HAMPI
A String Solver
for
Testing, Analysis and Vulnerability Detection

Vijay Ganesh
MIT
July 16, 2011
Software Engineering & SMT Solvers
An Indispensable Tactic
Software Engineering & SMT Solvers
An Indispensable Tactic
Traditional SMT Problem Statement
Efficient Solver for Analysis of Programs

Program

Specification

Program Reasoning Tool

Formulas

SMT Solver

SAT/UNSAT

Program is Correct? or Generate Tests
Traditional SMT Logics

Efficient Solvers for Program Expressions

- Integer/Real Linear Arithmetic
- Bit-vectors
- Arrays
- Uninterpreted Functions
- Abstract Datatypes
- Quantifiers
- Non-linear Arithmetic
- Strings?
Key SMT Concepts

Logician’s Question: What’s New?

• Approximations

• Asymptotically speaking: probably the same

• SAT
  • Clause learning using conflict analysis
  • Backjumping
  • Variable selection heuristics
  • Restarts

• SMT
  • Combinations
  • Under/Over approximations of formulas
  • DPLL(T)
  • Bounding
# Why a String Solver?

Efficient Solver for Analysis of String Programs

<table>
<thead>
<tr>
<th>Common String Operations</th>
<th>String Programs</th>
<th>Types of Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functions</strong></td>
<td><strong>Traditional Apps</strong></td>
<td><strong>Memory-related Errors</strong></td>
</tr>
<tr>
<td>String concatenation</td>
<td>C/C++/Java Apps (String Library)</td>
<td>Buffer overflow</td>
</tr>
<tr>
<td>String extraction</td>
<td>C#/.NET</td>
<td>Code injection</td>
</tr>
<tr>
<td><strong>Predicates</strong></td>
<td><strong>Web Apps</strong></td>
<td><strong>Improper Sanitization</strong></td>
</tr>
<tr>
<td>String comparison</td>
<td>Sanitization code in PHP, JavaScript</td>
<td>SQL injection</td>
</tr>
<tr>
<td>String assignment</td>
<td>Client-side and server-side</td>
<td>XSS scripting</td>
</tr>
<tr>
<td>Sanity checking of strings using RE</td>
<td>Scripting code</td>
<td>Incomplete sanity checking</td>
</tr>
</tbody>
</table>
String Solver Problem Statement
Efficient Solver for Analysis of String Programs

String Program Specification

Program Reasoning Tool

String Formulas

String Solver

SAT/UNSAT

Program is Correct? or Generate Tests
HAMPI String Solver

String Formulas

- $X = \text{concat}("\text{SELECT..."},v) \text{ AND } (X \in \text{SQL\_grammar})$
- JavaScript, PHP, ... string expressions
- NP-complete
- ACM Distinguished Paper Award 2009

SAT

UNSAT
Take Home Message

• Theories of Strings are increasingly key for reliability/security

• Conceptual idea: Bounded logics

• Use HAMPI
Rest of the Talk

• **HAMPI Logic**: A Theory of Strings

• Motivating Example: HAMPI-based Vulnerability Detection App

• How HAMPI works

• Experimental Results

• Related Work: Practice and Theory

• HAMPI 2.0

• SMTization: Future of Strings
# Theory of Strings

## The Hampi Language

<table>
<thead>
<tr>
<th>PHP/JavaScript/C++...</th>
<th>HAMPI: Theory of Strings</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var a; $a = 'name'</td>
<td>Var a : 1...20; a = 'name'</td>
<td>Bounded String Variables</td>
</tr>
<tr>
<td>string_expr.&quot; is &quot;</td>
<td>concat(string_expr,&quot; is &quot;);</td>
<td>Concat Function</td>
</tr>
<tr>
<td>substr(string_expr;1,3)</td>
<td>string_expr[1:3]</td>
<td>Extract Function</td>
</tr>
<tr>
<td>assignments/strcmp</td>
<td>equality</td>
<td>Equality Predicate</td>
</tr>
<tr>
<td>a = string_expr; a /= string_expr;</td>
<td>a = string_expr; a /= string_expr;</td>
<td></td>
</tr>
<tr>
<td>Sanity check in regular expression RE</td>
<td>string_expr in RE</td>
<td>Membership Predicate</td>
</tr>
<tr>
<td>Sanity check in context-free grammar CFG</td>
<td>string_expr in SQL</td>
<td></td>
</tr>
<tr>
<td>string_expr NOT in SQL</td>
<td>string_expr NOT?contains sub_str</td>
<td>Contains Predicate (Substring Predicate)</td>
</tr>
</tbody>
</table>

