

Methods and tools for design time and runtime formal analysis of security protocols and web applications

Marco Rocchetto



REsearch Group in Information Security
Department of Computer Science
University of Verona, Italy

Verona, May 8, 2015

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification

What

- Formal techniques for the **verification** of:
 - **security protocols**
 - **web applications**
- Design time: **SPiM**, protocol verification using **Craig interpolation** as a speed-up technique
- Runtime: **Formalization** of web applications (major EU bank) searching for **CSRF**

Verification

- protocol/webapp verification:
 - difficult & error-prone
 - state space explosion (DY)

Interpolation

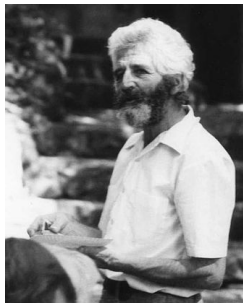
- good results in sw verification:
 - can we use interpolation with protocols?
 - can we use interpolation with DY?

Web applications

- good results with formal verification of protocols:
 - can we use it with web applications?
 - can we use the DY or a specific web intruder?
 - how can we model a webapp?

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification

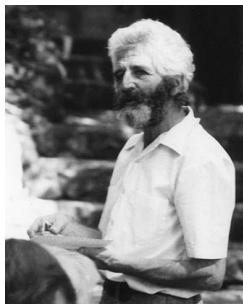


William Craig, UC Berkeley Logic Group picnic 1977.

William Craig, *"Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory"*, The Journal of Symbolic Logic, 1957

Craig interpolation

LEMMA 1. *If $\vdash A \supset A'$ and if A and A' have a predicate parameter in common, then there is an "intermediate" formula B such that $\vdash A \supset B$, $\vdash B \supset A'$, and all parameters of B are parameters of both A and A' . Also, if $\vdash A \supset A'$ and if A and A' have no predicate parameter in common,⁴ then either $\vdash \neg A$ or $\vdash A'$.*



William Craig, UC Berkeley Logic Group picnic 1977.

William Craig, *“Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory”*, The Journal of Symbolic Logic, 1957

Craig interpolation

LEMMA 1. (ALTERNATIVE VERSION.) *If A and C are each a consistent formula of PC_I , and if there is no B whose parameters are parameters of both A and C such that $\vdash A \supset B$ and $\vdash C \supset \neg B$, then $A.C$ is also consistent.*



William Craig, UC Berkeley Logic Group picnic 1977.

William Craig, “*Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory*”, The Journal of Symbolic Logic, 1957

Craig interpolation

In FOL, if $\alpha \wedge \beta$ is inconsistent, then there exists $\hat{\alpha}$ s.t.

- α implies $\hat{\alpha}$
- $\hat{\alpha}$ implies $\neg\beta$
- $\mathcal{L}(\hat{\alpha}) \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$



Kenneth McMillan, somewhere in Japan

Kenneth L. McMillan, “*Lazy Annotation for Program Testing and Verification*”, Computer Aided Verification, 2010

IntraLA - for sequential programs

The remaining case is that some outgoing edge $e = (l_1, a, l_2)$ is not blocked, but every possible symbolic step along that edge leads to a blocked state. In this case, the LEARN rule infers a new label ϕ that blocks the edge. The new edge label can be any formula ϕ that both blocks the current query and is justified. **Such a formula can be obtained as an interpolant** for (A, B) , where $A = \chi(s_1)$ and $B = \text{Sem}(a) \wedge \neg A(l_2)'$. Thus we can derive ϕ , if it exists, using an interpolating theorem prover [7].



Kenneth McMillan, somewhere in Japan

Kenneth L. McMillan, *"Lazy Annotation for Program Testing and Verification"*, Computer Aided Verification, 2010

IntraLA - for sequential programs

- Init - initialize the graph
- Decide - performs symbolic execution
- Learn - calculates interpolants
- Conjoin - backward if DFS is not leading to a goal



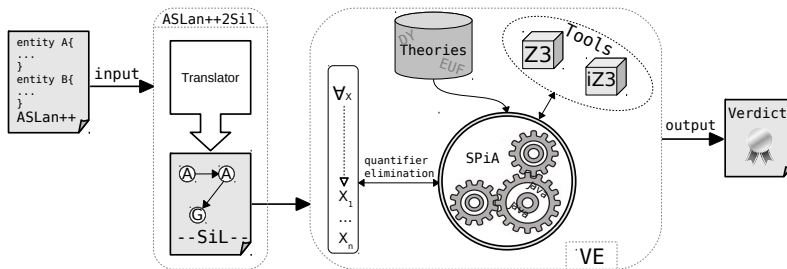
Kenneth McMillan, somewhere in Japan

Kenneth L. McMillan, *"Lazy Annotation for Program Testing and Verification"*, Computer Aided Verification, 2010

IntraLA - for sequential programs

- Successfully applied in formal methods for model checking and test-case generation for sequential programs
- Unsuitable to the direct application for security protocols:
 - sequential programs only
 - no intruder logic

