Labeling Library Functions in Stripped Binaries

Emily R. Jacobson, Nathan Rosenblum, and Barton P. Miller
Computer Sciences Department
University of Wisconsin - Madison

PASTE 2011
Szeged, Hungary
September 5, 2011
Why Binary Code?

- Source code isn’t available
- Source code isn’t the right representation
Binary Tools Need Symbol Tables

- **Debugging Tools**
  - GDB, IDA Pro…

- **Instrumentation Tools**
  - PIN, Dyninst,…

- **Static Analysis Tools**
  - CodeSurfer/x86,…

- **Security Analysis Tools**
  - IDA Pro,…

Labeling Library Functions in Stripped Binaries
Restoring Information

Function locations

<table>
<thead>
<tr>
<th>targ80c3bd0</th>
<th>targ80c3df4</th>
<th>targ80c3df4</th>
</tr>
</thead>
</table>

Complicated by:

- Missing symbol information
- Variability in function layout (e.g. code sharing, outlined basic blocks)
- High degree of indirect control flow
What about semantic information?

- Program’s interaction with the operating system (system calls) encapsulated by wrapper functions

Library fingerprinting: identify functions based on patterns learned from exemplar libraries
unstrip

stripped binary parsing +
library fingerprinting +
binary rewriting

targ80c3bd0
getpid
targ80c3df4
targ80c3df4
accept
Save registers

Invoke a system call

Set up system call arguments

Error check and return

<accept>:

```
mov %ebx, %edx
mov %0x66,%eax
mov $0x5,%ebx
lea 0x4(%esp),%ecx
int $0x80
mov %edx, %ebx
cmp %0xffffffff83,%eax
jae syscall_error
ret
```
The same function can be realized in a variety of ways in the binary.
Binary-level Code Variations

- Function inlining
- Code reordering
- Minor code changes
- Alternative code sequences
Semantic Descriptors

- Rather than recording byte patterns, we take a semantic approach
- Record information that is likely to be invariant across multiple versions of the function

```
<accept>:
    mov %ebx, %edx
    mov %0x66,%eax
    mov $0x5,%ebx
    lea 0x4(%esp),%ecx
    int $0x80
    mov %edx, %ebx
    cmp %0xffffffff83,%eax
    jae 8048300
    ret
    mov %esi,%esi
```
We parse an input binary, locate system calls and wrapper function calls, and employ dataflow analysis.
Building Semantic Descriptors Recursively

sethostid:
  ...
  call open
  ...
  call write
  ...
  mov $0x6, eax
  int $0x80

{<open, "/etc/hostid", 577, 420>}

write:
  ...
  mov $0x4, eax
  int $0x80
  ...

{<write, ?, ?, 4>}

{<close>, <open, "/etc/hostid", 577, 420>, <write, ?, ?, 4>}

{<close>}

Labeling Library Functions in Stripped Binaries
Building a Descriptor Database

Labeling Library Functions in Stripped Binaries

Locate wrapper functions

unstrip

glibc reference library

Build semantic descriptors

Descriptor Database

Locate wrapper functions

unstrip

{<socketcall, 5>}: accept

{<socketcall, 4>}: listen

{<getpid>}: getpid

...
Building a Descriptor Database

Locate wrapper functions

Descr**iptor Database

unstrip

{<socketcall, 5>}: accept

{<socketcall, 4>}: listen

{<getpid>}: getpid

…

Build semantic descriptors

glibc reference library

Labeling Library Functions in Stripped Binaries
Pattern Matching Criteria

- **Two stages**
  1) Exact matches
  2) Best match based on coverage criterion

- Handle minor code variations by allowing flexible matches
Pattern Matching Criteria

fingerprint from the \textbf{database}

\[
\downarrow
\]

A: \{<socketcall,5>\}

B: \{<socketcall,5>, <socketcall,5>, <futex>\}

\[
\uparrow
\]

semantic descriptor from the \textbf{code}

\[
\text{coverage}(A,B) = \frac{|A \cap B|}{|B|}
\]

\[
A \cap B = \{ b \in B \mid b \in A \}
\]

\[
\text{coverage}(A,B) = \frac{2}{3}
\]
Multiple Matches

- It’s possible that two or more functions are indistinguishable
- Policy decision: return set of potential matches
- In practice, we’ve observed 8% of functions have multiple matches, but the size of the match set is small ($\leq 3$)
Identifying Functions in a Stripped Binary

For each wrapper function
{
  1. Build the semantic descriptor.
  2. Search the database for a match (apply two-stage matching process).
  3. Add label to symbol table.
}

stripped binary

Descriptor Database

unstrip

unstripped binary
Implementation

stripped binary parsing +

library fingerprinting +

binary rewriting
Evaluation

- To evaluate across three dimensions of variation, we constructed three data sets:
  - GCC version
  - glibc version
  - distribution vendor

- In each set, compile statically-linked binaries, build a DDB, compare unstrip to IDA Pro’s FLIRT

- Evaluation measure is accuracy
Evaluation Results: GCC Version Study

GCC 3.4.4 Patterns Predicting Each Library

Accuracy

unstrip
IDA Pro
Evaluation Results: glibc Version Study

Labeling Library Functions in Stripped Binaries
Evaluation Results: Distribution Study

Fedora Patterns Predicting Each Library

<table>
<thead>
<tr>
<th></th>
<th>Fedora Patterns Predicting Each Library</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fedora</td>
</tr>
<tr>
<td>unstrip</td>
<td></td>
</tr>
<tr>
<td>IDA Pro</td>
<td></td>
</tr>
</tbody>
</table>
unstrip is available at

Backup slides follow
Evaluation Results: GCC Version Study
(Temporal: backwards)

Accuracy of GCC 4.2.1 Patterns Predicting Each Library

- 3.4.4
- 4.0.2
- 4.1.2
- 4.2.1

Labeling Library Functions in Stripped Binaries
Evaluation Results: glibc Version Study (Temporal: backwards)

Accuracy of glibc 2.11.1 Patterns Predicting Each Library

- **unstrip**
- **IDA Pro**

<table>
<thead>
<tr>
<th>Version</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.4</td>
<td></td>
</tr>
<tr>
<td>2.3.2</td>
<td></td>
</tr>
<tr>
<td>2.3.4</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.11.1</td>
<td></td>
</tr>
</tbody>
</table>

Labeling Library Functions in Stripped Binaries
Evaluation Results: Distribution Study
(one predicts the rest)

Mandrivia Patterns Predicting Each Library

- Fedora
- Mandrivia
- OpenSuse
- Ubuntu

Accuracy

- unstrip
- IDA Pro
Evaluation Results: GCC Version Study (one predicts the rest)

Accuracy

<table>
<thead>
<tr>
<th>GNU C Compiler Version</th>
<th>3.4.4</th>
<th>4.0.2</th>
<th>4.1.2</th>
<th>4.2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>unstrip</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDA Pro</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Labeling Library Functions in Stripped Binaries
Evaluation Results: glibc Version Study
(one predicts the rest)

- glibc
- IDA Pro

Accuracy

<table>
<thead>
<tr>
<th>glibc version</th>
<th>Unstrip</th>
<th>IDA Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.4</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2.3.2</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2.3.4</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2.5</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2.11.1</td>
<td>1</td>
<td>0.25</td>
</tr>
</tbody>
</table>
Evaluation Results: Distribution Study
(one predicts the rest)

Accuracy

Distribution Vendor

Fedora
Mandrivia
OpenSuse
Ubuntu

unstrip
IDA Pro