# Mitigating Security Risks in Web Service Invocations: Contract-Based Approaches

**Gabriele Costa**
*University of Genova, Italy*

**Roberto Mandati, Fabio Martinelli, Ilaria Matteucci, ArtsiomYautsiukhin**
*Institute of Informatics and Telematics of CNR, Italy*

## ABSTRACT

The pervasiveness of Web services increases the necessity for consumers to access and use them in a secure way. Besides secure communications, consumer security also involves providing strong guarantees that a requested security policy is satisfied. Needless to say, remote services are adverse to most techniques of control and analysis that usually require direct access to either execution or implementation. In this chapter, we classify service execution paradigms and provide a characterization of the security threats that may affect a Web service infrastructure depending on the elements composing it. In particular, we provide a discussion of the threat models for several different Web service paradigms involving service consumers, providers, and platforms and illustrate how and when contract-based security approaches and its variants can be applied for mitigating risks in service invocations in the identified paradigms.

## INTRODUCTION

Web services offer various functionalities to its consumers, including data storage, information retrieval, social interaction, and more. A service and its clients interact through specific interfaces defining the syntax and semantics of the exchanged messages (and their parameters). Papazoglou (2007) defines some key roles for Service Oriented Computing; among them, the *service consumer* and the *service provider* are the two most important ones. The two entities share knowledge only about the service interface, i.e., the protocols that they use to communicate. Existing protocols can guarantee security properties, e.g., authenticity and secrecy, on these communications. However, messages can carry complex data or even executable instructions, for example, mobile code, which makes the computation distributed over the involved systems. Needless to say, mobile systems exacerbate the problem of providing security guarantees for both service consumer and provider.

Recently, some proposals highlighted the advantages of including a third entity in services architectures, that is, the *service platform*. In general, a service platform offers support to both consumers and providers. For instance, consumers may ask the platform for information about a service, e.g., its cost and its provider identity. Similarly, providers may obtain support for the orchestration with other services. Clearly, each feature of the platform must be implemented by

appropriate components. When consumers and providers interact with a service platform, they implicitly accept it as a trusted entity and they expect it to provide protection against possible misbehaviors. According to its designated purpose, the platform may include support for service publication and discovery, service composition, mobile code signature, and even execution.

In this chapter, we provide a characterization of the security threats which may affect a Web service infrastructure, according to the elements composing it. In particular, we provide a classification of the threat models for different Web service paradigms involving service consumers, providers, and platforms. We consider the consumer point of view and examine three main *resources* in each paradigm: the service *code*, the service *contract*, and the consumer *policy*. Furthermore, we survey techniques that have been proposed for assessing security issues. Then, according to the availability and reliability of the resources, we show how contract-based approaches, such as *Security-by-Contract* and its variants can be applied to guarantee the security of Web services. The result is a precise characterization of the necessary conditions for the application of the security assessment methods for the Web services.

The chapter is organized as follows. The next section provides background information on protection techniques for assessing security aspects in a service-driven environment and presents related work in the area. Next, service composition paradigms considered in this work are discussed followed by an illustration of how and when Security-by-Contract and its extensions can be applied for guaranteeing security. Finally, a discussion about future work and some concluding remarks are presented.

## BACKGROUND

Several techniques have been proposed to tackle specific security aspects. These approaches may be combined in security frameworks or used to guarantee the reliability of third-party provided resources. In our context, resources are the service code, the service contract, and the consumer policy.

We assume that each consumer specifies its security requirements, herein referred to as *policies*. A *policy* is a security requirement that a consumer wants to apply to a service execution. In general, consumers want to be sure that their policies will be respected during service execution. A violation happens when a service S behaves in a way that is not allowed by the policies. Thus, a threat corresponds to the possibility that a service violates a policy. Usually, consumers have more than one policy to be satisfied by a service.

Generally, service providers release interface information about the provided service, called the *service contract*. A service contract is a formal description of the service behavior. Contracts typically describe the service in terms of interaction protocol (e.g., input and output channels, message syntax, parameter types, and encryption algorithms) and service computation (e.g., message semantics, service state transitions, and resource usage).

The techniques listed below exploit these resources for obtaining security guarantees. In general, these guarantees consist in proving that precise relations exist among (two or more) resources.

***Evidence checking:*** In order to verify a piece of code statically, some methods use an entity provided proof (evidence) and validate it. This entity can be a trusted third party or the developer itself. The proof is linked to the code, e.g., through instrumentation, and the provider originally computes it. Verifying the validity of the proof is more efficient than generating it. The

verification procedure follows the steps of the proof and if all of them are correct, validates its conclusion. In our context, the proof conclusion can be the compliance either with a contract (code-contract verification) or with a policy (code-policy verification). Notice that, as the proof is computed by the provider, allowing customers to specify their own policies requires new proofs to be generated. By the way, that makes this solution extremely cumbersome for service providers. An instance of this approach is the model-carrying code method (Sekar et al., 2003).

