SATMC: a SAT-based Model Checker for Security-critical Systems

Alessandro Armando[1,2], Roberto Carbone[2], and Luca Compagna[3]

[1] DIBRIS, University of Genova, Genova, Italy
[2] Security & Trust, FBK, Trento, Italy
[3] Product Security Research, SAP AG, Sophia Antipolis, France
armando@fbk.eu carbone@fbk.eu luca.compagna@sap.com

Abstract. We present SATMC 3.0, a SAT-based bounded model checker for security-critical systems that stems from a successful combination of encoding techniques originally developed for planning with techniques developed for the analysis of reactive systems. SATMC has been successfully applied in variety of application domains (security protocols, security-sensitive business processes, and cryptographic APIs) and for different purposes (design-time security analysis and security testing). SATMC strikes a balance between general purpose model checkers and security protocol analyzers as witnessed by a number of important success stories including the discovery of a serious man-in-the-middle attack on the SAML-based Single Sign-On (SSO) for Google Apps, an authentication flaw in the SAML 2.0 Web Browser SSO Profile, and a number of attacks on PKCS#11 Security Tokens. SATMC is integrated and used as back-end in a number of research prototypes (e.g. the AVISPA Tool, Tookan, the SPaCIoS Tool) and industrial-strength tools (e.g. the Security Validator plugin for SAP NetWeaver BPM).

1 Introduction

With the convergence of the social, cloud, and mobile paradigms, information and communication technologies are affecting our everyday personal and working live to unprecedented depth and scale. We routinely use online services that stem from the fruitful combination of mobile applications, web applications, cloud services, and/or social networks. Sensitive data handled by these services often flows across organizational boundaries and both the privacy of the users and the assets of organizations are often at risk.

Solutions (e.g. security protocols and services) that aim to securely combine the ever-growing ecosystem of online services are already available. But they are notoriously difficult to get right. Many security-critical protocols and services have been designed and developed only to be found flawed years after their deployment. These flaws are usually due to the complex and unexpected interactions of the protocols and services as well as to the possible interference of malicious agents. Since these weaknesses are very difficult to spot by traditional verification techniques (e.g. manual inspection and testing), security-critical systems are a natural target for formal method techniques.

SATMC is a SAT-based bounded model checker for security-critical systems that combines encoding techniques developed for planning [26] with others developed for the analysis of reactive systems [15]. The approach reduces the problem of determining whether the system violates a security goal in $k > 0$ steps to the problem of checking the satisfiability of a propositional formula (the SAT problem). Modern SAT solvers can tackle SAT problems of practical relevance in milliseconds. Since its first release in 2004 [8], SATMC has been enhanced to support Horn clauses and first-order LTL formulae. This makes SATMC 3.0 able to support the security analysis of distributed systems that exchange messages over a wide range of secure channels, are subject to sophisticated security policies, and/or aim at achieving a variety of security goals.

Since (both honest and malicious) agents can build and exchange messages of finite, but arbitrary complexity (through concatenation and a variety of cryptographic primitives), most security-critical, distributed systems are inherently infinite state. For this reason general purpose model checkers (e.g. SPIN [24], NuSMV [19])—which assume the input system to be finite state—are not suited for the analysis of a large and important set of security-critical systems (e.g. cryptographic protocols and APIs). Special purpose tools (most notably, security protocol analyzers, e.g. CL-AtSe [27], OFMC [13], Proverif [16]) are capable of very good performance and support reasoning about the algebraic properties of cryptographic operators. SATMC complements security protocol analyzers by supporting a powerful specification language for communication channels, intruder capabilities, and security goals based on first-order LTL.

SATMC strikes a balance between general purpose model checkers and security protocol analyzers. SATMC has been successfully applied in variety of application domains (namely, security protocols, security-sensitive business processes, and cryptographic APIs) and for different purposes (e.g., design-time security analysis and security testing). SATMC is integrated and used as a backend in a number of research prototypes (the AVISPA Tool [2], Tookan [18], the AVANTSSAR Platform [1], and the SPaCIoS Tool [28]) and industrial-strength tools (the Security Validator plugin for SAP NetWeaver BPM[4]). The effectiveness of SATMC is witnessed by the key role it played in the discovery of:

- a flaw in a "patched" version of the protocol for online contract signing proposed by Asokan, Shoup, and Waidner (ASW) [3],
- a serious man-in-the-middle attack on the SAML-based SSO for Google Apps [6] and, more recently, an authentication flaw in the SAML 2.0 Web Browser SSO Profile and related vulnerabilities on actual products [5],
- a number of attacks on the PKCS#11 Security Tokens [18], and
- a flaw in a two-factor and two-channel authentication protocol [7].

As shown in Figure 1, the applicability of SATMC in different domains is enabled by domain-specific connectors that translate the system and the property specifications into ASLan [12], a specification language based on set-rewriting, horn clauses, and first-order LTL which is amenable to formal analysis. As shown

[4] `http://scn.sap.com/docs/DOC-32838`

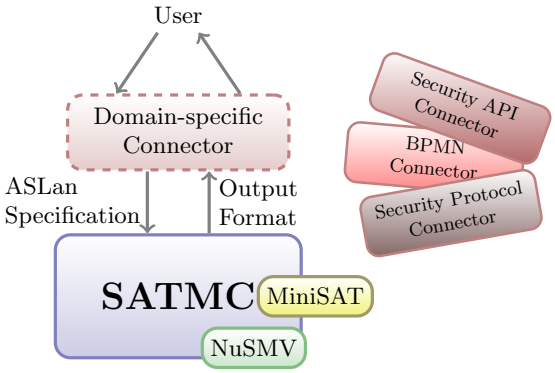

Fig. 1: High-level Overview

in the same figure, SATMC leverages NuSMV to generate the SAT encoding for the LTL formulae and MiniSAT [22] to solve the SAT problems.

Structure of the paper. In the next section we present some success stories related to the application domains wherein SATMC has been so far employed. In Section 3 we provide the formal framework and in Section 4 we illustrate how the ASLan specification language can be used to specify security-critical systems, the abilities of the attackers, and the security goals. In Section 5 we present the bounded model checking procedure implemented in SATMC and the architecture of the tool, and we conclude in Section 6 with some final remarks.

2 Success Stories

SATMC has been successfully used to support the security analysis and testing in a variety of industry relevant application domains: security protocols, business processes, and security APIs.

