HAL
open science

Enforcing private policy via security-by-Contract

Gabriele Costa, Ilaria Matteucci

To cite this version:

Gabriele Costa, Ilaria Matteucci. Enforcing private policy via security-by-Contract. 2010. inria-00458882

HAL Id: inria-00458882

https://inria.hal.science/inria-00458882v1

Preprint submitted on 22 Feb 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Enforcing Private Policy via Security-by-Contract[1]

Gabriele Costa[A,B] , Ilaria Matteucci[B]
[A] Dipartimento di Informatica -Università di Pisa, Italy
[B] Istituto di Informatica e Telematica - C.N.R., Pisa, Italy
{Gabriele.Costa, Ilaria.Matteucci}@iit.cnr.it

Abstract.
This work aims to investigate how the Security-by-Contract (SxC) paradigm, developed for providing security assurances to mobile applications, can be used for guaranteeing the security of communicating systems composed by several, heterogeneous components. These components need to communicate with each other by establishing direct, point to point connections. Direct connections can involve components sharing no common communication protocols and need a suitable interface.
Enablers are in charge of providing these communication interfaces. Each component has a local security policy composed by a public and a private part. When a communication between two components has to be established, each component asks the enabler for providing a communication interface that respects its public policy.
We exploit the Security-by-Contract approach for assuring that the application implementing the communication interface is always safe, i.e., it satisfies the security policies setted by components.
Moreover, we present an extension of the Security-by-Contract for dealing with trust. Trust management is useful when one of the involved actors is considered to be potentially untrusted and the others want to measure its trust level.

Keywords: Private Policy Enforcement, Security-by-Contract, Distributed Connecting System.

1. INTRODUCTION

The growing diffusion of computational entities and their composition in networks has made security a critical issue in every phase of the software life-cycle. The modern scenarios present a complexity degree that only few years ago was unpredictable.
Let us consider a communicating system composed by two classes of actors: components and enablers. Components are the end-users in our network. They can be both software agents (e.g., programs) or physical devices (e.g., mobile phones). Whenever two components want to communicate with each other they need to agree on a common protocol. If this is not possible, they ask to an enabler to provide them with a way to communicate. Enablers store information for creating/choosing an appropriate application for allowing the communication between different components. Moreover, each component has requires a local security policy to be always respected by applications running on it. Such security policy is the combination of two local security policies: a private policy and a public one. A private policy is a security policy hidden to the external world, e.g., because it speaks about some sensitive data. On the contrary, a public policy is made publicly accessible, e.g., because it does not involve information that the component considers to be private.
Our goal is to guarantee that the communication between components is always secure, i.e., a provided application satisfies local security policies. We firstly provide a description of the threat

[1] Work partially supported by EU project FP7-231167 CONNECT (Emergent Connector for Eternal Software Intensive Networked Systems) and by the EU project FP7-214859 Consequence (Context-aware data-centric infor-mation sharing).

models for that system. Secondly, we show that the Security-by-Contract paradigm [6] can be suitably applied for dealing with them. In doing this, we reduces our analysis to systems in which components trust to each other but they do not trust to enabler.
The Security-by-Contract (SxC) framework was developed for providing security of mobile application. It is based on the idea of contract of an application. Roughly, a contract is a description of the behaviour of an application that comes together with the application itself. The typical scenario in which the SxC is applied consists in a device user defining his own security policy. When the user receives an application verifies whether its code and its contract actually match. If the check fails the user can decide to delete the application or to enforce the security policy on it by exploiting a proper enforcement infrastructure. Otherwise, the system can proceed and verifies whether the contract satisfies the user's policy. Once again, if this step fails the solution consists in enforcing the active policy on the execution. Finally, if the previous checks were positively passed, the application can be executed with no active runtime enforcement.
In the simplest case, we assume to have a single enabler. Two components send to the enabler a communication request and their public security policies. The enabler provides an application, e.g., by synthesizing it on-the-fly or selecting it from an existing pull, with a contract describing its behaviour. Components expect to receive a contract that satisfies their public security policies. Remarkably, being not aware about the private security policies of the components, the enabler does not guarantee the application to comply with them. Each component involved in the communication checks whether the contract is compliant with both its own public and private policy.
Going ahead, we also consider trust aspects by extending the Security-by-Contract paradigm with trust (SxCxT) [3]. Indeed, we extend our scenario by assuming to have many enablers forming a cloud. Two components that want to communicate with each other send the request to the cloud of enablers. Similarly to the previous case, each enabler provides an application. However, in this scenario different enablers concur to provide their applications and the components decide which application will be adopted. Clearly, components can discriminate according to their levels of trust w.r.t. the enablers.
Hereafter, assuming that components associate each enabler with a certain level of trust, we propose an integration of the SxC paradigm with a monitoring/enforcement infrastructure dealing with contracts. Contract monitoring allows for trust level management of enablers depending on their applications behaviour. Indeed let us assume that enablers are initially considered to be trusted. When components obtain some application provided by an enabler, they start monitoring it. Monitoring follows the execution an updates the trust value associated with the enabler depending on the actions fired by the application.

