SAFE SOFTWARE UPDATES VIA MULTI-VERSION EXECUTION

PETR HOSEK
CRISTIAN CADAR

Petr Hosek is a recipient of the Google European Fellowship in Software Engineering and this research is supported in part by this Google Fellowship.
The fundamental problem with program maintenance is that fixing a defect has a substantial (20*-50%) chance of introducing another. So the whole process is two steps forward and one step back.

—F. Brooks, 1975
The Mythical Man-Month

*More than 14.8~24.4% for major operating system patches
Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., and Bairavasundaram, L. How Do Fixes Become Bugs? ESEC/FSE’11
Software updates often present a high risk

Many admins (70%) and users refuse to upgrade software

Reliance on outdated versions flawed with vulnerabilities

Crameri, O., Knezevic, N., Kostic, D., Bianchini, R., Zwaenepoel, W. Staged deployment in Mirage, an integrated software upgrade testing and distribution system. SOSP’07
Real-world example
for (h = 0, i = 0; i < etag->used; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

HTTP ETag hash value computation in etag_mutate
Real-world example

```c
for (h = 0, i = 0; i < etag->used - 1; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

HTTP ETag hash value computation in `etag_mutate`
Real-world example

HTTP ETag hash value computation in `etag_mutate`

```c
for (h = 0, i = 0; i < etag->used - 1; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

Bug diagnosed in issue tracker
Real-world example

HTTP ETag hash value computation in `etag_mutate`

```c
for (h = 0, i = 0; i < etag->used - 1; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

etag_mutate(con->physical.etag, srv->tmp_buf);
```

Bug diagnosed in issue tracker

File (re)compression in `mod_compress_physical`
Real-world example

```
if (use_etag) {
    etag_mutate(con->physical.etag, srv->tmp_buf);
}
```

File (re)compression in `mod_compress_physical`

HTTP ETag hash value computation in `etag_mutate`

for (h = 0, i = 0; i < etag->used - 1; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
if (use_etag) {
    etag_mutate(con->physical.etag, srv->tmp_buf);
}

HTTP ETag hash value computation in `etag_mutate`

for (h = 0, i = 0; i < etag->used - 1; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

File (re)compression in `mod_compress_physical`

Bug diagnosed in issue tracker
Goal

Improve the execution of upgraded software to provide:

Benefits of the newer version
Stability of the older version
Multi-core CPU becoming a standard

Abundance of resources and a high degree of parallelism with no benefit to inherently sequential applications

Cadar, C., Pietzuch P., Wolf, A. L. Multiplicity computing: A vision of software engineering for next-generation computing platform applications. FoSER’10
Multi-version execution based approach

Run the new version in parallel with the existing one
Synchronise the execution of the two versions
Use output of correctly executing version at any given time
Can be extended to work with multiple versions
Synchronisation possible at multiple levels of abstraction:

- Application inputs-outputs
- Function/library calls
- System calls
System calls define **external behaviour**

**VERSION 1**

```c
void fib(int n) {
    int f[n+1];
    for (int i = 3; i <= n; ++i) {
        f[i] = f[i-1] + f[i-2];
    }
    printf("%d\n", f[n]);
}
```

**VERSION 2**

```c
void fib(int n) {
    int a = 1, b = 1;
    for (int i = 3; i <= n; ++i) {
        int c = a + b;
        a = b, b = c;
    }
    printf("%d\n", b);
}
```

`int main(int argc, char **argv)`

```c
{
    fib(5);
    fib(6);
}
```

**Example testing code**
Tested with both implementations
System calls define **external behaviour**

**VERSION 1**

```c
void fib(int n)
{
    int f[n+1];
    for (int i = 3; i <= n; ++i)
        f[i] = f[i-1] + f[i-2];
    printf("%d\n", f[n]);
}
```

... write(1, "5\n", 2) = 2
write(1, "8\n", 2) = 2
...

**Snippet of system call trace**
Obtained using `strace` tool

**VERSION 2**

```c
void fib(int n)
{
    int a = 1, b = 1;
    for (int i = 3; i <= n; ++i) {
        int c = a + b;
        a = b, b = c;
    }
    printf("%d\n", b);
}
```

**Example testing code**
Tested with both implementations
System calls define external behaviour

<table>
<thead>
<tr>
<th>VERSION 1</th>
<th>VERSION 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>void fib(int n)</td>
<td>void fib(int n)</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>int f[n+1];</td>
<td>int a = 1, b = 1;</td>
</tr>
<tr>
<td>f[1] = f[2] = 1;</td>
<td>for (int i = 3; i &lt;= n; ++i) {</td>
</tr>
<tr>
<td>for (int i = 3; i &lt;= n; ++i)</td>
<td>int c = a + b;</td>
</tr>
<tr>
<td>f[i] = f[i-1] + f[i-2];</td>
<td>a = b, b = c;</td>
</tr>
<tr>
<td>printf(&quot;%d\n&quot;, f[n]);</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>printf(&quot;%d\n&quot;, b);</td>
</tr>
</tbody>
</table>

... write(1, “5\n”, 2) = 2
write(1, “8\n”, 2) = 2
...

Snippet of system call trace
Obtained using strace tool

... write(1, “5\n”, 2) = 2
write(1, “8\n”, 2) = 2
...

