

Securing Security Policies in Autonomic Computing Systems

G. Jabbour¹ and D.A. Menasce²

¹Information Technology & Engineering, George Mason University, Fairfax, Virginia, U.S.A.

²Department of Computer Science, George Mason University, Fairfax, Virginia, U.S.A.

Abstract - Protecting valuable enterprise assets, such as data, computer systems, and networks is becoming increasingly complex especially in autonomic computing systems due to their reduced reliance on human intervention and verification, and increased reliance on the ability of the system to make business or strategic decisions. This naturally makes the system extremely vulnerable to security breaches and malicious attacks. Successfully mitigating security threats in autonomic systems requires a two-pronged approach that ensures solid and continuous protection. First, it requires that defense mechanisms be made an integral and inseparable part of the system to be protected. Second, it requires that the defense mechanisms themselves be secure to ensure uninterrupted protection. In addition, these defense mechanisms must be flexible, extensible, and scalable enough to adapt to changing threats and be effective in neutralizing them. This paper presents a framework for addressing the protection of the security policies of autonomic systems. The security policies are partitioned into several layers in a way that significantly increases the difficulty of attacking the system.

Keywords: Autonomic computing, security policies, databases.

1 Introduction

The increasing complexity of computer systems has necessitated an ever increasing investment in resources [1]. Modern computer systems tend to be highly networked and therefore exposed to a multitude of attacks. Unable to cope with all the overhead of managing complexity, system owners started (to some extent) to embrace the adoption of autonomic features (e.g., self-management, self-configuration, self-optimization) into their systems. The reader is referred to [10-16] for relevant literature on autonomic computing. Building secure and trustworthy autonomic systems remains a challenging task [2] because these systems have a reduced reliance on human intervention and verification since such systems are mostly driven by embedded policies [3], [4].

For autonomic systems to properly defend themselves, they must be aware of their internal state and the environment that surrounds them. Through self-awareness

and context-awareness features, these systems can defend themselves by adhering to the embedded policies that have been set by their owners in order to insure proper operations. This means that self-aware and context-aware autonomic computing systems are able to provide the right services at the right time to the right clients. They can detect hostile intrusive behavior and take action to protect the integrity and operational status of the system. They can also correctly fulfill policy requirements for secure environments [5]. On the other hand, in order for system owners to embrace and accept the decisions of their autonomic systems, they have to first trust them. Trusting autonomic systems and autonomic elements has been realized as an important success factor by many [2], [6-8]. If users of autonomic computing systems do not trust them, they will not use them regardless of the value that they add to their business needs. Today, there is still no clear process of explicitly designing and building autonomic systems that are geared towards earning the user's trust, despite the clear awareness of trust as a main contributor to embracing autonomic computing [6].

The behavior of autonomic computing systems is basically driven by high-level policies established by their owners. Attackers that are able to gain access and are capable of modifying such policies have the potential to cause more damage than when attacks are perpetrated to non autonomic computing systems. This paper proposes a framework for ensuring that the security policies of an autonomic system or element are secure. The work presented in this paper builds on our previous work [9] on the topic of embedding security policies into the core existence of a database system in order to provide uninterrupted autonomic enforcement of security configurations as set by the system owner. The rest of this paper is organized as follows. Section 2 introduces a framework for protecting the security policies in autonomic systems. Section 3 discusses related work and section 4 concludes the paper.

2 Framework for Securing the Security Policies

2.1 Layered approach to preserving secure policies

The foundation of the layered approach is twofold. The first is to partition the building blocks of a security policy into several layers thus adding complexity to the overall architecture of the policy. This serves the purpose of adding perceived ambiguity, which is intended to retract potential hackers from deciphering the policy's contents and mandates. The second is to inject into the different partitions of the policy false elements (parameters and their values) whose characteristics have the sole purpose of confusing a hacker and giving a false sense of achieving his/her objective. The four main aspects (explained in greater detail in the balance of this section) of the Securing the Security Policy (SSP) architecture presented in Figure 1 and explained in what follows are: partitioning & obfuscation, verification, awareness, and monitoring.

