

Bachelor's thesis (UAS)
Information Technology

2011

Pablo Molina Martínez

SECURITY ALERTS COLLECTING SYSTEM (SYRAS)

– Sistema Recolector de Alertas de Seguridad



TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology

Completion of the thesis: June 2011 | Total number of pages: 154

Instructor: Patric Granholm

Author: Pablo Molina Martínez

TURUN AMMATTIKORKEAKOULU THESIS

Nowadays, maintaining the integrity of the systems used for information technology is becoming increasingly important, but at the same time, the task becomes more and more complex due to the great number of factors involved.

One of the most important measures to maintain secured our systems is to apply available security updates and patches. Such updates, free and given by software vendors, serve to fix vulnerabilities in their programs. Keeping updated all the used software we avoid the risk that our systems are damaged by attackers attempting to exploit known vulnerabilities.

Every new vulnerability or security update published is usually enclosed by a small security alert or report (called security advisory), which typically includes: the affected products, a brief description of the problem and the download links to a security patch or newer version which solve the vulnerability. The security advisories can be received from three different ways depending on its origin: email alerts, websites that post security alerts, or RSS newsfeeds.

Depending on the source, it will be necessary to implement a type of module for the collection of such alerts. Therefore, the part responsible for harvesting the advisories will consist of three modules: E-mail, Web and RSS. Those three modules represent the core of the SyRAS system.

The main idea of the project is to build a centralized system that is able to collect multiple security advisories from different sources, parse them and save them in a database with a standard structure. Later on the new advisory, called entry in the database, can be linked to the product affected by the vulnerability. And in case that product is in an administrator's list of managed systems, the administrator will have all the information needed to compose and send a new security advisory in a few minutes.

KEYWORDS:

IT security, security advisory, security alert, collecting system, Python

FOREWORD

I would like to use this page thank a lot of people who have helped me along this way:
to my mum, dad and sister for supporting me all the time,
to the rest of my family and specially my grandparents for being an example for me,
to my teachers for doing a great job,
to all my friends because I won't be who I am without them,
and finally, to little A for loving me so much despite the distance.

CONTENTS

1 INTRODUCTION	1
1.1 Objectives	3
1.2 Required configuration	6
1.3 Thesis summary	6
2 COMPUTER SECURITY	9
2.1 Definition	9
2.1.1 Information security	9
2.2 Computer vulnerabilities	11
2.3 Managing IT risks	12
3 SECURITY ADVISORIES	15
3.1 Definition	15
3.2 Sources	18
3.2.1 By Origen	19
3.2.2 By Type	25
4 THE SYRAS PLATFORM	27
4.1 Definition	27
4.2 Python programming language	27
4.2.1 Python applications	28
4.3 Eclipse	29
4.3.1 Eclipse architecture	30
4.3.2 Pydev	31
4.3.3 Subclipse	32
4.4 Subversion	32
4.5 SQLite	33
4.5.1 Features of SQLite	33
4.5.2 SQLite Manager	34
4.6 Ubuntu	35
5 ANALYSIS	37
5.1 General overview	37
5.2 Objectives of the system	39
5.3 System requirements catalogue	39

5.3.1	Information requirements	39
5.3.2	User level requirements	40
5.3.3	Functional requirements	40
5.3.4	Non-functional requirements	41
5.4	Static and dynamic model of the system	41
5.4.1	Use case diagrams	42
5.4.2	Use case descriptions	42
5.4.3	Static and dynamic diagrams	43
6	DESIGN AND IMPLEMENTATION	45
6.1	General overview	45
6.2	The DB component and the SyRAS database	46
6.2.1	The SyRAS sqlite database	46
6.2.2	The DB class	49
6.3	The Users class	51
6.4	The Products class	53
6.5	The Sources class	56
6.5.1	The Email_Sources class	60
6.5.2	The Html_Sources class	62
6.5.3	The Rss_Sources class	64
6.6	The Entries class	66
6.7	The Console class	70
6.8	The syras.py and syrasloop.py modules	73
6.8.1	The syras.py	73
6.8.2	The syrasloop.py	74
7	INSTALLATION, TESTING AND USER MANUAL	75
7.1	Installation	75
7.2	SyRAS testing and user manual	77
7.2.1	Starting SyRAS	77
7.2.2	Using SyRAS	78
8	CONCLUSIONS AND FUTURE WORK	85
8.1	Conclusions	85
8.2	Future work	86

8.2.1	Addition of new sources	86
8.2.2	User alerts	87
8.2.3	New body scripts	87
8.2.4	Web interface	87
	REFERENCES	89
	APPENDIX 1: DIAGRAMS AND REQUIREMENTS	93
	APPENDIX 2: CODE	119
	APPENDIX 3: SECURITY ADVISORIES	151

FIGURES

Figure 1: Security vulnerabilities	12
Figure 2: FreeBSD Security Advisory	16
Figure 3: Red Hat Errata RSS feed	22
Figure 4: Secunia website (www.secunia.com)	25
Figure 5: SyRAS basic scheme	38
Figure 6: Simple static model for SyRAS	43
Figure 7: General class diagram	45
Figure 8: Users table in SyRAS database	47
Figure 9: Products table in SyRAS database	47
Figure 10: Entries table in SyRAS database	48
Figure 11: Sources table in SyRAS database	49
Figure 12: DB class diagram	50
Figure 13: Users class diagram	51
Figure 14: Products class diagram	54
Figure 15: Simple Sources class diagram showing inheritance	56
Figure 16: Sources class diagram	57
Figure 17: Email_Sources class diagram	60
Figure 18: Html_Sources class diagram	63
Figure 19: Rss_Source class diagram	65
Figure 20: Entries class diagram	67
Figure 21: Console class diagram	70
Figure 22: SyRAS files and source code	76
Figure 23: SyRAS filesystem, execution and help command	78
Figure 24: SyRAS login command	79
Figure 25: SyRAS browse command	79
Figure 26: Browse command result	80
Figure 27: SyRAS collect command	80
Figure 28: SyRAS delete command	81
Figure 29: SyRAS list command	81
Figure 30: SyRAS logout command	81
Figure 31: SyRAS priority command	82
Figure 32: SyRAS save command	82
Figure 33: SyRAS save command result	82
Figure 34: SyRAS show command	83
Figure 35: SyRAS exit command	83

LIST OF ABBREVIATIONS (OR) SYMBOLS

API	Application Programming Interface
DB	Database
CGI	Common Gateway Interface
CLI	Command Line Interface
CVE	Common Vulnerabilities and Exposures
GNU	General Public License
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
LGPL	Lesser General Public License
LOPD	Ley Orgánica de Protección de Datos de Carácter Personal
OS	Operative System
RSS	Really Simple Syndication
SDK	Software Development Kit
SQL	Structured Query Language
SVN	Subversion
UML	Unified Modeling Language
UNIX	Trademarked operative system
XML	Extensive Markup Language

1 INTRODUCTION

Nowadays, maintaining the integrity of the systems used for information technology is becoming increasingly important, but at the same time, the task becomes more and more complex due to the great number of factors involved.

The term computer system security means the collective processes and mechanisms by which sensitive and valuable information and services are protected from publication, tampering or collapse by unauthorized activities or untrustworthy individuals and unplanned events, respectively.

The objective of computer security [1] includes protection of information and property from theft, corruption, or natural disaster, while allowing the information and property to remain accessible and productive to its intended users. For a system to be defined as secure, it should have several basic characteristics, such as: confidentiality, integrity, availability, accountability and assurance.

In practice, new security flaws or vulnerabilities [2] are published every day in critical software used by thousands of people, hence the need to manage the security of our information systems in an efficient manner.

In corporate environments the need to keep computer systems secure is even greater. If an attacker gained access to systems getting sensitive information such as future projects, customer data, this could pose serious problems for the company: loss of large amounts of money, poor corporate image towards clients and possible legal consequences in case personal information is exposed. In

that case, it would inflict the Spanish Data Protection Act (*LOPD - Ley Orgánica de Protección de Datos de Carácter Personal*) [3] [4].

One of the most important measures to maintain our systems secured is to apply available security updates and patches. Such updates, free and given by software vendors, serve to fix vulnerabilities in their programs. By keeping updated all the used software, we avoid the risk that our systems are damaged by attackers attempting to exploit known vulnerabilities.

Every new vulnerability or security update published usually encloses a small security alert or report (called security advisory [5]), which typically includes: the affected products, a brief description of the problem and the download links to a security patch or newer version which solve the vulnerability.

The problem here is that keeping systems up to date is not an easy task. System administrators should be aware of new security updates for products used and apply the updates as soon as possible to minimize risk.

To facilitate this task, some manufacturers have decided to publish security bulletins regularly. An example can be Oracle, Cisco or Microsoft [6]. Unfortunately the vast majority of manufacturers does not use this policy.

There are programs with self-updaters, but to make this more complicated, there is another type of software that does not give us any clue about security problems or updates, and only discloses them via its security bulletins, sometimes published on websites and others via RSS or mailing lists.

Ultimately, there are specialized channels where researchers disclose vulnerabilities before manufacturer reports on them or make available a countermeasure. This case is especially sensitive, since these channels can get to publish proof-of-concept (or exploits) that could be used to attack a failure in our systems before there is a solution.

1.1 Objectives

Given the need to implement security updates as soon as possible, and how difficult it is to get all the information and security alerts, the main objective of this project is to build a centralized system to collect all the advisories. A **Security alerts-collecting system which will be referred to as “SyRAS” from now on.**

As was mentioned above, there is a variety of sources from which security advisories can be collected. Then a classification has been carried out in response to two different criteria, by origin and type.

Criteria 1: Origin: The sources of the security advisories can be received from three different ways depending on their origin: email alerts, websites that post security alerts, or RSS newsfeeds.

Email: It would require implementing an email account to receive all security alerts that come from various mailing lists.

Ex: Vupen mailing list [7], Gentoo mailing list [8]

Website: We need an automated system to visit the web pages, download, and parse the HTML code.

Ex: Secunia website [9], SecurityTracker website [10]

RSS: System for collecting news feeds (RSS). It can be done by visiting the website directly or in a separate module for RSS. The second option will be the one used in this work.

Ex: Debian [11], Ubuntu [12], Gentoo [13], Securityfocus [14], OSVDB [15].

Criteria 2: Type: This refers to sources that affect different products, or just one product.

A- General Sources: They are global sources that can contain security alerts for different kind of products.

Ex: Vupen mailing list, Secunia website, Securitytracker website, Securityfocus RSS, OSVDB RSS.

B- Specific Sources: They are the ones which are concerned only with a product or manufacturer.

Ex: Gentoo mailing list, Debian RSS, Ubuntu RSS, Gentoo RSS.

Depending on the source, it will be necessary to implement a type of module for the collection of such alerts. Therefore, the part responsible for harvesting the advisories will consist of three modules: E-mail, Web and RSS. Those three modules represent the core of the SyRAS system.

Once the security alerts are collected, they will be converted to a single prefixed format for greater convenience when it comes to using them, and they will be inserted into the databases.

From there, the system will attempt to associate the alert to a particular product. Different techniques can be applied, although it was initially carried out by search strings or making use of Regular Expressions (Regex).

Once various security alerts have been collected, it is possible to do a series of transformations on them to get the title, the body of the alert, permalink, etc. In addition, scripts can be applied to obtain the additional identifier CVE [16], or to modify the body of the alert.

As a final part, a command line interface has been developed, including a login system for users. The interface can collect, show, save and modify all the new alerts and also show the different tables for users, products, and sources.

The main idea of the project is to build a centralized system that is able to collect multiple security advisories from different sources, parse them, and save them in a database with a standard structure. Later on the new advisory, called entry in the database, can be linked to the product affected by the vulnerability. And in case that product is in the list of managed systems, it will make available all the information needed to compose and send a new security advisory in a few minutes.

1.2 Required configuration

The following systems set up have been used in this project:

- Ubuntu 10.10 *Maverick Merkaat* as Operative System.
- Eclipse 3.5.2 *Galileo* as Software Development Kit.
- SQLite 3.7.3 as Database.
- Python 2.6.5 as Programming Language.
- Mozilla Firefox 3.6.12 + SQLite Manager 0.6.5 as Database Manager.
- Several standard and non-standard Python libraries have been used. They will be referred to in the following chapters.

1.3 Thesis summary

The thesis is organized in eight chapters. After this introduction chapter, the next chapters have been structured in the following way:

❖ **Chapter 2: Computer Security**

This chapter introduces computer and information security. It defines several concepts that will be used in the following chapters, such as IT security, computer vulnerabilities, etc. It reviews the current situation and gives some tips about security management systems.

❖ **Chapter 3: Security Advisories**

Every new vulnerability or security update published is usually enclosed by a small security alert or report (called **security advisory** or **security bulletin**), which typically includes: affected products, a brief description of the problem, and the download links to a security patch, or newer version which solve the vulnerability. This chapter defines what a security advisory is and the different sources to look for them.

❖ **Chapter 4: The SyRAS platform**

This chapter provides a small introduction to the different software and platforms used in the SyRAS project. It will give an overview of the Python programming language, the Eclipse Software Development Kit, the SQLite database and the Ubuntu Operative System. Programming language, SDK, DB and OS are the tools used for building the SyRAS software.

❖ **Chapter 5: Analysis**

In this chapter we will see the analysis phase of the application developed for the SyRAS project. We will study the requirements that the application must meet to complete the objectives of the project. Based on these requirements, we will extract the use cases from the application and obtain a static view of system and dynamic view using sequence diagrams.

❖ **Chapter 6: Design and implementation**

In this chapter we will revise the static models defined in the analysis section, but with more detail and including the entity-relationship model. This chapter will also explain the implementation of the most important parts of the system.

❖ **Chapter 7: Installation, testing and user manual**

This chapter shows the steps to be followed for installation, configuration and testing of the SyRAS system.

❖ **Chapter 8: Conclusions and future work**

This is the last chapter of the thesis memory; it embodies reflections on the project addressed, and possible improvements and future work that could be carried out related to the SyRAS project.

2 COMPUTER SECURITY

2.1 Definition

This chapter introduces computer and information security. It defines several concepts that will be used in the following chapters, such as IT security, computer vulnerabilities, etc. It reviews the current situation and gives some tips about security management systems.

Computer security is a branch of computer technology known as information security as applied to computers and networks. The objective of computer security includes protection of information and property from theft, corruption, or natural disaster, while allowing the information and property to remain accessible and productive to its intended users. The strategies and methodologies of computer security often differ from most other computer technologies because of its somewhat elusive objective of preventing unwanted computer behavior instead of enabling wanted computer behavior.

2.1.1 Information security

Information security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording, or destruction. [17]

The key concepts for **Information security** are:

- **Confidentiality** is the term used to prevent the disclosure of information to unauthorized individuals or systems.
- **Integrity** means that data cannot be modified undetectably.
- **Availability:** The information must be available always when it is needed.
- **Authenticity** ensures that the data, transactions, communications or documents are genuine, and also that both parties involved are who they claim they are.
- **Non-repudiation** implies that one party of a transaction cannot deny having received a transaction nor can the other party deny having sent a transaction.

The terms **information security**, **computer security** and information assurance are frequently used interchangeably. These fields are interrelated often and share the goals of protecting the confidentiality, integrity and availability of information; however, there are some differences between them. These differences lie primarily in the approach to the subject, the methodologies used, and the areas of concentration. Information security is concerned with the confidentiality, integrity, and availability of data regardless of the form the data may take: electronic, print, or other forms. **Computer security** can focus on ensuring the availability and correct operation of a computer system without concern for the information stored or processed by the computer.

Governments, military, corporations, financial institutions, hospitals, and private businesses handle a great amount of confidential information about their employees, customers, products, research, and financial status. If it falls into the hands of a competitor, such a breach of security could lead to lost business, law suits or even bankruptcy of the business.

Protecting confidential information is a business requirement, and in many cases also an ethical and legal requirement, as seen in the Spanish LOPD. [3] [4]

The field of information security has evolved significantly in recent years. It has many areas for specialization including: securing networks and allied infrastructure, securing applications and databases, security testing, information systems auditing, business continuity planning and digital forensics science, etc.

2.2 Computer vulnerabilities

In computer security, vulnerability is a weakness which allows an attacker to reduce a system's information assurance. Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw. To be vulnerable, an attacker must have at least one applicable tool or technique that can connect to a system weakness. [2]

According to Microsoft: “A **security vulnerability** is a flaw in a product that makes it infeasible – even when using the product properly - to prevent an attacker from usurping privileges on the user's system, regulating its operation, compromising data on it, or assuming ungranted trust.(...) These are the types of issues we generally address via **security bulletins**.” [18]

A resource (physical or logical) can have one or more vulnerabilities that can be exploited by an agent in a threat action, as shown in Figure 1. The result can compromise the confidentiality, integrity or availability properties of resources

(potentially different than the vulnerable one) of the organization and other involved parties (customers, suppliers).

The attack can be active when it attempts to alter system resources or affect their operation: so it compromises integrity or availability. A "passive attack" attempts to learn or make use of information from the system but does not affect system resources, so it compromises confidentiality.

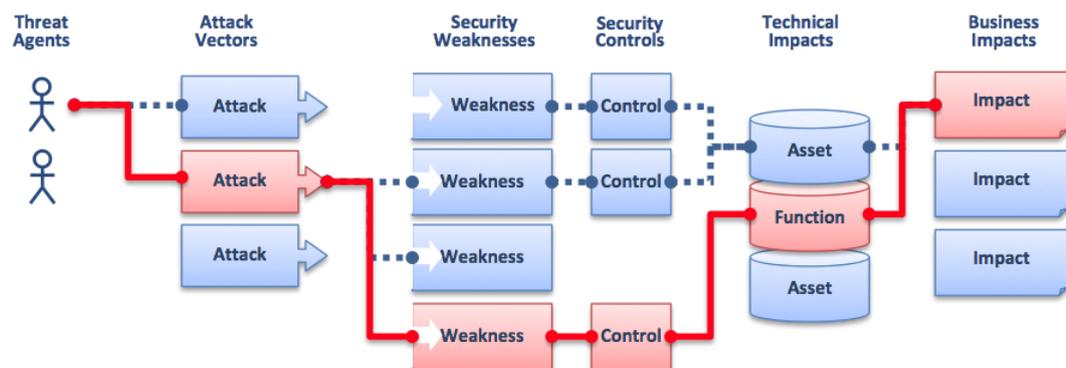


Figure 1. Security vulnerabilities

2.3 Managing IT risks

The impact of a security breach can be very high. The fact that IT managers, or upper management, can know that IT systems and applications have vulnerabilities and do not perform any action to manage the IT risk is seen as misconduct in most legislations. Privacy law forces managers to act to reduce the impact or likelihood of security risk.

The only way to reduce the chance of a vulnerability being used against a system is through constant vigilance, including careful system maintenance (e.g., applying software patches), best practices in deployment and auditing. As mentioned before, one of the most important measures that IT managers have to do is to apply available security updates and patches. Such updates, free and given by software vendors, serve to fix vulnerabilities in their programs.

3 SECURITY ADVISORIES

3.1 Definition

Every new vulnerability or security update published is usually enclosed by a small security alert or report (called **security advisory** or **security bulletin**), which typically includes: affected products, a brief description of the problem, and the download links to a security patch or newer version which solve the vulnerability. This chapter defines what a security advisory is and the different sources to look for them.

For a better understanding the author will use the description, shown in Figure 2, of a security advisory made by FreeBSD (an advanced operating system for modern server, desktop, and embedded computer platform based on BSD UNIX). [5]

```

=====
FreeBSD-SA-XX:XX.UTIL                                     Security Advisory
                                                         The FreeBSD Project

Topic:           denial of service due to some problem1

Category:        core2
Module:          sys3
Announced:      2003-09-234
Credits:         Person@EMAIL-ADDRESS5
Affects:         All releases of FreeBSD6
Corrected:       FreeBSD 4-STABLE prior to the correction date
                 2003-09-23 16:42:59 UTC (RELENG_4, 4.9-PRERELEASE)
                 2003-09-23 20:08:42 UTC (RELENG_5_1, 5.1-RELEASE-p6)
                 2003-09-23 20:07:06 UTC (RELENG_5_0, 5.0-RELEASE-p15)
                 2003-09-23 16:44:58 UTC (RELENG_4_8, 4.8-RELEASE-p8)
                 2003-09-23 16:47:34 UTC (RELENG_4_7, 4.7-RELEASE-p18)
                 2003-09-23 16:49:46 UTC (RELENG_4_6, 4.6-RELEASE-p21)
                 2003-09-23 16:51:24 UTC (RELENG_4_5, 4.5-RELEASE-p33)
                 2003-09-23 16:52:45 UTC (RELENG_4_4, 4.4-RELEASE-p43)
                 2003-09-23 16:54:39 UTC (RELENG_4_3, 4.3-RELEASE-p39)7
CVE Name: CVE-XXXX-XXXX8

For general information regarding FreeBSD Security Advisories,
including descriptions of the fields above, security branches, and the
following sections, please visit
http://www.FreeBSD.org/security/.

I.   Background9

II.  Problem Description10

III. Impact11

IV.  Workaround12

V.   Solution13

VI.  Correction details14

VII. References15

```

Figure 2. FreeBSD Security Advisory

1. The `Topic` field indicates exactly what the problem is. It is basically an introduction to the current security advisory and notes the utility with the vulnerability.
2. The `Category` refers to the affected part of the system which may be one of `core`, `contrib`, or `ports`. The `core` category means that the vulnerability affects a core component of the FreeBSD operating system. The `contrib` category means that the vulnerability affects software contributed to the FreeBSD Project, such as **sendmail**. Finally

the `ports` category indicates that the vulnerability affects add-on software available as part of the Ports Collection.

3. The `Module` field refers to the component location, for instance `sys`. In this example, we see that the module, `sys`, is affected; therefore, this vulnerability affects a component used within the kernel.
4. The `Announced` field reflects the date the security advisory was published, or announced to the world. This means that the security team has verified that the problem does exist and that a patch has been committed to the FreeBSD source code repository.
5. The `Credits` field gives credit to the individual or organization who noticed the vulnerability and reported it.
6. The `Affects` field explains which releases of FreeBSD are affected by this vulnerability. For the kernel, a quick look over the output from `ident` on the affected files will help in determining the revision. For ports, the version number is listed after the port name in `/var/db/pkg`. If the system does not sync with the FreeBSD CVS repository and rebuild daily, chances are that it is affected.
7. The `Corrected` field indicates the date, time, time offset, and release that was corrected.
8. Reserved for the identification information is used to look up vulnerabilities in the Common Vulnerabilities Database system.
9. The `Background` field gives information on exactly what the affected utility is. Most of the time this is why the utility exists in FreeBSD, what it is used for, and a bit of information on how the utility came to be.
10. The `Problem Description` field explains the security hole in depth. This can include information on flawed code, or even how the utility could be maliciously used to open a security hole.

11. The `Impact` field describes what type of impact the problem could have on a system. For example, this could be anything from a denial of service attack, to extra privileges available to users, or even giving the attacker superuser access.
12. The `Workaround` field offers a feasible workaround to system administrators who may be incapable of upgrading the system. This may be due to time constraints, network availability, or a slew of other reasons. Regardless, security should not be taken lightly, and an affected system should either be patched or the security hole workaround should be implemented.
13. The `Solution` field offers instructions on patching the affected system. This is a step-by-step tested and verified method for getting a system patched and working securely.
14. The `Correction Details` field displays the CVS branch or release name with the periods changed to underscore characters. It also shows the revision number of the affected files within each branch.
15. The `References` field usually offers sources of other information. This can include web URLs, books, mailing lists, and newsgroups.

3.2 Sources

As mentioned before, there are a variety of sources from which security advisories can be collected. Having in mind the need to keep the systems updated at anytime, we should collect the information from as many of them as possible. Then, depending on the reliability of every one of them, we can contrast the information coming from different sources to have a good overall picture of the vulnerabilities, problems and risks that can affect our managed systems.

Then a classification has been carried out in response to two different criteria, by origin and type.

3.2.1 By Origen

The security advisories can be received from three different ways depending on the origin of its sources: email alerts, websites that post security alerts, or RSS newsfeed.

