Mapping Linux Security Targets to Existing Test Suites

C.A. Ardagna[1], E. Damiani[1], N. El Ioini[2], F. Frati[1], P. Giovannini[2] and R. Tchokpon[3]
[1] Department of Information Technology - University of Milan
via Bramante, 65 – 26013 Crema (CR) - Italy
{ardagna,damiani,frati}@dti.unimi.it
[2] Free University of Bozen-Bolzano
{Pietro.Giovannini,Nabil.ElIoini}@stud-inf.unibz.it
[3] Institut de Mathématiques et de Sciences Physiques – Benin
ricotchfr@yahoo.fr

Abstract. The Common Criteria standard provides an infrastructure for evaluating security functions of IT products and for certifying that security policies claimed by product suppliers are correctly enforced by the security functions themselves. Certifying Open Source software (OSS) can pave the way to OSS adoption in a number of security-conscious application environments. Recent experiences in certifying Linux distributions has pointed out the problem of finding a mapping between descriptions of OSS security functions and existing test suites developed independently, such as the Linux Test Project. In this paper, we describe a mechanism, based on matching techniques, which semi-automatically associates security functions to existing test suite such as the ones developed by Open Source communities.

1 Introduction

Security software evaluation and certification have become an important technique for selecting IT products to be deployed in security-conscious environments. The main goal of software security evaluation is to certify that software products provide and correctly implement some required security functionalities. This means that a good evaluation criterion should give an assessment of system security revealing all the security problems and weaknesses of IT products, which could lead to any unexpected behaviour.
The first attempt (carried out in 1985) to create a standard for security certification was the Trusted Security Evaluation Criteria (TCSEC) by U.S Department of Defense, also known as *Orange Book*. The Orange book has been defined to assist the design and development of security requirements and to fill the communication gap between vendors, evaluators, and customers [16]. However, the Orange Book soon proved too rigid to adapt to the new changes.

Please use the following format when citing this chapter:

Ardagna, C.A., Damiani, E., El Ioini, N., Frati, F., Giovannini, P. and Tchokpon, R., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 29–45.

From 1990, a specific European standard known *as Information Technology Security Evaluation Criteria* (ITSEC) has been adopted in Europe [20]. A few years later, several countries like Germany, United Kingdom, France and Canada have introduced their own national standards for security evaluation. The diversity of certifications standards, however, made the process of certifying an IT product at an international level somewhat difficult and expensive.

More recently, the *Common Criteria* (CC) certification standard has been defined to fulfil the needs of an international standard for affordable software security certification. Common Criteria is now an ISO standard (ISO 15408); it provides an unified process and a flexible framework to specify, design, and evaluate the security properties of IT products [9]. The CC standard is the result of a joint effort of many international organizations that have worked for two decades to specify a common structure for evaluating security properties of IT products [9]. A major goal of the CC evaluation is to certify that the security policies claimed by the developer are correctly enforced by the security functions of the product under evaluation.

Today, many development communities and other organizations working on Open Source software (OSS) recognize that a standard way to obtain the security certification of an OSS product can pave the way to its adoption in a number of security-conscious application environments.

In this paper, we focus on CC certification testing, which is aimed at ensuring that security functions under evaluation are correctly implemented and work according to the specification. In this context, one of the main problems to be faced is to find an easy map between test suites and security functions. This is especially true in OSS scenario, where security functions developers are often different from test suites developers. Recent experiences in certifying Linux distributions [18,19] has pointed to the problem of finding a mapping between descriptions of OSS security functions and existing test suites developed independently, such as the Linux Test Project. Semi-automatic mapping allows, using existing test suites, to support certification and may be the only possibility when tests are developed and made available independently with respect to source code. In the fullness of time, our approach will support a mechanism based on semantics-aware test descriptions for automatically associating security functions to existing test suites.

