Bintrimmer: Towards Static Binary Debloating Through Abstract Interpretation

DIMVA, June 19th 2019

Nilo Redini
Computer Science @ UC Santa Barbara
nredini@cs.ucsb.edu
Motivation

- Software complexity pushes developers toward **component re-use**
- Programs **bloated** with unused code
Motivation

- Software complexity pushes developers toward **component re-use**
- Programs **bloated** with unused code

Unused code can be used to **harm** users

```
0x804fd2c:   pop rdi       pointer to “bin/sh”
             ret
...

0x7fff4cdda: pop rsi       pointer to null
             ret
...

0x805ccac:   pop rdx       pointer to null
             ret

0x7fff39cd4: spawn shell  (execve)
```
Motivation

- Software complexity pushes developers toward **component re-use**
- Programs **bloated** with unused code

Unused code can be used to **harm** users

Remove **dead code** to reduce **attack surface**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x804fd2c</td>
<td><code>pop rdi</code></td>
<td>pointer to &quot;bin/sh&quot;</td>
</tr>
<tr>
<td></td>
<td><code>ret</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>...</strong></td>
<td></td>
</tr>
<tr>
<td>0x7fff4cdda</td>
<td><code>pop rsi</code></td>
<td>pointer to null</td>
</tr>
<tr>
<td></td>
<td><code>ret</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>...</strong></td>
<td></td>
</tr>
<tr>
<td>0x805ccac</td>
<td><code>pop rdx</code></td>
<td>pointer to null</td>
</tr>
<tr>
<td></td>
<td><code>ret</code></td>
<td></td>
</tr>
<tr>
<td>0x7fff39cd4</td>
<td>spawn shell</td>
<td>(execve)</td>
</tr>
</tbody>
</table>
Current Techniques

State-of-the-art debloating techniques require:

- Source code
- Test cases
- Runtime support
Current Techniques

State-of-the-art debloating techniques require:

- Source code not always available
- Test cases
- Runtime support
Current Techniques

State-of-the-art debloating techniques require:

- Source code
- Test cases → unreliable programs
- Runtime support
Current Techniques

State-of-the-art debloating techniques require:

- Source code
- Test cases
- Runtime support for different architectures
Debloating

Can we statically identify and remove unused code when only the binary program is available?
Debloating

Build a *complete* & *sound* Control-Flow Graph, and remove the code not referenced
Debloating

Build a *complete & sound* Control-Flow Graph, and remove the code not referenced

Undecidable $\rightarrow$ *Impossible*!
Debloating

Build a *complete* & *sound* Control-Flow Graph, and remove the code not referenced
Undecidable $\Rightarrow$ *Impossible!*

Sound debloating requires a *complete* Control-Flow Graph
Debloating

Build a *complete* & *sound* Control-Flow Graph, and remove the code not referenced
Undecidable ~> *Impossible*

Sound debloating requires a *complete* Control-Flow Graph
Completeness without precision ~> Uneffective debloating
Debloating

Assuming we have a complete but imprecise CFG, how do we increase its precision?
Debloating

Assuming we have a complete but imprecise CFG, how do we increase its precision?

Through a precise approximation of variable values (e.g., function pointers)
Assuming we have a complete but imprecise CFG, how do we increase its precision?

Through a precise approximation of variable values (e.g., function pointers)

Define a precise abstract domain
Example

```c
void main() {
    uint8_t opt;
    void (*f_ptr)( void ) = [foo, bar, baz]; // foo, bar, and baz are defined in another module

    scanf("%"SCNu8, &opt);
    opt = (opt * 2) + 1;
    // ...
    if (opt == 0) {
        f_ptr[0](); // call to foo
    } else if (opt == 100) {
        f_ptr[1](); // call to bar
    } else if (opt < 0) {
        f_ptr[2](); // call to baz
    }
}
```
Example

```c
void main() {
    uint8_t opt;
    void (*f_ptr)( void ) = [foo, bar, baz];  // foo, bar, and baz are
        // defined in another module

    scanf("%"SCNu8, &opt);
    opt = (opt * 2) + 1;
    // ... 
    if (opt == 0) {
        f_ptr[0](); // call to foo
    } else if (opt == 100){
        f_ptr[1](); // call to bar
    } else if (opt < 0) {
        f_ptr[2](); // call to baz
    }
}
```
void main() {
    uint8_t opt;
    void (*f_ptr)( void ) = [foo, bar, baz]; // foo, bar, and baz are
    // defined in another module

