Extracting Compiler Provenance From Program Binaries

Nathan Rosenblum
Computer Sciences Department
University of Wisconsin

nater@cs.wisc.edu
http://pages.cs.wisc.edu/~nater/

Joint work with
Barton Miller and Xiaojin Zhu
Provenance in two parts

source

binary
production
toolchain

program
binary
Provenance in two parts

- compiler family
- compiler version
- library/operating system versions
- optimization level
- link-time optimization
- binary rewriting [obfuscation]
Provenance in two parts

- compiler family
- compiler version
- library/operating system versions
- optimization level
- link-time optimization
- binary rewriting [obfuscation]
Provenance in two parts

- compiler family
- compiler version
- library/operating system versions
- optimization level
- link-time optimization
- binary rewriting [obfuscation]

source

binary production toolchain

program binary

GCC? ICC?
Provenance in two parts

- source
- binary production toolchain
- program binary

- compiler family
- compiler version
- library/operating system versions
- optimization level
- link-time optimization
- binary rewriting [obfuscation]
Provenance in two parts

source

binary production toolchain

Program binary

stripped binary artifact

exhaustive disassembly

control flow graph
Why compiler provenance?
Why compiler provenance?

Dyn

inst

ParseAPI

A Dyninst Component

IDA Pro
Why compiler provenance?

Dyn
inst

ParseAPI
A Dyninst Component

IDA Pro

Thursday, October 7, 2010
Why compiler provenance?

 Dyn inst

 ParseAPI

 IDA Pro

 Thursday, October 7, 2010
Why should this work?
int bar(int foo) {
    int i, j;

    for(i=0; i<foo; ++i) {
        i = j + i;
        j *= i;
    }
    return j;
}

int bar(int foo) {
    int i, j;
    for(i=0;i<foo;++i) {
        i = j + i;
        j *= i;
    } 
    return j;
}

GCC

    test    edi,edi
    jle    4004ae <bar+0x16>
    mov    eax,0x0
    lea    eax,[rdx+rax]
    imul   edx,eax
    add    eax,0x1
    cmp    edi,eax
    jg     4004a1 <bar+0x9>
    mov    eax,edx
    ret

ICC

    xor    edx,edx
    test   edi,edi
    jle    400989 <bar+0x11>
    add    edx,eax
    imul   eax,edx
    inc    edx
    cmp    edx,edi
    jl     40097e <bar+0x6>
    ret
int bar(int foo) {
    int i, j;

    for(i=0; i<foo; ++i) {
        i = j + i;
        j *= i;
    }
    return j;
}
Binary code model

… c7 04 24 10 70 05 08 ff d0 c9 c3 90 81 ec e4 00 00 00 8b b4 24 ec 00 00 00 …

program bytes
Binary code model

 GCC  ICC  ICC  GCC

program binary

... c7 04 24 10 70 05 08 ff d0 c9 c3 90 81 ec e4 00 00 00 8b b4 24 ec 00 00 00 ...

program bytes
Binary code model

... c7 04 24 10 70 05 08 ff d0 c9 c3 90 81 ec e4 00 00 00 8b b4 24 ec 00 00 00 ...

Program bytes

addresses

8d b4 26
00 00 00
00 8d bc
27 00 00
00 00 90
80 4c 90
80 4c 94
80 4c 98
80 4c 9b

match_init
zp_init_keys
seekable

data

Thursday, October 7, 2010
Binary code model

 GCC   ICC   ICC   GCC

 program binary

\[ \ldots \text{c7 04 24 10 70 05 08 ff d0 c9 c3 90 81 ec e4 00 00 00 8b b4 24 ec 00 00 00} \ldots \]

program bytes
Binary code model

sequence labels

\[ y_{i-1} \rightarrow y_i \rightarrow \cdots \rightarrow y_j \rightarrow y_{j+1} \in \{icc, gcc, \ldots, data\} \]

program bytes

\[ \ldots \text{c7 04 24 10 70 05 08 ff d0 c9 c3 90 81 ec e4 00 00 00 8b b4 24 ec 00 00 00 \ldots} \]
Binary code model

 GCC  ICC  ICC  GCC

sequence labels

\[ y_{i-1} \rightarrow y_i \rightarrow \ldots \rightarrow y_j \rightarrow y_{j+1} \in \{icc,gcc,\ldots,\text{data}\} \]

... c7 04 24 10 70 05 08 ff d0 c9 c3 90 81 ec e4 00 00 00 8b b4 24 ec 00 00 00 ...
Binary code model

... c7 04 24 10 70 05 08 ff d0 c9 c3 90 81 ec e4 00 00 00 8b b4 24 ec 00 00 00 ...

sequence labels

$y_{i-1}$  $y_i$  ...

