Tools and Techniques for Model-based Testing with UML

Jan Jürjens
Computing Department
The Open University

J.Jurjens@open.ac.uk
http://www.umlsec.org
Testing Critical and Embedded Systems

Very challenging.
For high level of assurance, would need full coverage (test every possible execution).
Usually infeasible (especially reactive systems).
Have heuristics for trade-off between development effort and reliability.

Need to ask yourself:
• How complete is the heuristic?
• How can I validate it?
Model-based System Assurance

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

(tool) Models

Source Code

Requirements

Weave in

Analyze against

Reverse Engin.

Configure

Configure

Verify.

Gener.

tool-supported, theoretically sound, efficient automated design & analysis.
Critical System Lifecycle

Model-based System Assurance

Design: Encapsulate prudent engineering rules.
Analysis: Formally based, automated, efficient tools.
Note: emphasis on high-level requirements.
System Assurance: Model or Code?

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

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

\(\Rightarrow\) Do both where feasible.
Verify Code against Models

Assumption: Have textual specification.

Then:

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

- Elements of connections
  - Sent and received data
  - Implement-ation

- Backtrace assignments
  - „meaning“

- Compare meaning!
  - Defined during model creation

- Has
  - Equal?
  - Find

- Abstract model

Jan Jürjens, Open University: Model-based Testing with UML [with David Kirscheneder]
How to Verify Code Against Models

Model-based Testing (e.g. based on Real-time UML). Advantages:

• Precise measures for completeness.
• Can be formally validated.

Two complementary strategies:

• Conformance testing
• Testing for criticality requirements
Conformance Testing

Classical approach in model-based test-generation (much literature).
Can be superfluous when using code-generation [except to check your code-generator, but only once and for all].
Works independently of real-time requirements.
Conformance Testing: Caveats

• Complete test-coverage still infeasible (although can measure coverage).

• Can only test code against what is contained in model. Usually, model more abstract than code. May lead to "blind spots".

For both reasons, may miss critical test-cases. Want: "criticality testing".
Criticality Testing: Strategies

**Internal**: Ensure test-case selection from models does not miss critical cases: Select according to information on criticality.

**External**: Test code against possible environment interaction generated from parts of the model (e.g. deployment diagram with information on physical environment).
Criticality Testing

Shortcoming of classical model-based test-generation (conformance testing) motivates „criticality testing“.

Goal: model-based test-generation adequate for critical real-time systems.
Internal Criticality Testing

Need behavioral semantics of used specification language (precise enough to be understood by a tool).

Here: semantics for simplified fragment of UML in „pseudo-code“ (ASMs).

Select test-cases according to criticality annotations in the class diagrams.

Test-cases: critical selections of intended behavior of the system.
External Criticality Testing

Generate test-sequences representing the environment behaviour from the criticality information in the deployment diagrams.
Automated White-Box Testing

- Generate control flow graph.
- Analyze for criticality requirements.
- Use to generate critical test-cases.
Meaning of diagrams stated **informally** in (OMG 2003).

**Ambiguities** problem for

- tool support
- establishing **behavioral properties** (safety, security)

**Need precise** semantics for used part of UML, especially to ensure security requirements.
Tool-support: Pragmatics

Commercial modelling tools: so far mainly syntactic checks and code-generation.

Goal: sophisticated analysis. Solution:

• Draw UML models with editor.
• Save UML models as **XMI** (XML dialect).
• Connect to **verification** tools (automated theorem prover, model-checker ...), e.g. using XMI Data Binding.
CSDUML Framework: Features


Upload UML model (as .xmi file) on website. Analyse model for included critical requirements. Download report and UML model with highlighted weaknesses.
Tool Support

For example:

• **consistency checks**
• **mechanical analysis of complicated requirements** on model level (bindings to model-checkers, constraint solvers, automated theorem provers, …)
• **code generation**
• **test-sequence generation**
• **configuration data analysis** against UML.
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 übertragen 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); ... }

Identify: randomBytes
(in message ClientHello)
2nd parameter of ClientHello constructor

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));

"meaning"

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

SSLSocket.doClientHandshake()

Automate this using patterns

Random.write()

ProtocolVersion.write()

Handshake.write()

ClientHello.write()
Checking Guards

Guard $g$ enforced by code?

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

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

d) 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);
}

Guard:
checkServerTrusted()

private void checkTrusted(X509Certificate[] chain, String authType) throws CertificateException {
    ...  
}

calls verify() for every member of certificate chain

calls doVerify()

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

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);

```
try

catch
```

only possible way without throwing exception

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

msg = new Handshake(Handshake.Type.CLIENT_KEY_EXCHANGE, ckex);
msg.write (dout, version);
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.
Biometric Authentication System

In development by company in joint project.

