Tables

1  Introduction ........................................................................................................... 7
1.1 Overview........................................................................................................... 7
1.2 esCERT-UPC ................................................................................................... 8
1.3 ALTIAIR .......................................................................................................... 8
1.4 The Common Model of System Information - CMSI ..................................... 9
2  Specifications ..................................................................................................... 11
2.1 Overview.......................................................................................................... 11
2.2 Use cases......................................................................................................... 11
2.3 Sequence diagrams ....................................................................................... 13
3  Preliminary study ............................................................................................... 17
3.1 Analysis of existing tools .............................................................................. 17
3.1.1 H-Inventory ............................................................................................... 17
3.1.2 ManagePC ................................................................................................ 18
3.1.3 Open-Audit ............................................................................................... 19
3.1.4 zCI Computer Inventory System ............................................................... 19
3.1.5 OCS Inventory New Generation ................................................................. 21
3.1.6 Conclusion .................................................................................................. 24
3.2 How to transfer the report? ............................................................................ 25
3.2.1 FTP (File Transfer Protocol) ................................................................... 25
3.2.2 SFTP (SSH File Transfer Protocol) and FTPS (FTP over SSL) .............. 25
3.2.3 RCP and SCP (Remote Copy and Secure Copy) ....................................... 26
3.2.4 Network Share .......................................................................................... 26
3.2.5 SMB (Server Message Block) .................................................................... 27
3.2.6 SOAP (Simple Object Access Protocol) .................................................. 27
3.2.7 HTTP upload ............................................................................................ 28
3.2.8 Email ......................................................................................................... 28
3.2.9 Telnet, Netcat or any socket method .......................................................... 29
3.2.10 Conclusion ............................................................................................... 29
3.3 Analysis of security ......................................................................................... 30
3.3.1 Integrity ..................................................................................................... 30
3.3.2 Authenticity ............................................................................................... 31
3.3.3 Confidentiality ........................................................................................... 31
3.3.4 What kind of method to use ..................................................................... 32
4  Designing ............................................................................................................. 33
4.1 Overview.......................................................................................................... 33
4.2 Agent .............................................................................................................. 33
4.2.1 Programming language .......................................................................... 33
4.2.2 Fingerprinting the system ........................................................................ 35
4.2.3 Interesting data to collect .......................................................................... 36
4.2.4 Installation (Cron and Task scheduler) .................................................... 38
4.3 Server ............................................................................................................. 40
4.3.1 Overview .................................................................................................. 40
4.3.2 Writing CMSI reports ............................................................................. 40
4.4 Configuration .................................................................................................. 41
5  Implementation .................................................................................................. 43
5.1 Agent .............................................................................................................. 43
5.1.1 Overview of the Agent ............................................................................. 43
5.1.2 Getting system information ..................................................................... 43
5.1.3 Retrieving the list ..................................................................................... 45
5.1.4 Signing the report ..................................................................................... 46
5.1.5 Encrypting the report ............................................................................... 48
5.1.6 Sending the report ................................................................................... 49
5.1.7 Compilation and dependencies ................................................................. 50
5.2 Server .............................................................................................................. 51
5.2.1 Overview of the Server ........................................................................... 51
5.2.2 Behaviour ................................................................. 51
5.3 A secondary database .................................................... 56
  5.3.1 Structure ............................................................. 56
  5.3.2 Access ............................................................... 57
6 Tests ............................................................................. 59
  6.1 On client side ............................................................ 59
  6.2 On Server side ........................................................... 61
7 Conclusion ....................................................................... 67
Annexes ........................................................................... 69
  Annexe 1 - The CMSI Database ............................................. 69
  Annexe 2 - Creating certificates .......................................... 71
  Annexe 3 - CMSI XML-based report ................................. 75
  Annexe 4 - PHP Application ............................................. 79
References ......................................................................... 79
1 Introduction

1.1 Overview

This project takes place in the evolution of the vulnerability management system ALTAIR, designed by esCERT-UPC, the computer security crew of the Polytechnic University of Catalonia. In order to unify European CERTs efforts, a new standard called CMSI for describing with accuracy software information was designed by the German CERT. It was not only aimed at making easier data exchange between those organisms, but also at enhancing conciseness of software vulnerability advisories.

The purpose of this project is to design a complete system composed by an agent, a server and a web application, that will finally dress an inventory of every piece of software installed on a machine. Its main value relies on its ability to produce CMSI XML reports. Given the CMSI database, the server will be able to take advantage of its data to generate a fully functional report. The agent should besides be multi platform, so that the solution could be applied on lots of workstations. Security the components will be emphasised, for example in ciphering communications between agents and server. The web application will moreover provide access to an administrator to manually check and change its configurations.

Nowadays the malware threat is so high that an administrator cannot ignore it. Pirates and hackers use all their knowledge to compromise machines, so leaving one unsecure even for a couple of hours could lead to a complete compromise of the network. Besides, bots and automated tools keep scanning the Internet for always more vulnerable hosts. Today it is assumed that a non updated machine connecting to the Internet will be fully compromised within less than 10 minutes. New vulnerabilities appear on a daily basis, and sometimes, it might take several days before releasing a patch, leaving administrators on their own.

That is why the administrator has to be daily informed on every vulnerability advisory and every update available for its software. But watching for those updates might require time, since most of the time they only apply to some specific versions of a piece of software. Consequently the administrator has to take its working time to sort information, read the bulletins, and then, if available, proceed to an update. And in large-scale networks, finding updates information for all the workstations may require a significant labour.

Sometimes, a vulnerability only concerns a specific system. The last flaw in OpenSSL random number generator was a perfect illustration of an inherent flaw that targets only a few systems, leaving the others safe. But such a case illustrates also that the administrator has to carefully pay attention to advisories. Not all versions are targeted, only a few ones. This leads him to devote more time to analyse those advisories that will finally not concern him.
esCERT developed the idea of realising a specific agent, that would analyse a system and report its contents to a remote server. By taking advantage of a new format for security advisories developed to enhance the cooperation of European CERTs, it would be possible for them to automate the detection of machines vulnerable to a given flaw.

1.2 esCERT-UPC
(http://escert.upc.edu/)

Composed by its director Manel Medina, it was created, as other European CERTs in the 90s, in 1994. Its purpose is to assist members of UPC University community in responding to computer incidents when they occur. One can send a report on an incident, and ask for a security analysis of this incident. The esCERT is also committed to proactively reduce the risk of computer security providing vulnerability alerts, proactive network scans, intrusion detection systems deployment and related measures.

It is authorized to address all types of computer security incidents which might occur, and its support will be adapted to the severity of the threat or the issue. In most case it will provide responses for the following cases:
- Investigating whether indeed an incident occurred.
- Determining the extent of the incident.
- Determining the initial cause of the incident.
- Facilitating contact with other sites which may be involved.
- Facilitating contact with UPC University Security.
- Making reports to other CSIRTs.
- Composing announcements to users.
- Removing the vulnerability.
- Securing the system from the effects of the incident.

When needed, the esCERT can perform some system maintenance along with an administrator, but most of the time it will only give him some information to solve the issue.