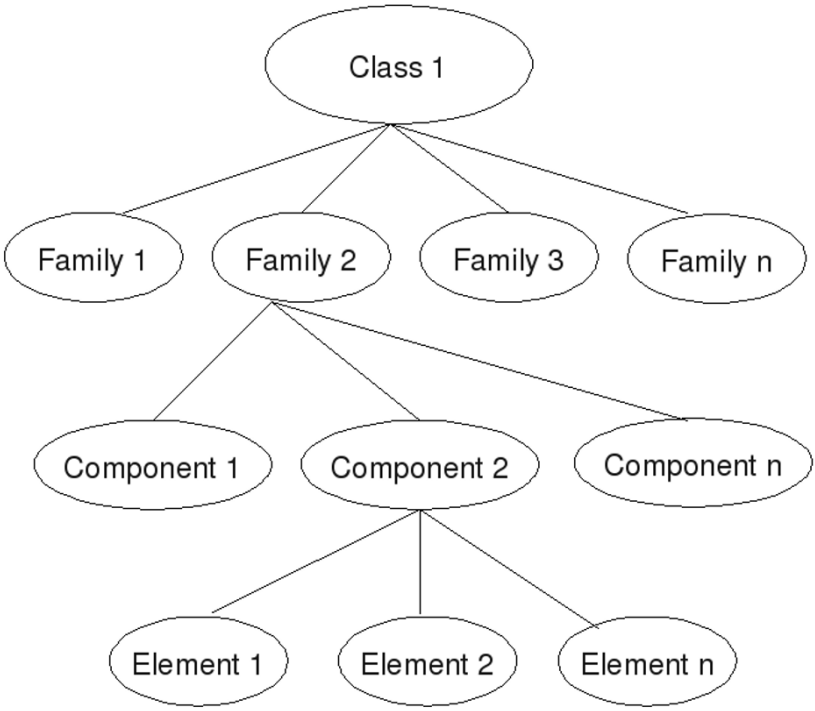

Fig. 2.1 The hierarchical structure of the SFRs.

2 Structure of Common Criteria Certification

The CC certification is based on different components that fall in the following categories [8].

Security Functional and Assurance Requirements.

The CC certification defines two types of security requirements: *Security Functional Requirements* (SFRs) and *Security Assurance Requirement* (SARs). SFRs define the requirements the security functions of the product under evaluation should satisfy [8].The CC standard includes a predefined extendable catalogue of security functional requirements "that are known and agreed to be of value by the CC part 2 authors" [6].

SFRs are organized in a hierarchical structure, as depicted in Fig. 2.1 [9].
SARs describe practical ways to check the effectiveness of the security functions of the product under evaluation [18]. The SARs catalogue includes many predefined requirements focusing on different phases of the product life cycle such as development, configuration management, testing and so forth.

Protection Profile and Security Target.

A Protection Profile (PP) is a document produced by consumers groups and communities that describes its authors' security needs.
A PP is not meant to be referred to a specific product (i.e., the Target of Evaluation (TOE)), but rather to a product type (i.e., a TOE category). PP defines an implementation-independent set of IT security requirements for a category of TOEs, to be used as a starting point for the evaluation of a particular IT product.
Users can create their PP based on their high-level security needs, without looking at any implementation detail [9]. A Security Target (ST) instead contains the security requirements of a given IT product, to be achieved by a set of specific security functions.
The ST is a basis for agreement between the developers, evaluators and, where appropriate, users on the TOE security properties and on the scope of the evaluation.
A ST can be derived from a given PP by instantiation; in general, the construction of a ST corresponds to a particular PP definition. A ST may then claim conformance to a PP by providing the implementation details concerning the security requirements defined by that PP [12]. Also, ST may augment the requirements derived from the PP.
There might be cases where there is no PP that matches the security properties of a specific product. In this case, the product developer can still create its own ST without claiming conformance to any PP [12].
An important aspect about ST requirements specification is the definition of the *threats* and *security objectives* of the TOE and its environment. In particular, threats identify situations that could compromise the system assets, while *security objectives* contain all the statements about the intents to counter identified threats and/or satisfy identified organization security policies and assumptions. Based on threats and security objectives, the ST defines the *security requirements* that the TOE security functions need to satisfy to achieve the security objectives.

Evaluation Assurance Levels.

The Common Criteria standard defines seven hierarchical Evaluation Assurance Levels (EAL), which balance the desired level of security and the cost of deploying the corresponding degree of assurance [3]. EALs identify different sets of security assurance requirements, which are shown in Table 2.1. In case the predefined requirements do not match the level of assurance required, the EAL might be augmented by adding additional assurance requirements.

Table 2.1 Evaluation Assurance Levels.

EAL	Description
EAL1	Functionally tested (black box testing)
EAL2	Structurally tested
EAL3	Methodologically tested and checked
EAL4	Methodologically designed, tested and reviewed
EAL5	Semiformally designed and tested
EAL6	Semiformally verified design and tested
EAL7	Formally verified design and tested

3 The Linux Security Target

As mentioned earlier, a PP defines a security target template directed to particular application domains, describing the security needs of consumers. This potentially results in different PPs listing different requirements for the same type of products. For instance, two groups of users of the same operating system may have different security needs. The first group could require the operating system to provide secure authentication and authorization mechanisms, while the second group could require the operating system to include a secure data transmission mechanism. These two groups of consumers would construct or cite two different PPs for the same product type, based on their expectations.