Security protocols. Security protocols are communication protocols aiming to achieve security assurances of various kinds through the usage of cryptographic primitives. They are key to securing distributed information infrastructures, including—and most notably—the Web. The SAML 2.0 Single Sign-On protocol [21] (SAML SSO, for short) is the established standard for cross-domain browser-based SSO for enterprises. Figure 2 shows the prototypical use case for the SAML SSO that enables a Service Provider (SP) to authenticate a Client (C) via an Identity Provider (IdP): C asks SP to provide the resource at URI (step 1). SP then redirects C to IdP with the authentication request $AReq(ID, SP)$, where ID uniquely identifies the request (steps 2 and 3). IdP then challenges C to provide valid credentials (gray dashed arrow). If the authentication succeeds, IdP builds and sends C a digitally signed authentication assertion ($\{AA\}_{\mathrm{K}_{\mathrm{IdP}}^{-1}}$) embedded into an HTTP form (step 4). This form also includes some script

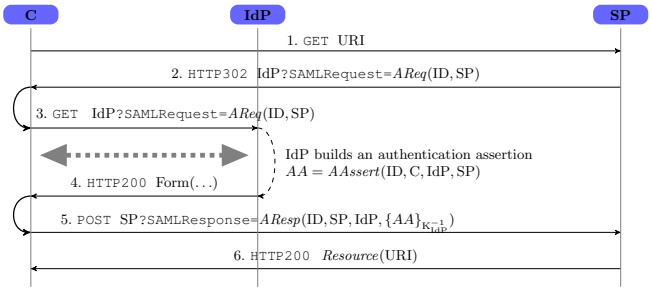

Fig. 2: SAML SSO Protocol

that automatically posts the message to SP (step 5). SP checks the assertion and then deliver the requested resource to C (step 6). Upon successful execution of the protocol, C and SP are mutually authenticated and the resource has been confidentially delivered to C. To achieve this, SAML SSO—as most of the application-level protocols—assumes that the communication between C and SP as well as that between C and IdP is carried over unilateral SSL/TLS communication channels. It must be noted that even with secure communication channels in place, designing and developing application-level security protocols such as the SAML SSO remains a challenge. These protocols are highly configurable, are described in bulky natural language specifications, and deviations from the standard may be dictated by application-specific requirements of the organization.

The SAML-based SSO for Google Apps in operation until June 2008 deviated from the standard in a few, seemingly minor ways. By using SATMC, we discovered an authentication flaw in the service that allowed a malicious SP to mount a severe man-in-the-middle attack [6]. We readily informed Google and the US-CERT of the problem. In response to our findings Google developed a patch and asked their customers to update their applications accordingly. A vulnerability report was then released by the US-CERT.[5] The severity of the vulnerability has been rated High by NIST.[6] By using the SATMC we also discovered an authentication flaw in the prototypical SAML SSO use case [5]. This flaw paves the way to launching Cross-Site Scripting (XSS) and Cross-Site Request Forgery (XSRF) attacks, as witnessed by a new XSS attack that we identified in the SAML-based SSO for Google Apps. We reported the problem to OASIS which subsequently released an *errata* addressing the issue.[7]

We also used SATMC at SAP as a back-end for security protocol analysis and testing (AVANTSSAR [1] and SPaCIoS [28]) to assist development teams in the design and development of the SAP NetWeaver SAML Single Sign-On (SAP NGSSO) and SAP OAuth 2.0 solutions. Overall, more than one hundred different protocol configurations and corresponding formal models have been

[5] http://www.kb.cert.org/vuls/id/612636
[6] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-3891
[7] http://tools.oasis-open.org/issues/browse/SECURITY-12

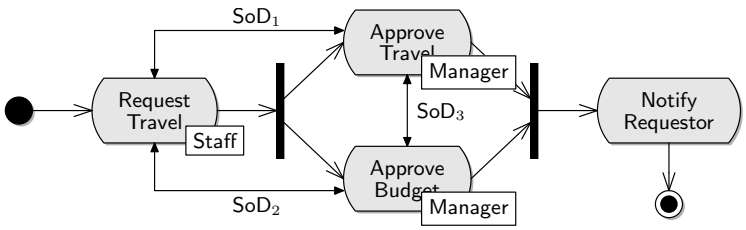

Fig. 3: A simple travel approval process with annotated security requirements

analyzed, showing that both SAP NGSSO and SAP OAuth2 services are indeed well designed.

All in all, SATMC has been key to the analysis of various security protocols of industrial complexity, leading to the discovery of a number of serious flaws, including a vulnerability in a "patched" version of the optimistic fair-exchange contract signing protocol developed by Asokan, Shoup, and Weidner [3] and a protocol for strong authentication based on the GSM infrastructure [7].

Business Processes. A Business Process (BP) is a workflow of activities whose execution aims to accomplish a specific business goal. BPs must be carefully designed and executed so to comply with security and regulatory compliance requirements. Figure 3 illustrates a simple example of a BP for travel request approval: a staff member may issue a travel request. Both the reason and the budget of the travel must be approved by managers. Afterwards, the requesting user is notified whether the request is granted or not. The BP shall ensure a number of authorization requirements: only managers shall be able to approve the reason and budget of travel requests. Moreover, a manager should not be allowed to approve her own travel request (Separation of Duty). Finally, the manager that approves the travel reason should not get access to the details of the travel budget (and vice versa) to perform her job (Need-to-Know Principle). Checking whether a given BP of real-world complexity complies with these kind of requirements is difficult.

SATMC has been used to model check BPs against high-level authorization requirements [10]. Moreover, SATMC lies at the core of a Security Validation prototype for BPs developed by the Product Security Research unit at SAP. This prototype can integrate off-the-shelf Business Process Management (BPM) systems (e.g. SAP Netweaver BPM and Activity) to support BP analysts in the evaluation of BP compliance. It enables a BP analyst to easily specify the security goals and triggers SATMC via a translation of the BP workflow, data, security policy and goal into ASLan. As soon as a flaw is discovered, it is graphically rendered to the analyst [20, 11].

