CS 6V81-05
Program Analysis I: Dynamic Detection of Likely Invariants and Robust Signatures for Kernel Data Structure

Junyuan Zeng

Department of Computer Science
The University of Texas at Dallas

November 4th, 2011
Outline

1. Dynamic Invariant Detection
   - Concepts
   - Daikon

2. Robust Signatures for Kernel Data Structures
   - Background
   - Robust Signatures
1. Dynamic Invariant Detection
   - Concepts
   - Daikon

2. Robust Signatures for Kernel Data Structures
   - Background
   - Robust Signatures
Outline

1. Dynamic Invariant Detection
   • Concepts
     • Daikon

2. Robust Signatures for Kernel Data Structures
   • Background
     • Robust Signatures
what is program invariant?

A program invariant is a property that holds at a certain point or points in a program being constant \( x = a \), non-zero \( x \neq 0 \), ordering \( x \leq y \), being in a range \( a \leq x \leq b \), containment \( x \in y \), linear relationships \( y = ax + b \), functions from a library \( x = fn(y) \), and often used in assert statements, documentation, and formal specifications. Invariant detection is also useful in program understanding and a host of other applications.
what is program invariant?

- refer to a **property** that holds at a certain point or points in a program
what is program invariant?

- refer to a **property** that holds at a certain point or points in a program
  - being constant \( x = a \), non-zero \( x \neq 0 \), ordering \( x \leq y \), being in a range \( a \leq x \leq b \), containment \( x \in y \)
what is program invariant?

- refer to a **property** that holds at a certain point or points in a program
  - being constant \( x = a \), non-zero \( x \neq 0 \), ordering \( x \leq y \), being in a range \( a \leq x \leq b \), containment \( x \in y \)
  - linear relationships \( y = ax + b \), functions from a library \( x = fn(y) \)
what is program invariant?

- refer to a **property** that holds at a certain point or points in a program
  - being constant $x = a$, non-zero $x \neq 0$, ordering $x \leq y$, being in a range $a \leq x \leq b$, containment $x \in y$
  - linear relationships $y = ax + b$, functions from a library $x = fn(y)$
  - ...

- often used in assert statements, documentation and formal specifications
What is program invariant?

- refer to a **property** that holds at a certain point or points in a program
  - being constant $x = a$, non-zero $x \neq 0$, ordering $x \leq y$, being in a range $a \leq x \leq b$, containment $x \in y$
  - linear relationships $y = ax + b$, functions from a library $x = fn(y)$
  - ...

- often used in assert statements, documentation and formal specifications
- also useful in program understanding and a host of other applications
\begin{verbatim}
\begin{verbatim}
i, s := 0, 0;
do i \neq n \rightarrow 
i, s := i + 1, s + b[i]
\end{verbatim}
\end{verbatim}

Precondition: \( n \geq 0 \)
Postcondition: \( s = (\sum j : 0 \leq j < n : b[j]) \)
Loop invariant: \( 0 \leq i \leq n \) and \( s = (\sum j : 0 \leq j < i : b[j]) \)
dynamic invariant detection
Dynamic invariant detection

- expecting programmers to fully annotate code with invariants but programmers do not usually explicitly annotate or document code with invariants
expecting programmers to fully annotate code with invariants but programmers do not usually explicitly annotate or document code with invariants

dynamic invariant detection
Dynamic Invariant Detection

Dynamic Invariant Detection

- expecting programmers to fully annotate code with invariants but programmers do not usually explicitly annotate or document code with invariants

- **dynamic invariant detection**
  - runs a program
Dynamic Invariant Detection

Robust Signatures for Kernel Data Structures

dynamic invariant detection

- expecting programmers to fully annotate code with invariants but programmers do not usually explicitly annotate or document code with invariants

- dynamic invariant detection
  - runs a program
  - observes the values that the program computes
**Dynamic Invariant Detection**

Dynamic invariant detection

- expecting programmers to fully annotate code with invariants but programmers do not usually explicitly annotate or document code with invariants

