Self-Protection Mechanisms for Web Applications
A Case Study

Claudia Raibulet, Alberto Leporati and Andrea Metelli
Department of Informatics, Systems and Communication, University of Milano-Bicocca, Viale Sarca 336, Milan, Italy

Keywords: Self-protection, Security, Adaptive Systems, Home Banking Case Study.

Abstract: Self-protection mechanisms aim to improve security of software systems at runtime. They are able to automatically prevent and/or react to security threats by observing the state of a system and its execution environment, by reasoning on the observed state, and by applying enhanced security strategies appropriate for the current threat. Self-protection mechanisms complement traditional security solutions which are mostly static and focus on the boundaries of a system, missing in this way the overall picture of a system's security. This paper presents several self-protection mechanisms which have been developed in the context of a case study concerning a home banking system. Essentially, the mechanisms described in this paper aim to improve the security of the system in the following two scenarios: users' login and bank operations. Furthermore, the proposed self-protection mechanisms are presented through the taxonomy proposed in (Yuan, 2014).

1 INTRODUCTION

Many modern software systems are delivered in the form of Web applications, due to the many advantages of such a solution in terms of using, updating, and maintaining the applications. Also some applications for mobile devices sometimes use the so called Web views to connect to a Web site and show relevant information to the user, thus essentially behaving like a Web browser. It is thus clear that ensuring a trusted and secure operation of Web applications is of primary importance, and it is even more relevant for Web applications dealing with users' identities or with their money, such as a home banking application.

Traditional security solutions (Anderson, 2008; Peeger, 2006; Stallings, 2013) are mostly static, and protect the system at its boundaries, usually by means of appliances operating at the network level, comprising firewalls, intrusion detection systems, and multi-factor authentication servers. However, these solutions cannot observe the behavior of the application, and thus fail to recognize attacks coming, for example, from attackers that succeed to impersonate legitimate users.

Self-protecting software systems are a class of autonomic systems capable of detecting and mitigating security threats at runtime (Schmerl, 2014; Yuan, 2014; Yuan, 2013). They are able to prevent and/or react to security threats by observing the state of a system and its execution environment, by reasoning on the observed state, and by applying enhanced security strategies appropriate for the current threat. All these steps are performed automatically, without human intervention. Self-protection mechanisms complement the traditional security solutions which operate at the network level, thus allowing to obtain a global protection of the software system.

In this paper we present several self-protection mechanisms which have been developed in the context of a case study concerning a home banking system, named UNIBANK. In particular, the proposed mechanisms aim to improve the security of the system during the access (login) of users to the application, and the execution of bank operations (by authenticated users) involving the movement of money to or from bank accounts.

The proposed self-protection solution follows the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) control feedback loop (Cheng, 2009; de Lemos, 2013), and it is composed of three main parts: sensors (called Monitors), which observe the behavior of the application and of its environment, an analysis and planning module, which identifies security issues and decides what security strategies

Raibulet, C., Leporati, A. and Metelli, A.
Self-Protection Mechanisms for Web Applications - A Case Study.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 181-188
ISBN: 978-989-758-189-2

to apply for the requested operation, and actuators (coordinated by an Executor), which activate or deactivate the security countermeasures as needed and indicated by the analysis and planning module.

We tried to make our solution as general as possible, so that it can be applied to other application domains without modifications. Indeed, the proposed solution can be applied not only to new Web applications but also to already existing ones, provided that some mechanisms are implemented to allow the self-protection system observe the events occurred and the operations performed at both the application and the network level. Writing all relevant events and operations on log files should suffice; such files are then read in a continuous way by the self-protection mechanisms. From the architectural point of view, the Web application to which the self-protection mechanisms is applied should be written in a modular way, so that each security countermeasure can be activated or deactivated at runtime.

The remainder of this paper is structured as follows. Section 2 introduces the architecture of our solution and the main self-protection mechanisms. Section 3 illustrates two of the self-protection security scenarios implemented in our case study. Section 4 presents the proposed self-protection mechanisms in the context of the taxonomy described in (Yuan, 2014; Yuan, 2012). Conclusions and directions for future work are given in Section 5.