***Compliance verification (Matching):*** Static analysis techniques can be applied in order to certify that a specification, e.g., a contract, is compliant with a required policy (contract-policy matching). However, tools for verification suffer from the high complexity that rapidly grows with both the expressiveness of the formalisms and the size of the specifications. Moreover, since the steps of the verification chains can take place at different locations, we need some guarantee on the reliability of these results.

***Enforcement:*** The run-time enforcement approach consists of running a service code inside the scope of a controller, which checks its execution step by step. At each operation, the behavior of the service is compared with the consumer policy (policy enforcement). As violations are prevented, the controller guarantees that the service running in the controller context satisfies the policy. Many implementations of the policy enforcement approach have been proposed (Bauer, Ligatti, & Walker, 2005). Nevertheless, they poorly fit with the distributed computation, typical of Web services, where often there is no single location where the code is executed. In addition, the reaction of the enforcement mechanism to violation detection can modify the standard service behavior, e.g., by closing a session beforehand, thus introducing session failures. However, in some cases, an enforcement facility can be included in the service execution environment, provided by a service platform.

***Monitoring:*** In contrast to enforcement, the monitoring approach only observes the behavior of the service. The monitor does not intervene in the execution and only notifies the interested party (e.g., consumer) about the events that occur. This information (e.g., audit logs) is analyzed and decisions are made about possible violations of the contract (*contract monitoring*) or the policy (*policy monitoring*). Such a monitor, which does not alter the behavior of the code, is called a passive monitor.

***Quantitative assessment:*** Security metrics aim at assessing security threats by releasing the assumption that a system is either secure or not. These techniques use non-binary quantities, e.g., real or natural numbers, in security domains for representing the available knowledge about a system. For instance, metrics can describe a system in terms of its reputation in a community, number of past successful interactions, or average number of failures per year. Then, these values are exploited in order to make security aware decisions. Among the others, *trust and risk* aspects are receiving major interest.

Henceforth, we will discuss contract-based approaches that are able to guarantee security at run-time in Web services. These approaches cover almost all functionalities described above combining several aspects of these assessment techniques in an integrated way. In (Dragoni & Massacci, 2007), the Security-by-Contract framework has been applied to Web services for client-server interactions. In particular, the authors only consider cases in which the client retrieves and executes the full service implementation. In this chapter, we cover this case and extend the application of the Security-by-Contract framework by presenting new variants, which fit better with more realistic and common scenarios.

Jones & Hamlen (2011) present a mechanism for securing remote service execution. Briefly, they propose a reference monitor in-lining system able to inject security instructions in service implementations. Although they can produce a secured version of the service implementation, their method can only be applied when the whole computation takes place on the client side.

Recently, OSGi (OSGi Alliance, 2012) technology has received major attention for the development of service platforms and several authors have proposed techniques to enhance its security. In (Phung & Sands, 2008) the authors use a program transformation based approach to add security monitoring instructions in OSGi bundles. Since they did not aim at assessing the security of Web services in general, they did not consider the assumptions discussed in the present work and we cannot compare their technique with those described here. Still, their work shows how a service platform can automatically instrument a piece of code with security checks.

The *Security-by-Contract* methodology has also been proposed to deal with OSGi security issues in (Gadyatskaya, Massacci, & Philippov, 2012), where the authors consider the case of an OSGi platform applying a local security policy. Under these assumptions, they can apply the Security-by-Contract framework with minor modifications. This approach offers no guarantees to bundle providers and users; nevertheless, the extensions we describe can be applied to this work.

Carminati, Ferrari, & Hung (2006), consider the problem of guaranteeing that the composition of Web services complies with security constraints. In practice, they propose a secure Web service broker, i.e., a component that can generate a composition satisfying the security goals of both provider and consumer. Since their proposal does not consider the necessity of handling mobile code, it does not apply to our assumptions. However, our service platform definition is compliant with the security components presented in (Gadyatskaya, Massacci, & Philippov, 2012), and they can be included in our model.

The next section presents the security assessments under different Web service paradigms.

## ASSESSING SECURITY IN WEB SERVICE PARADIGMS

In this section, we describe the possible architectural models considered in this chapter, which arise from different service deployment configurations.

Beyond security specifications, which we assume to be always present, service networks differ in their *organization*. In other words, a service implementation can be carried out by different network participants, i.e., provider, consumer, and if present, a service platform. We classify different paradigms according to the parties actually executing the service implementation. Needless to say, the party executing a piece of software has full access to its structure and behavior, including its security properties.

