Back to the Future: Fault-tolerant Live Update with Time-traveling State Transfer

Cristiano Giuffrida  Călin Iorgulescu  Anton Kuijsten
Andrew S. Tanenbaum

Vrije Universiteit Amsterdam

USENIX Large Installation System Administration Conference

Washington, D.C., USA
November 3-8, 2013
The Update-without-downtime Problem

update complete, would you like to restart no—

update complete

YEAH, YEAH, WHATEVER.

shutting down...

WAIT! NO! I MEANT NO!
"If you think database patching is onerous, then try patching a SCADA system that’s running a power plant."

Kelly Jackson Higgins on the SCADA patch problem, 2013

Solution 1: “Spare time for downtime”
"In one of the biggest computer errors in banking history, Chemical Bank mistakenly deducted about $15 million from more than 100,000 customers’ accounts."

Saul Hansell, New York Times, 1994

Solution 2: “Roll your upgrades”
"Our research shows that 75% of successful attacks occur against previously known vulnerabilities for which a remediation was already available."

Neil MacDonald, Gartner Research, 2012

Solution 3: “Don’t patch, don’t tell”
"All problems in computer science can be solved by another level of indirection—but that will usually create another problem."

Butler Lampson, quoting David Wheeler

Our solution: “Live update”
Live Update in the Real World

Servers protected with Ksplice Uptrack:
100,000+ at more than 700 companies

Updates applied on production systems:
More than 2 million and counting

Source: http://www.ksplice.com
--- a/drivers/md/dm-crypt.c
+++ b/drivers/md/dm-crypt.c
@@ -690,6 +690,8 @@ bad3:
   bad2:
     crypto_free_tfm(tfm);
   bad1:
+  /* Must zero key material before freeing */
+  memset(cc, 0, sizeof(*cc) + cc->key.size * sizeof(u8));
   kfree(cc);
   return -EINVAL;
 }
@@ -706,6 +708,9 @@ static void crypt_dtr(...)
     crypto_free_tfm(cc->tfm);
     dm_put_device(ti, cc->dev);
+  
+  /* Must zero key material before freeing */
+  memset(cc, 0, sizeof(*cc) + cc->key.size * sizeof(u8));
   kfree(cc);
 }

Linux kernel security patch for CVE-2006-0095
--- a/example.c
+++ b/example.c
@@ -1,13 +1,12 @@
 struct s {
  int count;
- char str[3];
- short id;
+ int id;
+ char str[2];
 union u u;
- void *ptr;
 int addr;
- short *inner_ptr;
+ int *inner_ptr;
 } var;

 void example_init(char *str) {
- snprintf(var.str, 3, "%s", str);
+ snprintf(var.str, 2, "%s", str);
 }
WWW: What We Want

- Support for arbitrarily complex software updates.
- Support for generic C programs found “in the wild”.
- Automatic state transfer and state validation.
- Automatic detection of run-time and memory errors.
- Automatic error recovery from failed updates (hot rollback).
Process-level updates
Time-traveling State Transfer Overview

Automatic state transfer
Time-traveling State Transfer Overview

**Forward**

V1

State transfer

V2

**Backward**

State differencing

Automatic state validation
% readlink -f `which my-program`
/path/to/my-program-3.5.24

% ttst-ctl update /path/to/my-program-3.6.10 \`pidof my-program`

```
ttst: Live update requested for my-program.
ttst: Loading /path/to/my-program-3.6.10...
ttst: Applying changes...
ttst: Validating changes...
ttst: Cleaning up old version...
ttst: Live update done.
```

Version update
Sysadmin Interface

% readlink -f `which my-program`
/path/to/my-program-3.5.24

% ttst-ctl update /path/to/my-program-3.6.10 \ `pidof my-program`

ttst: Live update requested for my-program.
ttst: Loading /path/to/my-program-3.6.10...
ttst: Applying changes...
ttst: Validating changes...
ttst: Cleaning up old version...
ttst: Live update done.

ttst-ctl tool
Sysadmin Interface

% readlink -f `which my-program`
/path/to/my-program-3.5.24

% ttst-ctl update /path/to/my-program-3.6.10 \ `pidof my-program`

| ttst: Live update requested for my-program. |
| ttst: Loading /path/to/my-program-3.6.10... |
| ttst: Applying changes... |
| ttst: Validating changes... |
| ttst: Cleaning up old version... |
| ttst: Live update done. |

Live update applied automatically
Live update initialization
TTST-enabled Live Update

Future Version

Instrumented Program
Live Update Lib
ST Framework Lib
TTST Control Lib

Past Version
Instrumented Program
Live Update Lib
ST Framework Lib
TTST Control Lib

Reversed Version
Instrumented Program
Live Update Lib
ST Framework Lib
TTST Control Lib

