DES-05**



### **Informal Analysis**

This is where we go on a thought experiment ‘how do we get round this’, or ‘can this system be used to do x and y’. This is a useful experiment that we could carry out.

SV

### **DES-06**

#### **Think about risks**

Ask the question: are there any particular risks associated with this particular piece of middleware? If there are, think about whether they are handled.

VRR

### **DES-07**

#### **Discovering loopholes**

This is where as a result of testing or usage specific loopholes are discovered.

Document loopholes – but not for public consumption! Ensure they are fixed at the earliest opportunity. More on this is in chapter 5.

Check that known loopholes/vulnerabilities are handled.

SV

## **3.2. CODING PRACTICE**

There is no coding practice that guarantees vulnerability or bug free code. But good coding practice reduces the chances of vulnerabilities being unwittingly introduced.

### **CP-01**

#### **Code Re-use**

Re-use well tested code where possible. Software which has been subjected to extended analysis and use is less likely to have exploitable security holes than new software.

Check that when appropriate existing software is re-used and integrated with new middleware.

Ref – [7]

VRR

### **CP-02**

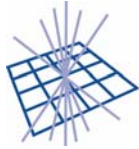
#### **Clarity**

Code for clarity first and optimize during testing if necessary.

Complexity is the enemy of security. Well-structured, clear code allows for better understanding of intent and appropriate algorithms. It also reduces the likelihood of the introduction of errors which may lead to security vulnerabilities.

Check that coding standards are in place, and that the code is clear and readable.

Ref – [7]



VRR.

### **CP-03**

#### **Check good quality procedures have been carried out.**

Check what procedures are in place

VRR

### **CP-04**

#### **Code quality indicators**

Use of quality code indicators can help indicate for example how stable the developed code is.

Check any quality indicators in place, if they are then look at their values. E.g. EGEE development specifies some quality indicators in Ref – [10].

VRR

## **3.3. COMMUNICATIONS**

### **COM-01**

#### **Use established protocols**

Do not invent new protocols when existing ones can be used. It is often tempting to assume that for performance or other reasons an application requires a new or modified protocol to be developed. Experience shows that this is usually not the case and existing standards which have been open to study and use over an extended period of time avoid the many subtle failures that can be induced in the development of security protocols. Ref –[7]

Check that established protocols are used for all connections.

VRR

### **COM-02**

#### **Use well tested software**

In particular, for communications, try to avoid writing your own software if at all possible. If appropriate established software is available then use this.

Check that well established software is used. If it is not (e.g. the appropriate software was not available in the appropriate language) check that it has been very well tested.

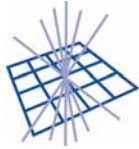
VRR

### **COM-03**

#### **Authentication**

It must be possible to authenticate and check the integrity of all network communications. (e.g. with GSS API, GSI API).

Ensuring that communicating parties are trusted (or at least known) and that communications are not altered makes it much harder for malicious or accidental behaviour to damage the system without being traceable to a cause. Ref- [7]



---

Check that authentication is in place. If it is not, consider the possible impact and consider adding authentication.

SV, ADL

#### **COM-04**

##### **Fail authentication if any data is missing**

If any files or other information is missing, for example the Certificate Revocation list, which is expected to be there by the authentication process, the authentication must fail.

Check this is the case

SV

#### **COM-05**

##### **Disconnect on Authentication Failure**

The principal must be disconnected if the authentication fails.

Check that the Authentication method causes a disconnection when authentication fails. Check that the authentication method is integrated in such a way that the principal disconnects completely if the authentication fails.

SV

#### **COM-06**

##### **Encrypt sensitive information**

Any network communication containing sensitive or personal data should be encrypted. Ref - [7]

Check that sensitive information is encrypted.

SV

#### **COM-07**

##### **Consider the impact of high connection request rate**

Consider what will happen if a principal makes a high number of requests to connect, whether authenticated successfully or not. Consider the impact on the system and how it will be handled so that it degrades appropriately.

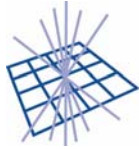
SV.

### **3.4. INPUT CHECKING**

It is important that you do not assume all input is valid, or from a friendly user or client. More is said in Ref - [9].

#### **INP-01**

##### **Validate all input**



---

All input should be validated, to ensure that the processing doesn't attempt to process invalid input which may cause the program to crash or behave in a way that is not intended.

Check that all input is validated.

VRR.

### **INP-02**

#### **Validate at each connection**

Don't assume that the client is friendly or the intended client is used. A piece of software may have been written which is only expected to be used as a service to other parts of the system with the client also provided by the author. This must not be assumed, an attacker will look for weaknesses in the system. This does not mean that for convenience and efficiency clients shouldn't check that input from the user is valid.

Check input is validated at each connection point. This is essential

VRR.

### **INP-03**

#### **Validate at all levels**

It is preferable to validate all input to e.g. all classes or subroutines.

Check input is validated at all levels. This is preferable.

VRR.

### **INP-04**

#### **Look for valid input**

It is better to look for input that is understood and valid, and reject anything else.

(This does not mean that checking for potentially threatening input cannot be done additionally.)

Check that this is done.

VRR

### **INP-05**

#### **Check that input values are within an acceptable range**

Check that all input values are within an acceptable range that can be handled by the program.

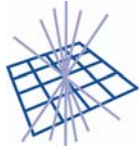
Check this is done

SV

### **INP-06**

#### **Scan for potentially dangerous input**

If input contains a system command, then it is possible that this command could get executed on the system. Similarly checking for inclusion of URLs when they are not expected may reduce the risk of vulnerability. If input is expected in alpha-numeric form, check that there are no non alpha numeric characters in the input.



Check that input is scanned for invalid input.

VRR.

#### **INP-07**

##### **Check for input that links to input or fetches input from elsewhere.**

If the input includes or fetches input from elsewhere either reject it, or check the entire input including that fetched from elsewhere.

Check that no import can be linked from elsewhere, or if it is the validity is checked.

VRR

#### **INP-08**

##### **Reject if input is invalid**

Reject with an error message and disconnect if the input is invalid. This is highly recommended, rather than attempting to clean up or convert to valid input.

Check invalid input is rejected. If not check it is sanitised adequately and consider whether it poses a risk.

VRR

#### **INP-09**

##### **Handle a high load.**

Implement a strategy for handling a high load on the system. The system should degrade gracefully and still function.

Check that such a strategy is in place.