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification



Source code and case studies available at
<http://regis.di.univr.it/spim.php>

Needham-Schroeder Public Key (NSPK) protocol

$$A \rightarrow B : \{N_A, A\}_{pk(B)}$$

$$B \rightarrow A : \{N_A, N_B\}_{pk(A)}$$

$$A \rightarrow B : \{N_B\}_{pk(B)}$$

Man-in-the-middle attack

$$A \rightarrow i : \{N_A, A\}_{pk(i)}$$

$$i(A) \rightarrow B : \{N_A, A\}_{pk(B)}$$

$$B \rightarrow i(A) : \{N_A, N_B\}_{pk(A)}$$

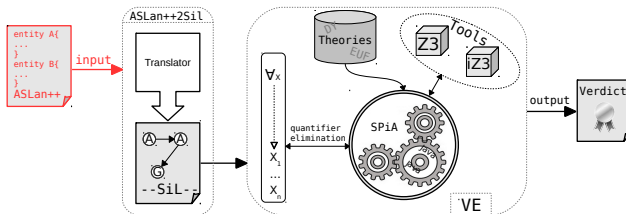
$$i \rightarrow A : \{N_A, N_B\}_{pk(A)}$$

$$A \rightarrow i : \{N_B\}_{pk(i)}$$

$$i(A) \rightarrow B : \{N_B\}_{pk(B)}$$

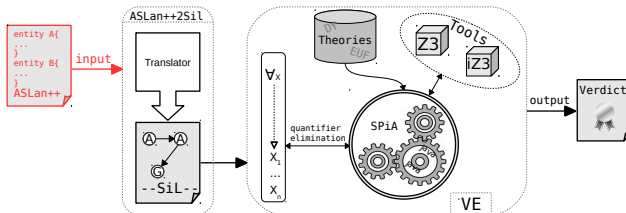
Needham-Schroeder with Lowe's fix (NSL) protocol

$$A \rightarrow B : \{N_A, A\}_{pk(B)}$$
$$B \rightarrow A : \{N_A, N_B, \textcolor{red}{B}\}_{pk(A)}$$
$$A \rightarrow B : \{N_B\}_{pk(B)}$$



Why ASLan++?

- strong background (AVANTSSAR)
- high level language (similar to Java)
- made for protocol analysis, one can specify:
 - entities as parallel processes
 - communication between entities



ASLan++ NSL Code example

Alice(Actor,B:agent){

Na:=fresh();

Actor->B:{Na.Actor}_pk(B);

B->Actor:{Na.Nb.B}_pk(Actor);

Actor->B:{Nb}_pk(B);

}

Bob(Actor,A:agent){

?->Actor:{?Na.?A}_pk(Actor);

Nb:=fresh();

Actor->A:{Na.Nb.B}_pk(A);

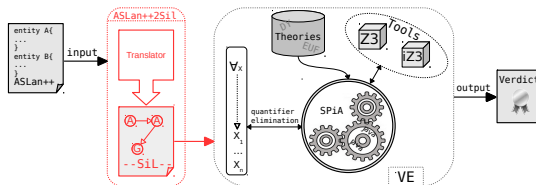
A->Actor:{Nb}_pk(Actor);

}

Goal:
Bob authenticates Alice

Instantiation:

	Alice	Bob
(1)	a	i
(2)	i	b



- IntraLA algorithm designed for sequential programs
(K. McMillan. *Lazy annotation for program testing and verification*. CAV'10)
- To apply (a modified version of) IntraLA to security protocols, we define a translation of a specification of a protocol P for a given scenario into a sequential non-deterministic program

Alice := a, Bob := i

```
1.1) Alice.Actor := a;
1.2) Alice.B := Y_1;
1.3) IK := {a,b,i,pk_a,pk_b,pk_i,pk_i~-1};
1.4)
1.5) Alice.Na := c_1;
1.6) IK := IK + {Alice.Na,Alice.Actor}_pk(Alice.B);
1.7)
1.8) if (IK |- {Alice.Na,?Alice.Nb,Alice.B}_pk(Alice.Actor))
1.9)   then
1.10)    Alice.Nb = Y_2;
1.11)   else
1.12)    end
1.13)
1.14) IK := IK + {Alice.Nb}_pk(Alice.B);
```

```
Alice(Actor,B:agent){
  Na:=fresh();
  Actor->B:{Na.Actor}_pk(B);
  B->Actor:{Na.?Nb.B}_pk(Actor);
  Actor->B:{Nb}_pk(B);
}
```

```
Bob(Actor,A:agent){
  ?->Actor:{?Na.?A}_pk(Actor);
  Nb:=fresh();
  Actor->A:{Na.Nb.B}_pk(A);
  A->Actor:{Nb}_pk(Actor);
}
```

Alice := i, Bob := b

```
2.1) Bob.Actor := b;  
2.2) IK := {a,b,i,pk_a,pk_b,pk_i,pk_i~-1};  
2.3)  
2.4) if (IK |- {?Bob.Na,?Bob.A}_pk(Bob.Actor))  
2.5)   then  
2.6)     Bob.Na = Y_1;  
2.7)     Bob.A = Y_2;  
2.8)   else  
2.9)     end  
2.10)  
2.11) Bob.Nb := c_1;  
2.12) IK := IK + {Bob.Na,Bob.Nb,Bob.Actor}_pk(Bob.A);  
2.13)  
2.14) if (IK |- {Bob.Nb}_pk(Bob.Actor))  
2.15)   then  
2.16)     do nothing  
2.17)   else  
2.18)     end
```

