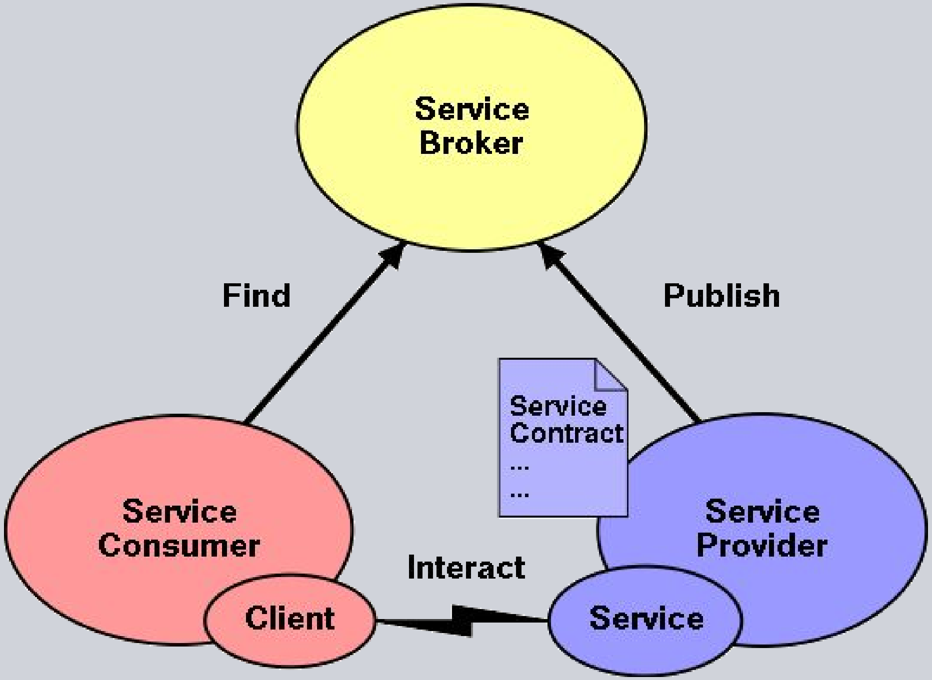
avantssar.eu

The AVANTSSAR[1] Language and Tool



Tools session of FOSAD 2013 summer school in Bertinoro, Italy, 2013-09-04

[1] Automated VAlidatioN of Trust and Security of Service-oriented Architectures

EU FP7-2007-ICT-1, ICT-1.1.4, STREP project no. 216471
Jan 2008 - Dec 2010, 590 PMs, 6M€ budget, 3.8M€ EC contribution