The esCERT is committed to keeping the UPC University system administration community informed of potential vulnerabilities, and when possible, will inform this community of such vulnerabilities before they are actively exploited. Distribution lists are available for such task.

1.3 ALTAIR
(http://escert.upc.edu/index.php/web/es/serv_altair.html)

Currently, Altair is a service developed by the esCERT to provide some services for administrators who want to be constantly aware of computer threats. Altair's first objective is to keep system administrators informed on new security related advisories that are daily published in order to protect them from any active exploitation and intrusion in their systems.

Each notification contains details on the vulnerability, on its severity in case of successful exploitation, on its possible workaround or on the way to update the system.
This vulnerability advisory system is actually highly reliable for system administrators due to its gathering of information and due to the amount of details provided.

1.4 The Common Model of System Information - CMSI
(http://www.cert-verbund.de/cmsi/en.html)

Currently, there is no universal format structure that implements a description of vulnerability advisories. Lots of types of vulnerability advisories exist, but they all tend to present a vulnerability under different aspects. Some of them list every piece of software vulnerable, with every single version and on the other hand, others embed a whole suite mentioning only versions under a threshold are vulnerable. These advisories sometimes contain additional details, referring to counter measures that might be deployed to avoid the threat, workarounds that might be taken into account to mitigate the impact of the flaw. There are also advisories in which are described the flaw, its provenance and every technical detail related to. Each time a modification (patch) is released for an application, lots of files with their checksums are then published. The user just has to select its own version, download and apply the patch. This method has been well known for administrators for years, but since they have a administrate lots of workstations, they are bound to know which software are installed on which machines which is really fastidious.

In order to improve the cooperation between European CERTs, the German CERT has developed a new model for representing those advisories. It soon appeared to them it was necessary to design a new schema to represent software data, and since no universal standard had been built yet, a new model called CMSI (Common Model of System Information) was then imagined. Its aim is to represent the contents of a system in a universal way: a tree whose branches would describe software and their properties. Its main purpose is to integrate consistency to an advisory, to report a flaw in such a way than nobody could misunderstand the targeted application. As a result, several rules might be set to coordinate those efforts. Application should for example be described using some common identifiers and their versions should follow regular expression rules like "version X.YZ" and not "v 6.0".

Having received an implementation of this model as a SQL data base (see Annexe 1), and I was charged to design an inventory system that would take advantage of this totally new concept, where applications installed in a machine would be recognized by their CMSI qualifiers.
2 Specifications

2.1 Overview

This system would be composed of three distinct modules, each of them responding to a different purpose:
- An agent, designed to create and report a list of installed software.
- A server, waiting for lists coming from agents and storing them into a database.
- A web application, which would help administrators to manually enter their own workstations and configurations. Besides, they could also manually consult or remove their own machines and software.

The following details refer to the specifications of the agent: the application should be a stand alone script, or an executable, and should be able to run without dependencies, so that the administrator’s job to install would be reduced to the absolute necessary. Moreover it has been specified that this tool must be able to run under several platforms, mostly Linux or Windows but also Solaris or AIX.

Once installed, it should be run autonomously in the background without requiring any administrator interaction. Under normal conditions, the report should be sent on a daily basis. Administrators who express the needs might be able to use this tool to create a local report and afterwards send it manually via email.

Confidentiality and integrity of the reports must be respected; therefore communications between hosts must be encrypted and signed. The web application also requires valid credential, so an administrator must have to logon successfully before managing its workstations and software. The server should be able to treat several connections at the same moment, and has to interact with a MySQL database.

2.2 Use cases

Given these specifications, simple use cases emerge. First, the agent has to be as much autonomous as possible. So it must be started automatically. The system thus plays a weighty role: start the agent. The administrator can also start it by himself, to generate just a local report. Then both administrator and system can load it. Configuration step is important too. In this step the administrator indicates the IP of the server, its port and the frequency of its execution.

Reports can be sent by both server and user. The user can send it by mail, in case of network issues of example, and of course server sends it automatically. This leads to the following graphic:
Concerning the web application, everything was specified above. Users can log, check their configuration and modifies it as they want:

Figure 1 - Agent's use case

Figure 2 - Web form's use case
From the server side, there is no specific use case. This application is started by the system and silently runs in background, so no use case diagrams was very necessary to illustrate such behaviour.

2.3 Sequence diagrams

The following two sequences show how agent's use cases could be implemented in simple steps. They illustrate the main functionalities of the agent: being automatically or manually started, retrieving a list of applications and sending it to the server or saving it onto the hard drive.

Figure 3 - Administrator runs its agent to get a list of its applications
Figure 4 - Agent connects and sends its report to server
The next diagram shows the corresponding tasks executed by the server once a report has been received. After having queried the CMSI database, it is able to create a CMSI report and update the database with the machine applications.

Figure 5 - Server receives a report and updates the database
Despite the PHP application seems to have lots of different functionalities, they all look like to each other, so there is no point is showing different sequence diagrams. Hereunder is described its behaviour when a user decides to remove one of its machines. After having logged on, he uses the remove machine functionality, selects a machine and simply removes it. All these operations require an interaction with a data base, illustrated as a third actor.

![Diagram](image)

Figure 6 - Administrator removes a machine from the web application
3 Preliminary study

3.1 Analysis of existing tools

There are several different existing tools that can produce a full system inventory. Some are free and open sourced, others are distributed as shareware or even require the user to pay before being able to download them. Among the different open source tools that have been designed to manage computer inventories, five of them seem to be really interesting for us. Hereunder is shown the list of the one I tested:

H-Inventory – http://www.h-inventory.com/
ManagePC – http://managepc.net/
Open-AudIT – http://www.open-audit.org/
zCI Computer Inventory System – http://zci.sourceforge.net/
OCS Inventory NG – http://www.ocsinventory-ng.org/

They all have their own ways to conduct an inventory. Some targets only a few kinds of systems whereas other privileges to be compatible with lots of architectures. Their reports also differ. Some produces a report with only a list of known applications (they check if a known application is installed on the host) and other do a full inventory, including a whole list of software, their description, version, additional notes, and even the hardware configuration of the machine.

3.1.1 H-Inventory

Composed by a web server running on Apache and taking advantage of PHP and MySQL, and agents that can run on both Windows, Linux, Solaris and BSD systems, this application has the main characteristics of an administration tool. The administrator in charge of its workstations can have a detailed insight of its machine. The functionalities covered by the agent are far-reaching. It can retrieve lots of hardware information such as processor’s characteristics, the Bios manufacturer, memory and hard drive information and even a full description of the IDE controllers. In term of software the agent audits the system and gives a detailed report of every piece of software installed. On Windows and Linux machines, it gives the detail description of the kernel (version and service pack for Windows, and distribution name for Linux), and also provides a complete list of every package present in the system. It is besides possible to know information such as the serial key entered during the installation of Windows, the time since the system has been started, or the user whose session is currently active.

