Deriving Secure Network Protocols for Enterprise Services Architectures

Matthias Anlauff*, Dusko Pavlovic*, and Asuman Suenbuel†
*Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94304
†SAP Research Labs
3175 Deer Creek Road
Palo Alto, CA 94304

***Abstract*— Enterprise Service Architectures are emerging as a promising way to compose Web-Services as defined by the W3C consortium, to form complex, enterprise level services. However, due to the fact that each Web-Service composition is also a protocol composition, this composition gets problematic, if security protocol mechanisms are used for the individual Web-Services, because security properties are not preserved under composition. This paper outlines the approach of *protocol derivations* that on the one hand mimics the general engineering practice when combining security features, but on the other hand avoids the problems that can arise during the composition of Web-Services by using well-founded mathematical concepts. The Protocol Derivation Assistant, a tool that supports this approach, is also introduced in this paper.**

I. Introduction

Today's challenges in providing web-based services have manifested itself in the definition of enterprise level frameworks for the efficient and controlled composition of *Web-Services*, a W3C initiative to provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks, see [2]. Enterprises have discovered the potential behind the concepts of Web-Services and have started to provide frameworks that make use of this approach. SAP Netweaver is a prominent example of such a framework with its Enterprise Service Architecture component ([13]). Other companies, especially those providing integration solutions for business customers, are starting to also develop similar framework based on the Web-Services architecture. An atomic Web-Services-based exchange is sketched in Figure 1.

It shows the roles and different phases of a service request and execution. The enterprises use the Web-Services to automate tool connections that often are made by hand, for example, by a human manually typing information that one tool has produced as output into the input forms of one or more other tools that further process the data. The existence of a standard like Web-Services nourishes the hope that these interface help automating these processes. Figures 2 and 3 show an example of the approach taken by the integration frameworks on top of Web-Services: migrating from hard-wired, often manual-input based connections between the different tasks, the Web-Services can be used to automate this process.

This paper focuses on a very important aspect of Enterprise Service Architectures based on Web-Services: network security. In the following section, we will explain the parallels between Enterprise Service Architecture and protocol derivation, and explain the implications of this, namely the problem of non-compositionality of security concepts. After that, we will introduce the concepts of the approach of protocol derivations, and try to explain how this can be used in the context of Enterprise Services.

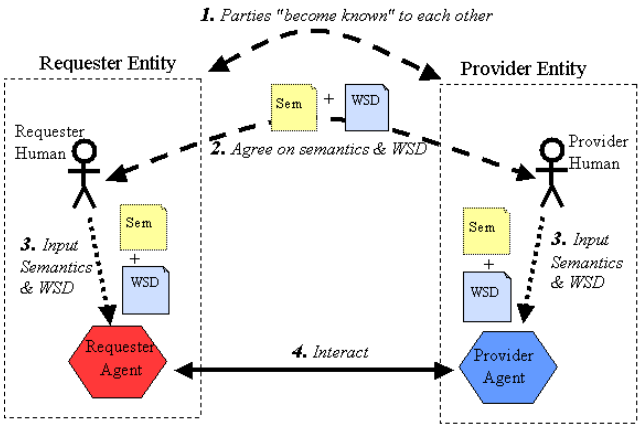

Fig. 1. The General Process of Engaging a Web Service (source: [2])

II. (Secure) Enterprise Service Composition = (Security) Protocol Design

The Enterprise Service Architecture frameworks are based on the idea of composing Web-Services so that new services emerge from that process. Those can in turn become part in the definition of even more complex processes. If we look at this from a communications point of view, we notice that each web-service establishes a data exchange between the integration framework server and the web-service client providing the service. The composition of Web-Services, as promoted by the Enterprise Service Architectures, is therefore also a composition of message exchanges, which is in general referred to as *protocols*. In other words, by composing Web-Services from different servers, one is establishing a communication protocol between the participating agents. In many cases, the individual Web-Service invocations make use of standard encryption mechanisms in order to protect sensitive business data from being compromised. Thus, the