In the partitioning and obfuscation aspect we separate the main building blocks of the policy, which collectively provide an insight into its mandate, from each other and make them look as if they are foreign entities that are completely unrelated to each other. This is accomplished through two main actions. The first is implemented by separating each of the policy identification information, the policy parameters, and the policy mandates from each other. The second is fulfilled by the addition of false columns that mimic the actual columns, which have been deliberately omitted to protect the policy. The injection of these fake columns is meant to confuse a potential hacker and deviate his/her attacks from the real components of a policy. Next we address the verification aspect where components that verify signature behavior of the system, the system owner, and the system user are introduced into the makeup of a policy to ensure that only its real owners can alter the policy. In the awareness aspect we endow the security policy with capabilities that allow it to be aware of its own state as well as its surrounding environment. Finally, in the monitoring aspect we wrap the security policy with a monitoring agent that is charged, in conjunction with other components of the policy, with invoking self-protection measures when it perceives a certain action as a threat. In what follows, we address each of these aspects separately to illustrate their use in securing the policies that are usually tasked with securing an autonomic system.

2.1.1 Partitioning & Obfuscation of policy entities and injection of false ones

The separation of policy components for the purpose of protecting them is accomplished partially by hosting the main building blocks of the policy in separate database tables inside the policy schema itself, and partially by

injecting false columns that mimic their counterparts in their respective database tables. To illustrate we use a database policy that enforces the archive mode of a database (see [9]).

```
[type]           Enforce database archive mode policy
[name]           Archive Log Mode
[id]             1123070001V10
[parameter]     log_archive_start
[paramvalue]    TRUE
[dbcolumn]      log_mode
[dbvalue]       archivelog
[verification]  SELECT LOG_MODE FROM
                SYS.V$DATABASE;
[validation]    IF LOG_MODE = NOARCHIVELOG
[actionrejec]   Alert system owner of failed attempt, notify
                user of disallowed parameter change, and
                record action in audit trail for further
                investigation
[actionexec]     Execute enf_archive_log_proc
```

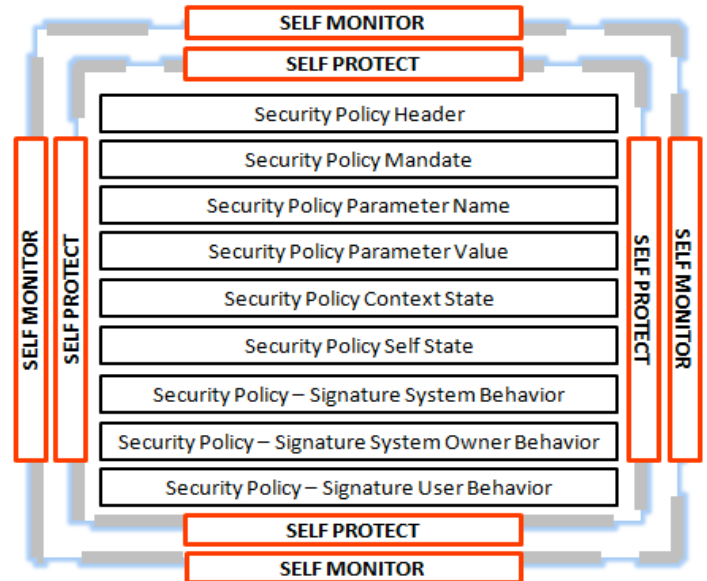


Figure 1. Architecture of securing the autonomic security policy framework

In the SSP framework, we partition the policy above into four different parts, each residing in its own database table within the same policy schema as follows:

Partition 1: Includes policy header information

```
[type]          enforce DB archive mode policy
[name]          Archive Log Mode
[id]            1123070001V10
```

Partition 2: Includes policy parameter identifiers

```
[parameter]    log_archive_start
[dbcolumn]     log_mode
```

Partition 3: Includes policy parameter values

```
[paramvalue]   TRUE
[dbvalue]      archivelog
```

Partition 4: Includes policy mandates

```
[verification] SELECT LOG_MODE FROM
                SYS.V$DATABASE;
[validation]   IF LOG_MODE =
                NOARCHIVELOG
[actionrejec]  Alert system owner of failed
                attempt, notify user of
                disallowed change, record
                action in audit trail
[actionexec]   Execute enf_archive_log_proc
```

Then, we inject into each partition false columns that mimic the original columns that were initially omitted. Therefore, each policy partition looks again like the original complete policy with the difference that only certain columns are real while the rest are phony. The bolded columns in each partition are the real ones. Unauthorized changes to the values of the injected (false) columns have no impact on the mandate and the effectiveness of the security policy. The phony columns serve the only purpose of distracting and deceiving a potential hacker. The data in the false columns could also be falsified to the point that it has no bearing on understanding how the policy works.