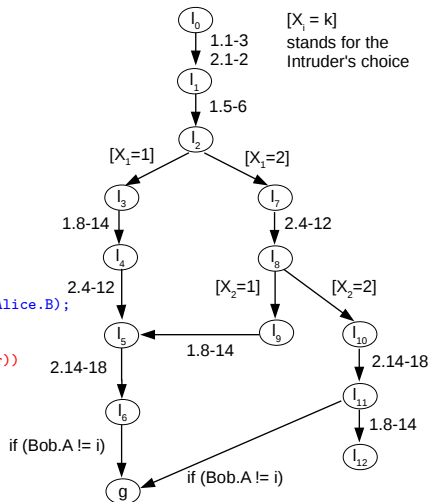
```
Alice(Actor,B:agent){  
  
  Na:=fresh();  
  Actor->B:{Na.Actor}_pk(B);  
  B->Actor:{Na.?Nb.B}_pk(Actor);  
  Actor->B:{Nb}_pk(B);  
  
}
```

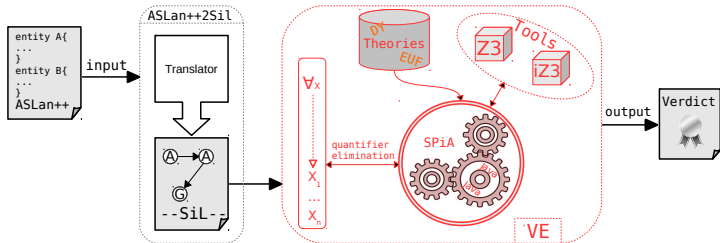
```
Bob(Actor,A:agent){  
  
  ?->Actor:{?Na.?A}_pk(Actor);  
  Nb:=fresh();  
  Actor->A:{Na.Nb.B}_pk(A);  
  A->Actor:{Nb}_pk(Actor);  
  
}
```

- Combining sessions
- Input variables X_i to switch between sessions

```

1.1) Alice.Actor := a;
1.2) Alice.B := Y_1;
1.3) IK := {a,b,i,pk_a,pk_b,pk_i,pk_i^-1};
1.4)
1.5) Alice.Na := c_1;
1.6) IK := IK + {Alice.Na,Alice.Actor}_pk(Alice.B);
1.7)
1.8) if (IK |-
{Alice.Na,?Alice.Nb,Alice.B}_pk(Alice.Actor))
1.9) then
1.10) Alice.Nb = Y_2;
1.11) else
1.12) end
1.13)
1.14) IK := IK + {Alice.Nb}_pk(Alice.B);
    
```





$$\text{Init} \frac{}{\{l_0, s_0\}, A_0, G_0}$$

$$\text{Decide} \frac{Q, A, G}{Q + (l_2, s_2), A, G}$$

$$\begin{aligned} q &= (l_1, s_1) \in Q \\ e &= (l_1, a, l_2) \in \Delta \\ \neg B(q, A(e)) \\ s_2 &\in SI(a)(s_1) \\ \neg B((l_2, s_2), A(l_2)) \end{aligned}$$

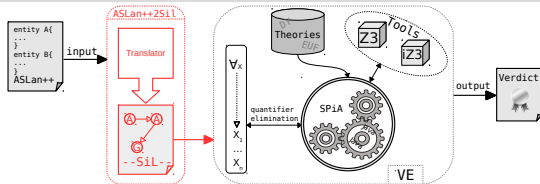
$$\text{Learn} \frac{Q, A, G}{Q, A + e: \phi, G}$$

$$\begin{aligned} q &= (l_1, s_1) \in Q \\ e &= (l_1, a, l_2) \in \Delta \\ B(q, \phi) \\ J(e: \phi, A) \end{aligned}$$

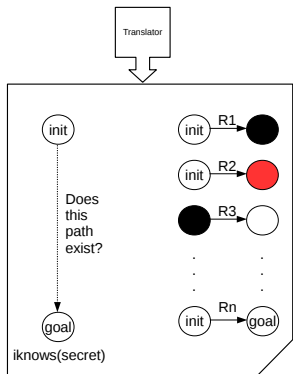
$$\text{Conjoin} \frac{Q, A, G}{Q - q, A + l: \phi, G - l}$$

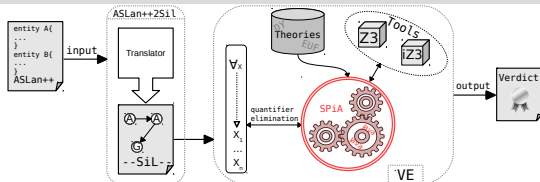
$$\begin{aligned} q &= (l, s) \in Q \\ \neg B(q, A(l)) \\ \forall e \in \text{Out}(l), e: \phi_e \in A \wedge B(q, \phi_e) \\ \phi &= \bigwedge \{\phi_e \mid e \in \text{Out}(l)\} \end{aligned}$$

- Decide symbolically executes one program action
- Learn used to generate annotations
- Conjoin used to backtrack and merge annotations coming from different branches