Security APIs. A Security API is an Application Program Interface that allows untrusted code to access sensitive resources in a secure way. Figure 4 shows a few methods of the Java interface[8] for the security API defined by the PKCS#11

[8] `http://javadoc.iaik.tugraz.at/pkcs11_wrapper/1.2.15/iaik/pkcs/pkcs11/wrapper/PKCS11.html`

```
// modifies the value of one or more object attributes
void C_SetAttributeValue(long hSes, long hObj, CK_ATTRIBUTE[] pAtt)
// initializes a decryption operation
void C_DecryptInit(long hSes, CK_MECHANISM pMechanism, long hKey)
// decrypt encrypted data
byte[] C_Decrypt(long hSes, byte[] pEncryptedData)
// wraps (i.e., encrypts) a key
byte[] C_WrapKey(long hSes, CK_MECHANISM pMechanism, long hWrappingKey, long hKey)
```

Fig. 4: PKCS#11: Java interface

standard [25]. The Java interface and its implementation allow to access the PKCS#11 modules of smart cards or other hardware security modules (HSM) where sensitive resources (e.g., cryptographic keys, pin numbers) can be stored. These resources can be associated with attributes (cf. `C_SetAttributeValue`) stating, e.g., whether they can be extracted from the device or not, whether a certain key can be used to wrap (encrypt) another key, etc. For instance, if an object is set to be non-extractable, then it cannot be reset and become extractable again. More in general, changes to the attribute values and access to sensitive resources must comply the policy that the security API is designed to enforce and this must hold for any possible sequence of invocation of the methods offered by the API (`C_Decrypt`, `C_WrapKey`, etc.).

SATMC lies at the core of Tookan [18], a tool capable to automatically detect and reproduce policy violations in commercially available cryptographic security tokens, exploiting vulnerabilities in their RSA PKCS#11 based APIs. Tookan can automatically reverse-engineer real PKCS#11 tokens, deduce their functionalities, construct formal models of the API for the SATMC model checker, and then execute the attack traces found by SATMC directly against the actual token. Tookan has been able to detect a variety of severe attacks on a number of commercial tokens (e.g., SecurID800 by RSA, CardOS V4.3 B by Siemens) [23]. Cryptosense[9] is a spin-off recently established on top of the success of Tookan.

3 Formal Framework

A *fact* is an atomic formula of a first-order language. We consider a language $\mathcal{L}$ defined as the smallest set of formulae containing facts and equalities between (first-order) terms as atomic propositions as well as the formulae built with the usual propositional connectives ($\neg$, $\vee$, ...), first-order quantifiers ($\forall$ and $\exists$), and temporal operators (**F** for "some time in the future", **G** for "globally", **O** for "some time in the past", ...). A formula is *closed* if and only if all variables in it are bound by some quantifier.

Let $\mathcal{V}$ be the set of variables in $\mathcal{L}$; let $\mathcal{F}$ and $\mathcal{T}$ be the (possibly infinite) sets of ground (i.e. variable-free) facts and terms of $\mathcal{L}$ respectively. A *model* is 4-uple $M = \langle \mathcal{I}, \mathcal{R}, \mathcal{H}, \mathcal{C} \rangle$, where

- $\mathcal{I} \subseteq \mathcal{F}$ is the *initial state*,

[9] `http://cryptosense.com/`

- $\mathcal{R}$ is a set of *rewrite rules*, i.e. expressions of the form $(L \xrightarrow{r(v_1,\ldots,v_n)} R)$, where L and R are finite sets of facts, r is a *rule name* (i.e. an n-ary function symbol uniquely associated with the rule) for $n \geq 0$, and $v_1, \ldots, v_n$ are the variables in L; it is required that the variables occurring in R also occur in L.
- $\mathcal{H}$ is a set of *Horn clauses*, i.e. expressions of the form $(h \xleftarrow{c(v_1,\ldots,v_n)} B)$, where h is a fact, B is a finite set of facts, c is a *Horn clause name* (i.e. an n-ary function symbol uniquely associated with the clause) for $n \geq 0$, and $v_1, \ldots, v_n$ are the variables occurring in the clause.
- $\mathcal{C} \subseteq \mathcal{L}$ is a set of closed formulae called *constraints*.

An *assignment* is a total function from $\mathcal{V}$ into $\mathcal{T}$, i.e. $\sigma : \mathcal{V} \to \mathcal{T}$. Assignments are extended to the facts and terms of $\mathcal{L}$ in the obvious way. Let $S \subseteq \mathcal{F}$, *the closure of* $S \subseteq \mathcal{F}$ *under* $\mathcal{H}$, in symbols $\lceil S \rceil^{\mathcal{H}}$, is the smallest set of facts containing S and such that for all $(h \xleftarrow{c(\ldots)} B) \in \mathcal{H}$ and $\sigma : \mathcal{V} \to \mathcal{T}$ if $B\sigma \subseteq \lceil S \rceil^{\mathcal{H}}$ then $h\sigma \in \lceil S \rceil^{\mathcal{H}}$. A set of facts S *is closed under* $\mathcal{H}$ iff $\lceil S \rceil^{\mathcal{H}} = S$. We interpret the facts in $\lceil S \rceil^{\mathcal{H}}$ as the propositions holding in the state represented by S, all other facts being false (closed-world assumption). Let $(L \xrightarrow{\rho} R) \in \mathcal{R}$ and $\sigma : \mathcal{V} \to \mathcal{T}$. We say that *rule (instance)* $\rho\sigma$ *is applicable in state* S if and only if $L\sigma \subseteq \lceil S \rceil^{\mathcal{H}}$ and when this is the case $S' = \mathrm{app}_{\rho\sigma}(S) = (S \setminus L\sigma) \cup R\sigma$ is the state resulting from the execution of $\rho\sigma$ in S. A *path* π is an alternating sequence of states and rules instances $S_0\rho_1 S_1 \rho_2 \ldots$ such that $S_i = \mathrm{app}_{\rho_i}(S_{i-1})$, for $i = 1, 2, \ldots$. If, additionally, $S_0 \subseteq \mathcal{I}$, then we say that the path is *initialized*. Let $\pi = S_0\rho_1 S_1 \ldots$ be a path; we define $\pi(i) = S_i$ and $\pi_i = S_i\rho_{i+1}S_{i+1}\ldots$; $\pi(i)$ and π_i are the i-th state of the path and the suffix of the path starting with the i-th state, respectively. We assume that paths have infinite length. (This can be always obtained by adding stuttering transitions to the system.) Let π be an initialized path of M. An LTL formula ϕ *is valid on* π *under* σ, in symbols $\pi \models_\sigma \phi$, if and only if $\pi_0 \models_\sigma \phi$, where $\pi_i \models_\sigma \phi$ is inductively defined as follows.