**Dynamic invariant detection**

- runs a program
- observes the values that the program computes
- reports properties that were true over the observe executions
Dynamic Invariant Detection

Robust Signatures for Kernel Data Structures

---

dynamic invariant detection

- expecting programmers to fully annotate code with invariants but programmers do not usually explicitly annotate or document code with invariants

- **dynamic invariant detection**
  - runs a program
  - observes the values that the program computes
  - reports properties that were true over the observe executions

- Daikon proposes to automatically determine program invariants and report them in a meaningful manner
1 Dynamic Invariant Detection
   - Concepts
   - Daikon

2 Robust Signatures for Kernel Data Structures
   - Background
   - Robust Signatures
Daikon

- an implementation of dynamic detection of likely invariants
Daikon

- an implementation of dynamic detection of likely invariants
- uses machine learning techniques
Daikon

- an implementation of dynamic detection of likely invariants
- uses machine learning techniques
- can detect invariants in C, C++, Java and Perl programs
Daikon

- an implementation of dynamic detection of likely invariants
- uses machine learning techniques
- can detect invariants in C, C++, Java and Perl programs
- input is the program and output is a set of likely program invariants
Dynamic Invariant Detection

Robust Signatures for Kernel Data Structures

architecture

1. **Original program**
2. **Instrument**
3. **Run**
4. **Data trace database**
5. **Detect invariants**
6. **Invariants**
7. **Test suite**
architecture: original program

```
i, s := 0, 0;
do i != n ->
i, s := i + 1, s + b[i]
od
```
Dynamic Invariant Detection

architecture: instrumenters

- Original program
- Instrument
- Instrumented program
- Run
- Data trace database
- Detect invariants
- Test suite
- Invariants
architecture: instrumenters

- add instructions to the target program so that it will output the values of variables
architecture: instrumenters

- add instructions to the target program so that it will output the values of variables
- language dependent (C/C++, Java and Perl etc.)
architecture: instrumented program

```plaintext
print b, n;
i, s := 0, 0;
do i != n ->
print i, s, n, b[i];
i, s := i + 1, s + b[i]
od
```
architecture: trace file

```plaintext
ENTER
B = 92 56 -96 -49 76 92 -3 -88, modified
N = 8, modified

LOOP
B = 92 56 -96 -49 76 92 -3 -88, unmodified
N = 8, modified
I = 0, modified
s = 0, modified

LOOP
B = 92 56 -96 -49 76 92 -3 -88, unmodified
N = 8, unmodified
I = 1, modified
s = 92, modified
....
```
Dynamic Invariant Detection

architecture: inference engine

Original program → Instrument → Run → Detect invariants → Data trace database → Invariants

- read the trace data and produces likely invariants
- adopt optimizations to reduce redundant invariant checking
- equal variables: if $x = y$, then for any invariant $f$, $f(x) \Rightarrow f(y)$
- dynamical constant variables: $x = 5 \Rightarrow \text{odd}(x)$ and $x \geq 5$
- variable hierarchy: for two program points A and B, if all samples for B also appear at A = $\Rightarrow$ (invariants true at A $\Rightarrow$ true at B)
- suppression of weaker invariants: $x > y \Rightarrow x \geq y$
### Architecture: Inference Engine

- **Original program** → **Instrument** → **Run** → **Detect invariants** → **Invariants**

- **Test suite**

- **Read the trace data and produces likely invariants**
**Dynamic Invariant Detection**

**Robust Signatures for Kernel Data Structures**

**architecture: inference engine**

![Diagram of inference engine process]

- Read the trace data and produces likely invariants
- Adopt optimizations to reduce redundant invariant checking
architecture: inference engine

- read the trace data and produces likely invariants
- adopt optimizations to reduce redundant invariant checking
  - equal variables: if $x = y$, then for any invariant $f$, $f(x) \implies f(y)$
architectural: inference engine