Substring Predicate
Theory of Strings
The Hampi Language

- \( X = \text{concat}("\text{SELECT msg FROM msgs WHERE topicid = }", v) \)
  \( \quad \text{AND} \quad \)
  \( (X \in \text{SQL\_Grammar}) \)

- input \( \in \text{RegExp}([0-9]+) \)

- \( X = \text{concat} (\text{str\_term1}, \text{str\_term2}, \text{"c"})[1:42] \)
  \( \quad \text{AND} \quad \)
  \( X \text{ contains } \text{"abc"} \)
SELECT m FROM messages WHERE id='1' OR 1 = 1
HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

Web Vulnerabilities by Class
Q1-Q2 2009

Source: IBM Internet Security Systems, 2009
Source: Fatbardh Veseli, Gjovik University College, Norway
if (input in regexp("[0-9]+"))
query := "SELECT m FROM messages WHERE id=‘ ’ + input + ‘ ’")

- **input** passes validation (regular expression check)
- **query** is syntactically-valid SQL
- **query** can potentially contain an attack substring (e.g., 1’ OR ‘1’ = ‘1’)}
Buggy Script

\[
\text{if (input in regexp("[0-9]+"))}
\text{query := \"SELECT m FROM messages WHERE id=' " + input + \" '\")}
\]

- **input** passes validation (regular expression check)
- **query** is syntactically-valid SQL
- **query** can potentially contain an attack substring (e.g., 1' OR '1' = '1')
if (input in regexp("[0-9]+"))
query := "SELECT m FROM messages WHERE id=' " + input + " '")
Rest of the Talk

• HAMPI Logic: A Theory of Strings

• Motivating Example: HAMPI-based Vulnerability Detection App

• How HAMPI works

• Experimental Results

• Related Work: Theory and Practice

• HAMPI 2.0

• SMTization: Future of Strings
Expressing the Problem in HAMPI
SQL Injection Vulnerabilities

Input String

\[ \text{Var v : 12;} \]

SQL Grammar

\[ \text{cfg SqlSmall := "SELECT " [a-z]+ " FROM " [a-z]+ " WHERE " Cond;} \]

\[ \text{cfg Cond := Val "=" Val | Cond " OR " Cond;} \]

\[ \text{cfg Val := [a-z]+ | "" [a-z0-9]* "" | [0-9]+;} \]

SQL Query

\[ \text{val q := concat("SELECT msg FROM messages WHERE topicid=", v, ")}; \]

\[ \text{assert v in [0-9]+;} \]

\[ \text{assert q in SqlSmall;} \]

SQLI attack conditions

\[ \text{assert q contains "OR 1='1";} \]

\[ \text{“q is a valid SQL query”} \]

\[ \text{“q contains an attack vector”} \]
### Hampi Key Conceptual Idea

Bounding, expressiveness and efficiency

<table>
<thead>
<tr>
<th>$L_i$</th>
<th>Complexity of $\emptyset = L_1 \cap \ldots \cap L_n$</th>
<th>Current Solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context-free</td>
<td>Undecidable</td>
<td>n/a</td>
</tr>
<tr>
<td>Regular</td>
<td>PSPACE-complete</td>
<td>Quantified Boolean Logic</td>
</tr>
<tr>
<td>Bounded</td>
<td>NP-complete</td>
<td>SAT Efficient in practice</td>
</tr>
</tbody>
</table>
Hampi Key Idea: Bounded Logics
Testing, Vulnerability Detection,...