Uses bio-reference template on smart-card.

Analyze given UML spec.

Discovered three major weaknesses in subsequently improved versions (misuse counter circumvented by dropping / replaying messages, smart-card insufficiently authenticated by mixing sessions). [ACSAC05]
Bank Application

Security analysis of web-based banking application, to be put to commercial use (clients fill out and sign digital order forms).

Layered security protocol (first layer: SSL protocol, second layer: client authentication protocol)

Security requirements:
• confidentiality
• authenticity
Common Electronic Purse Specifications

Global elec. purse standard (Visa, 90% market). Smart card contains account balance, performs crypto operations securing each transaction. Formal analysis of load and purchase protocols: three significant weaknesses: purchase redirection, fraud bank vs. load device owner.

[ASE01]
Load Protocol


Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.
Load «data security»

Card «critical»
{secrecy={KC1}}
{integrity={KC1,cep,nt,rc_nt}}

cep,nt,rc_nt : Data; KC1 : Keys

Init(lda,m)
Credit(s2,r1)

LSAM «critical»
{secrecy={KL1}}
{integrity={KL1,lda,n,r1n,r2ln,rn,mn}}

lda,n,r1n,r2ln,mn : Data
KL1,rn : Keys
Resp1(cep,nt,s1,hc)
RespC(s3,rc)
RespL(s2)

Issuer «critical»
{secrecy={KC1,KL1,rc_m}}
{integrity={KC1,KL1,rc_m}}

rc_m : Data; KL1,KC1 : Keys

Load(cep,lda,m,nt,s1,m1,h1,h2l)
Comp(cep,lda,m,nt,r2l,s3)
Audit Security

No direct communication between card and cardholder. Manipulate load device display.

Use post-transaction settlement scheme.

Relies on secure auditing.

Verify this here (only executions completed without exception).
Flaw I

$m_{l_n}$: „Proof“ for bank that load machine received money. But: $r_n$ shared between bank and load machine.
Flaw II

\( r_{cnt} \): „Proof“ for LSAM that load device received only amount \( m_n \).

But: LSAM cannot prove validity of \( r_{cnt} \).
Conclusions

Model-based Testing using UMLsec:

• formally based approach
• automated tool support
• industrially used methods
• integrated approach (source-code, configuration data)
Overview

Security Engineering

Patterns - Architectures - Aspects - Requirements - Processes

Components - Layers - Models - Code


Runtime Checks - Model-based Testing - Autom. Theorem Proving

Foundations

Applications - Code - Security - Engineering

CEPS - Biometry - Cryptokey - Jessie - TLS variant - Wiesncard

Fault-tolerance - Reliability - Real-time

Risk Assessment - Permissions - Business Processes - Security Investment - Firewall Configurations

IT Security

Secure Systems Development with UML
Questions?

More information (papers, slides, tool etc.):
http://www.umlsec.org

J.Jurjens@open.ac.uk
Backup
Decrease misuse counter

Message order not enforced by smart card!
Authent. Prot. Pt. 2: Problem

Replay MAC_{skh} (FBZ2')

10: return(FBZ2:: Mac_{sksc}(FBZ2))

11: send("writeFBZ2::FBZ2':: Mac_{skh}(FBZ2')")

12: send("getFBZ2'":{"getFBZ2'"} _{skh} )

13: return(FBZ2:: Mac_{sksc}(FBZ2))

14: send("getData" _{sksc}

15: return({tuser::Sign^{inv(ka)}(Hash(idsc::tuser)}) _{sksc} )

[snd(arg_{sc,5,1}) = Mac_{sksc}(snd(arg_{sc,5,1}))]
[Dec_{sksc}(fst(arg_{sc,5,1})) = "getFBZ2"]

[thd(arg_{sc,6,1}) = Mac_{sksc}(snd(arg_{sc,6,1}))]
[fst(arg_{sc,6,1}) = "writeFBZ2"]

FBZ2 := fst(arg_{sc,5,1})

[snd(arg_{sc,7,1}) = Mac_{sksc}(fst(arg_{sc,7,1}))]
[Dec_{sksc}(fst(arg_{sc,7,1})) = "getFBZ2'"]

[Dec_{sksc}(arg_{sc,8,1}) = "getData"]

[mh := Dec_{skh}(arg_{h,7,1})]
[t'user::=fst(mh)]

[Hash(arg_{h,1,1}::t'user) = Ext^{ka}(snd(mh))]

[snd(arg_{h,5,1}) = Mac_{kn}(fst(arg_{h,5,1}))]
[fst(arg_{h,5,1} >0 )]

FBZ2' := fst(arg_{h,5,1}) - 1
Authentic Protocol
Part 1: Problem.

Forged smart-card after authentic.; replay old session key

Mutual authentication with challenge & response

Generate shared key