Misuse for Security Hardening Assessment in Application Software Deployment

Kwanchanok Limbandit and Yunyong Teng-Amnuay

***Abstract*—In software installation, the hardening status of the target system is difficult to assessed and usually depends on the expertise and care of system administrator. These non-functional requirements can be rendered functional by using misuses in misuse case diagram. This allows the assessment to be incorporated into the software design process and implemented as part of the deployment module. The assessment can thus be carried out automatically during software installation. As system hardening is mostly independent from software functionalities, the assessment can be expressed as design patterns to accommodate the design process. Four examples of system hardening were used: program and data memory separation, Mandatory Access Control (MAC), firewall, and logging.**

***Index Terms*—Misuse, hardening, installation, security pattern.**

I. Introduction

A key issue in system security is verifying the target system, the system where application will install [1]. This is non-functional and is usually taken care by system administrator with varying degree of expertise [2].

This research employed misuses, in use case diagram, to change these non-functional requirements to functional ones. We focused on assessment of hardening of the target system. The use of misuse will ensure that the assessment will happen automatically during software deployment without relying on system administrator.

This work analyzed four cases of hardening as examples: separation of program and data memory, limiting user privileges, use of firewall, and recording accesses to system. We also employed security patterns so it will be easy to apply functional requirement as hardening assessment during the installation process.

In section 2 we present theories and related works. Section 3 is the research methodology. In section 4 we present four security patterns corresponding to the four hardening examples. In section 5 we test the implementation of the security patterns in a system hardening assessment module. And the last section in section 6 we summarize results of this research and suggest future work.

Manuscript received April 07, 2012; revised May 02, 2012.

The authors are with Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand (Corresponding author. Tel.: +66-22186991, fax: +66-22186955, e-mail: kwanchanok.l@student.chula.ac.th, 2 yunyong.t@chula.ac.th).

II. Theories and Related Works

This section consists of three parts: hardening, misuse case diagram, and software deployment.

A. Hardening

Hardening increases security by reducing the attack surface [3] and has the following steps [4]:

Reduce attack surface by getting rid of unnecessary software, users, services, and network ports.

Update patches to eliminate system vulnerabilities.

Install Intrusion Detection System (IDS), Firewall, and Intrusion Prevention System (IPS).

Examples of hardening for default installation of Linux [5] include things like encrypting transmitted data up to disabling Internet Protocol version (IPv6) when not required.

In related works, Linux still has much weakness so many researches focused on the development of a more secure Linux. Loscocco and Smalley [6] proposed MAC by using Security-Enhanced Linux (SELinux) and de Ven [7] presented Exec Shield to separate the reading and writing memories. Fox [8] proposed method for preventing intruders by using SELinux, Exec Shield, and iptables.

B. Misuse Case Diagram

Misuse case diagram is an enhanced use case diagram for software development. It describes actions that are inappropriate, and consists of four parts [9]: misuses, misusers, relationships, and descriptions.

In related works, the constraints on security are usually considered non-functional requirements. By using misuses software designer can explicitly express inappropriate actions and attacks in order to prevent them in advance. Sindre and Opdahl [10] proposed a format of misuse case description. Matulevicius, Mayer, and Heymans [11] presented misuse case with security risk management. Braz, Fernandez, and VanHilst [12] proposed a collection of security requirements by using misuses. Okubu, Taguchi, and Yoshioka [13] proposed misuse case diagram for analyzing and requirements gathering that focus on resources and security.

III. Research Methodology

This section consists of three parts on system hardening assessment: conceptual model, analysis of attacks and preventions, and misuse cases.

A. Conceptual Model

In Fig. 1, we studied hardening assessments which are non-functional and then employed misuse to express them as functional requirements. As the hardening assessments are rather independent from normal functions of application we can build security patterns for them. We focused on assessing hardening status of the target system during software deployment.