- Finding SAT assignment is key
- Short assignments are sufficient

- Bounding is sufficient
- Bounded logics easier to decide
Hampi Key Idea: Bounded Logics
Bounding vs. Completeness

• Bounding leads to incompleteness

• Testing (Bounded MC) vs. Verification (MC)

• Bounding allows trade-off (Scalability vs. Completeness)

• Completeness (also, soundness) as resources
HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

**Input String**

```plaintext
Var v : 12;
```

**SQL Grammar**

- `cfg SqlSmall := "SELECT " [a-z]+ " FROM " [a-z]+ " WHERE " Cond;`
- `cfg Cond := Val "=" Val | Cond " OR " Cond;`
- `cfg Val := [a-z]+ | "'" [a-z0-9]"" | [0-9]+;`

**SQL Query**

```plaintext
val q := concat("SELECT msg FROM messages WHERE topicid='", v, "'");
```

**SQLI attack conditions**

- `assert v in [0-9]+;`
- `assert q in SqlSmall;`
- `assert q contains "OR '1'='1";`

"q is a valid SQL query"

"q contains an attack vector"
How Hampi Works

Bird’s Eye View: Strings into Bit-vectors

Find a 4-char string \( v \):
- \((v)\) is in \( E \)
- \((v)\) contains \((()())\)

\[
\begin{align*}
\text{var } v & : 4; \\
\text{cfg } E := \text{“}()\text{”} | E \ E | \text{“(} \ E \text{“)”}; \\
\text{val } q & := \text{concat(“} , v , \text{”}); \\
\text{assert } q \text{ in } E; \\
\text{assert } q \text{ contains “}()\text{”};
\end{align*}
\]

STP Encoder

STP Decoder

Normalizer

Hampi

STP
Program Expressions → STP Solver

(x = z+2 OR mem[i] + y <= 01)

- Bit-vector or machine arithmetic
- Arrays for memory
- C/C++/Java expressions
- NP-complete

SAT

UNSAT
The History of STP

- 100,000 Constraints
- 2005
- STP
  - Enabled Concolic Testing
  - EXE by Engler et al.
  - BAP/BitBlaze by Song et al.
  - Model checking by Dill et al.

- 1,000,000 Constraints
- 2009
- HAMPI: String Solvers
  - Ardilla by Ernst et al.
  - Kudzu & Kaluza by Song et al.
  - Klee by Engler et al.
  - George Candeas's Cloud 9 tester
  - STP + HAMPI exceed 100+ projects

- Today
- Solver-based languages (Alloy team)
- Solver-based debuggers
- Solver-based type systems
- Solver-based concurrency bugfinding
# Impact of STP: Notable Projects

- Enabled Concolic Testing
- 100+ reliability and security projects

<table>
<thead>
<tr>
<th>Category</th>
<th>Research Project</th>
<th>Project Leader/Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal Methods</td>
<td>ACL2 Theorem Prover + STP Verification-aware Design Checker Java PathFinder Model Checker</td>
<td>Eric Smith &amp; David Dill/Stanford</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jacob Chang &amp; David Dill/Stanford</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mehlitz &amp; Pasareanu/NASA</td>
</tr>
<tr>
<td>Program Analysis</td>
<td>BitBlaze &amp; WebBlaze BAP</td>
<td>Dawn Song et al./Berkeley</td>
</tr>
<tr>
<td></td>
<td></td>
<td>David Brumley/CMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Molnar &amp; Wagner/Berkeley</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Saxena &amp; Song/Berkeley</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bucur &amp; Candea/EPFL</td>
</tr>
<tr>
<td>Hardware Bounded Model-checking (BMC)</td>
<td>Blue-spec BMC BMC</td>
<td>Katelman &amp; Dave/MIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Haimed/NVIDIA</td>
</tr>
</tbody>
</table>
How STP Works

Eager for BV and Lazy for Arrays

Input

Read(A,i₀)=0
Read(A,i₁)=1
...
Read(A,iₙ)=10,000
Θ(i₀,i₁)

Refinement Loop

Substitutions

Simplifications

Linear Solving

Array Abstraction

Conversion to SAT

Boolean SAT Solver

Result
How Hampi Works
Unroll Bounded CFGs into Regular Exp.

var v : 4;
cfg E := "()" | E E | "(" E ");
val q := concat("(" , v , ")");
assert q in E;
assert q contains "()";

Hampi

Normalizer

STP Encoder

STP Decoder

Bound(E,6) -> ([() + (())]) + ()[() + (())] + [(()) + (())]