$\pi_i \models_\sigma f$	iff	$f\sigma \in \lceil \pi(i) \rceil^{\mathcal{H}}$ (f is a fact)
$\pi_i \models_\sigma t_1 = t_2$	iff	$t_1\sigma$ and $t_2\sigma$ are the same term
$\pi_i \models_\sigma \neg\phi$	iff	$\pi_i \not\models_\sigma \phi$
$\pi_i \models_\sigma \phi \vee \psi$	iff	$\pi_i \models_\sigma \phi$ or $\pi_i \models_\sigma \psi$
$\pi_i \models_\sigma \mathbf{F}(\phi)$	iff	there exists $j \geq i$ such that $\pi_j \models_\sigma \phi$
$\pi_i \models_\sigma \mathbf{G}\phi$	iff	for all $j \geq i$ $\pi_j \models_\sigma \phi$
$\pi_i \models_\sigma \mathbf{O}\phi$	iff	there exists $0 \leq j \leq i$ such that $\pi_j \models_\sigma \phi$
$\pi_i \models_\sigma \exists x.\phi$	iff	there exists $t \in \mathcal{T}$ such that $\pi_i \models_{\sigma[t/x]} \phi$

where $\sigma[t/x]$ is the assignment that associates x with t and all other variables y with $\sigma(y)$. The semantics of the remaining connectives and temporal operators, as well as of the universal quantifier, is defined analogously. Let $M_1 = \langle \mathcal{I}_1, \mathcal{R}_1, \mathcal{H}_1, \mathcal{C}_1 \rangle$ and $M_2 = \langle \mathcal{I}_2, \mathcal{R}_2, \mathcal{H}_2, \mathcal{C}_2 \rangle$. The *parallel composition of* M_1 *and* M_2 is the model $M_1 \| M_2 = \langle \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{H}_1 \cup \mathcal{H}_2, \mathcal{C}_1 \cup \mathcal{C}_2 \rangle$. Let $M = \langle \mathcal{I}, \mathcal{R}, \mathcal{H}, \mathcal{C} \rangle$ be a model and $\phi \in \mathcal{L}$. We say that ϕ *is valid in* M, in symbols $M \models \phi$, if and only if $\pi \models_\sigma \phi$ for all initialized paths π of M and all assignments σ such that $\pi \models_\sigma \psi$ for all $\psi \in \mathcal{C}$.

Table 1: Facts and their informal meaning

	Fact	Meaning
Domain Independent	$\texttt{sent}(s, b, a, m, c)$	s sent m on c to a pretending to be b
	$\texttt{rcvd}(a, b, m, c)$	m (supposedly sent by b) has been received on c by a
	$\texttt{contains}(d, ds)$	d is member of ds
	$\texttt{ik}(m)$	the intruder knows m
Protocols	$\texttt{state}_r(j, a, ts)$	a plays r, has internal state ts, and can execute step j
Business Processes	$\texttt{pa}(r, t)$	r has the permission to perform t
	$\texttt{ua}(a, r)$	a is assigned to r
	$\texttt{executed}(a, t)$	a executed t
	$\texttt{granted}(a, t)$	a is granted to execute t
APIs	$\texttt{attrs}(as)$	security token has attributes as

Legenda: s, a, b: agents; m: message; c: channel; r: role; j: protocol step; ts: list of terms; t: task d: data; ds: set of data; o: resource object; as: set of attributes

4 Modeling Security-critical Systems

We are interested in model checking problems of the form:

$$M_S \| M_I \models G \tag{1}$$

where $M_S = \langle \mathcal{I}_S, \mathcal{R}_S, \mathcal{H}_S, \mathcal{C}_S \rangle$ and $M_I = \langle \mathcal{I}_I, \mathcal{R}_I, \mathcal{H}_I, \mathcal{C}_I \rangle$ are the model of the security-sensitive system and of the intruder respectively and G is an LTL formula expressing the security properties that the combined model must enjoy.

Table 1 presents an excerpt of the facts (2nd column) used in the different application domains (1st column). Their informal meaning is explained in the rightmost column. Some facts have a fixed meaning: `ik` models the intruder knowledge, `sent` and `rcvd` are used to model communication, and `contains` expresses set membership. Other facts are domain specific: $\texttt{state}_r(j, a, ts)$ models the state of honest agents in security protocols; $\texttt{pa}(r, t)$, $\texttt{ua}(a, r)$, $\texttt{executed}(a, t)$ and $\texttt{granted}(a, t)$ are used to represent the security policy and task execution in business processes; finally $\texttt{attrs}(as)$ is used to model attribute-value assignments to resource objects in security APIs. Here and in the sequel we use typewriter font to write facts and rules with the additional convention that variables are capitalized (e.g. `C`, `AReq`), whereas constants and function symbols begin with a lower-case letter (e.g. `hReq`).

Formal modeling of Security-critical Systems. In the case of security protocols, the initial state $\mathcal{I}_S$ contains a state-fact $\texttt{state}_r(1, a, ts)$ for each agent a. In case of business processes, the initial state specifies which tasks are ready for execution as well as the access control policy (e.g. the user-role and the role-permission assignment relations). In case of security APIs, the initial state specifies some attribute-value assignments.

Let us now consider an example of rewriting rules in $\mathcal{R}_S$. The reception of message 2 by the client and the forwarding of message 3 in Figure 2 are modeled

by the following rewriting rule:

$$\begin{aligned}&\mathtt{rcvd}\,(\mathtt{C},\mathtt{SP},\mathtt{hRsp}(\mathtt{c30x},\mathtt{IdP},\mathtt{AReq}),\mathtt{C}_{\mathtt{SP2C}})\,.\\ &\mathtt{state}_{\mathtt{c}}(2,\mathtt{C},[\mathtt{SP},\ldots,\mathtt{C}_{\mathtt{C2IdP}}]) \xrightarrow{\mathtt{send}_2(\mathtt{C},\ldots,\mathtt{C}_{\mathtt{C2IdP}})} \mathtt{state}_{\mathtt{c}}(3,\mathtt{C},[\mathtt{Areq},\mathtt{SP},\ldots,\mathtt{C}_{\mathtt{C2IdP}}])\,.\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\mathtt{sent}\,(\mathtt{C},\mathtt{C},\mathtt{IdP},\mathtt{hReq}(\mathtt{get},\mathtt{IdP},\mathtt{AReq}),\mathtt{C}_{\mathtt{C2IdP}})\end{aligned}$$

The clauses in $\mathcal{H}_S$ support the specification, e.g., of access control policies. For instance, in case of business processes, the Role-based Access Control (RBAC) model can be naturally and succinctly specified as follows:

$$\mathtt{granted}(\mathtt{A},\mathtt{T}) \xleftarrow{\mathtt{grant}(\mathtt{A},\mathtt{R},\mathtt{T})} \mathtt{ua}(\mathtt{A},\mathtt{R}),\mathtt{pa}(\mathtt{R},\mathtt{T})$$

Moreover, security-critical systems often rely on assumptions on the behavior of the principals involved (e.g. progress, availability). These assumptions can be specified by adding suitably defined LTL formulae to $\mathcal{C}_S$. Some examples are provided in the last two rows of Table 2.

