MICS Summer Internship Project Report

System for Testing Antispam Solutions

by

Raphael Naefen

The work is proposed and supervised by:

Slavisa Sarafijanovic and Jean-Yves Le Boudec

EPFL, October 2006

Content

Content ... 1
Abstract ... 2
Credits ... 3
1. Related work ... 4
   1.1 An artificial immune system for spam being developed at EPFL [1] ... 4
   1.2 TREC framework for testing spam solutions [2] ... 4
   1.3 Postfix [7] ... 4
2. Architecture of our testing system ... 5
3. Design and implementation of the components ... 6
   3.1 Email servers ... 6
   3.2 Email filters and appropriate interfaces ... 6
   3.3 Email clients / users ... 7
   3.4 Testing-services server ... 7
   3.5 Scripts to automate the testing procedure ... 8
4. Testing software and files' description ... 9
5. Checking the testing system ... 11
6. Conclusion: ... 12
References: ... 13
Appendix 1: Usage instructions ... 14
   A1.1 Requirements ... 14
   A1.2 Overview of the testing software files ... 14
   A1.3 Steps to prepare and run the test: ... 15
   A1.3.1 General Settings that should be done/checked at the host machine: ... 15
   A1.3.2 Steps to install the testing system ... 15
   A1.3.3 Steps to run the testing system ... 15
   A1.3.4 Testing a custom filter ... 16
   A1.4 Where are (raw) results stored and what is their meaning ... 16
Appendix 2: Unsolved bugs ... 17

Abstract

Today, surprisingly, there is not a known ready-to-use solution for off-line testing of antispam software in the scenarios that would include all next features: a) use of multiple networked mail servers and clients, b) included e-mail user behavior, c) use of important spamming techniques. On the other hand, many filters are based not only on processing an incoming email per se, but also use networking to obtain or exchange additional information used for filtering, such is email signatures, black-listed senders etc. Also, the timing in the messages exchange and user behavior might strongly impact the filtering results.
In this project we developed some basic ready-to-use testing system that tries to meet the above requirements. Though the testing system is primarily aimed at testing the antispam solution developed at EPFL within the MICS project [1], the general interfaces are provided for adopting the system quickly and easily in order to test other antispam solutions. Appropriate usage instructions are also provided.

Note: The work has been focused on producing and checking the testing system. In the project proposal we also planned to prototype and test the antispam system designed within the larger MICS project [1], but during the project we decided to give up this part in order to be able to make a good testing system that can be easily used not only for our but also for other filters, with the idea to make it publicly available and usable in near future.

Credits

In this project I did the implementation of the antispam testing system. The design of the antispam testing system's architecture and proposals how to implement the components are done by Slavisa Sarafijanovic, one of the supervisors of the project.

Special thanks to Marc-Andre Luthi, the hosting lab (LCA at EPFL) IT administrator, for useful discussions and hints on some implementation choices and some choices of the existing software used in the project, as well as direct help with initial and subsequent tricky settings on the machine on which we developed the system.

1. Related work

1.1 An artificial immune system for spam being developed at EPFL [1]

The system [1] introduces two possibly advantageous novelties compared to the existing antispam solutions: a) a representation of the email content designed for fundamentally better resistance to the spam obfuscations, and b) processing of both the profiles of the users and implicit or explicit feedback from the users is integrated with collaborative spam-bulk information processing. Both the representation and processing are based on analogies to the human immune system. Different antispam systems, typically placed at email servers, are networked and collaborate to each other for better filtering. Testing and evaluating such an antispam system requires a complex testing system that meets the requirements listed in the abstract of this document.

1.2 TREC framework for testing spam solutions [2]

TREC is one of rare publicly known efforts to provide systematic tools for testing different antispam solutions. Nevertheless, their solution allows to test only filters that process emails as an independent email-stream seen by one recipient, and is not appropriate for the solutions such is [1] and for many other state-of-the-art deployed solutions based on networking and users feedback. In the absence of appropriate tools to test our antispam solution, we decided to make our own testing system that will meet these requirements, but also be easily usable for test other antispam solutions.

1.3 Postfix [7]

Postfix is a free software / open source mail transfer agent (MTA), a computer program for the routing and delivery of email. It is intended as a fast, easy-to-administer, and secure alternative to the widely-used Sendmail MTA. Postfix is the default MTA for a number of Unix (-like) operating systems.