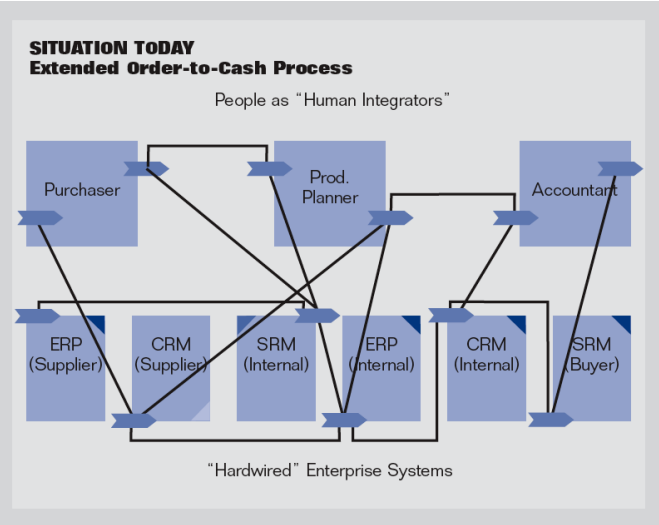

Fig. 2. Service connections without Web-Services-based techniques (source: [13])

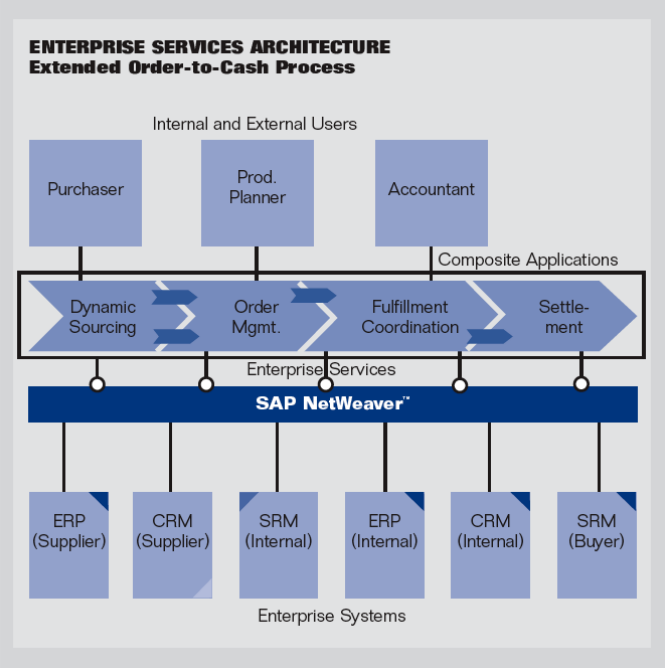

Fig. 3. Using Web-Services-based techniques to automate business processes involving components from different providers (source: [13])

composition of Web-Services involves the composition of security mechanisms in the hope that the security features of one individual Web-Services protocol survive the composition with others. But this is exactly the error, many developers of systems involving security protocol composition make: *the composition of two or more secure (sub-)protocols can lead to insecure protocols*. In other words, security properties of protocols are not compositional. A simple example would the composition of a protocol establishing a secret key between client and server with a protocol sending this key in the clear as part of a message exchanged between the client and the server. Although this fact is known and has been published by researchers, the common practice of protocol development composes security protocols and tries to minimize the possible security holes by extensive testing.

The basis of the work we are presenting in this paper is a framework for deriving security protocols based on mathematically well-founded concepts, as outlined in [4], [6], [11]. The general idea of this approach is it to capture the common practice of security protocol engineering and put it onto a solid foundation. The mentioned approach has been used for detecting a security hole in the GDOI protocol, a internet draft for group communication applications (e.g. pay-per-view). The mentioned approach helped uncovering a security flaw that otherwise remained undetected by an expert group over several years and through half a dozen internet drafts; see [11]. With Enterprise Service Architectures and with it the composition of Web-Services becoming more and more popular, this scenario is very likely being repeated, if the services are composed without having this problem concerning the security features in mind.

In the following, we will give a brief overview of our the basics of our protocol composition approach; after that, we will outline the functionality of the *Protocol Derivation Assistant*, a tool supporting this approach.