The application gives the administrator the ability to inspect statistics, for example a percentage of the kind of Operative System (OS) currently used in the company, but lets him as well organizing a monitoring of the applications, and provides a perfect interface to manage incidents. It eventually provides an interesting way to isolate a machine according to its particularities (installed software, memory, etc.).
1 Introduction

1.1 Overview

This project takes place in the evolution of the vulnerability management system ALTAIR, designed by esCERT-UPC, the computer security crew of the Polytechnic University of Catalonia. In order to unify European CERTs efforts, a new standard called CMSI for describing with accuracy software information was designed by the German CERT. It was not only aimed at making easier data exchange between those organisms, but also at enhancing conciseness of software vulnerability advisories.

The purpose of this project is to design a complete system composed by an agent, a server and a web application, that will finally dress an inventory of every piece of software installed on a machine. Its main value relies on its ability to produce CMSI XML reports. Given the CMSI database, the server will be able to take advantage of its data to generate a fully functional report. The agent should besides be multi platform, so that the solution could be applied on lots of workstations. Security the components will be emphasised, for example in ciphering communications between agents and server. The web application will moreover provide access to an administrator to manually check and change its configurations.

Nowadays the malware threat is so high that an administrator cannot ignore it. Pirates and hackers use all their knowledge to compromise machines, so leaving one unsecure even for a couple of hours could lead to a complete compromise of the network. Besides, bots and automated tools keep scanning the Internet for always more vulnerable hosts. Today it is assumed that a non updated machine connecting to the Internet will be fully compromised within less than 10 minutes. New vulnerabilities appear on a daily basis, and sometimes, it might take several days before releasing a patch, leaving administrators on their own.

That is why the administrator has to be daily informed on every vulnerability advisory and every update available for its software. But watching for those updates might require time, since most of the time they only apply to some specific versions of a piece of software. Consequently the administrator has to take its working time to sort information, read the bulletins, and then, if available, proceed to an update. And in large-scale networks, finding updates information for all the workstations may require a significant labour.

Sometimes, a vulnerability only concerns a specific system. The last flaw in OpenSSL random number generator was a perfect illustration of an inherent flaw that targets only a few systems, leaving the others safe. But such a case illustrates also that the administrator has to carefully pay attention to advisories. Not all versions are targeted, only a few ones. This leads him to devote more time to analyse those advisories that will finally not concern him.
esCERT developed the idea of realising a specific agent, that would analyse a system and report its contents to a remote server. By taking advantage of a new format for security advisories developed to enhance the cooperation of European CERTs, it would be possible for them to automate the detection of machines vulnerable to a given flaw.

1.2 esCERT-UPC
(http://escert.upc.edu/)

Composed by its director Manel Medina, it was created, as other european CERTs in the 90s, in 1994. Its purpose is to assist members of UPC University community in responding to computer incidents when they occur. One can send a report on an incident, and ask for a security analysis of this incident. The esCERT is also committed to proactively reduce the risk of computer security providing vulnerability alerts, proactive network scans, intrusion detection systems deployment and related measures.

It is authorized to address all types of computer security incidents which might occur, and its support will be adapted to the severity of the threat or the issue. In most case it will provide responses for the following cases:
- Investigating whether indeed an incident occurred.
- Determining the extent of the incident.
- Determining the initial cause of the incident.
- Facilitating contact with other sites which may be involved.
- Facilitating contact with UPC University Security.
- Making reports to other CSIRTs.
- Composing announcements to users.
- Removing the vulnerability.
- Securing the system from the effects of the incident.

When needed, the esCERT can perform some system maintenance along with an administrator, but most of the time it will only give him some information to solve the issue.

The esCERT is committed to keeping the UPC University system administration community informed of potential vulnerabilities, and when possible, will inform this community of such vulnerabilities before they are actively exploited. Distribution lists are available for such task.

1.3 ALTAIR
(http://escert.upc.edu/index.php/web/es/serv_altair.html)

Currently, Altair is a service developed by the esCERT to provide some services for administrators who want to be constantly aware of computer threats. Altair's first objective is to keep system administrators informed on new security related advisories that are daily published in order to protect them from any active exploitation and intrusion in their systems.

Each notification contains details on the vulnerability, on its severity in case of successful exploitation, on its possible workaround or on the way to update the system.
This vulnerability advisory system is actually highly reliable for system administrators due to its gathering of information and due to the amount of details provided.

1.4 The Common Model of System Information - CMSI
(http://www.cert-verbund.de/cmsi/en.html)

Currently, there is no universal format structure that implements a description of vulnerability advisories. Lots of types of vulnerability advisories exist, but they all tend to present a vulnerability under different aspects. Some of them list every piece of software vulnerable, with every single version and on the other hand, others embed a whole suite mentioning only versions under a threshold are vulnerable. These advisories sometimes contain additional details, referring to counter measures that might be deployed to avoid the threat, workarounds that might be taken into account to mitigate the impact of the flaw. There are also advisories in which are described the flaw, its provenance and every technical detail related to. Each time a modification (patch) is released for an application, lots of files with their checksums are then published. The user just has to select its own version, download and apply the patch. This method has been well known for administrators for years, but since they have a administrate lots of workstations, they are bound to know which software are installed on which machines which is really fastidious.

In order to improve the cooperation between European CERTs, the German CERT has developed a new model for representing those advisories. It soon appeared to them it was necessary to design a new schema to represent software data, and since no universal standard had been built yet, a new model called CMSI (Common Model of System Information) was then imagined. Its aim is to represent the contents of a system in a universal way: a tree whose branches would describe software and their properties. Its main purpose is to integrate consistency to an advisory, to report a flaw in such a way than nobody could misunderstand the targeted application. As a result, several rules might be set to coordinate those efforts. Application should for example be described using some common identifiers and their versions should follow regular expression rules like "version X.YZ" and not "v 6.0".

Having received an implementation of this model as a SQL data base (see Annexe 1), and I was charged to design an inventory system that would take advantage of this totally new concept, where applications installed in a machine would be recognized by their CMSI qualifiers.
2 Specifications

2.1 Overview

This system would be composed of three distinct modules, each of them responding to a different purpose:
- An agent, designed to create and report a list of installed software.
- A server, waiting for lists coming from agents and storing them into a database.
- A web application, which would help administrators to manually enter their own workstations and configurations. Besides, they could also manually consult or remove their own machines and software.

The following details refer to the specifications of the agent: the application should be a stand alone script, or an executable, and should be able to run without dependencies, so that the administrator’s job to install would be reduced to the absolute necessary. Moreover it has been specified that this tool must be able to run under several platforms, mostly Linux or Windows but also Solaris or AIX.

Once installed, it should be run autonomously in the background without requiring any administrator interaction. Under normal conditions, the report should be sent on a daily basis. Administrators who express the needs might be able to use this tool to create a local report and afterwards send it manually via email.

Confidentiality and integrity of the reports must be respected; therefore communications between hosts must be encrypted and signed. The web application also requires valid credential, so an administrator must have to logon successfully before managing its workstations and software. The server should be able to treat several connections at the same moment, and has to interact with a MySQL database.

2.2 Use cases

Given these specifications, simple use cases emerge. First, the agent has to be as much autonomous as possible. So it must be started automatically. The system thus plays a weighty role: start the agent. The administrator can also start it by himself, to generate just a local report. Then both administrator and system can load it. Configuration step is important too. In this step the administrator indicates the IP of the server, its port and the frequency of its execution.