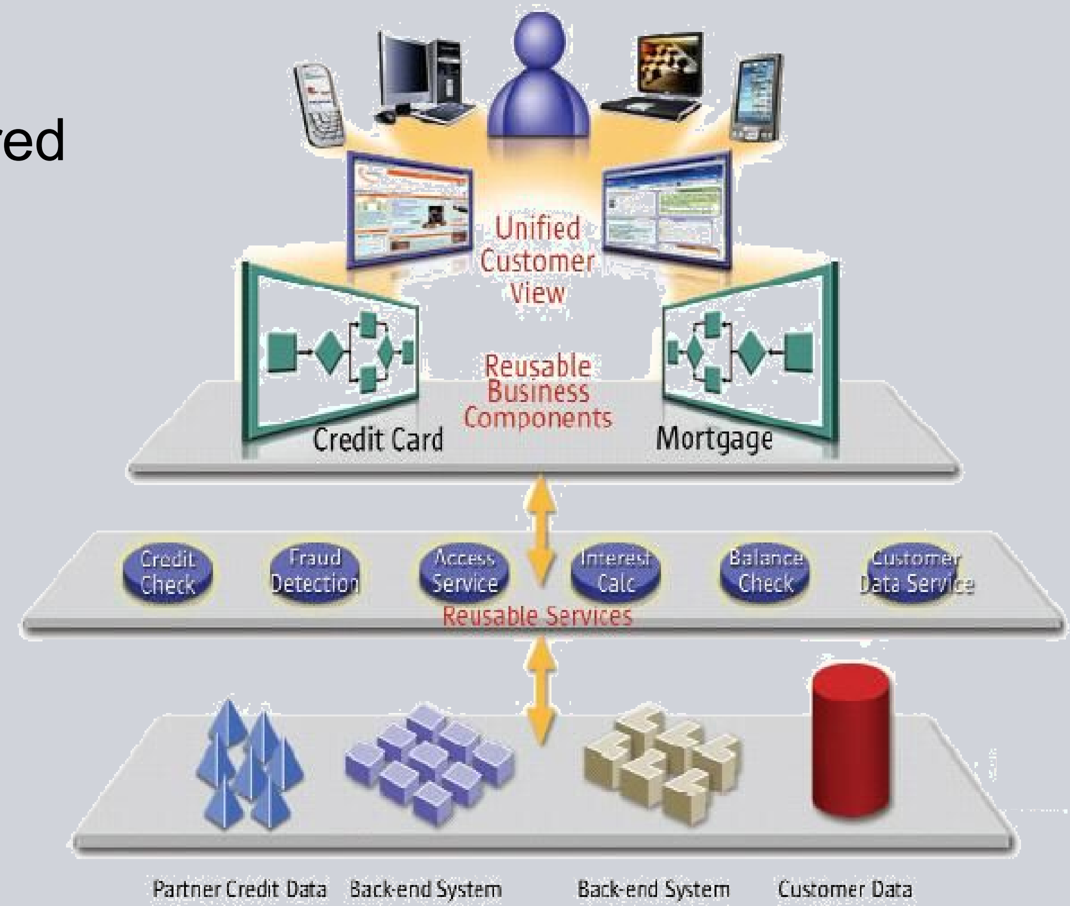
AVANTSSAR project motivation

SIEMENS

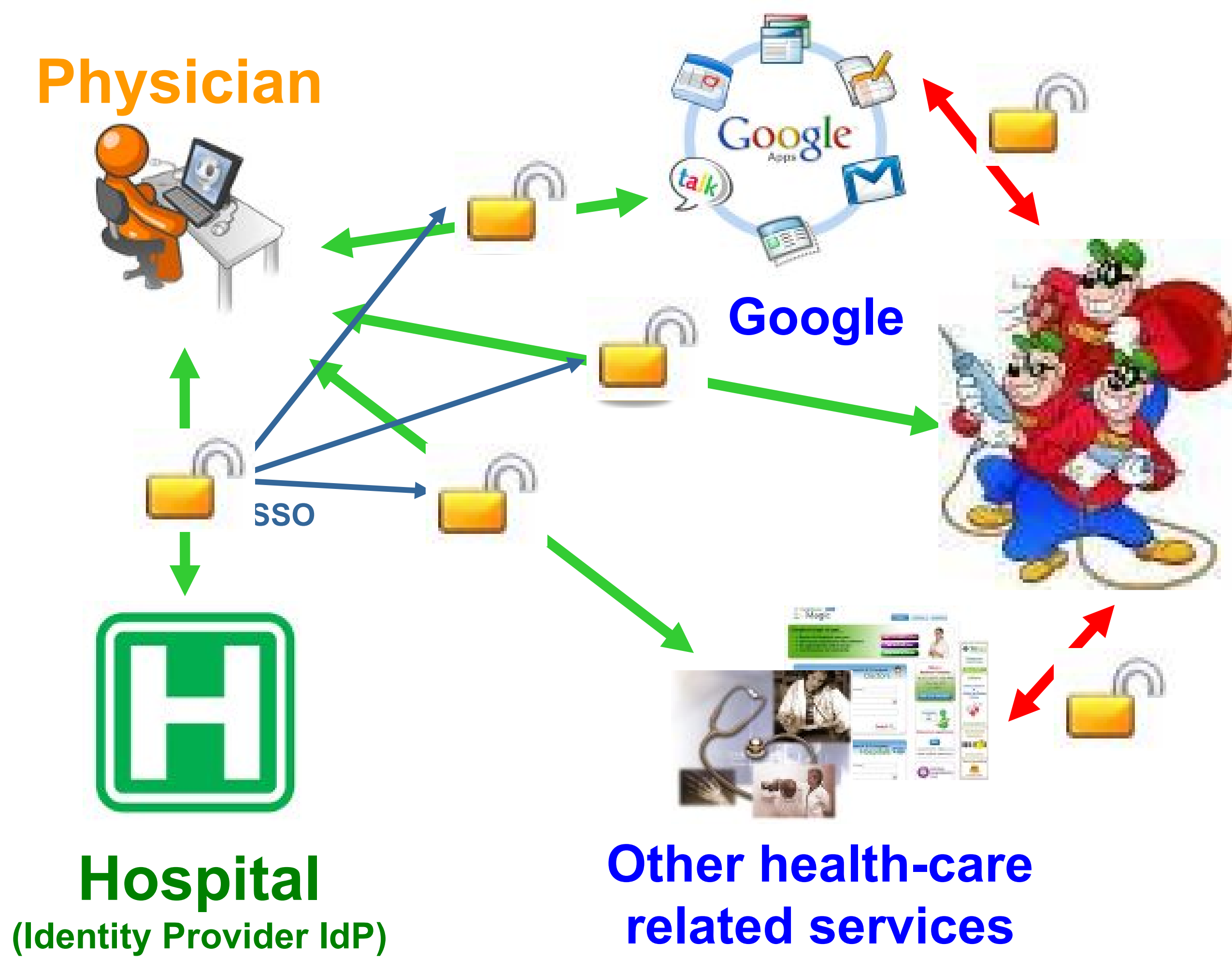
ICT paradigm shift: from components to services, composed and reconfigured dynamically in a demand-driven way.

Trustworthy service may interact with others causing novel trust and security problems.

For the composition of individual services into service-oriented architectures, validation is dramatically needed.



Example 1: Google SAML-based Single Sign-On (SSO)



A malicious service provider can access the data of the physician located at all other services connected via Google SSO

Example 1: Google SAML SSO protocol flaw

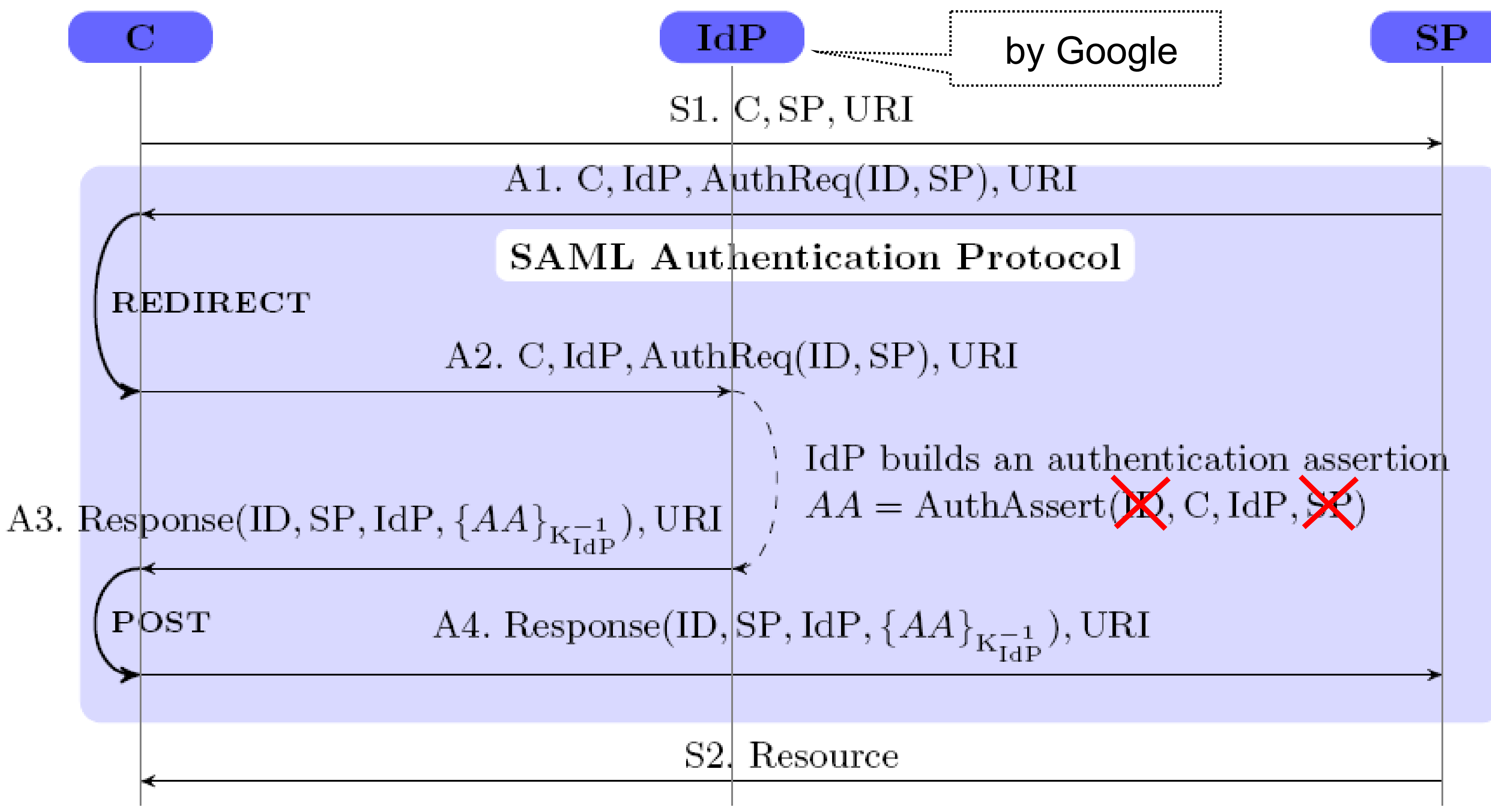


Fig. 1. SP-Initiated SSO with Redirect/POST Bindings

SIEMENS

Example 1: Impact of the Google SAML SSO findings (1)

Google

Corporate Information

[Home](#)

[About Google](#)

Corporate Overview
- Company
- Features
- Technology
- Business
- Culture
- Diversity & Inclusion
- Green Energy

At a Glance
- Quick Profile
- Address
- Management
- Milestones

Our Philosophy
- Ten things
- Software principles
- Design principles
- No pop-ups
- ▸ **Security**

Related Links
- Investor Relations
- Press Center

Find on this site:

Search

Google Security and Product Safety

Google's Security Philosophy

As a provider of software, services and monetization for users, advertisers and publishers on the Internet, we feel we have a responsibility to protect your privacy and security. We recognize that secure products are instrumental in maintaining the trust you place in us and strive to create innovative products that both serve your needs and operate in your best interest.