We identify three basic paradigms:

- *Consumer-based* paradigm: where code is entirely executed by the consumer.
- *Provider-based* paradigm: where code is entirely executed by the provider.
- *Platform-based* paradigm: where code is entirely executed by the platform.

Moreover, the recent trend is to distribute the service implementation over multiple entities. In particular, we identify the following hybrid paradigms.

- *Provider-Platform* paradigm: where the service is implemented by the provider that also uses platform-provided facilities.
- *Consumer-Provider* paradigm: where the service implementation is the composition of consumer and provider executions.
- *Consumer-Platform* paradigm: where the service implementation consists of consumer and platform execution.

Below, we detail this classification, also providing examples of each paradigm and highlighting specific threats for each of them.

In order to provide a better understanding of the different paradigms and assumptions presented in this chapter, we propose a simple example. The example consists of two services, i.e., H and T. Service H offers access to a hotel reservation infrastructure, while T implements a travel booking service. Since the tasks of T may include hotel booking, its developer is interested in composing T with H. There is also an alternative service H', being a competitor of H, which offers a different hotel booking service. The developer of T may select either H or H' for hotel booking. Thus, T is a consumer for both H and H', while T itself is a service provider to its customers. Along the text, we will adapt this example to the presented cases.

## Consumer-based Paradigm

First, we consider a paradigm allowing a consumer to execute the provided code in order to get some service. In this paradigm, the consumer has complete control over the execution. Thus, the consumer is responsible for searching for a service it prefers, downloading the execution code of the service, and executing the service on its own platform. Despite the fact that the consumer is not often allowed to modify the downloaded service itself, it still has full control over what and how it is executed on its platform. Examples of this scenario are *plugins* and *applets*. Plugins are software packages that extend a client application, e.g., a web browser, with new functionalities necessary to access certain network contents, e.g., audio and video streams. Applets are small, general-purpose applications that the user can interact with, e.g., for data insertion. Moreover, similar approaches are commonly used in other contexts like software bundles and smart-phones applications.

***Threat models:*** The typical threats experienced when running untrusted software provided by a third party are viruses and Trojans. Since the attacking code runs on the target system, illegal accesses may be extremely harmful. However, security mechanisms installed by the service consumer can significantly reduce the chances for threats. For instance, applets can be executed in a sandbox guaranteeing isolation of policies and plug-ins and can be guarded by reference monitors, e.g., access control policies.

In this case, the consumer holds enough security resources (policy, code, and contract) for applying the security analysis and enforcement techniques. Indeed, the consumer can verify contract validity, compare a contract against the policies, and eventually enforce a policy during the execution.

*Example:* Assume H's developer has released a software library implementing H's functionalities. Hence, T's developer just needs to include this code and invoke proper functions in T's workflow. Moreover, the developer can apply several kinds of security analyses and runtime controls to H library.

## Provider-based Paradigm

The most traditional service implementation consists of a remote execution, i.e., on the provider-side, satisfying consumer requests. A consumer, which invokes the service, interacts with the provider by sending the required information. Once the consumer has sent data to the server, the consumer cannot check if the data is used according to consumer's requirements; the consumer can only trust the service contract.

***Threat models:*** In this case, the consumer has no control on the execution. The only information available to the consumer is the contract offered by the service provider. In fact, this contract could be fake or incorrect and no security guarantees are based on it. Thus, most of today's service usages are merely based on the user's perception of provider's reliability or reputation.

*Example:* Many web applications still work in this way. For instance, online document storage and editing services allow users to manage their text documents via their browser without executing any word processing task on their machines. In this case, H is a web application, which can be invoked through a predefined communication protocol. Thus, T's implementation sends messages and collects return values according to the protocol defined by H. Even if H provides T with a contract, T can only compare the protocol flow against it, with no actual control on the internal activities of H.

## Platform-based Paradigm

The lack of security guarantees and the need for service standardization are among the motivations favoring the creation of a service platform. A platform is a collection of facilities, i.e., support services, which aim at mediating the service provision. If these facilities include a service execution environment, service code can be hosted and executed by the platform. Hence, the platform plays the role of a *trusted third party* (TTP) for the consumers. In particular, the consumer trusts the contract provided by the platform. Examples of platforms are SAP's NetWeaver (SAP, 2012) and Microsoft's Azure (Microsoft Corporation, 2012).

***Threat models:*** Similar to the previous case, the consumer has no control on the execution. However, relying on the trust relationship, consumers can assume contract validity and use them for policy verification.