Reports can be sent by both server and user. The user can send it by mail, in case of network issues of example, and of course server sends it automatically. This leads to the following graphic:
Concerning the web application, everything was specified above. Users can log, check their configuration and modifies it as they want:
From the server side, there is no specific use case. This application is started by the system and silently runs in background, so no use case diagrams was very necessary to illustrate such behaviour.

2.3 Sequence diagrams

The following two sequences show how agent's use cases could be implemented in simple steps. They illustrate the main functionalities of the agent: being automatically or manually started, retrieving a list of applications and sending it to the server or saving it onto the hard drive.

![Sequence Diagram]

Figure 3 - Administrator runs its agent to get a list of its applications
Figure 4 - Agent connects and sends its report to server
The next diagram shows the corresponding tasks executed by the server once a report has been received. After having queried the CMSI database, it is able to create a CMSI report and update the database with the machine applications.

Figure 5 - Server receives a report and updates the database
Despite the PHP application seems to have lots of different functionalities, they all look like to each other, so there is no point is showing different sequence diagrams. Hereunder is described its behaviour when a user decides to remove one of its machines. After having logged on, he uses the remove machine functionality, selects a machine and simply removes it. All these operations require an interaction with a data base, illustrated as a third actor.

Figure 6 - Administrator removes a machine from the web application
3 Preliminary study

3.1 Analysis of existing tools

There are several different existing tools that can produce a full system inventory. Some are free and open sourced, others are distributed as shareware or even require the user to pay before being able to download them. Among the different open source tools that have been designed to manage computer inventories, five of them seem to be really interesting for us. Hereunder is shown the list of the one I tested:

- H-Inventory - http://www.h-inventory.com/
- ManagePC - http://managepc.net/
- Open-AudIT - http://www.open-audit.org/
- zCI Computer Inventory System - http://zci.sourceforge.net/
- OCS Inventory NG - http://www.ocsinventory-ng.org/

They all have their own ways to conduct an inventory. Some targets only a few kinds of systems whereas other privileges to be compatible with lots of architectures. Their reports also differ. Some produces a report with only a list of known applications (they check if a known application is installed on the host) and other do a full inventory, including a whole list of software, their description, version, additional notes, and even the hardware configuration of the machine.

3.1.1 H-Inventory

Composed by a web server running on Apache and taking advantage of PHP and MySQL, and agents that can run on both Windows, Linux, Solaris and BSD systems, this application has the main characteristics of an administration tool. The administrator in charge of its workstations can have a detailed insight of its machine. The functionalities covered by the agent are far-reaching. It can retrieve lots of hardware information such as processor's characteristics, the BioS manufacturer, memory and hard drive information and even a full description of the IDE controllers. In term of software the agent audits the system and gives a detailed report of every piece of software installed. On Windows and Linux machines, it gives the detail description of the kernel (version and service pack for Windows, and distribution name for Linux), and also provides a complete list of every package present in the system. It is besides possible to know information such as the serial key entered during the installation of Windows, the time since the system has been started, or the user whose session is currently active.

The application gives the administrator the ability to inspect statistics, for example a percentage of the kind of Operative System (OS) currently used in the company, but lets him as well organizing a monitoring of the applications, and provides a perfect interface to manage incidents. It eventually provides an interesting way to isolate a machine according to its particularities (installed software, memory, etc.).
As it was mentioned above, the server takes advantage of PHP to give a full administration board to the administrator. PHP provides interactivity and a user friendly interface that reveals to be a determining criterion when looking for a management application. Agents are entirely written in script language, VB Script for Windows and Shell Script for UNIX systems. Reports are generated in xml form, and can be inspected by using our favourite web browser. Five different modes are offered to transfer a report to the server: ftp, http, samba, network shares and scp. They will all be discussed later and reviewed according to their security, their deployments on several systems (installed by default) and the complexity to implement them if necessary.

Eventually, it is possible to elect "local" mode where reports would be stored locally and need to be manually sent by a user. The configuration is stored in hinventory.conf, and this file is accessed each time a script is run. In accordance with the platform the agent is executed on, it is unfortunately not possible to benefit of all these transfer modes. And finally, it must be noted that some modes require an additional package that needs to be separately downloaded.

Example:

```bash
### Upload HTTP (for OpenBSD scripts)
http
  echo "HTTP upload not implemented yet."
::
FreeBSD: "For HTTP Method, you must install curl package"
```

One could also regret that there is no universal script for UNIX platforms. User has to choose which one fits with its system, before making a report. Information reported might sometimes be erroneous, as it was the case with my test machine. Report showed that the machine was run by Debian, while Ubuntu was the current distribution.

3.1.2 ManagePC

This tool is completely different from the others. Originally developed as a HTA script, it is now based on the .NET technology (Visual Basic .Net). Compatible only with Windows systems, it however provides a full management service. Besides than providing the basic functionalities of an inventory application, it also lets an administrator to remotely administrate a computer. VNC and Remote Desktop features allow him to control and check in real conditions the behaviour of the machines. Reports are given in three different formats: RTF, HTML and XLS and the application can be configured to use a LDAP active directory to register machines.

Reports are very complete like the other tools, but they also display characteristics like active processes, and, a more interesting thing, the services run by Windows during the start up. Despite its incompatibility with UNIX systems, the main problem comes from the size of the executable which is more than 26 MB, making it too heavy to be widely distributed in a huge company.
3.1.3 Open-AudIT

As H-Inventory, Open-AudIT embeds a web application in PHP with MySQL and its agents are written in scripts: Shell Script for Linux and VB script for Windows. The main frame lets someone who has access to consult the configuration of every machine present on the network. Unlike H-Inventory, there is no login page, which makes everyone able to browse the site.

Agents behave however slightly differently that H-Inventory’s since they do not reference every application installed on a Linux system. They only look for a list, and when querying the package repository, it compares the entry selected in its list to the available packages. Therefore, the report cannot be considered thorough because some additional packages could be installed and not spotted. By default, here is the list of the targeted packages (taken from auditUbuntu.sh):

```
packages="apt azureus bash build-essential cedparanoia cdrecord cpp cron cupsys cvs dbus dhcp3-client diff dpkg epiphany-browser esound evolution firefox flashplugin-nonfree foomatic-db g++
gaim gcc gdm gedit gimp gnome-about gnucash gnumeric gtk+ httpd inkscape iptables k3b kdebase koffice libgnome2-0 linux-image-386 metacity mozilla-browser mysql-admin mysql-query-browser
mysql-server-4.1 nautilus openssh-client openssh-server perl php4 php5 postfix postgresql python python2.4 rdesktop rhythmbox samba-common sendmail smbclient subversion sun-j2re1.5 swf-player synaptic thunderbird tsclient udev vim vlc vnc-common webmin xfe xmmms xserver-xorg"
```

Although this list is not complete, it represents the most critical applications that could be targeted by an attacker. Those are very common, so they must be carefully analysed. Besides, this way to retrieve applications has another advantage. On a standard Linux distribution, tons of packages are installed, so the list is so huge that it becomes considerably difficult to have a good insight of the system components. The agents have therefore been designed to solve this problem.