In this paper, we focus on the certification of Suse Linux Enterprise Server V8 distribution with service pack 3 (SLES8). The ST adopted for SLES8 claims conformance to the Controlled Access Protection Profile (CAPP), which has been released by the Information Systems Security Organization (ISSO) as part of its program to promote security standards for information systems [17]. According to the ISSO "CAPP-conformant products support access controls that are capable of enforcing access limitations on individual users and data objects CAPP-conformant products also provide an audit capability which records the security-relevant events which occur within the system" [17].

In the following of this section, we discuss the SLES8 ST, describing its main components and giving some examples as reported in the SLES8 ST.

Threats, Objectives, and Security Requirements.

The threats part of a ST identifies the threats that could compromise the system assets. In the case of SLES8, the assets to be protected include the information stored, processed or transmitted by the TOE (SLES8) [11]. The threats are generally divided in two categories: *i)* threats to be countered by the TOE, which exploit weaknesses in the TOE itself, and *ii)* threats to be countered by the TOE environment, which exploit the weaknesses of the TOE environment.

Examples of threats to be countered by the TOE and by its environment are, respectively [11]:

- an authorized user of the TOE may access information resources without having permission from the person who owns, or is responsible for the information resource for the type of access;
- an attacker with legitimate physical access to the hardware of the TOE may cause a hardware malfunction with the effect that a user is losing stored data due to this hardware malfunction.

Security objectives provide statements about the intents to address and counter the identified threats [9]. Based on the above examples, the following security objectives for the TOE and TOE environment, respectively, can be defined [11]:

- the TOE security function must control access to resources based on the identity of users, it must also allow the authorized users to specify which resources may be accessed by which users;
- those responsible for the TOE must ensure that those parts of the TOE critical to security policy are protected from physical attack.

Once the threats have been identified and the objectives to counter them have been set, the ST specifies how those objectives will be achieved by defining the security requirements. Security requirements section provides all the details concerning each of the SFRs selected to reach the security objectives. An example of a SFR provided by the SLES8 ST is: "the TOE security function shall be able to include or exclude auditable events from the set of audited events based on the following attributes [11]: *i)* User identity, *ii)* System call number, and *iii)* Directory or file name."
The SLES8 ST includes seven security functions to be evaluated with respect to SFRs: *identification and authentication*, *audit*, *discretionary access control*, *objective reuse*, *security management*, *security communication*, and *TOE security function protection* [11].

4 Associating Tests to Target

One of the main responsibilities of CC security evaluation teams is to ensure that the security functions claimed by developers have been implemented and are being correctly enforced. To this aim, security development and evaluation teams execute a set of tests to check each security function of the TOE. However, to be able to carry out this task successfully, security teams need to know in advance which tests will be used for which security functions. In other words, a mapping between security functions and test cases is necessary. When products are developed from scratch following the CC standard, developers can create a mapping between the security functions

and their test cases during the test phase of the product life cycle. For existing products, however, security development and evaluation teams need to find out which existing tests match the desired security functions.
This problem is hard in general, as test documentation can be terse or entirely lacking, and is even harder in case of OSS evaluation. OSS development activities are not tracked, configuration management is skeletal and in many cases developers of the system functionalities do not write test cases. Therefore, finding the right mappings becomes a time-consuming task.
Our approach is aimed at providing a general automatic solution, based on matching techniques, for associating ST's security functions to tests. In the following, we show how our solution is used to map SLES8 security functions to Linux Test Project existing test suites. We relied on an extended set of test suites that has been used for SLES8 EAL3 certification [11].

4.1 The Linux Test Project

A fundamental aspect of CC certification is represented by extensive testing of the security functions of the TOE, using tests which cover all the range of security functions under evaluation.
The Linux Test Project (LTP) [15] defines a set of tests for Linux with a huge amount of security functions-related tests, which simplify the certification of Linux kernels.
LTP, which started in 2000 with one hundred test programs developed by Silicon Graphics Inc. (SGI), has been developed to improve the Linux kernel by providing automated testing of kernel functionalities [10]. LTP represented a “revolution” in Linux testing and assurance, since no formal testing environment was previously available to developers. The major advantage provided by the development of LTP was not the definition of batteries of tests, but the provisioning of a framework for systematic integration of testing activities. Furthermore, LTP includes an environment for defining new tests, integrating existing benchmarks and analyzing test results.
Nowadays, the latest version of LTP contains more than three thousands tests. In addition to the test suites, LTP also provides test results, a test tools matrix, technical papers and *HowTos* on Linux testing, and a code coverage analysis tool [15].
LTP tests cover a wide range of kernel functions, including system calls, networking and file system functionalities, and their results are restricted to *PASS* or *FAIL*. LTP provides a testing environment simple to install and use, which includes three scripts for executing three different suites of automatic tests [14]: *i) runalltests.sh, ii) network.sh, and iii) diskio.sh*. To give a clear understanding of a LTP test suite evaluation, in Table 4.1 we provide the summary report of the evaluation of Linux kernel 2.6 on a 64 bit Intel Itanium architecture (formerly called IA-64)[1].