Formal modeling of the Intruder. A model that corresponds to the Dolev-Yao intruder is given by $M_{DY} = \langle \emptyset, \mathcal{R}_{DY}, \mathcal{H}_{DY}, \emptyset \rangle$, where the rewrite rules in $\mathcal{R}_{DY}$ model the ability to overhear, divert, and intercept messages and the clauses in $\mathcal{H}_{DY}$ model the inferential capabilities, e.g. the ability to decrypt messages when the key used for encryption is known to the intruder as well as that to forge new messages. The model of the Dolev-Yao intruder M_{DY} can be complemented by a model $M_{I'} = \langle \mathcal{I}_{I'}, \mathcal{R}_{I'}, \mathcal{H}_{I'}, \mathcal{C}_{I'} \rangle$, where $\mathcal{I}_{I'}$ contains the facts representing the initial knowledge in the scenario considered, $\mathcal{R}_{I'}$ and $\mathcal{H}_{I'}$ model additional, domain specific behaviors of the intruder, and $\mathcal{C}_{I'}$ may instead constrain the otherwise allowed behaviors. Thus the model of the intruder stems from the parallel combination of M_{DY} and $M_{I'}$, i.e. $M_I = (M_{DY} \| M_{I'})$.

For instance, the behavior of the intruder for security APIs can be extended using the following rule that models the Java method `C_Decrypt` of Figure 4:

$$\begin{aligned}&\mathtt{ik}\,(\mathtt{crypt}(\mathtt{K},\mathtt{R}))\,.\mathtt{ik}\,(\mathtt{hand}(\mathtt{N},\mathtt{inv}(\mathtt{K})))\,.\\ &\quad\mathtt{attrs}(\mathtt{KeyAttrs})\,.\mathtt{contains}(\mathtt{attr}(\mathtt{decrypt},\mathtt{true},\mathtt{N}),\mathtt{KeyAttrs})\\ &\qquad\qquad\qquad\qquad\xrightarrow{\mathtt{decrypt\_key\_asym}(\mathtt{KeyAttrs},\mathtt{K},\mathtt{R},\mathtt{N})} \mathtt{ik}(\mathtt{R})\,.\,LHS\end{aligned}$$

where LHS abbreviates the left hand side of the rule. This rule states that if *(i)* the intruder knows a cipher-text encrypted with key `K` and the handler `N` of the key `inv(K)`, and *(ii)* `N` has the attribute decryption set to true (i.e., the key associated with that handler can be used for decryption), then the decrypt method can be applied and the intruder can retrieve the plain-text `R`. (Notice that knowing the handler does not mean to know the key associated to the handler.)

As a further instance, a transport layer protocol such as SSL/TLS can be abstractly characterized by including suitable formulae in $\mathcal{C}_{I'}$. To illustrate consider the first four rows of Table 2, where, here and in the sequel, $\forall(\psi)$ stands

Table 2: LTL constraints

Property	LTL Formula
$confidential\_to(c,p)$	$\mathbf{G}\,\forall(\texttt{rcvd}(A,B,M,c) \Rightarrow A = p)$
$authentic\_on(c,p)$	$\mathbf{G}\,\forall(\texttt{sent}(RS,A,B,M,c) \Rightarrow (A = p \wedge RS = p))$
$weakly\_confidential(c)$	$\mathbf{G}\,\forall((\texttt{rcvd}(A,B,M,c) \wedge \mathbf{F}\,\texttt{rcvd}(A',B',M',c)) \Rightarrow A = A')$
$resilient(c)$	$\mathbf{G}\,\forall(\texttt{sent}(RS,A,B,M,c) \Rightarrow \mathbf{F}\,\texttt{rcvd}(B,A,M,c))$
$progress(a,r,j)$	$\mathbf{G}\,\forall(\texttt{state}_r(j,a,ES) \Rightarrow \mathbf{F}\,\neg\texttt{state}_r(j,a,ES))$
$availability(a,c)$	$\mathbf{G}\,\forall(\texttt{rcvd}(a,P,M,c) \Rightarrow \mathbf{F}\,\neg\,\texttt{rcvd}(a,P,M,c))$

for the universal closure of the formula ψ. The formula in the 3rd row formalizes the property of a weakly confidential channel, i.e., a channel whose output is exclusively accessible to a single, yet unknown, receiver. In our model this amounts to requiring that, for every state S, if a fact $\texttt{rcvd}(a,b,m,c) \in S$, then in all the successor states the `rcvd` facts with channel c must have a as recipient (see corresponding LTL formula in the 3rd row). More details can be found in [6, 4].

Security Goals. For security protocols, besides the usual secrecy and authentication goals, our property specification language $\mathcal{L}$ allows for the specification of sophisticated properties involving temporal operators and first-order quantifiers. For instance, a fair exchange goal for the ASW protocol discussed in [4] is:

$$\mathbf{G}\,\forall n_O.\forall n_R.(\texttt{hasvc}(r,txt,n_O,n_R) \Rightarrow \mathbf{F}\,\exists n_R.\texttt{hasvc}(o,txt,n_O,n_R))$$

stating that if an agent r has a valid contract, then we ask o to possess a valid contract relative to the same contractual text txt and secret commitment n_O.

Finally, the separation of duty property SoD_3 exemplified in Figure 3 can be expressed as the following LTL formula:

$$\mathbf{G}\,\forall(\texttt{executed}(\texttt{A},\texttt{approve\_travel}) \Rightarrow \mathbf{G}\,\neg\texttt{executed}(\texttt{A},\texttt{approve\_budget}))$$

This goal states that if an agent `A` has executed the task `approve_travel` then he should not execute the task `approve_budget`.