2. Architecture of our testing system

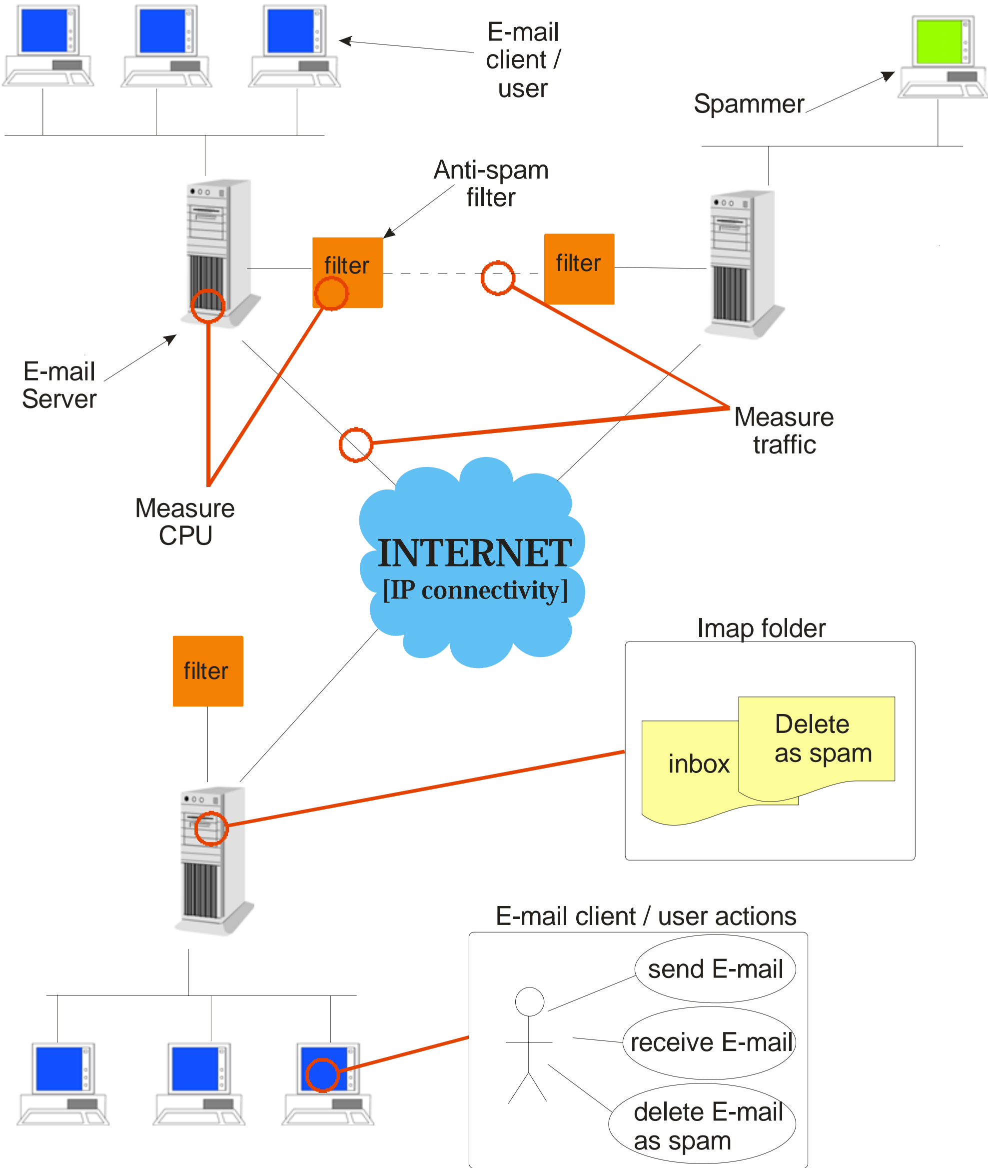

Figure 1 Logical architecture of the testing system
(**Note:** spam filters are connected to each other through Internet and the dashed line on the figure only represents logical connection; a filter may be on the same machine on which the corresponding email server is, or on a different machine which is usually on the same LAN.)