Only three different agents are available by default:

- Fedora
- Ubuntu
- Windows

Consequently administrators who run OS such as Solaris might not opt for this solution to address a complete inventory. However, this lack of OS checks might be compensated by other functionalities, like auditing a full network to retrieve also the characteristics of printers and additional resources.

Reports are generated using HTML pages. Yet, it seems impossible with a default installation to transfer automatically a report to a server, which makes this solution less adapted for distributed and automated tasks.

3.1.4 zCI Computer Inventory System

Like the other ones, zCI is presented as a web application handling PHP and MySQL. Users can also be pleased to try it safely without needing to do a full
installation since a free demo has been created for Windows users (people have
to download and execute the application). It differs a lot from the others due to
the script language adopted for agents. On Windows, the agent has been written
in Java Script, and UNIX based agents are powered by Java. I did not personally
test it on Windows 95 or 98, but it is explicitly signified in the documentation that
MS Windows Scripting Host 5.6 is needed to run JavaScript files. By default, this
is available in Windows versions newer than Windows 2000, so nowadays
administrators should not suffer from this incompatibility (the package is also
freely downloadable on Microsoft web site).
Only one solution of transfer has been adopted: http. The configuration takes
place in the script itself:

```
var zcipath="http://localhost/zci";
```

A more detailed study shows that the script takes advantage of various
ActiveX objects:

```
var myshell = WScript.CreateObject("WScript.Shell");
var fs = new ActiveXObject("Scripting.FileSystemObject");
zcitemp.WriteLine("<html><body><form id=myform method=post
action=""+zcipath+"engine/readdata.php!">;
[…]
var objLocator = new ActiveXObject("WbemScripting.SWbemLocator");
[…]
var objService = objLocator.ConnectServer("","root/cimv2");
instances = objService.ExecQuery("SELECT Version,Vendor,Name,IdentifyingNumber FROM Win32_ComputerSystemProduct");
```

Installed applications are found by exploring the sub registry key defined
below:

```
mainpath="SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall"
```

and the form in finally sent through the InternetExplorer activeX object:

```
var myie=new ActiveXObject("InternetExplorer.Application");
```

It is also important to note that computer hardware information is retrieved
by the method ExecQuery which is working with SQL queries. As a consequence
retrieving those data appears to be really easy. There is no call to complicated
functions demanding additional parameters. The third line underlies the fact that
HTTP POST method is used to save data. So the report is carried out over the
network to finally be considered as a variable by the PHP application.

Java implementation is also interesting to analyse. The script uses system
commands to fill its inventory:

```
Runtime rt = Runtime.getRuntime();
Process proc = rt.exec("lshw"); <- list hardware configuration
```

One could nevertheless regret that some instructions are too complicated to
be directly understood by a programmer:

```
if (myval[108]==0 && myval[109]==0 && myval[110]==0 && (myval[111]==0xFC || myval[111]==0xFF) && myval[112]==0)
```
In such a comparison, regular expressions for example developed in Perl, are a great workaround.

The report is then eventually sent through a socket. Actually it works without any HTTP additional controls, and only with hard coded sockets:

```perl
Socket socket = new Socket(InetAddress.getByName(zciserver), zciserverport);
```

The analysis of those two agents was very interesting due to their implementations. Among the five applications analysed here, it was the only one whose agents were written in JavaScript and Java making it unique and unprecedented in its designing.

### 3.1.5 OCS Inventory New Generation

By far this tool was the most useful, complete and user friendly of the five. It is also composed by a web server with PHP and MySQL, but provides agents for many systems. Some of them have been developed by the community of active users and testers, and so it makes it resistant to lots of potential troubles. The web application lets an administrator manage its whole workstations, like in the other tools, but also lets him coordinate its networks, automatically deploy new packages on machines, check statistics, check for redundancy (if elements of one machine exist are present in another). Configuring this is so easy that finally it is impossible to opt for another tool than this. For instance, if one wants to distribute an update, a new application or a new package to its workstations, he just have to prepare the packet, select the network where he wants stations to receive it and program the scheduler. Distribution will be absolutely automatic, and users can even not figure something changed on their computer. There is no nuisance like a need to warn users that they must leave their station free at a certain time because in such case, agents would grab the packet as soon as they can. They connect to the server and then check if something is available.

For Linux, agents have been written in Perl (ocsinventory-client.pl). It requires some additional packages that may already exist on the UNIX distribution, but even if they have not been installed by default, the package installation contains them. So there is no need to focus on downloading dependencies.

```perl
use XML::Simple;
use LWP::UserAgent;
use Compress::Zlib;
use Net::IP qw(PROC);
use Digest::MD5 qw(md5_base64);
use Ocsinventory::Agent::Common qw::all;
use Fcntl qw:flock;
```

Unfortunately, it requires root credentials, or at least some sufficient privileges to be executed, though it can be configured once for all to run at the system startup.

```perl
# Checking Permissions
unless(-r "/dev/mem"){
    die localtime()." => You don't have enough rights to run this program\n";
}
```
As for H-Inventory's agent, it performs then a full review of the system characteristics, listing hardware and system configurations. But unlike H-Inventory, only one agent has been designed to be run on different versions of Linux. Therefore the following part of the code is perhaps the most important one because it determines the type of system:

```perl
# Which distro?
if(-e "/etc/debian_version"){
    $distro = 'Debian';
    $distro_file = "/etc/debian_version";
}elsif(-e "/etc/mandrake-release"){
    $distro = 'Mandrake';
    $distro_file = "/etc/mandrake-release";
}elsif(-e "/etc/redhat-release"){
    $distro = 'Redhat';
    $distro_file = "/etc/redhat-release";
}elsif(-e "/etc/SuSE-release"){
    $distro = 'SuSe';
    $distro_file = "/etc/SuSE-release";
}elsif(-e "/etc/slackware-version"){
    $distro = 'Slackware';
    $distro_file = "/etc/slackware-version";
}elsif(-e "/etc/knoppix_version"){
    $distro = 'Knoppix';
    $distro_file = "/etc/knoppix_version";
}elsif(-e "/etc/fedora-release"){
    $distro = 'Fedora Core';
    $distro_file = "/etc/fedora-release";
}elsif(-e "/etc/trustix-release"){
    $distro = 'Trustix';
    $distro_file = "/etc/trustix-release";
}elsif(-e "/etc/gentoo-release"){
    # D LIROULET 2006/08/31 Gentoo Support
    $distro = 'Gentoo';
    $distro_file = "/etc/gentoo-release";
}
```

As one can see, nine versions of Linux distributions are recognised, which makes it the most useful agent among the tools (Note: extra compatibility for BSD systems has been developed as single part agents, but the newest version supports almost all of the common distributions). Later in the code, a second distinction is done according to the distribution:

```perl
if($distro =~/debian\knoppix/){
    my $dpkg_path = "/get_path('dpkg');
    ...
}elsif($distro =~/mandrake\redhat\suse\fedora\trustix/){
    my $rpm_path = "/get_path('rpm');
    ...
}elsif($distro =~/gentoo/){
    # D LIROULET 2006/08/31 Gentoo support with "eqquery"
    my $eqquery_path = "/get_path('eqquery');
```
This part of the code determines what application retriever to use in accordance with the distribution (This theme will be discussed later).