Bit-vector Constraints

Bit-vector Solution

String Solution
v = )()(
How Hampi Works
Unroll Bounded CFGs into Regular Exp.

```
var v : 4;
cfg E := "()" | E E | "(" E ")";
val q := concat ("(" , v , ")");
assert q in E;
assert q contains "()";
```

```
Bound(E,6) →
()() + (()) + ()[() + (())] + [() + (())]
```

```
v = )()(
```
Find a 4-char string v:
- (v) is in E
- (v) contains ()()

var v : 4;
cfg E := “()” | E E | (“ E “);
val q := concat(“(“, v, “)”);
assert q in E;
assert q contains “()”;

Hampi

STP Encoder

STP Decoder

Normalizer

String Solution
v = )()(

STP

Bit-vector Constraints
Bit-vector Solution
How Hampi Works
Unroll Bounded CFGs into Regular Exp.

Step 1:

```
var v : 4;
cfg E := "()" | E E | "(" E ")";
val q := concat("(" , v , ")");
assert q in E;
assert q contains "()";
```

Step 2:

```
cfg E := "()" | E E | "(" E ")"
```

Auto-derive lower/upper bounds [L,B] on CFG

[6,6]

Look for minimal length string

"()"
How Hampi Works

Unroll Bounded CFGs into Regular Exp.

Step 3:

\[
\text{cfg E := } 
\begin{align*}
& \text{"(" } | \\
& E \ E \ | \\
& \text{"(" E \text{"\" )"}
\end{align*}
\]

Length: 6

Recursively expand non-terminals:

Construct Partitions

Min. length constant: 

Min. length constant: 

[4,2]

[2,4]

[3,3]

[5,1]

[1,5]

[1,4,1]

Step 4:

\[
\text{cfg E := } 
\begin{align*}
& \text{"(" } | \\
& E \ E \ | \\
& \text{"(" E \text{"\" )"}
\end{align*}
\]

Length: 6

Recursively expand non-terminals:

Construct RE

Min. length constant: 

Min. length constant: 

((()))

(()())

(()())

(()())
Unroll Bounded CFGs into Regular Exp.
Managing Exponential Blow-up

- Dynamic programming style
- Works well in practice

cfg $E ::= "()" | E E | "(" E ")"

Length: 6

Recursively expand non-terminals:

Min. length constant: "()"

Construct RE

$(())$
$(())$
$(())$
...

Sunday, July 17, 2011
Unroll Bounded CFGs into Regular Exp. 
Managing Exponential Blow-up

cfg E := “()” | E E | (“ E “)

Length: 6

Recursively expand non-terminals:

Min. length constant: ”()”

Construct RE

Bound(E, 6) →

( [ () + (()) ] ) +
( [ [ () + (()) ] ] +
[ () + (()) ]() )

((())
(()()
((())))
...
Encode regular expressions recursively

- Alphabet \( \{ (, ) \} \rightarrow 0, 1 \)
- constant \( \rightarrow \) bit-vector constant
- union \( + \rightarrow \) disjunction \( \lor \)
- concatenation \( \rightarrow \) conjunction \( \land \)
- Kleene star \( \ast \rightarrow \) conjunction \( \land \)
- Membership, equality \( \rightarrow \) equality

\[
( \lor ) \in ( ( ) ( ) + ( ( ) ) ) + [ ( ( ) ( ) + ( ( ) ) ) ( ) + ( ( ( ) ( ) + ( ( ) ) ) ) ]
\]

Formula \( \Phi_1 \lor \) Formula \( \Phi_2 \lor \) Formula \( \Phi_3 \)

\[
\]
How Hampi Works

Converting Regular Exp. into Bit-vectors

\[(v) \in ([]) [([() + (())]) + ([(()) + (())]) + (()) + (())]]\]

Formula $\Phi_1 \lor \text{Formula } \Phi_2 \lor \text{Formula } \Phi_3$

\[
\]

- Constraint Templates
- Encode once, and reuse
- On-demand formula generation
How Hampi Works
Decoder converts Bit-vectors to Strings

Find a 4-char string \( v \):
- \( (v) \) is in \( E \)
- \( (v) \) contains \( ()() \)

```
var v : 4;
cfg E := "()" | E E | "(" E ");
val q := concat("(" , v , ")");
assert q in E;
assert q contains "()";
```

```
String Solution
v = )()(
```
Rest of the Talk