We've learned that when security is done right, it's done best as a community, and this includes everybody: the people who use Google services (thank you all!), the software developers who make our applications, and the external security enthusiasts who keep us on our toes. These combined efforts go a long way in making the Internet safer and more secure.

Reporting Security Issues

If you are a Google user and have a security issue to report regarding your personal Google account, please visit our contact page. This includes password problems, login issues, spam reports, suspected fraud and account abuse issues.

If you have discovered a vulnerability in a Google product or have a security incident to report, please email security@google.com. Please include a detailed summary of the issue including the name of the product (e.g., Gmail) and the nature of the issue you believe you've discovered. Be sure to include an email address where we can reach you in case we need more information.

This process of notifying a vendor before publicly releasing information is an industry-standard best practice known as *responsible disclosure*. Responsible disclosure is important to the ecology of the Internet. It allows companies like Google to keep users safe by fixing vulnerabilities and resolving security concerns before they are brought to the attention of the bad guys. We strongly encourage anyone who is interested in researching and reporting security issues to observe the simple courtesies and protocols of responsible disclosure.

Working together helps make the online experience safer for everyone.

We take security issues seriously and will respond swiftly to fix verifiable security issues. Some of our products are complex and take time to update. When properly notified of legitimate issues, we'll do our best to acknowledge your emailed report, assign resources to investigate the issue, and fix potential problems as quickly as possible.

We value the security of Google services as well as your privacy when you report vulnerabilities or incidents to us. If you feel the need, please use our public key to encrypt your communications with us when sending email to security@google.com.

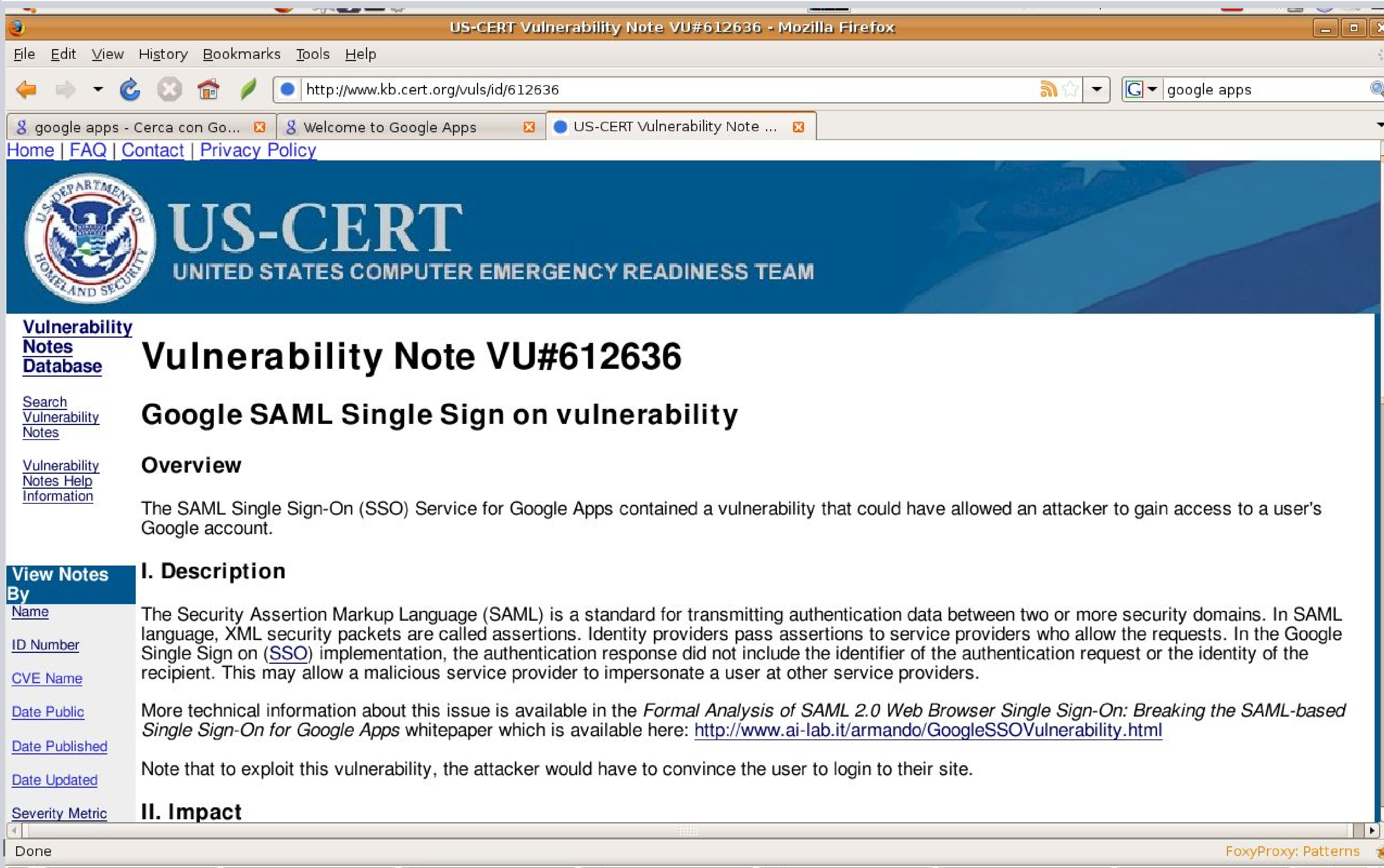
We Thank You

People and organizations with an interest in security issues have made a tremendous contribution to the quality of the online experience. We are grateful for the responsible disclosure of security vulnerabilities. On behalf of our millions of users, we would like to thank the following individuals and organizations for going out of their way to improve the Google experience for everyone:

- Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, Llanos Tobarra Abad with the AVANTSSAR project
- Chris Boyd, FaceTime Communications
- Alex Eckelberry, Sunbelt Software
- Castlecops
- Team Cymru
- Yahoo! Paranoids
- Finjan

SIEMENS

Example 1: Impact of the Google SAML SSO findings (2)



Home | FAQ | Contact | Privacy Policy

US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Vulnerability Notes Database

Search Vulnerability Notes

Vulnerability Notes Help Information

View Notes By
Name
ID Number
CVE Name
Date Public
Date Published
Date Updated
Severity Metric

Vulnerability Note VU#612636

Google SAML Single Sign on vulnerability

Overview

The SAML Single Sign-On (SSO) Service for Google Apps contained a vulnerability that could have allowed an attacker to gain access to a user's Google account.

I. Description

The Security Assertion Markup Language (SAML) is a standard for transmitting authentication data between two or more security domains. In SAML language, XML security packets are called assertions. Identity providers pass assertions to service providers who allow the requests. In the Google Single Sign on (SSO) implementation, the authentication response did not include the identifier of the authentication request or the identity of the recipient. This may allow a malicious service provider to impersonate a user at other service providers.

More technical information about this issue is available in the *Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps* whitepaper which is available here: http://www.ai-lab.it/armando/GoogleSSOVulnerability.html

Note that to exploit this vulnerability, the attacker would have to convince the user to login to their site.

