Model-based Security Engineering: Models vs. Code

Jan Jürjens

Department of Computing
The Open University, GB

J.Jurjens@open.ac.uk

http://www.umlsec.org
Challenge: Security

Security is holistic property:
• Attackers often circumvent (not: break) mechanisms.
• Transform (in)secure components to secure systems?

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (B. Lampson / R. Needham).
Model-based Security Engineering

Requirements

Weave in

Analyse against

(UML) Models

Idea: Extract models from artefacts in development and use of software.

Verify.

Gener.

Reverse Engin.

Source Code

Configure

Tool-supported, theoretically sound, efficient automated security design & analysis.
Secure System Lifecycle

Model-based Security Engineering

Design: Encapsulate prudent security engineering rules.
Analysis: Formally based, automated, efficient tools.
Note: emphasis on high-level requirements.

[McGraw 2003]
Model-based Security with UMLsec

Extension of the Unified Modeling Language (UML) for secure systems development.

• evaluate UML models for security
• encapsulate established rules of prudent secure engineering
• make available to developers not specialized in secure systems
• consider security requirements from early design phases, in system context
• can use in certification
UMLsec

Insert recurring security requirements, adversary scenarios, security mechanisms as predefined markers. Use associated logical constraints to verify specifications using model checkers and ATPs based on formal semantics. Ensures that UML specification enforces the relevant security requirements wrt Dolev-Yao type adversaries. [FASE01, UML02, FOSAD05, ICSE05]
Example: Crypto-based Distributed System

Adversary knowledge:
- $k^{-1}$, $y$, $x$
- $\{z\}_k$, $z$

(cf. [Dolev, Yao 1982])

Attacker may ...
- control system parts,
- know data in advance,
- intercept messages,
- delete messages,
- inject messages.

Jan Jürjens, Open University: Model
Security Analysis in First-order Logic

Approximate adversary knowledge set from above:

Predicate \textit{knows}(E) meaning that adversary may get to know \( E \) during the execution of the system.

E.g. secrecy requirement:
For any secret \( s \), check whether can derive \textit{knows}(s) from model-generated formulas using automatic theorem prover. \[ \text{ICSE05} \]
Cryptographic Expressions I

$Exp$: quotient of term algebra generated from sets $Data$, $Keys$, $Var$ of symbols using

- $\_::\_$(concatenation), $head(\_), tail(\_)$
- $\_^{-1}$(inverse keys)
- $\{\_\}_$(encryption)
- $Dec(\_)$ (decryption)
- $Sign(\_)$ (signing)
- $Ext(\_)$ (extracting from signature)

under equations …
Cryptographic Expressions II

\[ \forall \forall E,K. \ Dec_K^{-1}(\{E\}_K) = E \]

\[ \forall \forall E,K. \ Ext_K(Sign_K^{-1}(E)) = E \]

\[ \forall \forall E_1,E_2. \ head(E_1 :: E_2) = E_1 \]

\[ \forall \forall E_1,E_2. \ tail(E_1 :: E_2) = E_2 \]

• Associativity for ::.

Write \( E_1 :: E_2 :: E_3 \) for \( E_1 :: (E_2 :: E_3) \) and \( fst(E_1 :: E_2) \) for \( head(E_1 :: E_2) \) etc.

Can include further crypto-specific primitives and laws (XOR, ...).
Example: Translation to Logic

\[
\begin{align*}
\text{C: Client} & \quad \text{S: Server} \\
\text{init}(N, K_C, \text{Sign}_{K_C^{-1}}(C::K_C)) \quad & \rightarrow \\
\text{resp} \left( \{\text{Sign}_{K_S^{-1}}(K::\text{init}_1)\}_{\text{init}_2}, \text{Sign}_{K_{CA}^{-1}}(S::K_S) \right) \\
\text{xchd}(\{s\}_k) \quad & \leftarrow \\
\text{ext}_{K_{CA}}(c_S) = S \land (\text{ext}_{K''}(\text{Dec}_{K_C^{-1}}(c_k))) = N \\
\end{align*}
\]

\[
\begin{align*}
\text{knows}(N) \land \text{knows}(K_C) \land \text{knows}(\text{Sign}_{K_C^{-1}}(C::K_C)) \\
\land \forall \text{init}_1, \text{init}_2, \text{init}_3. [\text{knows}(\text{init}_1) \land \text{knows}(\text{init}_2) \land \\
\text{knows}(\text{init}_3) \land \text{snd}(\text{ext}_{\text{init}_2}(\text{init}_3)) = \text{init}_2 \\
\text{knows}(\{\text{Sign}_{K_S^{-1}}(\ldots)\}_{\ldots}) \land [\text{knows}(\text{Sign}_{\ldots})] \\
\land \forall \text{resp}_1, \text{resp}_2. [\ldots] \ldots]]
\end{align*}
\]
Analysis

Check whether can derive \textit{knows(s)} e.g. using e-Setheo.