- HAMPI Logic: A Theory of Strings
- Motivating Example: HAMPI-based Vulnerability Detection App
- How HAMPI works
- Experimental Results
- Related Work: Theory and Practice
- HAMPI 2.0
- SMTization: Future of Strings
HAMPI: Result 1

Static SQL Injection Analysis

- 1367 string constraints from Wasserman & Su [PLDI’07]
- Hampi scales to large grammars
- Hampi solved 99.7% of constraints in < 1 sec
- All solvable constraints had short solutions
• Attackers inject client-side script into web pages

• Somehow circumvent same-origin policy in websites

• `echo “Thank you $my_poster for using the message board”;`

• Unsanitized `$my_poster`

• Can be JavaScript

• Execution can be bad
HAMPI: Result 2
Security Testing

• Hampi used to build Ardilla security tester [Kiezun et al., ICSE’09]

• 60 new vulnerabilities on 5 PHP applications (300+ kLOC)
  • 23 SQL injection
  • 37 cross-site scripting (XSS)

• 46% of constraints solved in < 1 second per constraint

• 100% of constraints solved in <10 seconds per constraint

5 added to US National Vulnerability DB
HAMPI: Result 3
Comparison with Competing Tools

- HAMPI vs. CFGAnalyzer (U. Munich): HAMPI ~7x faster for strings of size 50+
Comparison with Competing Tools

**RE intersection problems**

- HAMPI 100x faster than Rex (MSR)

- HAMPI 1000x faster than DPRLE (U. Virginia)

- Pieter Hooimeijer 2010 paper titled ‘Solving String Constraints Lazily’
HAMPI: Result 4
Helping KLEE Pierce Parsers

Generate Input Using HAMPI; Mark Partially Symbolic

Semantic Core

Parser

Symbolic Execution Engine with Implicit Spec

KLEE

Formulas

STP

SAT/UNSAT

Crashing Tests
How to Automatically Crash Programs?

KLEE: Concolic Execution-based Tester

Problem: Automatically generate crashing tests given only the code

Program

Symbolic Execution Engine with Implicit Spec

Formulas

STP

SAT/UNSAT

Crashing Tests

Automatic Tester
How to Automatically Crash Programs?

KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player,...

Buggy_C_Program(int* data_field, int len_field) {
    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized
    while (i++ < process(len_field)) {
        //1. Integer overflow causing NULL deref
        //2. Buffer overflow
        *(ptr+i) = process_data(*(data_field+i));
    }
}

• Formula captures computation
• Tester attaches formula to capture spec
How to Automatically Crash Programs?
KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player,...

Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
        //1. Integer overflow causing NULL deref
        //2. Buffer overflow
        *(ptr+i) = process_data(*(data_field+i));
    }
}

Equivalent Logic Formula derived using symbolic execution

data_field, mem_ptr : ARRAY;
len_field : BITVECTOR(32); //symbolic
i, j, ptr : BITVECTOR(32); //symbolic

  .
  .
mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);

  .
  .

• Formula captures computation
• Tester attaches formula to capture spec
How to Automatically Crash Programs?

KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player, ...

Buggy_C_Program(int* data_field, int len_field) {

    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
        // 1. Integer overflow causing NULL deref
        // 2. Buffer overflow
        *(ptr+i) = process_data(*(data_field+i));
    }
}

Equivalent Logic Formula derived using symbolic execution

data_field, mem_ptr : ARRAY;
len_field : BITVECTOR(32); //symbolic
i, j, ptr : BITVECTOR(32); //symbolic

    mem_ptr[ptr+i] = process_data(data_field[i]);
    mem_ptr[ptr+i+1] = process_data(data_field[i+1]);

• Formula captures computation
• Tester attaches formula to capture spec
How to Automatically Crash Programs?
KLEE: Concolic Execution-based Tester

Structured input processing code:
PDF Reader, Movie Player,...