*Example:* Assume T and H are implemented as mobile Apps, running on tablets or smartphones. In this case, T's developer can invoke H by launching it with appropriate parameters. Being hosted in the same location, i.e., the mobile device, each application can rely on platform-provided security mechanisms and check other's permissions.

## Platform-Provider Paradigm

This configuration consists of a platform and a provider that share the execution of a service. Basically, the provider delegates part of the service implementation to the platform while keeping the ownership on some operations. This solution allows the distribution of responsibilities (e.g., orchestration and business logic) and computational load and it is receiving attention from platform and service developers. An instance of this paradigm is Amazon Web Services (AWS).

***Threat models:*** This scenario may be exposed to vulnerabilities similar to those for platform-only and provider-only paradigms. Indeed, the platform support and guarantees cannot be applied to the part of the service running outside. Hence, consumers can only exploit the trust relationship to verify that the platform-implemented part of the service meets the security requirements.

## Consumer-Provider Paradigm

This paradigm can be seen as a hybrid of consumer-based and provider-based paradigms. The consumer is required to execute some piece of code, when the other part is executed by the provider. Such fragments of code may fulfill different purposes like performing operations, which depend on consumer's resources, e.g., file system synchronization, or lightweight support operations, such as animation scripts. In this model, the consumer has complete control only over the part of the code it executes. Nevertheless, in most cases, the client is not allowed to modify the code.

***Threat models:*** In this case, the consumer can only check the compliance between the contract and the policy and control the part of the code that it is going to execute. If the consumer trusts the provider for executing the part of the code that resides in the provider, then the consumer can be guaranteed that the provider will satisfy its security requirements and simply continue to monitor the communication between the pieces of code, i.e., code run by the consumer and the provider. If the consumer does not trust the provider, it has no guarantee that the service will satisfy security requirements.

## Consumer-Platform Paradigm

This paradigm is almost identical to the previous one. The only difference is that the second part of code is executed by the trusted platform, rather than by the provider itself. Thus, the consumer trusts that the execution is correctly controlled as much as it trusts the platform.

***Threat models:*** In this case, the consumer can only check the compliance between the contract and the policy and control the part of the code that it is going to execute. If the consumer trusts the platform for providing the contract and executing the part of the code that resides in the platform, then the consumer can be guaranteed that the platform will satisfy its security requirements and simply continue to monitor the communication between the pieces of code, i.e., code run by the consumer and the platform. If the consumer does not trust the platform, it has no guarantee that the service will satisfy security requirements.

To summarize,

| Paradigms | Code | Contract | Policies |
|---|---|---|---|
| **Consumer** | X | X | X |
| **Provider** | - | - | X |
| **Platform** | - | X | X |
| **Consumer-provider** | X(part.) | - | X |
| **Consumer-platform** | X(part.) | X | X |
| **Platform-provider** | - | X(part.) | X |

relates the three resources of *code*, *contract*, and *policies*, with the six paradigms presented above. In particular, it shows the resources that the service consumer can rely on for security analysis under each configuration. In the table, we assume that the consumer trusts the platform for providing the contract and does not trust the provider either for providing the contract or for executing the code. According to these assumptions, it is worth noticing that in some cases (e.g., in the provider paradigm), even if the consumer receives a contract from the provider, it has no evidence about its validity. Clearly, invalid contracts cannot be exploited for security reasons.

*Table 1. Available security resources for service consumer under various paradigms*

| Paradigms | Code | Contract | Policies |
|---|---|---|---|
| **Consumer** | X | X | X |
| **Provider** | - | - | X |
| **Platform** | - | X | X |
| **Consumer-provider** | X(part.) | - | X |
| **Consumer-platform** | X(part.) | X | X |
| **Platform-provider** | - | X(part.) | X |

## CONTRACT-BASED APPROACHES FOR WEB SERVICE SECURITY

Security-by-Contract and Security-by-Contract-with-Trust are two contract-based approaches that have been developed for dealing with security aspects in a mobile environment. Here, we aim to present how and when they can be applied for guaranteeing security aspects in Service Oriented Architecture.

### Security-by-Contract

The Security-by-Contract (S×C) (Dragoni et al., 2008) paradigm provides a full characterization of the contract-based interaction. It combines different functionalities in an integrated way (see Figure 1). In particular, there is a module for automatically checking the formal correspondence between the code and the contract (Evidence Checking). If the result is negative then the monitor runs to enforce the policies (Policy Enforcement), otherwise a matching between the contract and the policies (Contract & Policy Matching) is performed to establish if the contract is compliant with the policy. In this case, the code is executed without overhead (EXECUTE CODE), otherwise the policies are enforced again (Policy Enforcement). Finally, if the previous checks were positively passed, the code can be executed with no active runtime monitor.