To conclude, the report is transferred through a HTTP POST request after having been compressed. This last part can however be discussed because reports whose size is larger than 50 KB are quite rare, and compression requires a supplementary package (although Zlib is relatively often found on default installations). On the other hand, if 200 machines start to send at the same time their reports, it could consume a bandwidth up to 200*50 KB = 10 MB. For small companies, such a problem cannot be ignored.

```php
###
# HTTP
###
$req = HTTP::Request->new(POST => $URI);
$req->header('Pragma' => 'no-cache', 'Content-type' => 'application/x-compress');
&debug($message, 'SENDING') if $debug and $debug>1;

$message = Compress::Zlib::compress($message) or die localtime(" => Probleme de compression(prolog)n";

$req->content($message);
```

Another theme that has not really been explored yet concerns the start up of the program. As stated above it needs root privileges to be run, so the most efficient method to do so is to create a new Cron entry. Cron jobs are configured in `/etc/cron.d` and consist of scheduled tasks that the system will automatically execute. It is the safest way to ensure a server will run the agent.

```
# Launch OCS Inventory NG Agent once a day under root account, at HH:MM
#
MM HH *** root /usr/bin/ocsinventory-agent > /dev/null 2>&1
```

A secondary method is to add a script to `init.d` directory in such a way that the agent will be run when system starts up. Assuming that the current workstation belongs to a user and is not a server, it might be nicer to specify it to run on start up than to wait for a specific hour.

```
/etc/init.d/
#! /bin/sh
### BEGIN INIT INFO
# Provides:      ocs-inventory-agent
# Required-Start: $local_fs $remote_fs
# Required-Stop: $local_fs $remote_fs
# Default-Start: 2 3 4 5
# Default-Stop:  0 1 6
# Short-Description: OCS-Inventory Agent
# Description:   This script launches OCS Agent as daemon
### END INIT INFO
```

The last version available (released on May, 1st) is even more competitive due to the recognition of new types of systems.
3.1.6 Conclusion

Reviewing all those possibilities has considerably expanded the possibilities to develop the agent / server system. This analysis has revealed that lots of languages could be considered, and choosing one instead of another relies just on the fact that one may use functionalities that cannot be often found on default configurations. The fewer packages are needed to be installed, the better is the client module. Above all, the question how to fingerprint a system has been resolved due to the wide range of options that have been exposed here.

Could one of these applications be considered as a candidate to this project? Although they all report system information and software lists they do not exactly fit with our requirements. Each of those tools provides also hardware information that is absolutely useless for us. They have been designed in order to satisfy the needs of an administrator responsible for numerous workstations, but however, from our point of view lots of the information reported can be deemed as extra. Why should we care about the serial number of this version of Windows for instance? Nonetheless, the code of the application could be adapted to fit with our requirements. Contemplating this possibility would then lead to think about redesigning their database, and obviously completely redesigning the web application. In the end, a very different program that has nothing to do with the original would be created. Furthermore, none of those tools uses a satisfying policy in term of security, save maybe H-Inventory that offers to upload with scp.

It may also be noted that no agent tries to retrieve the language of the OS. And finally, the CMSI compatibility seems to need further changes, to generate among others a CMSI XML report (discussed later).

One may have noticed, but Mac case has not been much developed through this analysis. Actually only one application had a Mac support, which was still in beta tests. I asked one developer of H-Inventory to have a glance. They decided that a PHP script would be the best way to fit with Mac exigencies. Their script can be run manually or automatically, needs root privileges and the transfer mode is again based upon HTTP POST method.
3.2 How to transfer the report?

The analysis of the existing tools have illustrated that several different methods can be employed to transfer the report to the server. The machines where agents are run do not necessarily form part of the local network. They might be placed in another network that does not share its resources. Therefore security over network remains the main problem.

Here follows a description of every method that has been noticed in the analysis.

3.2.1 FTP (File Transfer Protocol)

The File Transfer Protocol was created very early to transfer files from peer to peer. Typically, a server waits for incoming connections from a client on port 21 and then starts to dial with it. Commands are passed through a control channel, and files are transferred via a port that client and server determine. Provided an account (login, password), clients can access the server and start to upload or download files. FTP is widespread, because at least a client program exists on every system, so the method is commonly employed. Nevertheless, when the description of the protocol was imagined, it specified that username and password would be sent in plain text, that is to say, without any encryption. Therefore, anyone who sniffs data on a network might intercept those credentials and then be able to usurp identity of a valid user. Moreover, it is not possible to choose any encryption system other the data channel. Thus data might be easily intercepted. Furthermore data transfer channel works over a specified channel whose ports are defined by both server and client. As a consequence it is quite complicated to write rules on a Firewall to allow those kinds of transfers. (Let us remember that the server that will receive the report may be exposed to Internet, so the more ports are closed, the safer it is).

3.2.2 SFTP (SSH File Transfer Protocol) and FTPS (FTP over SSL)

These protocols were imagined in order to solve the security problems exposed above in FTP. They rely on an implementation of secure methods to transmit passwords and data over the network. SFTP can be commonly encountered in UNIX distributions because it is an inner part of OpenSSH. Like SCP (see below), it creates a secure SSH tunnel where client and server can then use the FTP protocol over the control channel. Data are then encrypted and sent through the network without any risk of leak.

FTPS is slightly different because it describes the implementation of FTP through SSL. Like FTP, the server listens on port 21, but if a client specifies AUTH TLS, then the connection becomes encrypted. The server needs a public signed certificate that will be sent to the client before any encryption process. Then data will be enciphered using the public key contained in the certificate and transmitted to the remote host. The weakness of this protocol is that, like ftp,
several other connections might be opened to transfer files. And as the control connection might be enciphered, firewalls cannot guess which port will be used to transfer data so they might drop the connection. Besides, unlike SFTP, FTPS implementations are rarer in a UNIX environment. On the test machine there were not any FTPS client or server installed by default, and as a rule, the same goes for Windows.

3.2.3 RCP and SCP (Remote Copy and Secure Copy)

RCP was created to make easier the file transfer across UNIX platforms and have been widely replaced by its new version, scp. On my test machine, rcp is a shortcut to scp:

```
which rcp: /usr/bin/rcp
ls -l /usr/bin/rcp: /usr/bin/rcp -> /etc/alternatives/rcp
ls -l /etc/alternatives/rcp: /etc/alternatives/rcp -> /usr/bin/scp
```

Embedded in UNIX systems this command is used for copying files from a location to another relying on the Secure Shell (SSH) protocol on port 22. Basically, client and server negotiate the connection specifying the algorithm that will be used to securely transfer files (handshake) and then proceed to the transfers. The main difference between FTP is that file attributes are kept during the transfer. Consequently, if the file to transfer can only be read by root, then so will go for the destination.

```
scp SourceFile.ext user@host:directory/TargetFile.ext
scp user@host:folder/SourceFile.ext TargetFile.ext
```

Unlike UNIX systems, no SCP client exists on default Windows installations, which brings again the problem to install a new one or else program one.