Buggy_C_Program(int* data_field, int len_field) {
    int * ptr = malloc(len_field*sizeof(int));
    int i; //uninitialized

    while (i++ < process(len_field)) {
        //1. Integer overflow causing NULL deref
        //2. Buffer overflow
        *(ptr+i) = process_data(*(data_field+i));
    }
}

Equivalent Logic Formula derived using symbolic execution

data_field, mem_ptr : ARRAY;
len_field : BITVECTOR(32); //symbolic
i, j, ptr : BITVECTOR(32); //symbolic

mem_ptr[ptr+i] = process_data(data_field[i]);
mem_ptr[ptr+i+1] = process_data(data_field[i+1]);

//INTEGER OVERFLOW QUERY
0 <= j <= process(len_field);
ptr + i + j = 0?

• Formula captures computation
• Tester attaches formula to capture spec
HAMPI: Result 4
Helping KLEE Pierce Parsers

KLEE

Parser
Semantic Core

Symbolic Execution Engine with Implicit Spec

Mark Input Symbolic

Compressed Tests

Formulas

STP

SAT/UNSAT
HAMPI: Result 4

Helping KLEE Pierce Parsers

Generate Input Using HAMPI; Mark Partially Symbolic

Parser

Semantic Core

Symbolic Execution Engine with Implicit Spec

Formulas

STP

SAT/UNSAT

Crashing Tests

KLEE
HAMPI: Result 4
Helping KLEE Pierce Parsers

• Klee provides API to place constraints on symbolic inputs

• Manually writing constraints is hard

• Specify grammar using HAMPI, compile to C code

• Particularly useful for programs with highly-structured inputs

• 2-5X improvement in line coverage
## Impact of Hampi: Notable Projects

<table>
<thead>
<tr>
<th>Category</th>
<th>Research Project</th>
<th>Project Leader/Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Analysis</td>
<td>SQL-injection vulnerabilities</td>
<td>Wasserman &amp; Su/UC, Davis</td>
</tr>
<tr>
<td>Security Testing</td>
<td>Ardilla for PHP (SQL injections, cross-site scripting)</td>
<td>Kiezun &amp; Ernst/MIT</td>
</tr>
<tr>
<td>Concolic Testing</td>
<td>Klee</td>
<td>Engler &amp; Cadar/Stanford</td>
</tr>
<tr>
<td></td>
<td>Kudzu</td>
<td>Saxena &amp; Song/Berkeley</td>
</tr>
<tr>
<td></td>
<td>NoTamper</td>
<td>Bisht &amp; Venkatakrishnan/U Chicago</td>
</tr>
<tr>
<td>New Solvers</td>
<td>Kaluza</td>
<td>Saxena &amp; Song/Berkeley</td>
</tr>
</tbody>
</table>
# Impact of Hampi: Notable Projects

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Description</th>
<th>Project Leader/ Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kudzu</td>
<td>JavaScript Bug Finder &amp; Vulnerability Detector</td>
<td>Saxena, Akhawe, Hanna, Mao, McCamant, Song/Berkeley</td>
</tr>
<tr>
<td>NoTamper</td>
<td>Parameter Tamper Detection</td>
<td>Bisht, Hinrichs/U of Chicago, Skrupsky, Bobrowicz, Vekatakrishnan/U. of Illinois, Chicago</td>
</tr>
</tbody>
</table>
**Impact of Hampi: Notable Projects**

**NoTamper**

- Client-side checks (C), no server checks
- Find solutions $S_1, S_2, \ldots$ to C, and solutions $E_1, E_2, \ldots$ to $\neg C$ by calling HAMPI
- $E_1, E_2, \ldots$ are candidate exploits
- Submit $(S_1, E_1), \ldots$ to server
- If server response same, ignore
- If server response differ, report error
## Related Work (Practice)

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Project Leader/Institution</th>
<th>Comparison with HAMPI</th>
</tr>
</thead>
</table>
| Rex       | Bjorner, Tillman, Vornkov et al. (Microsoft Research, Redmond) | • HAMPI  
+ Length+Replace($s_1,s_2,s_3$)  
- CFG  
• Translation to int. linear arith. (Z3) |
| Mona      | Karlund et al. (U. of Aarhus) | • Can encode HAMPI & Rex  
• User work  
• Automata-based  
• Non-elementary |
| DPRLE     | Hooimeijer (U. of Virginia) | • Regular expression constraints |
## Related Work (Theory)