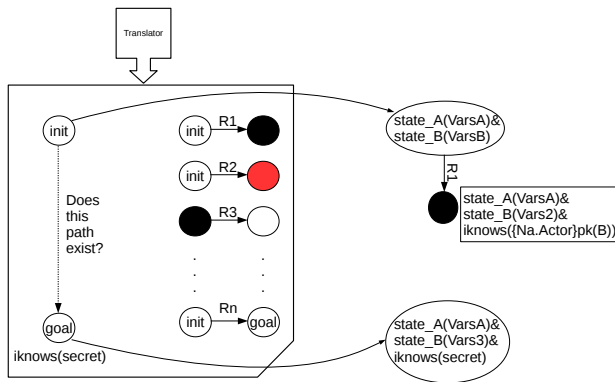


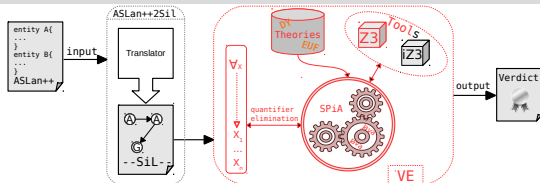
$A \rightarrow B$:	$\{N_A, A\}_{pk(B)}$
$B \rightarrow A$:	$\{N_A, N_B, B\}_{pk(A)}$
$A \rightarrow B$:	$\{N_B\}_{pk(B)}$



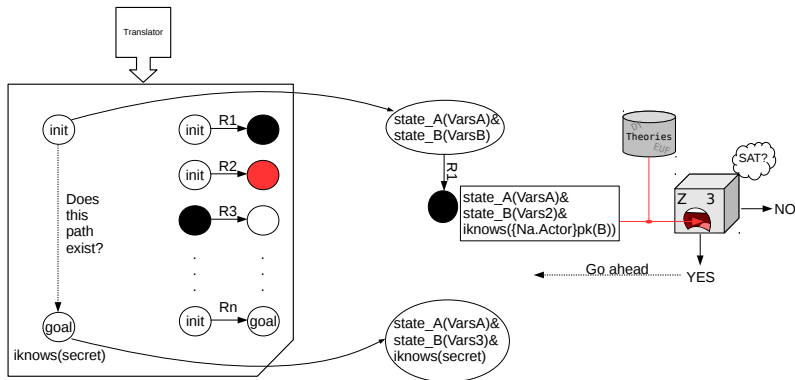


$A \rightarrow B$:	$\{N_A, A\}_{pk(B)}$
$B \rightarrow A$:	$\{N_A, N_B, B\}_{pk(A)}$
$A \rightarrow B$:	$\{N_B\}_{pk(B)}$





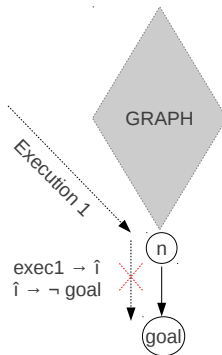
$A \rightarrow B$:	$\{N_A, A\}_{pk(B)}$
$B \rightarrow A$:	$\{N_A, N_B, B\}_{pk(A)}$
$A \rightarrow B$:	$\{N_B\}_{pk(B)}$



Craig's Interpolation

In FOL, if $\alpha \wedge \beta$ is inconsistent, then there exists $\hat{\imath}$ s.t.

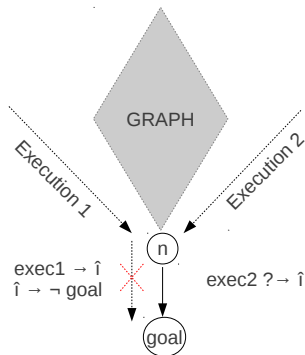
- α implies $\hat{\imath}$
- $\hat{\imath}$ implies $\neg\beta$
- $\mathcal{L}(\hat{\imath}) \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$



Craig's Interpolation

In FOL, if $\alpha \wedge \beta$ is inconsistent, then there exists $\hat{\imath}$ s.t.

- α implies $\hat{\imath}$
- $\hat{\imath}$ implies $\neg\beta$
- $\mathcal{L}(\hat{\imath}) \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$



Craig's Interpolation

In FOL, if $\alpha \wedge \beta$ is inconsistent, then there exists $\hat{\imath}$ s.t.

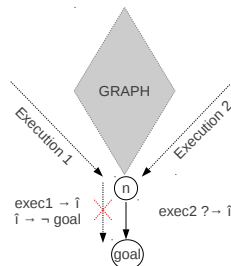
- α implies $\hat{\imath}$
- $\hat{\imath}$ implies $\neg\beta$
- $\mathcal{L}(\hat{\imath}) \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$

We can define α and β as follows:

- $\alpha = PC \bigwedge_{v \in Var} v = Env(v)$
- $\beta = Sem(a) \wedge \neg ann'$

where:

- PC is a conjunction of *path constraints*
- Var is the set of *program variables*
- Env is the *environment*
- $Sem(a)$ is the *semantics* (expressed as a transition formula) of the last *action* a
- ann is the current *annotation* of the node