II. Impact

AVANTSSAR consortium

Industry

SAP Research France, Sophia Antipolis

Siemens Corporate Technology, München

IBM Zürich Research Labs (initial two years)

OpenTrust, Paris

Academia

Università di Verona

Università di Genova

ETH Zürich

INRIA Lorraine

UPS-IRIT, Toulouse

IEAT, Timişoara

Expertise

Service-oriented enterprise architectures

Security solutions

Standardization and industry migration

Security engineering

Formal methods

Automated security validation

SIEMENS

AVANTSSAR main technical objectives and aims

AVANTSSAR product: Platform for formal specification and automated validation of trust and security of SOAs

- Formal language for specifying security properties of services, their policies, and their composition into service-oriented architectures
- Automated tool set supporting the above
- Library of validated industry-relevant case studies

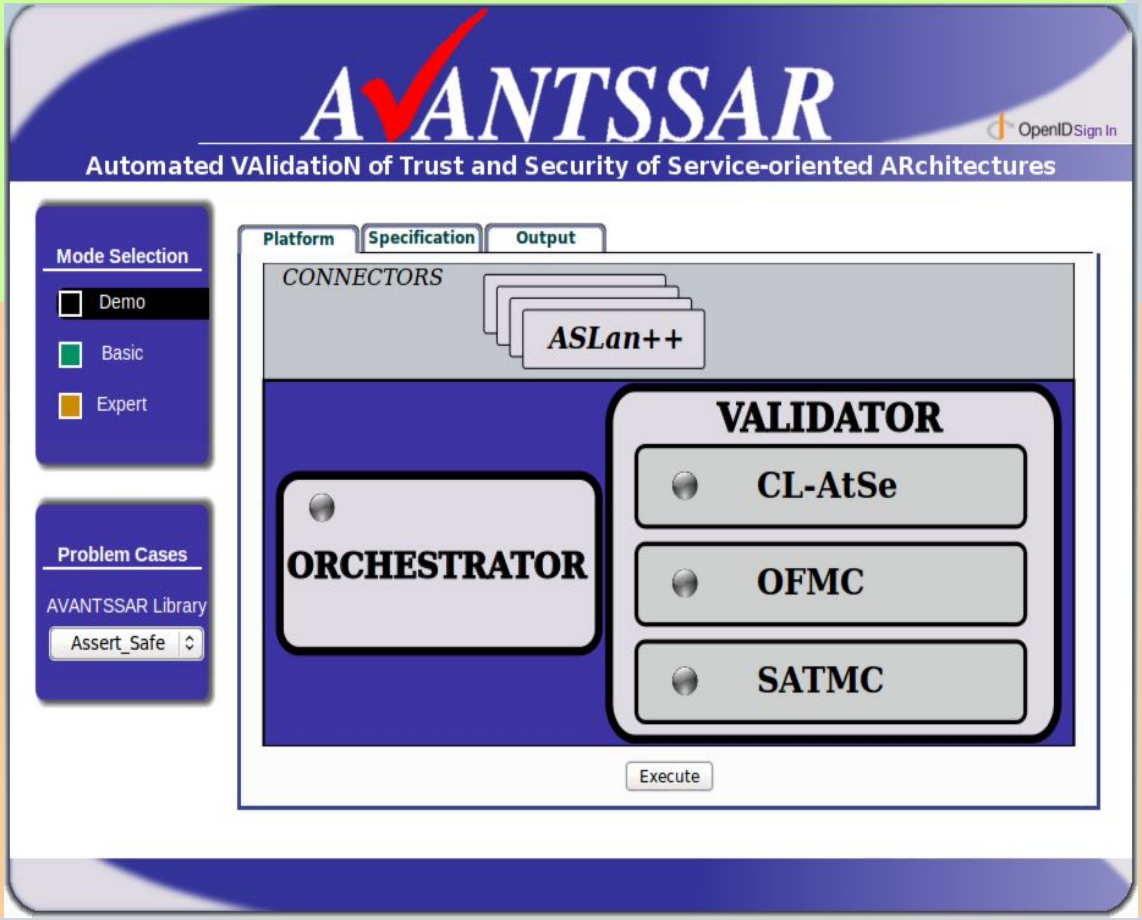
Migration of platform to industry and standardization organizations

- Speed up development of new service infrastructures
- Enhance their security and robustness
- Increase public acceptance of SOA-based systems

SIEMENS

The AVANTSSAR Tool main components

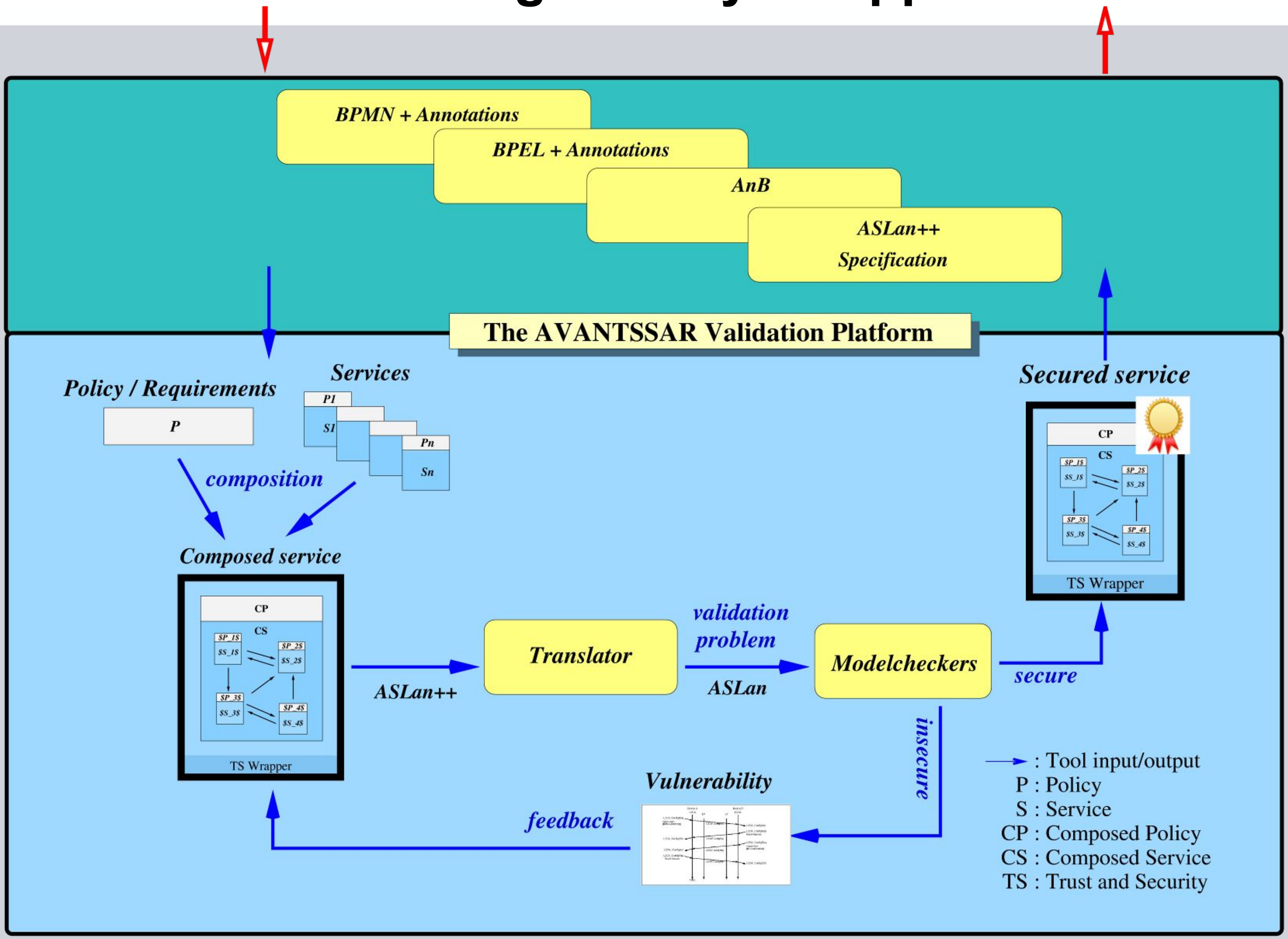
- **ASLan++ Connector** translates to ASLan (and validation results back). All other tools operate at ASLan level.
- **Orchestrator** combines service components.