5 SAT-based Model Checking of Security-critical Systems

A high-level overview of the architecture of SATMC 3.0 is depicted in Figure 5. SATMC takes as input the specification of the system M_S, (optionally) a specification of the custom intruder behavior $M_{I'}$, a security goal $G \in \mathcal{L}$, and checks whether $M_S \| M_I \models G$ via a reduction to a SAT problem. Since $M_I = (M_{DY} \| M_{I'})$, this boils down to checking whether $M_S \| M_{DY} \| M_{I'} \models G$. The **DY Attacker** module computes and carries out a number of optimizing transformations on $M_S \| M_{DY}$. These transformations specialize the (otherwise very prolific) rules and clauses of M_{DY} and produce a model $M_{S\text{-}DY}$ which

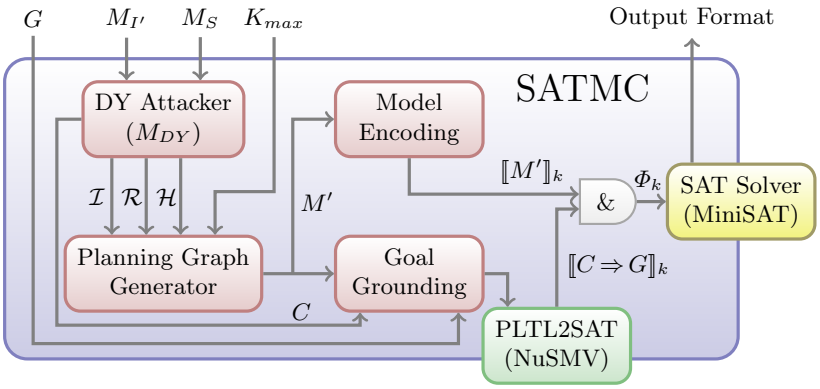

Fig. 5: SATMC Internals

is easier to analyze than (yet equivalent to) $M_S \| M_{DY}$ [9]. The module finally computes and yields the model $M_{S\text{-}DY} \| M_{I'}$ which is equivalent to $M_S \| M_I$ with $M_I = (M_{DY} \| M_{I'})$. Thus the problem of checking whether $M_S \| M_{DY} \| M_{I'} \models G$ is reduced to checking whether $M_{S\text{-}DY} \| M_{I'} \models G$.

Let $M = (M_{S\text{-}DY} \| M_{I'}) = \langle \mathcal{I}, \mathcal{R}, \mathcal{H}, \mathcal{C} \rangle$. SATMC now builds a propositional formula Φ_k such that every truth-value assignment satisfying Φ_k corresponds to a counterexample of $M \models G$ of length k and vice versa, with $k \leq k_{max}$ (where the upper bound k_{max} is an additional input to SATMC). The formula Φ_k is given by the conjunction of the propositional formulae (*i*) $[\![M']\!]_k$ encoding the unfolding (up to k-times) of the transition relation associated with $M' = \langle \mathcal{I}, \mathcal{R}, \mathcal{H}, \emptyset \rangle$, and (*ii*) $[\![C \Rightarrow G]\!]_k$, where C is a conjunction of the formulae in $\mathcal{C}$, encoding the set of attack traces of length k satisfying the constraints C. The module **Model Encoding** takes as input M' and generates $[\![M']\!]_k$, while $[\![C \Rightarrow G]\!]_k$ is generated by the cascade of the **Goal Grounding** and **PLTL2SAT** modules.

The **PLTL2SAT** module leverages existing Bounded Model Checking (BMC) techniques [14]. However, the techniques available in the literature assume that *(i)* a propositional encoding of the transition relation of M is available and that *(ii)* the goal formula belongs of propositional LTL. Both assumptions are violated by the model checking problem we consider, since *(i)* the models we consider are defined over a set of states which is not bounded *a priori* and *(ii)* first-order LTL formulae are allowed. In [9] we showed that the first problem can be tackled by computing a planning graph of the problem of depth k. A planning graph [17] (module **Planning Graph Generator**) is a succinct representation of an over-approximation of the states reachable in k steps. The planning graph is also key to reducing the first-order LTL formula $(C \Rightarrow G)$ to an equivalent (in a sense that will be defined later) propositional LTL formula (via the **Goal Grounding** module) which is then reduced to SAT using techniques developed for bounded model checking of reactive systems [14] (via the **PLTL2SAT** module).

A SAT solver is finally used to check the satisfiability of the formula Φ_k (module **SAT Solving**). Although not shown in Figure 5, SATMC carries out an

iterative deepening strategy on k. Initially k is set to 0, and then it is incremented till an attack is found (if any) or k_{max} is reached. If this is the case no attack traces of length up to k_{max} exist. Notice that it is possible to set k_{max} to ∞, but then the procedure may not terminate (i.e. it is a semi-decision procedure). It is worth noticing that even though the planning graph may represent spurious execution paths, the encoding in SAT is precise and thus false positives are not returned by SATMC.

The Planning Graph. A *planning graph* is a sequence of layers Γ_i for $i = 0, \ldots, k$, where each layer Γ_i is a set of facts concisely representing the set of states $\|\Gamma_i\| = \{S : S \subseteq \Gamma_i\}$. The construction of a planning graph for M' goes beyond the scope of this paper and the interested reader is referred to [9] for more details. For the purpose of this paper it suffices to know that *(i)* Γ_0 is set to the initial state of M', *(ii)* if S is reachable from the initial state of M' in i steps, then $S \in \|\Gamma_i\|$ (or equivalently $S \subseteq \Gamma_i$) for $i = 0, \ldots, k$, and *(iii)* $\Gamma_i \subseteq \Gamma_{i+1}$ for $i = 0, \ldots, k-1$, i.e. the layers in the planning graph grow monotonically.

