SafeDrive: Safe and Recoverable Extensions Using Language-Based Techniques

Feng Zhou, Jeremy Condit, Zachary Anderson, Ilya Bagrak, Rob Ennals, Matthew Harren, George Necula, Eric Brewer
CS Division, UC Berkeley
and Intel Research Berkeley

http://ivy.cs.berkeley.edu/safedrive/
The Problem

• OSes and applications often run loadable extensions
  – e.g. Linux kernel, Apache, Firefox
  – Run in the same protection domain

• Extensions are often buggier than hosts
  – Device drivers cause a large percentage of Windows crashes
  – Xbox hacked due to memory bugs in games

• SafeDrive detects and recovers from type-safety and memory-safety errors in Linux device drivers
Approaches

- Separate hardware protection domains: Nooks [Swift et al], L4 [LeVasseur et al], Xen [Fraser et al]
  - Relatively high overhead due to cross-domain calls, system specific
- Binary instrumentation: SFI [Wahbe et al, Small/Seltzer]
  - High overhead, coarse-grained
- Static analysis + software guards: XFI [Erlingsson et al]
  - Control flow safety
- What can be done at the C language level?
  - Add fined-grained type-safety, to extensions only
  - A way to recover from failures
A Language-Based Approach to Extension Safety

• Light annotations in extension code and host API
  – Buffer bounds, non-null pointers, nullterm strings, tagged unions
• Deputy src-to-src compiler emits safety checks when necessary
• Key: compatible extension-host binary interface
• Runtime tracks resource usage and restores system invariants at fail time
Deputy: Motivation

```c
struct {
    unsigned int len;
    int * data;
} x;
for (i=0;i<x.len;i++) {
    ... x.data[i] ...
}
```

- Common C code
- How to check memory safety?
- C pointers do not express extent of buffers (unlike Java)
Previous Approach: Fat Pointers

- Used in CCured and Cyclone
- Compiler inserts extra bounds variables
- Changes memory layout
- Cannot be applied modularly

```c
struct {
    unsigned int len;
    int * data;
    int * data_b;
    int * data_e;
} x;

for (i = 0; i < x.len; i++) {
    if (x.data+i<x.data_b) abort();
    if (x.data+i>=x.data_e) abort();
    ... x.data[i] ...
}
```
Deputy Bounds Annotations

```c
struct {
    unsigned int len;
    int * count(len) data;
} x;
for(i = 0; i < x.len; i++) {
    if (i<0 || i>=x.len) abort();
    ... x.data[i] ...
}
```

- Annotations use existing bounds info in programs, or constants
- Compiler emits runtime checks
- No memory layout change ➔ Can be applied to one extension a time
- Many checks can be optimized away
Deputy Features

- Bounds: \texttt{safe}, \texttt{count(n)}, \texttt{bound(lo,hi)}
  - Default: \texttt{safe}
- Other annotations
  - Null terminated string/buffer
  - Tagged unions
  - Open arrays
  - Checks for \texttt{printf()} arguments
- Automatic bounds variables for local variables
  $\rightarrow$ reduced annotation burden
Deputy Guarantees

• Deputy guarantees type-safety if,
  – Programmer correctly annotates globals and function parameters used by the extension
  – Deallocation does not create dangling pointers
  – Trusted casts are correct
  – External modules / trusted code establish and preserve Deputy annotations
Failure Handling

- Everything runs inside the same protection domain
- After Deputy check failure: could just halt
- More useful: clean-up extension and let host continue
- Assumption: restarts should fix most transient failures
Update Tracking and Restarts

- Free resources and undo state changes done by driver
- Kernel API functions “wrapped” to do update tracking
  - Compensations: `spin_lock(l)` vs. `spin_unlock(l)`
- After failure, undo updates in LIFO order
- Then restart driver
Return Gracefully from Failure

Invariants:

• No driver code is executed after failure
Return Gracefully from Failure

Invariants:

• No driver code is executed after failure
• No kernel function is forced to return early
Discussion

• Compared to Nooks
  – Significantly less interception → Much simpler overall
  – Deputy does fine-grained per-allocation checks
    → No separate heap/stack
  – No help from virtual memory hardware
  – Works for user-level applications and safe languages

• Compared to C++/Java exceptions
  – Compensation does not contain any code from driver
  – Only restores host state, not extension state
Implementation

- Deputy compiler: 20K lines of OCaml
- Kernel patch to 2.6.15.5: 1K lines
- Kernel headers patch: 1.9K lines
- Patch for 6 drivers in 4 categories
  - Network: e1000, tg3
  - USB: usb-storage
  - Sound: intel8x0, emu10k1
  - Video: nvidia
Evaluation: Recovery Rate

• Inject random errors with compile-time injection: 5 errors from one of 7 categories each time
  – Faults chosen following empirical studies
    [Sullivan & Chillarege 91], [Christmansson & Chillarege 96]
    – Scan overrun, loop fault, corrupt parameter, off-by-one, flipped condition, missing call, missing assignment
• Load buggy e1000 driver w/ and w/o SafeDrive
• Exercise by downloading a 89MB file, verifying it and unloading the driver
• Then rerun with original driver
Recovery Rate Results