The SyRAS software implements three different modules to perform the collection of security advisories coming for the different sources. The modules will be explained in the following sections.

1. E-mail

It is very common to subscribe to security mailing lists managed by software vendors, which allow receiving new security alerts in an email inbox. Of course, this option would require having an email account to receive all security alerts that come from various mailing lists. The account syrasfeed@gmail.com is the one used for this purpose.

The following is a clear example of this policy and it is taken as-is from the “Notifications and Advisories” website by Red Hat Linux [19]:

Get notified about new security advisories

A number of public mailing lists send notifications about new security advisories for Red Hat products:

- Subscribe to [rhsa-announce](#) if you want advisories for every Red Hat product and service.
- Subscribe to [enterprise-watch-list](#) if you want advisories only for Red Hat Enterprise products.
- Subscribe to [jboss-watch-list](#) if you want advisories only for JBoss Enterprise Middleware products.
- Subscribe to [rhev-watch-list](#) if you want advisories only for Red Hat Enterprise Virtualization products.

There are lots of mailing lists similar to the given example that provide information about a product or vendor, but there are other vendor independent lists worth mentioning. As an example the author would like to point out the most popular of them called **Full-Disclosure**, an unmoderated high-traffic forum for disclosure of security information that has been working since 2002. [20]

Here there are some of the most important vendor dependent mailing lists, including its permalinks:

- [Apache](#) - Apache Week, includes Apache security alerts.
- [Debian: Security Announcements](#) - Security announce mailing list for Debian Linux.
- [Gentoo Linux](#) - List of security announcements and information on how to sign up to security mailing list.
- [Microsoft](#) - How to subscribe to the (free) e-mail notification service that Microsoft uses to send information about security of Microsoft products.
- [NetBSD](#) - Security mailing list subscription.
- [OpenBSD](#) - All OpenBSD mailing lists, including security alerts.
- [Oracle](#) - Oracle security mailing list requiring registration.
- [Red Hat Linux](#) - Notifications and advisories. Security mailing list subscription is on this page.
- [SuSE Linux](#) - All SuSE Linux mailing lists, including security alerts.

- [Sun Security Information](#) - It includes subscription instructions for the Sun security-alert mailing list. Subscribe to this list to be alerted when security patches are released. Very low volume.

The mailing lists used in the SyRAS working demo are the following:
Vupen mailing list [7], Gentoo mailing list [8].

2. RSS

RSS (most commonly expanded as Really Simple Syndication) is a family of web feed formats used to publish frequently updated works in a standardized format. This format is widely used and helps to be updated with the latest news published in several IT security websites.

The following lines could be a good example and are also taken as-is from the “Notifications and Advisories” website by Red Hat Linux: [19]:

An RSS feed for Red Hat advisories is also available. To take advantage of this service, point your favorite RSS client to the [Red Hat advisory RSS feed](#).

And the result of the RSS feed is shown in Figure 3:

Red Hat Errata

The latest Red Hat errata for Red Hat Linux

[RHBA-2010:1006-1: v7 bug fix and enhancement update - RHEL6](#)

Red Hat Enterprise Linux: An updated v7 package that fixes several bugs and adds enhancements is now available. [Updated 21st September 2010] This advisory has been updated with the correct URL for accessing the v7 Technical Notes, and a corrected Solution text. This update does not change the packages in any way. [Updated 20 October 2010] This advisory has once more been updated with a corrected URL for accessing the v7 Hardware Certification Test Suite. Neither this update nor the previous one have changed the packages in any way.

[RHBA-2010:1004-1: initscripts bug fix update](#)

Red Hat Enterprise Linux: An updated initscripts package that fixes a bug is now available for Red Hat Enterprise Linux 6.

[RHBA-2010:1005-1: mkinitrd bug fix update](#)

Red Hat Enterprise Linux: An updated mkinitrd package that fixes a bug is now available for Red Hat Enterprise Linux 4.

Figure 3. Red Hat Errata RSS feed

This option would require implementing an RSS client to receive all security alerts that come from various RSS feeds.

The RSS feeds used in the SyRAS working demo are the following:

Ex: Debian [11], Ubuntu [12], Gentoo [13], Securityfocus [14] and OSVDB [15].

3. Website

As expected, websites are the most common source for finding security advisories and IT security information. There are many websites related to the topic, coming from vendors, researchers and IT security companies.

Here there are some of the most important among them:

- [CERIAS](#) - Center for Education and Research in Information Assurance and Security. University center for multidisciplinary research and education in areas of information security.
- [US-CERT](#) - Established in 2003 to protect the nation's Internet infrastructure, US-CERT coordinates defense against and responses to cyber attacks across the nation.
- [Apache HTTP Server Vulnerability Lists](#) - Lists of security problems fixed in released versions of the Apache HTTP Server.
- [AusCERT](#) - Australian Computer Emergency Response Team. Advisories and tools.
- [Bugtraq](#) - Independent source for security vulnerabilities, alerts, and threats.
- [CERT Coordination Center](#) - studies Internet security vulnerabilities, provides incident response services to sites that have been the victims of attack.
- [ISS X-Force](#) - Security alerts, advisories, and alert summaries from ISS.
- [Linux Security Group](#) - Security Advisories, Anti Hackers, programming books and related links.
- [New Zealand Computer Emergency Readiness Team](#) - Charitable trust established to improve the general information security posture of New Zealand society. Provides news, alerts, trends and statistics.
- [Open Source Vulnerability Database](#) - Searchable database of vulnerabilities. It offers data for download in XML format as well as via website.

- [Oracle Security Center](#) - Tips, tools, and technologies to keep Oracle products safe, secure, and patched.
- [Patch Management Forum](#) - Mailing list facilitates networking and information exchange related to patch management: announcements, testing, verification, operations processes, and vulnerabilities.
- [PatchAdvisor](#) - Fee based patch alert service.
- [PatchManagement.org](#) - Mailing list dedicated to the discussion of patch management.
- [SANS Internet Storm Center](#) - Cooperative cyber threat monitor and alert system.
- [Secunia](#) – It provides security advisories information about patches, software and vulnerability management.
- [Symantec DeepSight Threat Management System](#) – Fee-based security alert service that provides early warning of active attacks.
- [VUPEN Security](#) – It provides security advisories and real-time information about vulnerabilities, exploits, and threats. It also provides vulnerability management and pentesting solutions.

Figure 4 shows an example of one of the most famous and reliable sources, the website of the Danish IT security company Secunia.

Home • Community • Advisories

Community

Advisories

- [Database](#)
- [Search](#)
- [Advisories by Product](#)
- [Advisories by Vendor](#)
- [Terminology](#)
- [Report vulnerability](#)
- [Insecure Library Loading](#)

Research ▾

Forum ▾

My Profile ▾

Our Commitment ▾

[Insecure Library Loading](#)

Community Login

Username:

Password:

[Register now](#)
[Forgot password?](#)

Highlights

Microsoft IIS FTP Server Telnet IAC Character Encoding Vulnerability ████████

Issued on 22 December, 2010 // Highly critical // From remote // 3,105 views

Matthew Bergin has discovered a vulnerability in Microsoft Internet Information Services (IIS), which can be exploited by malicious people to cause a DoS (Denial of Service) and potentially compromise a vulnerable system.

Microsoft Windows Fax Cover Page Editor Buffer Overflow Vulnerability ████████

Issued 5 days ago // Highly critical // From remote // 1,789 views

rgod has discovered a vulnerability in Microsoft Windows, which can be exploited by malicious people to compromise a vulnerable system.

Internet Explorer Multiple Vulnerabilities ████████

Issued on 3 November, 2010 // Extremely critical // From remote // 30,018 views

Multiple vulnerabilities have been reported in Internet Explorer, which can be exploited by malicious people to disclose potentially sensitive information or to compromise a user's system.

Follow us

Vuln Researcher

Latest advisories

New advisories: 1
New vulnerabilities: 2
Updated advisories: 5

- ████████ 322 views
[Linux Kernel "load_mixer_volumes\(\)" Vulnerabilities](#)
- ████████ 292 views
[CA ARCserve D2D Axis2 Default Account Security Issue](#)
- ████████ 248 views
[TorrentTrader Cross-Site Scripting and SQL Injection Vulnerabilities](#)
- ████████ 208 views
[Streber Multiple Cross-Site Scripting Vulnerabilities](#)
- ████████ 217 views
[PHP-addressbook "group_name" SQL Injection Vulnerability](#)

Most popular - 3 hours

1. [Adobe Flash Player Multiple Vulnerabilities](#) // 39 views
2. [Linux Kernel "load_mixer_volumes\(\)" Vulnerabilities](#) // 22 views
3. [Internet Explorer CSS Import Rule Processing Use-After-Free Vulnerability](#) // 19 views
4. [Internet Explorer Multiple Vulnerabilities](#) // 18 views
5. [Adobe Reader / Acrobat SING "uniqueName" Buffer Overflow Vulnerability](#) // 16 views
6. [Microsoft Windows Fax Cover Page Editor Buffer Overflow Vulnerability](#) // 16 views
7. [Adobe Flash Player / AIR Code Execution Vulnerability](#) // 13 views

Figure 4. Secunia website (www.secunia.com)

Another module of SyRAS is an automated system that visits web pages, downloads their content and parses the HTML code, extracting the important parts to compose the security advisories.

The websites used in the SyRAS working demo are the following:
Secunia website [9], SecurityTracker website [10].

3.2.2 By Type

The SyRAS software is able to collect security advisories from two different sources according to its type. There are **general sources** that can publish information for any kind of software, and **specific sources** that publish advisories affecting just one product or vendor.

The sources of the security advisories can be E-mail, RSS or Website depending on its origin, and general or specific depending on its type.

1. General sources

Global sources may contain security alerts for different kind of products. They publish advisories from multiple products and are vendor-independent. Usually, these sites belong to some public organization, research center or IT security company.

All the examples, except for the Oracle source, shown in the previous section (3.2.1 - Website) are general sources.

The general sources used in the SyRAS working demo are the following: Vupen mailing list [7], Secunia website [9], Securitytracker website [10], Securityfocus RSS [14] and OSVDB RSS [15].

2. Specific sources

They are the ones which are concerned only with a product or manufacturer. Usually, belong to a company or vendor and are essential to provide updated information about the latest bugs and patches that affect a concrete system or software.

All the examples, except Vupen, shown in the previous section (3.2.1 – E-mail) are specific sources.

The specific sources used in the SyRAS working demo are the following:
Ex: Gentoo mailing list [8], Debian RSS [11], Ubuntu RSS [12], Gentoo RSS [13].

4 THE SYRAS PLATFORM

4.1 Definition

This chapter provides a small introduction to the different software and platforms used in the SyRAS project. It will give an overview of the **Python** programming language, the **Eclipse** Software Development Kit, the **SQLite** database and the **Ubuntu** Operative System. Programming language, SDK, DB and OS are the tools used for building the SyRAS software.

4.2 Python programming language

Python is an interpreted, general-purpose high-level programming language whose design philosophy emphasizes code readability. Python aims to combine "remarkable power with very clear syntax", and its standard library is large and comprehensive. Its use of indentation for block delimiters is unusual among popular programming languages. [21]

Python supports multiple programming paradigms, primarily but is not limited to object-oriented, imperative and, to a lesser extent, functional programming styles. It features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl, and Tcl. Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts.

The reference implementation of Python (CPython) is free and open source software and has a community-based development model, as do all or nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

Python interpreters are available for many operating systems, and Python programs can be compiled into stand-alone executable code for those systems, using tools included with the interpreter installation package. The Python interpreter used for the SyRAS software is: *Python 2.6.6 (r266:84292, Sep 15 2010, 15:52:39). [GCC 4.4.5] on linux2*

4.2.1 Python applications

Python offers a wide range of choices for developers, including the following: [22]

❖ For web and internet:

- Writing basic CGI scripts.
- Frameworks such as Django and TurboGears.
- High-end solutions such as Zope.
- Advanced content management systems such as Plone.
- Extensive support for HTML and XML.
- E-mail processing.
- Processing RSS feeds.
- Support for many other Internet protocols.

❖ For database access:

- Custom and ODBC interfaces to MySQL, Oracle, MS SQL Server, PostgreSQL, SybODBC, and others are available for free download.
- Standard Database API.

- Object databases such as ZODB and Durus.

- ❖ For desktop GUIs:
 - The Tk GUI development library is included with most binary distributions for Python.
 - wxWidgets
 - GTK+
 - Qt via pyqt or pyside
 - Microsoft Foundation Classes through the win32 extensions
 - Delphi

And many others applications used for networking, game developing, education, scientific and numeric computing, 3D graphics and so on.

4.3 Eclipse

Eclipse is a multi-language software development environment comprising of an Integrated Development Environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, **Python**, Ruby (including Ruby on Rails framework), Scala, and Scheme. [23]

Users can extend its abilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Eclipse is released under the terms of the Eclipse Public License and it is free and open source software. The Eclipse SDK used for programming the SyRAS software is: *Eclipse SDK Version: 3.5.2 (codename Galileo). Build id: M20100211-1343*

4.3.1 Eclipse architecture

Eclipse employs plug-ins in order to provide all of its functionality on top of (and including) the runtime system, in contrast to some other applications where functionality is typically hard coded. The runtime system of Eclipse is based on Equinox, an OSGi standard compliant implementation.

This plug-in mechanism is a lightweight software componentry framework. In addition to allowing Eclipse to be extended using other programming languages such as **Python**, the plug-in framework allows Eclipse to work with typesetting languages like LaTeX, networking applications such as telnet and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. The **Subversion** support is provided by third-party plug-ins. With the exception of a small run-time kernel, everything in Eclipse is a plug-in. [24]

The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards.

4.3.2 Pydev

Pydev (Python Development Environment) is a third-party plug-in for Eclipse. It is an Integrated Development Environment (IDE) used for programming in Python. It uses advanced type inference techniques to provide features such as code completion and code analysis, while still providing many others such as a debugger, interactive console, refactoring, etc: [25] [26]

Below are some of its highlighted features CPython, Jython and IronPython support:

- Django integration
- Code completion
- Code completion with auto import
- Syntax highlighting
- Code analysis
- Go to definition
- Refactoring
- Mark occurrences
- Debugger
- Remote debugger
- Tokens browser
- Interactive console
- Unittest integration

The Pydev IDE used for programming the SyRAS software is: *Pydev - Python Development Environment 1.6.3.2010100513*.

4.3.3 Subclipse

Subclipse is an Eclipse Team Provider plug-in providing support for Subversion within the Eclipse IDE. Developed and maintained by Subversion core committers, Subclipse is always in synchronisation with the latest Subversion features and releases. [27]

Subclipse includes a powerful revision graph feature that is built with Eclipse GEF/Draw2D. This allows to visualize commits and merges across Subversion branches. The Subclipse plug-in used for synchronizing the different versions of the SyRAS software is: *Subclipse – SVN Client Adapter 1.6.12*

4.4 Subversion

Apache Subversion (often abbreviated SVN) is a software versioning and a revision control system founded in 2000. Developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation. [24]

It is a mostly-compatible successor to the widely used Concurrent Versions System (CVS) and uses the Apache License, making it free software and open source.

The Subclipse plug-in is the one used as SVN client for synchronizing the different versions of the SyRAS: *Subclipse – SVN Client Adapter 1.6.12*

4.5 SQLite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain. [28]

Unlike the client–server database management systems, the SQLite engine has no standalone processes with which the application program communicates. Instead, the SQLite library is linked in and thus becomes an integral part of the application program. The library can also be called dynamically. The application program uses SQLite functionality through simple function calls, which reduce latency in database access - function calls within a single process are more efficient than inter-process communication. SQLite stores the entire database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine. It implements this simple design by locking the entire database file during writing.

The SQLite database is used for storing all the information collected by the SyRAS software is: *SQLite 3.7.1*.

4.5.1 Features of SQLite

The most important features of SQLite are the following: [29]

- Atomic, consistent, isolated, and durable (ACID) transactions even after system crashes and power failures.
- Zero-configuration - no setup or administration needed.
- It implements most of SQL92.

- A complete database is stored in a single cross-platform disk file.
- It supports terabyte-sized databases and gigabyte-sized strings and blobs.
- It has small code footprint: less than 325KiB fully configured
- It is faster than popular client/server database engines for most common operations.
- It has simple, easy to use API.
- Written in ANSI-C. TCL bindings included. Bindings for dozens of other languages available separately.
- Well-commented source code with 100% branch test coverage.
- Available as a single ANSI-C source-code file that you can easily drop into another project.
- Self-contained: no external dependencies.
- Cross-platform: Unix (Linux and Mac OS X), OS/2, and Windows (Win32 and WinCE) are supported out of the box.
- Sources are in the public domain. Use for any purpose.
- Comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases.

4.5.2 SQLite Manager

SQLite Manager is a SQLite database manager provided as a Firefox extension. By providing the software as a Firefox extension, SQLite Manager is available on many different platforms and trivially easy to install. It is also available for a few other environments. [30]

It allows the user to:

- View the contents of one or more SQLite databases.
- Manage tables, indexes, views and triggers.
- View and manage the records.

- Build an SQL query and execute it.
- Export data to CSV or XML.

The SQLite Manager used for handling the SyRAS database information is: *SQLite 3.7.1*.

4.6 Ubuntu

Ubuntu is a computer operating system based on the Debian GNU/Linux distribution and distributed as free and open source software. Ubuntu is a fork of the Debian project's codebase and focuses on usability, security and stability. [31]

Ubuntu is composed of many software packages, the vast majority of which are distributed under a free software license, making an exception only for some proprietary hardware drivers. The main license used is the GNU General Public License (GNU GPL) which, along with the GNU Lesser General Public License (GNU LGPL), explicitly declares that users are free to run, copy, distribute, study, change, develop, and improve the software. On the other hand, there is also proprietary software available that can run on Ubuntu. [32] [33]

The Ubuntu OS used for hosting the SyRAS system is: *Ubuntu 10.10 - Maverick Meerkat*.

5 ANALYSIS

5.1 General overview

In this chapter we will analyse the application developed for the SyRAS project. We will study the requirements that the application must meet to complete the objectives of the project. We will extract the use cases from the application and obtain a static view of system and dynamic view using sequence diagrams.

Given the need to implement security updates as soon as possible, and how difficult it is to get all the information and security alerts, the goal of this project is to build a centralized system to collect all the advisories. A Security alerts-collecting system (SyRAS).

As was mentioned before, there is a variety of sources from which security advisories can be collected. Therefore, the part responsible for harvesting the advisories will consist of three modules: E-mail, Web and RSS, as shown in Figure 5.

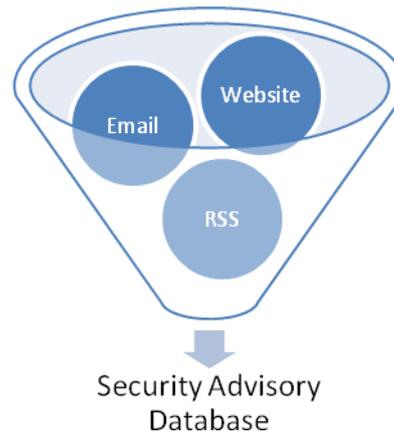


Figure 5. SyRAS basic scheme

Once the security alerts are collected, they will be converted to a single prefixed format for greater convenience when it comes to using them, and they will be inserted into a database, where all the information will be stored.

From there, the system will attempt to associate the alert to a particular product. Different techniques can be applied, although it is initially carried out by string search or using Regular Expressions (Regex).

Once we have collected the various security alerts, it is possible to do a series of transformations on them to get the title, the body of the alert, permalink, etc. In addition, scripts can be applied to obtain the additional identifier CVE or to modify the body of the alert.

As a final part, a command line interface has been developed, including a login system for users. The interface can collect, show, save, and modify all the new alerts and also show the different tables for users, products and sources.

5.2 Objectives of the system

The objectives to reach by the SyRAS system, widely explained in the Appendix 1, are the following:

- 1- Collect security advisories and
- 2- Manage and show the collected information,

5.3 System requirements catalogue

Here are the requirements the system must meet to achieve the objectives. The requirements will be of three different types: information and restrictions, functional and non-functional. All the requirements can be checked in Appendix 1.

5.3.1 Information requirements

The information requirements the SyRAS system, widely explained in the Appendix 1, are the following:

- 1- Information about SyRAS users
- 2- Information about SyRAS sources
- 3- Information about SyRAS products
- 4- Information about SyRAS entries (advisories)

5.3.2 User level requirements

This section will describe the three different user levels that can be used to manage the SyRAS software.

- ❖ **Guest:** It is the default user and does not need to log in.
 - It only has access to the following options:
 - Login
 - Help (general and specific)
 - Exit

- ❖ **User:** It is the regular user to interact with the SyRAS software. It needs to login using username and password.
 - It has access to all the options but the delete option.

- ❖ **Administrator:** It is the supervisor user to interact with the SyRAS software. It needs to login using username and password.
 - It has access to all the options including the delete option to erase an entry, product, source or user from the database.

5.3.3 Functional requirements

This section will list the SyRAS system functional requirements. All the details can be seen in Appendix 1.

1- Login a user

- 2- Collect new entries
- 3- List data
- 4- Query a specific database entry
- 5- Browse a specific source or advisory entry
- 6- Change priority
- 7- Delete a specific database entry
- 8- Command history
- 9- Command auto complete
- 10-Collecting loop

5.3.4 Non-functional requirements

This section will list the SyRAS system non-functional requirements. All the details can be seen in Appendix 1.

- 1- PC requirements
- 2- Application name
- 3- Portability
- 4- Future improvements

5.4 Static and dynamic model of the system

The next section shows the analysis of the requirements obtained and the diagrams produced showing the static and dynamic behavior of the system.

The SyRAS application allows the user to interact with the system, so an analysis will be made through use cases and sequence diagrams. The static model of the system will also be displayed. The full information can be seen in Appendix 1.

We can start by defining the system's stakeholders. As explained before, there are three different users that can interact with the SyRAS system:

- 1- SyRAS guest
- 2- SyRAS user
- 3- SyRAS administrator

5.4.1 Use case diagrams

The use case diagrams for the different actors: guest, user and administrator, are also included in Appendix 1.

5.4.2 Use case descriptions

The next step is make a description of the use cases previously specified, all of them are specified in Appendix 1.

5.4.3 Static and dynamic diagrams

After defining the use cases and taking into account the requirements, the next step is to define a very simple static model that will help us to show the interaction between the different components. The simple static model of the system is as shown in Figure 6.

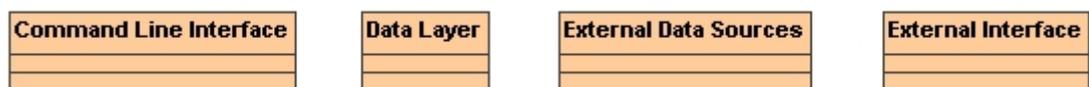


Figure 6: Simple static model for SyRAS

The Command Line Interface is used by the user (or actor) to interact with the system. The Data Layer is used to store and collect data from the database. The External Data Sources are used to collect new information from the sources that will be stored as new entries in the database. Finally, the External Interface will be used to show some information from the Data Layer in an interface different from the Command Line.

Having defined the static model of the system, the next step is to define the system dynamic model using sequence diagrams. There is a diagram for each use case where the exchange of messages between the various system objects can be appreciated. All the diagrams are in Appendix 1.

6 DESIGN AND IMPLEMENTATION

6.1 General overview

The following class diagram gives a general overview of all the components included in the SyRAS system, specifying the way they are related with each other to interact inside the system.