Policy with **Partition 1** as ONLY active component

```
[type]          Enforce DB archive mode
                  policy
[name]          Archive Log Mode
[id]            1123070001V10
[parameter]    log_archive_start
[paramvalue]   TRUE
[dbcolumn]     log_mode
[dbvalue]      archivelog
[verification] SELECT LOG_MODE FROM
                SYS.V$DATABASE;
[validation]   IF LOG_MODE =
                NOARCHIVELOG
[actionrejec]  Alert system owner of failed
                attempt, notify user of
                disallowed parameter change,
                record action in audit trail
[actionexec]   Execute enf_archive_log_proc
```

Policy with **Partition 2** as ONLY active component

```
[type]          Enforce DB archive mode
                  policy
[name]          Archive Log Mode
[id]            1123070001V10
[parameter]    log_archive_start
[paramvalue]   TRUE
[dbcolumn]     log_mode
[dbvalue]      archivelog
[verification] SELECT LOG_MODE FROM
                SYS.V$DATABASE;
[validation]   IF LOG_MODE =
                NOARCHIVELOG
[actionrejec]  Alert system owner of failed
                attempt, notify user of
                disallowed parameter change,
                record action in audit trail
[actionexec]   Execute enf_archive_log_proc
```

Policy with **Partition 3** as ONLY active component

```
[type]          Enforce DB archive mode
                policy
[name]          Archive Log Mode
[id]           1123070001V10
[parameter]    log_archive_start
[paramvalue]  TRUE
[dbcolum]     log_mode
[dbvalue]     archivelog
[verification] SELECT LOG_MODE FROM
                SYS.V$DATABASE;
[validation]  IF LOG_MODE =
                NOARCHIVELOG
[actionrejec]  Alert system owner of failed
                attempt, notify user of
                disallowed parameter change,
                record action in audit trail
[actionexec]   Execute enf_archive_log_proc
```

Policy with **Partition 4** as ONLY active component

```
[type]          Enforce DB archive mode
                policy
[name]          Archive Log Mode
[id]           1123070001V10
[parameter]    log_archive_start
[paramvalue]  TRUE
[dbcolum]     log_mode
[dbvalue]     archivelog
[verification] SELECT LOG_MODE FROM
                SYS.V$DATABASE;
[validation]  IF LOG_MODE =
                NOARCHIVELOG
[actionrejec] Alert system owner of failed
                attempt, notify user of
                disallowed parameter
                change, record action in
                audit trail
[actionexec] Execute
                enf_archive_log_proc
```

2.1.2 Verification : signature behavior of system owners, users, and potential hackers

Another essential part of the proposed framework is the introduction of components that verify signature behavior of the system owner, the system user, and potential hackers in order to ensure that the policy can only be altered by its legitimate owners. The policy itself would be capable of knowing certain aspects about its users by observing and recording their behavior during their interaction with the system. Some examples of observable user behavior are:

-Time of day or the week when changes are allowed. Upon the initial creation of a policy, the system owner would set a certain timeframe, a maintenance window, during which changes are allowed to take place. Changes outside the set timeframe would not be permitted unless a sequence of personal identification information is provided by the system owner in such a way that the system can accurately identify him/her as its legitimate owner. For example, the security policy would not normally allow the shutdown of a database during normal hours of operation.

-The age of a parameter value. Upon the initial setting of a policy's parameter value, the system owner would finalize the setting thus initiating the beginning of its lifetime period. As time passes by, the age of the parameter value increases, and the system would recognize it as a permanent setting and a part of its "normal state." Changing such a parameter value by a hacker would become almost impossible. In turn, the system owner would have to provide a sequence of personal information that identifies him/her as the true system owner before changes are allowed.