Surprise: Yes!

\rightarrow Protocol does not preserve secrecy of \textit{s}.

Why? Use Prolog-based attack generator.
Man-in-the-Middle Attack

$C \rightarrow A \rightarrow S$

$N_i::K_C :: \text{Sign}_{K_C^{-1}}(C::K_C)$

$N_i::K_A :: \text{Sign}_{K_A^{-1}}(C::K_A)$

$A \leftarrow S$

$\{\text{Sign}_{K_S^{-1}}(K_j::N_i)\}_{K_A :: \text{Sign}_{K_{CA}^{-1}}(S::K_S)}$

$A \leftarrow S$

$C \leftarrow A$

$\{\text{Sign}_{K_S^{-1}}(K_j::N_i)\}_{K_C :: \text{Sign}_{K_{CA}^{-1}}(S::K_S)}$

$C \rightarrow A \rightarrow S$

$\{s\}_{K_j}$

$\{s\}_{K_j}$

$\rightarrow S$
The Fix

e-Setheo: Proof that *knows*(s) not derivable.

Note completeness of FOL (but also undecidability).
Security Analysis: Model or Code ?

Model:
+ earlier (less expensive to fix flaws)
+ more abstract ➔ more efficient
- more abstract ➔ may miss attacks
- programmers may introduce security flaws
- even code generators, if not formally verified

Code:
+ „the real thing“ (which is executed)

➔ Do both !

Surprise: Essentially no existing work (eg for crypto prots) !
Code Analysis vs. Model Analysis

Options:

• generate code from models
  ➔ currently not available in general

• generate models from code
  ➔ next slides

• create models and code manually and verify code against models
  ➔ later in this talk
Generate control flow graph (e.g. aicall (Absint)).

Transform to state machine:

\[ \text{trans(state, inpattern, condition, action, nextstate)} \]

where action can be outpattern or localvar:=value.

[ASE05, ASE06]
Real Life Challenges...
Experiences

Can generate behavioral models from code (e.g. CFGs). Problem: too concrete

→ understanding + automated verification hard (even with annotations).

Constructing abstract specifications from practical software is manually intensive.
Code Analysis vs. Model Analysis

Options:

• generate code from models
  ➔ currently not possible in general

• generate models from code
  ➔ challenging

• create models and code manually and verify code against models
  ➔ next slides
Verify Code against Models

Assumption: Have textual specification.

Then:

• construct interface spec from textual spec
• analyze interface spec for security
• verify that software satisfies interface spec
Model vs. Implementation

- „meaning“
  - Backtrace assignments
  - Sent and received data
  - Implement-ation
    - .java
  
- „meaning“
  - Defined during model creation
  - Elements of connections
  - Has
  - Equal?

Compare meaning!

Find

Implement-ation

Jessie – using RSA & Server authentication

Abstract model

Jan Jürjens, Open University: Model-based Security Engineering [with David Kirscheneder]
Example: Interface spec of SSL

I) Identify program points:
value \( r \), receive \( p \), guard \( g \), send \( q \)

II) Check guards enforced
<table>
<thead>
<tr>
<th>Parameter der kryptographischen ClientHello Nachricht</th>
<th>Effektiv übertragene Daten der ClientHello Nachricht der Jessie Implementierung</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>type.getValue()</td>
</tr>
<tr>
<td>Pver</td>
<td>major</td>
</tr>
<tr>
<td></td>
<td>minor</td>
</tr>
<tr>
<td></td>
<td>(((gmtUnixTime &gt;&gt;&gt; 16) &amp; 0xFF)</td>
</tr>
<tr>
<td></td>
<td>(((gmtUnixTime &gt;&gt;&gt; 8) &amp; 0xFF)</td>
</tr>
<tr>
<td></td>
<td>(gmtUnixTime &amp; 0xFF)</td>
</tr>
<tr>
<td>r_c</td>
<td>randomBytes</td>
</tr>
<tr>
<td></td>
<td>sessionId.length</td>
</tr>
<tr>
<td>Sid</td>
<td>sessionId</td>
</tr>
<tr>
<td></td>
<td>(((suites.size() &lt;&lt; 1) &gt;&gt;&gt; 8) &amp; 0xFF)</td>
</tr>
<tr>
<td></td>
<td>(((suites.size() &lt;&lt; 1) &amp; 0xFF)</td>
</tr>
<tr>
<td>LCip</td>
<td>suites_1</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>suites_N</td>
</tr>
<tr>
<td>LKomp</td>
<td>comp.size()</td>
</tr>
<tr>
<td></td>
<td>comp_1</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>comp_N</td>
</tr>
</tbody>
</table>