[1] The complete evaluation is available at http://surfnet.dl.sourceforge.net/sourceforge/ltp/ltp-full-20071130_ia64_output.html.

Table 4.1 LTP summary report of Linux kernel evaluation.

Test Summary	Pan reported some tests FAIL
LTP Version	LTP-20071130
Start Time	Tue Dec 4 02:11:29 PST 2007
End Time	Tue Dec 4 03:11:52 PST 2007
Log Result	/root/subrata/ltp/ltp-full-20071130/results
Output/Failed Result	/root/subrata/ltp/ltp-full-20071130/output
Total Tests	849
Total Failures	0
Kernel Version	2.6.16.21-0.8-default
Machine Architecture	ia64
Hostname	elm3b159

In our context, LTP is used to evaluate the security related functionalities of Linux kernels with respect to CC certification.
As an example, starting from LTP, test suites for the CAPP/EAL3+ certification of SuSE Linux Enterprise Server 8 has been implemented [11]; test cases have been written by IBM, SuSE, and atsec, or taken directly from the LTP.
A detailed test plan has been produced to test the functions of SLES8 on each evaluated platform, and such plan includes an analysis of the test coverage, the functional interfaces tested, and the testing against the high level design.

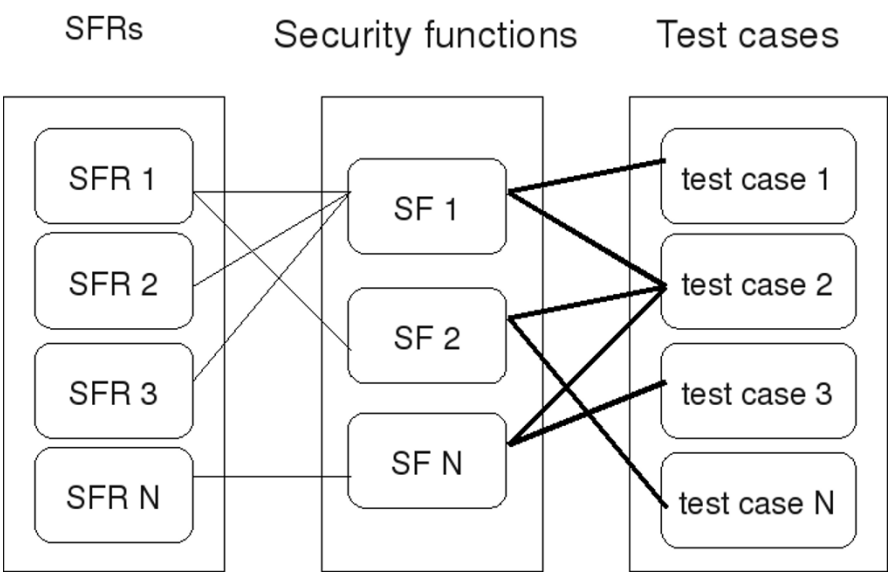

Fig. 4.1 Mapping between SFRs, security functions and testcases.

LTP has been used to test more than fifty kernel versions, where more than five hundreds vulnerabilities have been found. Nevertheless, LTP lacks of adaptability from a security certification point of view, that is, it lacks of an effective technique for associating batteries of tests to specific security functions to be evaluated. LTP, in fact, makes available only three pre-configured scripts for a general purpose testing, making difficult for developers to intuitively identify the batteries of tests that should be adopted for testing specific functions. In the remainder of this section, we discuss a possible solution to this problem.