Encoding the Model. The first step is to add a time-index to the rules and facts to indicate the state at which the rules apply or the facts hold. Facts and rules are thus indexed by 0 through k. If p is a fact, a rule, or a Horn clause and i is an index, then p^i is the corresponding time-indexed propositional variable. If $\boldsymbol{p} = p_1, \ldots, p_n$ is a tuple of facts, rules, or Horn clauses and i is an index, then $\boldsymbol{p}^i = p_1^i, \ldots, p_n^i$ is the corresponding time-indexed tuple of propositional variables. The propositional formula $[\![M']\!]_0$ is $I(\boldsymbol{f}^0, \boldsymbol{hc}^0)$, while $[\![M']\!]_k$, for $k > 0$, is of the form:

$$I(\boldsymbol{f}^0, \boldsymbol{hc}^0) \wedge \bigwedge_{i=0}^{k-1} T_i(\boldsymbol{f}^i, \boldsymbol{\rho}^i, \boldsymbol{hc}^i, \boldsymbol{f}^{i+1}, \boldsymbol{hc}^{i+1}) \tag{2}$$

where $\boldsymbol{f}$, $\boldsymbol{\rho}$, and $\boldsymbol{hc}$ are tuples of facts, rules, and Horn clauses, respectively. The formula $I(\boldsymbol{f}^0, \boldsymbol{hc}^0)$ encodes the initial state whereas the formula $T_i(\boldsymbol{f}^i, \boldsymbol{\rho}^i, \boldsymbol{hc}^i, \boldsymbol{f}^{i+1}, \boldsymbol{hc}^{i+1})$ encodes all the possible evolutions of the system from step i to step $i+1$. The encoding of the system follows the approach proposed in [9], adapted from an encoding technique originally introduced for AI planning, extended to support Horn clauses.

Grounding First-order LTL Formulae. Planning graphs are also key to turn any first-order LTL formula ψ into a propositional LTL formula ψ_0 such that if π is an execution path of M' with k or less states that violates ψ_0, then π violates also ψ, and vice versa. This allows us to reduce the BMC problem for any first-order LTL formula ψ to the BMC for a propositional LTL formula ψ_0 (module **Goal Grounding**) which can in turn be reduced to SAT by using the techniques available in the literature. The functionalities of the module **PLTL2SAT** are currently given by the NuSMV model checker, used as a plugin by SATMC. From the key properties of the planning graph described in Section 5 it is easy to see that if a fact does not occur in Γ_k, then it is false in all states reachable from the initial state in k steps and this leads to the following fact.

Fact 1 *Let $\psi \in \mathcal{L}$, Γ_i for $i = 0, \ldots, k$ be a planning graph for M', and p a fact such that $p \notin \Gamma_k$. Then, if π is an execution path of M' with k or less states that violates ψ then π violates also $\psi[\bot/p]$ (and vice versa), where $\psi[\bot/p]$ is the formula obtained from ψ by replacing all occurrences of p with $\bot$.*

To illustrate consider the problem of generating a propositional version of:

$$\exists \mathtt{A}.\, \mathbf{F}(\neg\, \mathbf{O}\, s(\mathtt{A}, \mathtt{b}) \wedge r(\mathtt{b}, \mathtt{A})) \tag{3}$$

when $\Gamma_k = \{s(\mathtt{a1}, \mathtt{b}), r(\mathtt{b}, \mathtt{a1}), r(\mathtt{b}, \mathtt{a2})\}$. If the variable $\mathtt{A}$ ranges over the (finite) set of constants $D_{\mathtt{A}} = \{\mathtt{a1}, \ldots, \mathtt{an}\}$, then we can replace the existential quantifier with a disjunction of instances of the formula in the scope of the quantifier, where each instance is obtained by replacing the quantified variable, namely $\mathtt{A}$, with the constants in $D_{\mathtt{A}}$: $\mathbf{F}(\neg\, \mathbf{O}\, s(\mathtt{a1}, \mathtt{b}) \wedge r(\mathtt{b}, \mathtt{a1})) \vee \ldots \vee \mathbf{F}(\neg\, \mathbf{O}\, s(\mathtt{an}, \mathtt{b}) \wedge r(\mathtt{b}, \mathtt{an}))$ By repeatedly using Fact 1, this formula can be rewritten into: $\mathbf{F}(\neg\, \mathbf{O}\, s(\mathtt{a1}, \mathtt{b}) \wedge r(\mathtt{b}, \mathtt{a1})) \vee \ldots \vee \mathbf{F}(\neg\, \mathbf{O} \bot \wedge \bot)$and finally be simplified to

$$\mathbf{F}(\neg\, \mathbf{O}\, s(\mathtt{a1}, \mathtt{b}) \wedge r(\mathtt{b}, \mathtt{a1})) \vee \mathbf{F}(r(\mathtt{b}, \mathtt{a2})) \tag{4}$$

Even if the resulting formula is compact thanks to the simplification induced by the planning graph, the instantiation step, namely the replacement of the existential quantifier with a disjunction of instances, can be very expensive or even unfeasible (if the domain of the existentially quantified variable is not bounded).

A better approach is to let the instantiation activity be driven by the information available in the planning graph. This can be done by generating instances of the formula by recursively traversing the formula itself in a top down fashion. As soon as an atomic formula is met, it is matched against the facts in Γ_k and if a matching fact is found then the formula is replaced with the ground counterpart found in Γ_k and the corresponding matching substitution is carried over as a constraint. The approach is iterated on backtracking in order to generate all possible instances and when no (other) matching fact in Γ_k for the atomic formula at hand is found, then we replace it with $\bot$.

To illustrate this, let us apply the approach to (3). By traversing the formula we find the atomic formula $s(\mathtt{A}, \mathtt{b})$ which matches with $s(\mathtt{a1}, \mathtt{b})$ in Γ_k with matching substitution $\{\mathtt{A} = \mathtt{a1}\}$. The atomic formula $s(\mathtt{A}, \mathtt{b})$ is instantiated to $s(\mathtt{a1}, \mathtt{b})$ and the constraint $\mathtt{A} = \mathtt{a1}$ is carried over. We are then left with the problem of finding a matching fact in Γ_k for the formula $r(\mathtt{b}, \mathtt{A})$ with $\mathtt{A} = \mathtt{a1}$, i.e. for $r(\mathtt{b}, \mathtt{a1})$. The matching fact is easily found in Γ_k and we are therefore left with the formula ($i1$) $\mathbf{F}(\neg\, \mathbf{O}\, s(\mathtt{a1}, \mathtt{b}) \wedge r(\mathtt{b}, \mathtt{a1}))$ as our first instance. On backtracking we find that there is no other matching fact for $r(\mathtt{b}, \mathtt{a1})$ which is then replaced by $\bot$, thereby leading to the instance ($i2$) $\mathbf{F}(\neg\, \mathbf{O}\, s(\mathtt{a1}, \mathtt{b}) \wedge \bot)$. By further backtracking we find that there is no other matching fact for $s(\mathtt{A}, \mathtt{b})$ in Γ_k and therefore we generate another instance by replacing $s(\mathtt{A}, \mathtt{b})$ with $\bot$ and carry over the constraint $\mathtt{A} \neq \mathtt{a1}$. The only fact in Γ_k matching $r(\mathtt{b}, \mathtt{A})$ while satisfying the constraint $\mathtt{A} \neq \mathtt{a1}$ is $r(\mathtt{b}, \mathtt{a2})$. We are then left with the formula ($i3$) $\mathbf{F}(\neg\, \mathbf{O} \bot \wedge r(\mathtt{b}, \mathtt{a2}))$ as our third instance. No further matching fact for $r(\mathtt{b}, \mathtt{a2})$ exists. The procedure therefore turns the first-order LTL formula (3) into the disjunction of ($i1$), ($i2$), and ($i3$), which can be readily simplified to (4).