The paper is structured as follows: Section 2 recalls some background notions about the Security-by-Contract paradigm. Section 3 describes threat models for the considered system and Section 4 and 5 present the application of the SxC and SxCxT paradigms to threat models, respectively. Section 6 compares our work with what already exists in literature. Section 7 concludes the paper and describes some future work.

2. BACKGROUND

Security-by-contract (SxC) is a paradigm for providing security assurances to mobile applications [6]. According to [7], the basic idea of Security-by-Contract is to certify the code by binding it together with a contract. Loosely speaking, a contract contains a description of the relevant features of the application and the relevant interactions with the hosting platform. Relevant features include, among others, sensitive memory usage, secure and insecure web connections, user privacy protection, data confidentiality.
By signing the code, developers certify its compliance with the stated claims on its security-relevant behaviour.

Definition 2.1: A contract is a formal complete and correct specification of the behavior of the application for what concerns relevant security actions (e.g., Virtual Machine API call, Operating System Calls). It defines a subset of the traces of all possible security actions.

The second cornerstone of the SxC approach is the concept of policy. A mobile platform can specify its own contractual requirements through a customised policy. Moreover, policies can be user-defined. In both cases, if a contract matching the policy means that the corresponding application (provider) declares an acceptable behaviour.

Definition 2.2: A policy is a formal complete specification of the acceptable behavior of applications to be executed on the platform for what concerns relevant security actions (e.g., Virtual Machine API calls, Operating System Calls). It defines a subset of the traces of all possible security actions.

The Security-by-Contract paradigm works as follows: when a client receives an application, it verifies whether the code and the contract actually match by using an evidence checking procedure. If the check fails, the user can decide to refuse (delete) the MIDlet or to enforce the security policy on it by exploiting a monitoring/enforcement infrastructure.
The enforcement approach has been shown to be feasible on mobile devices exploiting several different techniques [1, 4].
Otherwise, the system can proceed to verify whether the contract (correctly representing the application) satisfies the user's policy. Once again, if this step fails the solution consists in enforcing the active policy on the execution. Finally, if the previous checks were positively passed, the MIDlet can be executed with no active runtime monitor.
The Contract-Policy matching function ensures that any security relevant behavior declared by the contract is also allowed by the policy. This matching step can be done w.r.t. different behavioural relation, e.g., language inclusion [5] or simulation relation [8]. Hence, the matching procedure formally guarantees that applications having a contract that is compliant with the active policy never violate it. Figure 1 shows the SxC pre-execution strategy.