- read the trace data and produces likely invariants
- adopt optimizations to reduce redundant invariant checking
  - equal variables: if \( x = y \), then for any invariant \( f \), \( f(x) \implies f(y) \)
  - dynamically constant variables: \( x = 5 \implies \text{odd}(x) \) and \( x \geq 5 \)
read the trace data and produces likely invariants

adopt optimizations to reduce redundant invariant checking

- equal variables: if $x = y$, then for any invariant $f$, $f(x) \implies f(y)$
- dynamically constant variables: $x = 5 \implies odd(x)$ and $x \geq 5$
- variable hierarchy: for two program points A and B, if all samples for B also appear at A $\implies$ (invariants true at A $\implies$ true at B)
architecture: inference engine

- read the trace data and produces likely invariants
- adopt optimizations to reduce redundant invariant checking
  - equal variables: if $x = y$, then for any invariant $f$, $f(x) \implies f(y)$
  - dynamically constant variables: $x = 5 \implies \text{odd}(x)$ and $x \geq 5$
  - variable hierarchy: for two program points A and B, if all samples for B also appear at A $\implies$ (invariants true at A $\implies$ true at B)
  - suppression of weaker invariants: $x > y \implies x \geq y$
architecture: invariants

1) n >= 0
2) s = SUM(B)
3) i >= 0
Java class **StackAr**

**fields:**

```java
Object[] theArray;  // Array that contains the stack elements.
int topOfStack;    // Index of top element. -1 if stack is empty.
```

**methods:**

```java
void push(Object x)   // Insert x
void pop()           // Remove most recently inserted item
Object top()         // Return most recently inserted item
Object topAndPop()   // Remove and return most recently inserted item
boolean isEmpty()    // Return true if empty; else false
boolean isFull()     // Return true if full; else false
void makeEmpty()     // Remove all items
```
Object invariants for StackAr

this.theArray != null
this.theArray.getClass() == java.lang.Object[].class
this.topOfStack >= -1
this.topOfStack <= this.theArray.length - 1
this.theArray[0..this.topOfStack] elements != null
this.theArray[this.topOfStack+1..] elements == null
output (2)

pre-conditions for the StackAr constructor

capacity >= 0

post-conditions for the StackAr constructor

orig(capacity) == this.theArray.length
this.topOfStack == -1
this.theArray[] elements == null
post-conditions for the isFull method

\[
\begin{align*}
\text{this.theArray} & \equiv \text{orig(this.theArray)} \\
\text{this.theArray[]} & \equiv \text{orig(this.theArray[])} \\
\text{this.topOfStack} & \equiv \text{orig(this.topOfStack)} \\
\text{return} \equiv \text{false} & \iff (\text{this.topOfStack}<\text{this.theArray}.\text{length} - 1) \\
\text{return} \equiv \text{true} & \iff (\text{this.topOfStack}==\text{this.theArray}.\text{length} - 1)
\end{align*}
\]
other tools

- Agitator product by Agitar
other tools

- Agitator product by Agitar
- DIDUCE tool for Java programs
other tools

- Agitator product by Agitar
- DIDUCE tool for Java programs
- IODINE tool for hardware designs
other tools

- Agitator product by Agitar
- DIDUCE tool for Java programs
- IODINE tool for hardware designs
- SPIN model checker which checks if two variables are related by $=, <, >, \leq$ or $\geq$; the output is a graph with variables at the nodes and edges labeled by the comparison relations
other tools

- Agitator product by Agitar
- DIDUCE tool for Java programs
- IODINE tool for hardware designs
- SPIN model checker which checks if two variables are related by $=,$ $<,$ $>,$ $\leq$ or $\geq$; the output is a graph with variables at the nodes and edges labeled by the comparison relations
- ...
summary

a full-featured, scalable, robust tool for dynamic invariant detection supports multiple programming languages likely invariants that it produces have been used from refactoring to testing to data structure repair etc.
summary

- a full-featured, scalable, robust tool for dynamic invariant detection
Summary

- A full-featured, scalable, robust tool for dynamic invariant detection
- Supports multiple programming languages
a full-featured, scalable, robust tool for dynamic invariant detection
supports multiple programming languages
likely invariants that it produces have been used from refactoring to testing to data structure repair etc.
1. Dynamic Invariant Detection
   - Concepts
   - Daikon