III. Brief Overview of Challenge Response Logic

The logic we use is built out of simple axioms describing the conclusions that a principal can draw from her observations of protocol actions, using the known axioms. These axioms are usually of the form: "If A performs certain sequence of actions, then A can conclude that some other sequence of actions by other parties, also occurred". For instance, if A receives a message, then she can conclude that someone must have sent that message; if that message contains a term only B can form, then she knows that B must have been on the path of the message.

The notation will be used as follows. A language of terms t, sent in messages, is assumed. It includes variables for unknown terms or agents, and sufficient typing. The expression (νm) describes the generation of a fresh nonce m. The expression $\langle\langle t\rangle\rangle_A$ describes the action of the agent A sending a message containing a term t, while $\langle t\rangle_A$ denotes A sending just t. The expression $((t))_A$ describes A's receiving a message containing a term t, while $(t)_A$ denotes A's receiving just t. When the source and the destination of a sent or received message are relevant, then these actions can be extended to $\langle\langle t : A \rightarrow B\rangle\rangle_C$ and $((t : A \rightarrow B))_C$, where A and B are respectively the purported source and destination fields, whereas C is the principal who actually performs the action. On the other hand, the subscript "$C <$" (e.g. as in $\langle\langle t\rangle\rangle_{C<}$ etc.) means that C is the *originator* of the relevant message.[1]

Atomic statements are generated over actions in one of the following forms:

- $a < b$ — *"the action a has occurred before b"*,
- $a = b$ — *"the actions a and b are identical"*, and

[1]Formally, $\langle\langle t\rangle\rangle_{C<}$ abbreviates $\exists c.c = \langle\langle t\rangle\rangle_C \wedge \forall b.b = \langle\langle t\rangle\rangle_B \Rightarrow b \geq c$, and $\langle\langle t\rangle\rangle_B$ itself abbreviates $\langle U(t)\rangle_B$, for some transparent term $U(t)$. The details are in [12].

- a — *"the action a has occurred"*.

The equality of actions identifies the expressions describing them. The composite statements are now generated by the usual connectives and the first order quantifiers.

Each statement is annotated by a prefix "A :", denoting the view or knowledge of agent A.

There are two basic axioms that describe the semantics of the actions of sending and receiving messages.

$$(t) \Longrightarrow \exists a.\ a = \langle t \rangle \wedge a < (t) \qquad \text{(rcv)}$$

$$(\nu m)_M \Longrightarrow \forall a_A.\ \big(m \in FV(a) \Rightarrow a > (\nu m)\big) \wedge A \neq M \Rightarrow (\nu m)_M < \langle\langle m \rangle\rangle_M < ((m))_A \leq a_A \qquad \text{(new)}$$

The (rcv) axiom says that if a message is received, it must have been sent. The (new) axiom says that, if a fresh value is generated, then any action involving that fresh value must occur after its generation; moreover, if some principal other than the originator receives a message containing the fresh value, then the originator of the value must have sent a message containing it. All principals are assumed to know these axioms[2].

Axiom (cr) supports the reasoning of the initiator of a challenge-response protocol. It is formalized as follows:

$$A:\ (\nu m)_A\Big(\langle\langle c^{AB} m\rangle\rangle_A < ((r^{AB} m))_A \Longrightarrow \langle\langle c^{AB} m\rangle\rangle_A < ((c^{AB} m))_B < \langle\langle r^{AB} m\rangle\rangle_{B<} < ((r^{AB} m))_A\Big) \qquad \text{(cr)}$$

The expression $c^{AB}m$ denotes a challenge function applied to m, while the expression $r^{AB}m$ denotes a response function applied to m. The axiom can be viewed as a specification of the requirement defining these two functions. It tells that A can be sure that if she issues a message containing a challenge $c^{AB}m$, and receives response containing $r^{AB}m$, then B must be the originator of that response. In other words, B is the only agent who could have transformed $c^{AB}m$ to $r^{AB}m$, given the A's own observed actions.

In the various instances of axiom (cr), functions c and r satisfying the above specification, can be implemented in various ways, e.g. taking B's signature as the response, or B's public key encryption as the challenge. In each case, it will need to be proved that the particular implementation, satisfies the specified requirement.