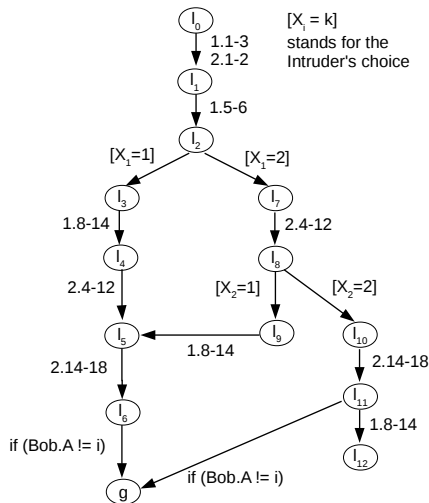


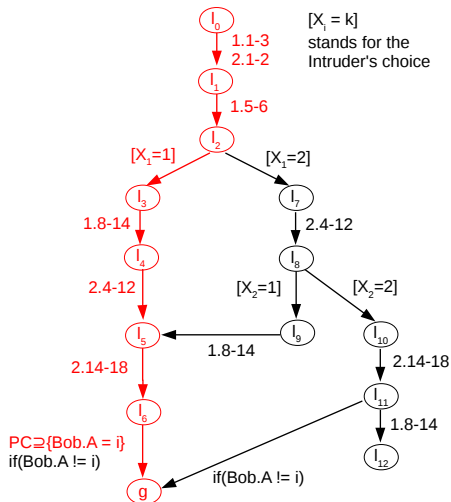
- new speed-up technique for security protocol verification
- DY creates huge and interleaved (sessions) CFGs
- “cheap” wrt verification technique

cheap interpolants

- 1 suppose a path $\dots S \xrightarrow{a} S'$
- 2 Decide: sat solving on “ S' plus DY theory”
- 3 Learn:
 - $\alpha := S$ with DY
 - $\beta := \text{sem}(a)$ and previous annotations with DY
 - sat check on $\alpha \wedge \beta$ (refutation by resolution steps)
 - interpolant in linear time from refutation (“An interpolating theorem prover”, K.McMillan)

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification

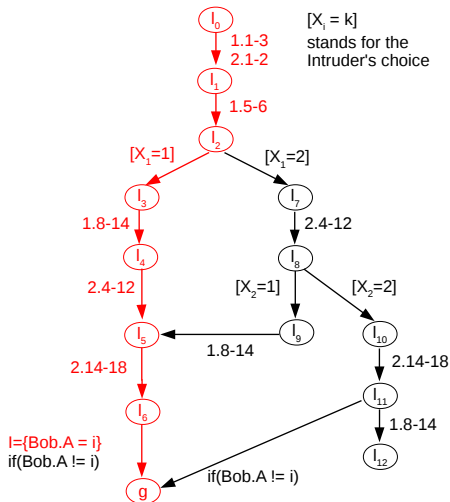




Learn on (l_6, g)

$\alpha \Rightarrow \hat{\alpha} \Rightarrow \neg \beta$

$\hat{\alpha} = \{Bob.A = i\}$



Learn on (l_5, l_6)

$\alpha \Rightarrow \hat{1} \Rightarrow \neg\beta$

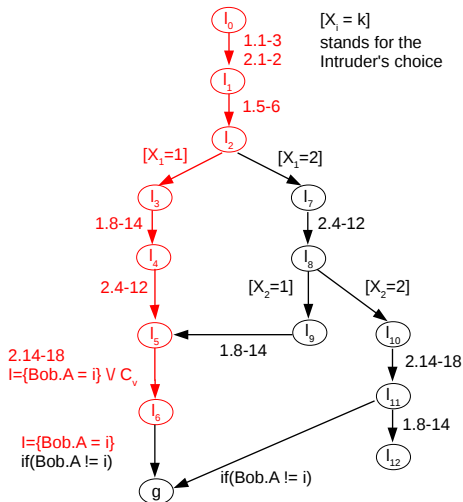
$\hat{1} = \{Bob.A = i\} \vee C_V$

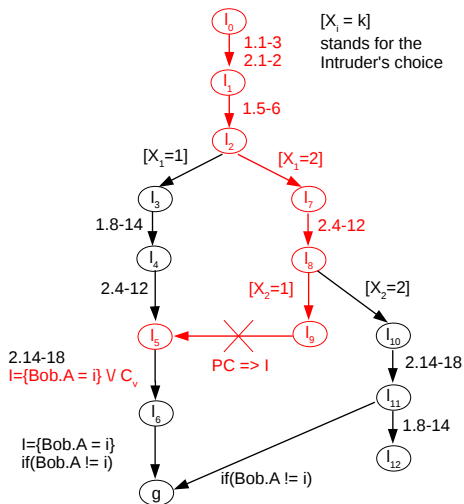
$C_V \in \mathcal{L}(Var)$

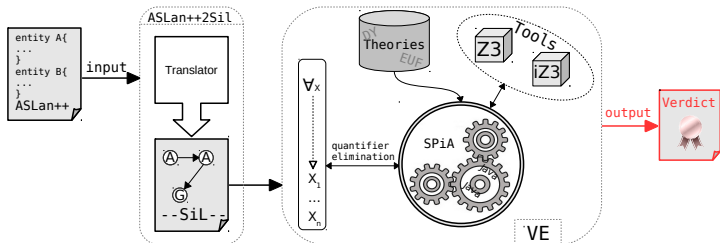
is a constraint

over Var s.t. C_V entails

$IK \not\models \{Bob.Nb\}_{pk\{Bob.Actor\}}$







Without Lowe's fix we obtain a MITM attack:

$$\begin{array}{ll} a \rightarrow i & : \{c_1, a\}_{pk(i)} \\ i(a) \rightarrow b & : \{c_1, a\}_{pk(b)} \\ b \rightarrow i(a) & : \{c_1, c_2\}_{pk(i(a))} \\ i \rightarrow a & : \{c_1, c_2\}_{pk(a)} \\ a \rightarrow i & : \{c_2\}_{pk(i)} \\ i(a) \rightarrow b & : \{c_2\}_{pk(b)} \end{array}$$