<table>
<thead>
<tr>
<th>Result</th>
<th>Person (Year)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undecidability of Quantified Word Equations</td>
<td>Quine (1946)</td>
<td>Multiplication reduced to concat</td>
</tr>
<tr>
<td>Decidability (PSPACE-complete) of QF Theory of Word Equations + RE</td>
<td>Schultz (1992)</td>
<td>RE membership predicate</td>
</tr>
<tr>
<td>QF word equations + Length() (?)</td>
<td>Matiyasevich (1971)</td>
<td>Unsolved Reduction to Diophantine</td>
</tr>
<tr>
<td>QF word equations in solved form + Length() + RE</td>
<td>G. (2011)</td>
<td>Practical</td>
</tr>
</tbody>
</table>
Future of HAMPI & STP

• HAMPI will be combined with STP
  • Bit-vectors and Arrays
  • Integer/Real Linear Arithmetic
  • Uninterpreted Functions
  • Strings
  • Floating Point
  • Non-linear

• Additional features planned in STP
  • UNSAT Core
  • Quantifiers
  • Incremental
  • DPLL(T)
  • Parallel STP
  • MAXSMT?

• Extensibility and hackability by non-expert
Future of Strings

• **Strings SMTization effort started**
  • Nikolaj Bjorner, G.
  • Andrei Voronkov, Ruzica Piskac, Ting Zhang
  • Cesare Tinelli, Clark Barrett, Dawn Song, Prateek Saxena, Pieter Hooimeijer, Tim Hinrichs

• **SMT Theory of Strings**
  • Alphabet (UTF, Unicode,...)
  • String Constants and String Vars (parameterized by length)
  • Concat, Extract, Replace, Length Functions
  • Regular Expressions, CFGs (Extended BNF)
  • Equality, Membership Predicate, Contains Predicate

• **Applications**
  • Static/Dynamic Analysis for Vulnerability Detection
  • Security Testing using Concolic Idea
  • Formal Methods
  • Synthesis
Conclusions & Take Away

• String solvers essential for testing, analysis, vulnerability detection
  • String applications in C/C++/Java/C#
  • Web applications in PHP/JavaScript (client and server-side)

• HAMPI
  • Multiple string vars, constants
  • Concat/extract function
  • Equality between string terms
  • Membership predicate over RE/CFGs
  • Contains predicate

• Demand for even richer theories
  • Attribute grammars
  • String theories with length

• Bounding: powerful and versatile idea (BMC, bounded logics,...)

• Using completeness as a resource (can we be more systematic?)
Topics Covered Today

• HAMPI Logic: A Theory of Strings

• HAMPI-based vulnerability detection app

• How HAMPI works

• Another HAMPI-based app: Tamper Detection

• Experimental results (Static, Dynamic, Competing tools, KLEE)

• Related work (Kaluza, Rex,...)

• Future of strings & SMTization
# Key Contributions

http://people.csail.mit.edu/vganesh

<table>
<thead>
<tr>
<th>Name</th>
<th>Key Concept</th>
<th>Impact</th>
<th>Pubs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit-vector &amp; Array Solver(^1,^2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HAMPI</strong></td>
<td>App-driven Bounding for Solving</td>
<td>Analysis of Web Apps</td>
<td>ISSTA 2009(^3), TOSEM 2011 (CAV 2011)</td>
</tr>
<tr>
<td>String Solver(^1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Taint-based Fuzzing</strong></td>
<td>Information flow is cheaper than concolic</td>
<td>Scales better than concolic</td>
<td>ICSE 2009</td>
</tr>
<tr>
<td><strong>Automatic Input Rectification</strong></td>
<td>Acceptability Envelope: Fix the input, not the program</td>
<td>New way of approaching SE</td>
<td>Under Submission</td>
</tr>
</tbody>
</table>

1. 100+ research projects use STP and HAMPI
2. STP won the SMTCOMP 2006 and 2010 competitions for bit-vector solvers
3. HAMPI: ACM Best Paper Award 2009
4. Retargetable Compiler (DATE 1999)
5. Proof-producing decision procedures (TACAS 2003)
6. Error-finding in ARBAC policies (CCS 2011)