The architecture of our testing system is show in the Figure 1. In order to implement this naturally distributed architecture in a local testing environment we used Linux VMs (virtual machines). Thus every computer present in Figure 1 is implemented as one VM. To run these virtual machines we decided to use Xen [4] (Virtualization Software) and Debian Linux distribution. We run all the virtual machine on the same real machine, though very small modifications are needed in order to run all the testing system on more then one real machines, for example on a laboratory cluster. This might be interesting if many VMs need to be run. To setup the testing system, we have developed python scripts that build all VMs and specialize them to be: server, filter or email client / user.

3. Design and implementation of the components

3.1 Email servers

One important goal of the project is to provide interfaces to the filter products that are practically the same as in reality, so that testing of an antispam product requires only the work of installing that product as if it is installed into a real email system. This means the system can be easily used to comparatively test different products, and it can also be easily used by other people without looking into the details of its implementation. Another goal is to achieve a realistic instantiation of the traffic and CPU usage by important emailing system components and to measure these metrics easily. Also, the email messages should be modified while in the transit through the emailing network as realistically as possible. To achieve the above goals, we use real email servers in our testing system. We use Postfix SMTP server [7,11] and dovecot IMAP Server [8], and run one pair of these servers on one virtual machine for each email domain.

3.2 Email filters and appropriate interfaces

Postfix already has a built in interface for adding an email filter, whether the filter is running on the same (local host) or separate machine. To implement the interface for the feedback from the user, we patch the IMAP server. The interfaces and flow of the information among the SMTP server, IMAP server, and filter is shown on the Fig. 2.

New incoming email is received by SMTP server and is optionally passed to the filter. If the filter finds the email being spam it might drop it or mark it as spam and pass it back to the SMTP server to be put into the memory for the marked recipient. If it finds the email to not be spam, it always passes it back to the SMTP server. User access/reads/moves emails using email client that connects to the IMAP server which manipulates the emails in the users account space on the server machine. If the user deletes the message as spam (DAS) this information is passed to the filter (the patch).

We decided to patch the server instead of adding plug-ins to all emails clients, because in this way DAS action is totally transparent to the users' email software, and we don't need to develop a plug-in for all different email client that exist on the market. This patch was done using C language in order to replace some part of dovecot source code. When the patch detects DAS event, it transmits the information to the antispam filter. The current version of the patch only resends the email to the user through Postfix and adds some necessary information so that filter knows the email is actually destined to the filter and not to the user. This can be changed to communicate directly to the filter (for example SMTP interface is already done for outgoing filters on port 10027), and exchange already formatted information that contains unmodified email, user action, and other information if needed.