2 OUR SOLUTION

Fig. 1 provides a coarse grain view of the system's architecture, and shows how the self-protection mechanisms interface with the rest of the system. We assume that the Web application has a client-server architecture, and that it can be accessed remotely through a Web browser. There are two main locations where the self-protection mechanisms may be exploited: (1) outside the application, avoiding intruders to access it, through firewall/network level self-protection, and (2) inside the application, after an intruder managed to access as an authorized user, through application-level self-protection mechanisms.

2.1 MAPE-K Control Feedback Loop

The proposed solution is based on the MAPE-K control feedback loop (Cheng, 2009; de Lemos, 2013), which is composed of five elements:

- M - Monitor, which gathers information from the software system and its execution environment (i.e., the network);
- A - Analyzer, which examines the information gathered by the Monitor and identifies if variations relevant for self-protection occurred;
- P - Planner, which establishes which changes should be made in the system to address the variations discovered by the Analyzer; it identifies the set of operations, i.e., the strategy, to be applied in the system;
- E - Executor, which applies the changes identified by the Planner in the system (either in the application or in the firewall configuration, as shown in Fig. 1);
- K - Knowledge Base, which contains information relevant for self-protection such as filtered data coming from the Monitors, statistical information, and self-protection solutions adopted in previous cases.

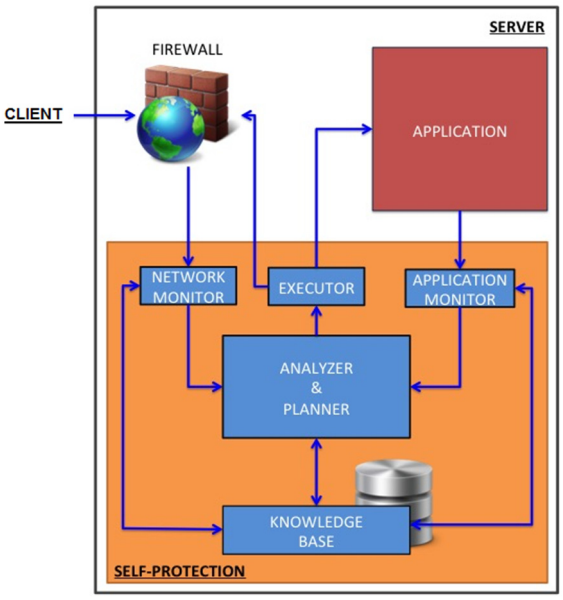

Figure 1: The Overall Architecture of our Solution.

Our solution considers two Monitors for gathering information, one for the network (whose events are observed by the "Firewall" in Fig. 1) and one for the application (the "Application" in Fig. 1). This choice has been made due to the different nature of the observed data in the two cases, and to the different mechanisms required to access them and to distill the meaningful information.

The Analyzer and Planner work together closely and need to rapidly exchange security-related information; hence they can be merged together, as shown in Fig. 1, to increase the overall performance of the self-protection system. They implement the

intelligent behavior of the self-protection solution.

The Executor is in charge of applying the self-protection strategies concerning both the firewall and the application.

The Knowledge Base contains distilled information, concerning both the network and the application. The Analyzer and Planner module exploits both types of information when identifying the variations and the most appropriate self-protection strategy to be applied. This is due to the fact that the self-protection process exploits both types of information to improve the security of the system, as shown in Section 3 through examples.

2.2 Quality Attributes of our Solution

Fig. 1 shows that the application (the home banking system, in this case study) is separate from the self-protection mechanisms. This leads to several advantages: first, the *separation of concerns* principle is met because the business logic is independent of the self-protection logic; second, several aspects of our solution are reusable. The design knowledge can be reused in other case studies from different application domains. Further, the solution may be applied either to a new or to an existing Web application, provided that it can communicate in real time (for example, through a log file) to the application Monitor all the events and the application-related operations that occur, and that the security mechanisms already embedded in the application are divided into modules, each of which can be either activated or deactivated at runtime. A third advantage of our solution is that the modularity of the system is insured. Fourth, the maintainability of the system is easier to perform, as it is easier to identify whether a problem occurred in the business logic or in the self-protection part.