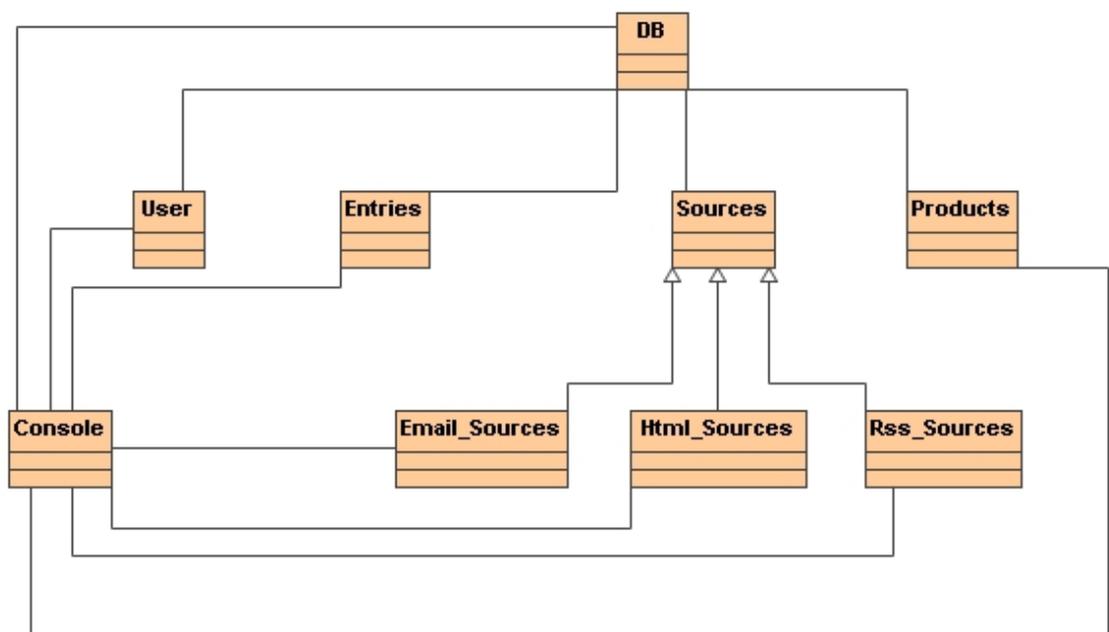


Figure 7: General class diagram

As it can be seen in Figure 7, there are two main components interacting with the others:

- The first one is the **Console** component (named Command Line Interface in the simple static model explained previously). This component is used to get

the orders from the system user and it is connected with the different source components (Email_Sources, Html_Sources, Rss_Sources), with User, Entries, Products and also with **DB**, the other core component.

- The second is the **DB** component (named Data Layer in simple static model explained previously, but with fewer functionalities in this case). This component interacts with the SyRAS database (syras.sqlite file) to store data, select, modify or delete them according to the actions requested by the other components: User, Entries, Products, or Sources.

All the components will be explained in more detail in next sections.

6.2 The DB component and the SyRAS database

The DB component is a wrapper class for sqlite database management and it is included in the file `mod_db.py` of the SyRAS system. It is used for managing the SyRAS sqlite database.

6.2.1 The SyRAS sqlite database

As the DB component manages the database, the SyRAS database and its structure will be explained in the following sections.

The database is located in a single file in the SyRAS directory (`syras.sqlite`) and its entity-relationship model, along with the data types are shown in the following figures.

❖ **USERS****Columns (6)**

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	id	INTEGER	1		1
1	date	DATETIME	1	CURRENT_TIMESTAMP	0
2	name	TEXT	1		0
3	password	TEXT	1		0
4	email	TEXT	0		0
5	warnings	INTEGER	0		0

Figure 8. Users table in SyRAS database

As shown in Figure 8, the database stores the information of all the users with permissions in the SyRAS system. For security reasons, passwords are encrypted when stored.

❖ **PRODUCTS****Columns (8)**

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	id	INTEGER	1		1
1	date	DATETIME	1	CURRENT_TIMESTAMP	0
2	brand	TEXT	0		0
3	name	TEXT	1		0
4	version	TEXT	0		0
5	general_search	TEXT	0		0
6	specific_search	TEXT	0		0
7	importance	INTEGER	0		0

Figure 9. Products table in SyRAS database

As shown in Figure 9, the database stores all the information of the products used in the SyRAS system.

❖ ENTRIES

Columns (12)

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	id	INTEGER	1		1
1	date	DATETIME	1	CURRENT_TIMESTAMP	0
2	permalink	TEXT	1		0
3	linked_source	INTEGER	1		0
4	subject	TEXT	1		0
5	body	TEXT	1		0
6	matched_string	TEXT	0		0
7	linked_product	INTEGER	0		0
8	automatic_cat	INTEGER	0		0
9	assigned_cat	INTEGER	0		0
10	by_user	INTEGER	0		0
11	md5	TEXT	0		0

Figure 10. Entries table in SyRAS database

As shown in Figure 10, the database stores all the information of the entries used in the SyRAS system.

There is an additional field called “md5” that stores the md5 hash of the permalink. It is used for improving the performance when checking if a new entry already exists.

- The field “linked_source” is a foreign key that points to the primary key “id” of the table SOURCES.
- The field “linked_product” is a foreign key that points to the primary key “id” of the table PRODUCTS.
- The field “by_user” is a foreign key that points to the primary key “id” of the table USERS.

❖ SOURCES

Columns (11)

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	id	INTEGER	1		1
1	date	DATETIME	1	CURRENT_TIMESTAMP	0
2	name	TEXT	1		0
3	source_type	INTEGER	1		0
4	type	INTEGER	1		0
5	subscription	TEXT	0		0
6	linked_product	INTEGER	0		0
7	search_brand	TEXT	0		0
8	url_start	TEXT	0		0
9	pattern	TEXT	0		0
10	last_checked	DATETIME	0		0

Figure 11. Sources table in SyRAS database

As shown in Figure 11, the database stores all the information of the different sources used in the SyRAS system to collect the information.

- The field “linked_product” is a foreign key that points to the primary key “id” of the table PRODUCTS.

6.2.2 The DB class

DB is a wrapper class for sqlite database management. This class is used by the rest of the classes every time they have to interact with the database. The DB class is contained in the module mod_db.py and its class diagram is shown in Figure 12.

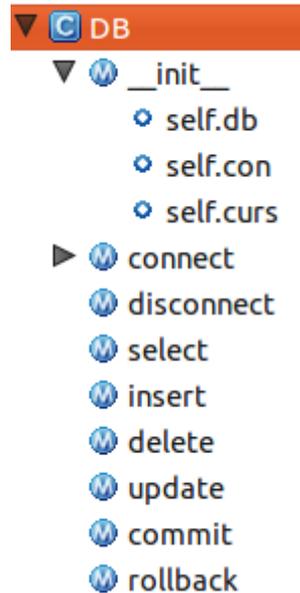


Figure 12. DB class diagram

Code samples and comments (after the symbol # or between “” and “”) from the module `mod_db.py` (in Appendix 2) describe the different attributes and methods explained in the following lines:

- The first thing to point out is the import statement in the first line; the module `sqlite3` is imported because it is needed for interacting with sqlite databases as it can be seen in the `connect` method. [35]
- The second thing to point out is the constructor `__init__`. It uses a default path and name for the database, but a different database file can be taken as an argument if needed.
- The class attributes are: `db` (representing the database), `curs` (representing its cursor) and `con` (representing its connection).

Next are the methods used for performing the most common operations when handling databases:

- The methods are: connect, disconnect, select, insert, delete, update, commit and rollback.
- Note that exception handling is used in most of the methods for treating properly the errors that can happen when performing operations with the database.

The DB is a very simple class that does not need further explanation.

6.3 The Users class

The Users class is contained in the module `mod_uses.py` and it is the one used for handling users. This class performs all the actions related with users and the user table in the SyRAS database. Its class diagram is as shown in Figure 13.



Figure 13. Users class diagram

Code samples and comments from the module `mod_users.py` (in Appendix 2) describe the different attributes and methods explained in the following sections:

- The first things to point out are the import statements in the first two lines. The module `sys` is imported because it is needed for changing the standard output to a file when saving data in the `save` method. On the other hand, the module `base64` is used for encrypting and decrypting data stored in the database. [36] [37]
- The second thing to point out is the constructor `__init__`. It uses an instance of the database as an argument when called.
- The class attributes are equal to fields in the database table `USERS`:
 - `id` (representing the user id)
 - `date` (representing the user date of creation)
 - `name` (representing the user name)
 - `password` (representing the user password for gaining access to the system)
 - `email` (representing the user email to send alerts if configured)
 - `warnings` (representing the user warning level).

Next are the methods used for performing operations with users. They are related with the operations that can be performed from the Command Line Interface and also with the operations for managing users in the database.

- The methods are:
 - `get_one`: Gets one user with known id
 - `get_all`: Gets all users from the SyRAS database.
 - `exists`: Returns the user with known user and password if exists.
 - `insert`: Insert “self” user checking if it exists already.

- delete: Delete “self” user if it exists.
 - modify: Update the user with self values.
 - show: Show “self” user attributes on the command line interface.
 - save: Save “self” user attributes on a text file.
- Note that exception handling is used in most of the methods for treating properly the errors that can happen when performing the actions.

The User class implements methods very similar to the ones used in other classes explained in next sections.

6.4 The Products class

The Products class is contained in the module `mod_product.py` and it is the one used for handling products. This class performs all the actions related with products and the products table in the SyRAS database. Its class diagram is as shown in Figure 14.

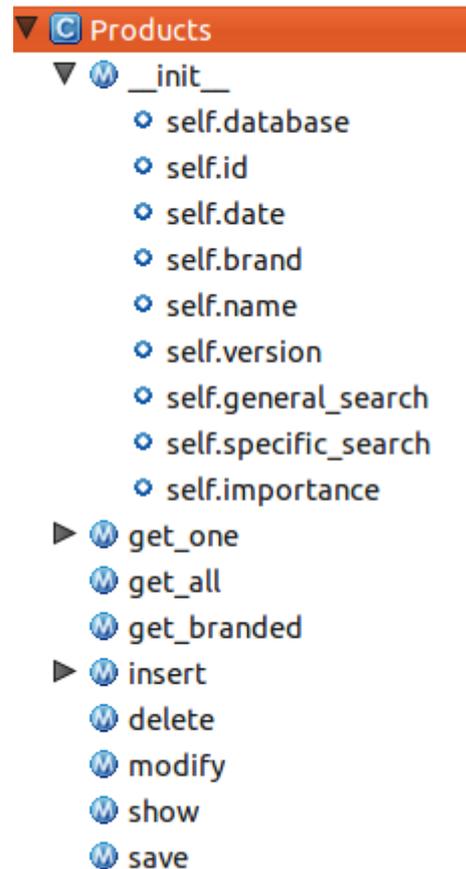


Figure 14. Products class diagram

Code samples and comments from the module `mod_products.py` (in Appendix 2) describe the different attributes and methods are explained in the following sections:

- The first thing to point out is the import statement in the first line. The module `sys` is imported because it is needed for changing the standard output to a file when saving data in the `save` method.
- The second thing to point out is the constructor `__init__`. It uses an instance of the database as an argument when called.
- The class attributes are equal to the fields in the database table **PRODUCTS**:
 - `id` (representing the product id)

- date (representing the product date of creation)
- brand (representing the product brand name)
- name (representing the product name)
- version (representing the product version)
- general_search (representing the product search string when looking for it in entries coming from a general type source)
- specific_search (representing the product search string when looking for it in entries coming from a specific type source)
- importance (representing how relevant the product is).

Next are the methods used for performing operations with products. They are related with the operations that can be performed from the Command Line Interface and also with the operations for managing products in the database.

- The methods are:
 - get_one: Gets one product with known id
 - get_all: Gets all products from the SyRAS database.
 - get_branded: Get the id of all products with same given brand from the database.
 - insert: Insert “self” product checking if it exists already.
 - delete: Delete “self” product if it exists.
 - modify: Update the product with self values.
 - show: Show “self” product attributes on the command line interface.
 - save: Save “self” product attributes on a text file.
- Note that exception handling is used in most of the methods for treating properly the errors that can happen when performing operations.

The Products class is not very complicated and it implements methods very similarly to the ones used in the class Users explained previously.

6.5 The Sources class

The Sources class (full class diagram in Figure 16) is contained in the module `mod_sources.py` and it is the one used for handling sources. This class performs general actions related with products and the products table in the SyRAS database. This class represents the parent class used by `Html_Sources`, `Email_Sources` and `Rss_Sources` as it can be seen in the reduced class diagram showed in Figure 15:

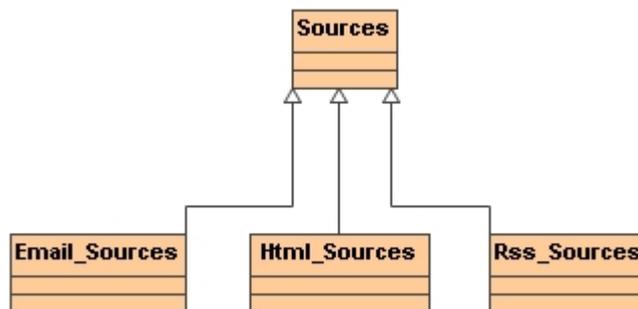


Figure 15. Simple Sources class diagram showing inheritance

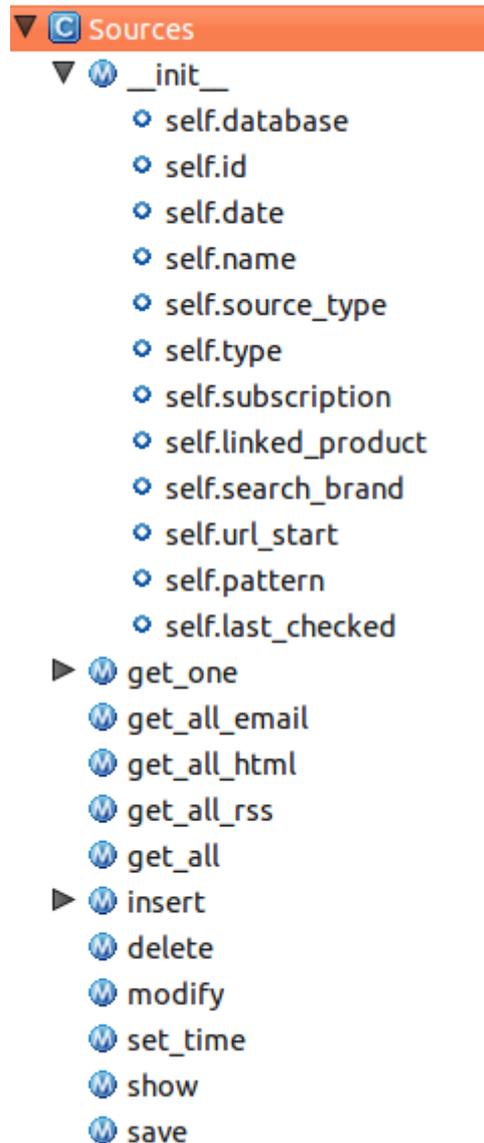


Figure 16. Sources class diagram

Code samples and comments from the module `mod_sources.py` (in Appendix 2) describe the different attributes and methods explained next:

- The first thing to point out is the import statement in the first line. The module `sys` is imported because it is needed for changing the standard output to a file when saving data in the `save` method.

- The second thing to point out is the constructor `__init__`. It uses an instance of the database as an argument when called.

- The class attributes are equal to the fields in the database table **SOURCES**:

- `id` (representing the product id)
- `date` (representing the product date of creation)
- `name` (representing the source name)
- `source_type` (representing the `source_type` of the source: email, html or rss)
- `type` (representing the type of the source: general or specific)
- `subscription` (representing the URL to subscript for this source alerts)
- `linked_product` (if the source is specific and it could be associated to just one product, gets the product id, otherwise is 0)
- `search_brand` (If its and specific source, it specifies the brand of products to search for association).
- `url_start` (This attribute is only used in `sources_type HTML` and it gives the URL to start looking for new entries).
- `pattern` (If `source_type=2`, this attribute gives the Regex used for searching new entries from this source).
- `last_checked` (It represents the timestamp when the source was checked for new entries for the last time).

Next are the methods used for performing operations with Sources. They are related with the operations that can be performed from the Command Line Interface and also with the operations for managing Sources in the database. Most of the methods are generic and used by the subclasses `Html_Sources`, `Rss_Sources` and `Email_sources`.

- The methods are:
 - `get_one`: Gets one source with known id
 - `get_all_email`: Gets all email sources from the SyRAS database
 - `get_all_html`: Gets all HTML sources from the SyRAS database
 - `get_all_rss`: Gets all RSS sources from the SyRAS database
 - `get_all`: Gets all sources from the SyRAS database
 - `insert`: Inserts “self” source checking if it exists already.
 - `delete`: Deletes “self” source if it exists.
 - `modify`: Updates the source with self values.
 - `set_time`: Updates the `last_checked` field of the source to current time.
 - `show`: Shows “self” source attributes on the command line interface.
 - `save`: Saves “self” source attributes on a text file.

- Note that exception handling is used in most of the methods for treating properly the errors that can happen when performing the operations.

The Sources class is very important because is the superclass used by `Html_Sources`, `Email_Sources` and `Rss_Sources` to perform the most common actions. These three classes are representing the core of the SyRAS system because they perform the actions to collect the new entries and insert them in the database, main functionality of the system.

The next step is to explain the subclasses of the Sources parent class.

6.5.1 The Email_Sources class

The Email_Sources class is contained in the module mod_email.py and it is the one used for handling sources collected via email (source_type=1). This class inherits from the Sources class and implements other methods needed for collecting e-mail alerts.

The class diagram of Email_Sources is as shown in Figure 17.

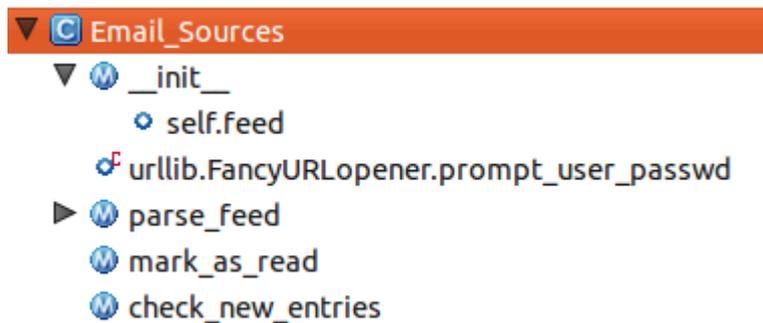


Figure 17 Email_Sources class diagram

Code samples and comments from the module mod_email.py (in Appendix 2) describe the different attributes and methods explained in the following:

- It is important to explain the import statements in the first lines:
 - Import Sources class: due to it will be used as superclass by Email_Sources inheriting methods and attributes.
 - Import Entries class; it is needed for saving the new entries collected.

- Import feedparser: a module for parsing the atom entries. In this case, it is used for accessing the new entries because the gmail inbox is accessed and parsed as rss. [38]
- Import urllib :a module for handling and accessing URLs. [39]
- Import imaplib: a module for handling IMAP protocol that allows handling email requests. [40]
- GMAIL_ATOM_URL is a constant that stores the URL to access gmail accounts as atom RSS.
- As it can be seen in the class definition (class Email_Sources(Sources):) Email_Sources inherits from Sources.
- gmail_user and gmail_pass are the username and password for the gmail account. In this case they are embedded in the code in plain text because the email does not store any personal information, although it would be more correct to cipher and store them in a safer place.
- The second thing to point out is the constructor `__init__`.
 - It uses an instance of the database as an argument when called.
 - It uses the superclass creator.
- The class attributes are inherited from the parent class Sources, adding a new one: feed (representing the inbox parsed as RSS newsfeed using the module feedparser).

Next are the new methods introduced by the class and used for performing operations with the email sources. The main objective of this class and its methods is to collect new entries coming from mailing lists.

- The first thing to point out is that we are using FancyURLopener, that subclasses URLopener providing default handling for 401 response codes (authentication required), performing basic HTTP authentication. The `prompt_user_passwd()` method was overridden

using a lambda function to provide the authentication required (username and password) to access the gmail account. [41][42]

- The methods are:
 - `parse_feed`: It parses the email source using `feedparser` and stores it in the attribute `feed`.
 - `mark_as_read`: It is used to mark as read all the email entries in the inbox once they are introduced into the database using the following method.
 - `check_new_entries`: It collects all new entries coming from e-mail and introduces them into the SyRAS database as new entries, checking previously if they already exist in the database.
- Note that exception handling is used in most of the methods for treating properly the errors that can happen when performing the operations.

As explained before, the class `Email_Sources` inherits from the `Sources` class and it introduces in the database all new entries collected via e-mail.

6.5.2 The `Html_Sources` class

The `Html_Sources` class is contained in the module `mod_html.py` and it is the one used for handling sources collected visiting websites (`source_type=2`). This class inherits from the `Sources` class and implements other methods needed for collecting html alerts straight from websites.

The class diagram of `Html_Sources` is as shown in Figure 18.

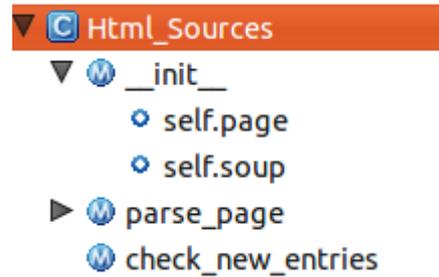


Figure 18. Html_Sources class diagram

Code samples and comments from the module `mod_html.py` (Appendix 2) describe the different attributes and methods explained below:

- It is important to explain the import statements in the first lines:
 - Import Sources class, because it will be used as superclass by Html_Sources inheriting methods and attributes.
 - Import Entries class, it is needed for saving the new entries collected.
 - Import BeautifulSoup, a module for parsing HTML websites. In this case, it is used for accessing the html sources and it parses them in a given way that enables extracting the new security alerts. [43]
 - Import re, a module for using regular expressions. [44]
 - Import urllib2, a module for handling and accessing URLs a bit more complete than urllib. [45]
- As it can be seen in the class definition (`class Html_Sources(Sources):`) Html_Sources inherits from Sources.
- The second thing to point out is the constructor `__init__`.
 - It uses an instance of the database as an argument when called.
 - It uses the superclass creator.
- The class attributes are inherited from the parent class Sources, adding two new ones:

- page (representing the downloaded html code of the webpage).
- soup (representing the website parsed using the module BeautifulSoup).

Next are the new methods introduced by the class and used for performing operations with the HTML sources. The main objective of this class and its methods is to collect new entries coming from the source websites.

- The methods are:
 - parse_page: It parses the web source using BeautifulSoup and stores it in the attribute soup.
 - check_new_entries: It collects all new entries coming from a website and introduces them into the SyRAS database as new entries, checking previously if they already exist in the database.
- Note that exception handling is used in most of the methods for treating properly the errors that can happen when performing the operations.

As explained before, the class `Html_Sources` inherits from the `Sources` class and it introduces in the database all new entries collected from website sources.

6.5.3 The `Rss_Sources` class

The `Rss_Sources` class (Figure 19) is contained in the module `mod_rss.py` and it is the one used for handling sources collected via RSS (`source_type=3`). This

class inherits from the Sources class and implements other methods needed for collecting email alerts.

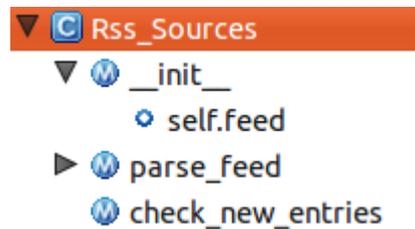


Figure 19. Rss_Source class diagram

Code samples and comments from the module `mod_rss.py` (Appendix 2) describe the different attributes and methods explained below:

- It is important to explain the import statements in the first lines:
 - Import Sources class, it will be used as superclass by Rss_Sources inheriting methods and attributes.
 - Import Entries class, it is needed for saving the new entries collected.
 - Import feedparser, a module for parsing the RSS entries from newsfeeds. Import urllib, a module for handling and accessing URLs.
- As it can be seen in the class definition (`class Rss_Sources(Sources):`) Rss_Sources inherits from Sources.
- The second thing to point out is the constructor `__init__`.
 - It uses an instance of the database as an argument when called.
 - It uses the superclass creator.
- The class attributes are inherited from the parent class Sources, adding a new one: `feed` (representing the RSS newsfeed parsed using the module `feedparser`).