- 140 runs, 20 per fault category

<table>
<thead>
<tr>
<th>SafeDrive off</th>
<th>44 crashes</th>
<th>21 failures</th>
<th>75 passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SafeDrive on</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static error</td>
<td>10</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Runtime error</td>
<td>34</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>No problem detected</td>
<td>0</td>
<td>19</td>
<td>67</td>
</tr>
</tbody>
</table>

Recovery successes: 44 (100%) 2 (100%) 8 (100%)

- SafeDrive is effective at detecting and recovering from crashing problems, and can detect some statically.
Annotation Burden

- 1%-4% of lines with Deputy annotations
- Recovery wrappers can be automatically generated
Annotatons Break-down

<table>
<thead>
<tr>
<th>Lines Changed</th>
<th>Bounds</th>
<th>Strings</th>
<th>Tagged Unions</th>
<th>Trusted Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>All 6 drivers</td>
<td>1544</td>
<td>379</td>
<td>83</td>
<td>2</td>
</tr>
<tr>
<td>Kernel headers</td>
<td>1866</td>
<td>187</td>
<td>260</td>
<td>8</td>
</tr>
</tbody>
</table>

- Common reasons for trusted casts and trusted code
  - *Polymorphic private data*, e.g. `netdev->priv`
  - Small number of cases where buffer bounds are not available
  - Code manipulating pointer values directly, e.g. `PTR_ERR(x)`
Performance

Relative %, SafeDrive vs. native

-20 -10 0 10 20

<table>
<thead>
<tr>
<th>Component</th>
<th>Throughput</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1000 TCP recv</td>
<td>-17</td>
<td>4</td>
</tr>
<tr>
<td>e1000 UDP recv</td>
<td>-11</td>
<td>9</td>
</tr>
<tr>
<td>e1000 TCP send</td>
<td>-1.3</td>
<td>8</td>
</tr>
<tr>
<td>e1000 UDP send</td>
<td>-1.1</td>
<td>13</td>
</tr>
<tr>
<td>tg3 TCP recv</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>tg3 TCP send</td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>usb-storage untar</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>emu10k aplay</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>intel8x0 aplay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nvidia xinit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Nooks (Linux 2.4): e1000 TCP recv: 46% (vs. 4%),
e1000 TCP send: 111% (vs. 12%)
Conclusion

• SafeDrive does fine-grained memory safety checking for extensions with low overhead and few code changes
• A recovery scheme for in-process extensions via restarts
• It is feasible to get much of the safety guarantee in type-safe languages in extensions without abandoning existing systems in C
• Language technology makes extension isolation easier

http://ivy.cs.berkeley.edu/safedrive
http://deputy.cs.berkeley.edu
How do you change bounds/tags

```c
struct {
    unsigned int len;
    int * count(len) data;
} x;

1 x.data = NULL;
if (x.data!=NULL && (A<0||A>len)) abort

2 x.len = A;
if (B<sizeof(int)*x.len) abort

3 x.data = malloc(B);
```
Related Work

• Improving memory safety of C
  – Safe C-like language: Cyclone [Morrisett et al]
  – Hybrid checking (non-modular): CCured [Necula et al]
  – Type qualifiers for static checking: CQual [Foster et al, Johnson/Wagner], Sparse [Torvalds]

• Improving OS/extension reliability
  – Hardware protection: Nooks [Swift et al], L4 [LeVasseur et al], Xen [Fraser et al]
  – Binary instrumentation: SFI [Wahbe et al, Small/Seltzer], XFI [Erlingsson]
  – Using Cyclone: OKE [Bos/Samwel]
  – Static validation of API usage: SLAM [Ball et al]
  – Writing OS with safe language: Singularity [Patel et al]
More Deputy Features

• Checking types of arguments in `printf`-like functions
• Bounds for open arrays
• Special support for `memset()`, `memcpy()`
• Trusted casts for programmer to override the type system
Recovery Rate Results

<table>
<thead>
<tr>
<th>SafeDrive</th>
<th>Crashes</th>
<th>Malfunctions</th>
<th>Innocuous Errors</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>44</td>
<td>21</td>
<td>n/a</td>
<td>75</td>
</tr>
<tr>
<td>On</td>
<td>0</td>
<td>19</td>
<td>8</td>
<td>113</td>
</tr>
</tbody>
</table>

- 140 runs, 20 per fault category
- SafeDrive prevented all 44 crashes with 100% recovery rate
  - 5 of 7 categories caused crashes
  - All caused by memory-safety errors
Recovery Rate Results (2)

<table>
<thead>
<tr>
<th>Detection</th>
<th>Crashes</th>
<th>Mal-function</th>
<th>Innocuous</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>10</td>
<td>0</td>
<td>3</td>
<td>13 (24%)</td>
</tr>
<tr>
<td>Dynamic</td>
<td>34</td>
<td>2</td>
<td>5</td>
<td>47 (76%)</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>2</td>
<td>8</td>
<td>54</td>
</tr>
</tbody>
</table>

- 24% problems are detected statically, including 10 crashes
  - e.g. wrong constant size for `memcpy()`, deref of uninitialized ptr
– Wrappers are currently hand-written
– No session restoration for failed drivers