2.3 Risk Levels

Our solution considers that one of three qualitative levels of risk may be associated to each user as well as to the overall system: LOW, MEDIUM, and HIGH. A LOW level of risk indicates that a basic self-protection strategy suffices: for example, it is safe enough to close and lock the door when a person is at home. A MEDIUM level of risk indicates that additional self-protection strategies are needed: for example, a person may use the external alarm system in addition to the closed and locked door when at home. A HIGH level of risk indicates that exceptional self-protection strategies should be adopted: for example, use an alarm system directly connected to a security company when at home, or hire a guard.

In our solution each user, as well as the system, has an associated risk level. The final risk level associated to a given functionality delivered by the system is computed from the risk level of the user requiring such functionality and the current risk level of the entire system. For example, the overall risk level of a home is given by its owner risk level and by the risk level of its building/neighborhood/city/country.

Table 1 shows an example of risk levels. The user Max Fox has associated a LOW risk level, while John Smith has a HIGH risk level. The current risk level of the system, called ALL in the table, is MEDIUM. In our current implementation, we have decided to compute the resulting risk level as the maximum among the user and the system risk levels. Based on this information, Fox's risk level for the current bank operations is max{Max Fox's risk level; ALL risk level} = MEDIUM, whereas the risk level of the operations performed by John Smith is max{John Smith's risk level; ALL risk level} = HIGH.

Table 1: Examples of risk levels for the system (User Name: ALL) and for users Max Fox and John Smith.

User Name	Risk Level
ALL	MEDIUM
Max Fox	LOW
John Smith	HIGH

2.4 Security Map

The proposed solution uses a Security Map, which indicates the security information required for each risk level and for each functionality offered by the application. The Security Map plays an important role in the system because it constitutes a centralized mechanism which indicates the security strategies to be activated for the current functionality offered by the application to a user. It is stored in the Knowledge Base, and it is used by the Executor to activate the various security strategies and modules. This is made possible by the previous assumption that the security mechanisms embedded in the application are divided into modules, each of which can be either activated or deactivated at runtime.

Table 2 shows an example of Security Map. For example, for a login operation, a user having an associated LOW risk is required to insert his user name and password and the answer to a secret question, while a HIGH risk level requires in addition an OTP (One Time Password) generated by a security token (a physical device provided by the

bank to its users, that generates pseudo-random numbers at a fixed rate, typically every 30 or 60 seconds), and a captcha.

The Security Map is an important element of the system, also from the usability and user experience point of view. In fact, a system which always operates assuming a HIGH risk level will certainly be secure, but in terms of usability it will be annoying to use; as a consequence, the users will tend to leave from it after a short time. The Security Map thus provides a reasonable trade-off among security and usability, configurable at runtime.

Table 2: An Example of a Security Map.

	Login			Bank Operations		
Risk Level	Low	Medium	High	Low	Medium	High
Uername	X	X	X			
Password	X	X	X			
OTP		X				X
Keybank			X		X	X
Secret Question			X	X	X	X
Captcha			X			X

3 SECURITY SCENARIOUS USING SELF-PROTECTION

This section presents two of the self-protection scenarios we have currently implemented in our home banking case study called UNIBANK. The first scenario concerns the self-protection mechanisms for the login of individual users to their home banking account. The second concerns the protection of bank operations involving money transfers. These scenarios aim to improve the security of the overall system by dynamically changing the security strategies applied, based on the current risk level of a user and of the system.

3.1 Premises

Before describing the above mentioned scenarios, some premises should be introduced. Initially, UNIBANK has been developed without self-protection mechanisms, hence we have added the self-protection mechanisms without significantly changing the existing application. To enable the activation and deactivation of security modules dynamically at runtime, the security requirements have been implemented in separate modules. Each module has an attribute which indicates if it should be currently active or not for the current user request. This approach also enables us to easily extend the solution with further security mechanisms, if needed. Further, it allows us to apply this solution both to existent and to new systems.

