Analysing Security Properties using Refinement

Claus Pahl

School of Computer Applications
Dublin City University
Dublin 9, Ireland
eMail: cpahl@compapp.dcu.ie
fax: ++353 +1 700 5442

Abstract. Security properties are essential in open and distributed environments with high dependability requirements. An approach to development and analysis of safety- and security-critical systems based on refinement as the central concept can offer an integrated solution. We analyse the Online Certificate Status Protocol (OCSP), showing how to use refinement as an interference analysis tool for secure communication protocols and intruders.

1 Motivation

Dependable systems require a high level of safety and security. Security is becoming increasingly important since more and more systems are deployed in open distributed and networked environments, and are therefore subject to external threads. Security is concerned with aspects such as *confidentiality* – the prevention of disclosure of information to unauthorised users –, *integrity* – the prevention of unauthorised modification of information –, and *authentication* – the proven establishment of the identity of another agent in a distributed system.

Several methods have been suggested for the development and analysis of protocols in distributed and networked systems [1, 2, 3, 4]. We present an integrated framework based on refinement as the key concept for development and analysis. We use the Online Certificate Status Protocol (OCSP) as our case study. OCSP [5] is an information acquisition protocol used in public key infrastructures (PKI), i.e., systems that manage cryptographic keys in security-critical environments. Its main purpose is to check the revocation status of a certificate (the key concept to guarantee authentication in networked systems).

2 The Notation

The Command Language. A protocol consists of several communicating agents. Each of these agents has the capacity to perform a set of basic actions, such as sending or receiving messages:

- $snd(R, M_1, \dots, M_n)$ is the send-action. R denotes the receiver; the elements M_1 to M_n form the message content.

- $rcv(S, x_1, \ldots, x_n)$ is the receive-action. S denotes the sender; x_1 to x_n are names that will be substituted by the actual message elements.

All identifiers, such as the messages, are names. The basic actions can be combined by a set of classical combinators: sequential composition $a_1; a_2$, non-deterministic choice $a_1 + a_2$, parallel composition $a_1|a_2$, and iteration a^*. A protocol is usually described as a parallel composition of agents. Agents composed in parallel can interact if a send and a receive operation match – determined by matching sender/receiver IDs and the length of the message list.

OCSP is a request/response protocol. It provides a generic message envelope for a variety of services. A set of standardised request/response-types exist. One of these is ORS (Online Revocation Status) providing information about the revocation status of a certificate. Requests contain the elements service ID, a certificate (the certificate to be checked), and an optional signature (the client signs the message). The response contains a response status (successful, malFormedRequest, etc.), a certificate status (good, revoked, unknown), and an optional signature (the server signs the message). A protocol P between client C and server S is defined as the parallel composition $P \equiv C|S$ with:

$$\begin{aligned} C &:= snd_C(S, serv, cert, sig); rcv_C(S, respStat, certStat, sig) \\ S &:= rcv_S(C, serv, cert, sig); snd_S(C, respStat, certStat, sig) \end{aligned}$$

The Specification Language. The command language is embedded into a specification language based on modal logic [6] – a logic with a notion of state suitable for the description of reactive systems. We derive our variant from the modal μ-calculus [7] – a branching time temporal logic. Actions are explicit in this logic. Besides the usual logical combinators, two modal combinators are provided:

- $[a]\phi$ is based on the box- or always operator, describing that if a terminates it does so in a state satisfying ϕ.
- $\langle a \rangle \phi$ is based on the diamond- or eventually operator, describing that a can terminate in a state satisfying ϕ.

The syntax for formulas is $\phi ::= true|false|\alpha|\phi_1 \wedge \phi_2|\phi_1 \wedge \phi_2|\ldots|[a]\phi|\langle a \rangle \phi$ where $\alpha \in P$ is a predicate. Names can be compared for equality. Additionally, a set of security predicates are provided:

- The predicate $Knows_A(X)$ describes that an agent A 'knows' about an item X, i.e., A has access to X either by creating it or by having received it.
- The predicate $unMod_A(X)$ is true if A has received an item X that has not been modified during transmission, i.e., is unmodified.
- $Auth_A(B)$ expresses that an agent A has authenticated another agent B.

A formula $\phi \rightarrow [a]\ \psi$ or $\phi \rightarrow \langle a \rangle\ \psi$ with precondition ϕ and postcondition ψ shall be called a contract for a.