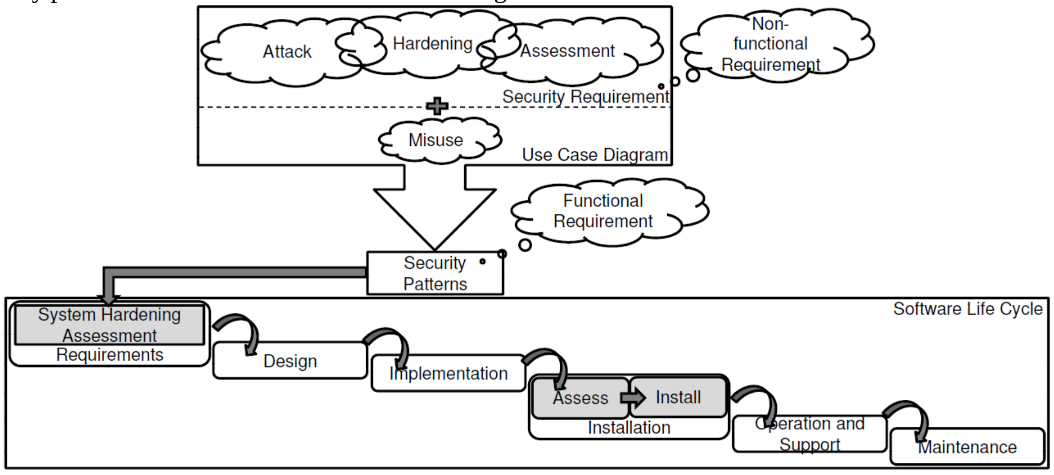


Fig. 1. Conceptual model.

B. Attacks and Preventions

We have to determine attacks, how to attack, and then how to prevent attacks.

The basic idea is that a malicious action is still an action. So we can consider non-functional attacks as misuses and convert malicious activities into definite actions or functional requirements.

We used the following four hardening activities as examples for analysis:

Table I. Separation of program and data memory.

Table II. Limiting user privileges.

Table III. Use of firewall.

Table IV. Recording accesses to system.

We also specify how to assess each hardening activity. This is shown in Table 1.

TABLE I: SYSTEM HARDENING ASSESSMENT.

Attack	Misuse	Protection	Assessment
Memory	Mix program memory with data memory	Execute restriction	Exec Shield
Access control	Use Discretionary Access Control (DAC)	Use MAC	SELinux
Firewall	Inject malicious traffic	Configure firewall	iptables
Log	Undetected break-in	Keep log	rsyslog

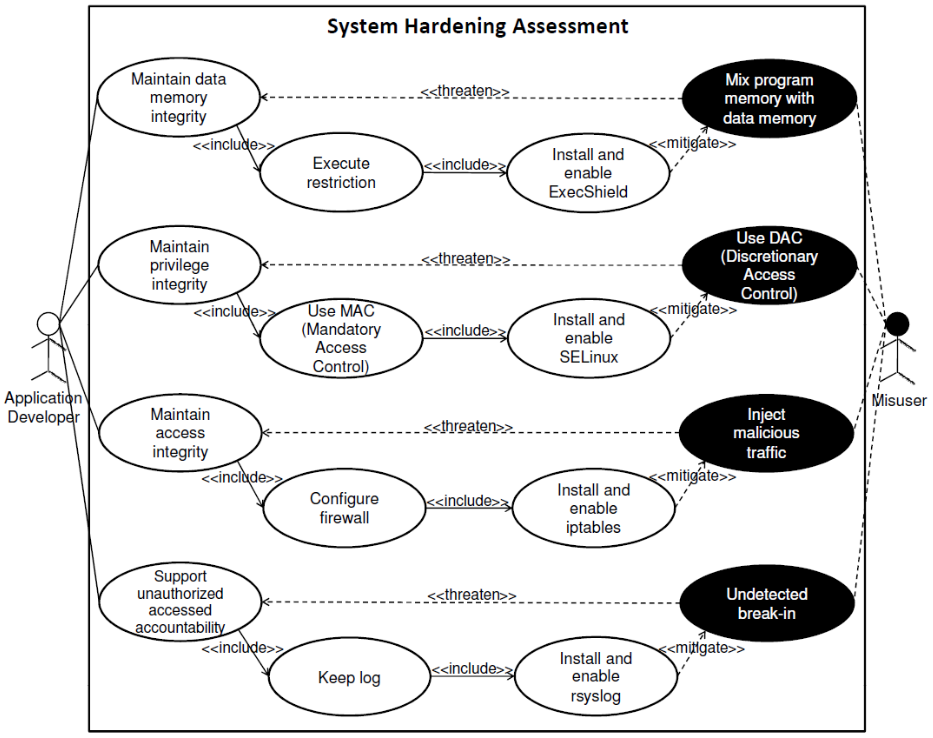

System Hardening Assessment
Maintain data memory integrity
<<threaten>>
Mix program memory with data memory
<<include>>
Execute restriction
<<include>>
Install and enable ExecShield
<<mitigate>>
Maintain privilege integrity
<<threaten>>
Use DAC (Discretionary Access Control)
<<include>>
Use MAC (Mandatory Access Control)
<<include>>
Install and enable SELinux
<<mitigate>>
Application Developer
Misuser
Maintain access integrity
<<threaten>>
Inject malicious traffic
<<include>>
Configure firewall
<<include>>
Install and enable iptables
<<mitigate>>
Support unauthorized accessed accountability
<<threaten>>
Undetected break-in
<<include>>
Keep log
<<include>>
Install and enable rsyslog
<<mitigate>>

Fig. 2. Misuse case diagram for system hardening assessment.