-The policy collects information related to the system users. This information is classified into two categories. The 'derived behavior information' and the 'volunteered behavior information'. The former is automatically collected by the system over time, based on user actions and their outcome. The latter, on the other hand, is requested of the user or the system owner as a means for proper identification. The system requests information such as a personal identification number, password, personal questions such as place of marriage, age at marriage, first owned car, favorite sports, etc., and possibly the last four digits of the social security number. The system stores this information, as well as any user interaction patterns, for use in future identification of events or requests.

2.1.3 Awareness: endowment of self and context awareness

In order for any security policy to be able to defend itself against malicious attacks, it must be aware of its

normal state and the environment that surrounds it including any change that takes place. This enables the policy to become proactive in protecting its critical elements. On the other hand, an autonomic system must also be aware of its context in order to defend against unusual or abnormal user behavior. The self-awareness and context-awareness approaches have been used in the security context of autonomic systems but not in the context of securing the security policies themselves. In our framework, self-awareness and context-awareness are made an integral and inseparable part of the security policy itself. In turn, the security policy itself is also an integral and inseparable part of the autonomic system (or database) that it is meant to protect.

2.1.4 Monitoring: initiation of a monitoring agent for self-protection

The proposed framework also includes the implementation and initiation of a monitoring agent that serves as an essential element in the self-protection mechanism. Each security policy is wrapped with an embedded monitoring agent that is charged, in conjunction with other components of the policy, with invoking self-protection measures when it perceives a certain action as a threat. The monitoring agent registers all requests that attempt to change security policy settings, and monitors the response of the policy to such requests. If the agent detects unfavorable conditions, it intercepts the communication and renders it ineffective.

2.2 Process Workflow

In our previous work [9] on the applicability of policy-based enforcement of database security configuration, we demonstrated how that approach can be implemented in an Oracle 10g database and showed how its application can be very essential in providing an uninterrupted mechanism for securing a database. In this paper we argue that unless the security policies themselves are protected, it is useless to use them for protecting an autonomic system. Figure 2 presents a flowchart of the processes that are invoked and the events that take place from the point of initiating or requesting a change to the configuration settings of an autonomic database to the point where the request is either accepted and applied or deemed malicious and rejected. This entire process is carried out by the autonomic database itself without any outside help or interference. The absolute strength of our approach lies in the notion that an autonomic system or database cannot fully protect itself unless its defense mechanism is made an integral and inseparable part of its core being.