Another important aspect pertains the separation of concerns. Ideally, self-adaptive systems aim to maintain the functional part independent of the adaptive part of a system. In other words, the functional part should not be aware of the existence of the adaptive part. Only the adaptive part should monitor and apply the identified changes in the functional part. To implement this communication between UNIBANK and the self-protection mechanisms we introduced two log files where UNIBANK and the firewall trace all the events occurred in the application and in the network, respectively (in the following subsections we mention only the trace of the failures; however, both successes and failures are traced in the log files). These log files are continuously parsed dynamically by the self-protection part, which extracts only the information which is relevant from the self-protection point of view. The extracted information is then stored in the Knowledge Base, and used at runtime to dynamically change the security strategies. Further, from this information the self-protection part creates statistics about users' habits.

3.2 Self-protection for Users' Login

This scenario aims to protect the login to a user account by changing the strategies for the access to the application based on the current risk level of the user and of the entire system (see Section 2).

The login to a user account is performed in two steps, through a two-factor authentication. First, a user is required to provide some basic authentication information (e.g., the account number, a password, the birth date of the user), which is always used at each bank account access. The application retrieves this information and verifies its correctness.

In the meantime, the Analyzer and Planner module retrieves the risk level currently associated to the user and to the system from the Security Map available in the Knowledge Base, to establish the risk level for the current access to the user's account. In parallel, the Network Monitor retrieves the Internet Protocol (IP) address of the device used by the user to access the bank account. The Analyzer and Planner module verifies if the IP address is not one of the IPs usually used by the user (as these IPs are available in the Knowledge Base) and, in the affirmative case, it raises the risk level associated to the user, meaning that a two-factor authentication is

required; the user is thus requested to provide additional information to login to the bank account. The rationale behind this behaviour is that usually, when an attacker tries to access online bank accounts, he uses services offered by proxy servers, or makes IP spoofing operations, or uses some anonymization network such as Tor (Tor Project, 2015) to continuously change the IP address, to avoid being traced. Of course, it is possible that the user is travelling and hence is using different IP addresses; however, the self-protection system also verifies this situation. In fact, the Analyzer and Planner is able to identify the location of an IP address; based on this information, it computes the time needed to reach the current location starting from the location of the previous user's login attempt. If the location change is compatible with the elapsed time, the risk level of the user becomes MEDIUM, otherwise it becomes HIGH (and the user account is possibly blocked, depending on the system's security configuration).

The Analyzer and Planner module considers the two risk levels, i.e., the one retrieved from the Security Map and the one influenced by the IP address, and identifies the self-protection strategy to be applied to the user for his current login operation. The newly obtained risk level is also stored in the Knowledge Base. If the new risk level is LOW, the user should just provide the answer to a secret question. Otherwise, if the new risk level is different from LOW, further authentication information is asked to the user to complete the login operation, such as an OTP (One Time Password) generated by a security token device or sent via an SMS (Keybank and OTP in Table 2), and a captcha.

When the system retrieves the authentication information provided by the user, it verifies its correctness. If the information is correct, then the system allows the user to access his bank account; on the contrary, if the information is not correct then access is denied and the system increments the number of consecutive failed accesses for the current user. When the number of failures becomes greater than a predefined threshold (e.g., three attempts) the user is notified that the access to the system has been blocked, and that he should contact the bank to solve the problem.

The risk level associated to a user decreases when he performs a successful login. In addition, a user may indicate if he wants to maintain a MEDIUM or a HIGH risk level for his account during a given period of time (e.g., when abroad), after which the risk level is (possibly) decreased. This change is performed automatically and asynchronously from

