Satisfiability Modulo Theories
and
Network Verification

Nikolaj Bjørner
Microsoft Research
Formal Methods and Networks Summer School
Ithaca, June 10-14 2013
Lectures

Wednesday 2:00pm-2:45pm:
  An Introduction to SMT with Z3

Thursday 11:00am-11:45am
  Algorithmic principles of SAT/SMT

Friday 9:00am-9:45am
  Theories, Solvers and Applications
Z3 source is (evil) subverting 8/
Z3 is open shared source

http://z3.codeplex.com/
Z3 architecture - new

Tactics

- SMT (legacy core)
  - Bit-Vectors
  - Arrays
  - Lin-arithmetic
  - Recursive Datatypes
  - Free (uninterpreted) functions
  - Quantifier instantiation

- And-then
- Or-else
- Try-for
- Par-or
- Par-then

- SAT core for Bit-vectors
- ∃R: Non-linear real arithmetic
- Floating point arithmetic
- Horn clauses
- Simplification

- SMT-LIB
- Record & replay
- OCaml
- .NET
- C
- Java
- Python
Some Microsoft Tools based on Z3

- Program Verification
- Auditing
- Type Safety
- Testing
- Analysis
- Synthesis
Cool tools using Z3

Sledge Hammer

Liquid Types

Leon Online

ESBMC

Scala Z3

MetiTarski

KeYmaera

Jeves
Testing
Hunting for Security Bugs

- Two main techniques used by “black hats”:
  - Code inspection (of binaries) and Blackbox fuzz testing

- Blackbox fuzz testing:
  - A form of blackbox random testing
  - Randomly fuzz (=modify) a well-formed input
  - Grammar-based fuzzing: rules that encode how to fuzz

- Heavily used in security testing
  - At MS: various internal tools
  - Conceptually simple yet effective in practice...
    - Has been instrumental in weeding out 1000’s of bugs during development and test

Old introduction slide to SAGE & Pex

State today:

Pex is mature & Moles in VS2012
SAGE actively used internally at Microsoft
Method: Dynamic Test Generation

Run program with random inputs.

Gather constraints on inputs.

Use constraint solver to generate new inputs.

Combination with randomization: DART Godefroid-Klarlund-Sen-05,...

Can’t statically generate value for x that satisfy “z==hash(x)”

But we can solve for y in:

\[ x + y = \text{hash}(x) \]
Fuzzing and Test Case Generation

SAGE

Internal. For Security Fuzzing
Runs on x86 instructions

External. For Developers
Runs on .NET code

Try it on: http://pex4fun.com

Finding security bugs before the black hat hackers
Fuzzing and Test Case Generation

SAGE

Internal. For Security

Runs on x86 instructions

Dr. Strangelove?

Bug: ***433
“2/29/2012 3:41 PM Edited by *****
SubStatus -> Local Fix

I think the fuzzers are starting to become sentient. We must crush them before it is too late.

In this case, the fuzzer figured out that if 
[X was between A and B then Y would get set to Z triggering U and V to happen……]

…..

And *if this fuzzer asks for the nuclear launch codes, don’t tell it what they are …*”

Finding security bugs before the hackers
SAGE by numbers

100s CPU-years - largest dedicated fuzz lab in the world

100s apps - fuzzed using SAGE

100s previously unknown bugs found

Billion+ computers updated with bug fixes

Millions of $ saved for Users and Microsoft

10s of related tools (incl. Pex), 100s DART citations

3+ Billion constraints - largest usage for any SMT solver

Adapted from [Patrice Godefroid, ISSTA 2010]
Verification
It is cool
easy
fun
attractive
Building Verve

Source file
- Nucleus.bpl (x86)
- Kernel.obj (x86)

Verification tool
- Boogie/Z3
- TAL checker

Compilation tool
- C# compiler
- Linker/ISO generator

Verified
- Verve.iso

9 person-months

Safe to the Last Instruction / Jean Yang & Chris Hawblitzl PLDI 2010
Validation

SecGuru: Automatic Validation of Network Connectivity Restrictions

Karthick Jayaraman, Charlie Kaufman, and Ramanathan Venkatapathy

Nikolaj Bjørner
Network Policies: Complexity, Challenge and Opportunity

Several devices, vendors, formats
- Net filters
- Firewalls
- Routers

Challenge in the field
- Do devices enforce policy?
- Ripple effect of policy changes

Arcane
- Low-level configuration files
- Mostly manual effort
- Kept working by
  “Masters of Complexity”

Human errors > 4 x DOS attacks

Human Errors by Activity
- Config Changes: 74%
- Device hw/sw updates: 13%
- WA Cluster Setup: 13%
A Data-center Architecture
A Data-center Architecture