4.2 Automatic Test Selection for Security Functions

Security functions testing, which is aimed at confirming that the functions operate according to their specification and satisfy the SFRs, is an important part of Common Criteria certification.
However, a major problem to be faced in security functions testing is that no framework for automatic linking between test suites and security functions is available. Developers and evaluators are then forced to associate tests to security functions basing on their experience only. To the best of our knowledge, in the past, this task has been done manually by creating two types of mappings. The first mapping connects each SFR with the security function or set of security functions that implement it. For instance, considering SLES8 security target, the security function *User Identity Changing (IA.4)* contributes to satisfy the SFR *User-Subject Binding (FIA_USB.1)* [11]. The second mapping is done between the security functions and available testcases. Figure 4.1 depicts the two mappings. Note that, whereas the first mapping is

part of the TOE security specification section of the ST, the second one is not part of the ST [5].
Our solution focuses on the second mapping and consists of a generic framework for the automatic selection of test suites for the evaluation of security functions. Our framework, depicted in Figure 4.2, is composed of three main components:

- *Certification Test Suites*, which represents a hierarchical structure containing the set of test suites to be used in the certification process. Each node of the structure is assumed to be labelled using a self-explaining name and enriched of metadata describing its semantics. Each leaf-node represents a single test;
- *Security Functions*, which include the functions description of the TOE to be evaluated;
- *Matching Engine*, which is the component responsible for associating tests to security functions.

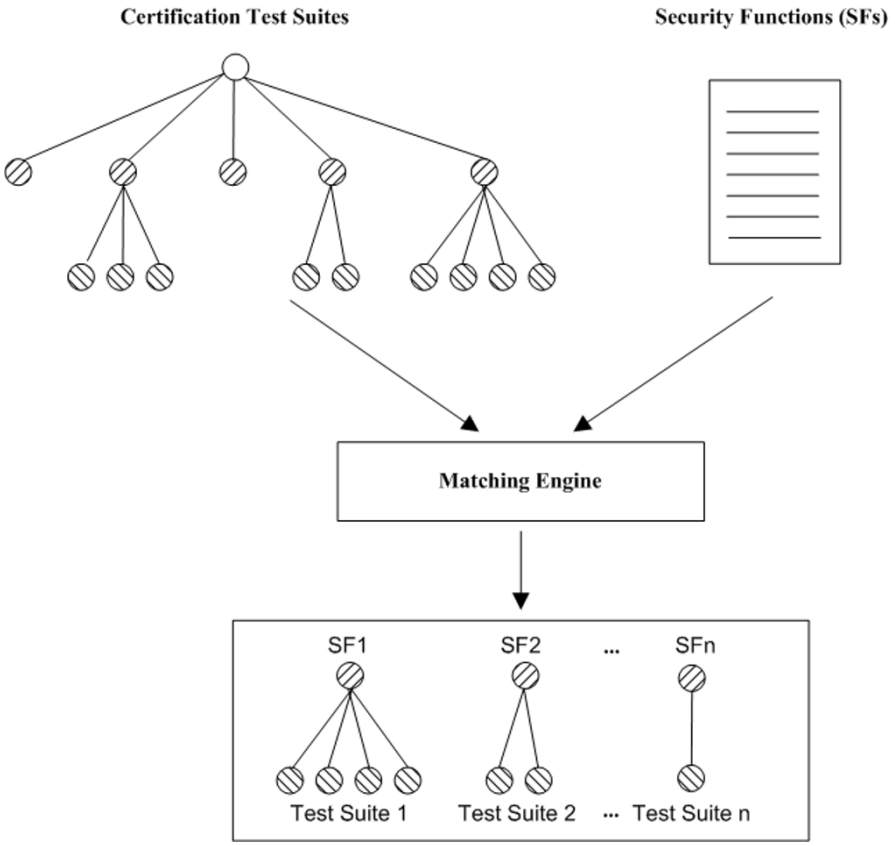

Fig. 4.2 Matching engine.

The first step towards the development of a matching engine for automatic test selection is the introduction of a common vocabulary. A vocabulary can be defined as a set of keywords $\{k_1,\ldots,k_n\}$ to be used in the definition of test suites metadata and name, and for the description of the security functions to be evaluated.

The vocabulary V allows to define both certification test suites and security functions descriptions based on a common grammar, which simplify the matching engine process. Our matching engine can be defined as a function f that takes in input the vocabulary *V*, the security functions *SF* and the test suites *TS*, and returns the association between the security functions *SF* and the test suites *TS*. The mapping process is composed of three phases: *keyword extraction*, *semantic expansion*, and *search/match*.