$y_j$  $y_{j+1}$  $\in \{\text{icc, gcc, ... data}\}$

program bytes

Thursday, October 7, 2010
Binary code features

```plaintext
<push EBP ; * ; mov ESP, EBP>

<mov [IMM], RAX ; sub [IMM], RAX>
```
Binary code features

- single-instruction wildcard
- opcode class abstraction
- hidden immediates

```plaintext
<push EBP ; * ; mov ESP, EBP>
<mov [IMM], RAX ; sub [IMM], RAX>
```
Binary code features

single-instruction wildcard

<push EBP ; * ; mov ESP, EBP>

<mov [IMM], RAX ; sub [IMM], RAX>

opcode class abstraction

hidden immediates

branch

long-range control flow interaction
Learning framework

feature functions

\( f_{I(u,c)}(x_i, y_i) \)  \( f_{T(c,c')}(y_i) \)  \( f_{CF}(y_i, y_j) \)

weights (learned)

\( \lambda_{I(u,c)} \)  \( \lambda_{T(c,c')} \)  \( \lambda_{CF} \)
Learning framework

\[ f_{I(u,c)}(x_i, y_i) \quad f_{T(c,c')}(y_i) \quad f_{CF}(y_i, y_j) \]

\[ \lambda_{I(u,c)} \quad \lambda_{T(c,c')} \quad \lambda_{CF} \]

Linear-chain CRF

Lafferty, et al. 2001
Learning framework

\[ f_{\text{I}(u,c)}(x_i, y_i) \quad f_{\text{T}(c,c')}(y_i) \quad f_{\text{CF}(y_i,y_j)} \]

\[ \lambda_{\text{I}(u,c)} \quad \lambda_{\text{T}(c,c')} \quad \lambda_{\text{CF}} \]

Linear-chain CRF
Lafferty, et al. 2001

[approximate]
Single-compiler provenance

\[
\begin{array}{ccc}
01110101101 & 01110101101 & 01110101101 \\
01010101011 & 01010101011 & 01010101011 \\
1011001010 & 1011001010 & 1011001010 \\
1011000100 & 1011000100 & 1011000100 \\
10010110101 & 10010110101 & 10010110101 \\
1001010101 & 1001010101 & 1001010101 \\
01010101001 & 01010101001 & 01010101001 \\
\end{array}
\]

GCC    ICC    MSVC

20      20      20

training binaries

\[
\begin{align*}
\lambda & \quad I(u,c) \\
\lambda & \quad T(c,c')
\end{align*}
\]

\~80k parameters
Single-compiler provenance

- GCC
- ICC
- MSVC

~80k parameters

\[ \lambda \quad I(u,c) \quad \lambda \quad T(c,c') \]

sequence labeling

92.5% accuracy
Mixed-compiler provenance

GCC 4.3  ICC 10.1
(per-file random selection)

10 × 10

GNU coreutils
programs

random compilations

\( \lambda \ I(u,c) \) \( \lambda \ T(c,c') \)

\( x \) program location
\( y \) assigned label
\( u \) idiom
\( c \) compiler

(Previously learned)
Mixed-compiler provenance

GCC 4.3  ICC 10.1
(per-file random selection)

10  x  10

GNU coreutils  random compilations
programs

( previously learned)

\[ \lambda I(u,c) \lambda T(c,c') \]

sequence labeling

93.8% accuracy
Stripped binary parsing

exhaustive disassembly

CFG
Stripped binary parsing

exhaustive disassembly

CFG

Which model?

\[ \lambda_{\mathcal{U}} \]  
GCC

\[ \lambda_{\mathcal{U}} \]  
ICC

\[ \lambda_{\mathcal{U}} \]  
MSVC

Rosenblum, et al. 2008
Integrating compiler provenance

$\lambda I(u)$
Integrating compiler provenance

\[ \lambda \mathcal{I}(u) \times \{ \text{GCC, ICC, MSVS} \} \rightarrow \lambda \mathcal{I}(u,c) \]
Integrating compiler provenance

\[ \lambda \Omega(u) \times \{\text{GCC, ICC, MSVS}\} \rightarrow \lambda \Omega(u,c) \]
Integrating compiler provenance

\[ \lambda I(u) \times \{ \text{GCC, ICC, MSVS} \} \rightarrow \lambda I(u,c) \]
Integrating compiler provenance

\[
\lambda \mathcal{I}(u) \times \{\text{GCC, ICC, MSVS}\} \rightarrow \lambda \mathcal{I}(u,c)
\]

\[
\begin{array}{c}
\text{GCC} \\ \\
\text{ICC} \\ \\
\text{ICC} \\ \\
\text{ICC} \\ \\
\text{GCC}
\end{array}
\]

\[
\mathcal{f}_{\mathcal{I}(u, \text{GCC})}(x_i, y_i) \quad \mathcal{f}_{\mathcal{I}(u, \text{ICC})}(x_i, y_i)
\]
Integrating compiler provenance

\[ \lambda \mathcal{I}(u) \times \{\text{GCC, ICC, MSVS}\} \rightarrow \lambda \mathcal{I}(u,c) \]

\[ f_{\mathcal{I}(u, \text{GCC})}(x_i, y_i) \quad f_{\mathcal{I}(u, \text{ICC})}(x_i, y_i) \]

18% error reduction
Future directions

toolchain provenance
Future directions

Compiler
  family
  version
  optimization level
  source language
toolchain provenance
Future directions

System
- glibc static code
- library imports

Compiler
- family
- version
- optimization level
- source language

toolchain provenance

done

Thursday, October 7, 2010
Future directions

System
- glibc static code
- library imports

Compiler
- family
- version
- optimization level
- source language

Link & post-link
- whole-program optimization
- rewriting tools
- obfuscation tools

toolchain provenance

done

Thursday, October 7, 2010
questions