From Table 1 an attack on memory can be translated into a misuse of mixing program memory with data memory. From this misuse, it is straightforward to find protection mechanism which in this case is execution restriction using Exec Shield. So, assessment is to make sure that the target system has Exec Shield installed and enabled. The other three attacks also result in the three assessments: SELinux, iptables, and rsyslog.

C. Misuse Case Diagram

We analyzed and converted Table 1 to misuse case diagram in Fig. 2. An example of the analysis is as follows. A misuser is an actor who misuses the system via "Mix program memory with data memory" misuse case. It can threaten "Maintain data memory integrity" use case, which is a functional requirement about memory protection. Therefore an application developer needs to include "Execution restriction" use case and "Install and enable ExecShield" use case respectively in order to mitigate the chance that a misuse case will complete successfully.

IV. Security Patterns and Integration Scheme

As the system hardening assessments are normally not dependent on functionalities of application we can represent the assessments as security patterns: separation of program and data memory, limiting user privileges, configuring firewall, and recording accesses. For brevity we present only the first pattern as follows.

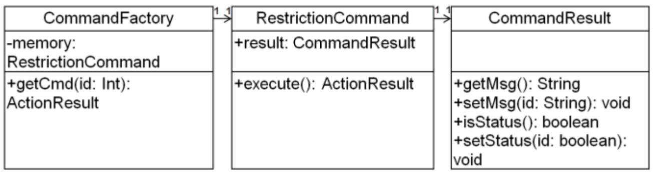


Fig. 3. Class diagram for memory assessment.

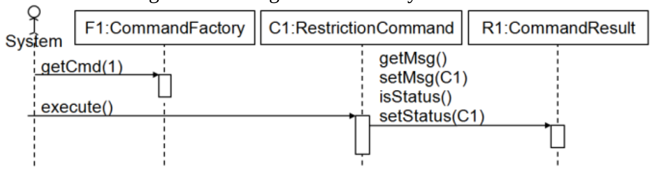


Fig. 4. Sequence diagram for memory assessment.

A. Pattern for Separation of Program and Data Memory

Context: Determine whether the system is protected from execution of malicious codes.

Problem: Attacker can inject code through misuse as shown in upper part of Fig. 2.

Solution: Verify that Exec Shield is installed and enabled.

Structure: Convert upper part of Fig. 2 to class diagram in Fig. 3.

Dynamics: Use command: cat /proc/sys/kernel/exec-shield

If result = 1, then separation of program and data memory is in effect, or active.

Implementation: Perform during installation by assessing the system. Convert Fig. 3 to sequence diagram in Fig. 4.

Example Resolved**:** The existence of Exec Shield verifies a separation of program and data memory.

Variants: Linux only.

Known Uses: Assess that Exec Shield is installed and enabled.

Consequences: Prevent threats.

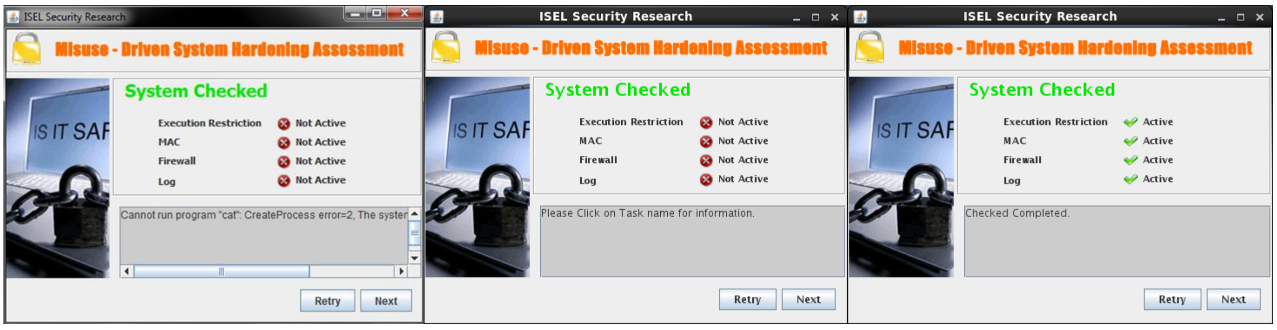
V. Testing Assessment Module

After the security patterns were defined it is rather straight forward to implement the assessment module from the patterns. To test the module we used three test cases: system is not hardened, hardening is installed but disabled, and hardening is installed and enabled. This is shown in Table 2 and results are in Fig. 5 where in Case A Hardening is not installed so it is not active, Case B Hardening is not enabled so it is not active, and Case C Hardening is enabled so it is active.