In the first phase, given a security function $sf \in SF$, the matching engine searches inside *sf* all keywords $\{k_1,\ldots,k_n\} \in V$ extracting a set $K \subseteq V$ of all the keywords used in *sf* description. Then, each keyword in $k_i \in K$ is expanded by means of a suitable thesaurus, and the expanded set of keywords is searched in the test suites paths and in each node metadata. All nodes that contain at least one matching, i.e., where at least one keyword is found, are selected and used by the matching engine as the roots of

the subtrees to be considered. Finally, the matching engine returns all the selected subtrees, which are used to test security function *sf*.
Although general enough to be adopted in a generic scenario, our solution is used in the context of the SLES8 Linux distribution. In particular, our framework relies on an extended version of Linux Test Project, which has been used for SLES8 EAL3 certification [11], and on the security functions specification used for the definition of the security target of SLES8.
Since no formal metadata describing the test suites for SLES8 EAL3 certification are available and no agreement between security functions and test suites vocabulary is in place for this application, we assume that the vocabulary used by our matching engine is composed by the set of node names used in the hierarchical structure of the certification test suites, together with some predefined keywords. For instance, the certification test suites used in SLES8 EAL3 certification contains the node path *network/nsfv4/acl* that includes a large set of tests among which: *create_users*, *test_acl*, and many others. All the node names in the path and the test names are used as keywords for searching inside the security functions description and finding the association security functions/test suites.
In the following section, we present a worked-out example based on SLES8 test suites and security target. In particular, after selecting three security functions, we used our engine to select the set of tests to be used for the certification process.

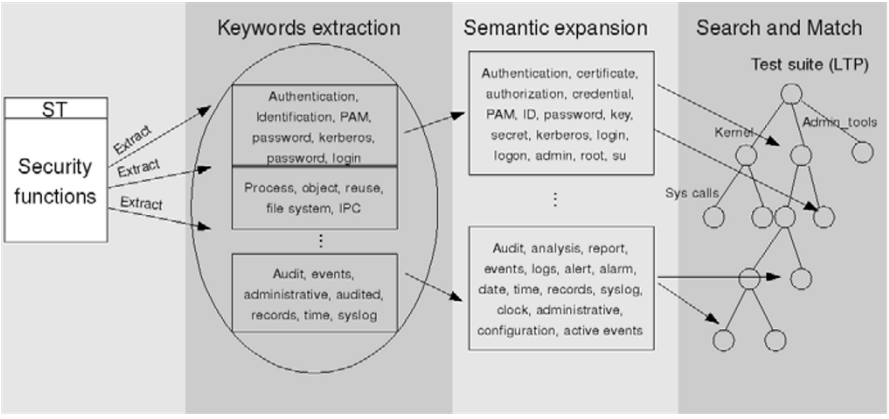

Fig. 4.3 The automated mapping approach between security functions and testcases.

4.3 Automatic Test Selection Example Based on SLES8 Security Functions

The SLES8-based example is summarized in Fig. 4.3 and the results are provided in Table 4.2. During the keyword extraction phase, all the security-related vocabulary keywords are extracted by security functions description (see underlined words in second column of Table 4.2. In the second phase, keywords are semantically expanded using an *ad-hoc* security related thesaurus, generating the column tree of Ta-

ble 4.2. Finally, in the search/match phase, the expanded set of keywords is searched in the LTP testcases directory hierarchy[2]. If a node matches at least one keyword, it will be automatically mapped to the security function to which that keyword belongs (see column four in Table 4.2).
In our approach, the search/match phase relies on the Linux command *grep*, which is applied to the extended LTP tree structure.
The *grep* command, developed within the GNU project, searches one or more input strings and returns any line containing a match to a specified pattern. Based on *grep* command, scripts are generated automatically starting from the keywords retrieved in the semantic expansion phase. These scripts search the keywords retrieved during the keyword extraction and semantic expansion phases, inside the directory structure of extended LTP tree and inside the comments in the testcases. Figure 4.4 presents a sample script, which is executed to retrieve the set of tests for the SC.1 security function described in Table 4.2[3].

```
grep -l -i -e 'tunneling' -e 'port 22'
           -e 'secure channel' -e 'secure socket layer'
           -e 'ssl' -e 'ssh'  -e 'secure shell'
           -r linux_security_test_suite_EAL3
```

Fig. 4.4 Example of search/match script applied to the SC.1 security function. The -l and -i options allow to print the list of matching files ignoring case distinctions.

Our experimental results prove the suitability of our solution, since our matching engine always returns all the relevant test cases selected manually by developers during SLES8 certification. Of course, our matching is not entirely precise, and some redundant tests are also selected.

Table 4.2 Automatic test selection results. For sake of conciseness, the table only reports a significant subpart of each security function and a partial list of tests. A complete description of the functions can be found in [11].

Security Function	Description	Semantic Expansion	Tests
Access Control Lists supported by SLES (DA.3)	SLES provides support for POSIX type ACLs for the ext3 file system allowing defining a fine grained access control on a user basis.	permission, allowed, entry, discretionary, DAC, etc.	access04.c, signal01.c, fcntl07B.c, readdir02.c, etc.
Kernel Modules (TP.3)	SLES supports dynamically loadable kernel modules that are loaded automatically on demand. Kernel modules are actually a part of the kernel that is not resi-	Kernel mode, loadable module, modules, trusted program, modprobe etc.	create_module01.c, create_module02.c, delete_module01.c, delete_module02.c, delete_module03.c,

[2] Search process considers also the comments inside the testcases files.