Next are the new methods introduced by the class and used for performing operations with RSS sources. The main objective of this class and its methods is to collect new entries coming from RSS newsfeeds.

- The methods are:
 - `parse_feed`: It parses the email source using `feedparser` and stores it in the attribute `feed`.
 - `check_new_entries`: It collects all new entries coming from RSS and introduces them into the SyRAS database as new entries, checking previously if they already exist in the database. This method also checks every RSS source last update timestamp to avoid operating again entries already processed.
- Note that exception handling is used in the last method for treating properly the errors that can happen when performing the operations.

As explained before, the class `Rss_Sources` inherits from the `Sources` class and it introduces in the database all new entries collected via RSS.

6.6 The Entries class

The `Entries` class is contained in the module `mod_entries.py` and it is the one used for handling products. This class performs all the actions related with entries and the entries table in the SyRAS database. Its class diagram is as shown in Figure 20.

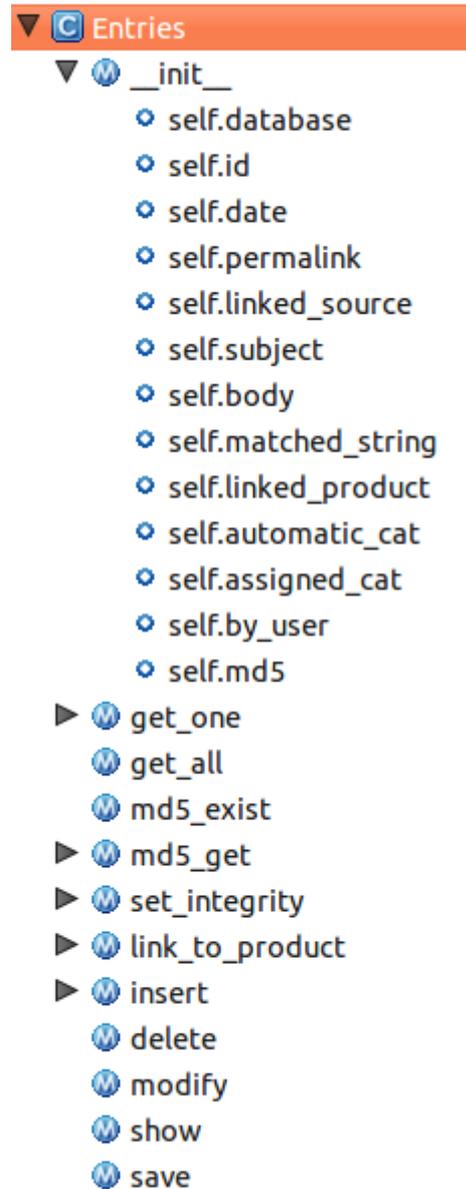


Figure 20. Entries class diagram

Code samples and comments from the module `mod_entries.py` (Appendix 2) describe the different attributes and methods explained below:

- The first things to point out are the import statements in the first lines.
 - The module `hashlib` is imported because it is needed for using the `md5` hash in one of the attributes. [46]

- The module `sys` is imported because it is needed for changing the standard output to a file when saving data in the `save` method.
- The classes `Sources` and `Products` are also imported because an instance of them will be used in some of the methods of this class.
- The second thing to point out is the constructor `__init__`. It uses an instance of the database as an argument when called.
- The class attributes are equal to the fields in the database table **ENTRIES**:
 - `id` (representing the entry id)
 - `date` (representing the entry date of creation)
 - `permalink` (representing the entry permanent link brand name)
 - `linked_source` (representing the source id where the entry comes from)
 - `subject` (representing the subject or title of the entry)
 - `body` (representing the body text of the entry)
 - `matched_string` (representing the string that matched if the entry is associated to a product)
 - `linked_product` (representing the id of the associated product if any).
 - `automatic_cat` (representing the entry importance level assigned automatically)
 - `assigned_cat` (representing how relevant the entry is in a level assigned by the user).
 - `by_user` (representing the user that introduced or last modified the entry)
 - `md5` (representing the md5 hash of the entry's permalink).

Next are the methods used for performing operations with entries. They are related with the operations that can be performed from the Command Line Interface and also with the operations for managing entries in the database.

Note that most parts of the code have been omitted because these methods work in very similar way to the ones described in previous sections.

- The methods are:
 - `get_one`: Gets one product with known id
 - `get_all`: Gets all products from the SyRAS database.
 - `md5_exist`: Checks if a given entry exists in the database.
 - `md5_get`: Gets the md5 hash of the permalink of an entry.
 - `set_integrity`: Sets to 0 some values in the “self” entry to avoid operation problems.
 - `link_to_product`: This is one of the most important methods. It tries to link the new entry to an existing product using different operations depending on the source type (general or specific) and also depending if it can be associated to one or several products.
*For more detailed information review the code comments.
 - `insert`: Insert “self” entry checking if it exists already.
 - `delete`: Delete “self” entry if it exists.
 - `modify`: Update the entry with self values.
 - `show`: Show “self” entry attributes on the command line interface.
 - `save`: Save “self” entry attributes on a text file.

- Note that exception handling is used in most of the methods for treating properly the errors that can happen when performing operations.

The Entries class is not very complicated and it implements methods very similarly to the ones used other classes explained previously. The most important modification is the addition of the field md5 for checking if an entry is already in the system and the method link_to_product that tries to link the new entry to product in the database using search strings.

6.7 The Console class

The Console class is contained in the module mod_syras.py and it is the one used by the user for interacting with the SyRAS system, it is the class that creates the command line interface. Its class diagram is as shown in Figure 21.

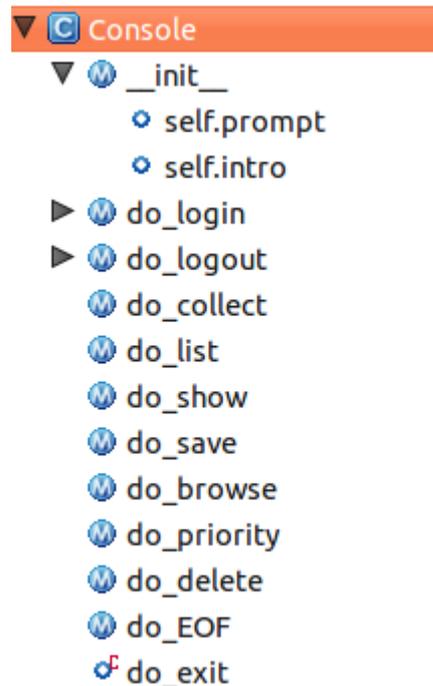


Figure 21. Console class diagram

Code samples and comments from the module `mod_syras.py` (Appendix 2) describe the different attributes and methods explained below:

- The first things to point out are the import statements in the first lines:
 - The module `sys` is imported because it is needed for closing the SyRAS system.
 - The module `cmd` is imported because it is needed for creating the console (Command Line Interface). [47]
 - The module `webbrowser` is imported because it is needed for opening permalinks (URLs) in the default web browser. [48]
 - The module `getpass` is needed for getting the passwords from the console in a safe way. [49]
 - The rest of the imported classes have been explained before and they will be used inside the methods of the console class.
- The console class inherits most part of its behavior from the class `Cmd` included in `cmd`, as it can be seen because it is used in its creator. `cmd.Cmd.__init__(self)`.
- The class attributes are:
 - `prompt` (representing the prompt used in the console. It changes to represent the user logged in)
 - `intro` (representing the SyRAS welcome message).

Next are the methods used for performing operations with console. They are related with the operations that can be performed by the user from the Command Line Interface to interact with the SyRAS system.

- First of all it is important to note that all methods start with `do_` with means that are methods that can be called from the console.

- Another important thing to point out is that the description of the methods located right after the name is the one shown when using the help command and the name of the method.
- The methods are:
 - do_login: Used for login a user requesting for username and password.
 - do_logout: Logs out the current user.
 - do_collect: Collects entries from different sources: RSS, HTML or email.
 - do_list: Lists all entries, products, sources or users from the database.
 - do_show: Shows in the console all the information of one entry, product, source or user with given id.
 - do_save: Saves in a text file all the information of one entry, product, source, or user with given id.
 - do_browse: Opens one entry or source permalink in the web browser.
 - do_priority: Changes to important the priority of a given entry or product.
 - do_delete: Deletes one entry, product, source or user from the database. The admin is the only user allowed to perform this operation.
 - do_eof / do_exit: Closes the SyRAS system.

The Console class implements methods needed by the user to interact with the SyRAS system. The Console class has some properties inherited from its superclass cmd, such as auto completion of commands and command history, apart from the already mentioned help command.

6.8 The syras.py and syrasloop.py modules

The `syras.py` and the `syrasloop.py` modules are both runnable programs because they contain a main function. They are the ones used for starting the SyRAS system but they work in two different ways, explained in the following sections.

6.8.1 The `syras.py`

In addition to the `console` class, the `syras.py` file or module also hosts a main function that is used to start the SyRAS system using the command line interface.

Code samples and comments from the module `syras.py` (Appendix 2) describe the different attributes and methods explained below:

- The first line is: `if __name__ == '__main__':` which means that the `syras.py` module can be called as an executable script and then it will start the execution from the main function.
- The next line is the initialization of the database and the rest of the class instances that will be used along the module.
- The last line is the creation of the console so the user can interact with the system. The console will be running in a loop (`con.cmdloop()`) until the user exits or uses the keyboard interrupt (CTRL + C) command.
- Finally, it is important to remember that all the needed imports were made at the beginning of the module. They were explained in previous sections when the console class was described.

To sum up, the `syras.py` module has the main function used to launch the SyRAS system with the regular behavior explained previously.

6.8.2 The `syrasloop.py`

The `syrasloop.py` is another module that can be used as an executable to start the SyRAS system, but with reduced functionality. When executed from `syrasloop.py`, SyRAS will work as a “daemon” running in the background. Its only task will be to check for new entries and collect them, then the system will be inactive for a time lapse and it will wake up and check for new entries again. This process will be repeated in an infinite loop until the user stops it.

Code samples and comments from the module `syrasloop.py` (Appendix 2) describe the different attributes and methods explained below:

- The first lines are the needed imports, `time` is the only new import. [50]
- Then the main function and the initializations have been explained in the previous section.
- First of all, the system asks the user to log in.
- Once the login is confirmed the loop start checking for new entries from email, HTML and RSS sources. Any new entry is inserted into the SyRAS database after trying the association with the existing products.
- Then the system sleeps for a given time and start checking again in an infinite loop until the user stops the system.

As explained before, `syrasloop.py` is used by a logged user only for collecting new entries, but it cannot be used to interact with the system.

7 INSTALLATION, TESTING AND USER MANUAL

7.1 Installation

The installation of the SyRAS system is very easy, once we have a computer with Ubuntu 9.x or greater, because we just have to copy the source python source code and the database file, and add couple of modules needed for the proper functioning of the SyRAS system.

As explained previously in the section “Non-functional requirements”, the SyRAS system must be executed in a PC with the operative system Ubuntu 9.04 or greater, with Python interpreter 2.5 or greater and with database SQLite 3.6 or greater. The system must also have Internet access.

Once the host computer follows the requirements, we need to copy the SyRAS source code (included in the “src” folder) and database files in the host, as shown in Figure 22.

▼ src		10 elementos carpeta
mod_db.py	2,5 KiB	script en Python
mod_email.py	3,9 KiB	script en Python
mod_entries.py	12,0 KiB	script en Python
mod_html.py	2,5 KiB	script en Python
mod_products.py	6,7 KiB	script en Python
mod_rss.py	2,6 KiB	script en Python
mod_sources.py	9,9 KiB	script en Python
mod_users.py	5,8 KiB	script en Python
syras.py	13,0 KiB	script en Python
syrasloop.py	3,0 KiB	script en Python
syras.sqlite	235,0 KiB	base de datos SQLite3

Figure 22. SyRAS files and source code

There are a few Python modules needed for the proper functioning of the SyRAS system:

- Feedparser (Universal Feed Parser): Used for parsing RSS feeds.
 - Version: feedparser 4.1
 - Installation:
 - Download and unzip the file feedparser-4.1.zip
 - Go to the unzipped folder and run the command:


```
$ sudo python setup.py install
```
 - Now feedparser is installed and ready to use.
- BeautifulSoup: Python HTML/XML parser
 - Version: Beautiful Soup 3.1.0.1
 - Installation:
 - Download the file BeautifulSoup-3.1.0.tar.gz
 - Unzip the file and include both files (BeautifulSoup.py in the SyRAS “src” folder)
 - Now BeautifulSoup is installed and ready to use.

No additional components are needed, so the SyRAS system is now ready to be executed.

7.2 SyRAS testing and user manual

Once installed, the next step is to test the whole system to make sure it works properly. The user manual will be explained at the same time, using the help commands included in the command line interface. Several screenshots will be used for that purpose.

7.2.1 Starting SyRAS

The screenshot in Figure 23 shows the SyRAS folder, the starting command and the general help option for SyRAS.

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
pableras@pableras-pc:~/Escritorio/syras$ ls
src  syras.sqlite
pableras@pableras-pc:~/Escritorio/syras$ cd src
pableras@pableras-pc:~/Escritorio/syras/src$ ls
BeautifulSoup.py  mod_email.py      mod_products.py  mod_users.py
mod_console.py    mod_entries.py    mod_rss.py       syrasloop.py
mod_db.py         mod_html.py       mod_sources.py   syras.py
pableras@pableras-pc:~/Escritorio/syras/src$ python syras.py
Welcome to SyRAS command line interface
      Please login or type 'help' for command guide of 'exit' to quit
SyRAS: _> help

Documented commands (type help <topic>):
=====
browse  collect  delete  list  login  logout  priority  save  show

Undocumented commands:
=====
EOF  exit  help

SyRAS: _>

```

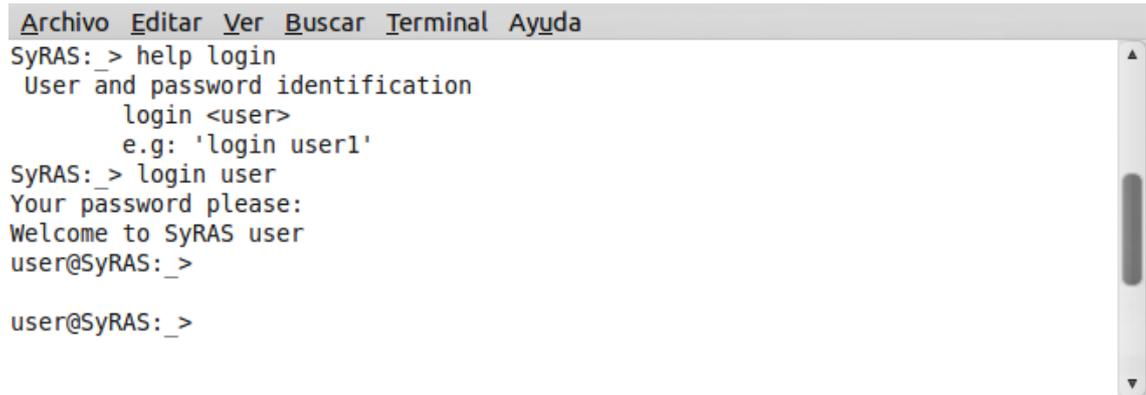
Figure 23. SyRAS file system, execution and help command

The way to start the `syrasloop.py` (the module for collecting new entries in an infinite loop) is similar to starting `syras.py` described previously. The command would be: `$ python syrasloop.py`.

7.2.2 Using SyRAS

The following screenshots, in Figures 24-39 show how to invoke the different commands of the SyRAS system using the command line interface.

- User login instruction and documentation, Figure 24:

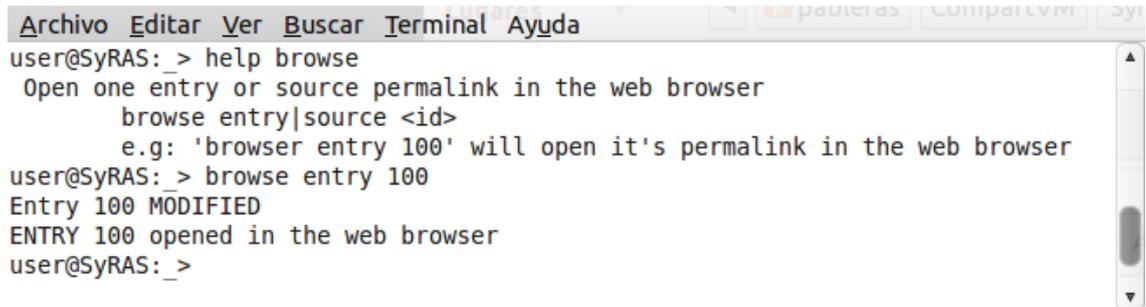
A terminal window with a menu bar containing 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal text shows a user logging in to SyRAS. The user enters 'help login' and receives instructions: 'User and password identification', 'login <user>', and an example 'e.g: 'login user1''. Then the user enters 'login user', is prompted for a password, and receives a welcome message: 'Welcome to SyRAS user'. The prompt changes to 'user@SyRAS: _>' and the user enters another prompt 'user@SyRAS: _>'.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
SyRAS: _> help login
User and password identification
      login <user>
      e.g: 'login user1'
SyRAS: _> login user
Your password please:
Welcome to SyRAS user
user@SyRAS: _>

user@SyRAS: _>
```

Figure 24. SyRAS login command

- Browse instruction and documentation, Figure 25:

A terminal window with a menu bar containing 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', 'Ayuda', 'pabieras', 'CompararVM', and 'Sy'. The terminal text shows a user using the 'browse' command. The user enters 'help browse' and receives instructions: 'Open one entry or source permalink in the web browser', 'browse entry|source <id>', and an example 'e.g: 'browser entry 100' will open it's permalink in the web browser'. Then the user enters 'browse entry 100', and receives the output: 'Entry 100 MODIFIED' and 'ENTRY 100 opened in the web browser'. The prompt changes to 'user@SyRAS: _>' and the user enters another prompt 'user@SyRAS: _>'.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda  pabieras  CompararVM  Sy
user@SyRAS: _> help browse
Open one entry or source permalink in the web browser
      browse entry|source <id>
      e.g: 'browser entry 100' will open it's permalink in the web browser
user@SyRAS: _> browse entry 100
Entry 100 MODIFIED
ENTRY 100 opened in the web browser
user@SyRAS: _>

user@SyRAS: _>
```

Figure 25- SyRAS browse command

And the result of the browse command is shown in Figure 26.

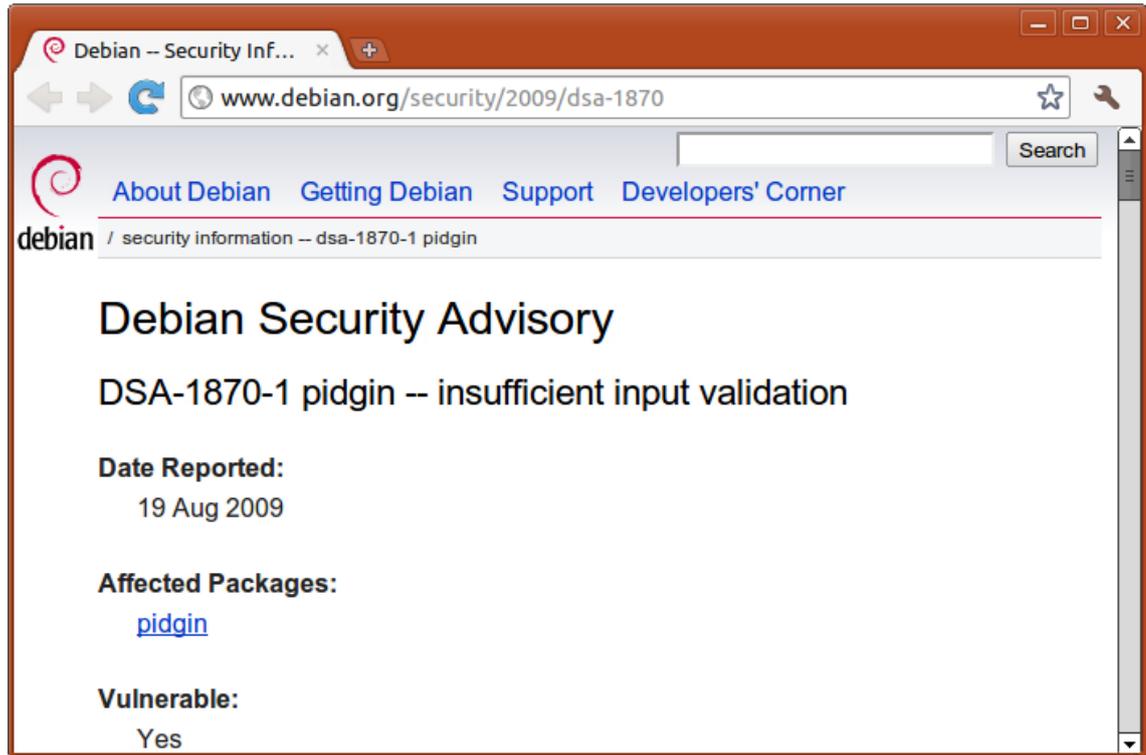


Figure 26. Browse command result

- Collect instruction and documentation, Figure 27:

```

Archivo  Editor  Ver  Buscar  Terminal  Ayuda
user@SyRAS: ~ > help collect
Collect entries from different sources
collect email|html|rss|all
e.g: 'collect rss'
user@SyRAS: ~ > collect email
Collecting email
New entry INSERTED: 'VUPEN Advisories and Exploits: Debian' 'http://mail.google.
com/mail?account_id=syrasfeed@gmail.com&message_id=12e1a97de9882bf8&view=conv&ex
tsrc=atom'
Entry 435 ASSOCIATED to 3 with string 'debian'
Entry 435 MODIFIED
Source 'generic email' updated
All email sources collected
user@SyRAS: ~ >

```

Figure 27. SyRAS collect command

- Delete instruction and documentation, Figure 28:

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
admin@SyRAS: > help delete
Delete one entry, product, source or user from the database (only admin)
delete entry|product|source|user <id>
e.g: 'delete user 3' will delete it from the database
admin@SyRAS: > delete entry 99
Entry 99 DELETED
admin@SyRAS: >

```

Figure 28. SyRAS delete command

- List instruction and documentation, Figure 29:

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
user@SyRAS: > help list
List all entries, products, sources or users
list entries|products|sources|users
e.g: 'list products'
user@SyRAS: > list users
Printing USERS:
(id, date, name, (password ommited), email, warning level)