*Figure 1. Security-by-Contract process*

Referring to the working example, after the consumer's request, both H and T services send their contracts. Both contracts are verified by the check evidence function in order to statically probe if the services adhere to their contracts. If the answer is yes, the contracts are compared with the policy by the contract-policy matching functionality. If the policy is satisfied, the service is executed without additional control; otherwise, the policy is enforced during the execution of the service.

### Security-by-Contract-with-Trust

The Security-by-Contract-with-Trust framework (S×C×T) (Costa, Dragoni, Issarny et. al., 2010; Costa, Dragoni, Lazouski et al., 2010) differs from the Security-by-Contract framework, because it substitutes the Evidence Checking functionality with a Trust Management mechanism. Indeed, since services in many cases refuse to release the source code for analysis and validation, only partial description of the service is often available to the consumer and third parties. Thus, there is no way to verify that the published contract correctly describes operations of the service. Thus, Security-by-Contract-with-Trust removes the assumption that verification may be performed and relies on how much a user trusts the correctness of the published contract.

*Figure 2. Security-by-Contract-with-Trust workflow*

Security-by-Contract-with-Trust strategy shown in Figure 2 takes place in two phases: at deploy time, by setting the monitoring state, and at run-time, by applying the contract monitoring procedure (Contract Monitoring), for adjusting the provider trust level (Trust Evaluation). When the service code enters the deployment procedure, i.e., before its first execution, the trust module decides about the trustworthiness of the code by checking its trust level with respect to the fixed threshold. If this check does not pass i.e., the system rejects the provider's trustworthiness, then the code runs within the scope of the policy enforcement mechanism (Policy Enforcement & Contract Monitoring).

While the policy enforcement process prevents the security violations, the monitoring facility keeps under control the possible contract violations. When a running code violates its contract, i.e., it tries to perform in an undeclared way, the system reacts by changing the trust level of the provider. Otherwise, if the trust check succeeds, the system checks whether the contract complies with the security policies (Contract & Policy Matching). In case of compliance, the system executes the code under a contract monitoring setting. It is worth noticing that we assume that both the policies and the contract are written by using the same language. By the way, the policy enforcement and the contract monitoring are performed by two different functionalities that manage them independently.

In order to manage trust, several strategies can be defined according to the considered scenario. We will define a strategy for dealing with Web service paradigms.

Referring to the working example, after the consumer's request, the available services are evaluated according to their trust level. It can be calculated according to several aspects (explained in the next section). If the trust level of the services is accepted, i.e., it is greater than the threshold fixed by the consumer, service's contracts are compared with the policy by the contract-policy matching functionality. If the policy is satisfied, the service is executed without additional control; otherwise, the policy is enforced during the execution of the service.

## Trust Management and Risk Measures

For proper operation of (S×C×T) we need a proper Trust Management approach. Here, we propose two simple trust management approaches, although other trust models may be used (Costa, Dragoni, Lazouski et al., 2010). We assume that a contract may be broken into contract statements (similar to consumer policies). For example, contract and policies consist of rules written in ConSpec language (Aktug & Naliuka, 2008), which is used in the Security-by-Contract framework. Note that if security statements address only specific properties of the service rather than formalize its complete functionality, we are no longer able to guarantee that the service does not do anything except what is stated in the policies. We can only check that what is stated in the policies is fulfilled by the contract.

First, we would like to group the policies and contract statements depending on the information available to a consumer about their correct fulfillment: *Enforced, Monitored, Uncontrolled.*

Naturally, a consumer should not care much about the enforced policy or contract statements since these statements are bound to be satisfied in any case (we assume that enforcement mechanism is trusted). A provider may fail to satisfy monitored contract statements but a consumer gets a report about these violations and may take actions according to the contract (e.g., fine the provider). Finally, a consumer does not know whether uncontrolled statements were satisfied. On the other hand, the consumer may get some indirect information about their failure.

For example, assume that a sample contract statement reads "we do not release contact information to external parties" and the consumer gets an advertisement message from a third party short after the involvement into the interaction with the service. This advertisement may indicate that the address of the consumer was given to that third party, i.e., the contract statement was violated.

Now we are able to propose two trust management approaches for S×C×T. The first approach uses only the monitored contract statements, while the second one sacrifices objectivity for the sake of wider application of the approach. In the following, we call the first approach as "SxCxT", i.e., it is applied with the usual assumptions for S×C×T. The second approach is referred to as "SxCxTR", since the assumption for objective monitoring is relaxed in this case and we use risk for aggregation of trust values. Naturally, enforced statements should not be taken into account, since they cannot fail.