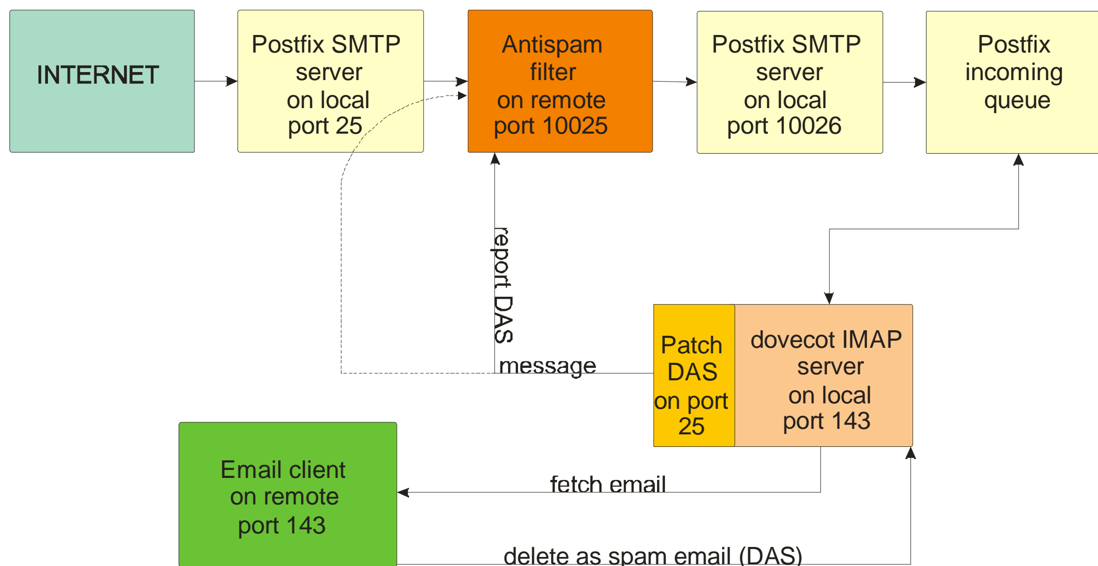

Figure 2. Antispam filter's position in the testing system

We have configured postfix to use a simulated remote filter developed during the project. We use this simulated filter in order to be able to validate our system and to check whether all other scripts and programs run correctly. The simulated antispam filter marks the messages as spam with given probability which is different for the spam and non-spam messages (we assume the testing corpus defines each message as spam or non-spam). The postfix is configured to use the filter in "Before-Queue Content Filter" mode. This mode has been chosen because it is simple, and permits to use a filter on a remote machine. It is important to note that by default the filter will process all messages incoming and outgoing if we want other behavior it can be obtain by editing postfix configurations files.

3.3 Email clients / users

To represent a user and its email client application, we have developed a python script that simulates both: the email client and the user behavior. This script is running in all client machines during a test. We use custom software for the client in order to save CPU and memory be keeping only the necessary client functions. For the moment the user behavior is scheduled to read and send email following a customizable distribution in time. When a user receive a SPAM message it can deleted it as SPAM ("DAS"), this action permit to report a SPAM message to the filter. User deletes received spam emails with given probability which is a parameter.

3.4 Testing-services server

We have developed a server program that provides emails and other information to the testing system components. This program is running on the real (physical) machine. It has two main functions. First it constructs a list of the unique identifiers of emails present in testing corpus, each identifier followed with correct email classification mark (spam or ham). During the test the script provides these emails to clients when they want to send a new message. The script also informs email client or filter on the message type (spam or ham). At the end of a test the script collects

logs from email clients and constructs a global log file that contains all email exchange and filtering information.

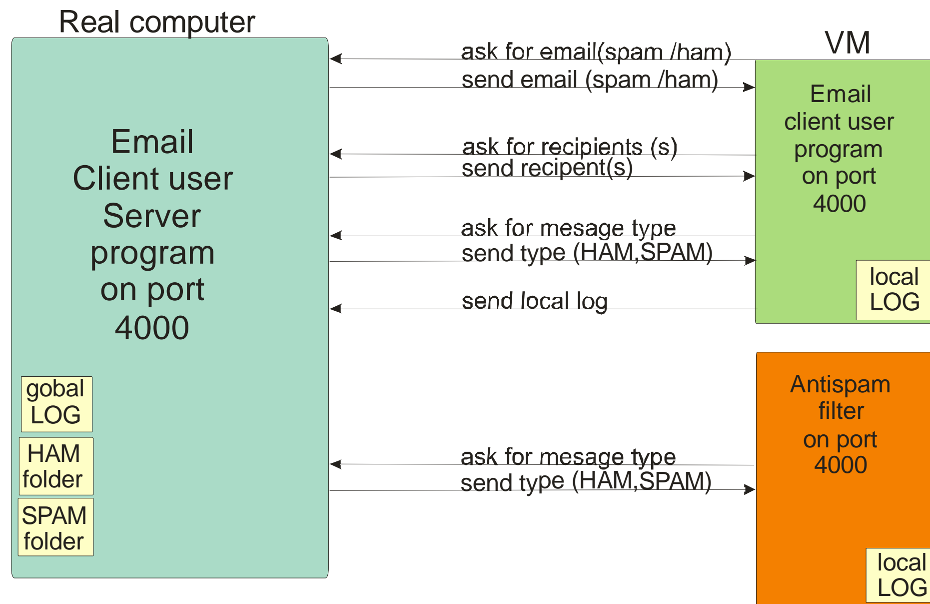

Figure 3 The role of the testing-services server (emailClientUserServer.py script)

We used XML format [9] in log files. We define a logger function that permits to add very easily new log entry. XML is convenient for further processing, and in this project we process this file using XSLT [6] to create a HTML output. More complex processing could be done in future work for example with python.

3.5 Scripts to automate the testing procedure

We have written the scripts that automate preparation and execution of the test. These scripts run and configure VMs, run appropriate components on each VM, configure IP addressing, user accounts etc., and start the system by activating users to send/read/possibly delete as spam emails, which further triggers all other events.