Implementation of SSL: Identify Values
public void write(OutputStream out) throws IOException {
    ... out.write(randomBytes); ... 
}

public void write(OutputStream out) throws IOException {
    ... random.write(out); ... 
}

ClientHello(... , Random random, ) {
    ... this.random = random; ... 
}

ClientHello clientHello = new ClientHello(..., clientRandom,...);

Random clientRandom =
    new Random(..., session.random.generateSeed(28));

class SecureRandom (specified in: FIPS 140-2, RFC 1750) of package java.security
Function: generateSeed

Identify: randomBytes

2nd parameter of Random constructor called by ClientHello.write()

2nd parameter of ClientHello constructor initialized in SSLSocket.doClientHandshake()

 initialization of the used Random object

"meaning"
Sending Messages

Automate this using patterns

SSLContext.doClientHandshake()

ClientHello.write()

Random.write()

ProtocolVersion.write()

Handshake.write()

traverse CFG

call of

OutputStream.write()

Security Engineering
Checking Guards

Guard $g$ enforced by code?

a) Generate runtime check for $g$ at $q$ from diagram: simple + effective, but performance penalty.

b) Testing against checks (symbolic crypto for inequalities).

c) Automated formal local verification: conditionals between $p$ and $q$ logically imply $g$ (using ATP for FOL).
public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {
    checkTrusted(chain, authType);
}
calls checkTrusted()

Guard:
checkServerTrusted()

private void checkTrusted(X509Certificate[] chain, String authType) throws CertificateException {
    ...
}
calls verify() for every member of certificate chain

public void verify(PublicKey key, String provider) throws CertificateException, ...
{
    ...
}
calls doVerify()

private void doVerify(Signature sig, PublicKey key) throws CertificateException, ...
{
    sig.initVerify(key);
    sig.update(tbsCertBytes);
    if (!sig.verify(signature))
    {
        throw new CertificateException("signature not validated");
    }
}
msg = Handshake.read(din, certType);

session.trustManager.checkServerTrusted(peerCerts, suite.getAuthType());

only possible way without throwing exception

msg = new Handshake(Handshake.Type.CLIENT_KEY_EXCHANGE, ckex);
msg.write(dout, version);
Verification of Guards in Code

send: represents send command

\( g \): FOL formula with symbols \( \text{msg}_n \) representing \( n^{th} \) argument of message received before program fragment \( p \) is executed

\([d] p^2 g : g \) checked in any execution of \( p \) initially satisfying \( d \) before any send
write \( p^2 g \) for \([\text{true}] p^2 g \).

\([d] \text{if } c \text{ then } p \text{ else } q \models g \) \( (c \land d \Rightarrow g, \text{ no send in } q) \)
Some Rules (Simplified)

\[
[d] \text{if } c \text{ then } p \text{ else } q \models g \quad (c \land d \Rightarrow g, \text{ no send in } q)
\]

\[
[d] \text{if } c \text{ then } p \text{ else } q \models g \quad (\neg c \land d \Rightarrow g, \text{ no send in } p)
\]

\[
[d]p \models g \quad (d \Rightarrow c)
\]

\[
[d]p \models g \quad (d \Rightarrow \neg c)
\]

\[
[d]q \models g \quad (d' \Rightarrow d)
\]

\[
[d]p \models g \quad (d' \Rightarrow d)
\]

\[
x := e; p \models g \quad d \Rightarrow x = e
\]
Java editor

UML editor

Java code

UMLsec model

Code with Assert's; Tests

Text Report

Attack Trace

Control Flow Graph

Assertion/Test Generator

Analyzer

Local Code Checker

Automated Theorem Prover

Attack generator

Security Analyzer

FOL formula

Prolog prog.

Tool Support

[UML04, FASE05, ICSE06]
Some Applications

Analyzed designs / implementations / configurations for

• biometry, smart-card or RFID based identification
• authentication (crypto protocols)
• authorization (user permissions, e.g. SAP systems)

Analyzed security policies, e.g. for privacy regulations.
Conclusion

Seemingly first approach to formally based security verification for crypto-based Java implementations.
As far as possible automated and relatively efficient due to abstraction tailored to verification problem.
Still many challenges to address – collaboration always welcome!
Don‘t miss my Tutorial on Tuesday and Experience Track Talk on Wednesday!
Questions?

More information (papers, slides, tool etc.):
http://www.umlsec.org
J.Jurjens@open.ac.uk