Defense in Depth = Crypto + Policies
- outside IP can be spoofed

Efficient and Flexible Defense by Policy only
- inside IP cannot be spoofed

Policies (Access Control Lists) matter
Network Policies
What they look like in routers

Defense in Depth = Crypto + Policies - outside IP can be spoofed

Efficient and Flexible Defense by Policy only - inside IP cannot be spoofed

<table>
<thead>
<tr>
<th>Type</th>
<th>Src Address</th>
<th>Src Port</th>
<th>Remote Address</th>
<th>Remote Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>6</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>17</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>4</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>Deny</td>
<td>*</td>
<td>*</td>
<td>65.52.244.0/22</td>
<td>*</td>
<td>4</td>
</tr>
</tbody>
</table>
A Need for Automating Policy Configuration

Constantly growing

New clusters – “same” policy, but different addresses

Constantly changing

Hardware – capacity, semantics
New Services – selectively exposed
Patches – to live site incidents

Bare metal low-level format of routers are also used for policy configurations
Towards automation:
Checking Policy Configuration with **SecGuru**

**Constantly growing**

*New clusters – enforce and check that new policies are instances of a master template (the intent)*

**Constantly changing**

*Hardware – capacity, semantics
New Services – check effect of new rules
Patches – check for regressions*
Towards automation: Checking Policy Configuration with **SecGuru**

Enforce and check that policies are instances of a master template (the intent)

\[
X := A
\]

- **Rules\(<X>\)**
- **Rules\(<A>\)**
- **Rules on Router**
Towards automation:
Checking Policy Configuration with **SecGuru**

Enforce and check that policies satisfy contracts

\[
X := A \\
\text{Rules}^{\text{<X>}} \rightarrow \text{Rules}^{\text{<A>}} \\
\text{Contracts}^{\text{<X>}} \rightarrow \text{Contracts}^{\text{<A>}} \\
\text{Does policy satisfy contracts?}
\]
# Policies as Bit-Vector Formulas

<table>
<thead>
<tr>
<th>Type</th>
<th>Src Address</th>
<th>Src Port</th>
<th>Remote Address</th>
<th>Remote Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>6</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>17</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>4</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>Deny</td>
<td>*</td>
<td>*</td>
<td>65.52.244.0/22</td>
<td>*</td>
<td>4</td>
</tr>
</tbody>
</table>

**IP, Port, and Protocol:** bit vectors

**Policy:** Bit-vector logic

\[
\begin{align*}
\text{Allow:} & \land (10.20.0.0 \leq rclp 10.20.31.255) \\
& \land (157.55.252.0 \leq dstIp \leq 157.55.252.255) \land
\end{align*}
\]
Policies as Bit-Vector Formulas

<table>
<thead>
<tr>
<th>Type</th>
<th>Src Address</th>
<th>Src Port</th>
<th>Remote Address</th>
<th>Remote Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>6</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>17</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>4</td>
</tr>
<tr>
<td>Allow</td>
<td>10.20.0.0/19</td>
<td>*</td>
<td>157.55.252.0/22</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>Deny</td>
<td>*</td>
<td>*</td>
<td>65.52.244.0/22</td>
<td>*</td>
<td>4</td>
</tr>
</tbody>
</table>

IP, Port, and Protocol: bit vectors
Policy: Bit-vector logic

\[\text{Allow:} \& (10.20.0.0 \leq s rcIp 10.20.31.255) \land (157.55.252.0 \leq dstIp < 157.55.252.255) \land \text{IsAllow} \downarrow\]
Two uses of SecGuru

**Change-Impact Analysis**

- **Policy\_1**
- **Policy\_2**

**SecGuru**

- **Z3**

Policy\_1 \equiv Policy\_2

or

Policy\_1 \oplus Policy\_2

**Contract Validation**

- **Policy**
- **Contracts**

**SecGuru**

- **Z3**

**Information about policy**
Two uses of SecGuru

Does policy permit outgoing traffic to some address in 65.52.244.0/22?

**Query:**

$$(65.52.244.0 \leq dst/p \leq 65.52.247.255)$$

Check Satisfiability of

**Query $\land$ Policy**

Does policy permit connections to all the remote addresses in the range 65.52.244.0/22?

Check Unsatisfiability of

**Query $\land \neg$ Policy**
Two uses of **SecGuru**

Change-Impact Analysis

<table>
<thead>
<tr>
<th>Policy₁</th>
<th>Policy₂</th>
</tr>
</thead>
</table>

**Semantic diff** between policies

Is $Policy₁ \equiv Policy₂$?