[3] The complete output produced by the execution of the *grep* command is summarized in Appendix A.

Security Function	Description	Semantic Expansion	Tests
	dent but loaded as part of the kernel when needed.		modules.conf01, modules.conf02, etc.
Secure Protocols (SC.1)	The TOE offers two protocols that applications can use to securely communicate with another trusted IT product (…). Those protocols are the Secure Shell Protocol Version 2 (SSH v2) and the Secure Socket Layer Protocol Version 3 SSL v3).	network, secure channel, data transmission, port 22, tunnelling, cryptographic protocol, secure communication, etc.	ssh01, ssh02, ssh03, ssh04, etc.

5 Conclusions and Outlook

In this paper, we outlined a framework aimed at supporting OSS security certifications including a matching engine for the automatic association of tests to security functions. Our solution is a first attempt to provide such an automatic infrastructure, and much additional work remains to be done. In particular, the proposed methodology has the main disadvantage of selecting a redundant set of tests. Our engine in fact chooses all the subtrees of the nodes whose names and/or description match with at least one of the keywords contained in the security functions under evaluation. This results in a situation where unnecessary tests are executed. To overcome this problem, we plan to provide two major improvements in our future work: *i)* the definition of a complete and formal vocabulary, and *ii)* the definition of a formal ontology used to reason about metadata describing the elements to be chosen (i.e., the nodes of the certification test suites hierarchy) and the correlations among them. Such an approach will allow us to exploit automatic reasoning techniques. Also, the combination of a complete vocabulary and a detailed ontology will lead to the development of an engine able to select only the suites of tests that effectively match the specified security functions, thus limiting unnecessary tests executions.

Another important issue is the actual testing environment to be used. It is well known that the development and running of intensive tests, especially for networking applications/devices, could affect the overall consistence of the system under evaluation and of the network that hosts the system. Furthermore, network tests need the participation of at least two physical machines, making them more expensive and demanding more implementation efforts. Also the risk of compromising the integrity of the real system becomes considerable. To eliminate the risk and to reduce the efforts introduced by such an extensive testing, a possible solution consists in setting up a *Virtual Test Environment* (VTE) that accurately reproduces the real environment in which the target application should be evaluated.

Several VTEs have been already provided in literature in different contexts and aimed at several goals [1,2,4,7]. Among them, we have provided a multi-purpose virtual environments, which supplies Virtual Laboratories for the on-line degree courses on Information Systems and Network Security of University of Milan [1,2]. This in-

frastructure also allows the use of the virtual systems for general testing and, in particular, for network functions testing. More in details, the system in [1,2] consists of a virtual subnet composed of a three hosts acting as a firewall, an external generic host, and an internal host working under the firewall. In our future work, we plan to adapt such a virtual network to make it suitable for security testing of applications that should be evaluated to achieve security certification.

Acknowledgments This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591, and within the SecureSCM project in the FP7-ICT Programme under contract n.AOR 213531 and by contract/grant sponsor FIRB research fund of MIUR, research project TEKNE (contract/grant n.RBNE05FKZ2).