    scanf("%"SCNu8, &opt);
    opt = (opt * 2) + 1;
    // ...
    if (opt == 0) {
        f_ptr[0](); // call to foo
    } else if (opt == 100) {
        f_ptr[1](); // call to bar
    } else if (opt < 0) {
        f_ptr[2](); // call to baz
    }
}
```c
void main() {
    uint8_t opt;
    void (*f_ptr)( void ) = [foo, bar, baz]; // foo, bar, and baz are defined in another module
    scanf("%"SCNu8, &opt);
    opt = (opt * 2) + 1;
    // ...
    if (opt == 0) {
        f_ptr[0](); // call to foo
    } else if (opt == 100) {
        f_ptr[1](); // call to bar
    } else if (opt < 0) {
        f_ptr[2](); // call to baz
    }
}
```
void main() {
uint8_t opt;
void (*f_ptr)( void ) = [foo, bar, baz]; // foo, bar, and baz are
                                          // defined in another module
scanf("%"SCNu8, &opt);
opt = (opt * 2) + 1;
// ...
if (opt == 0) {
    f_ptr[0](); // call to foo
} else if (op == 100){
    f_ptr[1](); // call to bar
} else if (opt < 0) {
    f_ptr[2](); // cal to baz
}
}
void main() {
    uint8_t opt;
    void (*f_ptr)(void) = [foo, bar, baz]; // foo, bar, and baz are defined in another module

    scanf("%" SCNu8, &opt);
    opt = (opt * 2) + 1;
    // ...
    if (opt == 0) {
        f_ptr[0](); // call to foo
    } else if (op == 100) {
        f_ptr[1](); // call to bar
    } else if (opt < 0) {
        f_ptr[2](); // cal to baz
    }
}
Example

```c
void main() {
    uint8_t opt;
    void (*f_ptr)(void) = [foo, bar, baz]; // foo, bar, and baz are defined in another module

    scanf("%"SCNu8, &opt);
    opt = (opt * 2) + 1;
    // ...
    if (opt == 0) {
        f_ptr[0](); // call to foo
    } else if (opt == 100) {
        f_ptr[1](); // call to bar
    } else if (opt < 0) {
        f_ptr[2](); // call to baz
    }
}
```
Signedness of Variables

While it is easy to detect the signedness of a variable in source code, it is harder on binary programs.
Signedness of Variables

While it is easy to detect the signedness of a variable in source code, it is harder on binary programs.

The abstract domain must be signedness-agnostic
BinTrimmer
High-level Idea

**Goal:** We want to recover a *complete and precise* CFG, thus guaranteeing program functionality and effective debloating.

The more precise the CFG is, the more we can trim!
**High-level Idea**

**Goal:** We want to recover a **complete and precise** CFG, thus guaranteeing program functionality and effective debloating.

The more precise the CFG is, the more we can trim!

*Signedness-Agnostic Strided Intervals (SASI)*
Signedness-Agnostic Strided Intervals

\[ I = s_I[lb, ub]w \]
Signedness-Agnostic Strided Intervals

\[ I = s_I [lb, ub]_w \]

\[ I = \left\{ (lb + k \times s_I) \mod 2^w \leq ub, k \in \mathbb{N} \right\} \]

+ represents modular addition of bit-width \( w \)

Example:

\[ 2[1010, 0010]_4 = \{1010, 1100, 1110, 0000, 0010\} \]
Signedness-Agnostic Strided Intervals

Number circle ~> Capture overflow behavior of variables on a computer
Signedness-Agnostic Strided Intervals

Number circle ~> Capture overflow behavior of variables on a computer

Stride ~> To increase the precision of the values represented by an element in SASI
Signedness-Agnostic Strided Intervals

Number circle ~> Capture overflow behavior of variables on a computer

Stride ~> To increase the precision of the values represented by an element in SASI