SV

### **3.5. BUFFER OVERFLOWS**

Some programming languages (in particular C and C++) contain flaws in their string handling that allow buffer overflows to take place. This can cause a program to crash, or even allow the program to be commanded to do what was not intended.

#### **BUF-01**

##### **Don't use gets()**

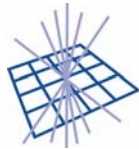
This is the most risky in C and C++. Avoid using this.

Check that it isn't there. Re-code if it is.

SV

#### **BUF-02**

##### **Avoid risky constructs**



Many constructs are risky in C, C++. For a list of risky constructs and more information on buffer overflows see

<http://www-106.ibm.com/developerworks/library/s-buffer-defend.html>

Also see Ref – [9]

Check that risky constructs are not used, or if they are the potential risks have been avoided. Consider using a well tested string handling library such as SafeStr (<http://www/zork.org/safestr/>).

SV

### 3.6. UNWELCOME CODE BEHAVIOUR

Sometimes a well designed and neatly coded system can contain flaws where in certain circumstances the code behaves in a way that isn't welcome. Such things include signed to unsigned coercion, overflowing integers. If the software can fail in certain circumstances, this can cause a denial of service, whether intentional or not. Most code unwelcome code behaviour is avoided by careful design, coding and testing. However, we point out a few additional things here that should be checked.

#### UCB-01

##### **Ensure all variables are typed**

Check that all variables are typed.

SV

#### UCB-02

##### **Ensure all variables are initialized**

Check that all variables are initialized

SV

#### UCB-03

##### **Ensure variables remain in range**

E.g. before adding two variables together, it is important to ensure check that they are not too large. Before incrementing a counter, check that it is not going to cause the integer to overflow. Alternatively, this can be done by ensuring input values are restricted such that overflows cannot occur.

Check that such checks are done before adding or incrementing.

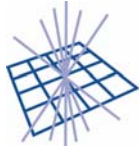
SV

#### UCB-04

##### **Check conversion of variables**

Ensure that if variables are converted from one type to another, this is done correctly. Ensure that there is no implicit casting of variables e.g. by other functions or classes which might have undesirable effects.

Check through the code looking for likely problems



SV

#### **UCB-05**

##### **Clean memory when allocating**

It is important to ensure memory allocated is clear, so that it does not contain any 'left over' information that may cause a leak of information or undesirable effect at some point. (E.g. with C and C++ use calloc rather than malloc)

Check appropriate memory allocation.

SV

#### **UCB-06**

##### **Erase memory after usage**

This must be carried out after using sensitive data, including passwords, encryption keys and other confidential data. It should also be carried out at other times.

Check this is carried out.

SV

#### **UCB-07**

##### **Avoid memory leaks**

Check that appropriate memory clearing is carried out, and that the program doesn't consume ever increasing amounts of memory.

SV

#### **UCB-08**

##### **Avoid thread build-up**

Take care on the handling of threads, to ensure that large numbers of threads do not get into use. This may be done by using a thread pool.

Check that the number of threads does not keep increasing with time.

SV

#### **UCB-09**

##### **Limit no. of connections or sockets**

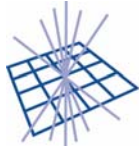
Avoid a large number of connection or sockets build up, e.g. by counting the number of sockets and limiting the number allowed. Ref – [9]

Check this is done.

SV

#### **UCB-10**

##### **Check looping**



Check that the program cannot get into an endless loop. E.g. a while statement cannot continue indefinitely.

SV

### **UCB-11**

#### **Use formatting functions carefully**

Certain use of formatting functions can expose vulnerabilities. E.g. in C printf and syslog can cause problems, Ref – [9] so they are best avoided.

Check your appropriate language if you include formatting.

SV

## **3.7. CONFIDENTIAL DATA**

In addition to practices associated with unwelcome code behaviour, such as clearing memory after usage, it is important to take special care when coding software that does or may handle confidential data.

### **CON-01**

#### **Represent keys as byte arrays**

Keys should be represented within the code as byte arrays, and not as integer arrays. Ref – [9]

SV

### **CON-02**

#### **Store keys securely**

Keys, whether they be private keys from the PKI structure or whether they be symmetric encryption keys used for the storage of confidential data must be stored securely. They must at least be password protected as well as not world readable. It is recommended that recognised methods for the secure storage of information are used.

Check this is done

SV

### **CON-03**

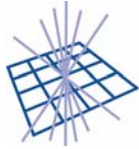
#### **Avoid caching sensitive data**

It is important that someone with legitimate access to a system cannot access sensitive data as the result of operating system activity, in particular caching. This is especially true for passwords and encryption keys.

Check appropriate measures are taken, e.g. using a recipe in ref – [9]

SV

### **CON-04**



### **Delete files securely**

Ensure that files that may contain confidential information are deleted in a secure manner. Just deleting them is not adequate, the information may still be present and a may still be recovered from disk. Use an appropriate recipe for cleanly deleting the file and information, e.g. in ref –[9]

Check this is done

SV

## **3.8. BACKDOORS**

It is important to check that there are no backdoors into the system. If there are, all other checks can be meaningless as they can be by-passed.

### **BCK-01**

#### **Test backdoors**

When writing the code, certain backdoors may have been introduced to enable easy testing. These could be in a form of e.g. a special command to access the system. Ensure that these are removed before the software is distributed or released.

Check any such backdoors have been removed.

SV

### **BCK-02**

#### **Database access backdoors**

It may be possible to do damage to the system by accessing a database the software depends upon. It is important to ensure that access to such a database is carefully controlled, e.g. isn't accessed by a commonly known user name and password, and isn't on a machine accessible to users.

Check how any database is accessed by the system.

SV

## **3.9. MIDDLEWARE ACCESS TO THE SYSTEM**

As stated earlier, it is important that the middleware runs on the least privilege basis, to minimize the impact of any vulnerability.

### **MAS-01**

#### **Restrict connectivity**

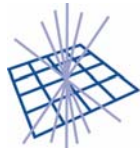
Ensure appropriate firewalls are in place, and onward connectivity is strictly controlled.

Check this is the case.

SV

### **MAS-02**

#### **Restrict ability to execute other programs**



---

Ensure that the middleware cannot be induced into executing a program that it should not, such as a Trojan or virus.

(Look up on how to do this.)

SV

### **MAS-03**

#### **Use of Environment Variables**

It is possible for an attacker to sabotage the environment variables.

See [R3]. Only use environment variables if there is no alternative, for example use a configuration file to set information. Any environment variables must be treated as un-trusted third party input.