2. Robust Signatures for Kernel Data Structures
   - Background
   - Robust Signatures
Outline

1. Dynamic Invariant Detection
   - Concepts
   - Daikon

2. Robust Signatures for Kernel Data Structures
   - Background
   - Robust Signatures
Dynamic Invariant Detection

Robust Signatures for Kernel Data Structures

attacks

objective: conceal presence of an object from the user

hooking: modify code, make the APIs lie

DKOM: manipulate kernel data structures instead of code
attacks

- objective: conceal presence of an object from the user
attacks

- objective: conceal presence of an object from the user
- hooking: modify code, make the APIs lie
Attacks

- Objective: conceal presence of an object from the user
  - Hooking: modify code, make the APIs lie
  - DKOM: manipulate kernel data structures instead of code
signature scans
signature scans

- some new tools (PTFinder, Volatility) use signature scans to find hidden objects in memory
signature scans

- some new tools (PTFinder, Volatility) use signature scans
to find hidden objects in memory
  - identify a set of invariants (signatures) for the object
some new tools (PTFinder, Volatility) use signature scans to find hidden objects in memory
- identify a set of invariants (signatures) for the object
- perform a linear scan through memory
some new tools (PTFinder, Volatility) use signature scans to find hidden objects in memory

- identify a set of invariants (signatures) for the object
- perform a linear scan through memory
- report regions of data where the fields match your invariants (signatures)
signature scans

naive signature for EPROCESS

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>0x03</td>
</tr>
<tr>
<td>Size</td>
<td>0x1b</td>
</tr>
<tr>
<td>ThreadListHead</td>
<td>&gt;= 0x80000000</td>
</tr>
<tr>
<td>DirectoryTableBase is aligned to</td>
<td>0x20</td>
</tr>
</tbody>
</table>
signature evasion
signature evasion

- assume attackers can read and modify any kernel data (like DKOM)
signature evasion

- assume attackers can read and modify any kernel data (like DKOM)
- attackers may modify some fields to hide objects from scanners without OS crashing
signature evasion

- assume attackers can read and modify any kernel data (like DKOM)
- attackers may modify some fields to hide objects from scanners without OS crashing
- false negative for a signature
1 Dynamic Invariant Detection
   - Concepts
   - Daikon

2 Robust Signatures for Kernel Data Structures
   - Background
   - Robust Signatures
architecture

- develop robust signatures that resist signature evasion
architecture

- develop robust signatures that resist signature evasion
  - feature selection:
develop robust signatures that resist signature evasion

- feature selection:
  - profile the data structure to determine which fields are most commonly accessed by the operating system
architecture

- develop robust signatures that resist signature evasion
  - feature selection:
    - profile the data structure to determine which fields are most commonly accessed by the operating system
    - fuzz the most commonly accessed fields to determine which can be modified without OS crashing
architecture

- develop robust signatures that resist signature evasion
  - feature selection:
    - profile the data structure to determine which fields are most commonly accessed by the operating system
    - fuzz the most commonly accessed fields to determine which can be modified without OS crashing
  - signature generation based on the selected fields
data structure profiling

- Idea: Watch the structure as OS executes. If a field is never accessed, an attacker can control it.
- Monitor OS execution and build a histogram of how frequently each field is accessed.

Diagram:
- Guest OS (WinXP) connected to Kernel, which then connects to Structure, which connects to Structure Access Profile.
- Xen Hypervisor connected to the Diagram.
data structure profiling

idea: watch the structure as OS executes

- Guest OS (WinXP)
- Kernel
- Structure
- Xen Hypervisor
- Structure Access Profile

if a field is never accessed, attacker can control it
monitor OS execution and build histogram of how frequently each field is accessed
data structure profiling

- Idea: watch the structure as OS executes
- If a field is never accessed, attacker can control it
**Dynamic Invariant Detection**