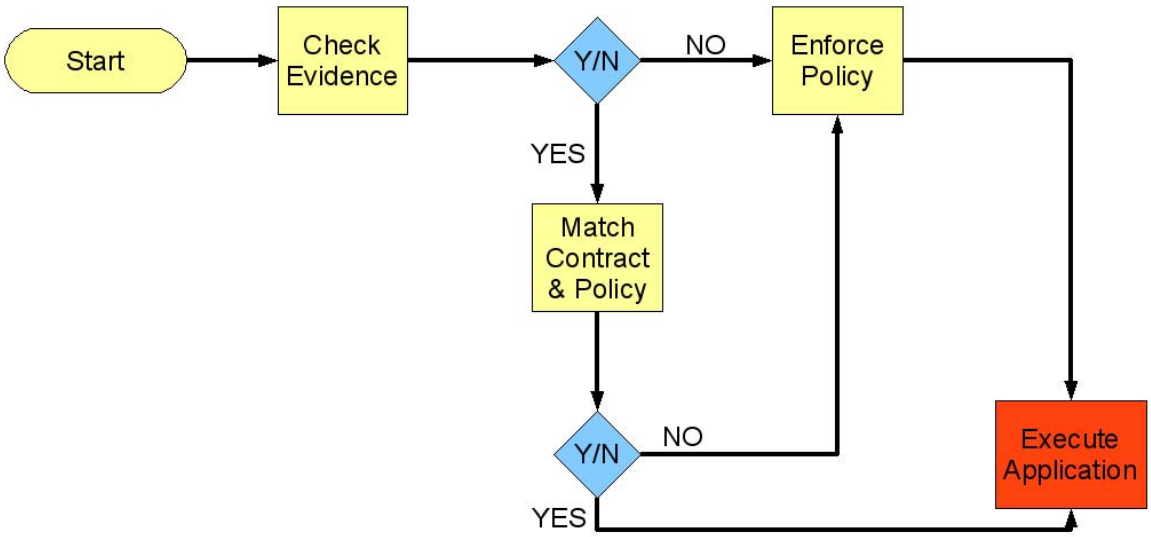


Figure 1: The Security-by-Contract Application life-cycle [7].

3. THREAT MODELS

We introduce and detail several threat models. Let us consider a basic scenario in which two components want to establish a communication. For sake of simplicity, we assume that there is only one enabler in the network. To communicate, the two components send their requirements to the enabler. Then, the enabler produces a suitable application by taking it from a pre-compiled repository or synthesizing a new one on-the-fly.
In this process we distinguish two security levels:

- Local security of each component of the network. Each component sets a local policy that has to be locally satisfied. Such security policy is the combination of a private policy and a public one. A private policy is a security policy that a component does not declare to the external world, e.g., a policy that speaks about sensitive data. On the contrary, a public policy is a policy that the component consider freely deliverable, e.g., a policy that does not involve private information. Clearly, two limit cases exist: i) a component has only a private policy or ii) it sets only a public policy.
- Global security of the distributed system. Interactions among components and enablers must satisfy some security policy describing the correct behaviour of the distributed system as a whole.

Focusing on trust aspects, the following sub-cases arise:

- The two components involved in the communication trust each other but they do not trust the enabler.
- Each component involved in the communication trusts the enabler but does not trust the other component.
- All actors (components and enabler) trust on each other.
- No trust relationships exist between two components or between a component and the enabler.

4. SECURITY-BY-CONTRACT FOR CONNECTING COMPONENTS

We show how to apply the SxC paradigm for providing network components with local security. As we said in Section 3, here we consider to have two components, trusting each other, that interact with an untrusted enabler.

4.1 Private and Public Security Policy

First of all, we investigate on the possible relationship existing between public and private policies defined on each single component. Three cases arise:

1. The public policy is a generalization of the private policy. This means that the public policies allows more execution than the private one. A particular instance of this case occurs when only the private policy has been defined.
2. The private policy is a generalization of the public one. Referring to the notion of compliance used in the SxC paradigm, this case means that whenever the public policy is satisfied also the private one will be. An instance of this case occurs when no private policy has been setted on the component.
3. Private and public policy partially overlap. There exists at least one execution that satisfies both the private and the public policy.

Some examples can clarify the above classification.

Example 4.1: Imagine a component requiring a connector and declaring the following, informally defined, policies:

- Ppub: "Always connect to the same host"
- Ppriv: "Encrypt every message with key ***k***"

As it refers to some sensitive data, i.e., ***k***, the component can be interested in keeping Ppriv secret. Clearly, these two policies do not imply each other (unencrypted messages can be delivered to a single host or correctly encrypted messages can be sent to multiple hosts). However, there are executions satisfying both of them, e.g., a single, encrypted message is sent. This example belongs to the third case introduced above.

Example 4.2: Imagine now a component similar to the previous one, but defining a different private policy.

- P^{pub}: "Always connect to the same host".
- P^{priv}: "Connect only with host ***h***".

Here, the set of behaviours accepted by P^{priv} also satisfies P^{pub}. Hence, we are in the second case.

In addition to the relation existing between private and public security policy, we must also investigate the relation among security policies and applications contract. All possible relations are summarized in Figure 2.