The logic also contains axioms for composing, refining and transforming protocols. A transformation or refinement usually adds a new property or functionality to the protocol, in which case it comes annotated with an axiom, leading to new conclusions. In authentication protocols, such axioms may expand principal's knowledge about the events in the run of the protocol that he is participating. For example, in the basic challenge-response axiom there is no indication that B *intended* its message as a response to A's particular challenge. This would need to be supplied by some refinement introducing a specific integrity token, such as computing a MAC.

Below, we describe a derivation of a simple challenge and response protocol. As usually, messages are represented by horizontal arrows from one principal to another. A vertical line corresponds to principal's internal change of state. If the principal creates a new value m, this is represented by putting νm next to the appropriate vertical line.

There are several properties of protocols that will be of interest here. One, known as *matching protocol runs*, due to Diffie, van Oorschot, and Wiener [5], says that after two principals complete a protocol successfully, then they both should have the same history of messages sent. Another, due to Lowe [10], known as *agreement*, says that the principals should not only agree on the message history, but who the participants were in the protocol, and which roles each played.

a) Assumptions.: A principal can be honest, and follow the protocol, or dishonest, and perform arbitrary actions. However, it is assumed that neither an honest nor a dishonest principal can compromise the private data used to authenticate him. This means that a response function in a challenge response protocol cannot be delegated. One way to interpret this is that the identity is reduced to possession of authenticating data: whoever has the data, is recognized as a legitimate carrier of the identity.[3] More innocently, the same assumption can be construed as a simplifying convention, introduced to avoid carrying explicit conditions in proofs; but they can be added when needed.

We also tacitly assume strong typing. If a principal, for example, attempts to use data of one type in place of data of another type (e.g. a key in the place of a nonce), this will be detected and the message will be rejected. This again is a relatively strong assumption, but has been shown to lead to no essential loss of analytical power, under certain reasonable provisos [9]).

These assumptions are a matter of convenience, which will undoubtedly be modified in future work. For example, one of the properties of interest for GDOI and many other protocols is perfect forward secrecy, which describes the behavior of the protocol after a master key is compromised.

IV. The Protocol Derivation Assistant

The Protocol Derivation Assistant (PDA) [1] is to support incremental derivations of practical security protocols, together with proofs of their security properties. The protocol derivations start from basic protocol components, and use generic refinements and transformations, just like proof derivations start from axioms, and use generic proof rules and proof transformations. But the guiding idea of PDA is capture incremental methods of protocol design arising in practice, and support them in a framework open for evolution rather than to attempt to reduce security to a predetermined set of formal

[2] $FV(a)$ stands for the set of free variables in a

[3] For instance, no principal can pass his fingerprints or handwriting to others. In cryptographic authentication, this means that no agent can disclose the master keys, used to authenticate him: they are strictly bound to his identity. Finally, there are protocols for which such assumptions do not lead to any loss of the ability to reason about security (the "Machiavellian" protocols of Cervesato et al., [3]), but we do not necessarily limit ourselves to these.

rules. The envisioned results should enable assured protocol derivations, running in parallel with proofs of the desired security properties. Such parallel derivations would be realized using a library of protocol components, generic refinements and transformations, carrying proofs of the relevant security and preservation properties. Alternatively, the same system could cater for attack derivations, running in parallel with the proofs of vulnerabilities.

PDA's user interface consists of a graphical editor; protocols are represented in a similar two-dimensional style as they can be found in text-books and papers. This representation resembles the "desired run" of the protocol, i.e. the flow of information that a protocol designer has in mind when designing a certain protocol. In the following, we will show some basic concepts of the tool and provide screen dumps to show the look-and-feel. PDA is implemented as an Eclipse [7] plugin using the GEF [8] framework.

The basic entity of a protocol derivation using PDA is a protocol. A protocol is a distributed program. To specify a protocol, we must specify a sequence of actions to be executed by every participant in the protocol. Actions divide into two groups: external actions i.e. sending and receiving a message and internal actions i.e. generating nonces, keys, decryption and other local computation. Protocols are entered into Pda via a straightforward interface. Program of every agent is a linear sequence of state descriptions (Stads), with transitions being either send, receive, or agent step (internal computation). For example, two-party protocol CR[I,R](c,r) is shown in Figure 4.