Validator gives choice of three model checkers**:**

- **CL-AtSe**: Constraint-Logic Attack Searcher
- **OFMC**: Open-source Fixed-point Model Checker
- **SATMC**: SAT-based Model Checker

AVANTSSAR modeling & analysis approach



SIEMENS

Example 1: ASLan++ model of NSPK (1): Alice and Bob

```
specification NSPK …
  entity Alice (Actor, B: agent) {
    symbols
      Na, Nb: text;
    body {
      secret_Na:(Na) := fresh();
      Actor -> B: {Na.Actor}_pk(B);
      B -> Actor: {Alice_freshly_authenticates_Bob:(Na).
                                          secret_Nb:(?Nb)}_pk(Actor);
      Actor -> B: {Bob_freshly_authenticates_Alice:(Nb)}_pk(B);
    }
  }
  entity Bob (A, Actor: agent) {
    symbols
      Na, Nb: text;
    body {
      ? -> Actor: {?Na.?A}_pk(Actor); % Bob learns A here!
      secret_Nb:(Nb) := fresh();
      Actor -> A: {Alice_freshly_authenticates_Bob:(Na).Nb}_pk(A);
      A -> Actor: {Bob_freshly_authenticates_Alice:(Nb)}_pk(Actor);
      secret_Na:(Na) := Na; % secrecy of Na cannot hold earlier than here
    }
  }
  …
}
```

Example 1: ASLan++ model of NSPK (2): outer structure

```
specification NSPK
channel_model CCM
entity Environment {
  entity Session (A, B: agent) {
    entity Alice (Actor, B: agent) {…}
    entity Bob   (A, Actor: agent) {…}
    body {
      new Alice(A,B);
      new Bob  (A,B);
    }
    goals
      secret_Na: {A,B};
      secret_Nb: {A,B};
      Alice_freshly_authenticates_Bob: B *->> A;
      Bob_freshly_authenticates_Alice: A *->> B;
  }
  body { % need two sessions for Lowe's attack
    any A B. Session(A,B) where A!=B;
    any C D. Session(C,D) where C!=D;
  }
}
```


SIEMENS

AVANTSSAR references

Web resources

- Official home page: avantssar.eu
- Unofficial tool download site: ddvo.net/AVANTSSAR/

Selected publications

- Luca Viganò et al.: The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. In Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2012, LNCS 7214, p. 267-282.
- David von Oheimb and Sebastian Mödersheim: ASLan++ — a formal security specification language for distributed systems. In Formal Methods for Components and Objects (FMCO) 2010, Graz, Austria. Springer LNCS 6957, pages 1-22.
- A. Armando, R. Carbone, L. Compagna, J. Cuéllar, 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 proceeding of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)

Backup slides

SIEMENS

ASLan++ language design

- **Design goals**
  - **Expressive** enough for *modeling a wide range of SOAs*
  - Enable **succinct** specifications, for *minimal handling effort*
  - High **abstraction** level, to *reduce model complexity*
  - Close to **specification languages** for security protocols and web services
  - Close to procedural and object-oriented **programming languages**
  - *Minimal learning effort* for **non-expert** modelers
- **Relation with ASLan**
  - ASLan++ more **high-level** than ASLan (formerly called IF)
  - ASLan++ semantics defined by **translation** to ASLan
  - Main differences:

hierarchy of classes	vs. flat transition system
procedural statements	vs. term rewriting rules
high-level security goals	vs. attack states & auxiliary events

SIEMENS

ASLan++ features for system modeling

- **Overall structure**
  - Hierarchy and modularity via entities (similar to classes)
  - Dynamic entity instantiation with (pot. underspecified) agents
  - Parallel composition of sequential instance execution
- **Local declarations**
  - Types with subtyping, generic tuples and sets
  - Constants, functions, statically scoped instance variables
  - Horn clauses allowing for (limited) deductions
- **Local execution**
  - Classical control flow constructs (e.g. if and while)
  - Cryptographic primitives and fresh value generation
  - Pattern matching (unification modulo some equalities)
  - Send and receive instructions with guards
  - Channels with security assumptions

SIEMENS

ASLan++ features for security property modeling

- **Security goals**
  - Invariants as LTL formulas
  - Assertions as LTL formulas
  - Secrecy of values among a group of agents
  - Channel goals: authenticity, confidentiality, freshness, …
- **Attacker model**
  - Built-in Dolev-Yao intruder model
  - Dishonest agents (agents may be compromised dynamically)
  - Extensible intruder knowledge
- **Limitations (mostly due to model-checking)**
  - No term evaluation except for limited equations
  - No arithmetic
  - No notion of time
  - No ‘semi-honest’ parties

Semantics of channel goals as LTL formulas

A channel goal requiring authentication, directedness, freshness, and confidentiality:

```
secure_Alice_Payload_Bob: A *->>* B: Payload;
```

On the sender side: `Actor -> B: ...Payload...;`

```
witness(Actor,B,auth_Alice_Payload_Bob,Payload);
secret(Payload,secr_Alice_Payload_Bob,{Actor,B});
```

On the receiver side: `A -> Actor: ...?Payload...;`

```
request(Actor,A,auth_Alice_Payload_Bob,Payload,IID);
secret(Payload,secr_Alice_Payload_Bob,{A,Actor});
```

Semantics of the **authentication** and **directedness** part:

```
forall A,B,P,M,IID. [] (request(B,A,P,M,IID) =>
 (<-> (witness(A,B,P,M)) || (dishonest(A) & iknows(M))))
```

Semantics of the **freshness** (replay protection) part:

```
forall A,B,P,M,IID IID'. [] (request(B,A,P,M,IID) =>
 (!(<-> (request(B,A,P,M,IID') & !(IID=IID'))  ||  dishonest(A)))
```

Semantics of the **confidentiality** part:

```
forall M,P,As. [] ((secret(M,P,As) & iknows(M)) => contains(i, As))
```


SIEMENS

Optimization: Merging transitions on translation

A series of transmission and internal computation ASLan++ commands like

```
receive(A, ?M);
N := fresh();
send(A, N);
```

could bet translated into individual ASLan transitions like:

```
state_entity(Actor, IID, 1, dummy, dummy) . iknows(M) =>
state_entity(Actor, IID, 2, M    , dummy)

state_entity(Actor, IID, 2, M    , dummy) =[exists N]=>
state_entity(Actor, IID, 3, M    , N    )

state_entity(Actor, IID, 3, M    , N    ) =>
state_entity(Actor, IID, 4, M    , N    ) . iknows(N)
```

but can be \`compressed´ into a single atomic ASLan transition:

```
state_entity(Actor, IID, 1, dummy, dummy) . iknows(M) =[exists N]=>
state_entity(Actor, IID, 4, M    , N    ) . iknows(N)
```

Even internal computations containing loops etc. can be \`glued together´ to avoid interleaving. This dramatically reduces the search space because a lot of useless branching is avoided.