SV

(needs more)

### **3.10. EXPOSING INFORMATION ABOUT THE SYSTEM**

Also see file handling – allowing access to the file system could expose information about the system.

Also see UCB-05, FIH-01

#### **EXP-01**

##### **Disable Core dumps**

In unix with C or C++ in particular, core dumps may reveal information that should not be revealed.

Check core dumps are disabled.

SV

#### **EXP-02**

##### **Ensure Error messages don't expose inappropriate information**

It is possible that error messages could reveal information that should not be revealed. Take care that error messages cannot be used by a hacker to obtain information about a system that should not be disclosed.

SV

### **3.11. FILE HANDLING**

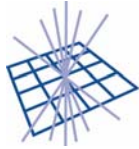
It is important that a hacker is not able to obtain information or interfere with the system by accessing or modifying files they should not. See also CON-04

#### **FIH-01**

##### **Access Control**

Ensure access control is set so that the program cannot access files it should not access, for example system files.

Check what files the program can access.



SV

#### **FIH-02**

##### **Ensure the program builds the location of a file**

The program should not take input from an external source concerning the location or directory of the file, this should be constructed by the program itself, for example from a logical file name.

Check this is the case.

SV

#### **FIH-03**

##### **Check use of relative file names.**

Relative file names can make it possible to change a reference from the file working directory to another directory. Ensure that if relative file names are used, the program accesses the appropriate directory and does not allow the user to move up the tree. Ref – [8]

Check this.

SV

#### **FIH-04**

##### **Don't make access decisions based on environment variables**

Such variables might not take on the correct value.

Check this.

SV/VRR

#### **FIH-05**

##### **Don't refer to a file twice in the same program by its name**

Open a file and use the file handle or other identifier from then on, this prevents the file being substituted by another one between references. Ref – [8]

Check this.

SV

#### **FIH-06**

##### **Reject input that may influence the path.**

Ensure that the user cannot force the program to access files it should not by requesting a filename including a path to that filename.

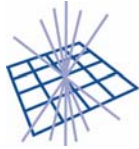
Check this.

SV

#### **FIH-07**

##### **The program should not accept input if it contains the potential to move up the directory tree**

For example, input containing things such as ../ in unix should be rejected.



Check this is the case.

SV

### **FIH-08**

#### **Restrict size of temporary files**

It is important to consider the size of temporary files in the system, and ensure that they cannot be used to fill up all disk space and cause a denial of service attack.

Check that the temporary file sizes are carefully controlled and handled.

SV

### **FIH-09**

#### **Avoid world readable files**

Check this is done

SV

### **FIH-10**

#### **Don't trust user-writable storage not to be tampered with**

Treat anything read from user-writable storage as non-validated input, and check it's validity before processing

Check this is done

SV

## **3.12. AUTHORIZATION**

Most actions should only be carried out by an authorized principal. However, there may be a few exceptions, such as access to public information.

### **AUZ-01**

#### **Authorize the action**

Check that authorization to carry out the action is in place, or if not there is a very good reason why authorization is not necessary and not a threat.

SV

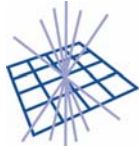
### **AUZ-02**

#### **Authorize all job execution**

Carry out authorization if a user is to be running a job where they are submitting any code to be executed.

Check that Authorization is carried out.

SV



### **AUZ-03**

#### **Use established authorization tools where possible**

As far as possible, use established tools. In the case of course grained authorization use the grid authorization tools developed jointly by the various projects. In the case of fine grained authorization, ensure common tools are used to check the credentials and pass on the validated credentials to the service.

Check this is the case.

VRR

### **AUZ-04**

#### **Check Credentials are valid on connection**

Whether course grained authorization is appropriate, or fine grained, the validity of the credentials should be checked on connection prior to the authorization decision point.

Check this is the case

VRR

### **AUZ-05**

#### **Disconnect if invalid Credentials are presented.**

Whatever authorization the service requires, if a principal presents credentials that are invalid or out of date, or if the checking fails in any way disconnect the principal.

Check this is the case.

SV

### **AUZ-06**

#### **Disconnect if full checking cannot be carried out**

If any of the data needed to carry out authorization is missing, disconnect rather than allow the connection to continue with incomplete checking

SV

### **AUZ-07**

#### **Disconnect on failure if access to resource is denied.**

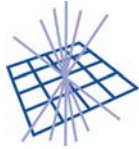
If authorization is course grained, i.e. the question is whether or not the principal can access the resource, disconnect if the authorization fails.

Check this is the case.

SV

### **AUZ-08**

#### **Use 'Default Deny' strategy**



Base the authorization logic on default deny as far as possible, and only allow access if the principal matches the criteria. This prevents any access by default. This does not prevent having e.g. a list of users who are denied access even if they satisfy other criteria.

Check this is the case.

VRR

### **AUZ-09**

#### **Check fine grained Authorization carefully**

If the software uses fine grained authorization, special care needs to be taken as although security middleware may be able to check the credentials and pass them into the system, it is up to the middleware itself to compute the authorization decision.

Check code very carefully.

VRR

### **3.13. USAGE**

The aim should be to allow the system to be used for legitimate purposes, but not abused. The aim should also be to allow all legitimate users to gain access, and not be blocked due to excessive use, whether the usage is due to a large legitimate job having being submitted or due to the software behaving in a way it should not and using an excessive amount of resources.

#### **USE-01**

##### **All usage of resources should be self-limiting.**

The programs should be written such that they do not take up all the resources.

Check this is done

SV

#### **USE-02**

##### **Quotas**

Have a quota on usage, this will help prevent a denial of service attack by over use of the system, or filling up disk space. It also prevents extreme accidental over use, if a user has unwittingly executed a program that requires a huge amount of resources.

Check some sort of quota mechanism is in place.

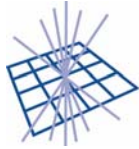
SV

#### **USE-03**

##### **Restrict what software can be executed**

A large amount of resources could be used to crack a private certificate.

To avoid this restrict what software can be used. This could be done by only allowing packages signed by a particular VO or other appropriate organisation to be used.



---

Check what is done to restrict usage and document any specific risks.

SV

#### **USE-04**

##### **Monitor usage**

Usage should be logged, as stated in the logging section.

Have a tool in place that allows the usage to be monitored, especially users of large amounts of CPU.

Check some sort of tool is available.