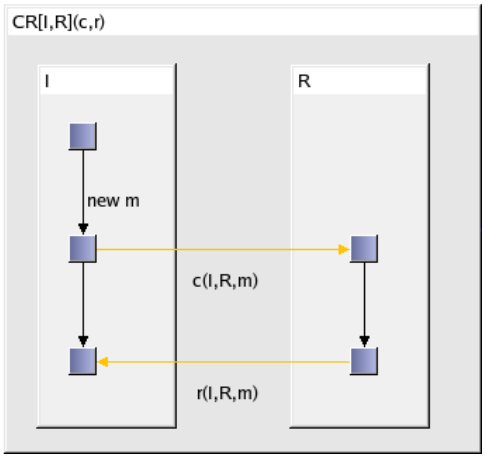

Fig. 4. The basic Challenge-Response protocol as entered into the PDA

Agent I (initiator) has the following sequence of actions: generate a new nonce m, send the message $c(I, R, m)$ to R, receive a message $r(I, R, m)$ from R. Agent R (responder) receives a message $c(I, R, m)$ from I and sends a message $r(I, R, m)$ to I. Messages exchanged in the protocol contain concrete cryptographic primitives such as encryption, signature and hash, and function variables such as c and r in this example. When a protocol contains function variables, we say that it is a protocol template. Protocol header $CR[I, R](c, r)$ should be read: CR is a two party protocol (agents are named I and R), using abstract function variables c and r.

A. Protocol Derivation (1): Instantiation

Simplest derivation step is protocol instantiation. Using the "Create New Instance" command, some (or all) of the function variables can be refined to concrete primitives or other function variables. For example, we can get a one-way challenge-response protocol using nonces and signature as an instance of $CR[I, R](c, r)$ with the following instantiation:

$$SCR[I, R] = CR[I, R](c(x, y, z) = z, r(x, y, z) = Sig(y, z, x))$$

In the resulting protocol SCR[I,R], agent I sends a fresh nonce to R who replies with his signature over the nonce and I's identity, as shown in Figure 5.

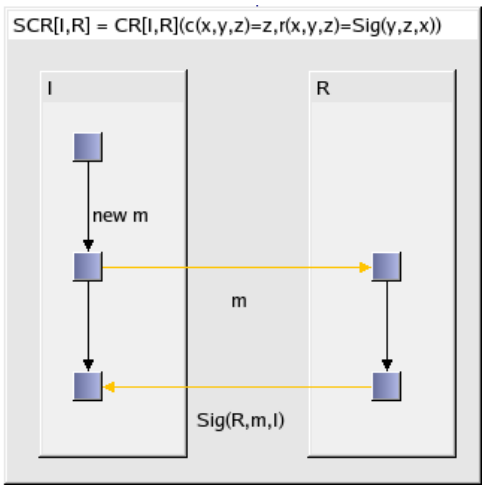

Fig. 5. The instantiated CR protocol

B. Protocol Derivation (2): Composition

As mentioned earlier, protocols can be combined using composition; we will outline this concept using a simple sequential composition operator. In a resulting protocol, program of each agent is a concatenation of their programs from the two protocols. For example, protocol $Two\_CR[I, R](c, r, c0, r0)$ is obtained by sequentially composing $CR[I, R](c, r)$ with it's reverse copy $Reverse\_CR[I, R](c0, r0)$ (which is in turn obtained by simple instantiation $Reverse\_CR[I, R](c0, r0) = CR[R, I](c0, r0)$).

Protocols can be composed using a concept that is called *Rules* in PDA. Rules can be regarded as graphical macros that allow for the definition of customized composition operators.

C. Proofs

Alongside the development of the protocols, the proof obligations that are affiliated with each of them are recorded and given to an automatic theorem prover. Using the basic challenge-response logic as basis, PDA provides not only a graphical tool for designing and deriving protocols, but also the ability to prove whether certain security properties survived the composition with another (sub-)protocol.