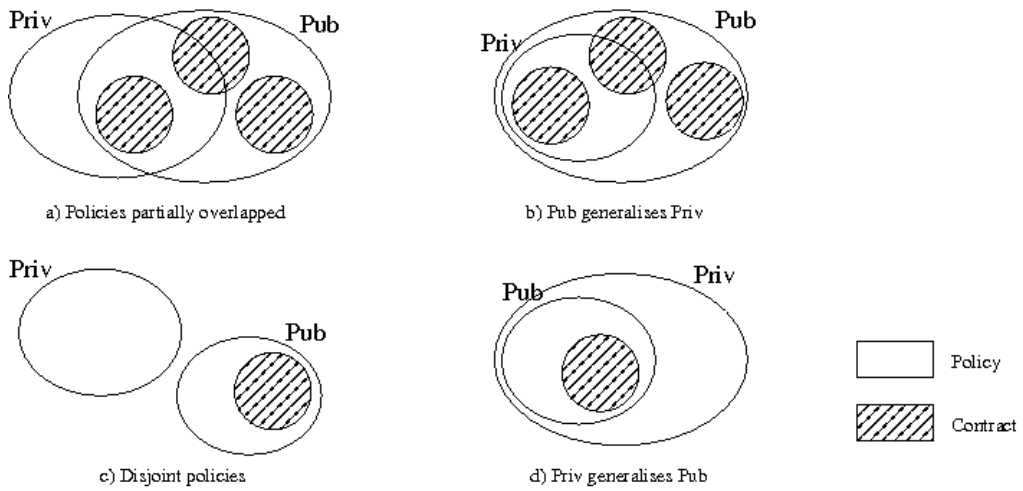


Figure 2: A graphical representation of possible relations among Private policy, Public policy and Contract.

In the following, we assume that the public policy generalizes the private policy (case (b) in Fig. 2). The relationship between the policy and the contract drives the Security-by-Contract run-time strategy. Figure 3 shows schematically the system behaviour.

If the contract complies with the private policy (M in figure), the system monitors the application run-time compliance w.r.t. its contract. Instead, if the contract does not comply with the private policy (E in figure), the private policy is enforced on the executing application. Note that we are intentionally ignoring the case in which private policy and contract have no intersection. Indeed, whenever this happens, the enforcement process forces the execution to generate only empty traces.

4.2 Enforcing Private Security Policy through SxC

Let us consider the scenario in Figure 4. Two components want to start a conversation and ask an enabler for a suitable application, i.e., a connector. Each component has its own security policy

composed by a private and a public part. Both components declare a public policy that generalizes the private one.
Let P_1 and P_2 be the public policies of the two components. To communicate, both components send a connection request and their local public policies to the enabler.

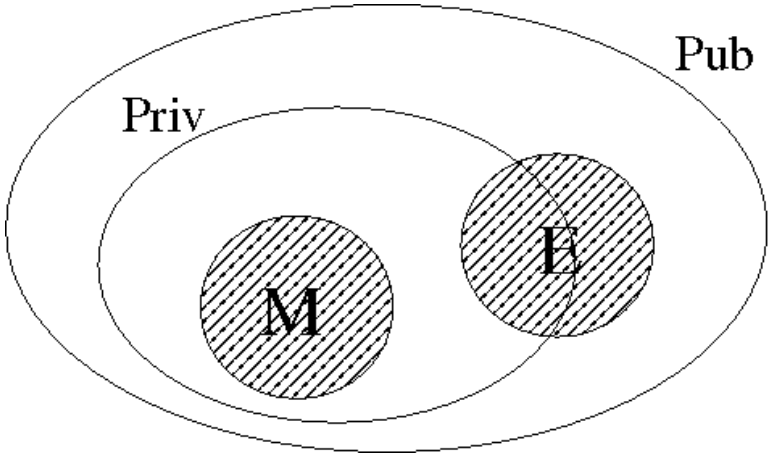


Figure 3: A graphical representation of scenario (b). M denotes contract monitoring and E policy enforcement.