(0, u'2010-10-17 17:57:59', u'name', u'email@email.com', 1)
(2, u'2009-07-24 11:50:19', u'Peps', u'pajb@hotmail.com', 2)
(5, u'2010-08-08 17:31:57', u'user', u'pajb@hotmail.com', 2)
(8, u'2010-11-01 13:04:43', u'admin', u'admin@admin.com', 2)
(10, u'2010-11-01 18:46:55', u'user', u'user@user.com', 1)
user@SyRAS: >

```

Figure 29. SyRAS list command

- Logout instruction and documentation, Figure 30:

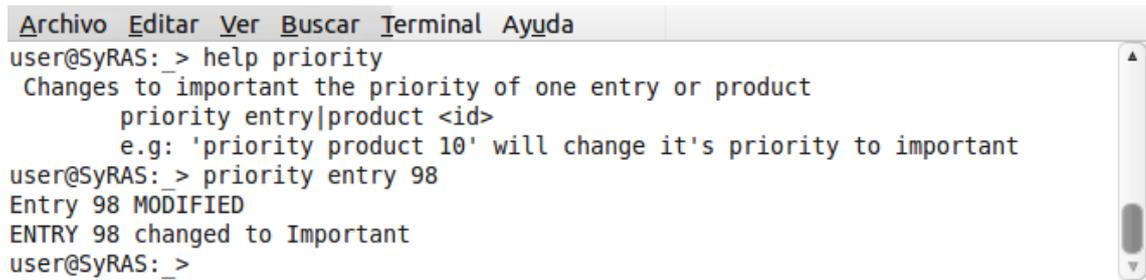
```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
user@SyRAS: > help logout
Logs out current user if any
user@SyRAS: > logout
User user logged out
SyRAS: >

```

Figure 30. SyRAS logout command

- Priority instruction and documentation, Figure 31:



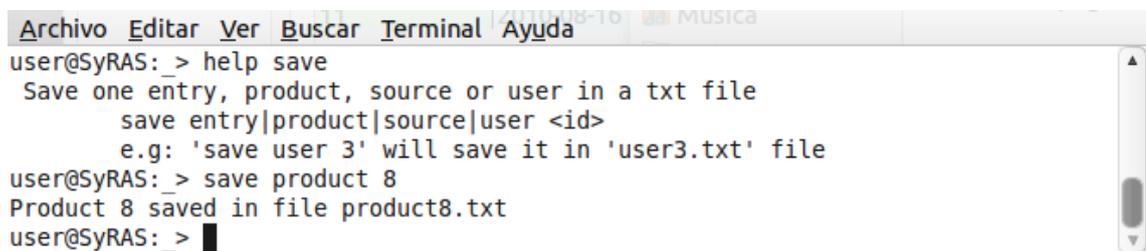
```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
user@SyRAS: ~ > help priority
Changes to important the priority of one entry or product
priority entry|product <id>
e.g: 'priority product 10' will change it's priority to important
user@SyRAS: ~ > priority entry 98
Entry 98 MODIFIED
ENTRY 98 changed to Important
user@SyRAS: ~ >

```

Figure 31. SyRAS priority command

- Save instruction and documentation, Figure 32:



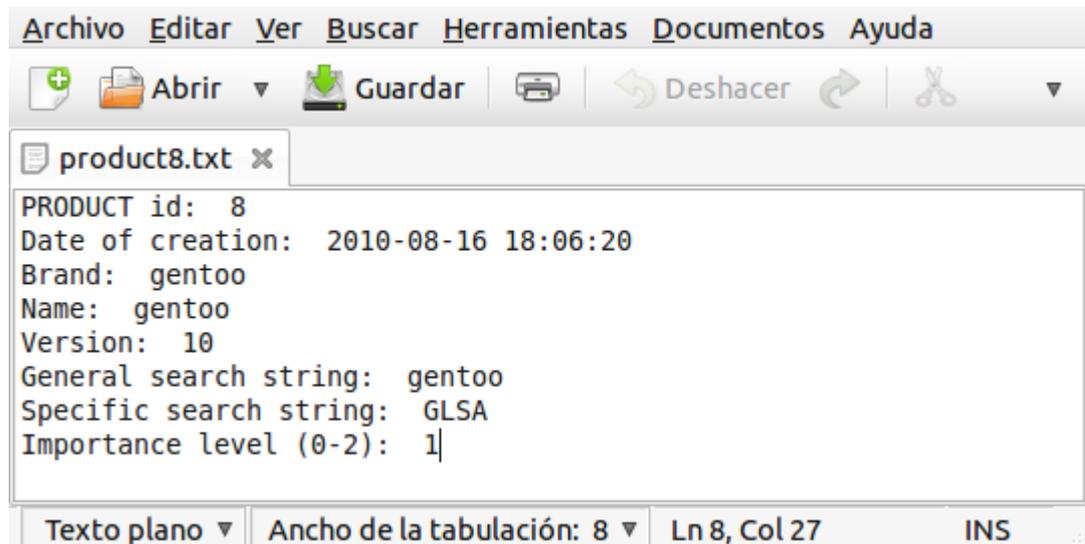
```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
user@SyRAS: ~ > help save
Save one entry, product, source or user in a txt file
save entry|product|source|user <id>
e.g: 'save user 3' will save it in 'user3.txt' file
user@SyRAS: ~ > save product 8
Product 8 saved in file product8.txt
user@SyRAS: ~ > █

```

Figure 32. SyRAS save command

and the result of the save command is shown in Figure 33.



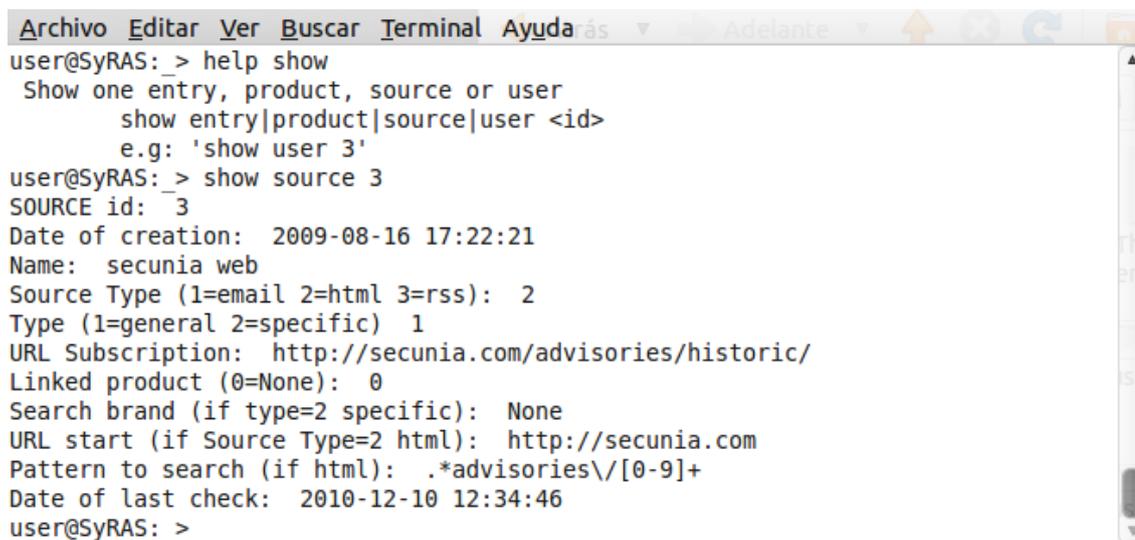
```

Archivo  Editar  Ver  Buscar  Herramientas  Documentos  Ayuda
+ Abrir  Guardar  Deshacer
product8.txt x
PRODUCT id: 8
Date of creation: 2010-08-16 18:06:20
Brand: gentoo
Name: gentoo
Version: 10
General search string: gentoo
Specific search string: GLSA
Importance level (0-2): 1
Texto plano  Ancho de la tabulación: 8  Ln 8, Col 27  INS

```

Figure 33.SyRAS save command result

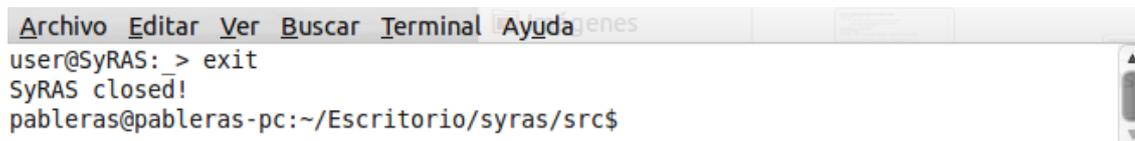
- Show instruction and documentation, Figure 34:

A screenshot of a terminal window with a menu bar at the top containing 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal text shows a user at the SyRAS prompt typing 'help show' and receiving a list of options: 'Show one entry, product, source or user' and 'show entry|product|source|user <id>' with an example 'show user 3'. The user then types 'show source 3' and receives detailed information for source ID 3, including creation date (2009-08-16 17:22:21), name (secunia web), source type (2), type (1), URL subscription (http://secunia.com/advisories/historic/), linked product (0), search brand (None), URL start (http://secunia.com), and search pattern (.*advisories\[0-9]+). The terminal ends with the user prompt 'user@SyRAS: _>'.

```
user@SyRAS: _> help show
  Show one entry, product, source or user
    show entry|product|source|user <id>
    e.g: 'show user 3'
user@SyRAS: _> show source 3
SOURCE id: 3
Date of creation: 2009-08-16 17:22:21
Name: secunia web
Source Type (1=email 2=html 3=rss): 2
Type (1=general 2=specific) 1
URL Subscription: http://secunia.com/advisories/historic/
Linked product (0=None): 0
Search brand (if type=2 specific): None
URL start (if Source Type=2 html): http://secunia.com
Pattern to search (if html): .*advisories\[0-9]+
Date of last check: 2010-12-10 12:34:46
user@SyRAS: _>
```

Figure 34. SyRAS show command

- Exit instruction and documentation, Figure 35:

A screenshot of a terminal window with a menu bar at the top containing 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal text shows a user at the SyRAS prompt typing 'exit', followed by the message 'SyRAS closed!' and the shell prompt 'pableras@pableras-pc:~/Escritorio/syras/src\$'.