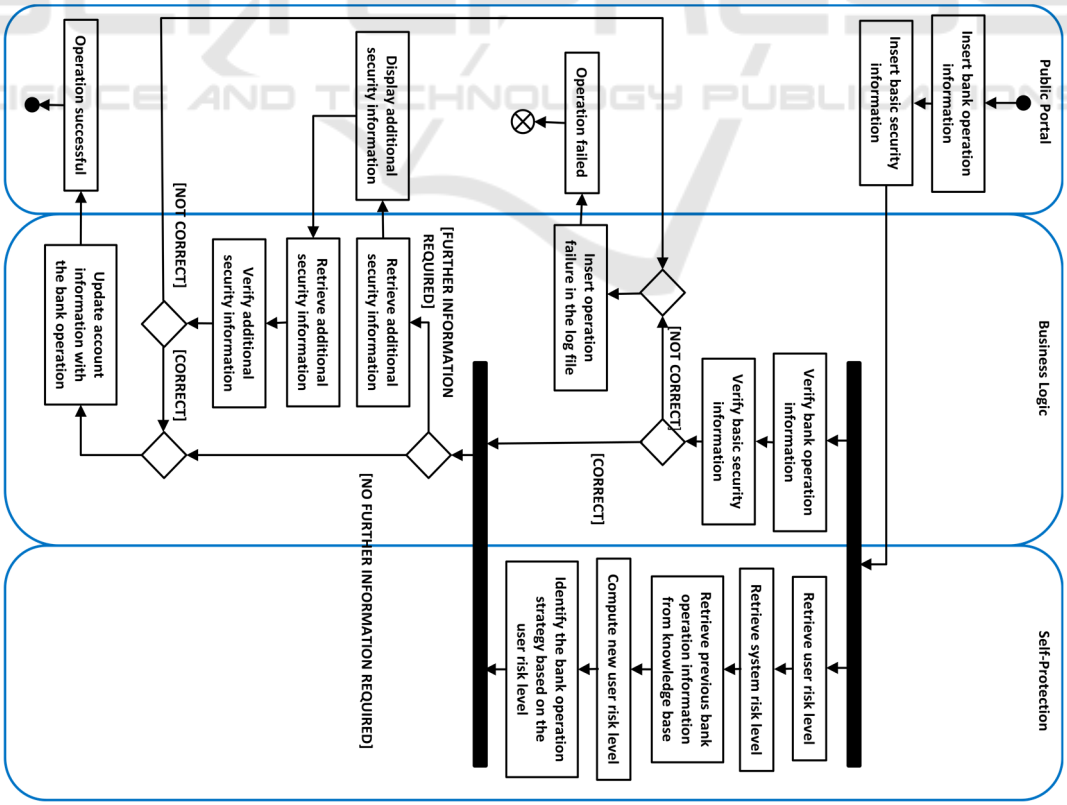

Figure 2: The Activity Diagram for Bank Operations.

other functionalities by the self-protection part.

3.3 Self-protection for Bank Operations

This scenario aims to protect the bank operations (e.g., wire transfers) performed by the users after they have been authenticated, based on their risk levels and on the risk level associated to the system. Fig. 2 shows the activity diagram for this scenario.

In this case the self-protection part monitors over time the users' habits, i.e., the number of bank operations performed by a user each month, the average amount of money transferred from each account, and the average amount of money for each wire transfer and each user. In addition, the self-protection part computes statistics on the number of the overall bank operations in a given time slot (e.g., each hour of a day for an entire week or month) and the amount of money involved in the bank operations (e.g., wire transfers) in a given time slot. This information is exploited and updated during the bank operations. For example, through this information the self-protection part becomes aware if a thief tries to withdraw from one or more accounts an important amount of money (an operation which is unusual for the owner), or if he tries to withdraw a small amount of money from as many accounts as possible (increasing significantly the number of operations in a short time).

As in the previous scenario, also this one is based on two steps. First, a user specifies all the required information concerning the bank operation (e.g., in case of a wire transfer, the receiver, the motivation of the transfer, the receiver's bank account number) and some basic security information (which should be different from the one used to verify the user's identity during authentication, e.g., an OTP sent by the system to the user's mobile phone). The business logic verifies the introduced information and the availability of the money amount involved in the operation in the bank account.

In the meantime, the self-protection part retrieves the current user risk level and the system risk level from the Knowledge Base. Further, it retrieves also the information concerning the statistics on the user's habits. It also increments the number of bank operation requests. Based on this information, it computes a new risk level for the current user and identifies the bank operation strategy to be applied, depending upon this new user risk level.

If the information concerning the bank operation is correct and the money amount is available in the bank account, the bank operation proceeds. If the user risk level is LOW then the business logic executes the bank operation, updates the available money in the bank account, and notifies the user that the operation has been successfully performed. On the other hand, if the user risk level is MEDIUM or HIGH, a second security step is executed, similar to the two-factor authentication, and the system thus requests additional security information to the user. If the information provided by the user (e.g., an OTP sent via SMS or the answer to a secret question) is correct, then the business logic executes the bank operation and updates the available amount of money for the current user. In this case, the business logic notifies the user of the successful operation.