**Robust Signatures for Kernel Data Structures**

**Data structure profiling**

- Idea: watch the structure as OS executes.
- If a field is never accessed, attacker can control it.
- Monitor OS execution and build histogram of how frequently each field is accessed.
EPROCESS profile

Always

Never

Field
profiling is not enough

don't know if functionality depends on a given member
being accessed is a necessary condition, but not sufficient
ensure that the field cannot be arbitrarily modified
profiling is not enough

- don’t know if functionality depends on a given member
profiling is not enough

- don’t know if functionality depends on a given member
- being accessed is a necessary condition, but not sufficient
profiling is not enough

- don’t know if functionality depends on a given member
- being accessed is a necessary condition, but not sufficient
- ensure that the field cannot be arbitrarily modified
fuzzing
modify data fields and see if OS crashes
fuzzing

- modify data fields and see if OS crashes
- if OS consistently crashes, attacker probably can’t modify that field
fuzzing

- modify data fields and see if OS crashes
- if OS consistently crashes, attacker probably can’t modify that field
- during fuzz testing, failures are **indications** that a field is robust
fuzzing methodology

- Run OS + Test Program
- Pause VM Fuzz Data Resume VM
- Field Confidence
- Field Confidence
- Restore Base Snapshot
how to fuzz

- for each field, replace its contents with test data from one of several classes:

```
<table>
<thead>
<tr>
<th>Zero</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000</td>
<td>24601 7669</td>
</tr>
<tr>
<td></td>
<td>9272 15151</td>
</tr>
<tr>
<td></td>
<td>2 159</td>
</tr>
<tr>
<td>Kernel</td>
<td>Random Aggregate</td>
</tr>
</tbody>
</table>

Random Primitive | Random Aggregate
```
### Original Signatures

<table>
<thead>
<tr>
<th>Field</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pcb.Header.Type</td>
<td>0x03</td>
</tr>
<tr>
<td>Pcb.Header.Size</td>
<td>0x1B</td>
</tr>
<tr>
<td>ThreadListHead.Flink</td>
<td>Kernel</td>
</tr>
<tr>
<td>ThreadListHead.Blink</td>
<td>Kernel</td>
</tr>
<tr>
<td>Pcb.DirectoryTableBase[0]</td>
<td>Aligned</td>
</tr>
<tr>
<td>WorkingSetLock.Event.Header.Type</td>
<td>0x01</td>
</tr>
<tr>
<td>WorkingSetLock.Event.Header.Size</td>
<td>0x04</td>
</tr>
<tr>
<td>AddressCreationLock.Event.Header.Type</td>
<td>0x01</td>
</tr>
<tr>
<td>AddressCreationLock.Event.Header.Size</td>
<td>0x04</td>
</tr>
</tbody>
</table>
so far...

<table>
<thead>
<tr>
<th>Field</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Pcb.Header.Type</code></td>
<td>0x03</td>
</tr>
<tr>
<td><code>Pcb.Header.Size</code></td>
<td>0x1B</td>
</tr>
<tr>
<td><code>ThreadListHead.Flink</code></td>
<td>Kernel</td>
</tr>
<tr>
<td><code>ThreadListHead.Blink</code></td>
<td>Kernel</td>
</tr>
<tr>
<td><code>Pcb.DirectoryTableBase[0]</code></td>
<td>Aligned</td>
</tr>
<tr>
<td><code>WorkingSetLock.Event.Header.Type</code></td>
<td>0x01</td>
</tr>
<tr>
<td><code>WorkingSetLock.Event.Header.Size</code></td>
<td>0x04</td>
</tr>
<tr>
<td><code>AddressCreationLock.Event.Header.Type</code></td>
<td>0x01</td>
</tr>
<tr>
<td><code>AddressCreationLock.Event.Header.Size</code></td>
<td>0x04</td>
</tr>
</tbody>
</table>
### after fuzz...