The semantic structures in which this logic is interpreted are Kripke transition systems – a combination of Kripke structures and labelled transition systems [6, 8]. A Kripke transition system (KTS) is a quadruple $M = (S, Act, \rightarrow, I)$ with a set of states S, a set of actions Act, a transition relation $\rightarrow \subseteq S \times Act \times S$, and an interpretation I. The structure M shall be defined over a set of predicates P. The elements in the sets Act and P shall be indexed by agents.

We interpret closed formulas ϕ as subsets of S whose states make ϕ true. We define $[\![true]\!]_\rho := S$ and $[\![false]\!]_\rho := \emptyset$; standard logical combinators are defined as usual. Predicates $\alpha \in P$ are defined via environments $\rho : P \rightarrow \mathcal{P}S$, i.e. $[\![\alpha]\!]_\rho := \rho(\alpha)$.

$$\begin{aligned}
[\![[a]\phi]\!]_\rho &:= \{s \mid \text{for all } t \text{ holds } (s,a,t) \in \rightarrow\} \\
[\![\langle a\rangle\phi]\!]_\rho &:= \{s \mid \text{exists } t \text{ such that } (s,a,t) \in \rightarrow\}
\end{aligned}$$

We use this language to specify security properties. Server S needs to authenticate client C – we assume additional signing and verifying actions.

$$[rcv_S(C, serv, cert, sig)]\ Knows_S(K_C^{pub}) \rightarrow vrf_S(K_C^{pub}, sig)$$

S validates C's signature sig using C's public key. This shall be abbreviated by the predicate $Auth_S(C)$. The OCSP definition states a success criterion: the client needs to accept a server response only if C authenticates S and S replies with respect to the request certificate.

$$[snd_C(S, serv, cert); rcv_C(S, rStat, cert', cStat)]\ cert = cert' \wedge Auth_C(S)$$

Other properties could be specified, e.g. an explicit exclusion of a replay attack, or properties related to the availability of data.

3 Interference Analysis using Refinement

We define a refinement relation for actions based on the notion of contracts. Let $C_{a_1} \equiv \phi \rightarrow [a_1]\ \phi'$ and $C_{a_2} \equiv \psi \rightarrow [a_2]\ \psi'$ be contracts.

C_{a_1} is *refined* by C_{a_2}, or $C_{a_1} \sqsubseteq C_{a_2}$, if $\phi \rightarrow \psi \wedge \psi' \rightarrow \phi'$ [1].

This follows classical definitions [9, 10] – here in a different semantical framework.

A central idea of our approach to security analysis of protocols is explicit intruder modelling. The intruder is added to the protocol specification through refinement. The original protocol specification P is the ideal protocol. It states expected security properties. In a refinement $P \sqsubseteq P|I$ the intruder I is added using parallel composition $P|I$. Either these properties are preserved or are violated. In the latter case, we have found a security flaw. In the former case, $P \sqsubseteq P|I$ holds, i.e., the intruder cannot interfere with the protocol. The protocol can be analysed by systematically varying the intruder behaviour. We look at different security aspects separately.

Theorem 1. *Refinements $A|B \sqsubseteq A|B|I$ are guaranteed if the constraints given by the refinement laws – see Figure 1 – are satisfied.*

[1] Additionally, we need to exclude some trivial cases such as $C_{a_2} \equiv true \rightarrow [a_2]\ false$, which result in proper refinements, but would also constitute a security problem.

Confidentiality $A|B \sqsubseteq A|B|I$ if $Conf(A|B|I) \rightarrow Conf(A|B)$ with

$$Conf(A|B) \quad = Knows_A(X_1, \ldots, X_k) \wedge Knows_B(Y_1, \ldots, Y_m)$$
$$Conf(A|B|I) = Knows_A(X'_1, \ldots, X'_{k'}) \wedge Knows_B(Y'_1, \ldots, Y'_{m'}) \wedge Knows_I(Z'_1, \ldots, Z'_{n'})$$

Integrity $A|B \sqsubseteq A|B|I$ if $Int(A|B|I) \rightarrow Int(A|B)$ with

$$Int(A|B) \quad = unMod_A(X_1, \ldots, X_k) \wedge unMod_B(Y_1, \ldots, Y_m)$$
$$Int(A|B|I) = unMod_A(X_1, \ldots, X_k) \wedge unMod_B(Y_1, \ldots, Y_m)$$