If possible, the enabler answer by providing the components with an application. This application will be a mobile code that the enabler annotated with a contract C denoting its behaviour. By design, when the enabler returns a contract C then it also declare, under its own responsibility, that C satisfies both P_1 and P_2.
After retrieving the application, the components must verify how its contract fits with their private policies (P^{priv}_1 and P^{priv}_2 respectively). Since $P^{pub}_{(1/2)}$ generalise $P^{priv}_{(1/2)}$ components only need to check contract against private policies. If the application is accepted from both the components, it is used to establish the communication.
An instance of this scenario arises when both components declare no public security policy. Since there is no assumption about the application contract that the enabler returns, this case reduces to the original SxC scenario for mobile devices [6].
However, when a component receives an application it decides which monitoring/enforcement configuration must be activated. Firstly, the component verifies whether the code comes from a trusted source. If the check fails then the component can decide to refuse the application or to enforce the security policy on it (Enforce Policy).
Otherwise, the component verifies whether the contract (considered to correctly denote the application) satisfies the private security policy (Contract-Policy Matching [8] at deployment-time). If this step fails the solution consists in enforcing the active policy on the execution and, possibly, also monitoring the application contract. In this scenario, the contract monitoring configuration can be activated on a statistical base in order to record possible application misbehaviours leading to a trust weight adjustment.
Finally, if the previous checks were positively passed, the communication is established and only the contract monitoring is turned on.

5. ENABLERS TRUST MANAGEMENT

We detail our trust management-oriented extension of the SxC paradigm, hereafter denoted by SxCxT [3]. The main novelty of our technique consists in adding a new component to the standard

architecture. This module is responsible for contract monitoring and reacts to contract violations by updating the trust level of the involved enabler.
This could be useful, for instance, if we consider more than one enabler in the system.

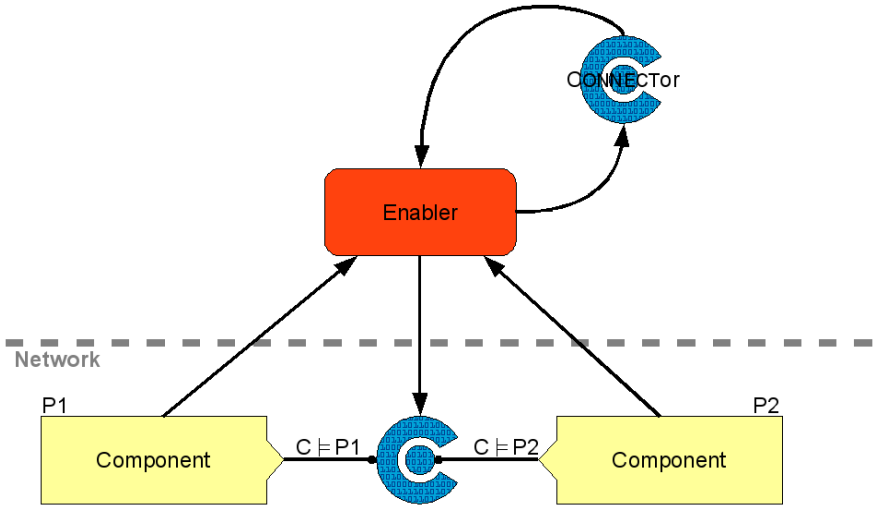


Figure 4: Secure application.

Figure 6 shows the application life-cycle. In model, the Check Evidence step has been replaced by a check on the enabler trust measure (Trusted Provider). This check discriminate between applications running guarded by a mere policy enforcement and applications being contract-monitored.
Whenever the contract satisfies the local public policy, two possible relations with the private policy arise:

1. The contract satisfies the local private security policy. In this case our monitoring/enforcement infrastructure is required to monitor only the application contract. Indeed, under these conditions, contract adherence also implies private policy compliance. If no violation is detected during the execution then the application worked as expected. Otherwise, we discovered that a trusted party provided us with a fake contract. Our framework reacts to this event by reducing the level of trust of the indicted provider and switching to the policy enforcement modality.
2. The contract does not satisfy the local private security policy. Since the contract declares some potentially undesired behaviour, policy enforcement is turned on. Similarly to a pure enforcement framework, our system guarantees that executions are policy-compliant. However, monitoring contract during these executions can provide a useful feedback for better tuning the trust vector. Hence, our framework also allows for a mixed monitoring and enforcement configuration. This configuration is statistically activated with a certain probability. Several strategies are possible for deciding the probability to activate the monitoring procedure, e.g., using a fixed value or a function of the current trust level.

Note that, in the second scenario, the monitoring component plays a central role. Indeed, a contract violation denotes that a trusted provider released a fake contract. The reaction of our system to this event is a instant correction of the trust measures.