<table>
<thead>
<tr>
<th>Field</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pcb.Header.Type</td>
<td>0x03</td>
</tr>
<tr>
<td>Pcb.Header.Size</td>
<td>0x1B</td>
</tr>
<tr>
<td>ThreadListHead.Flink</td>
<td>Kernel</td>
</tr>
<tr>
<td>ThreadListHead.Blink</td>
<td>Kernel</td>
</tr>
<tr>
<td>Pcb.DirectoryTableBase[0]</td>
<td>Aligned</td>
</tr>
<tr>
<td>WorkingSetLock.Event.Header.Type</td>
<td>0x01</td>
</tr>
<tr>
<td>WorkingSetLock.Event.Header.Size</td>
<td>0x04</td>
</tr>
<tr>
<td>AddressCreationLock.Event.Header.Type</td>
<td>0x01</td>
</tr>
<tr>
<td>AddressCreationLock.Event.Header.Size</td>
<td>0x04</td>
</tr>
</tbody>
</table>
signature generation

- dynamic invariant detection
signature generation

- dynamic invariant detection
- infer constraints (invariants) on robust fields
signature generation

- dynamic invariant detection
- infer constraints (invariants) on robust fields
- input: for each robust field, extract a list of the values for all processes found in the memory images (uninfected)
signature generation

- dynamic invariant detection
- infer constraints (invariants) on robust fields
- input: for each robust field, extract a list of the values for all processes found in the memory images (uninfected)
- output: constrains (invariants)
## a list of constrains

<table>
<thead>
<tr>
<th>Field</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pcb.ReadyListHead.Flink</td>
<td><code>val &amp; 0x80000000 == 0x80000000 &amp;&amp; val % 0x8 == 0</code></td>
</tr>
<tr>
<td>Pcb.ThreadListHead.Flink</td>
<td><code>val &amp; 0x80000000 == 0x80000000 &amp;&amp; val % 0x8 == 0</code></td>
</tr>
<tr>
<td>WorkingSetLock.Count</td>
<td><code>val == 1 &amp;&amp; val &amp; 0x1 == 0x1</code></td>
</tr>
<tr>
<td>Vm.VmWorkingSetList</td>
<td><code>val &amp; 0xc0003000 == 0xc0003000 &amp;&amp; val % 0x1000 == 0</code></td>
</tr>
<tr>
<td>VadRoot</td>
<td>`val == 0</td>
</tr>
<tr>
<td>Token.Value</td>
<td><code>val &amp; 0xe0000000 == 0xe0000000</code></td>
</tr>
<tr>
<td>AddressCreationLock.Count</td>
<td><code>val == 1 &amp;&amp; val &amp; 0x1 == 0x1</code></td>
</tr>
<tr>
<td>VadHint</td>
<td>`val == 0</td>
</tr>
<tr>
<td>Token.Object</td>
<td><code>val &amp; 0xe0000000 == 0xe0000000</code></td>
</tr>
<tr>
<td>QuotaBlock</td>
<td><code>val &amp; 0x80000000 == 0x80000000 &amp;&amp; val % 0x8 == 0</code></td>
</tr>
<tr>
<td>ObjectTable</td>
<td>`val == 0</td>
</tr>
<tr>
<td>GrantedAccess</td>
<td><code>val &amp; 0x1f07fb == 0x1f07fb</code></td>
</tr>
<tr>
<td>ActiveProcessLinks.Flink</td>
<td><code>val &amp; 0x80000000 == 0x80000000 &amp;&amp; val % 0x8 == 0</code></td>
</tr>
<tr>
<td>Pcb</td>
<td>`val == 0</td>
</tr>
<tr>
<td>Pcb.DirectoryTableBase.0</td>
<td><code>val &amp; 0x20 == 0</code></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Signature</th>
<th>Image Type</th>
<th>FP</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Existing Signatures</strong></td>
<td>Clean</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Malicious</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Robust Signature</strong></td>
<td>Clean</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Malicious</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
conclusions

- existing signatures are vulnerable to evasion
conclusions

- existing signatures are vulnerable to evasion
- new systematic method for selecting features for data structure signatures
conclusions

- existing signatures are vulnerable to evasion
- new systematic method for selecting features for data structure signatures
- constrains attackers’ ability to evade signatures
references