Forward state transfer
TTST-enabled Live Update

Future Version

Instrumented Program
Live Update Lib
ST Framework Lib
TTST Control Lib

Past Version

Instrumented Program
Live Update Lib
ST Framework Lib
TTST Control Lib

Reversed Version

Instrumented Program
Live Update Lib
ST Framework Lib
TTST Control Lib

Forward state transfer
TTST-enabled Live Update

Future Version

Instrumented Program
- Live Update Lib
- ST Framework Lib
- TTST Control Lib

CFT

Past Version

Instrumented Program
- Live Update Lib
- ST Framework Lib
- TTST Control Lib

Reversed Version

Instrumented Program
- Live Update Lib
- ST Framework Lib
- TTST Control Lib

Forward state transfer

Back to the Future: Fault-tolerant Live Update with Time-traveling State Transfer

Cristiano Giuffrida
Backward state transfer
Backward state transfer
Backward state transfer
TTST-enabled Live Update

**Past Version**
- Instrumented Program
- Live Update Lib
- ST Framework Lib
- TTST Control Lib

**Future Version**
- Instrumented Program
- Live Update Lib
- ST Framework Lib
- TTST Control Lib

**Reversed Version**
- Instrumented Program
- Live Update Lib
- ST Framework Lib
- TTST Control Lib

TIME-TRAVELING

STATE TRANSFER COMPLETED

Automatic state validation
TTST-enabled Live Update

Future Version

Past Version

Reversed Version

Instrumented Program
Live Update Lib
ST Framework Lib
TTST Control Lib

STATE DIFF

Automatic state validation
TTST-enabled Live Update

Future Version

Past Version

Reversed Version

Live update completed
State Transfer Instrumentation

Original Program
- Data
- Code

Before Instrumentation

LLVM

Statically Instrumented Program
- Data
- Metadata
- Instrumented code
- TTST libraries

After Instrumentation
State Transfer Example

```c
struct s { //old version
    int count;
    - char str[3];
    - short id;
    union IXFER(my_u) u;
    - void *ptr;
    PXFER(int) addr;
    - short *inner_ptr;
} var;

struct s { //new version
    int count;
    + int id;
    + char str[2];
    union IXFER(my_u) u;
    PXFER(int) addr;
    + int new_element;
    + int *inner_ptr;
} var;
```

<table>
<thead>
<tr>
<th></th>
<th>Old Version</th>
<th>New Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>str</td>
<td>'aa\0'</td>
<td>'a\0'</td>
</tr>
<tr>
<td>id</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>u</td>
<td>{12,32}</td>
<td>{12,32}</td>
</tr>
<tr>
<td>ptr</td>
<td>0x...3f</td>
<td>0x...7f</td>
</tr>
<tr>
<td>addr</td>
<td>0x...4f</td>
<td></td>
</tr>
<tr>
<td>inner_ptr</td>
<td>&amp;var.id</td>
<td>&amp;var.id</td>
</tr>
</tbody>
</table>
Sample diff result: 2 memory pages differ
### Time-traveling State Transfer Complexity

<table>
<thead>
<tr>
<th>Category</th>
<th>Total LOC</th>
<th>Trusted LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LLVM plugins</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>State transfer instrumentation</em></td>
<td>9,330</td>
<td>1,119</td>
</tr>
<tr>
<td><strong>Support libraries</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Live update library</em></td>
<td>382</td>
<td>235</td>
</tr>
<tr>
<td><em>TTST control library</em></td>
<td>3,209</td>
<td>412</td>
</tr>
<tr>
<td><em>State transfer framework</em></td>
<td>13,311</td>
<td>0</td>
</tr>
<tr>
<td><strong>Sysadmin interface</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>ttst-ctl tool</em></td>
<td>381</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>26,613</td>
<td>1,766</td>
</tr>
</tbody>
</table>
Experience

- Applied 40 updates (41 KLOC) on httpd, nginx, vsftpd, sshd.

- Forward state transfer required 896 LOC overall.

- Full-coverage backward state transfer required 299 extra LOC.

- TTST yields negligible run-time overhead.

- TTST yields modest virtual memory overhead (2.6 – 4.3x).

- TTST yields modest live update time (0.1 – 1.2s).
Fault Injection

Fault Injection Results (%)

Successful Update
Timeout
Abnormal Termination
State Differences

Back to the Future: Fault-tolerant Live Update with Time-traveling State Transfer

Cristiano Giuffrida
Summary


- Supports several classes of updates with minimal manual effort.

- Automates state transfer and state validation.

- Detects and recovers from arbitrary run-time and memory errors.

- Relies on a minimal amount of trusted code.
Back to the Future: Fault-tolerant Live Update with Time-traveling State Transfer

Cristiano Giuffrida, Călin Iorgulescu, Anton Kuijsten, Andy Tanenbaum
{giuffrida, calin.iorgulescu, kuijsten, ast}@cs.vu.nl

Vrije Universiteit Amsterdam