Example 1: Google's SSO in AnB notation (OFMC input)

```
Knowledge: C:   C,idp,SP,pk(idp);
           idp: C,idp,pk(idp),inv(pk(idp));
           SP:  idp,SP,pk(idp)
Actions:
  [C] *->* SP : C,SP,URI
  SP *->* [C] : C,idp,SP,URI
  C   *->* idp : C,idp,SP,URI
  idp *->* C   : {C,idp}inv(pk(idp)),URI
  [C] *->* SP : {C,idp}inv(pk(idp)),URI
  SP *->* [C] : Data
Goals:
  SP authenticates C on URI
  C authenticates SP on Data
  Data secret between SP,C
```


Example 1: Attack on Google's SSO (OFMC output)

The attack found by OFMC in nice notation:

```
1. [a] *->* i:   a.i.URI(1)


1.' [i] *->* b:  a.b.x306
2.' b *->* [i]:  a.idp.b.ID(2).x306


2. i *->* [a]:   a.idp.i.x505.URI(1)
3. a *->* idp: a.idp.i.x505.URI(1)
4. idp *->* a: {a.idp}_inv(pk(idp)).URI(1)
5. [a] *->* i:   {a.idp}_inv(pk(idp)).URI(1)


5.' [i] *->* b:  {a.idp}_inv(pk(idp)).x306
6.' b *->* [i]:  Data(6)
```


Example 1: Google's SSO: the problem

The authentication assertion from the idp:

$$\{ID, C, ipd, SP\}inv(pk(idp)$$

- Google had omitted some parts that were suggested but not required by the standard.
- This allows a dishonest SP to re-use the authentication assertion and log in to other sites as C.

Again, this is a problem related to a dishonest participant!

SIEMENS

Example 2: ASLan model of NSPK (1): types, functions

```
% Specification: NSPK
% Channel model: CCM
% Goals as attack states: yes
% Orchestration client: N/A
% Horn clauses level: ALL
% Optimization level: LUMP
% Stripped output (no comments and line information): no
section signature:
    message > text
    ak : agent -> public_key
    ck : agent -> public_key
    defaultPseudonym : agent -> agent
    descendant : nat * nat -> fact
    dishonest : agent -> fact
    isAgent : agent -> fact
    pk : agent -> public_key
    secr_Alice_Bob_PayloadA_set : nat -> set(agent)
    secr_Bob_Alice_PayloadB_set : nat -> set(agent)
    secret_Na_set : nat -> set(agent)
    secret_Nb_set : nat -> set(agent)
    sign : private_key * message -> message
    state_Alice : agent * nat * nat * agent * text * text * text * text -> fact
    state_Bob : agent * nat * nat * agent * text * text * text * text -> fact
    state_Environment : agent * nat * nat -> fact
    state_Session : agent * nat * nat * agent * agent -> fact
```


Example 2: ASLan model of NSPK (2): constants, variables

```
section types:
     A : agent
     Actor : agent
     Ak_arg_1 : agent
     B : agent
     Ck_arg_1 : agent
     Descendant_Closure_arg_1 : nat
     Descendant_Closure_arg_2 : nat
     Descendant_Closure_arg_3 : nat
     E_S_A_Actor : agent
     E_S_A_B : agent
     E_S_A_IID : nat
     E_S_A_SL : nat
     E_S_Actor : agent
     E_S_B_A : agent
     E_S_B_A_1 : agent
     E_S_B_A_2 : agent
     E_S_B_Actor : agent
     E_S_B_IID : nat
     E_S_B_Na : text
     E_S_B_Na_1 : text
     E_S_B_Nb : text
     E_S_B_Nb_1 : text
     E_S_B_PayloadA : text
     E_S_B_PayloadA_1 : text
     E_S_B_PayloadB : text
     E_S_B_PayloadB_1 : text
     E_S_B_SL : nat
     E_S_IID : nat
     E_S_SL : nat
     E_aABPA_IID : nat
     E_aABPA_Msg : message
     E_aABPA_Req : agent
     E_aABPA_Wit : agent
     E_aBAPB_IID : nat
     E_aBAPB_Msg : message
     E_sABPA_Knowers : set(agent)
     E_sABPA_Msg : message
     E_sBAPB_Knowers : set(agent)
     E_sBAPB_Msg : message
     E_sN_Knowers : set(agent)
     E_sN_Msg : message
     IID : nat
     IID_1 : nat
     IID_2 : nat
     IID_3 : nat
     IID_4 : nat
     Knowers : set(agent)
     Msg : message
     Na : text
     Na_1 : text
     Nb : text
     Nb_1 : text
     PayloadA : text
     PayloadA_1 : text
     PayloadB : text
     PayloadB_1 : text
     Pk_arg_1 : agent
     Req : agent
     SL : nat
     Sign_arg_1 : private_key
     Sign_arg_2 : message
     Wit : agent
     a : agent
     atag : text
     auth_Alice_Bob_PayloadA
           : protocol_id
     auth_Bob_Alice_PayloadB
           : protocol_id
     b : agent
     ctag : text
     dummy_agent : agent
     dummy_nat : nat
     dummy_text : text
     false : fact
     secr_Alice_Bob_PayloadA
           : protocol_id
     secr_Bob_Alice_PayloadB
           : protocol_id
     secret_Na : protocol_id
     secret_Nb : protocol_id
     stag : text
     true : fact
```

Example 2: ASLan model of NSPK (3): initial state, clauses

```
section inits:

initial_state init :=
    dishonest(i).
    iknows(a).
    iknows(atag).
    iknows(b).
    iknows(ctag).
    iknows(i).
    iknows(inv(ak(i))).
    iknows(inv(ck(i))).
    iknows(inv(pk(i))).
    iknows(stag).
    isAgent(a).
    isAgent(b).
    isAgent(i).
    state_Environment(
        dummy_agent,
        dummy_nat, 1).
    true
```

```
section hornClauses:

    hc public_ck(Ck_arg_1) :=
        iknows(ck(Ck_arg_1)) :-
            iknows(Ck_arg_1)

    hc public_ak(Ak_arg_1) :=
        iknows(ak(Ak_arg_1)) :-
            iknows(Ak_arg_1)

    hc public_pk(Pk_arg_1) :=
        iknows(pk(Pk_arg_1)) :-
            iknows(Pk_arg_1)

    hc public_sign(Sign_arg_1, Sign_arg_2) :=
        iknows(sign(Sign_arg_1, Sign_arg_2)) :-
            iknows(Sign_arg_1),
            iknows(Sign_arg_2)

    hc inv_sign(Sign_arg_1, Sign_arg_2) :=
        iknows(Sign_arg_2) :-
            iknows(sign(Sign_arg_1, Sign_arg_2))

    hc descendant_closure(Descendant_Closure_arg_1,
      Descendant_Closure_arg_2, Descendant_Closure_arg_3) :=
        descendant(Descendant_Closure_arg_1,
                    Descendant_Closure_arg_3) :-
            descendant(Descendant_Closure_arg_1,
                        Descendant_Closure_arg_2),
            descendant(Descendant_Closure_arg_2,
                        Descendant_Closure_arg_3)
```