If not, print $Policy₁ \oplus Policy₂$

Traffic accepted by $Policy₁$, but not $Policy₂$.

Models for $Policy₁ \land \neg Policy₂$

Traffic accepted by $Policy₂$, but not $Policy₁$.

Models for $Policy₂ \land \neg Policy₁$
Two uses of **SecGuru**

**Change-Impact Analysis**

Semantic diff between policies

Traffic accepted by policy1, but not policy2

<table>
<thead>
<tr>
<th>S.no</th>
<th>SrcIP</th>
<th>SrcPort</th>
<th>DstIP</th>
<th>DstPort</th>
<th>Protocol</th>
<th>TCP Flags</th>
<th>ICMP Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.25.252.0-10.25.252.255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>239.0.0.0-239.255.255.255</td>
</tr>
<tr>
<td>5</td>
<td>10.146.192.0-10.146.192.255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>239.0.0.0-239.255.255.255</td>
</tr>
<tr>
<td>6</td>
<td>210.126.9.11-210.126.9.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>7</td>
<td>210.126.9.11-210.126.9.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>8</td>
<td>210.126.9.11-210.126.9.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>9</td>
<td>216.52.28.197</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>10</td>
<td>216.52.28.197</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
</tbody>
</table>

Traffic accepted by policy2, but not policy1

<table>
<thead>
<tr>
<th>S.no</th>
<th>SrcIP</th>
<th>SrcPort</th>
<th>DstIP</th>
<th>DstPort</th>
<th>Protocol</th>
<th>TCP Flags</th>
<th>ICMP Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0.0.0-10.19.255.255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>5</td>
<td>10.7.32.0-10.7.35.255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>6</td>
<td>10.7.32.0-10.7.35.255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>7</td>
<td>10.7.32.0-10.7.35.255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>8</td>
<td>10.7.51.0-10.7.51.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>9</td>
<td>10.7.51.0-10.7.51.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
<tr>
<td>10</td>
<td>10.7.51.0-10.7.51.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*;</td>
<td>168.63.161.0-168.63.161.15</td>
</tr>
</tbody>
</table>
All-BVSAT: A compact model enumeration

Really naïve model enumeration:

- To generate the \((k+1)\)st model, negate all the \(k\) models seen so far

\[
(\forall i \leftarrow \text{Allow} \downarrow i) \land (\forall j \leftarrow \neg \text{Den} \downarrow j) \land (\forall k \leftarrow \neg \text{Model} \downarrow k) \quad \ldots
\]

2 \(\sim\) 32+16+32+16 models

Smarter model enumeration in SecGuru using All-BVSAT (idea):

- Find initial \(\text{srcIp} \downarrow 0, \text{srcPort} \downarrow 0 \models Policy \downarrow 1 \land \neg Policy \downarrow 2\)

Maximize bounds \(lo|\text{srcIp} \leq srcIp < hi|\text{srcIp} :\)
All-BVSAT: A compact model enumeration

Maximize bounds:

\[\text{lo}_{\text{srcIp}} \leq \text{srcIp} \leq \text{hi}_{\text{srcIp}} \land \text{lo}_{\text{srcPort}} \leq \text{srcPort} \leq \text{hi}_{\text{srcPort}} \land \text{lo}_{\text{dstIp}} \leq \text{dstIp} \leq \text{hi}_{\text{dstIp}} \land \models \& Policy_1 \land \neg Policy_2\]

Result is a cube:
All-BVSAT: A compact model enumeration

More succinct: Maximize *multiple* bounds

\[ (lo_{1} \leq srcIp \leq hi_{2} \lor lo_{2} \leq srcIp \leq hi_{2} ) \land lo_{srcPort} \leq srcPort \leq hi_{srcPort} \land (lo_{3} \leq dstIp \leq hi_{3} \lor lo_{4} \leq dstIp \leq hi_{4} ) \land Policy_{1} \land \neg Policy_{2} \]

Result is a *multi-cube*:
Long ago in a network far away, the rebel Hassel forces began to claim...
Do old-school tools apply to Network Verification?

- Experiments by Nuno Lopes, March-May 2013
- Queries over Stanford Network + Extensions
- Observation: Datalog is OK to express network queries.
- Query tools tried out.
  - Datalog + Hassel algebra
  - PDR engine
  - Hardware model checkers (ABC, IIMC, IC3)
  - Bounded model checking. Two encodings.
  - Symbolic simulation

Very preliminary finding: PDR ~ Hassel, others work on small (Stanford) network. Hardware tools chocked.
Summary

An Introduction to SMT with Z3

Algorithmic principles of SAT/SMT

Theories, Solvers and Applications