ADL

### **3.14. DEPENDENCIES**

#### **DEP-01**

##### **Consider whether the dependency is necessary and appropriate**

Care should be taken with dependencies, to ensure you only use ones that are well tested and reliable. If there is a vulnerability in the dependency, however carefully you code your own software

Check through dependencies, and see if they are from trustworthy sources

VRR

#### **DEP-02**

##### **Document dependencies**

Document all dependencies, including the earliest version that is acceptable. This is to ensure that a vulnerability is not introduced due to installing an old un-patched version.

Check that such documentation exists.

VRR

#### **DEP-03**

##### **Ensure a later version of the dependency can be used**

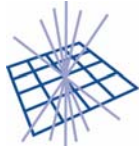
If possible, do not write the software such that it depends on a specific version of a dependency, ensure that it is written such that a later version can be used. This is particularly important if a vulnerability is found in the dependency, and it is necessary to upgrade the dependency.

Check that it is possible to run software with a later version. Check any appropriate tools that check for version on installation, e.g. check that they specify an earliest version rather than a specific version if possible.

VRR

### **3.15. ERROR HANDLING**

However carefully a system is written and tested, it may still contain bugs and fail in some way. It is important that it fails in a way that does not expose a vulnerability, in particular does not allow a



principal unauthorized access. It is also important that errors due to problems with the code are found quickly and easily, so that they can be fixed. When an error occurs e.g. due to lack of resources, loss of network or communication, determining the cause of the problem and subsequent corrective action is greatly assisted by appropriate logging. It is also important that the error messages don't expose information to a hacker that may be useful to them, so errors presented to the user need to be carefully formulated, EXP-2. Possibly further information needs to be logged in a way that is not accessible to the user, so the problem can be found.

### **ERR-01**

#### **Detect Errors**

Errors should be detected

ADL

### **ERR-02**

#### **Log Errors**

Errors should be logged on the system. Sufficient logging should occur to allow the cause of the problem to be determined by the people responsible for the system

ADL

### **ERR-03**

#### **Report Errors**

Errors should be reported to the appropriate user or other principal.

Appropriate errors should be reported to the user, so that the user can determine whether their input or credentials are at fault, or whether the error is due to a problem with the system. Errors should be reported to other appropriate principals so the cause can be determined.

Check this.

ADL/VRR

### **ERR-04**

#### **Have informative error messages**

Error messages should be clear to understand, and find what has gone wrong. This helps us find the source of the problem.

Check this.

ADL

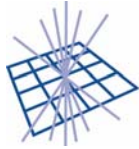
### **ERR-05**

#### **Degrade gracefully**

The software should degrade and fail gracefully if at all possible, which reduces the risk of a vulnerability being left open.

Check there is something in place to degrade/ recover gracefully.

VRR



### **3.16. FAILURE**

At some point software may crash, and it is important that this does not pose additional problems such as allowing unauthorized access or revealing secrets.

See also EXP-01

#### **FAI-01**

##### **Have a 'Fail Safe' strategy**

If the software crashes, or experiences an error it cannot recover from, it is important that it is left in a state that can do no harm. Consider what this state is, and take steps to leave it in a safe state.

Check whether this has been done, and whether the 'fail safe' strategy is a good one.

VIR

### **3.17. LOGGING MECHANISMS**

#### **LOG-01**

##### **Log all security 'state' transitions**

Log security state transitions: connected, authenticated, authorized (if appropriate coarse grained authorization is carried out) and disconnected.

Check that mechanisms are in place to log these.

ADL

#### **LOG-02**

##### **Log access to resources**

Ensure that logging mechanisms are available that allow for the logging of access to resources

Check mechanism is in place.

ADL

#### **LOG-03**

##### **Log usage of resources**

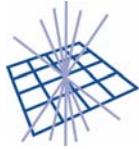
Ensure that logging mechanisms are available that allow for the logging of usage of resources

Check mechanism is in place.

ADL

#### **LOG-04**

##### **Traceable logging**



Ensure that the logging mechanisms include recording of the users credentials (e.g. DN, VO membership and roles) that enable access to be authorized.

Assess the mechanism in place.

ADL

## **LOG-05**

### **Configurable logging**

Ensure that the logging mechanisms are configurable, so that logging required can be set up. For example in some cases it may only be necessary to log that a particular principal has accessed a database, in other cases it may be necessary to log exactly what has been accessed.

Assess the mechanism in place.

ADL

## **LOG-06**

### **Non-Repudiation**

Logging should be carried out in such a way than an action cannot be denied. Logging needs to be carried out in such a way that it cannot later be altered (including by sysadmins) without detection.

Check whether there is a non-repudiation mechanism.

ADL

## **LOG-07**

### **Tools**

Tools must be available that allow easy analysis of the logs, including detecting abnormal usage, disallowed usage such as access using a service certificate when a user certificate is required. Also, tools should allow any appropriate operator to easily find problems.

Check they are in place

ADL

## **3.18. SPECIFIC CHECKS ON SECURITY MIDDLEWARE**

It is particularly important that security middleware works reliably, and does not behave in a way that allows unauthorized access. All other checks as described here still apply.

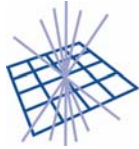
## **SMW-01**

### **Check each language**

There are APIs and middleware to carry out Authentication and Authorization in various languages to the security system. These all need to be checked. We need to ensure that these are examined at tested with the same rigor as all other tests.

Check this is done.

SV/VRR



## **SMW-02**

### **Ensure credentials are properly checked**

The software needs to be carefully examined and thoroughly tested to ensure that the credentials properly checked and passed into the system.

Check this

SV/VRR

## **3.19. INTEGRATION OF SECURITY MIDDLEWARE WITH OTHER MIDDLEWARE**

### **ISM-01**

#### **No By-pass for the security**

Check that the security middleware is integrated with other middleware does not allow for a bypass of the security middleware. The security middleware should be on front of the other middleware, and not allow the other middleware to be accessed if the security check fails.

Check this is the case.

SV

## **3.20. TESTING**

Most people are aware that it is important to test software thoroughly. All applications should be tested for function and fault conditions. Remember that friendly inputs cannot be assumed. Ref – [7].

### **TEST-01**

#### **Test with potentially threatening input**

Consider what input may present a threat to the software, and test with this as well as any functional testing. This should be rejected and give an error and disconnection.

VRR

### **TEST-02**

#### **Test with random input**

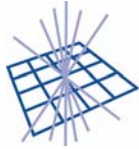
All such input should be rejected, unless in the unlikely case that some is valid.