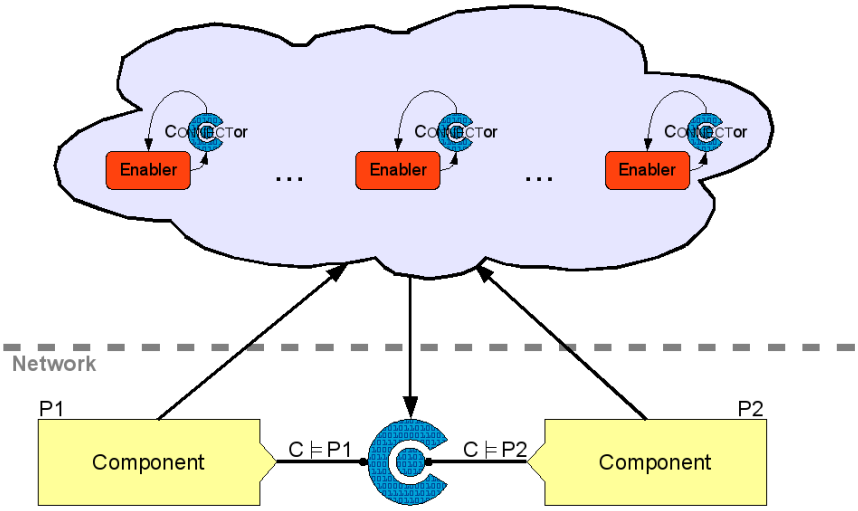


Figure 5: Secure and Trusted application

5.1 Trust adjustment via Contract Monitoring

We outline how trust measures assigned to security assertions can be adjusted by our contract monitoring strategy. Trust measures associated with an enabler mainly concern on the contract goodness. Hence, updated trust measures will influence future interactions with an enabler and its applications. In other words, our system penalizes enablers when their contracts do not specify the correct applications behaviour.

Here we present a possible extension of the monitoring/enforcement infrastructure model proposed in [4]. The monitoring/enforcement infrastructure mainly consists of two components: a Policy Decision Point (PDP) and Policy Enforcement Points (PEPs). Roughly, the PDP holds the actual security state and is responsible for accepting or refusing new actions. Actions are delivered to the PDP by PEPs that in charge of intercepting security-relevant operations done by the application. When the application tries to run a monitored command, it stimulates a corresponding PEP that suspends the execution and fires the action to the PDP. The PDP evaluate the action against the its current security state and answer allowing or denying the permission to proceed. Finally the PEP enforce the PDP decision by running or aborting the operation under analysis.

In our system the PDP is also responsible for the contract monitoring operations and for the trust vector updating.

According to [1, 4], we assume that both contracts and policies are specified through the same formalism. Hence, the policy enforcement configuration of the PDP keeps unchanged. The PDP must load connector contracts as well as local private security policies dynamically. Moreover, it must be able to run under three different execution scenarios (Fig. 6): policy enforcement enabled, contract monitoring enabled or both.

The base enforcement scenario (execution scenario 1) is actually unchanged w.r.t. the standard usage of the classical PDP. Hence, no contract monitoring nor trust management operations are involved.

Main interest resides in the other two scenarios. The contract monitoring scenario applies to application carrying a contract released by a trusted enabler (Section 5.1).

Similarly to the policy enforcement strategy, PEPs send action signals to the PDP. An important difference is that the PDP keeps in memory the program events trace. When a signal arrives, the PDP checks whether it is consistent with the monitored contract. If the contract is respected then the internal monitoring state is updated and the operation is allowed. Otherwise, if a violation attempt is detected, the PDP reacts by changing its state.