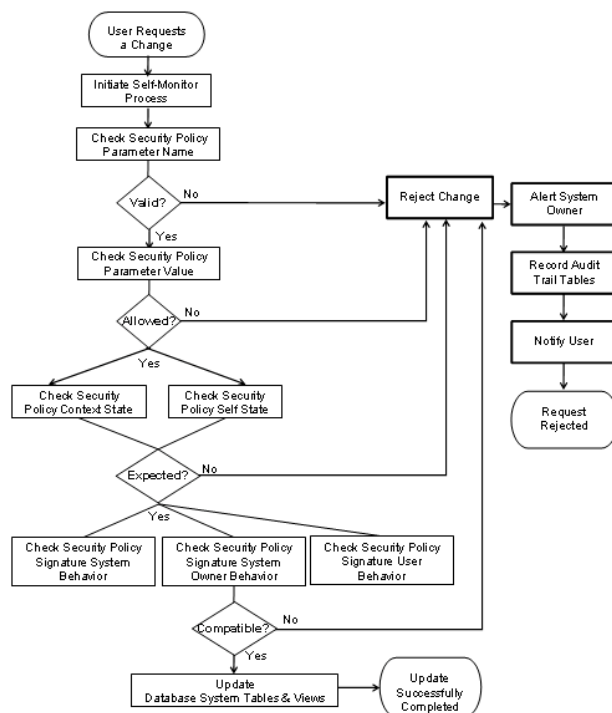


Figure 2. Processes involved in securing the autonomic security policies

3 Related Work

The notion of embedding policies into computer systems has been applied to several domains [4, 17-34]. Many commercial software scanning and monitoring tools have been developed [35-44]. Their major deficiency is that they run independently of the system they monitor. In addition, the policies are under the control of the system administrator who can change their parameters at will making the system vulnerable to attacks. In [6], the authors addressed the concept of trust as an important factor in the success of new autonomic features and in [5] the authors developed a security model based on the context in which the autonomic system functions. The authors in [2] concluded that while no functioning system is perfectly secure, the ability of an autonomic system to consistently and correctly react to various levels of events, benign or malicious, is the key to the achievement of self-protection. In addition, they believe that the implementation of autonomic computing techniques offer the promise of making systems more secure by effectively and automatically enforcing high-level security policies. This supports the vision we presented in [9]. In the current paper we expand that work towards the protection of autonomic security policies.

4 Conclusion

We presented what we believe to be an important area of research as it relates to protecting the elements that are entrusted with providing security to autonomic computing systems. The premise of our framework is based on two

main foundations. The first is to make the security policy an integral and inseparable part of the autonomic system that it is supposed to protect. The second is to partition the security policy in a way that increases the difficulty of attacks. Our framework equips the security policy with multiple defense mechanisms that allow it to understand its normal state and the context in which it operates, and to prompt it to investigate suspicious activities and directly and indirectly interrogate suspicious users comparing their behavior to what it considers part of its normal state.

References

- [1] Fink, G. and D. Frincke, Autonomic Computing: freedom or threat? ;LOGIN, 2007. 32(2).
- [2] Chess, D.M., C.C. Palmer, and S.R. White, Security in an autonomic computing environment. IBM Systems Journal, 2003. 42(1).
- [3] Bahati, R.M., et al., Using Policies to Drive Autonomic Management. Proc. IEEE 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06), 2006.
- [4] Badr, N., A. Taleb-Bendiab, and D. Reilly, Policy-Based Autonomic Control Service. Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04), 2004.
- [5] Wan, K. and V. Alagar, Security Contexts in Autonomic Systems. IEEE, 2006.
- [6] Duez, P.P., M.J. Zuliani, and G.A. Jamieson, Trust by Design: Information Requirements for Appropriate Trust in Automation. Pierre Guez, Greg Jamieson and IBM Canada Ltd., 2006.
- [7] Telford, R., et al., Usability and design considerations for an autonomic relational database management system. IBM Systems J., 2003. 42(4).
- [8] Lee, J.D. and K.A. See, Trust in Automation: Designing for Appropriate Reliance. Human Factors, 2004. 46: p. 50-80.
- [9] Jabbour, G. and D.A. Menasce, Policy-Based Enforcement of Database Security Configuration through Autonomic Capabilities. The Fourth International Conference on Autonomic and Autonomous Systems (ICAS 2008), March 16-21, 2008, 2008.
- [10] Menasce, D.A. and J.O. Kephart, Autonomic Computing, Guest Editor Introduction. IEEE Internet Computing, 2007. 11(1).
- [11] Wang, M., et al., Autonomic Element Design Based on Mind Agent Model. IJCSNS International Journal of Computer Science and Network Security, 2006. VOL.6(No.9B).
- [12] Nami, M.R. and K. Bertels, A Survey of Autonomic Computing Systems. Proceedings of the Third International Conference on Autonomic and Autonomous Systems (ICAS 07) 2007.
- [13] Vassev, E. and J. Paquet, Towards an Autonomic Element Architecture for ASSL. International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'07), 2007.
- [14] Parashar, M. and S. Hariri, Autonomic Computing: An overview. Lecture Notes in Computer Science, Rutgers University, 2005.
- [15] Parashar, M., et al., AutoMate: enabling autonomic applications on the Grid. Cluster Comput (2006) 9, 2006: p. 161-174.
- [16] Fuad, M.M. and M.J. Oudshoorn, System Architecture of an Autonomic Element. Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'07), 2007.
- [17] Bao, J.Q., L. Guo, and W.C. Lee, Ad hoc networks: Policy-based resource allocation in a wireless public safety network for incident scene management. Proceedings of the 2006 workshop on Dependability issues in wireless ad hoc networks and sensor networks DIWANS '06, 2006.
- [18] Itani, W., A. Kayssi, and A. Chehab, T1-B: computer and network security symposium: An enterprise policy-based security protocol for protecting relational database network objects. Proceedings of the 2006 international conference on Wireless communications and mobile computing IWCMC '06, 2006.
- [19] Wright, M.J., Using policies for effective network management International Journal of Network Management, 1999.
- [20] Kim, G., J. Kim, and J. Na, Design and implementation of policy decision point in policy-based network. Proc. 4th Annual ACIS International Conference on Computer and Information Science (ICIS'05), 2005.
- [21] Perez, G.M., et al., Dynamic Policy-Based Network Management for a Secure Coalition Environment. IEEE Communications Magazine 2006.
- [22] Olausson, E. and A. Karlsen, A policy-based priority and precedence framework for military IP networks. Military Communications Conference, MILCOM 2004. IEEE, 2004.