During one test all email clients/users scripts are running and exchanging email for a predetermined testing system time (real execution time might be smaller as we apply time scaling). Normal users send HAM messages, and the spammer sends SPAM through the network. The timing of users’ actions (these are the events that trigger all other events in the system) is randomly distributed with a configurable distributions and their parameters.

At the end of a test all programs are terminated and a global log file is built. This file contains all events that occur during the simulation. We process this file to obtain important results like percentage of true positives or number of exchanged emails.

4. Testing software and files' description

Most of the testing system code is written in python [3]. We decided to use python because it is convenient for both scripting and communication tasks. To fasten the development and simplify the code, we use python library and other libraries present on Internet (like paramiko [5] for SSH [10]) whenever it is possible.

The code is written in object oriented way. We try to put all important variables in a global configuration file.

Here we briefly review all scripts and programs developed during this project, as well as the important configurations files:

- setupVm.py: This script setups Xen VMs of the antispam testing system:
  It has three main parameters, the number of the domains that we want to create, the number of clients per domain, and the number of spammers present in the system.
  To create all VM the script does the following things. For all domains it takes the base Xen image and copies it to generate all VMs. It creates the Xen configuration file for all VM. Then it starts all the VMs, change their specific configurations (like IP addresses), and restarts all the VMs. The script then specialize each VM to be one of the following 3 types: client, spammer, server + filter. This specialization means changing configuration files for some programs and starting appropriate programs.

- emailClientUser.py: the script is divided in 4 main parts: 1) an email client with functions to send and receive message; 2) a user to do user actions like read mail, delete as spam, and compose new message; 3) logger of events; 4) "server connector" used to connect to emailClientUserServer.

- emailClientUserServer.py: this is a server script that is run on the real (physical) computer (not on one of the VMs). The script has 4 main functions. The first function is to provide emails to each emailClientUser running program whenever one of its simulated users wants to send an email. The second function is to maintain list of message-ids and their associated spam/ham types, and to answer by type when it is queried by the id from any testing system component. This emulates users' ability to distinguish if the email is spam or not (queries from emailClientUser programs) and was useful also for implementing simulated filter. The third function is to provide emailClientUser programs with the recipient information when an email is to be sent. And the fourth function is to build a global log file, buy collecting local logs form all emailClientUser programs at the end of the simulation.

- processLog.py: the process log script has been developed to collect log file on the remote filters and the log file done by emailClientUserServer. When all files are collected it creates the final global log XML file.

- globalLog.xml: this file isn't a script but the result of a test. In this file, there is a list of all events that happen during the simulation.

- globalLog.xslt: this XSLT program is used to generate a HTML output in order to render a result readable by human. This program also does some useful statistic, like number of sent messages, probability of true positive and so one.

- remotefilter.py: the remote filter is a python script that simulates the behavior of an antispam filter. This simulated filter is developed in order to be able to check if the complete system works as expected. It has 3 ESMTP [12] interfaces: two to communicate with postfix (one to receive mail, one to transmit checked email), and the third one to receive information from other antispam filters or from the IMAP server when a DAS event is announced. Filtering is simulated by using emailClientUserServer to learn the type of the message, and then based on the type marking the email as spam or not.

- Cmd-copy.c: this file is written in C language. It is a patched version of a dovecot function that is called when a message is copied from one IMAP folder to another. When this function is called we create an ESMTP connection and we transmit message and user information to the filter. (We have some problem do directly communicate with the filter. So we send the message to the local postfix server, therefore message arrive to the filter like other message. And the filter is able to detect that it isn't a email but it is a DAS event by recognizing keyword in the message's subject)

- Procmail: we configured procmail to route emails into the correct IMAP folders: the emails that passed the filter and that contain the SPAM Header are put into a Spam folder, and other emails that passed the filter are put into Inbox.

- Postfix: Postfix contains two configurations files that we have edited to install the filter and configure postfix on all servers.

- Dovecot: We have edit Dovecot to enable IMAP services and point them to use the correct folder (the same that are used by Procmail).

- SSH key: we decided to use SSH v2 with SSH key. All VM are configured to use the key that we provide with our project. If you need to use another key, just edit the base VM image to add your key.