3.2.4 Network Share

Network Shares have been developed to make easier for users the access to network shared resources such as printers, scanners or even shared directories. When a resource is shared, anybody that has sufficient credentials can profit of these resources like if they were directly available on its environment. For example, if the machine testmachine shares its folder c:\shared, it is possible for Windows users to mount the directory and access by a simple net command: net use z: \testmachine\shared. Thus any file or directory present in shared\ will be available to others. This technology relies on the SMB protocols. For UNIX machines, Networks Shares are realised by the Network File System (NFS) protocol. This job is basically performed by a NFS daemon, such as nfsd. Shared directories are typically written as a list in the file /etc/exports and the command exportfs sets up the environment so they can be recognised as shared resources:

```
# repertoire liste-machines (liste-options)
/home/ pc2(ro) pc3(rw)
/usr/bin pc2(ro) pc3(ro)
/var/www/html *.upc.edu(ro) pc3(rw)
```
On the client side, those shares are accessible using the mount command:
```
mount -t nfs servername:/path_to_shared_folder /path_to_mountpoint
```

This kind of share is extremely powerful, because no specific commands (like ftp or scp) need to be entered before accessing the resources and moreover, this is perhaps the most interesting point, the share protocol is entirely managed by the operating system. In other words, once the share folders have been mounted, it is possible to take advantage of them as if they already existed in the machine. Therefore a simple cp (copy on Windows) lets a user copy any file he wants to the network share. From the point of view of the server that will receive reports, it could be a good solution to leave a public access to a folder, and later inspect it to find whether reports have been put. Consequently there would not be any contact between the server and the agent. They should just upload their report onto the shared directory like a postman drops his letters and the server should periodically checks if the letter box is empty or not. However, some inauspicious consequences may appear due to such permission. Given that anybody can upload anything on a shared directory, some troubles must be considered. Someone with malicious intention could for example upload tons of file making the server crash while trying to analyse all those files. All the resources of the system could be consumed leaving it into some denial of service conditions. Another problem would concern the size of the uploaded file. If one can send whatever he wants, he can fill the whole space left on the repository making useless the attempts of other agents to send their report. Consequently the service could be interrupted until someone notices that one is trying to harm. A workaround would be to constantly check the size of the files put into the directory and raise a warning if the size of a file reaches a certain threshold. But in a certain way it would require a constant check from an administrator, leaving the server under a mitigated autonomy.

### 3.2.5 SMB (Server Message Block)

This protocol was the one used by default on Windows to share resources. Its functionalities are approximately the same as the Network Shares. It works as a peer to peer network where clients send their specific requests to servers. By default, Linux kernels have their own clients implemented and the technique is quite popular for users who want to access shares in a local network and even across the Internet.

### 3.2.6 SOAP (Simple Object Access Protocol)

This method consists in sending XML based messages via Hypertext Transfer Protocol (HTTP) requests. As a result, these requests are formed by ASCII characters and its use is definitely adapted to send text commands to a host. Nevertheless, it has not been designed for file transfers, but in our case, as a software report could be XML based, this technology may have good fallouts. Moreover, since the requests are made in HTTP, firewalls do not block the traffic,
so this kind of solution might be preferred if security measures are strict. For us, it would require agents to generate XML reports, which cannot be realised because of it is not considerable to let them query the CMSI database.

3.2.7 HTTP upload

All of the examined tools embed a web application to make easier the administration task of the workstations. There are more oriented to the administration task than our needs but despite this point they offer a real solution to send files to a server. Basically, the web application is run by PHP and holds a special page dedicated to receiving those reports. How does it work?

Agents are really autonomous and carry a small piece of code aimed at connecting to the web server, and through a HTTP request ask for uploading their files. This request is embedded by the PHP post method: the server receives the content of the report as if it was only a variable. The pros of this method are that every connection is processed via the HTTP port 80. Therefore, it is as if the agent acted as a web browser like Firefox, and benefited of HTTP protocol's features to drop its report. Moreover since it uses a HTTP connection to put its data, there is no need to configure firewalls, because they only see HTTP traffic, which is most of the time left open. Nevertheless, none of those tools were offering a real data protection in term of ciphering data. But it could as well be imagined that the Apache web server would be loaded with a certificate and MOD_SSL, and in such case, reports could be sent through the https (HTTP over SSL) protocol over the port 443. As a result it would imply that agents carry also an HTTPS module to perform the transfer, which might be more complicated to find. Anyway in all cases, the server would be a web application written in PHP holding a specific feature designed to grab files from agents using the HTTP protocol. Such a way to proceed can then be discussed, because it may not be necessary to run a web server just for receiving files. Unless some user interactions like browsing a data base or modifying data are needed, designing a web application just for this purpose might be too much. However, the consideration of dealing with the CMSI database with SQL requests written in PHP might carry a lot of weight. PHP is indeed designed to make the interaction with MySQL really easy for a developer so this perspective should comfort the choice to realise it.

3.2.8 Email

Perhaps the simplest idea that would come to anyone's mind! Obviously, when it is about to deal with text file, emailing has appears to be in good position compared to other competitors. The method described here places suggests to install or develop a mail daemon on the server side, that will receive every report. As it will be referred later, OpenSSL's functionalities implement everything needed to write a mail in a secured way. But sending this mail might require more skills, because the agent would need to use a mail agent, that most of the time has not been configured in a machine. There is no point in configuring an email agent on a VPN server. And designing our own mail agent
might also require some additional labour that could be avoided with other methods. And this strategy also looks like the one presented by the Network Share solution in the way that people animated with malicious attention could try to spam and fill out the mail server. Presented under these circumstances, emailing seems to lose many of its claimed assets. Nonetheless, as emailing is something natural for a user, the report could be written locally and later the administrator could send it the esCERT. This case could appear for instance in case of a network failure, where it is completely impossible to send automatically a report to a server, that is to say if it was unreachable.

In the tools tested, one had its agent written in Shell Script, it only generated an HTML file that could later be consulted by the administrator of the audited machine. In this specific case, the administrator was free to choose which method was the best to send the report. So emailing could be definitely a good one.

3.2.9 Telnet, Netcat or any socket method

This method has not been discussed yet, but is employed in a few tools. Some of them have implemented their own socket listener to negotiate incoming connections. Thus, the server has all the control of the flow. The protocol does not exist, so there is no need to follow any rule. Data sent with this method are called raw data. There is no encapsulation, like there is for example in FTP or HTTP. The server acts as it was in passive mode, receives a flow of data without any need to parse it or finding any header. It just processes it and does not leave any response to the agent. The connection is then one-sided since the agent is the only one active and sends data. In term of implementation on the server side, this process may be the simplest: accepting a connection, looping and writing to a variable or to a file until all bytes have been sent and closing the connection. On the agent side, it is also reduced in three steps: initiating a connection, looping while all data have not been sent and finally closing connection. Unfortunately, reality appears slightly different due to the lack of implementations of sockets in scripts. Shell scripts do not handle any socket descriptors that could do the job, so it is necessary to base this part of the code on a secondary command, like nc (netcat) or telnet. But luckily Perl and Python are most of the time provided with additional libraries that provide a full access to sockets for the developer. In Python, import socket let the developer take advantage of them while IO::Socket gives access to in Perl. The lack of socket implementations in Shell Script is balanced by several commands that do the job. telnet ip 8080 will try to connect to ip:8080 in text mode. It means that once the connection has been created, text is passed when the input contains a carriage return. Thus, it is possible to transfer ASCII text files from one host to another. Its leverage also comes from the fact that is tool has been extremely popular. Up to two years ago, every UNIX or Windows distributions carried a telnet client, but so far, it is considered as an unsafe tool. SuSE distributions have ceased to propose it on default installation. Consequently, it becomes difficult to design a universal agent if it relies on this command.