6 Conclusions

We presented SATMC 3.0, a SAT-based Model Checker for security-critical systems. SATMC successfully combines techniques from AI planning and for the analysis of reactive systems to reduce the problem of determining the existence of an attack of bounded length violating a given security goal to SAT. SATMC supports the specification of security policies as Horn clauses and of security assumptions and goals as first-order LTL formulae. Its flexibility and effectiveness is demonstrated by its successful usage within three industrial relevant application domains (security protocols, business processes, and security APIs) and its integration within a number of research prototypes and industrial-strength tools. SATMC 3.0 can be downloaded at `http://www.ai-lab.it/satmc`.

Acknowledgments. We are grateful to Luca Zanetti for his contribution in the design and implementation of the Goal Grounding and PLTL2SAT modules. This work has partially been supported by the FP7-ICT Project SPaCIoS (no. 257876), by the PRIN project "Security Horizons" (no. 2010XSEMLC) funded by MIUR, and by the Activity "STIATE" (no. 14231) funded by the EIT ICT-Labs.

References

1. A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuellar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. Oheimb, G. Pellegrino, S. E. Ponta, M. Rocchetto, M. Rusinowitch, M. Torabi Dashti, M. Turuani, and L. Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *TACAS'12*, volume 7214 of *LNCS*, pages 267–282. 2012.
2. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *CAV'05*. Springer-Verlag, 2005.
3. A. Armando, R. Carbone, and L. Compagna. LTL model checking for security protocols. In *20th IEEE Computer Security Foundations Symposium (CSF)*, pages 385–396. IEEE Computer Society, 2007.
4. A. Armando, R. Carbone, and L. Compagna. LTL Model Checking for Security Protocols. In *Journal of Applied Non-Classical Logics, special issue on Logic and Information Security*, pages 403–429. Hermes Lavoisier, 2009.
5. A. Armando, R. Carbone, L. Compagna, J. Cuéllar, G. Pellegrino, and A. Sorniotti. An authentication flaw in browser-based single sign-on protocols: Impact and remediations. *Computers & Security*, 33:41–58, 2013.
6. A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In V. Shmatikov, editor, *Proc. ACM Workshop on Formal Methods in Security Engineering*, pages 1–10. ACM Press, 2008.
7. A. Armando, R. Carbone, and L. Zanetti. Formal modeling and automatic security analysis of two-factor and two-channel authentication protocols. In *Network and System Security (NSS)*, volume 7873 of *LNCS*, pages 728–734. Springer, 2013.

8. A. Armando and L. Compagna. SATMC: a SAT-based model checker for security protocols. In *Proc. European Conference on Logics in Artificial Intelligence*, volume 3229 of *LNAI*, pages 730–733, Lisbon, Portugal, 2004. Springer-Verlag.
9. A. Armando and L. Compagna. SAT-based Model-Checking for Security Protocols Analysis. *International Journal of Information Security*, 7(1):3–32, January 2008.
10. A. Armando and S. E. Ponta. Model checking of security-sensitive business processes. In P. Degano and J. D. Guttman, editors, *Formal Aspects in Security and Trust*, volume 5983 of *LNCS*, pages 66–80. Springer, 2009.
11. W. Arsac, L. Compagna, G. Pellegrino, and S. E. Ponta. Security Validation of Business Processes via Model-checking. In *International Symposium on Engineering Secure Software and Systems (ESSoS 2011)*. LNCS, Springer-Verlag, 2011.
12. AVANTSSAR. Deliverable 2.1: Requirements for modelling and ASLan v.1. `www.avantssar.eu`, 2008.
13. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A Symbolic Model-Checker for Security Protocols. *International Journal of Information Security*, 2004.
14. A. Biere. Bounded Model Checking. In Biere A., Heule M., van Maaren H., and Walsh T., editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 457–481. IOS Press, 2009.
15. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proceedings of TACAS'99*, LNCS 1579. Springer-Verlag, 1999.
16. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW)*, pages 82–96, 2001.
17. A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1995.
18. M. Bortolozzo, M. Centenaro, R. Focardi, and G. Steel. Attacking and fixing PKCS#11 security tokens. In *Proc. ACM Conference on Computer and Communications Security (CCS'10)*, pages 260–269, Chicago, USA, 2010. ACM Press.
19. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, 2002. Springer.
20. L. Compagna, P. Guilleminot, and A. D. Brucker. Business process compliance via security validation as a service. In *6th Intl. Conf. on Software Testing, Verification and Validation (ICST)*, pages 455–462, 2013.
21. OASIS Consortium. SAML V2.0 Technical Overview. `http://wiki.oasis-open.org/security/Saml2TechOverview`, March 2008.
22. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
23. Riccardo Focardi, Flaminia L. Luccio, and Graham Steel. An introduction to security API analysis. In *Foundations of Security Analysis and Design – FOSAD Tutorial Lectures (FOSAD'VI)*, volume 6858 of *LNCS*, pages 35–65. 2011.
24. Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.
25. RSA Sec. Inc. PKCS#11: Cryptographic Token Interface Standard v2.20, 2004.
26. H. Kautz, H. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 374–384. Morgan Kaufmann, 1996.
27. M. Turuani. The CL-Atse Protocol Analyser. In F. Pfenning, editor, *Proc. International Conference on Rewriting Techniques and Applications, RTA*, LNCS, Seattle (WA), 2006. Springer.

28. L. Viganò. The SPaCIoS Project: Secure Provision and Consumption in the Internet of Services. *International Conference on Software Testing, Verification, and Validation*, 0:497–498, 2013.