6 References

1. Anisetti M, Bellandi V, Colombo A, Cremonini M, Damiani E, Frati F, Hounsou JT, Rebeccani D (2007) Learning computer networking on open paravirtual laboratories. IEEE Transactions on Education, 50(4):302-311
2. Anisetti M, Bellandi V, Damiani E, Frati F, Raimondi U, Rebeccani D (2006) The open source virtual lab: a case study. In Proc. the Workshop on Free and Open Source Learning Environments and Tools - FOSLET 2006, Como, Italy
3. Caplan K, Sanders JL (1999) Building an international security standard. IEEE Educational Activities Department, 22(3):29-34
4. Gambit Communications. Mimic virtual lab CCNA (2007) www.gambitcomm.com/site/products/vlab_ccna.shtml. Accessed December 2007.
5. The International Organization for Standardization and the International Electrotechnical Commission (2007) Common Criteria for Information Technology Security Evaluation, Evaluation methodology
6. The International Organization for Standardization and the International Electrotechnical Commission (2007) Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components
7. Grigoriadou M, Kanidis E, Gogoulou A (2006) A web-based educational environment for teaching the computer cache memory. IEEE Transactions on Education, 49(1):147-156
8. R. Harland R (2007) The canadian common criteria scheme. http://strategis.ic.gc.ca/epic/site/ad-ad.nsf/vwapj/CSE_Harland.ppt
9. Herrmann DS (2002) Using the Common Criteria for IT security evaluation. Auerbach publications, London
10. Hinds N (2004) Kernel korner: The Linux test project. Linux Journal
11. IBM, SuSE, atsec (2007) SuSE Linux Enterprise Server 8 w/SP3 CAPP/EAL 3+ Certification Test Suite. http://ltp.sourceforge.net/EAL3.html. Accessed December 2007
12. ISO/IEC. Guide for the production of Protection Pro_les and Security Targets, 2004.
13. Katzke S (2007) The common criteria (cc) paradigm. ieeeia.org/iasw/Katzke-CC.ppt. Accessed December 2007
14. Larson P (2002) Testing linux with the linux test project. In Proc. of the Ottawa Linux Symposium, Ottawa, Ontario, Canada
15. Linux Test Project (2007) http://ltp.sourceforge.net. Accessed December 2007
16. Lipner S (1999) Twenty years of evaluation criteria and commercial technology. In Proc. of the 1999 IEEE Symposium on Security and Privacy
17. Information Systems Security Organization (1999). Controlled Access Protection Profile version 1.d

18. Shankar KS, Kirch O, Ratliff E (2004) Achieving capp/eal3+ security certification for Linux. In Proc. of the Linux Symposium, volume 2:18-30
19. Shankar KS, Kurth H (2004) Certifying open source - The linux experience. IEEE Security & Privacy, 2(6):28-33
20. Shi W, Sun Y (2001) An investigation of cc's contribution to confidence in security. In Proc. of the 2001 International Conference on Computer Networks and Mobile Computing, 333-338

Appendix A

An example of a grep-based search/match phase

In Fig. A.1 we provide the complete output produced by our matching engine when mapping between security function SC.1 and testcases is searched. Based on the following grep-based script, where Fig. A.2 shows the set of testcases to be used in the evaluation of security function SC.1.

```
grep -l -i -e 'tunneling' -e 'port 22' -e 'secure channel'
           -e 'secure socket layer' -e 'ssl' -e 'ssh'
           -e 'secure shell' -r linux_security_test_suite_EAL3

linux_security_test_suite_EAL3/ltp_EAL2/testcases/network/tcp_cmds/ssh/ssh04
linux_security_test_suite_EAL3/ltp_EAL2/testcases/network/tcp_cmds/ssh/ssh01_s1
linux_security_test_suite_EAL3/ltp_EAL2/testcases/network/tcp_cmds/ssh/ssh03
linux_security_test_suite_EAL3/ltp_EAL2/testcases/network/tcp_cmds/ssh/ssh03_s1
linux_security_test_suite_EAL3/ltp_EAL2/testcases/network/tcp_cmds/ssh/ssh02_s1
linux_security_test_suite_EAL3/ltp_EAL2/testcases/network/tcp_cmds/ssh/ssh02
linux_security_test_suite_EAL3/ltp_EAL2/testcases/network/tcp_cmds/ssh/ssh01
linux_security_test_suite_EAL3/ltp_EAL2/testcases/user_databases/lastlog01
linux_security_test_suite_EAL3/ltp_EAL2/testcases/user_databases/faillog01
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/send_new_rsa_key.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/restore_date.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/rsa_login.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/rsa_test.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/dsatest.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/aestest.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/delete_accounts.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/sha1test.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/setup_date.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/rsa_auth.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/server.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/setup_accounts.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/rc4test.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/dsa_auth.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/stunnel_dsa.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/dhtest.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/password_auth.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/send_new_dsa_key.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/stunnel_rsa.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/dsa_login.sh
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/client.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/destest.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/test/randtest.c
linux_security_test_suite_EAL3/ltp_OpenSSL/testcases/openssl/openssl01
linux_security_test_suite_EAL3/laus_test/audit_tools/ssh01_s1
linux_security_test_suite_EAL3/laus_test/audit_tools/au_login
linux_security_test_suite_EAL3/laus_test/audit_tools/ssh01
linux_security_test_suite_EAL3/laus_test/libpam/tests/test_sshd.c
linux_security_test_suite_EAL3/laus_test/libpam/libpam.c
linux_security_test_suite_EAL3/laus_test/pam_laus/pam_laus.c
linux_security_test_suite_EAL3/laus_test/pam_laus/tests/test_sshd.c
```

Fig. A.1 Grep-based script for keyword matching and set of testcases retrieved for SC.1 security function evaluation.