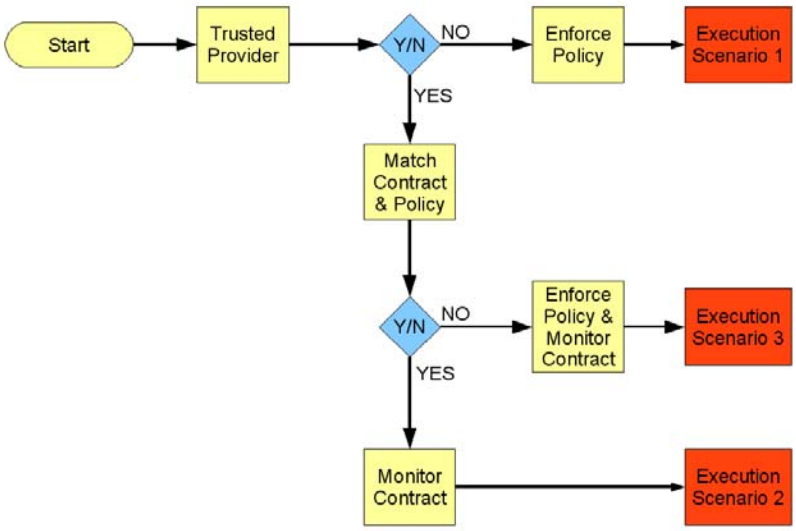
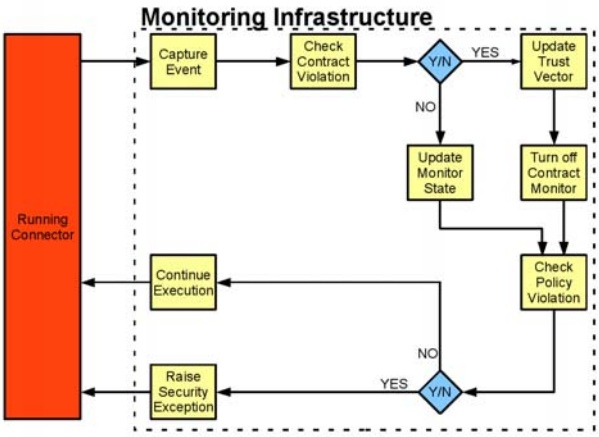


Figure 6: The Extended Security-by-Contract Application life-cycle.

The first consequence of a contract violation is a decreasing of the trust weight of the involved enabler.
Secondly, the PDP switches its state from contract monitoring to policy enforcement configuration. Since an instance of the policy is always present, this operation does not imply a serious computational overhead. Afterwards, the policy state is updated using the execution trace recorded during the monitoring phase. This step, that can be time consuming, is necessary for verifying whether, breaking the contract, the application has also violated the policy. However, this computational cost, being the consequence of an extraordinary event, must be paid at most once. Indeed, when the PDP is both monitoring a contract and enforcing a policy, the current policy state is known. Finally, the execution continues with the PDP enforcing the policy starting from the last action, that is the event breaking the contract.



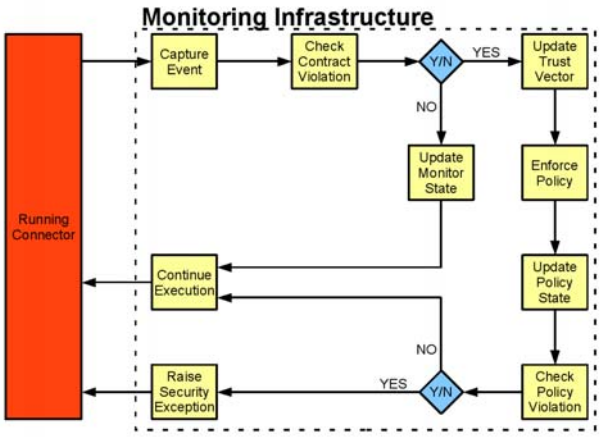


Figure 7: Mixed Enforcement/Monitoring strategy. Figure 8: Pure Monitoring strategy.

Figure 7 and Figure 8 show the behaviour of the monitor/enforcement infrastructure performing the contract monitoring task in the two previously discussed scenarios. Summing up, both execution scenario 2 and 3 check contract violations through the contract monitoring strategy described above

and update enablers trust level. Such updates will influence future interactions with applications and enablers.
In this way our system gains experience and refines its interactions with enablers by ignoring applications coming from malicious enablers. Moreover, the executions are always kept safe by the security enforcement infrastructure.

6. RELATED WORK

In literature there not exist a lot of work on possible extension of the Security-by-Contract paradigm for dealing with trust and, in particular, that integrate policy enforcement and trust management, especially for mobile code.
An attempt of integration of trust management and fine grained access control in Grid Architecture can be found in [10] where it is proposed an access control system the enhances the Globus toolkit with a number of features. This copes with the fact that access control policies and access rights management becomes one of the main bottleneck using Globus for sharing resource in a Grid architecture. Along this line of research [2] presents an integrated architecture, extending the previous one, with an inference engine managing reputation and trust credentials. This framework is extended again in [9] where it is introduced a mechanism for trust negotiating credential to overcome scalability problem. In this way the framework provided preserves privacy credentials and security policy of both users and providers. Even if the application scenario and the implementation is different, the basic idea of considering the trust as a metrics for deciding the reliability of an application provider.
Also [11] presents a reputation mechanism to facilitate the trustworthiness evaluation of entities in ubiquitous computing environments. It is based on probability theory and supports reputation evolution and propagation. The proposed reputation mechanism is also implemented as part of a QoS-aware Web service discovery middleware and evaluated regarding its overhead on service discovery latency. On the contrary, our approach is not a probabilistic. We provide a method according to we update the level of trust of an application provider.