On the other hand, if the user fails to provide correct basic or additional security information, the operation is denied and the business logic traces this failure in the log file. The self-protection part updates the information of the current user in the Knowledge Base by incrementing the number of consecutive failed operations, and possibly also the user and system risk levels.

In this way the system allows bank operations which are not usually performed by the users (e.g., payment of a new car). In case of a security issue (e.g., an attacker who aims to withdraw an amount of money from as many bank accounts as possible), the first bank operation will succeed, but the system will identify and stop the thief by raising the risk level and changing the security information required.

4 A TAXONOMY FOR OUR SOLUTION

Yuan et al. (Yuan, 2014; Yuan, 2012) propose a taxonomy for self-protection systems. The taxonomy contains 14 dimensions (see Fig. 3) grouped in 3 categories: approach positioning, focused on the objectives and intent of the self-protection, approach characterization, focused on how self-protection is achieved, and approach quality, focused on the evaluation of the self-protection.

This taxonomy is also presented through a home banking motivating example, which supposes five scenarios where self-protection can significantly improve the security of the system (Yuan, 2012).

For example, one of the scenarios detects an illegal access and automatically disables the connection from the source IP address. Our first scenario (see Section 3.2) also takes into consideration the IP addresses from which users usually access their accounts. If an unusual situation is identified, the system raises the risk level and

Approach Positioning															
Self-Protection Levels			Depths of Defence					Life Cycle Focus		Security Goals			Meta-Level Separation		
Monitor & Detect	Respond & Protect	Plan & Prevent	Network	Host	Middleware	Application	Depth-Independent	Development Time	Runtime	Availability	Integrity	Confidentiality	No Sepration	Partial Separation	Complete Separation
X	X	X				X			X	X		X			X

Approach Characterization																									
Theoretical Foundation					Meta-Level Decision-Making			Control Topology			Response Timing			Enforcement Locale			Self-Protection Patterns									
Logic/Formal Methods	Heuristics	Optimization	Learning	Other	Single Strategy	Multi-Strategy	Cost-sensitive	Local Only	Globally Centralized	Globally Decentralized	Reactive	Proactive	Hybrid	System Boundary	System Internal	Local-neutral	Protective Wrapper	Agreement-based Redundancy	Implementation Diversity	Countermeasure Broker	Agent-Orientation	Protective Recomposition	Attack Containment	Software Rejuvenation	Reconfiguration of Reflex	Artificial Immunization
	X					X			X		X	X			X							X		X	X	

Approach Quality							
Validation Method				Repeatability		Applicability	
Empirical	Proof	Simulation	None	Low	High	Low	High
		X			X		X

Figure 3: Yuan et al.'s Taxonomy Applied to Our Solution.

changes the self-protection strategies dynamically. If necessary, our solution denies the access.

Another example in (Yuan, 2012) considers a built-in access control which may detect abnormal usage patterns and disable users' accounts. Our 2nd scenario (see Section 3.3) implements and extends Yuan's et al.'s scenario with various risk levels and dynamic strategies. Our solution can be considered an enhanced solution of the ones in (Yuan, 2012).

Yuan and Malek have also collaborated with the authors of the Rainbow framework to add self-protection mechanisms (Yuan, 2013). Further, Rainbow's authors have implemented denial-of-service mechanisms in their framework (Schmerl, 2014). The Znn.com case study exploiting Rainbow may be compared to our solution, being a Web application which addresses security issues; the self-protection scenarios we have considered are complementary with respect to the ones implemented in Znn.com.

In the rest of this section, the dimensions of the three categories of the self-protection taxonomy are applied to our case study (see Fig. 3). We present the positioning and characterization together because, in our opinion, there are dependencies between them. The quality dimensions are separately addressed.

4.1 Approach Positioning and Characterization

The Self-Protection Level dimension concerns the sophistication level of the self-protection mechanisms. In our case study all possible levels are addressed. Our solution is able to Monitor the system and Detect security issues. It implements mechanisms to Respond and Protect the system against the detected issues. It provides support to Plan improvements and Prevent security problems, because the system is able to exploit information related to the current state of the system, as well as previous information (e.g., statistics, historical data).