That is the usual MITM attack on NSPK protocol:

$$\begin{array}{ll} A \rightarrow i & : \{N_A, A\}_{pk(i)} \\ i(A) \rightarrow B & : \{N_A, A\}_{pk(B)} \\ B \rightarrow i(A) & : \{N_A, N_B\}_{pk(A)} \\ i \rightarrow A & : \{N_A, N_B\}_{pk(A)} \\ A \rightarrow i & : \{N_B\}_{pk(i)} \\ i(A) \rightarrow B & : \{N_B\}_{pk(B)} \end{array}$$

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification

Specification (sessions)	Full-explore: Decide (time)	SPiA: Decide+Learn (time)	speedup (%)	Result
ISO6 (ab,ab)	467* (278m12s)	311+274 (205m6s)	✓ (-26.28%)	no attack found
NSL (ab,ab)	631 (173m7s)	257+234 (57m37s)	✓ (-66.71%)	no attack found
NSL (ai,ab)	119 (1m49)	89+22 (1m30s)	✓ (-17.43%)	no attack found
NSPK (ab,ab)	631 (76m20s)	257+234 (26m5s)	✓ (-65.82%)	no attack found
NSPK (ai,ab)	123 (0m51s)	101+22 (0m56s)	✗ (+8.92%)	attack found
Helsinki (ab,ab)	660* (261m47s)	311+274 (112m7s)	✓ (-57.17%)	no attack found
Helsinki (ai,ab)	407 (46m44s)	167+88 (13m41)	✓ (-70.72%)	attack found

- Marco Rocchetto, Luca Viganò, and Marco Volpe - "An interpolation-based method for the verification of security protocols". Submitted to a Journal
- Marco Rocchetto, Luca Viganò, and Marco Volpe - "Using Interpolation for the Verification of Security Protocols" (Extended Abstract). In the informal proceedings of Interpolation: From Proofs to Applications (2014)
- Marco Rocchetto, Luca Viganò, Marco Volpe, and Giacomo Dalle Vedove - "Using Interpolation for the Verification of Security Protocols". In the proceedings of Security and Trust Management (2013)
- Giacomo Dalle Vedove, Marco Rocchetto, Luca Viganò, and Marco Volpe - "Using Interpolation for the Verification of Security Protocols" (Extended Abstract). In the informal proceedings of FCS Workshop on Foundations of Computer Security (2013)

Specification (sessions)	Full-explore: Decide (time)	SPiA: Decide+Learn (time)	speedup (%)	Result
ISO6 (ab,ab)	467* (278m12s)	311+274 (205m6s)	✓ (-26.28%)	no attack found
NSL (ab,ab)	631 (173m7s)	257+234 (57m37s)	✓ (-66.71%)	no attack found
NSL (ai,ab)	119 (1m49)	89+22 (1m30s)	✓ (-17.43%)	no attack found
NSPK (ab,ab)	631 (76m20s)	257+234 (26m5s)	✓ (-65.82%)	no attack found
NSPK (ai,ab)	123 (0m51s)	101+22 (0m56s)	✗ (+8.92%)	attack found
Helsinki (ab,ab)	660* (261m47s)	311+274 (112m7s)	✓ (-57.17%)	no attack found
Helsinki (ai,ab)	407 (46m44s)	167+88 (13m41)	✓ (-70.72%)	attack found

- Marco Rocchetto, Luca Viganò, and Marco Volpe - "An interpolation-based method for the verification of security protocols". Submitted to a Journal
- Marco Rocchetto, Luca Viganò, and Marco Volpe - "Using Interpolation for the Verification of Security Protocols" (Extended Abstract). In the informal proceedings of Interpolation: From Proofs to Applications (2014)
- Marco Rocchetto, Luca Viganò, Marco Volpe, and Giacomo Dalle Vedove - "Using Interpolation for the Verification of Security Protocols". In the proceedings of Security and Trust Management (2013)
- Giacomo Dalle Vedove, Marco Rocchetto, Luca Viganò, and Marco Volpe - "Using Interpolation for the Verification of Security Protocols" (Extended Abstract). In the informal proceedings of FCS Workshop on Foundations of Computer Security (2013)

Specification (sessions)	SATMC (v.3.4) time	CL-AtSe (v.2.5-21)			OFMC (v.2012c)		Result
		transitions	states	time	nodes	time	
ISO6 (ab,ab)	6.318s	452	236	0.034s	8432	3.804s	no attack found
NSL (ab,ab)	14m28s	794	534	0.052s	3236	3.295s	no attack found
NSL (ai,ab)	6m51s	93	69	0.015s	575	0.327s	no attack found
NSPK (ab,ab)	14m10s	794	534	0.053s	8180	3.208s	no attack found
NSPK (ai,ab)	1m56s	14	10	0.014s	96	0.134s	attack found
Helsinki (ab,ab)	7.01s	794	534	0.061s	8180	3.795s	no attack found
Helsinki (ai,ab)	50.8s	14	10	0.017s	96	0.121s	attack found