We start with a simple and objective approach. The entity running a service gets objective evidence about the violated contract statements (through contract monitoring) and submits a report to a trust management entity. The trust management entity computes the trust levels for every contract statement "*i*" using the following formula:

$$t_i = {}^{N_i}/_{N},$$

where $N_i$ is the number of times when the contract statement holds and $N$ is the number of times the service was involved in cooperation with consumers. By trust level $t_i$ we mean the probability that a contract statement *"i"* will not be violated. In order to get the trust value of the service, i.e., the probability that the service will not violate any of the declared contract statements, it is necessary to multiply trust values for all contract statements:

$$t = \prod_{\forall i} t_i$$

Now, a new consumer simply needs to check whether the received value is higher than a desired threshold value.

In many Web service paradigms, it is the provider that executes the code. Therefore, there is no reliable way to receive objective evidence about contract violations. Thus, a consumer must rely on secondary information about the execution available to it. This information rarely explicitly indicates that a contract statement was violated by the provider, and must be treated with uncertainty in mind. Thus, in our second trust management approach the subjective trust value is computed as follows, using the subjective information provided by consumers and their reputations (similar to Huynh, Jennings, & Shadbolt (2006)):

$$t_i = \frac{\sum_{\forall k} \left( p_k^{rep} \times p_{i,k}^{conf} \right)}{\sum_{\forall k} \left( p_k^{rep} \right)}$$

where $p_k^{rep}$ is the reputation of the consumer involved in *k*-th interaction; $p_{i,k}^{conf}$ is the confidence of the consumer that a contract statement *"i"* was satisfied. In other words, $p_k^{rep}$ may be seen as the probability that the consumer involved in the *k*-th interaction provides correct and honest information about the outcome of the interaction and $p_{i,k}^{conf}$ is the subjective probability assigned

by the consumer "*k"* reflecting the degree of assurance that a contract statement *"i"* was satisfied. Both reputation and confidence are values between 0 and 1 and have all properties of probabilities.

In order to being more granular, it is possible to consider that the consumer associates a degree of importance to different policies. Hence, satisfaction of some contract statements is more important for a consumer (e.g., confidentiality of credit card information) than others (e.g., keeping contact information private). In this case we can use risk to aggregate trust levels of specific contract statements. Risk is usually computed as follows:

$$risk = \sum_{\forall i} p_i \times c_i$$

where $p_i$ is the probability that a contract statement "*i"* is violated and $c_i$ is the impact on the consumer's business caused by this violation, expressed as a cost.

In S×C×T, the probability is the reverse value to trust. The impact caused by property failure $c_i$ is a business specific value and can be assigned by a consumer only. This impact should be determined according to the policies, which may fail because of the failure of the contract statement *i*.

Thus, the risk value for the service could be computed as follows:

$$risk = \sum_{\forall i} (1 - t_i) \times c_i$$

Now, the consumer may compare the computed values with some threshold or with risk values of other services.

Note that the proposed formulas only outline how exactly the trust rating and risk may be computed. Naturally, these are the basics of the approaches for trust and risk management and advanced techniques (e.g., using time window for statistics collection, using number of deals for increasing credibility of the trust value, introducing bootstrapping strategies), which may be applied here. For example, other computations of trust value based on witness information may be applied for "SxCxTR".

*Example 1:* Assume that service H has two security statements: 1) confidentiality of credit card information is preserved and 2) contact information is kept secret. This service was involved in 120 interactions and failed the first statement only once, while the second statement failed five times.

Thus, if the first approach for trust management "SxCxT" is selected, then:

$t_1 = 119/120 = 0.9916$
$t_2 = 115/120 = 0.9583$
t (trust rating) = 0.95

If now T would like to know the risk level of H, with the assumption, that $c_1$ = \$100000, $c_2$ =\$10000, then:

Risk(H) = 100000*(1-0.9916) + 10000*(1-0.9583) = 840 + 417 = 1257

Now if the trust threshold for invocation of the service is 0.95, then H can be used without enforcement.

*Example 2:* To exemplify the second trust management approach "SxCxTR", we consider a simple example, where only three feedbacks about fulfillment of contract statements of H' (an alternative service) are available from users U1, U2, U3:

The reputation of user U1 is $p_{U1}^{rep}$ = 0.97, U2 is $p_{U2}^{rep}$ = 0.9, and user U3 is $p_{U3}^{rep}$ = 0.5.