5. Checking the testing system

By using a simulated filter with known performances, we statically determine that our antispam testing system behaves as expected with respect to the input information provided to and the output obtained from the simulated filter.

In order to compute confidence intervals of the filter performances observed by the testing system, we run the system twenty times. The tested system contained 3 email servers, and each server has one client, 2 domains contained normal users and one contained a spammer. During each test approximately 500 mails are exchanged in the system.
All simulated filters worked with the same probabilities:

- P_1 = P { mark as spam | SPAM } = 0.9
- P_2 = P{ mark as spam | HAM } = 0.1

For true positives we obtain:
TP mean 0.8951 and confidence interval 95% = [0.8867 - 0.9034]

For False positive we have obtain:
FP: mean 0.0978 and confidence interval 95% = [0.0884 - 0.1071]

We can see that the observed filter performances correspond well to its real (known and simulated) performances.

6. Conclusion:

This project was interesting because I had opportunity to learn and practically test building of networked software and scripting, and a good point was that this project regroups a lot of different domains.
I improved my Linux knowledge: configuration of a Linux machine and advanced use of Linux commands, like use of SSH for remote scripted operations.
I learned python, and now I I'm able to write complex scripts and programs or I can write very quickly a small but useful script.
I have written or patched some networked applications with Python and C (written: simulated email client and user; patched: SMTP server, IMAP client), and put them to work together. Doing this I learnt and improved in Internet application software, multithreading application, and use of regular expressions and processing XML file.
I also improved my knowledge on networking: I have learned some protocols that are part of our system (IMAP and SMTP). Maybe I will never use these protocols again but now I know how to find information and details about networking protocols.

I'm aware that my programs contain still some bugs. But I think that I have well implemented a good basic framework: we have a running and usable simple version of the testing system. I hope other people will be able to quickly extend my work, adding new stuff like better user behavior or CPU and traffic measurement.

References:

[1] Antispam MICS project: http://icawww1.epfl.ch/is/ais/antispam
[2] TREC: http://plg.uwaterloo.ca/~gvcormac/spam/
[3] Python http://www.python.org/
[4] Xen : http://www.xensource.com/
[5] Paramiko http://www.lag.net/paramiko/
[6] xslt http://www.w3.org/TR/xslt
[7] Postfix: http://www.postfix.org/
[8] dovecot http://www.dovecot.org/
[9] xml http://www.w3.org/XML/
[10] ssh http://www.openssh.com/fr/index.html
[11] SMTP http://rfc.net/rfc881.html
[12] ESMTP http://rfc.net/rfc3885.html
[13] IMAP http://rfc.net/rfc2060.html
[14] pycrypto 1.9+ http://www.amk.ca/python/code/crypto.html

Appendix 1: Usage instructions

A1.1 Requirements

- Linux operating system
- Xen v3
- Python v2.4
- pycrypto 1.9+ python library
- Paramiko python library
- SSH v2
- ~20 GB of free space on hard-disk (3GB per VM)
- User privilege:
  - Root privilege is required during installation (to install needed libraries and disable firewall; as Xen configuration file is put in /etc (/etc/xen), root privilege is also needed for configuring VMs, In order to avoid this restriction it is possible to change this folder in the configuration file.
  - User privilege is sufficient to run and test antispam system upon the VMs are configured and running

A1.2 Overview of the testing software files

The code is organized in 6 main folders (present on source directory):

- EmailClient folder's contains file for emailClientUser, emailClientUserServer and the main configuration file
  - emailClientUser.py
  - emailClientUserServer.py
  - SPAM and HAM folder
  - defaults.cfg
- Filter contains the basic antispam filter and the dovecot patch (in cmd-copy.c). We also include procmail configuration file
  - remoteFilter.py
  - cmd-copy.c
  - procmail.rc
  - Zspam.rc
- LogProcess contains script to process log after one test , result (xml file)
  - processLog.py
  - globalLog.xml
  - globalLog.xslt
- run Test contains the script to run a new test
  - runOneTest.py
- VM contains setup file to install the testing system
  - setupVm.py
- Utils contains some little script and tools to help for future development and main configuration file of used program, this folder also include the SSH key
  - some utility file for future development
  - ssh2 key
- XenBaseimage contains the base image of VM
  - Xen base image