```
user@SyRAS: _> exit
SyRAS closed!
pableras@pableras-pc:~/Escritorio/syras/src$
```

Figure 35. SyRAS exit command

8 CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

As explained previously, the main goal of the project was to build a centralized system able to collect multiple security advisories from different sources, parse them, save them (in a database with a standard structure) and link them to the product affected by the vulnerability. All those tasks have been completed along the implementation of the project giving a fully functional system that collects and provides all the information needed to compose and send a new security advisory, and most importantly, everything is in one place and updated automatically with the new advisories collected from the sources.

The part responsible for harvesting the advisories consisted of three modules: E-mail, Web and RSS. Those three modules represent the core of the SyRAS system.

The chapters of this thesis showed the development of the SyRAS application from scratch, using Python, a database, web technologies and a command line interface. The Python programming language was suitable and helpful for this thesis purpose, because it is complemented with lots of extra modules that were used during the coding, saving a lot of time and effort. The rest of the technologies used served as a good complement for building the application and they were easy to work with because they are standard and widely used.

To combine and integrate the different modules used in SyRAS was not an easy task, but the result was a robust and handy application, easy to use and with several options to be improved in future works.

From the personal point of view, I could say that I have learned a lot while working on the different stages of the project: I have learned a new programming language, database handling and also many different tools useful for developing a software application. I have learned about the software analysis and design process and how to write a good project review in English. Writing this thesis has been a long and hard work, but also a very valuable experience.

8.2 Future work

Throughout the development of the project, I figured out many improvements and possible extensions that could be carried out. The following are the most important:

8.2.1 Addition of new sources

The SyRAS database includes some real sources that have been used for testing the system. These could be taken as an example to add new sources of the different types the system is ready to handle (Email, HTML and RSS) or to implement a new module (inheriting from Sources) able to add other sources, such as Twitter accounts, for example.

The addition of sources should be complemented with the addition of other products affected by the new alerts.

8.2.2 User alerts

Another way to improve the system in the future is to add an alerts system. This could send an e-mail, or even a SMS, to a given address when the system collects a new entry that is especially relevant.

The database already stores the email of the SyRAS users, so this improvement will not be difficult to code.

8.2.3 New body scripts

Nowadays the system only collects the new entries and tries to link them to an existing product. The next step could be to code some specific scripts able to do a series of transformations on them to get the body of the alert in a specific format. In addition, scripts can be applied to obtain the additional identifier CVE or some other relevant information.

8.2.4 Web interface

As a major improvement, the system could be migrated to a web platform based in LAMP (Linux, Apache, MySQL and PHP) technologies. In this way the system would become more professional and able to handle considerably more information in an efficient way.

REFERENCES

- [1] Kinkus, Jane F. Fall 2002. Computer Security. Purdue University, [www-document] consulted on 07.01.2011. Available at: <http://www.library.ucsb.edu/istl/02-fall/internet.html>
- [2] Shirey, R. May 2000. Internet Security Glossary. Internet Engineering Task Force RFC 2828. The Internet Society. [www-document] consulted on 08.01.2011. Available at: <http://tools.ietf.org/html/rfc2828>
- [3] Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal
- [4] Real Decreto 1720/2007, de 21 de diciembre, por el que se aprueba el Reglamento de desarrollo de la Ley Orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal
- [5] FreeBSD Security Advisories. 1995-2011. FreeBSD Handbook. The FreeBSD Documentation Project. [www-document] consulted on 08.01.2011. Available at: <http://www.freebsd.org/doc/en/books/handbook/security-advisories.html>
- [6] Microsoft Security Bulletin Search. [www-document] consulted on 17.01.2011. Available at: <http://www.microsoft.com/technet/security/Current.aspx>
- [7] VUPEN security. [www-document] consulted on 09.01.2011. Available at: <http://www.vupen.com>
- [8] Gentoo Linux Security Announcements (GLSAs). Gentoo Linux Security. [www-document] consulted on 14.01.2011. Available at: http://www.gentoo.org/security/en/#doc_chap2
- [9] Secunia. Secunia ApS. [www-document] consulted on 14.01.2011. Available at: <http://secunia.com/>
- [10] SecurityTracker. SecurityGlobal.net LLC. [www-document] consulted on 16.01.2011. Available at: <http://securitytracker.com/>
- [11] Debian Security Advisories [RSS-subscription] consulted on 17.01.2011. Available at: <http://www.debian.org/security/dsa>
- [12] Ubuntu Security Notice [RSS-subscription] consulted on 17.01.2011. Available at: <http://www.ubuntu.com/usn/rss.xml>
- [13] Gentoo Linux Security Advisories [RSS-subscription] consulted on 17.01.2011. Available at: <http://www.gentoo.org/rdf/en/glsa-index.rdf?num=20>

- [14] Securityfocus Vulnerabilities [RSS-subscription] consulted on 17.01.2011. Available at: <http://www.securityfocus.com/rss/vulnerabilities.xml>
- [15] The Open Source Vulnerability Database (OSVDB) [RSS subscription] consulted on 17.01.2011. Available at: <http://osvdb.org/feed/vulnerabilities/latest.rss>
- [16] Common Vulnerabilities and Exposures (CVE). The MITRE Corporation. [www-document] consulted on 19.01.2011. Available at: <http://cve.mitre.org/>
- [17] Dhillon, G. 2007. Principles of Information Systems Security: text and cases. John Wiley & Sons.
- [18] Microsoft TechNet: Definition of a Security Vulnerability [www-document] consulted on 20.01.2011. Available at: [http://technet.microsoft.com/es-es/library/cc751383\(en-us\).aspx](http://technet.microsoft.com/es-es/library/cc751383(en-us).aspx)
- [19] Notifications and Advisories. Red Hat Networks. Red Hat Inc. [www-document] consulted on 21.01.2011. Available at: <https://access.redhat.com/security/updates/advisory/>
- [20] Full-Disclosure, An unmoderated mailing list for the discussion of security issues. [www-document] consulted on 27.01.2011. Available at: <https://lists.grok.org.uk/mailman/listinfo/full-disclosure>
- [21] Python programming language. Python Software Foundation [www-document] consulted on 27.01.2011. Available at: <http://www.python.org/>
- [22] Python Application Domains. Python Software Foundation. [www-document] consulted on 27.01.2011. Available at: <http://www.python.org/about/apps/>
- [23] The Eclipse Foundation – Open source community website [www-document] consulted on 27.01.2011. Available at: <http://www.eclipse.org/>
- [24] Apache Subversion. Apache Software Foundation. [www-document] consulted on 29.01.2011. Available at: <http://subversion.apache.org/>
- [25] Pydev, Python IDE for Eclipse. Appcelerator, Inc. [www-document] consulted on 27.01.2011. Available at: <http://pydev.org>
- [26] Pydev Features. Appcelerator, Inc. [www-document] consulted on 02.02.2011. Available at: http://pydev.org/manual_adv_features.html
- [27] Subclipse. Tigris.org. Colabnet, Inc. [www-document] consulted on 04.02.2011. Available at: <http://subclipse.tigris.org/>
- [28] SQLite [www-document] consulted on 09.02.2011. Available at: <http://www.sqlite.org/>
- [29] SQLite. List of features. [www-document] consulted on 07.01.2011. Available at: <http://www.sqlite.org/features.html>
- [30] SQLite Manager. [www-document] consulted on 17.02.2011. Available at:

<http://code.google.com/p/sqlite-manager/>

- [31] Ubuntu. Canonical Ltd. [www-document] consulted on 07.02.2011. Available at: <http://www.ubuntu.com/>
- [32] GNU General Public License. Version 3, 29 June 2007. Free Software Foundation, Inc.
- [33] GNU Lesser General Public License. Version 3, 29 June 2007. Free Software Foundation, Inc.
- [34] UML, Unified Modeling Language. Object Management Group, Inc. [www-document] consulted on 17.01.2011. Available at: <http://www.uml.org/>
- [35] sqlite3 - DB-API 2.0 interface for SQLite databases [www-document] consulted on 27.02.2011. Available at: <http://docs.python.org/library/sqlite3.html>
- [36] sys – System-specific parameters and functions [www-document] consulted on 17.02.2011. Available at: <http://docs.python.org/library/sys.html>
- [37] base64 - RFC 3548: Base16, Base32, Base64 Data Encodings [www-document] consulted on 27.02.2011. Available at: <http://docs.python.org/library/base64.html>
- [38] feedparser: Universal Feed Parser 4.1. Pilgrim, M. 2006 [www-document] consulted on 19.02.2011. Available at : <http://www.feedparser.org/>
- [39] urllib – Open arbitrary resources by URL [www-document] consulted on 28.02.2011. Available at: <http://docs.python.org/library/urllib.html>
- [40] imaplib – IMAP4 protocol client [www-document] consulted on 03.03.2011. Available at: <http://docs.python.org/library/imaplib.html>
- [41] Fancy URL opener. Python [www-document] consulted on 05.03.2011. Available at: <http://docs.python.org/library/urllib.html#urllib.FancyURLopener>
- [42] Lambdas. Python [www-document] consulted on 05.01.2011. Available at: <http://docs.python.org/reference/expressions.html?highlight=lambda#lambda>
- [43] Beautiful Soup 3.0. Richardson, L. [www-document] consulted on 07.01.2011. Available at: <http://www.crummy.com/software/BeautifulSoup/documentation.html>
- [44] re – Regular expression operations [www-document] consulted on 27.01.2011. Available at: <http://docs.python.org/library/re.html>
- [45] urllib2 - Extensible library for opening URLs [www-document] consulted on 19.02.2011. Available at: <http://docs.python.org/library/urllib2.html>
- [46] hashlib – Secure hashes and message digests [www-document] consulted on 20.02.2011. Available at: <http://docs.python.org/library/hashlib.html>

- [47] cmd — Support for line-oriented command interpreters [www-document] consulted on 07.03.2011. Available at: <http://docs.python.org/library/cmd.html>
- [48] webbrowser — Convenient Web-browser controller. [www-document] consulted on 07.01.2011. Available at: <http://docs.python.org/library/webbrowser.html>
- [49] getpass — Portable password input. [www-document] consulted on 11.01.2011. Available at: <http://docs.python.org/library/getpass.html>
- [50] time — Time access and conversions [www-document] consulted on 07.03.2011. Available at: <http://docs.python.org/library/time.html>
- [51] Hetland, M.L.2005. Beginning Python: From novice to professional. Apress.

APPENDIX 1: DIAGRAMS AND REQUIREMENTS

Note: All diagrams are implemented following the UML notation. [34]

SyRAS objectives

OBJ¹-01	Collect security advisories
Description	The system must collect the security advisories coming from the chosen sources, link them to the proper product and store them in the SyRAS database.

OBJ-02	Manage and show the collected information
Description	The system must insert, delete or query the information stored in the SyRAS database: Users, Sources, Products and Entries (advisories) via Command Line Interface.

Information requirements

IRQ²-01	Information about SyRAS users
Description	The system must store in the SyRAS database the information about the system users. Specifically:
Specific data	<ul style="list-style-type: none"> • Unique ID of the user. • Date and time of creation of the user. • Name of the user. • Password of the user. • Email of the user. • Warning level of the user.
Comments	<p>For security, a hash version of the password will be the one stored in the database.</p> <p>The warning level can be 1 if the user must be alerted by email or 0 if not.</p>

¹ Objective

² Information Requirement

CRQ³-01	Unique user name and password
Description	The information stored in the database must satisfy the following restriction: There cannot be two users stored in the system with the same name and password.

IRQ-02	Information about SyRAS sources
Description	The system must store in the SyRAS database the information about the system sources. Specifically:
Specific data	<ul style="list-style-type: none"> • Unique ID of the source. • Date and time of creation of the source. • Name of the source. • Source Type of the source. • Type of the source. • URL subscription of the source. URL to access to the source. • Linked Product of the source. • Search Brand of the source. • URL Start of the source. • Pattern to Search of the source. • Date of Last Check of the source.
Comments	<p>The Source Type can be 1 if email, 2 if html or 3 if RSS.</p> <p>The Type can be 1 if general or 2 if specific.</p> <p>The Linked Product can be 0 if the source is not liked to any.</p> <p>The Search Brand is used only if the Type of the source is 2 (specific). It associates a source with a brand of products.</p> <p>The URL Start is used only if the Source Type of the source is 2 (html). It is used for searching new entries in the HTML code of the website using Regex.</p> <p>The Pattern to Search is used only if the Source Type of the source is 2 (html). It is used for searching new entries in the HTML code of the website using Regex.</p>

³ Constraint Requirement

CRQ-02	Unique user name, source type and subscription
Description	The information stored in the database must satisfy the following restriction: There cannot be two sources stored in the system with the same name, source type and subscription.

IRQ-03	Information about SyRAS products
Description	The system must store in the SyRAS database the information about the system products. Specifically:
Specific data	<ul style="list-style-type: none"> • Unique ID of the product. • Date and time of creation of the product. • Brand of the product. • Name of the product. • Version of the product. • General Search String of the product. • Specific Search String of the product. • Importance level of the product.
Comments	The Importance level goes from 0 (less important) to 2 (very important).

CRQ-03	Unique product brand, name and version
Description	The information stored in the database must satisfy the following restriction: There cannot be two products stored in the system with the same brand, name and version.

IRQ-04	Information about SyRAS entries (advisories)
Description	The system must store in the SyRAS database the information about the system entries. Specifically:
Specific data	<ul style="list-style-type: none"> • Unique ID of the entry. • Date and time of creation of the entry. • Permalink of the entry. • Linked Source of the entry. • Subject of the entry.

	<ul style="list-style-type: none"> • Body of the entry. • Product matched string of the entry. • Linked Product of the entry. • Importance Category automatically assigned to the entry. • Category assigned of the entry. • ID of the user that inserted or made the last modification in the entry. • MD5 Hash of the entry's permalink.
Comments	<p>The Importance Category automatically assigned goes from 1 (less important) to 5 (critical).</p> <p>The Category assigned can be: 0 if not revised, 1 if discarded, 2 if important or 3 if processed.</p>

CRQ-04	Unique entry MD5
Description	The information stored in the database must satisfy the following restriction: There cannot be two entries stored in the system with the same MD5 hash of the permalink (and with same permalink for instance).

Functional requirements

FRQ⁴-01	Login a user
Description	<p>The system must prompt a login dialog that allows a user to login using his username and password.</p> <p>It should also allow a user to logout.</p>
Comments	Once the user logs in, the prompt will change from <code>SyRAS: _></code> to <code>user@SyRAS: _></code>

FRQ-04	Collect new entries
Description	The system must allow the user to collect the new entries coming from the different sources: email, RSS or web. It is possible to collect

⁴ Functional Requirement

	<p>from a single source type or collect from all at once.</p> <p>A message with for the new entries will be shown in the Command Line Interface when inserted.</p>
Comments	Every time the system finds a new entry, it checks if it already exists in the database before inserting it.

FRQ-05	List data
Description	The system must allow the user to query data from the database, showing it in a table. It lists information about users, products, sources and entries.
Comments	For security reasons, the system won't show the passwords when querying the user table.

FRQ-06	Query a specific database entry
Description	The system must allow the user to search for a specific database entry, such as: user, product, source or advisory entry. The information will be shown in the CLI and it can also be saved directly in a txt file.
Comments	For security reasons, the system won't show or save in the file any user password.

FRQ-07	Browse a specific source or advisory entry
Description	The system must allow the user to search for a specific source or advisory entry and open its permalink in the web browser.
Comments	An instance of the default browser in a new window will be opened showing the information.

FRQ-08	Change priority
Description	<p>The system must allow the user to change to important the priority of one entry or product.</p> <p>A message informing about the modification will be shown in the Command Line Interface when performed.</p>

FRQ-9	Delete a specific database entry
Description	The system must allow the administrator to delete a specific database entry, such as: user, product, source or advisory entry. A message informing about the deletion will be shown in the Command Line Interface when performed.
Comments	Only the administrator user has the right to perform this action.

FRQ-10	Command history
Description	The system must store a history with the commands that have been used recently, allowing a quick access to them. The history can be accessed using the <i>arrow</i> buttons.

FRQ-11	Command auto complete
Description	The command line interface of the system must allow auto complete the commands included in SyRAS. The auto complete can be accessed using the <i>tab</i> button.

FRQ-12	Collecting loop
Description	The system must have an independent application running in background that checks for new entries every time period configured by the user. The independent application must run in an infinite loop and it will insert the new entries in the database when found.

Non-functional requirements

NFR⁵-01	PC requirements
Description	The SyRAS system must be executed in a PC with the operative system Ubuntu 9.04 or greater, with Python interpreter 2.5 or greater

⁵ Non-Functional Requirement

	and with database SQLite 3.6 or greater. The system must also have internet access.
Comments	The internet access is needed to collect new entries.

NFR-02	Application name
Description	<p>The complete system will be called SyRAS (Security Alerts Collecting System).</p> <p>The application including the Command Line Interface will be called <i>syras.py</i> and the one including the collecting loop will be called <i>syrasloop.py</i>.</p> <p>The SQLite database will be called <i>syras.sqlite</i>.</p> <p>All SyRAS components must be placed in the same folder under the Operative System.</p>

NFR-03	Portability
Description	The complete SyRAS system must be easy to port to other devices as soon as they match the requirements included in NFR-01 .

NFR-04	Future improvements
Description	The complete system SyRAS must allow improvements, such as new web based interface, to be introduced easily.

Static and dynamic model of the system

ACT⁶-01	SyRAS guest
Description	This actor represents a user that uses the application SyRAS installed in the PC.

ACT-02	SyRAS user
Description	This actor represents a user that uses the application SyRAS

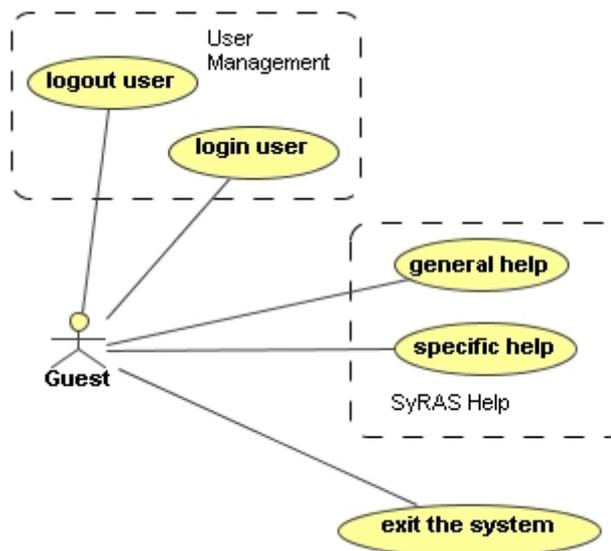
⁶ Actor

	installed in the PC. It needs to log in and can perform actions according to its privileges.
--	--

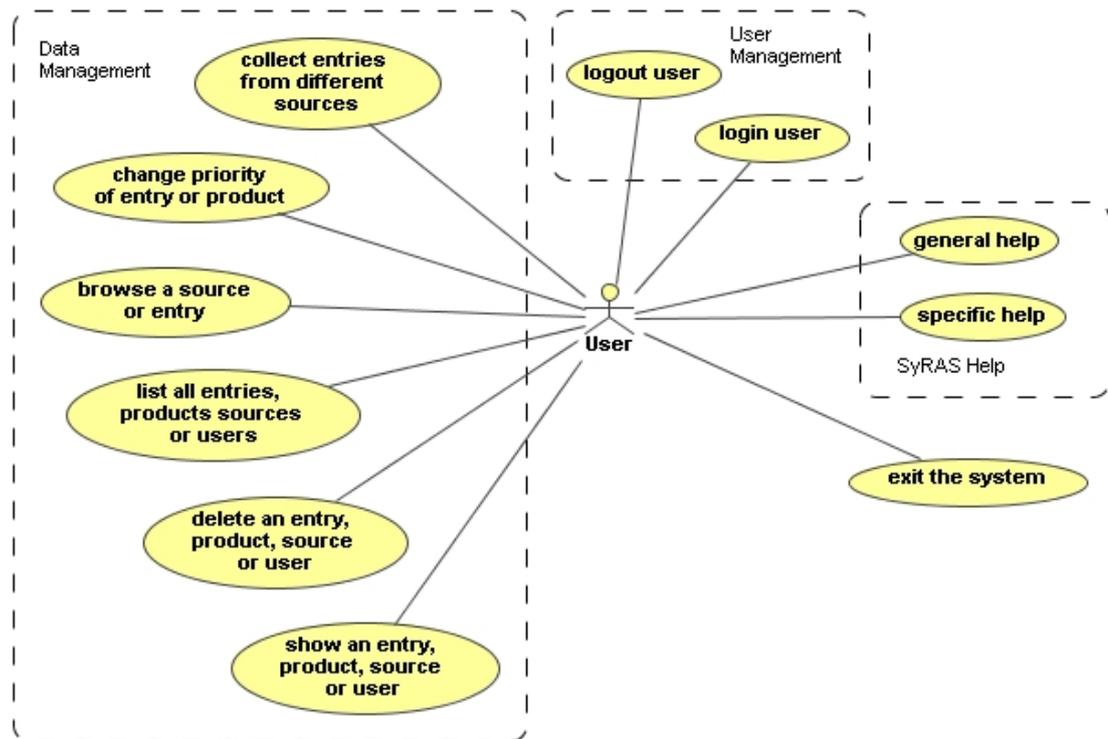
ACT-03	SyRAS administrator
Description	This actor represents a user that uses the application SyRAS installed in the PC. It needs to log in and can perform all actions.

Use case diagrams

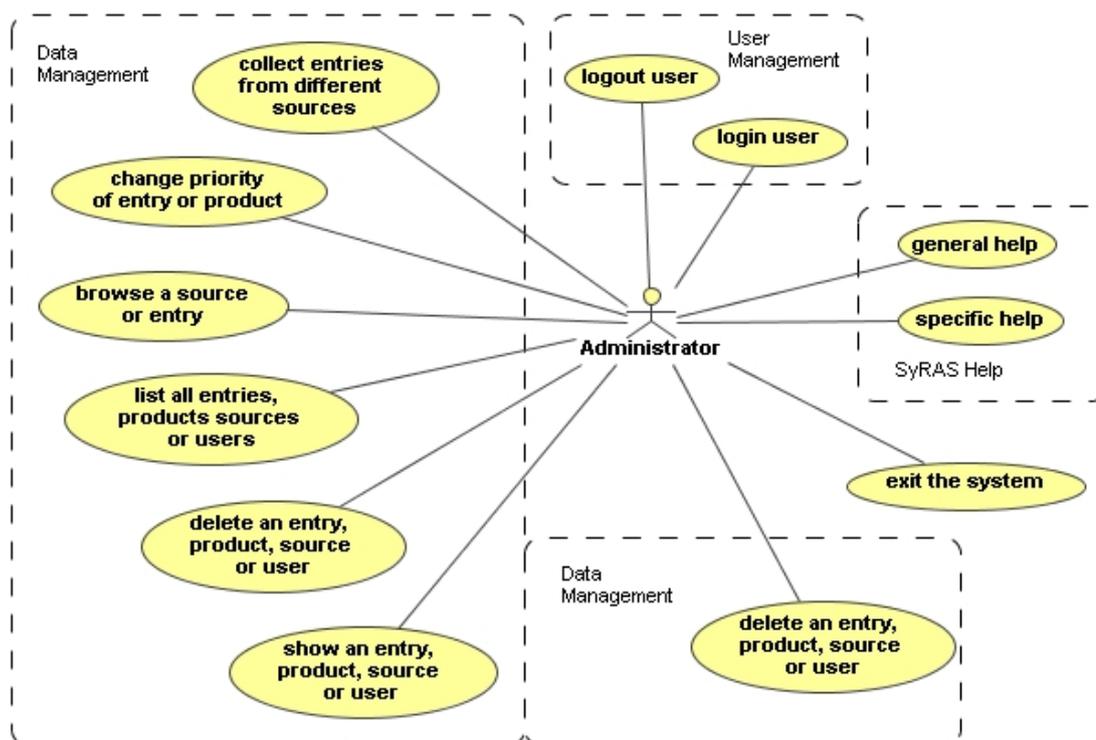
Use cases diagram for SyRAS “guest”



Use cases diagram for SyRAS "user"



Use cases diagram for SyRAS "administrator"



Use case descriptions

UC ⁷ -01	Login a user	
Description	The system must behave as is described in the following use case when a user wants to log in SyRAS.	
Precondition	None.	
Normal sequence	Step	Action
	1	The user asks the system to log in introducing his/her username.
	2	The system asks for the password.
	3	The user types the password and presses Enter.
	4	If the user and password match, the user logs in the system. The prompt changes to show that the user is logged in.

⁷ Use Case

		From now on the user can perform actions that require to be logged in. The system goes back to the main Command Line Interface.
Postcondition	The user is logged in the SyRAS system.	
Exceptions	Step	Action
	1	If there is a user currently logged in, the system shows a message asking to log out before login in with a different user. Then this use case doesn't take effect.
	1	If the username inserted is empty, the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.
	3	If the user and password combination do not match, the system shows a message reporting the error. Then this use case doesn't take effect.

UC-02	Browse a source or entry	
Description	The system must behave as is described in the following use case when the user asks the system to browse a source or entry introducing its ID number.	
Precondition	The user must be logged in.	
Normal sequence	Step	Action
	1	The user asks the system to browse a source or entry introducing the keyword "source" or "entry" and the ID number to browse.
	2	If the source or entry with the given ID exists, the system shows a message saying that its permalink will be opened in the web browser. The default web browser will show the web page with the permalink of the source or entry given. The system goes back to the main Command Line Interface.

Postcondition	None.	
Exceptions	Step	Action
	1	If there is no user currently logged in, the system shows a message asking to log in before performing this action. Then this use case doesn't take effect.
	1	If the keywords inserted are not right the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the given ID doesn't exist, the system shows a message reporting the error. Then this use case doesn't take effect.

UC-03	Collect entries from different sources	
Description	The system must behave as is described in the following use case when the user asks the system to collect new entries from one source (email, html, rss) or from all of them.	
Precondition	The user must be logged in.	
Normal sequence	Step	Action
	1	The user asks the system to collect new entries from one source type (email, html, rss) or from all of them introducing the keyword "email", "html", "rss" or "all".
	2	<p>The system retrieves information from the sources and inserts new entries in the database, and also tries to associate them to any product stored in the system.</p> <p>The system shows a message for every new entry inserted, and also if it was associated to any product.</p> <p>The system shows a message for every source type collected and source updated.</p> <p>The field "last_checked" of the sources collected is updated in the database.</p> <p>The system shows a message for every new entry inserted,</p>

		and also if it was associated to any product. The system goes back to the main Command Line Interface.
Postcondition	None.	
Exceptions	Step	Action
	1	If there is no user currently logged in, the system shows a message asking to log in before performing this action. Then this use case doesn't take effect.
	1	If the keywords inserted are not right the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.

UC-04	Delete an entry, product, source or user	
Description	The system must behave as is described in the following use case when the user asks the system to delete an entry, product, source or user introducing its ID number.	
Precondition	The user logged in must be "admin".	
Normal sequence	Step	Action
	1	The user asks the system to browse a source or entry introducing the keyword "entry", "product", "source" or "user" and the ID number to delete.
	2	If the entry, product, source or user with the given ID exists, the system shows a message saying that it will be deleted. The system deletes it from the database. The system goes back to the main Command Line Interface.
Postcondition	The deleted ID cannot be used any longer.	
Exceptions	Step	Action
	1	If the user logged is not "admin", the system shows a message asking the admin to log in before performing this action. Then this use case doesn't take effect.
	1	If the keywords inserted are not right the system shows a

		message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the given ID doesn't exist, the system shows a message reporting the error. Then this use case doesn't take effect.

UC-05	List all entries, products, sources or users	
Description	The system must behave as is described in the following use case when the user asks the system to list all entries, products, sources or users	
Precondition	The user must be logged in.	
Normal sequence	Step	Action
	1	The user asks the system to list all entries, products, sources or users introducing the keyword "entries", "products", "sources" or "users".
	2	The system shows a message saying that it will list the requested information and shows it. The system doesn't show user passwords. The system goes back to the main Command Line Interface.
Postcondition	None.	
Exceptions	Step	Action
	1	If there is no user currently logged in, the system shows a message asking to log in before performing this action. Then this use case doesn't take effect.
	1	If the keywords inserted are not right the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.

UC-06	Log out current logged user	
Description	The system must behave as is described in the following use case when the user asks to log out from SyRAS.	
Precondition	The user must be logged in.	
Normal sequence	Step	Action
	1	The user asks the system to log out.
	2	The prompt changes to show that the user is not logged in. The system goes back to the main Command Line Interface.
Postcondition	From now on the user cannot perform actions that require to be logged in.	
Exceptions	Step	Action
	1	If there is no user currently logged in, the system shows a message informing about it. Then this use case doesn't take effect.

UC-07	Change the priority of an entry or product	
Description	The system must behave as is described in the following use case when the user asks the system to change to <i>important</i> the priority of one entry or product.	
Precondition	The user must be logged in.	
Normal sequence	Step	Action
	1	The user asks the system to change to <i>important</i> the priority of an entry or product introducing the keyword "entry" or "product" and the ID number to change priority.
	2	If the entry or product with the given ID exists, the system shows a message saying that it will change its priority to <i>important</i> . The system changes the priority of the given entry or product to <i>important</i> in the database. The system goes back to the main Command Line Interface.

Postcondition	None.	
Exceptions	Step	Action
	1	If there is no user currently logged in, the system shows a message asking to log in before performing this action. Then this use case doesn't take effect.
	1	If the keywords inserted are not right the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the given ID doesn't exist, the system shows a message reporting the error. Then this use case doesn't take effect.

UC-08	Save an entry, product, source or user	
Description	The system must behave as is described in the following use case when the user asks the system to save an entry, product, source or user introducing its ID number.	
Precondition	The user must be logged in.	
Normal sequence	Step	Action
	1	The user asks the system to save all the information of an entry, product, source or user introducing the keyword "entry", "product", "source" or "user" and the ID number to save.
	2	If the entry, product, source or user with the given ID exists, the system shows a message saying that it will save its content in a text file. The system will store the information of the given ID in a file with the name: keyword+ID.txt. (ex: user3.txt.). The system goes back to the main Command Line Interface. The system will not show user passwords for security reasons.
Postcondition	None.	
Exceptions	Step	Action

	1	If there is no user currently logged in, the system shows a message asking to log in before performing this action. Then this use case doesn't take effect.
	1	If the keywords inserted are not right the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the given ID doesn't exist, the system shows a message reporting the error. Then this use case doesn't take effect.

UC-09	Show an entry, product, source or user	
Description	The system must behave as is described in the following use case when the user asks the system to show an entry, product, source or user introducing its ID number.	
Precondition	The user must be logged in.	
Normal sequence	Step	Action
	1	The user asks the system to show all the information of an entry, product, source or user introducing the keyword "entry", "product", "source" or "user" and the ID number to save.
	2	If the entry, product, source or user with the given ID exists, the system shows a message saying that it will show its content in a text file. The system will show on the screen all the information of the given ID. The system goes back to the main Command Line Interface. The system will not show user passwords for security reasons.
Postcondition	None.	
Exceptions	Step	Action
	1	If there is no user currently logged in, the system shows a message asking to log in before performing this action. Then

		this use case doesn't take effect.
	1	If the keywords inserted are not right the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the amount of parameters introduced is not the required, the system shows a message reporting the error. Then this use case doesn't take effect.
	1	If the given ID doesn't exist, the system shows a message reporting the error. Then this use case doesn't take effect.

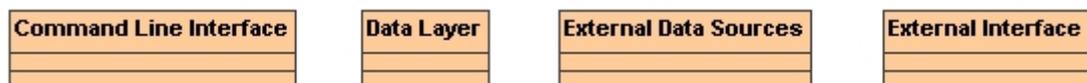
UC-10		Show general help	
Description	The system must behave as is described in the following use case when the user asks the system to show the general help.		
Precondition	None		
Normal sequence	Step	Action	
	1	The user asks the system to show the general help.	
	2	The system shows a message with the general help of the SyRAS software, including useful commands. The default web browser will show the web page with the permalink of the source or entry given. The system goes back to the main Command Line Interface.	
Postcondition	None.		
Exceptions	Step	Action	
		None.	

UC-11		Show specific help	
Description	The system must behave as is described in the following use case when the user asks the system to show the specific help of a useful command.		
Precondition	None		
Normal sequence	Step	Action	

	1	The user asks the system to show the specific help of a useful command introducing help + its name.
	2	The system shows a message with the specific help information for the command if it's included in SyRAS. The system goes back to the main Command Line Interface.
Postcondition	None.	
Exceptions	Step	Action
	1	If the given command doesn't exist, the system shows a message reporting the error. Then this use case doesn't take effect.

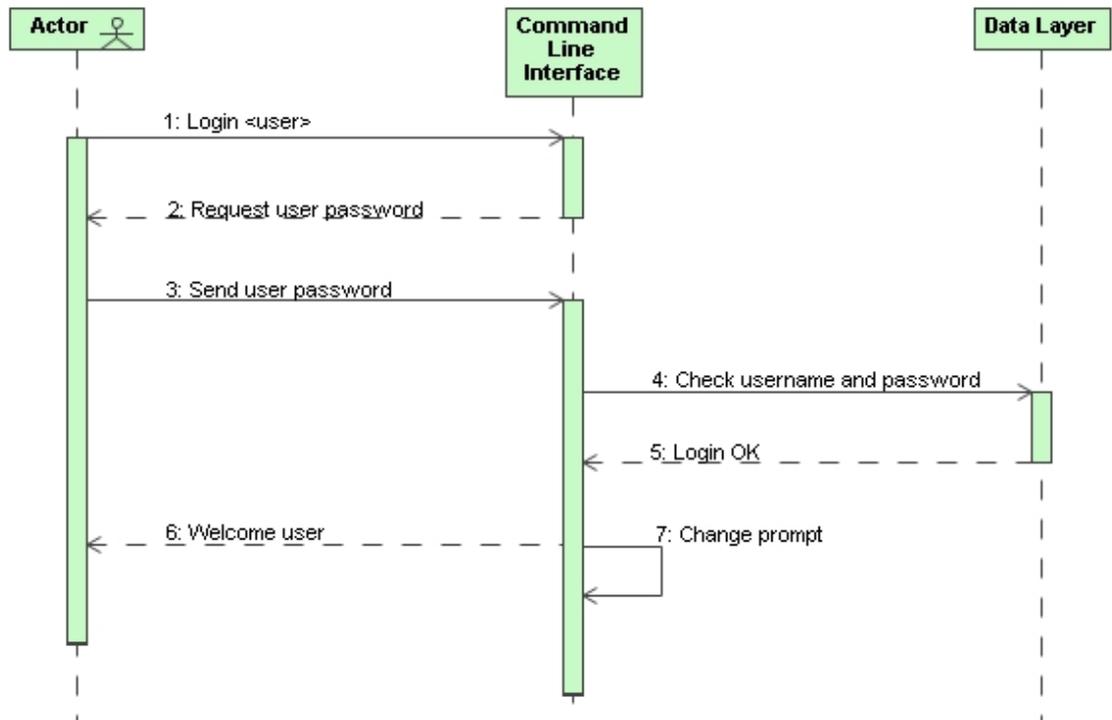
UC-12	Exit the system	
Description	The system must behave as is described in the following use case when the user asks the system to exit.	
Precondition	None	
Normal sequence	Step	Action
	1	The user asks the system to exit.
	2	The system exits the SyRAS application.
Postcondition	No actions can be taken after this command.	
Exceptions	Step	Action
		None.

Static diagram

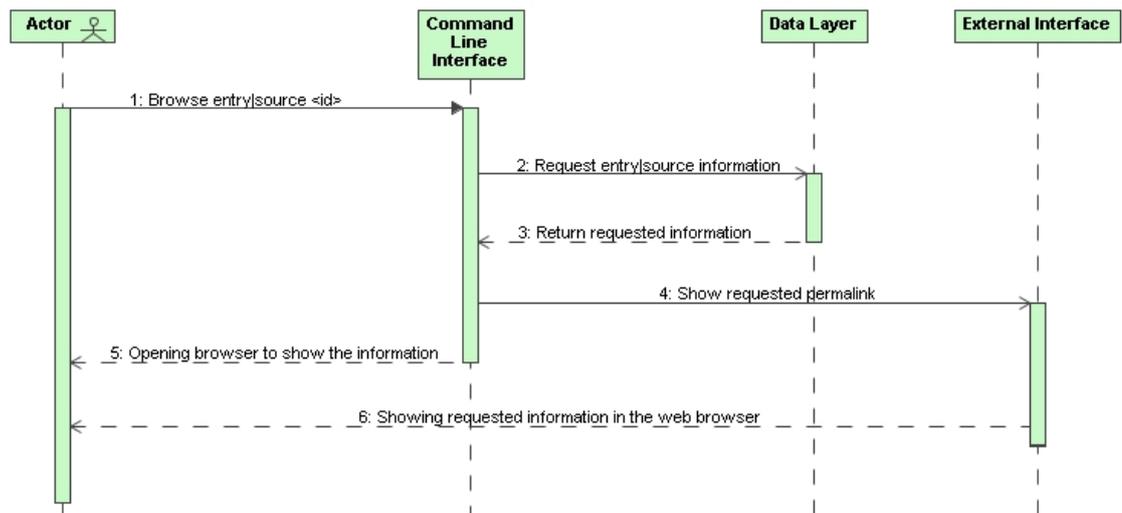


Dynamic diagrams: Sequence diagrams

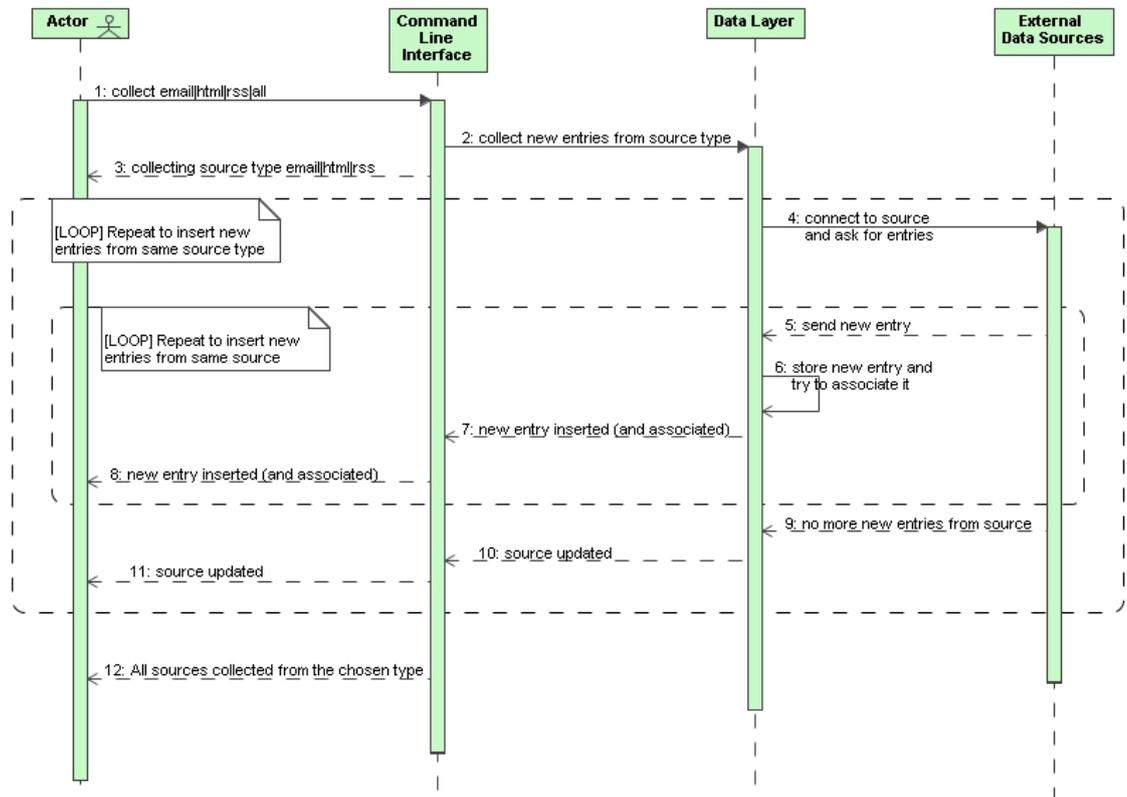
UC-01: Login a user



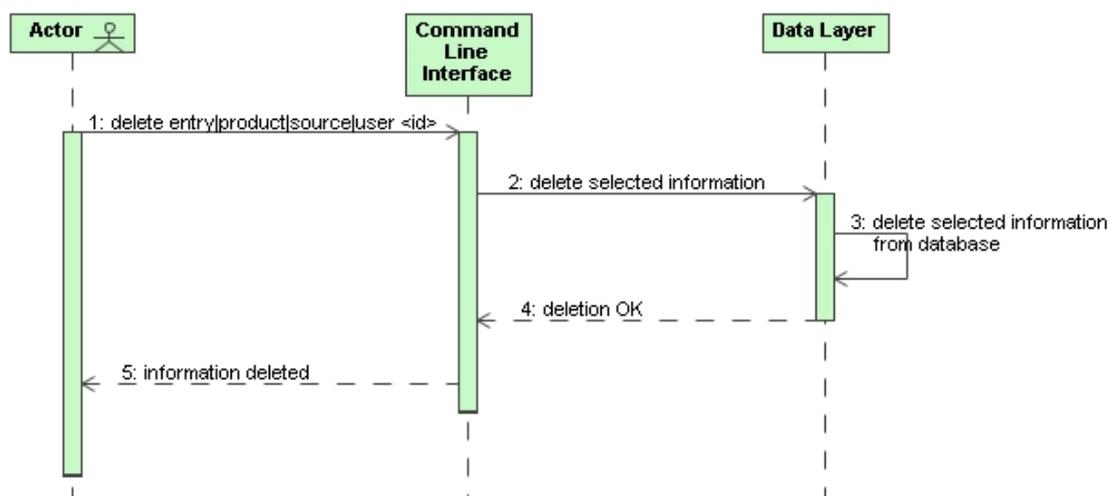
UC-02: Browse a source or entry



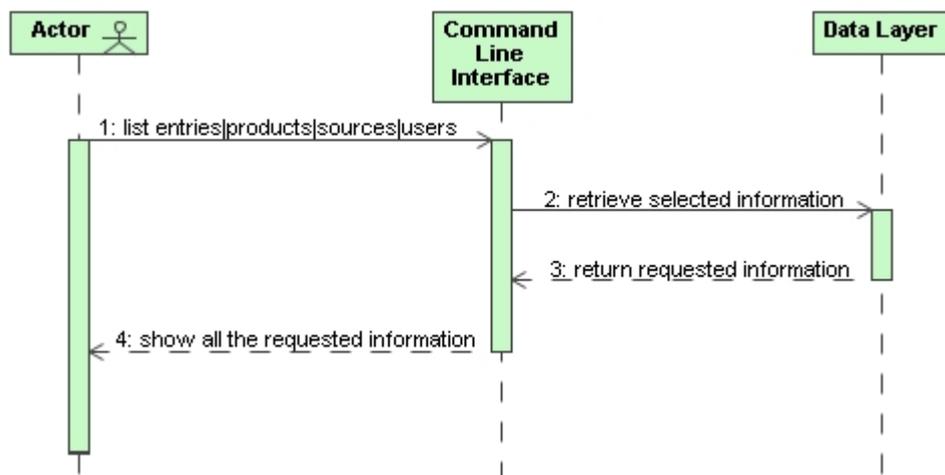
UC-03: Collect entries from different sources



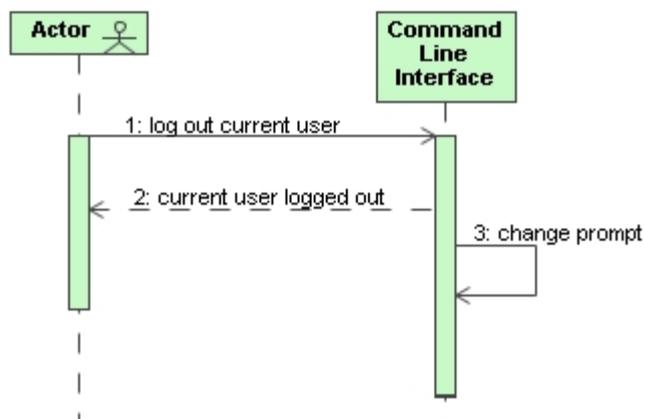
UC-04: Delete an entry, product, source or user



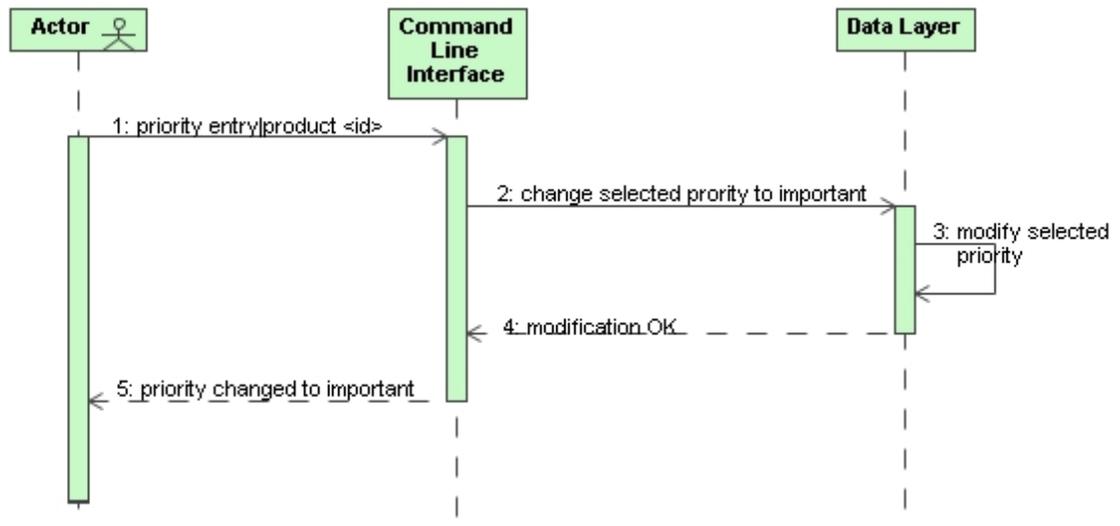
UC-05: List all entries, products, sources or users



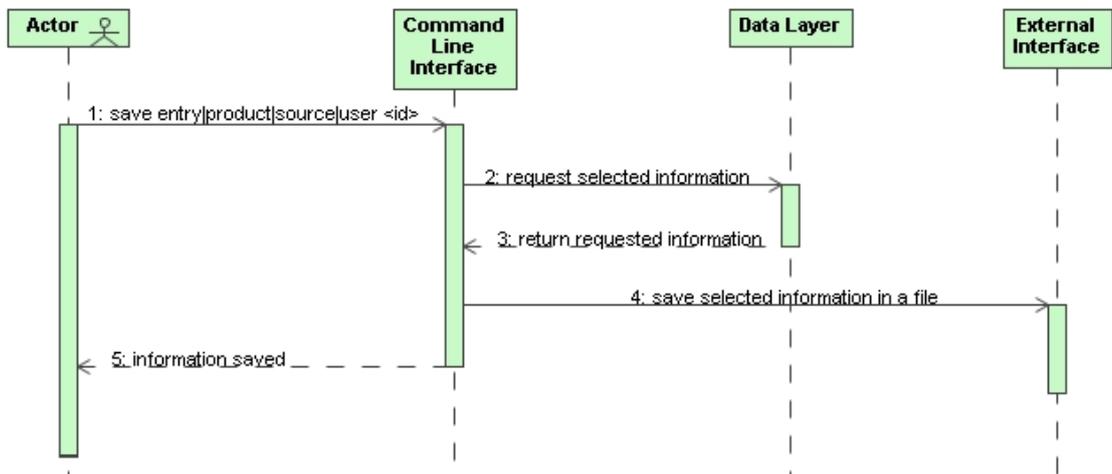
UC-06: Log out current logged user



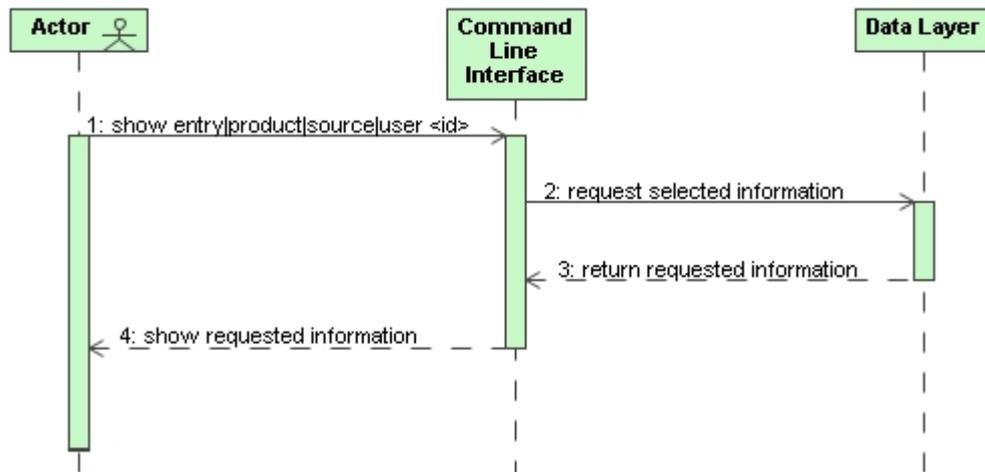
UC-07: Change the priority of an entry or product



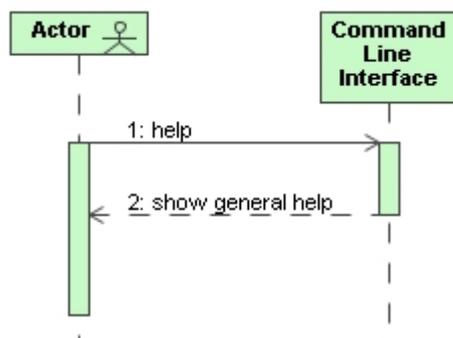
UC-08: Save an entry, product, source or user



UC-09: Show an entry, product, source or user



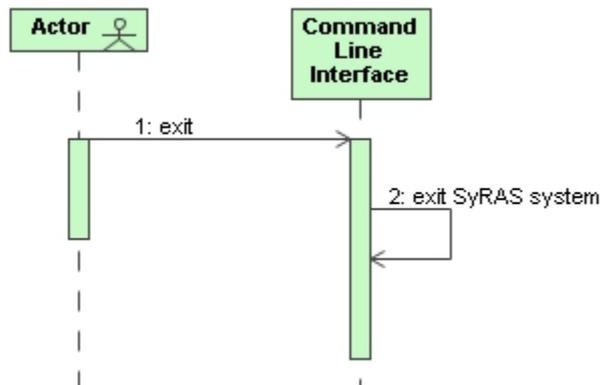
UC-10: Show general help



UC-11: Show specific help



UC-12: Exit the system



APPENDIX 2: CODE

This appendix shows parts of the code implemented for the SyRAS application. Note that parts of the code have been omitted, keeping only the important ones and the descriptions of the methods.

mod_db.py