- [23] Rudack, M., et al., Policy-based quality of service mapping in distributed systems. Network Operations and Management Symposium (NOMS), 2002, IEEE/IFIP, 2002.
- [24] Flegkas, P., P. Trimintzios, and G. Pavlou, A policy-based quality of service management system for IP DiffServ networks. IEEE Network, 2002.
- [25] Montanari, R., G. Tonti, and C. Stefanelli, Policy-based separation of concerns for dynamic code mobility management. Proceedings of the 27th Annual International Computer Software and Applications Conference, COMPSAC 2003, 2003.
- [26] McDaniel, P., On Context in Authorization Policy. 8th ACM SYMPOSIUM ON ACCESS CONTROL MODELS AND TECHNOLOGIES, 2003.
- [27] Bhatti, R., K. Moidu, and A. Ghafoor, Healthcare data integration and exchange: Policy-based security management for federated healthcare databases (or RHIOs). Proceedings of the international workshop on Healthcare information and knowledge management HIKM '06, 2006.
- [28] Tripathi, A., D. Kulkarni, and T. Ahmed, Policy-Driven Configuration and Management of Agent Based Distributed Systems. 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems SELMAS'05, 2005.
- [29] Alexandrescu, A. and E. Berger, Policy-Based Memory Allocation: Fine-tuning your memory management Doctor Dobb's C/C++ Journal, 2005.
- [30] Kawarasaki, M. and R. Atarashi, Policy Based Content Delivery Management Using Metadata. Proceedings of the IEEE 2004 International Symposium on Applications and the Internet Workshops (SAINTW'04), 2004.
- [31] Chau, C.-k., Policy-based Routing with Non-strict Preferences. SIGCOMM 2006, 2006.
- [32] Verdi, F.L., M. Magalhaes, and E.R.M. Madeira, Policy-based admission control in GMPLS optical networks. Proc. First International Conf. Broadband Networks (BROADNETS'04), 2004.
- [33] Yan, Y., et al., A policy-based admission control algorithm for UMTS end to end QoS provision. 2nd Intl. Conf, Mobile Technology, Applications and Systems, 2005.
- [34] Ramirez, S.L., et al., Performance Evaluation of Policy-Based Admission Control Algorithms for a Joint Radio Resource Management Environment. Electrotechnical Conference, MELECON 2006. IEEE Mediterranean, 2006.
- [35] AuditPro, Network Intelligence (I) Pvt. Ltd. www.niiconsulting.com.
- [36] AppDetectivePro, Application Security, Inc. www.appsecinc.com/products/appdetective/.
- [37] E-Trust Policy Compliance, Service Strategies, Inc., www.ebusiness-security.com/eTrust_Policy_compliance.htm.
- [38] NGSSQLCrack, Next Generation Security Software, www.ngssoftware.com/products/database-security/ngs-sqlcrack.php.
- [39] OScanner, cqure.net. www.cqure.net/wp/?page_id=3.
- [40] Symantec Enterprise Security Manager™ for Databases, Symantec, www.symantec.com/region/can/eng/product/esm/databases.
- [41] NeXpose™, Rapid7. www.rapid7.com.
- [42] NGSSQuirreL, Next Generation Security Software. www.ngssoftware.com/products/database-security/ngs-squirrel-oracle.php.
- [43] Appsentry, Integrigy Corporation. www.integrigy.com/products.
- [44] ISS, ISS Database Scanner. Internet Security Systems, Inc., www.iss.net/about/index.html.