Specification (sessions)	Full-explore: Decide (time)	SPiA: Decide+Learn (time)	speedup (%)	Result
ISO6 (ab,ab)	467* (278m12s)	311+274 (205m6s)	✓ (-26.28%)	no attack found
NSL (ab,ab)	631 (173m7s)	257+234 (57m37s)	✓ (-66.71%)	no attack found
NSL (ai,ab)	119 (1m49)	89+22 (1m30s)	✓ (-17.43%)	no attack found
NSPK (ab,ab)	631 (76m20s)	257+234 (26m5s)	✓ (-65.82%)	no attack found
NSPK (ai,ab)	123 (0m51s)	101+22 (0m56s)	✗ (+8.92%)	attack found
Helsinki (ab,ab)	660* (261m47s)	311+274 (112m7s)	✓ (-57.17%)	no attack found
Helsinki (ai,ab)	407 (46m44s)	167+88 (13m41)	✓ (-70.72%)	attack found

Other techniques

- Partial Order Reduction (POR)
- Constraint Differentiation (CDiff)

Other tools

- IntraLA, IMPACT II, for C source code (drivers) - different goals, no DY
- OFMC, CL-AtSe, SATMC - POR, CDiff, rules rewriting
- MaudeNPA - super lazy intruder, POR

- 1 Introduction
- 2 Design time security verification
 - History
 - Security Protocol interpolation Method (SPiM)
 - Example
 - The SPiM tool
- 3 Runtime security verification

Initial intuitions

- 1 we can use the “design-approach” for web applications
- 2 there is no need of DY crypto rule to validate web applications wrt XSS

Experiments:

- 37 specifications, SATMC with(out) crypto rules
- The crypto rule of DY is not used to detect XSS in WebGoat specs as reported in a SPaCloS deliv
- then maybe it is not the right way to validate webapps

Initial intuitions

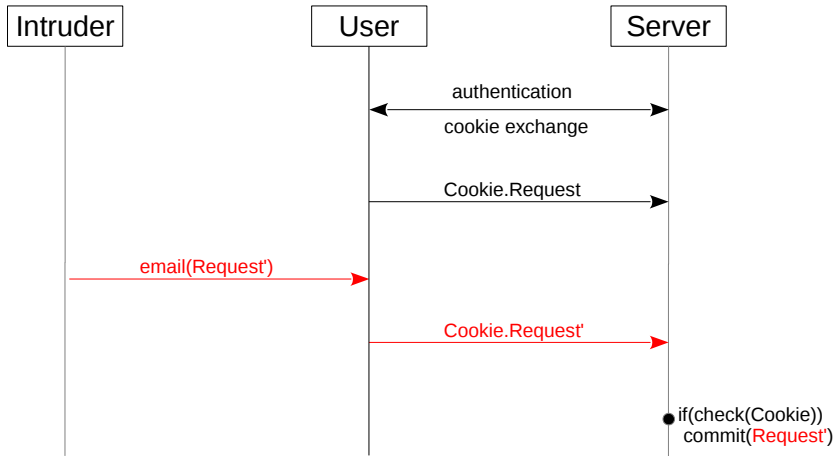
- 1 we can use the “design-approach” for web applications
- 2 there is no need of DY crypto rule to validate web applications wrt XSS

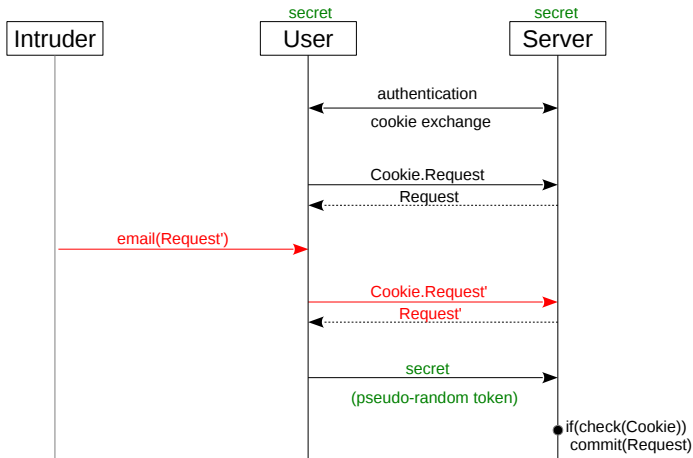
Experiments:

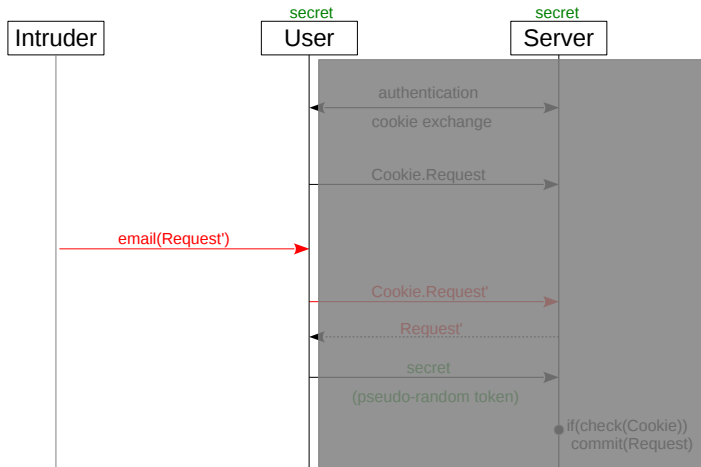
- 37 specifications, SATMC with(out) crypto rules
- The crypto rule of DY is not used to detect XSS in WebGoat specs as reported in a SPaCloS deliv
- then maybe it is not the right way to validate webapps

How can we model a webapp?