Signedness Agnosticity and Soundness ~> Achieved by a careful design of the operations on SASI
Example: Addition

Given two SASI $r = S_r [a, b]w$ and $t = S_t [c, d]w$, addition is defined as follows:

$$r +_w t = \begin{cases} \bot & \text{if } r = \bot \lor t = \bot \\ S_s [a +_w c, b +_w d] & \text{if } \#r + \#t \leq 2^w \\ \top & \text{otherwise} \end{cases}$$

where $S_s = \gcd (S_r, S_t)$
CFG Refinement

Function $P_{\text{CFG}}$
CFG Refinement
CFG Refinement

Diagram:

- "P_{CFG}", "functions set", "SASI", "VSA", "Target Solver", "Branch Annotator", and "Checker" are interconnected in the diagram.

Function flow:
- "Function" leads to "P_{CFG}"
- "functions set" leads to "SASI" and "VSA"
- "SASI" and "VSA" lead to "Target Solver" and "Branch Annotator"
- "Target Solver" and "Branch Annotator" lead to "Checker"
CFG Refinement

**Diagram Description:**

- The diagram shows a process involving CFG refinement.
- The process starts with a function $P_{CFG}$.
- The function $P_{CFG}$ outputs a set of functions which are then further processed.
- The processing includes steps such as SASI and VSA.
- The refined $CFG_i$ is then filtered to produce $CFG_p$.
- The workflow includes additional functions and checks.
Program Debloating