This leads to other two dimensions: Response Timing and Theoretical Foundation. Response Timing has as values Proactive, i.e., the system is able to foresee and prevent problems based on previous information, and Reactive, i.e., the system is able to respond to occurred problems. Theoretical

Foundation has the value Heuristic because our home banking system exploits knowledge-based and strategy-based mechanisms for self-protection.

The Depths-of-Defense dimension indicates the layer where self-protection mechanisms operate. In our case study, self-protection mechanisms are at the Application layer. This dimension leads to the Protection Goals, which in our case concern the Confidentiality and Availability of the system: the self-protection mechanisms aim to avoid illegal access, impersonation, and Denial of Service.

Lifecycle Focus indicates whether the self-protection mechanisms are used during the development or the execution, at runtime. As most of the self-protection mechanisms, security at Runtime is exploited for our case study.

Meta-Level Separation focuses on the separation of concern principle from the architectural point of view: we have a Complete separation because the self-protection mechanisms have been added to an existing system, modifying it as few as possible.

Meta-Level Decision Making concerns the decision making strategy. In our case it has a Multi-Strategy value because we exploit information coming from the firewall and the application, and we exploit the overall information to adapt the system.

Control Topology is Global, because self-protection concerns the entire application, and Centralized, because one brain makes decisions.

The Enforcement Locale dimension indicates the scope of the self-protection mechanisms, i.e., the application, hence the System internal value.

Adaptation Patterns indicate the recurring architectural patterns applied for the solution. We have Protective Recomposition, to dynamically change the security information needed for the current functionality for a user, and Reconfiguration on Reflex, to change the security level for a user and for the entire system. We are currently implementing the Software Rejuvenation pattern, which enables the graceful termination of an application and immediately restart it in a clean internal state.

4.2 Approach Quality

There are three dimensions for the evaluation of self-protecting systems. The Validation Method concerns the way in which the effectiveness of the proposed approach is performed. We have simulated a home banking system through a prototype.

From the Repeatability point of view, our home banking is documented in a BsC thesis, available on request. The software is available on GitHub at: https://github.com/MetelliAndrea/Knabinu.

The Applicability dimension concerns the specificity of the approach to an application domain or case study. Our self-protection mechanisms can be applied to other Web applications.

5 CONCLUSIONS

This paper has presented self-protection mechanisms for new or existing Web applications which aim to improve security at runtime. These mechanisms exploit the users' and system's risk levels and manage dynamically the security strategies.

Further work concerns the extension of the self-protection mechanisms for further security issues and case studies in different application domains. We also plan to evaluate the efficiency overhead introduced by the self-protection mechanisms in UNIBANK, by comparing the running times of the two currently available versions of this case study: with and without the self-protection mechanisms.

REFERENCES

Anderson, R.J., 2008. *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd Edition. Wiley.

Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., 2009. *Software Engineering or Self-Adaptive Systems*. LNCS 5525, Springer.

de Lemos, R., Giese, H., Muller, H., Shaw, M., 2013. *Software Engineering for Self-Adaptive Systems II*. LNCS 7475, Springer.

Pfleeger, C.P., Pfleeger, S.L, 2006. *Security in Computing*, 4th Edition Prentice Hall.

Schmerl, B., Camara, J., Gennari, J., Garlan, D., Casanova, P., Moreno, G. A., Glazier, T. J., Barnes, J. M., 2014. Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*.

Stallings. W., 2013. *Network Security Essentials: Applications and Standards*, 5th ed. Pearson, 2013.

Tor Project, 2015. www.torproject.org.

Yuan, E., Esfahani, N., Malek, S.,2014. A Systematic Survey of Self-Protecting Software Systems. In *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 8, Issue 4, Article No. 17.

Yuan, E., Malek, S., 2012. A Taxonomy and Survey of Self-Protecting Software Systems. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 109-118.

Yuan, E., Malek, S., Schmerl, B., Garlan, D., Gennari, J., 2013. Architecture-based self-protecting software systems. In *9th International ACM Sigsoft Conference on Quality of Software Architectures*, pp. 33-42.