Authentication $A|B \sqsubseteq A|B|I$ if $Authenticate(A|B|I) \rightarrow Authenticate(A|B)$ with

$$Authenticate(A|B) \quad = Auth_A(X_1, \ldots, X_k) \wedge Auth_B(Y_1, \ldots, Y_m)$$
$$Authenticate(A|B|I) = Auth_A(X_1, \ldots, X_k) \wedge Auth_B(Y_1, \ldots, Y_m) \wedge Auth_I(Z_1, \ldots, Z_n)$$

Fig. 1. Refinement Laws for Security Properties

We need axiomatisations – called laws – for the basic actions. Confidentiality laws shall be addressed first. An agent knows about an item by either creating it or receiving it. The agent remembers it after executing an action.

$$Knows_A(X) \rightarrow [snd_A(B, X)]\ Knows_A(X)$$

The interaction between agents is defined in terms of the $Knows$-predicate:

$$Knows_A(M) \rightarrow [snd_A(B, M)|rcv_B(A, x)]\ Knows_B(M)$$

which is our definition of the reaction between two agents. The message M replaces the input variable x for agent B.

An integrity law is $[snd_A(B, M)|rcv_B(A, x)|I]\ Knows_B(M) \rightarrow unMod_B(M)$. Authentication laws can also be formulated.

4 Protocol Analysis

Intruder models such as the Dolev-Yao model assign certain capabilities to an intruder. With a reduced set of actions, we could assume that the intruder capability is to non-deterministically iterate receive- and send-actions, i.e., $(rcv_I(X, M) + snd_I(Y, N))^*$. Varying the intruder behaviour allows us to analyse different forms of attacks. An example is the man-in-the-middle attack. If a client does not sign a message, the intruder I can act as the client and communicate with the server:

$$snd_I(S, serv, cert)$$

After this $\neg Auth_S(C)$ holds, which violates the authentication property:

$$[rcv_S(C, serv, cert)]\ Auth_S(C)$$

Our analysis tool is refinement, i.e., we need to check whether $C|S \sqsubseteq C|S|I$ holds. We get $C|S \not\sqsubseteq C|S|I$ since the authentication $Auth_S(C)$ was required. Signatures should be used, which would also help us to guarantee integrity.

Another attack type is the replay attack: the intruder I plays the role of the server and replays previously sent messages to the current client:

$$rcv_I(S, M); rcv_I(C, M'); snd_I(C, M)$$

assuming that M and M' are messages about the same certificate. This behaviour again matches the general intruder capacity. The remedy is to include nonces – randomly created values – which is not required by the OCSP definition.

The flaws that can be detected with our approach have been described in the literature. However, a formal account does not exist.

5 Concluding Remarks

An important aspect is tool support for this form of analysis. We have based our semantic framework on the modal μ-calculus [7] to be able to consider model checking [8]. A model, i.e., a Kripke transition system, $M = (S, Act, \rightarrow, I)$ over a set of predicates P can be constructed for a particular purpose, e.g. the confidentiality analysis. Another direction that could be pursued is testing. Aichernig [11] addresses test case generation based on refinement and abstraction.

References

[1] N.A. Durgin and J.C. Mitchell. Analysis of Security Protocols. In M. Broy and R. Steinbruggen, editors, *Calculational System Design*, pages 369–395. IOS Press, 1999.

[2] L.C. Paulson. Proving Properties of Security Protocols by Induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83. 1997.

[3] R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In H. Orman and C. Meadows, editors, *DIMACS Workshop on Design and Formal Verification of Security Protocols*. DIMACS, Rutgers University, 1997. http://dimacs.rutgers.edu/Workshops/Security.

[4] M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: the spi Calculus. *Information and Computation*, 148:1–70, 1999.

[5] IETF Internet Engineering Task Force Network Working Group. Online Certificate Status Protocol - OCSP, 2001. http://www.ietf.org/rfc/rfc2560.txt.

[6] Dexter Kozen and Jerzy Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 789–840. Elsevier Science Publishers, 1990.

[7] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[8] M. Müller-Olm, D. Schmidt, and B. Steffen. Model Checking – a Tutorial Introduction. In *Proc. 6th Static Analysis Symposium*. Springer-Verlag, LNCS 1694, 1999.

[9] R.J.R. Back and J. von Wright. *The Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.

[10] C. Morgan. *Programming from Specifications 2e*. Addison-Wesley, 1994.

[11] B.K. Aichernig. Test-case calculation through abstraction. In J.N. Oliveira and P. Zave, editors, *Proc. FME'2001 Symposium Formal Methods Europe*. Springer-Verlag, LNCS Series No. 2021, 2001.