**Delete code**

+ Lighter Binaries
- Pointers must be updated

**Modify code**

+ Guarantee Functionality (no need to fix pointers)
- Same size
BinTrimmer

Static Binary Trimming tool

Leverage SASI to refine CFG and identify dead code

Rewrite dead code with `halt`

Implemented on top of `angr`
Experimental Results
SASI vs. Wrapped Intervals (on Sources)
SASI vs. Wrapped Intervals (on Binaries)
## Trimming Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Total trimmed</th>
<th>Min Partials</th>
<th>Max Partials</th>
<th>Avg Partials</th>
<th>Gadgets Removed</th>
<th>Tot ICF</th>
<th>ICF Resolved angr</th>
<th>ICF Resolved BINTRIMMER</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyExpr1</td>
<td>53.69%</td>
<td>3.62%</td>
<td>83.63%</td>
<td>29.12%</td>
<td>41.3%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>43</td>
</tr>
<tr>
<td>TinyExpr2</td>
<td>7.43%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.9%</td>
<td>2449</td>
<td>98.65%</td>
<td>100%</td>
<td>87</td>
</tr>
<tr>
<td>TinyExpr3</td>
<td>65.67%</td>
<td>3.7%</td>
<td>83.63%</td>
<td>40.33%</td>
<td>56.9%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>37</td>
</tr>
<tr>
<td>b641</td>
<td>1.17%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3.0%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b642</td>
<td>50.37%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>10.6%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b643</td>
<td>34.43%</td>
<td>1.13%</td>
<td>0%</td>
<td>0%</td>
<td>36.4%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>22</td>
</tr>
</tbody>
</table>
## Trimming Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Total trimmed</th>
<th>Min Partials</th>
<th>Max Partials</th>
<th>Avg Partials</th>
<th>Gadgets Removed</th>
<th>Tot ICF</th>
<th>ICF</th>
<th>Resolved angr</th>
<th>ICF</th>
<th>Resolved BINTRIMMER</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyExpr1</td>
<td>53.69%</td>
<td>3.62%</td>
<td>83.63%</td>
<td>29.12%</td>
<td>41.3%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>100%</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>TinyExpr2</td>
<td>7.43%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.9%</td>
<td>2449</td>
<td>98.65%</td>
<td>100%</td>
<td>100%</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>TinyExpr3</td>
<td>65.67%</td>
<td>3.7%</td>
<td>83.63%</td>
<td>40.33%</td>
<td>56.9%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>100%</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>b641</td>
<td>1.17%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3.0%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>100%</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>b642</td>
<td>50.37%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>10.6%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>100%</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>b643</td>
<td>34.43%</td>
<td>1.13%</td>
<td>0%</td>
<td>0%</td>
<td>36.4%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>100%</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>
## Trimming Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Total trimmed</th>
<th>Min Partials</th>
<th>Max Partials</th>
<th>Avg Partials</th>
<th>Gadgets Removed</th>
<th>Tot ICF</th>
<th>ICF Resolved</th>
<th>ICF Resolved</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyExpr1</td>
<td>53.69%</td>
<td>3.62%</td>
<td>83.63%</td>
<td>29.12%</td>
<td>41.3%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>43</td>
</tr>
<tr>
<td>TinyExpr2</td>
<td>7.43%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.9%</td>
<td>2449</td>
<td>98.65%</td>
<td>100%</td>
<td>87</td>
</tr>
<tr>
<td>TinyExpr3</td>
<td>65.67%</td>
<td>3.7%</td>
<td>83.63%</td>
<td>40.33%</td>
<td>56.9%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>37</td>
</tr>
<tr>
<td>b641</td>
<td>1.17%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3.0%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b642</td>
<td>50.37%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>10.6%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b643</td>
<td>34.43%</td>
<td>1.13%</td>
<td>0%</td>
<td>0%</td>
<td>36.4%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>22</td>
</tr>
</tbody>
</table>
### Trimming Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Total trimmed</th>
<th>Min Partials</th>
<th>Max Partials</th>
<th>Avg Partials</th>
<th>Gadgets Removed</th>
<th>Tot ICF</th>
<th>ICF angr</th>
<th>Resolved BinTRIMMER</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyExpr1</td>
<td>53.69%</td>
<td>3.62%</td>
<td>83.63%</td>
<td>29.12%</td>
<td>41.3%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>43</td>
</tr>
<tr>
<td>TinyExpr2</td>
<td>7.43%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.9%</td>
<td>2449</td>
<td>98.65%</td>
<td>100%</td>
<td>87</td>
</tr>
<tr>
<td>TinyExpr3</td>
<td>65.67%</td>
<td>3.7%</td>
<td>83.63%</td>
<td>40.33%</td>
<td>56.9%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>37</td>
</tr>
<tr>
<td>b641</td>
<td>1.17%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3.0%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b642</td>
<td>50.37%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>10.6%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b643</td>
<td>34.43%</td>
<td>1.13%</td>
<td>0%</td>
<td>0%</td>
<td>36.4%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>22</td>
</tr>
</tbody>
</table>
## Trimming Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Total trimmed</th>
<th>Min Partials</th>
<th>Max Partials</th>
<th>Avg Partials</th>
<th>Gadgets Removed</th>
<th>Tot ICF</th>
<th>ICF Resolved angr</th>
<th>ICF Resolved BINTRIMMER</th>
<th>Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyExpr1</td>
<td>53.69%</td>
<td>3.62%</td>
<td>83.63%</td>
<td>29.12%</td>
<td>41.3%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>43</td>
</tr>
<tr>
<td>TinyExpr2</td>
<td>7.43%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.9%</td>
<td>2449</td>
<td>98.65%</td>
<td>100%</td>
<td>87</td>
</tr>
<tr>
<td>TinyExpr3</td>
<td>65.67%</td>
<td>3.7%</td>
<td>83.63%</td>
<td>40.33%</td>
<td>56.9%</td>
<td>2419</td>
<td>99.25%</td>
<td>100%</td>
<td>37</td>
</tr>
<tr>
<td>b641</td>
<td>1.17%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3.0%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b642</td>
<td>50.37%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>10.6%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>24</td>
</tr>
<tr>
<td>b643</td>
<td>34.43%</td>
<td>1.13%</td>
<td>0%</td>
<td>0%</td>
<td>36.4%</td>
<td>2389</td>
<td>99.6%</td>
<td>100%</td>
<td>22</td>
</tr>
</tbody>
</table>
Conclusions

New abstract domain: SASI

98% more precise than state-of-the-art!

BinTrimer: Static Binary Debloating

Sound debloating: programs **guaranteed** to work!

No test cases needed

No source code needed

Remove up to 65.6% of a library’s code
Thanks! && Questions?

Nilo Redini

nredini@cs.ucsb.edu
https://badnack.it
@badnack