3.2.10 Conclusion
This analyse leads us to prefer a simple method that is also compatible with satisfying security basics. As most of the above tools do not exist on Windows, it will be anyway necessary to implement manually one of them. The PHP web application and the raw socket methods seem to be the most interesting, because of their ease to be implemented.

3.3 Analysis of security

Security is definitely one of the main themes of this project. Reports have to be carefully sent to avoid any compromising or interception from a malicious person. Here are explained three of the main aspects of computer security that are related to the project.

3.3.1 Integrity

This term refers to the fact that when an agent sends its report to the server, no modifications of the report are done by a third part. Technically, it implies that whenever someone reads the message, he will not be able to silently modify its content. Integrity assumes that the data come from the one who emitted them, and that no one interfered during the transaction. It could also be used to deduce whether a packet has been corrupted during its transfer, but since TCP protocol is employed, it confers a sufficient proof of integrity.

This part is complied by the use of a private certificate to sign the information. Commonly, servers have their own private certificate distributed by the esCERT. The organism acts as a certificate supplier for every server running in the faculty. Once a server has obtained its certificate, it uses it to validate each form of data sent. Thus, it way of proceeding ensures one is really dealing with the real server, and nobody is stealing its identity. Those certificates are issued in PEM format. They contain a private RSA key that is used later to compute data. For example they are used to sign a checksum of data block. A checksum is first calculated for those data, and the certificate uses its key to compute the checksum with, resulting in a secured checksum. The resulting checksum is assumed to be unbreakable, which means that even if someone manages to steal this secured checksum, he will not be able to deduce the original checksum in a reasonable amount of time without knowing the key used for encryption. Once the report received, the server will compute the public key of the emitter with the secured checksum. This operation will result in calculating the original checksum. Later, the server will calculate the checksum for the report it has received and compare them. If they are equal, it means that the data have not been altered since they were emitted. In the other way, it means that those data have been prone to an unwanted change indicating corruption. In other words, it is impossible for an attacker to generate a valid checksum of a modified report if he does not know the original certificate.
3.3.2 Authenticity

It is also important to ensure that the current report really comes from a well known server. The certificate mentioned above justifies the identity of the sender because, in normal conditions, this certificate is highly private. Only the emitter knows about the secret key that has been used to compute the checksum. When a certificate is issued, it contains the identity of the target. Therefore, when ensuring the data have been correctly sent, the certificate is then checked to know whether the target has a valid identity. Besides the integrity, a certificate assures the identity of the sender. For example, if this step was not complied, one could forge a report pretending it comes from another server and send it to the processing server. This last one could then be lured, and would then enter false data into the database, compromising the state of the original server. Therefore, it could be assumes that the real machine is fully updated and does not require any attention although, obviously, some flawed software would be running. This case demonstrates the emphasis of the authentication process while registering the contents of a machine into the database.

3.3.3 Confidentiality

Despite it is the last one quoted, it might be the one that requires the most importance. When data are sent through the network, they are divided into several packets, each one crossing several machines before arriving to destination. Then, it is easy for a malicious person to intercept those data, and read their contents. For instance, a simple browse on Google send a lot of different packets in which it is possible, if someone intercepts them, to know what the user is currently doing. A Google search does not appear to be highly relevant, but a communication between two persons using an instant messaging system, might be much more interesting. In the case of this project, if someone manages to read the contents of the report, consequences could be disastrous. Actually, he would get a full fingerprint of the system, its version, its characteristics, and list of every piece of software associated to its version running on the system. This point is highly critical given that the attacker would know exactly what to do to attack the machine. That's why it is absolutely necessary to control and to secure the flow of data sent to the server.

This task is done by using the public certificate issued by the server. This certificate contains a public key that is used to compute data. Once computed, the original data become unreadable for anyone that does not possess the private key. Data can then be sent through the network without any risk of information leak. If someone intercepts the flow, he will only be able to see sequences of bits without any logic. Unless he has the key to decrypt those data it is absolutely impossible for him to discover the content of the original message. Encrypting with a certificate is one the most common method used to secure data. Most encryption algorithms use a shared secret, for example a password shared by the sender and the receiver, whereas a certificate works with two keys a public and a private. Anybody can then encrypt data using the public key provided by the recipient, and once encrypted, original data will only be read by the recipient. This method has the real advantage for the emitter not to share his secret. In the other way, the two persons indeed have to find a way to share their secret password, which can reveal to be a kind of leak. Whatever
the way the secret is shared, there is no way to be completely sure no one has been able to intercept it so that the process can then be considered unsafe. In the case of a certificate, the public key can be disclosed on a web site, that is to say, available to any one who desires it without caring of its security. In some cases, it is all the same possible to break such a key if for instance it has been generated with a design error in the generating process. Some facts like the flaw inserted by accident in the OpenSSL random number generator on Debian / Ubuntu may let an attacker break the key. Those design errors are however considered very rare, so in most cases using such a method for securing data is a synonym of high security.

3.3.4 What kind of method to use

At first sight, I want tempted to use Pretty Good Privacy (PGP) to secure the reports. This method is also highly efficient and widely used to secure emails. Someone who uses it may generate two different keys: a public one and a private one aimed at decrypting data encrypted with the public key. As a result and as for the certificate method, this method is also asymmetric. Public keys can generally be found in public repository. Someone who wants to send a PGP secured email has just to download the public key of his recipient and encrypt it. Once encrypted, the email can be sent leaving the insurance that data would be transported in a safer way. PGP is also used to ensure the origin of the message. As for a certificate, PGP provides a good authentication method. Using this method, it is possible to digitally sign an email, and send this signature with it. To confirm the origin of the mail, the receiver has to compute a hash (checksum) of the message, and next compute it with the public key of the sender. If the two hashes match, then the content of the email has not been altered and therefore the email is authentic. PGP is a robust system that has made its proofs and is widely spread in the world. I considered this possibility because the report that has to be sent only contains printable characters. Therefore it could be regarded as an email sent to the server. Using PGP would just be an adaptation of this method to our specific system. Nonetheless, certificates generated by the esCERT are not compatible with the PGP cryptographic system. So in order to make this method relevant, we would have to provide to any server that might receive an agent a PGP key. And as they already possessed a digital certificate, it would be useless to add an additional security system.

Implementing a shared secret method is perhaps the easiest way to provide security. But with nowadays' tools, one can in a few seconds completely reverse engineers an application and guess this secret. Asymmetric systems do not behave in the same way since the public key might be downloadable for everyone. Thus, there only exists one single private key able to decrypt the messages, which means that an attacker has to compromise the main server to be able to decrypt the reports. But in this case, he also has access to the database, so there is not point anymore in trying to uncover the reports. In our system OpenSSL will be in charge of encrypting the report.