7. CONCLUSION AND FUTURE WORK

Here we have investigate on how using the Security-by-Contract paradigm for enforcing private policy in a communicating system composed by several components that ask to communicate with each other. Firstly, we have investigated the possible threat models if the system and then we showed an application of SxC for establishing a secure communication among components of a network by guaranteeing that the application
for communication is secure, i.e., it works according the security policies required by components.
Secondly, we afterwards extended the SxC approach for managing trust measures by exploiting a contract monitoring procedure. Furthermore, we presented how this extension is useful for the considered system in which several actors are involved and measuring trust could be needed.
Many future directions are viable. Indeed we aim to adapt both the SxC and the SxCxT paradigms to all possible threat models we have pointed out in the introduction.
Furthermore, mainly our trust management strategy is still a work in progress. Currently, trust weights can only decrease monotonically as a consequence of contract violations with the only exception of a direct intervention of the user, see e.g., [12].
Moreover, quantitative representation of this adjustment we consider to present in the future work.

REFERENCES

[1] A. Castrucci, F. Martinelli, P. Mori, and F. Roperti. Enhancing java me security support with resource usage monitoring. In ICICS, pages 256–266, 2008.

[2] M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli. Fine grained access control with trust and reputation management for globus. In OTM Conferences (2), pages 1505–1515, 2007.

[3] G. Costa, N. Dragoni, A. Lazouski, F.Martinelli, F. Massacci, and I.Matteucci. Extending security-by-contract with quantitative trust on mobile devices. In IMIS'10: Proceedings of the 4th International Workshop on Intelligent, Mobile and Internet Services in Ubiquitous Computing, 2010. To appear.

[4] G. Costa, F. Martinelli, P. Mori, C. Schaefer, and T. Walter. Runtime monitoring for next generation java me platform. Computers & Security, July 2009.

[5] L. Desmet,W. Joosen, F. Massacci, P. Philippaerts, F. Piessens, I. Siahaan, and D. Vanoverberghe. Security-by-contract on the .net platform. volume 13, pages 25–32, Oxford, UK, 2008. Elsevier Advanced Technology Publications.

[6] N. Dragoni, F. Martinelli, F. Massacci, P. Mori, C. Schaefer, T. Walter, and E. Vetillard. Security-by-Contract (SxC) for software and services of mobile systems. In At your service - Service-Oriented Computing from an EU Perspective. MIT Press, 2008.

[7] N. Dragoni, F. Massacci, and K. Naliuka. Security-by-contract (SxC) for mobile systems or how to download software on your mobile without regretting it. Position Papers for W3C Workshop on Security for Access to Device APIs from the Web, December 2008.

[8] P. Greci, F. Martinelli, and I. Matteucci. A framework for contract-policy matching based on symbolic simulations for securing mobile device application. In ISoLA, pages 221–236, 2008.

[9] H. Koshutanski, A. Lazouski, F. Martinelli, and P. Mori. Enhancing grid security by fine-grained behavioral control and negotiation-based authorization. Int. J. Inf. Sec., 8(4):291–314, 2009.

[10] H. Koshutanski, F. Martinelli, P. Mori, L. Borz, and A. Vaccarelli. A fine grained and x.509 based access control system for globus. In OTM, pages 1336–1350. Springer, 2006.

[11] J. Liu and V. Issarny. An incentive compatible reputation mechanism for ubiquitous computing environments. Int. J. Inf. Secur., 6(5):297–311, 2007.

[12] S. Naqvi, P. Massonet, B. Aziz, A. Arenas, F. Martinelli, P. Mori, L. Blasi, and G. Cortese. Fine-Grained Continuous Usage Control of Service Based Grids - The GridTrust Approach. In ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet, pages 242–253, Berlin, Heidelberg, 2008. Springer-Verlag.