Check such tests are carried out.

VRR

### **TEST-03**

#### **Test with long random strings**



---

This is good for general input checking too. Construct a test which throws long random strings at the software, and see it is possible to cause the software to crash or behave in an undesirable way.

Check has been done, or do it.

SV

#### **TEST-04**

##### **Test with input just in range**

The software should be tested with input just in range, to ensure that it is processed correctly.

Check such tests are carried out

VRR

#### **TEST-05**

##### **Test with input out of range**

The software should be tested with input out of range, to ensure that it is successfully rejected.

Check this is done.

VRR

#### **TEST-06**

##### **Test with a high load.**

Carry out some tests with a high load. Test with more input than the system can handle, ensure the software degrades sensibly.

Check such tests have been carried out.

VRR

#### **TEST-07**

##### **Test for memory leak**

Carry out tests that stress the system over a length of time, to ensure that no memory leaks occur.

Check this is carried out.

SV

#### **TEST-08**

##### **Test for thread build up**

Carry out tests over a length of time, to ensure threads don't build up.

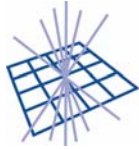
Check this is carried out.

SV

#### **TEST-09**

##### **Test with input containing the potential to move up the directory tree**

Check such input is rejected.



SV

### **TEST-10**

#### **Test at all connection points**

Tests should be carried out at all potential connection points. For example, if a service normally expects another part of the package to act as the client, then the service should still be tested as if it is a stand alone service.

Check such testing is done

SV

### **TEST-11**

#### **Test with invalid Credentials**

Tests should be carried out using credentials that are invalid. This should include Credentials from authorities that are not accepted, as well as out of date credentials.

Check that such credentials are rejected, and connection is not allowed.

SV

### **TEST-12**

#### **Test with missing and invalid security files**

Test with each required security file missing and invalid in turn. (e.g. for authentication the public certificate, the certificate revocation list, the signing policy file)

Check that in each case the connection is rejected.

SV

### **TEST-13**

#### **Test with requests beyond the principals rights**

Check with valid credentials, but try and request access beyond the rights of the principal.

Check such requests fail.

SV

### **TEST-14**

#### **Test in a distributed environment**

Test that the system worked in a distributed testbed at more than one site.

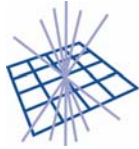
Check such tests are carried out

SV

### **TEST-15**

#### **Test in a distributed environment when some machines are switched off**

This should include tests where there are enough machines to provide the basic functionality, and ensure the system basically works, to ensure robustness.



Check such tests are carried out.

SV

### **TEST-16**

#### **Test with some machines incorrectly configured**

Check that such machines are handled appropriately (probably ignored) and they do not prevent the rest of the system from working. It is especially important, that 1 hacked or bad machine should not cause the rest to fail. This should include having machines with e.g. the firewall incorrectly set, as well as sites setup maliciously. If it is possible to add a site and configure it incorrectly and cause the whole deployment to fail or loose efficiency this is a vulnerability.

Check such tests are carried out.

SV

## **3.21. OTHER MIDDLEWARE CHECKS**

### **OTH-01**

**Check if other programs are invoked, and if so if they are trustworthy.**

Ref - [8]

SV

### **OTH-02**

**Check usage of setuid**

Ref – [8] suggests avoiding setuid. This is probably not possible

Flag if it is used, and check carefully how it is used. E.g. check against recipe in Ref – [9]

SV

### **OTH-03**

**Check usage of shell or command line**

Ref – [8] suggests avoiding shell and command line.

Flag if used, and check carefully how it is used.

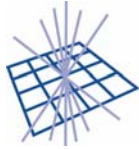
SV

### **OTH-04**

**Don't confuse Random with pseudo-random**

Ref – [8]

If random numbers are used, check how they are generated and check that they really are random, and that values are not predictable.



SV

### 3.22. DOCUMENTATION

#### DOC-01

##### **Clearly document the Grid Security model**

It is important that the Grid Security model is clearly documented, so that it can be well understood by users and system administrators.

Check a suitable document is available

VRR

#### DOC-02

##### **Ensure the Installation guide or instructions are clear and correct**

If the installation guide is clear and correct, especially concerning the security configuration it is less likely that there will be mis-configured or insecure installation.

For example, ensure you document what must be changed, what may be changed, and what should not normally be changed.

Check such instructions are clear.

VRR

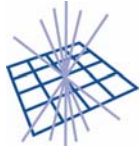
#### DOC-03

##### **Check that clear guidelines are available for care of credentials**

For users and administrators, it is important that they take care of their credentials, and do not leave copies available for public access.

Check such guidelines are clear.

VRR



## 4. DEPLOYMENT AND CONFIGURATION CHECKLIST

In order to deploy the software securely, it might be necessary to go back and ensure the software allows secure deployment. Here we list some checks that help us deploy in a secure way.

### 4.1. SOFTWARE VERSIONS

No matter how vulnerability free the software that is planned to be installed is, if the wrong version is installed or the software is maliciously substituted with another version then the care that has been taken is negated.

#### VER-01

##### **Distribute software from a central source**

This allows one team of people to check that the correct version of software is available, including dependencies so that there is less chance of a site installing the wrong software.

Check the method of distribution.

VRR

#### VER-02

##### **Apply all critical patches**

Ensure a system is in place to quickly apply any security patches quickly.

SV

#### VER-03

##### **Include mechanism for proving software is valid**

There should be some anti-tampering in place. This could be some sort of software signing or some sort of mechanism that allows a client to detect that the service contains the wrong software.

This should be investigated in the future.

SV

### 4.2. DEPLOYMENT METHODS

#### DM-01

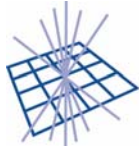
##### **Automate when possible**

Automate the installation and configuration where appropriate. It is inevitable that the installer will need to setup some things which define the configuration of that particular site or resource, as well as passwords.

Check this is done

VRR

#### DM-02



### **Include good configuration scripts**

Ensure these allow the installer to easily setup the software in a secure way.

For all configuration it is best to document the following:-

What must be changed

What may be changed

What should not normally be changed.

Check this is done clearly

VRR

### **DM-03**

#### **Deploy and configure with appropriate privileges**

Ensure that the principle of least privilege is adhered to in the deployment.

Check this.