SIEMENS

Example 2: ASLan model of NSPK (4): transition rules

```
section rules:

% line 75
% new instance
%     new Session(a,b)
% lumped line 76 (skipped step label 2)
% new instance
%     new Session(a,i)
step step_1_Environment__line_75(Actor, IID, IID_1, IID_2) :=
      state_Environment(Actor, IID, 1)
      =[exists IID_1, IID_2]=>
      descendant(IID, IID_1).
      descendant(IID, IID_2).
      state_Environment(Actor, IID, 3).
      state_Session(dummy_agent, IID_1, 1, a, b).
      state_Session(dummy_agent, IID_2, 1, a, i)

% line 62
% guard
%     !dishonest(A)
% lumped line 63 (skipped step label 2)
% new instance
%     new Alice(A,B)
step step_2_Session__line_62(A, B, E_S_Actor, E_S_IID, IID_3) :=
      not(dishonest(A)).
      state_Session(E_S_Actor, E_S_IID, 1, A, B)
      =[exists IID_3]=>
      descendant(E_S_IID, IID_3).
      state_Alice(A, IID_3, 1, B, dummy_text, dummy_text, dummy_text, dummy_text).
      state_Session(E_S_Actor, E_S_IID, 3, A, B)
```

... (some 5 more pages of rules)

Example 2: ASLan model of NSPK (5): goals

```
section goals:

attack_state auth_Alice_Bob_PayloadA(E_aABPA_IID, E_aABPA_Msg, E_aABPA_Req, E_aABPA_Wit) :=
      not(witness(E_aABPA_Wit, E_aABPA_Req, auth_Alice_Bob_PayloadA, E_aABPA_Msg)).
      request(E_aABPA_Req, E_aABPA_Wit, auth_Alice_Bob_PayloadA, E_aABPA_Msg, E_aABPA_IID) &
      not(equal(i, E_aABPA_Wit))

attack_state auth_Bob_Alice_PayloadB(E_aBAPB_IID, E_aBAPB_Msg, Req, Wit) :=
      not(witness(Wit, Req, auth_Bob_Alice_PayloadB, E_aBAPB_Msg)).
      request(Req, Wit, auth_Bob_Alice_PayloadB, E_aBAPB_Msg, E_aBAPB_IID) &
      not(equal(i, Wit))

attack_state secr_Alice_Bob_PayloadA(E_sABPA_Knowers, E_sABPA_Msg) :=
      iknows(E_sABPA_Msg).
      not(contains(i, E_sABPA_Knowers)).
      secret(E_sABPA_Msg, secr_Alice_Bob_PayloadA, E_sABPA_Knowers)

attack_state secr_Bob_Alice_PayloadB(E_sBAPB_Knowers, E_sBAPB_Msg) :=
      iknows(E_sBAPB_Msg).
      not(contains(i, E_sBAPB_Knowers)).
      secret(E_sBAPB_Msg, secr_Bob_Alice_PayloadB, E_sBAPB_Knowers)

attack_state secret_Na(Knowers, Msg) :=
      iknows(Msg).
      not(contains(i, Knowers)).
      secret(Msg, secret_Na, Knowers)

attack_state secret_Nb(E_sN_Knowers, E_sN_Msg) :=
      iknows(E_sN_Msg).
      not(contains(i, E_sN_Knowers)).
      secret(E_sN_Msg, secret_Nb, E_sN_Knowers)
```

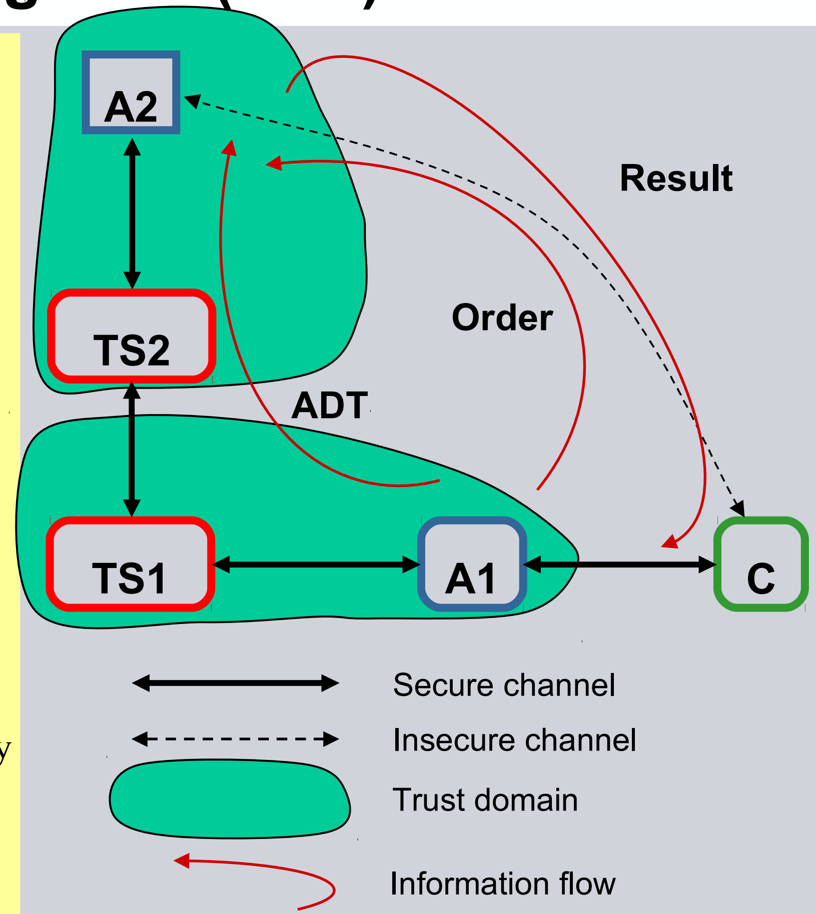
SIEMENS

Example 3: Process Task Delegation (PTD)

Authorization and trust management via token passing

- There are three roles in the protocol (**C, A, TS**) and potentially several instances for each role
- The ***client*** **C** (or *user*) uses the system for authorization and trust management, e.g. SSO
- Each ***application*** **A** is in one domain, each domain has exactly one active ***trust server*** **TS**
- **A1** uses the system to pass to **A2** some **Order** and an **ADT (Authorization Decision Token)**
  - **Order** contains:
    - workflow task information
    - application data
    - information about the client **C** and his current activity

    to be delivered securely (integrity and confidentiality)
  - **ADT** is mainly authorization *attributes* and *decisions*
    - sent via **TS1** and **TS2**, <u>who may weaken it</u>
    - must remain unaltered, apart from weakening by **TS**
    - must remain confidential among intended parties
- **C**, **A1**, and **A2** must be authenticated among each other



Security prerequisites:

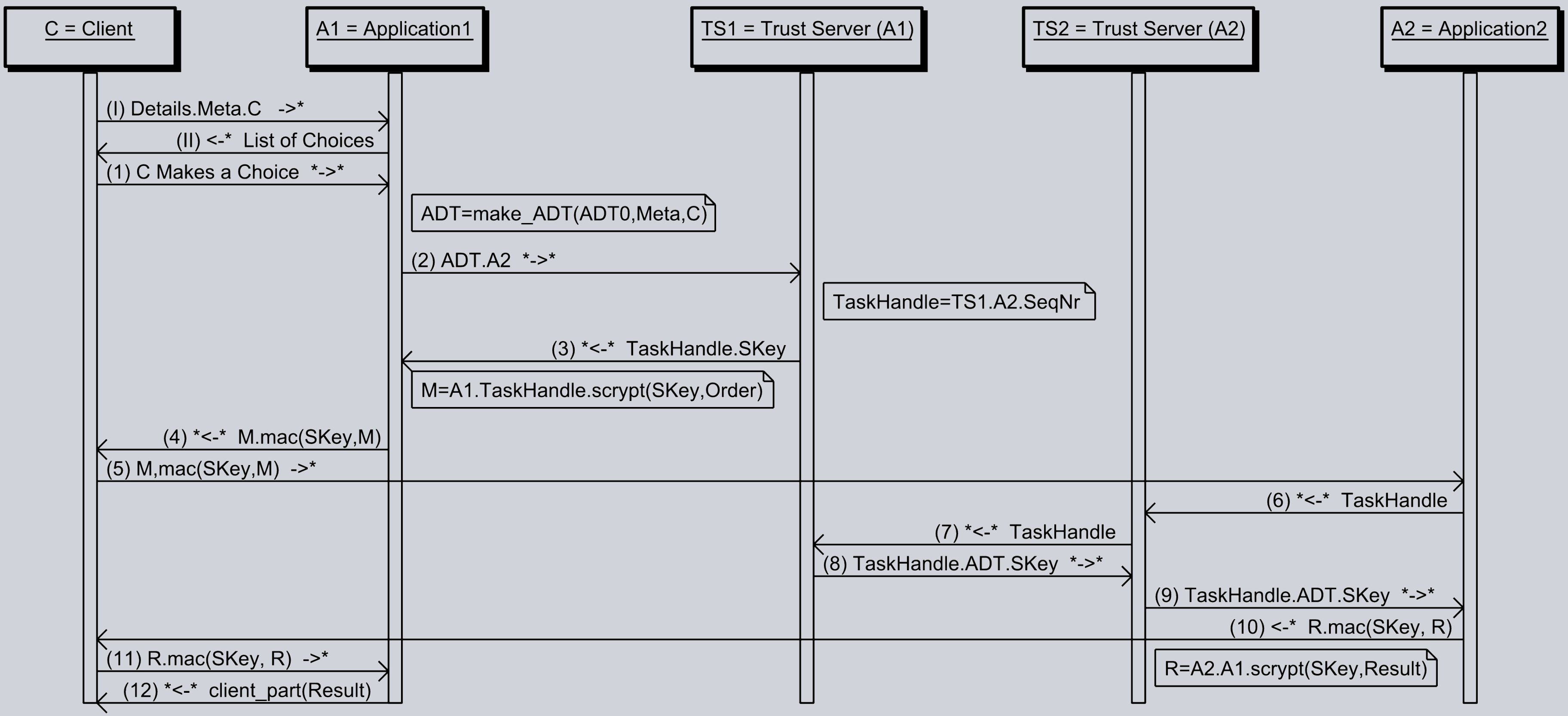
- PKI is used for **A** and **TS**, username & pwd for **C**
- The **TS** enforce a strict time-out

Example 3: ASLan++ model of PTD Application A2

```
entity A2 (Actor: agent, TS2: agent) {    % Application 2, connected with Trust Server 2
  symbols
    C0,C,A1: agent;
    CryptedOrder, Order, Details, Results, TaskHandle, ADT, MAC: message;
    SKey: symmetric_key;
  body { while (true) {
    select {
      % A2 receives (via some C0) a package from some A1. This package includes encrypted and
      % hashed information. A2 needs the corresponding key and the Authorization Decision Token.
      on (?C0 -> Actor: (?A1.Actor.?TaskHandle.?CryptedOrder).?MAC): {
        % A2 contacts its own ticket server (TS2) and requests the secret key SKey and the ADT.
        Actor *->* TS2: TaskHandle;
      }
        % A2 receives from A1 the SKey and checks if the decrypted data corresponds to the hashed data
      on (TS2 *->* Actor: (?ADT.?SKey).TaskHandle  & CryptedOrder = scrypt(SKey,?,?Details.?C)
          & MAC = hash(SKey, A1.Actor.TaskHandle.CryptedOrder)): {
        % A2 does the task requested by A1, then sends to A1 via C the results encrypted with the secret key.
        Results := fresh();  % in general, the result depends on Details etc.
        Actor -> C: Actor.C.A1. scrypt(SKey,Results);
   } } }
  goals
    authentic_C_A2_Details: C  *-> Actor: Details;
    secret_Order: secret (Order, {Actor, A1});
}
```

SIEMENS

Example 3: Message Sequence Chart of PTD



SIEMENS

Example 4: Electronic Car Registration policies

local policy

RegOffCA can say
who is RegOffHead
who is RegOffEmpl

certificate

Peter is RegOffEmpl
of CarRegOffice
(signed by RegOffCA)

ACL

anybody,
get empty forms
RegOffHead, write
RegOffEmpl, read
RegOffEmpl, write,
if *his* RegOffHead
says so

CentrRep

write(..)

Peter

Melinda

RegOffCA

certificate

Melinda is RegOffHead
of CarRegOffice
(signed by RegOffCA)

certificate

Peter can write
CentrRep
(signed by Melinda)

Question:

May Peter write to CentrRep?

Example 4: On-the-fly inferences via Horn clauses

DKAL-style trust inference, e.g. trust application:

```
trustapp(P,Q,AnyThing):
  P->knows(AnyThing) :-
    P->trusts(Q,AnyThing) &
    P->knows(Q->said(AnyThing));
```

Basic facts, e.g. the central repository fully trusts the CA

```
centrRepTrustCA(AnyThing):
  centrRep->trusts(theCA,AnyThing);
```

State-dependent (evolving) facts, e.g. department head manages a set of trusted employees:

```
trustedEmplsCanStoreDoc(Head): forall Empl.
  Head->knows(Empl->canStoreDoc) :-
    contains(TrustedEmpls, Empl);
```

Use of certificates, e.g. the central repository trusts the department head on employee's rights:

```
centrRepTrustHead(Head,Empl):
  centrRep->trusts(Head,Empl->canStoreDoc) :-
    centrRep->knows(theCA->said(Head->hasRole(head))) &
    centrRep->knows(theCA->said(Empl->hasRole(employee)));
```


AVANTSSAR: final status

SIEMENS

WP2: ASLan++ supports the formal specification of security related aspects of distributed systems, including explicit policies and service composition

WP3: Techniques for: model checking the security of systems including satisfiability check of dynamic policies and compositional reasoning for channel properties

WP4: Prototype of the **AVANTSSAR Platform**

WP5: Formalization of **industry-relevant problem cases** as ASLan++ specifications and their validation

WP6: **Ongoing dissemination and migration** into scientific community and industry

SIEMENS

AVANTSSAR: conclusion and industry migration

Contemporary SOA has complex structure and security requirements including dynamic trust relations and application-specific policies.

On integration of the AVANTSSAR Platform in industrial development, a rigorous demonstration that the security requirements are fulfilled will:

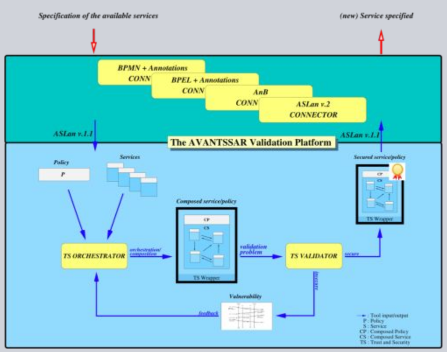
- assist developers with security architecture, analysis and certification
- increase customers' confidence in modern service-oriented architectures

The AVANTSSAR Platform will advance the security of industrial vendors' service offerings: validated, provable, traceable.

AVANTSSAR will thus strengthen the competitive advantage of the products of the industrial partners.

eBusiness

SW Dist



Portals

Health care