The ratings of U1 are: $p_{1,U1}^{rep}$ =1 and $p_{2,U1}^{rep}$ =0.9; ratings of U2 are $p_{1,U2}^{rep}$ =1 and $p_{2,U2}^{rep}$ =1; and the ratings of U3 are $p_{1,U3}^{rep}$ =0.3 and $p_{2,U3}^{rep}$ =0.7.

$$t_1 = \frac{(0.97 \times 1) + (0.9 \times 1) + (0.5 \times 0.3)}{0.97 + 0.9 + 0.5} = 0.8523$$

$$t_2 = \frac{(0.97 \times 0.9) + (0.9 \times 1) + (0.5 \times 0.7)}{0.97 + 0.9 + 0.5} = 0.9379$$

t (trust rating) = 0.7993

If now T would like to know the risk level of H', with the assumption that $c_1$ = \$100000, $c_2$ =\$10000, then:

Risk(H') = 100000*(1-0.8523) + 10000*(1-0.9379) = 14770 + 621 = 15391

Now if the trust threshold for invocation of the service is 0.7993, then H' can be used without enforcement.

## Summary of Results

Application of S×C and its variants depend on the information available to a consumer, i.e., Web service paradigms which are in place in a specific situation (
). In this table the columns are the techniques which may be applied in a specific Web service paradigm (depending on the information a consumer may rely on). The last column shows which proposed approach is the best to be applied for that paradigm.

*Table 2. The techniques available for a service consumer under various paradigms*

| Paradigms | Verification (code) | Matching (contract) | Enforcement/ Monitoring (policies/contract) | Metrics Management (metrics) | Viable Techniques |
|---|---|---|---|---|---|
| **Consumer** | X | X | X | X | S×C |
| **Provider** | - | X | - | X | "SxCxT" |
| **Platform** | X(part.)/X | X | X | X | "SxCxT"/ S×C |

| | | | | | |
|---|---|---|---|---|---|
| **Consumer-provider** | X(part.) | X | X(part.) | X | "SxCxTR" |
| **Consumer-platform** | X(part.)/X | X | X | X | "SxCxT"/ S×C |
| **Platform-provider** | X | X | X(part.) | X | "SxCxTR" |

Table 2 provides a complete picture of necessary conditions for applying the approaches presented in this chapter for guaranteeing security in Web services. In particular, using different frameworks, we are able to guarantee security in most of the considered paradigms as follows:

### S×C Approach

In the consumer paradigm, the platform paradigm, and the consumer-platform paradigm, we can apply the Security-by-Contract framework for guaranteeing security. This can be done because, since we have assumed that the platform is trusted, the consumer (or the platform) has all the resources (code, contract and policy), so it is able to perform all the functionalities of the SxC approach. Furthermore, it is possible to differentiate between two types of platforms depending on whether they can or cannot verify properties.

The reason for doing this is that verification is an operation which requires a large amount of resources. Moreover, verification is often policy specific, and thus, the platforms should either restrict the supported interactions allowing only those where policies are of a specific type (see ANIKETOS project (ANIKETOS, 2013) for example), or assume that some properties cannot be verified. The Security-by-Contract can be used when the platform is going to perform verification. In Table 2, some cells have a value separated by a backslash, e.g., "X(part.)/X" for consumer-platform paradigm in the first column. The first value means whether the corresponding technique may be applied if the platform does not perform code verification, while the second value means that the platform is able to perform the check.

### S×C×T Approach

In the platform paradigm and in the consumer-platform paradigm, security can be guaranteed by using the Security-by-Contract-with-Trust approach whenever the platform is not going to perform verification. Hence, a trust manager substitutes the verification functionality. In both these cases, the contract verification is not performed because the platform and the consumer do not have enough information for performing it. The compliance between the code and the contract is evaluated according to trust measures.

### S×C×TR Approach

In paradigms where the provider is involved, the execution of the code is not completely reliable. In these cases, some risk measures combined with trust ones can be considered in order to strengthen existing guarantees. Hence, we use a variant of the "SxCxT", named "SxCxTR", in which trust and risk measures are combined.

## FUTURE RESEARCH DIRECTIONS

In this chapter, we present our work about security mechanisms for guaranteeing security in Web service composition. As future work, we plan to extend the presented approach in order to consider other measures, such as performance, reliability, dependability, and probability of attack in addition to trust and risk.

Indeed, in the last several years, some research work has been done to consider security in a quantitative way. It is not always possible to guarantee security as defined in the Boolean way. However, it is possible to quantify the level of security that can be guaranteed by defining different enforcement strategies to assure security. As ongoing work, we are studying and analyzing different scenarios to define ways for comparing several strategies according to different quantitative aspects.

## CONCLUSION

In this chapter, we discussed different Web service architecture paradigms and classified them according to their security features. Security features depend on whether a paradigm satisfies the requirements for a certain security technique, e.g., runtime monitoring, to be correctly implemented. Mostly, requirements have to do with the availability of security resources such as policies and contracts. Hence, we introduced categories which take into account the resources according to their reliability.