TABLE II: Condition for Testing.

Case	Installed				Enabled			
	Exec Shield	SELinux	iptables	rsyslog	Exec Shield	SELinux	iptables	rsyslog
A	X	X	X	X	X	X	X	X
B	/	/	/	/	X	X	X	X
C	/	/	/	/	/	/	/	/

Remark: / means hardening is installed or enabled and X means hardening is not installed or is disabled.



System Checked

Execution Restriction ✗ Not Active
MAC ✗ Not Active
Firewall ✗ Not Active
Log ✗ Not Active

Cannot run program "cat": CreateProcess error=2, The syster

System Checked

Execution Restriction ✗ Not Active
MAC ✗ Not Active
Firewall ✗ Not Active
Log ✗ Not Active

Please Click on Task name for information.

System Checked

Execution Restriction ✓ Active
MAC ✓ Active
Firewall ✓ Active
Log ✓ Active

Checked Completed.

Fig. 5. Results from test cases.

VI. Conclusion and Future Work

This work proposes a methodology to apply misuse case diagram in the assessment of system hardening. For the software designer to express non-functional system hardening requirements in term of functional ones, attacks are expressed as misuses, protections as uses, and assessments as remediations. Four cases of hardening were used as examples: separation of program and data memory, limiting user privileges, installation of firewall, and recording accesses to system. This targets the system where the software is to be installed, so the misuses are developed into assessment of the target system and becomes a part of the deployment module of the software. Also security patterns are used to simplify inclusion of hardening assessments into the design. Expressing hardening assessment as functional requirements allows for automate assessment to be carried out during software installation without depending on the system administrator.

Our examples developed into four security patterns. The first pattern on separation of program and data memory uses Exec Shield as the basis for assessment. The second pattern verifies the use of SELinux to enforce MAC and limit user privileges. The third pattern verifies that firewall is installed and activated on the target system. And the last pattern checks on system logging to thwart undetected intrusions.

We give only one example of constructing and using the patterns. The pattern on separation of program and data memory verifies the use of Exec Shield as the solution to the misuse. This is implemented as an assessment module and the result of the test is satisfactory.

The use of misuse for attacks, preventions, and remediations allows automated assessment to be incorporated into the software requirements and design. As the assessment depends more on the type of hardening and the target system than the functionalities of the application, all types of hardening can further be explored and a library of security patterns built to accommodate the inclusion of assessment into secure software.

REFERENCES

[1] C. P. Pfleege. *Security in Computing*. 4th. NJ: Pearson Education, 2007.

[2] J. D. Meier, "Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, and Anandha Murukan," *Improving Web Application Security: Threats and Countermeasures*. [Online]. 2003. Available from : http://msdn.microsoft.com/en-us/library/ff648644.aspx [2011, December 1]

[3] C. Kopp, *Hardening Your Computing Assets*. Computer Magazine Group, 1997.

[4] Syngress. *Hardening the Operation System*. [Online]. 2007. Available from : http://www.syngress.com

[5] Vivek Gite. *20 Linux Server Hardening Security Tips*. [Online]. 2009. http://www.cyberciti.biz/tips/linux-security.html 2011, October 20.

[6] P. A. Loscocco and Stephen D. Smalley. *Meeting Critical Security Objectives with Security-Enhanced Linux*. Ottawa Linux Symposium, 2001.

[7] A. v. d. Ven, *New Security Enhancements in Red Hat Enterprise.* Red Hat Inc. WHP0006US 8/04, 2004.

[8] T. Fox, *Red Hat Enterprise Linux Administration Unleashed*. Sams Indianapolis, 2007.

[9] J. Whittle, D. Wijesekera, and M. Hartong, "Executable Misuse Cases for Modeling Security Concerns," *30th International Conference on Software Engineering* (ICSE'08). pp. 121–130, 2008.

[10] G. Sindre and A. L. Opdahl, "Templates for Misuse Case Description," *7th International Workshop on Requirements Engineering Foundation for Software Quality* (REFSQ'01), 2001.

[11] R. Matulevicius, N. Mayer, and P. Heymans, "Alignment of misuse cases with security risk management," *3rd International Conference on Availability, Reliability and Security* (ARES'08). pp. 1397–1404, 2008.

[12] F. A. Braz, Eduardo B. Fernandez, and M. VanHilst, "Eliciting Security Requirements through Misuse Activities," *19th International Conference on Database and Expert Systems Applications*. pp. 328-333, 2008.

[13] O. Takao, T. Kenji, and Y. Nobukazu, "Misuse Cases + Assets + Security Goals," *International Conference on Computational Science and Engineering*. pp. 129-144, 2009.