VIR

## **4.3. DEPLOYMENT AUTHENTICATION AND AUTHORIZATION**

### **DAA-01**

#### **Ensure in all cases non - authenticated connections are disabled.**

This means that any principal must have a certificate from an acceptable CA in order to be able to successfully connect.

Check no unauthenticated connections are allowed. If they are, assess the risk.

SV

### **DAA-02**

#### **Ensure in all cases non-authorized actions are disabled**

This means that all actions, any principal must have acceptable credentials from a VO in order to carry out an action

Check appropriate authorization is in place for all actions. If no authorization is required, assess the risk

SV

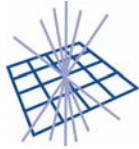
### **DAA-03**

#### **Ensure authorization is enabled in order to run a job**

For job execution, authorization should always be enabled.

Check this is the case

SV



## **DAA-04**

### **Carefully assess authorization to carry out potentially damaging actions**

Some actions can cause a lot of problems for others, such as the deletion of all files in a database. Carefully check who is authorized to carry out such actions, and that the system is correctly configured.

SV

## **4.4. FIREWALLS**

### **FIR-01**

#### **Use firewalls**

Check that a firewall is in place

VRR

### **FIR-02**

#### **Control over firewalls**

Check that the system administrator has control over which ports are open.

VRR

### **FIR-03**

#### **Ensure firewall is correctly configured**

Check ports needed for that S/W are open, but there isn't a blanket opening of firewalls. This means that appropriate specification and configuration files must be in place.

VRR

### **FIR-04**

#### **Ensure software is configured appropriately to restrict onward connectivity**

It is important that Grid resources cannot be used to e.g. launch a SPAM attack.

Check that appropriate software is in use and configured appropriately

SV

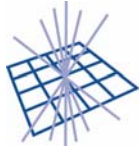
### **FIR-05**

#### **Handle incorrectly configured firewalls**

It is important that if the firewall at one site or to one resource is configured incorrectly, it does not cause the whole system to fail. The situation where certain connections cannot be made should be seen as an error condition and handled gracefully.

Check the handling of badly configured firewalls.

SV



---

## 4.5. USE OF CREDENTIALS

The CA and VO system for issuing credentials is checked by others, so we assume all appropriate checks are made concerning the issue of credentials. Here we just consider their handling in the Grid system.

Produce a Secure Credential Storage Procedures guide. (e.g. JRA3 guide –TBW)

Here are a few obvious checks.

### **CRD-01**

#### **Password protect all private keys**

Check that all private keys are password protected.

SV

### **CRD-02**

#### **Private keys should not be shared**

It is important not to share private keys, and this includes host keys.

Check this

SV

### **CRD-03**

#### **Ensure all passwords are passed in encrypted form**

Check this

SV

### **CRD-04**

#### **Don't echo passwords or display on users screen**

Check this

SV

### **CRD-05**

#### **Don't distribute passwords by E-mail**

If possible, distribute them person to person.

SV

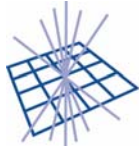
### **CRD-06**

#### **Secure management of encryption keys**

Some data is stored in encrypted form.

Check strategy for management of encryption keys

SV



#### **CRD-07**

##### **Avoid sharing user names and passwords.**

Do not use commonly known user names and passwords, even though this may ease deployment.

Check this

SV

#### **CRD-08**

##### **Avoid sharing grid credentials.**

Do not share grid credentials between users, or between different services, or between users and services.

Check this

SV

#### **CRD-09**

##### **Don't authenticate on un-trusted criteria**

Don't authenticate on anything that isn't intended to serve that purpose, such as IP numbers, MAC addresses, or E-mail addresses.

Check there is no such authentication

SV

#### **CRD-10**

##### **Ensure the end user is traceable**

Make sure that the user who submitted a job is traceable.

Check this is done

ADL

#### **CRD-11**

##### **Don't deploy services that break the rules.**

For example, don't deploy a service that allows a user to use the services credentials for authentication or authorization with other services.

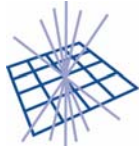
Check this is not done

VRR

#### **CRD-12**

##### **Protect Proxies**

A proxy hijack could be a serious problem, as it could appear that a user has done something they have not. Ensure the environment they are stored in is as secure as possible.



SV

#### **CRD-13**

##### **Ensure Authority security files are obtained by a trusted method**

Check the mechanism for obtaining e.g. CA certificates and VO certificates are adequate.

SV

#### **CRD-14**

##### **Ensure test security files are not present in the deployment**

E.g. test CAs and VOs are sometimes used for testing the software, which are just downloaded files which prove nothing about the user or site.

Ensure such files not present in the deployment.

SV

## **4.6. HANDLING CONFIDENTIAL DATA**

#### **CD-01**

##### **Ensure Confidential data is stored on an Authorized resource**

Check that procedures are in place to avoid accidentally copying data to a non-trusted principal.

SV

#### **CD-02**

##### **Ensure Confidential data is stored in encrypted form.**

Check this

SV

#### **CD-03**

##### **Don't store confidential data without password protection**

Don't store sensitive information in a database that does not have password protection, even if it is encrypted and in a secure area with limited access.

Check this

SV

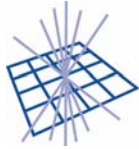
#### **CD-04**

##### **Ensure Authorized access is deployed**

Don't distribute Confidential data in unencrypted form

E.g. don't send it by e-mail

SV



See also the logging section.

## **4.7. LOGGING STRATEGY**

There is inevitably a trade-off between logging everything possible to ensure the maximum information is available to detect any misuse, and not setting up a system that allows a denial of service attack to be carried out by overloading the logging system and filling up disks.

### **LS-01**

#### **Enable logging all security 'state' transitions**

Log security state transitions: connected, authenticated, authorized (if appropriate coarse grained authorization is carried out) and disconnected.

Check that mechanisms are in place to log these.

ADL

### **LS-02**

#### **Log access to sensitive information**

It is important to setup the system to log access to sensitive information, even if this access is authorized.

Check appropriate logging is enabled.

ADL

### **LS-03**

#### **Log access to any resource.**

Check appropriate logging is enabled

ADL

### **LS-04**

#### **Ensure end user is logged**

Ensure that on any resource the user who submitted the job is logged.

Check this

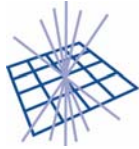
ADL

### **LS-05**

#### **Have a log checking system**

Have a procedure for checking the logs on a regular basis.