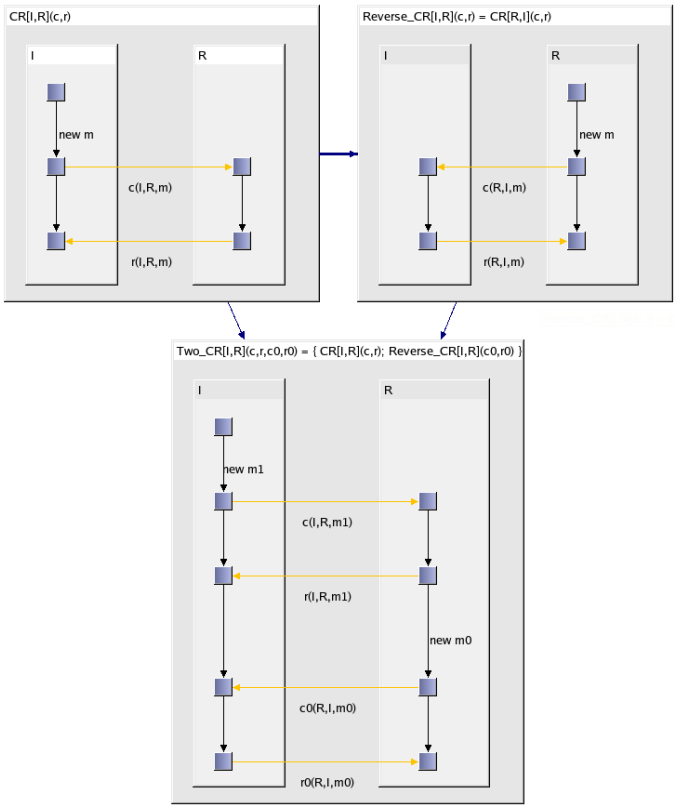

Fig. 6. Sequential composition of Protocols

V. Summary

In this paper, we have outlined the problems of Enterprise Service Architectures with respect to network security caused by the general lack of composability of message exchanges as an result of composing Web-Services. We have outlined and briefly explained our derivational approach to protocol development as well as the Protocol Derivation Assistant, a software system that supports the derivational design approach of the underlying protocol logic. In the near future, we will expand our research work in this area and we are planning to conduct more experiments, especially in the area of business applications.

References

[1] Matthias Anlauff and Dusko Pavlovic. The protocol derivation assistant, 2005. URL `http://www.kestrel.edu/software/pda`.

[2] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard (eds.). Web services architecture. W3C Working Group Note, 2004. URL `http://www.w3c.org/TR/ws-arch`.

[3] I. Cervesato, S. Meadows, and P. Syverson. Dolev-Yao is no better than Machiavelli. In *First Workshop on Issues in the Theory of Security: WITS'00*, July 8-9 2000.

[4] A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of ACM Workshop on Formal Methods in Computer Security 2003*, pages 109–125, Washington, DC, October 2003. ACM.

[5] W. Diffie, P. C. van Oorschot, and M.l J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography*, 2:107–125, 1992.

[6] N.A. Durgin, J.C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):667–721, 2003.

[7] Eclipse-Team. Eclipse, 2005. URL `http://www.eclipse.org`.

[8] GEF-Team. The Graphical Editing Framework (GEF), 2005. URL `http://www.eclipse.org/projects/gef`.

[9] J. Heather, S. Schneider, and G. Lowe. How to prevent type flaw attacks on security protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 2000.

[10] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 31–43. IEEE Computer Society Press, 1997.

[11] Catherine Meadows and Dusko Pavlovic. Deriving, attacking and defending the gdoi protocol. In Peter Ryan, Pierangela Samarati, Dieter Gollmann, and Refik Molva, editors, *Proceedings of ESORICS 2004*, volume 3193, pages 53–72. Springer Verlag, 2004.

[12] D. Pavlovic and C. Meadows. Deriving authenticity and integrity as order of actions. Technical report, Kestrel Institute technical report, January 2004.

[13] SAP. Enterprise services architecture an introduction. SAP White Papers, 2005. URL `http://www.sap.com/solutions/netweaver/pdf/-WP_Enterprise_Services_Architecture_Intro.pdf`.