We presented an overview of the possible threat models affecting each paradigm. In particular, we consider services as code that can be executed by several components of the system, consumer, provider, or platform. We distinguish among several cases, in which services are executed in a centralized way by only one component or in a distributed way by more than one component. We describe our approaches by using a simple working example.

For all of them, we discussed the possibility of applying the Security-by-Contract approach and its extension. When not possible, we proposed extensions that can cope with the existing threats. As a consequence, we detailed the structure of service platforms in terms of the tools they need to be equipped with in order to implement these techniques.

## REFERENCES


Aktug, I., & Naliuka, K. (2008). ConSpec – a formal language for policy specification. *Science of Computer Programming*, *74*(1-2), 2–12.

ANIKETOS. (2013). ANIKETOS project. Retrieved July 22, 2013, from http://www.aniketos.eu

Bauer, L., Ligatti, J., & Walker, D. (2005).Composing security policies with Polymer. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation*, pp. 305–314. ACM.

Carminati, B., Ferrari, E., & Hung P.C.K. (2006).Security Conscious Web Service Composition. In *Proceedings of the IEEE International Conference on Web Services*, pp.489–496.

Costa, G., Dragoni, N., Issarny, V., Lazouski, A., Martinelli, F., Matteucci, I. ... Massacci, F. (2010). Security-by-Contract-with-Trust for mobile devices. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *1*(4), 75–91.

Costa, G., Dragoni, N., Lazouski, A., Martinelli, F., Massacci, F., & Matteucci, I. (2010). Extending Security-by-Contract with quantitative trust on mobile devices. In *Proceedings of the Fourth International Conference on Complex, Intelligent and Software Intensive Systems*, pp. 872–877. IEEE Computer Society.

Dragoni, N., & Massacci, F. (2007). Security-by-contract for web services. In *Proceedings of the ACM Workshop on Secure Web Services*, pp. 90–98.

Dragoni, N., Martinelli, F., Massacci, F., Mori, P., Schaefer, C.,Walter, T., & Vetillar, E. (2008). Security-by-contract (SxC) for software and services of mobile systems. In E. Di Nitto, A.-M. Sassen, P. Traverso, & A. Zwegers (Eds.), *At your service, Service-Oriented Computing from an EU Perspective* (pp. 429-455*)*. MITPress.

Gadyatskaya, O., Massacci, F., & Philippov, A. (2012). Security-by-Contract for the OSGi Platform. In *Proceedings of the IFIP TC11Information Security and Privacy Conference*, pp. 364–375.

Huynh, T.D., Jennings, N.R., & Shadbolt, N.R. (2006). An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, *13*(2), 119-154.

Jones, M., & Hamlen, K.W. (2011). A service-oriented approach to mobile code security. *Procedia Computer Science*, *5*, 531–538.

Microsoft. (2012). Windows Azure. Retrieved from http://www.windowsazure.com

OSGi Alliance. (2012). OSGi – The Dynamic Module System for Java. Retrieved from http://www.osgi.org

Papazoglou, M.P. (2007). *Web Services: Principles and Technology*. Pearson/Prentice Hall.

Phung, P.H., & Sands, D. (2008). Security Policy Enforcement in the OSGi Framework Using Aspect-Oriented Programming. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference*, pp 1076–1082.

SAP. (2012). Netweaver. Retrieved from http://www.sap.com/platform/netweaver/index.epx

Sekar, R., Venkatakrishnan, V.N., Basu, S., Bhatkar, S., & Du Varney, D.C. (2003). Model-carrying code: a practical approach for safe execution of untrusted applications. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 15-28.

## KEY TERMS AND DEFINITIONS

**Contract-based Security:** A security framework that combines static analysis based on the information provided by the contract and run-time enforcement mechanisms in accordance with the policy in order to guarantee that a system is secure.

**Policy:** A policy is a security requirement that a consumer wants to apply to a service execution.

**Secure Web Service:** A Web service whose behavior does not violate the security policy.

**Security-by-Contract:** A security methodology that implements automatic checking of the formal correspondence between the code and the contract, i.e., provide evidence checking.

**Security-by-Contract-with-Trust:** A security methodology that substitutes the evidence checking functionality of security-by-contract with a trust management mechanism, and thus removes the assumption that verification may be performed and relies on how much a user trusts the correctness of the published contract.

**Service Contract:** A service contract is a formal description of the service behavior. Contracts typically describe the service in terms of interaction protocols such as input and output channels, message syntax, parameter types, encryption algorithms, etc.; and service computation such as message semantics, service state transitions, and resources usage.

**Violation:** A violation happens when a service behaves in a way that is not allowed by the policies.