Check this is in place



ADL

#### **LS-06**

##### **Check level of logging against resources available**

If too much logging is enabled, it would be possible to significantly slow down the system or fill up the disk space.

Ensure that the level has been checked and is realistic.

SV

### **4.8. STAFF**

Care needs to be taken over what roles or combination of roles principals may have.

Should this be here? Or is it too far into deployment?

#### **STF-01**

##### **Consider who has privileged roles**

Take care that you are confident of the people you issue with privileged roles. It may be a better strategy to give the more privileged roles to employees who have either been vetted for security or who have been with you for a long time than to give privileged roles to short term temporary staff.

VRR

#### **STF-02**

##### **Consider the combination of roles**

Combination of a rogue VO admin and CA admin could grant huge rights to a fictional user! Also, if the same person is in charge of detecting abnormal usage and administrating a VO or CA then abnormal usage could be overlooked.

VRR

#### **STF-03**

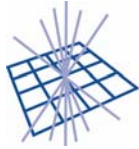
##### **Ensure that staff receive sufficient training**

If staff are not trained properly, especially sysadmins, they are more likely to configure a system in a way that is vulnerable than if they receive sufficient training and thoroughly understand what they are doing. This includes allowing staff time for training, including for example one of the sources of training is reading a document they need time to do that.

Check training plan is in place

VRR

### **4.9. INCIDENT RESPONSE AND DETECTION**



**IR-01**

**Have an incident response strategy at the time of deployment**

See for e.g. Ref – [11]

ADL

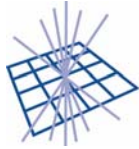
**IR-02**

**Develop a strategy for detecting abnormal usage**

This is in order to detect anything that goes wrong as early as possible.

This may be in the form of monitoring normal usage, and investigating usual usage.

ADL



## **5. SPECIFIC KNOWN VULNERABILITY LOGGING**

As people become aware of vulnerability in the software or deployment it is important to log it in a manner that allows it to be addressed, eventually eliminated and allow us to learn from this to prevent a re-occurrence. It is also important to log it in such a way that it is not readable by any unfriendly principal.

### **5.1. WHAT TO LOG**

The following is a list of what should be logged.

#### **5.1.1. Location**

Where is the problem

#### **5.1.2. Vulnerability**

Description of the problem, including what can be done that should be prevented.

#### **5.1.3. Exploitation**

Who has exploited it, and who can potentially exploit the vulnerability. This should include whether it can be exploited by e.g. someone who has no credentials, authenticated, authorized (including whether they can act beyond their intended rights).

#### **5.1.4. Analysis**

Where the problem lies, is it a configuration problem on a limited number of sites? A problem with configuration scripts, middleware package, or a combination of the effects of more than one piece of software and/or installation.

#### **5.1.5. Risk analysis**

A brief description of the risks associated with the vulnerability, including who it may impact.

#### **5.1.6. Proposed solution**

What should be done address this vulnerability, it may be that new software is needed, one or more middleware packages need modification, configuration scripts need modification, or a simple change at one site. It could also be a proposal for specific logging, in the case where it is difficult to prevent an authorized user doing something – to prevent access in the future.

#### **5.1.7. Bugs submitted**

Bugs in the middleware or in the configuration scripts may result from the analysis.

#### **5.1.8. Checklist reference**

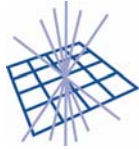
Items in the checklist would have picked up the vulnerability if the checklists had been applied

#### **5.1.9. Checklist proposal**

Any proposal for checks to add to the checklists

## **5.2. RESTRICT ACCESS TO LOG**

Access to logs of specific vulnerability needs to be carefully controlled.



## **6. VULNERABILITY 'USE CASES'**

It is worth while considering how one would go about an attack on the system, to achieve various aims, and look at how the system prevents the attacker achieving their aims. This could be considered to be an 'Anti use case'

### **6.1. USE CASE CHECKING**

For each use case, in various roles it is worth considering the following:--

#### **6.1.1. How would you achieve it?**

This is where you think about what you would do to go about achieving a use case. It is also worth considering each type of principal, whether a user without credentials? An Authenticated user? An authorized user? A system administrator? For each use case, you will then ask the following.

#### **6.1.2. What Specific Mechanism(s) prevent this?**

You look at the security mechanisms that prevent it.

#### **6.1.3. Try it - possibly**

Try it out on a testbed, but make sure you have written permission to try it out first. This is most important.

#### **6.1.4. Is the attempt successful?**

Is it possible to achieve it?

#### **6.1.5. Is the attempt logged?**

Does appropriate logging occur? If the attempt was successful, does it show up in the log?

#### **6.1.6. Is the attempt detected?**

How is an attempt detected? Does the routine analysis of the log pick up appropriate activity?

#### **6.1.7. How is the attempt handled?**

What would automatically be done to if this happened, particularly if it was successful?

### **6.2. USE CASE EXAMPLES**

Here we describe a few 'use case' type scenarios that the vulnerability prevention and assessment should guard against. They can be considered to be 'anti use cases', things that must not be allowed.

Here are some ant use case examples, which could be considered.

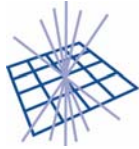
#### **6.2.1. Storage of illegal material for distribution**

This is where the Grid storage facilities are used to store material for illegal distribution. This may be e.g. pornographic material, or it may be copyright material.

#### **6.2.2. Certificate Cracking**

A person may wish to crack a banks certificate, and use it to steal money.

#### **6.2.3. Credentials theft**



---

A person may steal another persons' authentication and authorization credentials to carry out actions in another principal's name.

#### **6.2.4. Launch of a denial of service attack on the Grid**

This is where a person prevents usage of the grid for instance by overloading it, or by causing it to fail in some way. This could be done intentionally by someone with malicious intent, or by mistake by an authorized user if the system contains appropriate weaknesses which cause it to fail in certain circumstances.

#### **6.2.5. Launch of a denial of service attack from the Grid**

This is where a person uses the grid to launch a denial of service attack on another system.

#### **6.2.6. Theft of confidential information**

This is where someone steals information, it may confidential because it contains personal data, or it may be confidential because it required a large amount of expenditure to obtain and is owned by a company.

#### **6.2.7. Destruction of valuable data**

This is where valuable data becomes deleted, whether accidentally or intentionally.

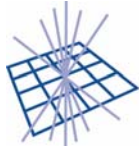
#### **6.2.8. Addition of an un-trusted host**

This is where someone attempts to add a host to the grid, which may be used in another form of attack.