- message exchange
- intruder rules
- goals (e.g. XSS, CSRF ...)







CSRF - OWASP definition

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email/chat), an attacker may force the users of a web application to execute actions of the attacker's choosing

- intruder doesn't see the communication client \leftrightarrow server
- client acts as an oracle for the intruder
- no crypto rule of DY is needed to detect CSRF but it can be used to find flaws in the logic of the application (and bypass CSRF protections)
- Adam Doupé, Marco Cova and Giovanni Vigna - "Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners"

Web application spec

The user acts as an Oracle for the intruder then we model the Oracle and the Server

Intruder

The crypto part could be ignored for the CSRF but it can be used to find flaws in the logic of the application (and bypass CSRF protections)

Goal

We check if there's a way to let the intruder ask for a resource of the web server

Example

Suppose we have an authentication procedure that produce a cookie that the system will use to check requests (e.g. Kerberos, WebAuth)

```
?->Actor: ?Req;  
Actor ->* Server: Actor.Cookie.Req;
```

ASLan++

- ?, save messages / don't care
- Actor, like this in Java

```
while(true){  
  select{  
    on(? ->* Actor: ?User.?Cookie.?Req):{  
      if(cookies(Actor)->contains((User, Cookie))){  
        commit(Req);  
      }  
    }  
  }  
}
```


```
csrf_goal: [](!commit(intruderRequest));
```


- WebGoat - straightforward attacks
- eHealth - no protection against CSRF
- DocumentRepository - SSL + CSRF Token (safe)
- WebAuth - authentication and CSRF
- **Unicredit** bank - unsafe

Marco Rocchetto, Martín Ochoa and Mohammad Torabi Dashti.
“Model-Based Detection of CSRF.” ICT Systems Security and
Privacy Protection (2014)

- Devdatta Akhawe, Adam Barth, Peifung E. Lam, John Mitchell, Dawn Song - “Towards a Formal Foundation of Web Security”
- Matthias Buchler, Johan Oudinet, Alexander Pretschner - “SPaCiTE — Web Application Test Engine ”

localhost/unicredit2/UniCredit2.html

 Servizio Clienti **800.57.57.57** (Estero +39 02.33408973)

FAQ CONTATTI MODIFICA PIN ESCI

HOME CONTI CARTE COMUNICAZIONI PROFILO NEGOZIO ONLINE

Nascondi


HELP MENU

CONTI CORRENTI
MONEYBOX CD
CONTI DI DEPOSITO
BONIFICI E GIROCONTI
Bonifico Italia
Bonifici periodici
Giroconto
Bonifico Europeo - SEPA
Bonifico Estero OnLine
Bonifico Estero multiplo
Trasferimento estero convenzionato
Ultimi bonifici/giroconti
Lista bonifici
Lista disposizioni estero
Lista bonifici SEPA
Lista disposizioni
IMPOSTE E TASSE
RICARICHE
BOLLETTE E UTENZE
ALTRI PAGAMENTI
ARCHIVIO PAGAMENTI
DOCUMENTI ONLINE
OPERAZIONI VELOCI

Inserimento dati

Conferma

3 Riepilogo

 **BONIFICO DISPOSTO CORRETTAMENTE**

ATTENZIONE: il bonifico verso altre Banche può essere annullato entro le ore 20.00 di oggi o - in caso di richiesta di esecuzione in data successiva a quella odierna - entro le ore 20 del giorno lavorativo precedente la data di esecuzione richiesta. Il bonifico su nostra Banca con data esecuzione corrispondente alla data odierna viene eseguito in tempo reale e non è annullabile. Può essere annullato solo in caso di richiesta di esecuzione in data successiva a quella odierna, entro le ore 20.00 del giorno lavorativo precedente la data di esecuzione richiesta. Per annullare il bonifico clicchi qui (e sceglia Bonifici e giroconti).


DATI ORDINANTE

N° rapporto
IT 22 L 02008 11773 000100971596

Ordinante:
ROCCHETTO MARCO

DATI BONIFICO

Beneficiario:
MRCO ROCCHETTO

 Per effettuare bonifici senza inserire password dispositive attivi le Operazioni Veloci

Indirizzo:

Località:

Prov.

CAP:

IBAN:
IT 38 S 03169 01600 000321027757

Banca:
ING DIRECT N.V.

Sede:
SEDE

Importo:
1.00 EUR

Commissioni:
1.10

M.Rocchetto

PhD Thesis

Verona, May 8, 2015

46

Design time verification of security protocols:

- showed how to use IntraLA in the verification of security protocols
- showed that “interpolants as annotations” can concretely speed-up the verification of security protocols
- SPiM

Runtime verification of web applications:

- defined how to model a web app to search for CSRF
- noticed that the DY intruder on his crypt side is not used for CSRF but possibly to bypass CSRF protection
- CSRF on major EU bank

Design time verification of security protocols:

- More complex protocols and goals (LTL)
- Test case generation and integration in testing phase
- POR + symbolic execution (+ interpolation?)
- CDiff + symbolic execution (+ interpolation?)

Runtime verification of web applications:

- extend to other attacks, e.g., XSS, SQL Injections
- study how to use connection between attacks
(ChainedAttacks - SPaCloS extension)

Thank you
Any questions?