A1.3 Steps to prepare and run the test:

- Download project onto the testing machine

A1.3.1 General Settings that should be done/checked at the host machine:

- Check requirements
- Disable Linux firewall
  - For example type: "/etc/init.d/iptables stop" if you use netfilter
- Check Xen is running
  - Type "xm list", if you get warning message:
    1. Verify Xen is correctly installed
    2. Verify Xen is running (Type "xend start" in order to start Xen daemon)
- Install the private SSH key
  - Copy the SSH-key locate in utils directory (file name is "id_dsa") to your "$USER/.ssh/ " folder
  - This key must be copied in all user account that wanted to run or install the testing system.

A1.3.2 Steps to install the testing system

- Do all steps as root user
- Run "sh unpackfiles" in emailClient directory
- Unzip base Xen image according to the directory given in configuration file(by default "/vserver/images")
  - VM username = "root" and password is empty
- Remove old virtual machines (if exist)
- Edit default configuration file to select how many domain, email clients and spammer you want
  - NB: if these parameters are changed some changes need to be done in different scripts manually. (see source code for more information)
- If you don't use the given SSH key you need to launch the VM base image and update the SSH key: (edit the "authorized_keys2" file by adding your public key)
- Setup the environment
  - Command "sh setup.sh" in source directory

A1.3.3 Steps to run the testing system

The project has still some non-automated parts or bugs that complicate steps to run a test:

- Change manually in emailClientUserServer.py the list of recipients
- Launch manually emailClientUserServer.py
  - Open a new terminal
  - go to email client directory
  - type "python emailClientUserServer.py"
- Connect to all servers and launch the remote filter (if you use it)
  - Type "xm console machine name" to connect to a VM
  - Login as "root" and empty password
  - type 2 times "killall python"
  - type "python remotefilter.py &"
- choose option (seed, duration) in default.cfg (section test)
- in source directory run "sh runTest.sh"

- wait until you seed a message on emaiClientUserServer that all log have been received
    - close emailClientUserServer
- process logs manually
    - go to log folder
    - type “python processLog.py”
- the xml file doesn’t open with every browser
    - type for example: “mozilla-firefox globalLog.xml”

A1.3.4 Testing a custom filter

By default the test is done with the simulated filter that we have developed. If you want to test another filter there are few steps to follow but the main job is only to install you filter in the base VM image, before installing the testing system (i.e. before running all the VMs), and to change scripts to run this filter instead of the simulated filter. The alternative is to first install the testing system (run all the VMs), and then by hand or by a custom script install and run the new filters, instead of running the existing simulated filters.

- Update postfix main.cf (if needed)
- Update postfix master.cf (if needed)
- Put the filter in VM base image before installing the testing system (or replace simulated by new filters in all VMs after running system is installed)

A1.4 Where are (raw) results stored and what is their meaning

Log file permits to show what is happening on the system. Each script has it own log file and at the end we construct a global log via the email client user server and the script process log. The global results file (“globalLog.xml”) is in “logProcess” folder and it contains all local logs and information about the test.

Appendix 2: Unsolved bugs

Unfortunately due the time limit this project contains some important unresolved bugs:

- Remote filter timeout problem: in case of no activity between the filter and Postfix, the connection is closed and the filter is closed. Therefore to bypass this bug you need to stop and start each filter manually before a test. (It is possible to do this by a script but take care of closing all threads and sockets).

- EmailClientUser.py is crashing: in some simulations one of the emailClientUser is crashing. We don’t find why for the moment. If this bug occurs, you will only receive n-1 logs (n = number of emailClientUsers). You need to close manually emailClientUserServer and restart the simulation again.

- EmailClientUserServer socket problem: when we close emailClientUserServer the socket is not released so we need to wait some time before starting it again. This problem prevents to include loading emailClientUserServer in a script.

- Seed value: In this project we use a python seed function in order to provide random and so different instances of the system run, but repeated runs with the same seed should lead to deterministic or at least very similar results (the system is run within one machine); we noticed that in many case the result isn’t deterministic for the same seed (the reason could be use of multiple virtual machines and their intercommunication randomness, but we didn’t clarify this issue).

- There are some other minor bugs that we don’t mention here, but they are mentioned in the source code directly.