### **6.3. DISCUSSION**

It is important to consider in what role such use cases may be achieved, and how they are handled. For example, it may be relatively easy for a system administrator to abuse the system, so the emphasis should be on carefully considering who you give the role of system administrator to, and then on detection and deterrence. For a principal without any credentials the emphasis should be on protection.

An attempt to carry out one use case, may be by invoking another. For example, if someone without credentials wanted to store illegal information on a storage element, they may decide to try and steal the credentials of another user who has access to that storage element. For someone with credentials, it is necessary to ensure that they cannot use the storage element in that way without detection. For authorized users and system administrators the main means of prevention is probably detection and withdrawal of credentials.



## 7. CHECKLIST TABLES

For each piece of middleware, the idea is to check each point.

Rel, is an indication if this is relevant to that piece of middleware, if it is not there is no point in checking.

Checked is the flag on whether such a check has been carried out.

Result is the result of the check, it might be that the S/W is O.K.

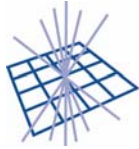
Ref is a reference to where to find more information. It may be in the form of a brief assessment of the risk of continuing with the vulnerability in place, or a plan of what to do about it if action is considered necessary.

In the case of the deployment checklist, if a problem is found it may be necessary to modify the software, which may generate a new check to add to the middleware checklist if an appropriate check isn't present.

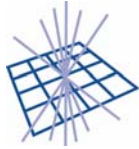
Comment is a few words if appropriate.

### 7.1. MIDDLEWARE CHECKLIST

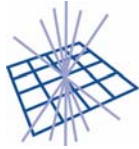
Vul No	Vul type	Rel	Checked	Result	Ref	Comment
DES-01	VRR					
DES-02	VIR					
DES-03	VIR					
DES-04	SV					
DES-05	SV					
DES-06	VRR					
DES-07	SV					
CP-01	VRR					
CP-02	VRR					
CP-03	VRR					
CP-04	VRR					
COM-01	VRR					
COM-02	VRR					
COM-03	SV/ADL					
COM-04	SV					
COM-05	SV					



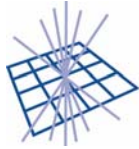
Vul No	Vul type	Rel	Checked	Result	Ref	Comment
COM-06	SV					
COM-07	SV					
INP-01	VRR					
INP-02	VRR					
INP-03	VRR					
INP-04	VRR					
INP-05	SV					
INP-06	VRR					
INP-07	VRR					
INP-08	VRR					
INP-09	SV					
BUF-01	SV					
BUF-02	SV					
UCB-01	SV					
UCB-02	SV					
UCB-03	SV					
UCB-04	SV					
UCB-05	SV					
UCB-06	SV					
UCB-07	SV					
UCB-08	SV					
UCB-09	SV					
UCB-10	SV					
UCB-11	SV					
CON-01	SV					
CON-02	SV					
CON-03	SV					
CON-04	SV					
BCK-01	SV					
BCK-02	SV					



Vul No	Vul type	Rel	Checked	Result	Ref	Comment
MAS-01	SV					
MAS-02	SV					
MAS-03	SV					
EXP-01	SV					
EXP-02	SV					
FIH-01	SV					
FIH-02	SV					
FIH-03	SV					
FIH-04	SV/VRR					
FIH-05	SV					
FIH-06	SV					
FIH-07	SV					
FIH-08	SV					
FIH-09	SV					
FIH-10	SV					
AUZ-01	SV					
AUZ-02	SV					
AUZ-03	VRR					
AUZ-04	VRR					
AUZ-05	SV					
AUZ-06	SV					
AUZ-07	SV					
AUZ-08	VRR					
AUZ-09	VRR					
USF-01	SV					
USF-02	SV					
USF-03	SV					
USF-04	ADL					
DEP-01	VRR					



Vul No	Vul type	Rel	Checked	Result	Ref	Comment
DEP-02	VRR					
DEP-03	VRR					
ERR-01	ADL					
ERR-02	ADL					
ERR-03	ADL/VR					
ERR-04	ADL					
ERR-05	VRR					
FA-01	VIR					
LOG-01	ADL					
LOG-02	ADL					
LOG-03	ADL					
LOG-04	ADL					
LOG-05	ADL					
LOG-06	ADL					
LOG-07	ADL					
SMW-01	SV/VRR					
SMW-02	SV/VRR					
ISM-01	SV					
TEST-01						
TEST-02						
TEST-03						
TEST-04						
TEST-05						
TEST-06						
TEST-07						
TEST-08						
TEST-09						
TEST-10						
TEST-11						

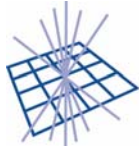


Vul No	Vul type	Rel	Checked	Result	Ref	Comment
TEST-12						
TEST-13						
TEST-14						
TEST-15						
TEST-16						
OTH-01	SV					
OTH-02	SV					
OTH-03	SV					
OTH-04	SV					
DOC-01	VRR					
DOC-02	VRR					
DOC-03	VRR					

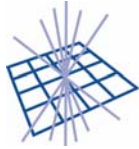
## 7.2. DEPLOYMENT CHECKLIST

Deployment checklist is less advanced.

Vul No	Vul type	Rel	Checked	Result	Ref	Comment
VER-01	VRR					
VER-02	SV					
VER-03	SV					
DM-01	VRR					
DM-02	SV					
DM-03	VIR					
DAA-01	SV					
DAA-02	SV					
DAA-03	SV					
DAA-04	SV					
FIR-01	VRR					
FIR-02	VRR					



Vul No	Vul type	Rel	Checked	Result	Ref	Comment
FIR-03	VRR					
FIR-04	SV					
FIR-05	SV					
CRD-01	SV					
CRD-02	SV					
CRD-03	SV					
CRD-04	SV					
CRD-05	SV					
CRD-06	SV					
CRD-07	SV					
CRD-08	SV					
CRD-09	SV					
CRD-10	ADL					
CRD-11	VRR					
CRD-12	SV					
CRD-13	SV					
CRD-14	SV					
CD-01	SV					
CD-02	SV					
CD-03	SV					
CD-04	SV					
LS-01	ADL					
LS-02	ADL					
LS-03	ADL					
LS-04	ADL					
LS-05	ADL					
LS-06	SV					
STF-01	VRR					
STF-02	VRR					
STF-03	VRR					



---

Vul No	Vul type	Rel	Checked	Result	Ref	Comment
IR-01	ADL					
IR-02	ADL					