```
import sqlite3

class DB:
    """
    Handling class for sqlite statements
    """
    def __init__(self,database='./syras.sqlite'):
        """
        Starts the database
        Parameter must be a string ('/path/syras.sqlite')
        """
        self.db = database
        self.con = None
        self.curs = None

    def connect(self):
        """
        Logging to the working database
        """
        connection = sqlite3.connect(self.db)
        self.con = connection
        self.curs = connection.cursor()

    def disconnect(self):
```

```
"""  
Closing connection and cursor, logging out from the working database  
"""  
  
self.curs.close()  
self.con.close()  
  
def select(self, sql):  
    """  
    Performs a custom query  
    Returns a list of rows with the results  
    """  
  
    self.curs.execute(sql)  
    return self.curs.fetchall()  
  
def insert(self, sql):  
    """  
    Performs insert statements  
    Returns 1 if ok or 0 if failed  
    """  
  
    try:  
        self.curs.execute(sql)  
  
    except:  
        print "Insert ERROR: "  
  
def delete(self, sql):  
    """  
    Performs delete statements  
    Returns 1 if ok or 0 if failed  
    """  
  
def update(self, sql):  
    """  
    Performs update statements  
    Returns 1 if ok or 0 if failed  
    """
```

```

def commit(self):
    """
    Commits last changes to the database
    """
    self.con.commit()

def rollback(self):
    """
    Discards changes since last commit
    """
    self.con.rollback()

```

mod_users.py

```

import sys # for changing the standard output
import base64 #for encoding the password

"This module contains the class for handling Users"

class Users:
    "Generic class for handling users"
    def __init__(self, dbinstance):
        "Set the user values to None"
        self.database = dbinstance
        self.id = None
        self.date = None
        self.name = None
        self.password = None    #User's password
        self.email = None      #User's email to send warnings
        self.warnings = None    #Warning options 1=email, 2=sms
    def get_one(self, id):
        "Get the user with known id"
        try:

```

```

sql = "select * from users where id = %d" % id

querylist = self.database.select(sql)

# querylist is a list of tuples

# We use the first tuple if querylist it's not empty

if querylist:

    query = querylist[0]

    self.id = id

    self.date = query[1]

    self.name = query[2]

    self.password = query[3]

    self.email = query[4]

    self.warnings = query[5]

else:

    print "ERROR: Trying to get_one empty user id %d" %id

    self.id = 0

except:

    print "ERROR: trying to get_one empty user id"

def get_all(self):

    """

    Get all users from the database

    Returns object "all_users" to iterate over it

    """

    try:

        sql = "select id, date, name, email, warnings from users"

        all_users = self.database.select(sql)

        return all_users

def exists(self, user, password):

    "Get the user with known user and password"

    try:

        encoded = base64.b64encode(password)

        sql = """select id from users where name = '%s' and password = '%s' """ % (user, encoded)

```

```

querylist = self.database.select(sql)

if querylist: return querylist[0][0]

def insert(self):
    """
    Insert 'self' user checking if it exists already
    To check if exists it compares the name
    """
    try:
        sql = "select * from users where name = '%s'" % (self.name)
        querydic = self.database.select(sql)

        if querydic:
            print "New user NOT INSERTED"
        else:
            sql = """INSERT INTO Users (date, name, password, email, warnings)
            VALUES (CURRENT_TIMESTAMP, '%s', '%s', '%s', %d)
            """ % (self.name, self.password, self.email, self.warnings)
            ok = self.database.insert(sql)

            if ok:
                print "New user '%s' INSERTED" % self.name
                self.database.commit()

def delete(self):
    "Delete self user"

    try:
        sql = "delete from Users where id = %d" % self.id
        ok = self.database.delete(sql)

        if ok:
            print "User %d '%s' DELETED" % (self.id, self.name)
            self.database.commit()

def modify(self):
    """
    Modify the user with self values
    """

```

```

try:
    sql = """UPDATE Users SET name = '%s', password = '%s',
    email = '%s', warnings = %d
    WHERE id = %d
    """ %(self.name, self.password,
        self.email, self.warnings,
        self.id)
    ok = self.database.update(sql)
    if ok:
        print "User %d '%s' MODIFIED" % (self.id, self.name)
        self.database.commit()

```

```
def show(self):
```

```
    """
```

```
    Show the self user on screen
```

```
    """
```

```
    try:
```

```
        print "USER id: ",self.id
        print "Date of creation: ",self.date
        print "Name: ",self.name
        print "Password: 'encrypted'"
        print "Email: ",self.email
        print "Warning level (1=email, 2=sms): ",self.warnings

```

```
def save(self):
```

```
    """
```

```
    Save the self user on a file user<id>.txt
```

```
    """
```

```
    try:
```

```
        filename = 'user' + str(self.id) + '.txt'
        file = open(filename, 'w')
        sys.stdout = file #change standard output to file
        self.show()
        file.close()
        sys.stdout=sys.__stdout__ #change back standard output

```

```
print "User %d saved in file %s" %(self.id, filename)
```

mod_products.py

```
import sys # for changing the standard output
```

```
"This module contains the class for handling Products"
```

```
class Products:
```

```
    "Generic class for handling products"
```

```
    def __init__(self, dbinstance):
```

```
        "Set the product values to None"
```

```
        self.database = dbinstance
```

```
        self.id = None
```

```
        self.date = None
```

```
        self.brand = None        #Product's brand   E.g: brand = Microsoft
```

```
        self.name = None        #Product's name     name = Office
```

```
        self.version = None     #Product's version   version = 2007
```

```
        self.general_search = None #General search string (used if source.type=1 -general-)
```

```
        self.specific_search = None #Specific search string (used if source.type=2 -specific-)
```

```
        self.importance = None    #0=Unknown, 1= Not important, 2=Important, 3=Client
```

```
    def get_one(self, id):
```

```
        "Get the product with known id"
```

```
    def get_all(self):
```

```
        """
```

```
        Get all products from the database
```

```
        Returns object "all_products" to iterate over it
```

```
        """
```

```
    def get_branded(self, selected_brand):
```

```
        """
```

```
        Get the id of all products with same brand from the database
```

```
        Returns object "selected_brand_products" to iterate over it
```

```
        """
```

```
try:
    sql = "select * from products where brand = '%s'" % selected_brand
    selected_brand_products = self.database.select(sql)
    return selected_brand_products
except:
    print "ERROR: trying to get_branded"
```

```
def insert(self):
    """
    Insert 'self' product checking if it exists already
    To check if exists it compares brand, name and version
    """
```

```
def delete(self):
    "Delete self product"
```

```
def modify(self):
    """
    Modify the product with self values
    """
```

```
def show(self):
    """
    Show the self product on screen
    """
```

```
def save(self):
    """
    Save the self product on a file product<id>.txt
    """
```

mod_sources.py

```
import sys # for changing the standard output
```

```
"This module contains the class for handling Sources"
```

```
class Sources:
```

```
    "Generic class for handling sources"
```

```
    def __init__(self, dbinstance):
```

```
        "Set the source values to None"
```

```
        self.database = dbinstance
```

```
        self.id = None
```

```
        self.date = None
```

```
        self.name = None
```

```
        self.source_type = None #1=email, 2=html, 3=rss
```

```
        self.type = None #1=general, 2=specific
```

```
        self.subscription = None #URL to connect to the source
```

```
        self.linked_product = None #If type=2 and is directly associated. 0 If could be associated to several
```

```
            self.search_brand = None #If type=2. Specifies the brand of products to search for association. self.url_start =
            None #If source_type=2 Url to append
```

```
        self.pattern = None #If source_type=2 RE for searching new entries
```

```
        self.last_checked = None #Update with current timestamp every time the source is checked
```

```
    def get_one(self, id):
```

```
        "Get the source with known id"
```

```
    def get_all_email(self):
```

```
        """
```

```
        Get all email sources from the database
```

```
        Returns object "all_email_sources" to iterate over it
```

```
        """
```

```
        try:
```

```
            sql = "select * from sources where source_type = 1"
```

```
            all_sources = self.database.select(sql)
```

```
            return all_sources
```

```
        except:
```

```
    print "ERROR: trying to get_all_email sources"

def get_all_html(self):
    """
    Get all html sources from the database
    Returns object "all_html_sources" to iterate over it
    """

def get_all_rss(self):
    """
    Get all rss sources from the database
    Returns object "all_rss_sources" to iterate over it
    """

def get_all(self):
    """
    Get all sources from the database
    Returns object "all_sources" to iterate over it
    """

def insert(self):
    """
    Insert 'self' source checking if it exists already
    To check if it exists it compares source_type and subscription
    """

def delete(self):
    "Delete self source"

def modify(self):
    """
    Modify the source with self values
    """

def set_time(self):
    """
```

Update self.last_checked to CURRENT_TIMESTAMP

```

"""
try:
    sql = r"""UPDATE sources SET last_checked = (SELECT datetime(strftime('%s','now'), 'unixepoch', 'localtime'))
    """ + """ WHERE id = %d
    """ % self.id
    #sql = """UPDATE sources SET last_checked = CURRENT_TIMESTAMP
    #WHERE id = %d
    #""" % self.id
    ok = self.database.update(sql)
    if ok:
        self.database.commit()
        print "Source '%s' updated" % self.name
        #print "Source '%s' updated at: " % self.name , self.last_checked
    except:
        print "ERROR: trying to set_time id: ", self.id

def show(self):
    """
    Show the self source on screen
    """

def save(self):
    """
    Save the self source on a file source<id>.txt
  
```

mod_email.py

```

#Imports
from mod_sources import Sources
from mod_entries import Entries

import feedparser    # For parsing the gmail atom feeds
import urllib        # For downloading the webpages
import imaplib       # For connecting to gmail and mark emails as read
  
```

"Module that contains class for email management"

```
GMAIL_ATOM_URL = "https://mail.google.com/gmail/feed/atom"
```

```
class Email_Sources(Sources):
```

```
    """
```

```
    Class for handling Email sources
```

```
    Inherits from class Sources
```

```
    Adds new function check_new_entries and attribute feed
```

```
    Uses gmail as rss to get entries from inbox and then mark as read
```

```
    """
```

```
    gmail_user = 'syrasfeed'
```

```
    gmail_pass = 'securitypw'
```

```
    def __init__(self, dbinstance):
```

```
        "Rss_Source object, like Source object but with new attributes: feed = parsed rss_source"
```

```
        Sources.__init__(self, dbinstance)
```

```
        self.feed = None #Parsed rss_source
```

```
    #SOURCE: http://docs.python.org/library/urllib.html#urllib.FancyURLopener
```

```
    #Provides the gmail account user and pw
```

```
    urllib.FancyURLopener.prompt_user_passwd = lambda self, host, realm: ('syrasfeed', 'securitypw')
```

```
    def parse_feed(self):
```

```
        "Parse the email source in self.feed"
```

```
        #Uses the prompt_user_passwd for providing user and pw for the email account
```

```
        opener = urllib.FancyURLopener()
```

```
        f = opener.open(self.subscription)
```

```
        self.feed = feedparser.parse(f.read())
```

```
    def mark_as_read(self):
```

```
        "Mark as read all emails from the gmail inbox"
```

```

#SOURCE: http://noandwhere.com/archive/archive-your-gmail-inbox-from-python

M = imaplib.IMAP4_SSL('imap.gmail.com')

M.login('syrafeed','securitypw')

M.select()

typ, data = M.search(None, 'ALL')

for num in data[0].split():

    #Remove the message from the inbox, but don't actually delete them

    M.store(num, '+FLAGS', '\\Seen')

M.expunge()

def check_new_entries(self):

    "Check one rss source to get the new entries"

    try:

        #Create a new entry object and fill in the fields

        for possible_entry in self.feed[ "items" ]: #For each possible new entry

            new_entry = Entries(self.database)

            new_entry.permalink = possible_entry[ "link" ]

            new_entry.md5_get()

            if not new_entry.md5_exist():

                new_entry.linked_source = self.id

                new_entry.subject = possible_entry[ "title" ]

                new_entry.body = possible_entry[ "summary" ]

                new_entry.insert()

                new_entry.link_to_product()

            self.mark_as_read() #Empty the inbox

            self.set_time()

    except:

        print "ERROR: trying to check_new_entries in Rss_sources"

```

mod_html.py

```

#Imports

from mod_sources import Sources

from mod_entries import Entries

#SOURCE: http://www.crummy.com/software/BeautifulSoup/documentation.html

from BeautifulSoup import BeautifulSoup # For parsing the HTML

import re # For regular expressions

import urllib2 # For downloading the webpages

"Module that contains class for html management"

class Html_Sources(Sources):
    """
    Class for handling HTML sources

    Inherits from class Sources

    Adds new function check_new_entries and attributes page and soup
    """
    def __init__(self, dbinstance):
        "Html_Source object, like Source object but with new atributes: page and soup"
        Sources.__init__(self, dbinstance)

        self.page = None

        self.soup = None

    def parse_page(self):
        "Parse the html source in self.page and self.soup"

        self.page = urllib2.urlopen(self.subscription) #downloaded html_source

        self.soup = BeautifulSoup(self.page) #parsed the downloaded html_source

    def check_new_entries(self):
        "Check one html source to get all the new entries"

        try:

            # Search for the <a> tags that match the RE pattern

            all_entries = self.soup.findAll('a', href=re.compile(self.pattern))

```

```

for possible_entry in all_entries:
    new_entry = Entries(self.database)
    new_entry.permalink = self.url_start + possible_entry["href"] # To compose the permalink
    new_entry.md5_get()
    if new_entry.md5_exist():pass
    else:
        new_entry.linked_source = self.id
        new_entry.subject = possible_entry.string
        new_entry.body = "empty body"
        new_entry.insert()
        new_entry.link_to_product()
    self.set_time()
except:
    print "ERROR: trying to check_new_entries in HTML_sources"

```

mod_rss.py

```

#Imports
from mod_sources import Sources
from mod_entries import Entries
from datetime import datetime

#SOURCE:http://www.feedparser.org/
import feedparser # For parsing the RSSfeed

"Module that contains class for rss management"

class Rss_Sources(Sources):
    """
    Class for handling RSS sources
    Inherits from class Sources
    Adds new function check_new_entries and attribute feed
    """
    def __init__(self, dbinstance):
        "Rss_Source object, like Source object but with new attributes: feed = parsed rss_source"

```

```

Sources.__init__(self, dbinstance)

self.feed = None #parsed rss_source

def parse_feed(self):

    "Parse the RSS source in self.feed"

    self.feed = feedparser.parse(self.subscription)

def check_new_entries(self):

    "Check one RSS source to get the new entries"

    try:

        #Create a new entry object and fill in the fields

        temp = datetime.strptime(self.last_checked, "%Y-%m-%d %H:%M:%S")

        last_checked_tuple = temp.timetuple()

        for possible_entry in self.feed[ "items" ]: #For each possible new entry

            try:

                if last_checked_tuple > possible_entry["updated_parsed"]: break

            except:

                new_entry = Entries(self.database)

                new_entry.permalink = possible_entry[ "link" ]

                new_entry.md5_get()

                if not new_entry.md5_exist():

                    new_entry.linked_source = self.id

                    new_entry.subject = possible_entry[ "title" ]

                    new_entry.body = "empty body"

                    new_entry.insert()

                    new_entry.link_to_product()

                self.set_time()

            except:

                print "ERROR: trying to check_new_entries in Rss_sources"

```

mod_entries.py

```

# Imports

import hashlib          # Module for md5

import sys             # for changing the standard output

from mod_sources import Sources    # My sources module

from mod_products import Products  # My products module

"This module contains the class for handling Entries"

class Entries:

    "Generic class for handling entries"

    def __init__(self, dbinstance):

        "Set the entry values to None"

        self.database = dbinstance

        self.id = None

        self.date = None

        self.permalink = None    #Permalink to the entry

        self.linked_source = None #Source of the new entry

        self.subject = None     #Title of the new entry

        self.body = None        #To get when assigned_cat=3 (processed)

        self.matched_string = None #String matched if assigned to any product (linked_product !=0)

        self.linked_product = None #Id of the liked_product (0 if Not matched)

        self.automatic_cat = None #Importance (from 1 to 5 -max-) assigned automatically

        self.assigned_cat = None #Assigned by user (0=Not revised, 1=Discarded, 2=Important, 3=Processed)

        self.by_user = None     #Id of the user that assigned category (by_user=0 if automatic)

        self.md5 = None         #md5 of the permalink

    def get_one(self, id):

        "Get the entry with known id"

    def get_all(self):

        """

        Get the last 20 entries from the database

        Returns object "all_sources" to iterate over it

        """

```

```

def md5_exist(self):
    "Check if the entry (self.md5) is already in the database"
    try:
        sql = "select * from entries where md5 = '%s'" % (self.md5)
        query = self.database.select(sql)
        if query: return 1 #The entry with this md5 exists already
        else: return 0 #The entry with this md5 does NOT exist yet
    except:
        print "ERROR: trying to get the md5 of the entry"

```

```

def md5_get(self):
    "Get the self.md5 from the self.permalink"
    try:
        self.md5 = hashlib.md5(self.permalink).hexdigest()
    except:
        print "ERROR: trying to md5_get entry %d" % self.id

```

```

def set_integrity(self):
    "Set to 0 int values that are 'None'"
    if not self.linked_source: self.linked_source = 0
    if not self.linked_product: self.linked_product = 0
    if not self.automatic_cat: self.automatic_cat = 0
    if not self.assigned_cat: self.assigned_cat = 0
    if not self.by_user: self.by_user = 0

```

```

def link_to_product(self):
    """
    Try to link the new entry to a product
    """
    try:
        associated = 0
        lowsubject = self.subject.lower()
        #Current source is the source of the entry in progress
        current_source = Sources(self.database)

```

```

current_source.get_one(self.linked_source)

current_product = Products(self.database)

if current_source.type is 1: # General source, look for 'general_search' y all products
    possible_linked_products = current_product.get_all() #Returns the id of all products
    for possible_product in possible_linked_products: #Try to link to every possible_linked_product
        #Search the general search string in the subject of the entry
        found = lowsubject.find(possible_product[5].lower())# + self.body.find(possible_product[5])
        if found>=0: #String found, associate 'self' entry to possible_product id
            self.linked_product = possible_product[0]
            self.matched_string = possible_product[5]
            self.automatic_cat = 3 #3rd highest category level
            associated = 1
            break

elif current_source.type is 2: # Specific source, look for 'specific_search' in one or more products
    if current_source.linked_product is 0: # The source can be associated to several products
        possible_linked_products = current_product.get_branded(current_source.search_brand)
        for possible_product in possible_linked_products:
            #Search the specific string in the subject of the entry
            found = lowsubject.find(possible_product[6].lower())
            if found>=0: #String found, associate 'self' entry to possible_product id
                self.linked_product = possible_product[0]
                self.matched_string = possible_product[6]
                self.automatic_cat = 4 #2nd highest category level
                associated = 1
                break

else: # The source can be associated just to 1 product
    self.linked_product = current_source.linked_product #Straight association
    current_product.get_one(current_source.linked_product)
    self.matched_string = current_product.specific_search
    associated = 1
    self.automatic_cat = 5 #Highest category level

else:
    print "An error occurred while linking the entry %d to a product" % self.id

```

```
    if associated:
        print "Entry %d ASSOCIATED to %d with string '%s'" % (self.id, self.linked_product, self.matched_string)
        self.modify()
    else:
        print "Entry %d not associated to any product" % self.id
except:
    print "ERROR: trying to link_to_product entry %d" % self.id

def insert(self):
    """
    Insert 'self' entry checking if it exists already using md5_exist
    """

def delete(self):
    """
    Delete self entry
    """

def modify(self):
    """
    Modify the entry with self values
    """

def show(self):
    """
    Show the entry on screen
    """

def save(self):
    """
    Save the self user on a file entry<id>.txt
    """
```

mod_syras.py

```

import sys      # For system exit

import cmd      # For command line interface

import webbrowser  # For opening a link with the browser

import getpass  # For getting secure password

from mod_db import DB      # My database module

from mod_sources import Sources  # My module for handling sources

from mod_products import Products  # My module for handling products

from mod_users import Users  # My module for handling users

from mod_email import Email_Sources  # My module for handling email sources

from mod_html import Html_Sources  # My module for handling html sources

from mod_rss import Rss_Sources  # My module for handling rss sources

from mod_entries import Entries  # My module for handling entries

class Console(cmd.Cmd):

    "Simple command line processor"

    #http://wiki.python.org/moin/CmdModule

    def __init__(self):

        "Constructor for the Console class"

        cmd.Cmd.__init__(self)

        self.prompt = "SyRAS:_"

        self.intro = """Welcome to SyRAS command line interface

        Please login or type 'help' for command guide of 'exit' to quit"""

    def do_login(self, line):

        """ User and password identification

        login <user>

        e.g: 'login user1'"""

        cad = line.split()

        if len(cad)!=1:

            print "ERROR: Invalid number of arguments"

            return

        elif (self.prompt != "SyRAS:_" ):

```

```

    print "There is a user logged, please logout first before changing the user"

    return

else:

    pw = getpass.getpass('Your password please: ')

    id = user.exists(cad[0], pw)

    if id:

        current_user.get_one(id)

        print "Welcome to SyRAS",cad[0]

        self.prompt = cad[0] + "@SyRAS:_> "

    else: print "Invalid user or password"

def do_logout(self,line):

    """ Logs out current user if any"""

    if self.prompt == "SyRAS:_> " :

        print 'There are none users logged'

    else:

        current_user.get_one(0)

        print 'User %s logged out' % self.prompt[:-10]

        self.prompt = "SyRAS:_> "

def do_collect(self, line):

    """ Collect entries from different sources

    collect email|html|rss|all

    e.g: 'collect rss'"""

    cad = line.split()

    if len(cad)!=1:

        print "ERROR: Invalid number of arguments"

        return

    elif (self.prompt == "SyRAS:_> "): print "You must be logged in to access this option"

    else:

        completed = 0

        if cad[0]!='email' or cad[0]!='all':

            print 'Collecting email'

            all_email = email.get_all_email()

            email.get_one(all_email[0][0])

```

```

    email.parse_feed()
    email.check_new_entries()
    completed = 1
    print 'All email sources collected'
if cad[0]=='html'or cad[0]=='all':
    print 'Collecting html'
    all_html=html.get_all_html()
    for id in all_html:
        html.get_one(id[0])
        html.parse_page()
        html.check_new_entries()
    completed = 1
    print 'All html sources collected'
if cad[0]=='rss'or cad[0]=='all':
    print 'Collecting rss'
    all_rss = rss.get_all_rss()
    for id in all_rss:
        rss.get_one(id[0])
        rss.parse_feed()
        rss.check_new_entries()
    completed = 1
    print 'All rss sources collected'
if completed == 0: print "Unknown source: '%s'" % cad[0]

```

```

def do_list(self, line):
    """ List all entries, products, sources or users
    list entries|products|sources|users
    e.g: 'list products'"""
    cad = line.split()
    if len(cad)!=1:
        print "ERROR: Invalid number of arguments"
        return
    elif (self.prompt == "SyRAS:_>"): print "You must be logged in to access this option"
    else:
        if cad[0]=='entries':

```

```

        print 'Printing ENTRIES (showing last 20):'

        print """(id, date, permalink, linked source, subject, body, matched string, linked product, importance, category, by
user, md5) \n"""

        entries = entry.get_all()

        for item in entries:

            print item

    elif cad[0]=='products':

        print 'Printing PRODUCTS:'

        print """(id, date, brand, name, version, general search string, specific search string, importance) \n"""

        products = product.get_all()

        for item in products:

            print item

    elif cad[0]=='sources':

        print 'Printing SOURCES:'

        print """(id, date, name, source type, type, subscription, linked product, search brand, url start, search pattern, last
check) \n"""

        sources = source.get_all()

        for item in sources:

            print item

    elif cad[0]=='users':

        print 'Printing USERS:'

        print "(id, date, name, (password omitted), email, warning level) \n"

        users = user.get_all()

        for item in users:

            print item

    else: print "Unknown source: '%s'" % cad[0]

def do_show(self, line):

    """ Show one entry, product, source or user

    show entry|product|source|user <id>

    e.g: 'show user 3'"""

    cad = line.split()

    if len(cad)!=2:

        print "ERROR: Invalid number of arguments"

        return

    elif (self.prompt == "SyRAS:_>"): print "You must be logged in to access this option"

```

```

elif cad[1].isdigit():
    id = int(cad[1])
    if cad[0]=='entry':
        entry.get_one(id)
        if entry.id!=0: entry.show()
    elif cad[0]=='product':
        product.get_one(id)
        if product.id!=0:product.show()
    elif cad[0]=='source':
        source.get_one(id)
        if source.id!=0:source.show()
    elif cad[0]=='user':
        user.get_one(id)
        if user.id!=0:user.show()
    else: print "Unknown parameter: '%s'" % cad[0]
else: print "Unknown parameter: '%s'" % line

```

```

def do_save(self, line):
    """ Save one entry, product, source or user in a txt file
    save entry|product|source|user <id>
    e.g: 'save user 3' will save it in 'user3.txt' file"""
    cad = line.split()
    if len(cad)!=2:
        print "ERROR: Invalid number of arguments"
        return
    elif (self.prompt == "SyRAS:_>"): print "You must be logged in to access this option"
    elif cad[1].isdigit():
        id = int(cad[1])
        if cad[0]=='entry':
            entry.get_one(id)
            if entry.id!=0:
                entry.assigned_cat=3
                entry.by_user=current_user.id
                entry.modify()
                entry.save()

```

```

elif cad[0]=='product':
    product.get_one(id)
    if product.id!=0:product.save()
elif cad[0]=='source':
    source.get_one(id)
    if source.id!=0:source.save()
elif cad[0]=='user':
    user.get_one(id)
    if user.id!=0:user.save()
else: print "Unknown parameter: '%s'" % cad[0]
else: print "Unknown parameter: '%s'" % line

def do_browse(self, line):
    """ Open one entry or source permalink in the web browser
    browse entry|source <id>
    e.g: 'browser entry 100' will open it's permalink in the web browser"""
    cad = line.split()
    if len(cad)!=2:
        print "ERROR: Invalid number of arguments"
        return
    elif (self.prompt == "SyRAS:_> "): print "You must be logged in to access this option"
    elif cad[1].isdigit():
        id = int(cad[1])
        if cad[0]=='entry':
            entry.get_one(id)
            if entry.id!=0:
                entry.assigned_cat=2
                entry.by_user=current_user.id
                entry.modify()
                webbrowser.open(entry.permalink)
                print "ENTRY %d opened in the web browser" % entry.id
        elif cad[0]=='source':
            source.get_one(id)
            if source.id!=0:
                webbrowser.open(source.subscription)

```

```

        print "SOURCE %d opened in the web browser" % source.id

    else: print "Unknown parameter: '%s'" % cad[0]

else: print "Unknown parameter: '%s'" % line

def do_priority(self, line):
    """ Changes to important the priority of one entry or product
    priority entry|product <id>
    e.g: 'priority product 10' will change it's priority to important"""
    cad = line.split()
    if len(cad)!=2:
        print "ERROR: Invalid number of arguments"
        return
    elif (self.prompt == "SyRAS:_> "): print "You must be logged in to access this option"
    elif cad[1].isdigit():
        id = int(cad[1])
        if cad[0]=='entry':
            entry.get_one(id)
            if entry.id!=0:
                if (entry.assigned_cat!=3):
                    entry.assigned_cat=2
                    entry.by_user=current_user.id
                    entry.modify()
                    print "ENTRY %d changed to Important" % entry.id
        elif cad[0]=='product':
            product.get_one(id)
            if product.id!=0:
                product.importance=2
                product.modify()
                print "PRODUCT %d changed to Important" % product.id
    else: print "Unknown parameter: '%s'" % cad[0]

else: print "Unknown parameter: '%s'" % line

def do_delete(self, line):
    """ Delete one entry, product, source or user from the database (only admin)
    delete entry|product|source|user <id>

```

```

e.g: 'delete user 3' will delete it from the database"""

cad = line.split()

if len(cad)!=2:

    print "ERROR: Invalid number of arguments"

    return

elif (self.prompt != "admin@SyRAS:> "): print "You must be administrator to access this option"

elif cad[1].isdigit():

    id = int(cad[1])

    if cad[0]=='entry':

        entry.get_one(id)

        if entry.id!=0: entry.delete()

    elif cad[0]=='product':

        product.get_one(id)

        if product.id!=0:product.delete()

    elif cad[0]=='source':

        source.get_one(id)

        if source.id!=0:source.delete()

    elif cad[0]=='user':

        user.get_one(id)

        if user.id!=0:user.delete()

    else: print "Unknown parameter: '%s'" % cad[0]

else: print "Unknown parameter: '%s'" % line

def do_EOF(self, line):

    print "SyRAS closed!"

    return True

do_exit=do_EOF

```

syras.py

```

if __name__ == '__main__':

    #Initiation of database

    database = DB()

```

```
database.connect()

#Initiation of user, product and entry
entry = Entries(database)
product = Products(database)
user = Users(database)

current_user = Users(database)
current_user.get_one(0)

#Initiation of sources
email = Email_Sources(database)
html = Html_Sources(database)
rss = Rss_Sources(database)

#Initiation of console
con = Console()
try:
    con.cmdloop()
except KeyboardInterrupt:
    con.do_EOF(None)
finally:
    database.disconnect()
    sys.exit()
```

syrasloop.py

```
        # Imports

import time      # For waiting a time interval
import sys      # For system exit
import getpass  # For getting secure password

from mod_db import DB      # My database module
```

```
from mod_sources import Sources    # My module for handling sources
from mod_products import Products  # My module for handling products
from mod_users import Users        # My module for handling users
from mod_email import Email_Sources # My module for handling email sources
from mod_html import Html_Sources  # My module for handling html sources
from mod_rss import Rss_Sources    # My module for handling rss sources
from mod_entries import Entries

if __name__ == '__main__':

    #Initiation of database

    database = DB()
    database.connect()

    #Initiation of user, product and entry

    entry = Entries(database)
    product = Products(database)
    source = Sources(database)
    user = Users(database)

    #Initiation of sources

    email = Email_Sources(database)
    html = Html_Sources(database)
    rss = Rss_Sources(database)

    #Waiting time in seconds between loops

    wait_time = 300

    logged = 0

    print "Welcome to SyRAS loop, please log in"
    while logged == 0 :
        us = raw_input('Your user: ')
        pw = getpass.getpass('Your password: ')
        id = user.exists(us, pw)
```

```

if id:
    user.get_one(id)
    print "Welcome to SyRAS",us
    logged = 1
else: print "Invalid user or password, please try again"

# Check all sources for new entries every "wait_time" seconds
try:
    while True:
        print '\nStarting the collecting process'
        print '-----'
        print '1- Collecting Email feeds'
        all_email = email.get_all_email()
        email.get_one(all_email[0][0])
        email.parse_feed()
        email.check_new_entries()

        print '\n2- Collecting HTML feeds'
        all_html=html.get_all_html()
        for id in all_html:
            html.get_one(id[0])
            html.parse_page()
            html.check_new_entries()

        print '\n3- Collecting RSS feeds'
        all_rss = rss.get_all_rss()
        for id in all_rss:
            rss.get_one(id[0])
            rss.parse_feed()
            rss.check_new_entries()

        print '\nWaiting for %d seconds and starting again' % wait_time
        time.sleep(wait_time)

except KeyboardInterrupt:

```

```
    print "SyRAS loop stopped by user"  
except:  
    print "Error while checking new entries"  
finally:  
    database.disconnect()  
    sys.exit()
```

APPENDIX 3: SECURITY ADVISORIES

This appendix shows several examples of real security advisories published by vendors or researchers (as they could be collected by the SyRAS system) and the final version of the security notice written by the author of this thesis and published in Spanish by the IT security company Hispasec Sistemas.

Security Advisory 1: Broad, Elazar. 10th March 2008. "Real Networks RealPlayer ActiveX Control Heap Corruption". Full-disclosure

Link: <http://archives.neohapsis.com/archives/fulldisclosure/2008-03/0157.html>

Text:

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

Who:

Real Networks

<http://www.real.com>

What:

Real Networks Real Player is a popular media player.

How:

Real Player utilizes an ActiveX control to play content within the users browser.

rmoc3260.dll version 6.0.10.45

{2F542A2E-EDC9-4BF7-8CB1-87C9919F7F93}

{CFCDA03-8BE4-11CF-B84B-0020AFBCCFA}

It is possible to modify heap blocks after they are freed and overwrite certain registers, possibly allowing code execution. Like

so:

- - - - -

```
var buf = "";
```

```
while (buf.length < 1005) buf = buf + 'A';
```

```
m = obj.Console;
```

```
obj.Console = buf;
```

```

obj.Console = m
//repeat
m = obj.Console;
obj.Console = buf;
obj.Console = m --> Should crash here
- -----
Workaround:
Set the killbit for this control. See
http://support.microsoft.com/kb/240797
Fix:
No official fix known
Exploit:
Working on it
Elazar
-----BEGIN PGP SIGNATURE-----

```

Security Notice 1: Molina, Pablo. 11th March 2008. "Ejecución de código a través de un ActiveX de RealPlayer". Una-al-día - Hispasec Sistemas S.L.

Link: <http://www.hispasec.com/unaaldia/3426>

Text:

Se ha descubierto una vulnerabilidad en RealPlayer que podría ser aprovechada por un atacante remoto para causar una denegación de servicio y posiblemente, ejecutar código arbitrario en un sistema vulnerable.

RealPlayer es empleado por millones de usuarios de Internet para reproducir archivos multimedia tanto de audio como de vídeo. El fallo ha sido destapado por Elazar Broad, investigador especializado en descubrir fallos en los controles ActiveX de algunas de las aplicaciones más extendidas.

Según Broad, el problema está causado por un desbordamiento en el componente ActiveX RealAudioObjects.RealAudio (rmoc3260.dll) versión 6.0.10.45, que podría permitir a un atacante sobrescribir en bloques de memoria de la pila basada en heap y modificar así ciertos registros. Esto podría ser aprovechado para hacer que, si se visita con Internet Explorer una página web especialmente manipulada, se ejecutase código arbitrario.

El problema podría afectar a todas las versiones de RealPlayer y su descubridor dice estar trabajando en una demo, así que es posible que en pocos días esté disponible un exploit público que haga uso de la vulnerabilidad para ejecutar código arbitrario de forma remota.

No existe parche disponible y como contramedida, se recomienda activar el kill bit del control ActiveX para evitar que sea llamado por Internet Explorer. Es posible hacerlo guardando este archivo con extensión .reg y ejecutarlo como administrador:

Windows Registry Editor Version 5.00

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\
ActiveX Compatibility\{2F542A2E-EDC9-4BF7-8CB1-87C9919F7F93}]
"Compatibility Flags"=dword:00000400
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\
ActiveX Compatibility\{CFCDAA03-8BE4-11CF-B84B-0020AFBCCFA}]
"Compatibility Flags"=dword:00000400
```

También existe la posibilidad de deshabilitar la ejecución automática de ActiveX en el navegador en páginas no confiables.

Según nuestros registros en el servicio de alertas SANA, desde el pasado mes de octubre se han encontrado hasta cinco vulnerabilidades de gravedad alta en RealPlayer relacionadas con controles ActiveX.

Security Advisory 2: Security Bulletin. 24th February 2009. "Flash Player update available to address security vulnerabilities". Adobe Systems Incorporated.

Link: <http://www.adobe.com/support/security/bulletins/apsb09-01.html>

Security Notice 2: Molina, Pablo. 25th February 2009. "Actualización por vulnerabilidades de ejecución de código en Adobe Flash Player". Una-al-día - Hispasec Sistemas S.L.

Link: <http://www.hispasec.com/unaaldia/3777>

Security Advisory 3: Microsoft Technet. 23th October 2008. "Microsoft Security Bulletin MS08-067 – Critical. Vulnerability in Server Service Could Allow Remote Code Execution (958644)". Microsoft Corporation.

Link: <http://www.microsoft.com/technet/security/bulletin/MS08-067.msp>

Security Notice 3: Molina, Pablo. 23th October 2008. "Microsoft publica una actualización crítica fuera de ciclo". Una-al-día - Hispasec Sistemas S.L.

Link: <http://www.hispasec.com/unaaldia/3652/>