Snippet of system call trace
Obtained using strace tool

int main(int argc, char **argv)
{
    fib(5);
    fib(6);
}

Example testing code
Tested with both implementations
External behaviour evolves sporadically

95% of revisions introduce no change

Measured using *lighttpd* regression suite on 164 revisions

Taken on Linux kernel 2.6.40 and glibc 2.14 using *strace* tool and custom post-processing (details in the paper)
Synchronisation and fail-recovery strategy
Synchronisation
Compare individual system calls and their arguments

Synchronisation and fail-recovery strategy
Synchronisation

Compare individual system calls and their arguments

GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip

Synchronisation and fail-recovery strategy
Synchronisation
Compare individual system calls and their arguments

Checkpointing
Use clone to take a snapshot of a process

Synchronisation and fail-recovery strategy
Synchronisation and fail-recovery strategy

**Synchronisation**

Compare individual system calls and their arguments

```c
for (h = 0, i = 0; i < etag->used; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

**Checkpointing**

Use `clone` to take a snapshot of a process
GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip

Synchronisation
Compare individual system calls and their arguments

for (h = 0, i = 0; i < etag->used; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

Checkpointing
Use `clone` to take a snapshot of a process

for (h = 0, i = 0; i < etag->used - 1; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

Crash
Segmentation fault

Synchronisation and fail-recovery strategy
Synchronisation and fail-recovery strategy

Synchronisation
Compare individual system calls and their arguments

for (h = 0, i = 0; i < etag->used; ++i)
h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

Failure recovery
Restart the snapshot and replace the code with the code of the new version

for (h = 0, i = 0; i < etag->used; ++i)
h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

Checkpointing
Use clone to take a snapshot of a process

for (h = 0, i = 0; i < etag->used - 1; ++i)
h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);

Crash
Segmentation fault
Synchronisation and fail-recovery strategy

Synchronisation
Compare individual system calls and their arguments

```c
for (h = 0, i = 0; i < etag->used; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

Failure recovery
Restart the snapshot and replace the code with the code of the new version

Checkpointering
Use `clone` to take a snapshot of a process

```c
for (h = 0, i = 0; i < etag->used - 1; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

Crash
Segmentation fault

```c
for (h = 0, i = 0; i < etag->used; ++i)
    h = (h << 5) ^ (h >> 27) ^ (etag->ptr[i]);
```

Reconvergence
Return to the original code and continue execution
Synchronisation
Compare individual system calls and their arguments

Failure recovery
Restart the snapshot and replace the code with the code of the new version

Checkpointing
Use clone to take a snapshot of a process

Crash
Segmentation fault

Reconvergence
Return to the original code and continue execution

Synchronisation and fail-recovery strategy
Recover crashing version using the state of the other one:

Assumes small bug propagation distance

Crashes are the only type of observable divergences

The non-crashing version used as an oracle
Guarantees

Recovery is successful if versions exhibit the same externally observable behaviour after recovery:

- If unrecoverable, continue with the non-crashing version
- Do not attempt to survive errors we cannot handle
Suitable for type of changes and applications:

Changes which do not affect memory layout

  e.g., refactorings, security patches

Applications which provide synchronisation points

  e.g., servers structured around the main dispatch loop

Where reliability is more important than performance

  e.g., interactive apps, some server scenarios
Mx architecture

- Multi-Version Application
- Static Analysis
- System Call Interposition
- Runtime Manipulation
- Conventional Application
- Linux Kernel
Implementation for x86 and x86-64 Linux

Combines binary static analysis, lightweight checkpointing and runtime code patching

Completely transparent, runs on unmodified binaries

Runs two versions with small differences in behaviour

Focus on application crashes and recovery
Multi-eXecution Monitor

Execute and monitor multi-version applications:
Intercepting system calls (via `ptrace` interface)
Semantically comparing system calls arguments
Environment virtualisation (e.g. files and sockets)
Runtime Execution Manipulator

Runtime code patching and fault recovery:

OS-level checkpointing (using `clone` syscall)
Runtime stack rewriting (`libunwind`)
Breakpoint insertion and handling
Static Executable Analyser

Create various mappings between the two version binaries:

- Extracting function symbols from binaries (libbfd)
- Machine code disassembling and analysis (libopcodes)
- Binary call graph reconstruction and matching
Execution stack rewriting
Execution stack rewriting
Execution stack rewriting
VERSION 1

0xdeadbeef <foo>:

f59: callq 0xdeadcafe <bar>

0xdeadcafe <bar>:

b07: mov -0x40(%rbp),%rax
b0a: callq *%rax

%rsp
0xdeadbf5e

VERSION 2'

0xdeadbeef <foo>:

f59: callq 0xdeadcafe <bar>

0xdeadcafe <bar>:

b07: mov -0x40(%rbp),%rax
b0a: callq *%rax

%rsp
0xdeadbf5e

Execution stack rewriting
Execution stack rewriting
Redis regression bug #344 introduced during refactoring
HMGET command implementation in hmgetCommand function

Survived a number of crash bugs in several popular server applications

In-memory NoSQL database
Redis regression bug #344 introduced during refactoring

HMGET command implementation in hmgetCommand function
## Interactive applications:

<table>
<thead>
<tr>
<th>UTILITY</th>
<th>BUG</th>
<th>TIME SPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5sum</td>
<td>Buffer underflow</td>
<td>1,124 revs. (1 year 7 months)</td>
</tr>
<tr>
<td>sha1sum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mkdir</td>
<td>NULL-pointer dereference</td>
<td>2,937 revs. (over 4 years)</td>
</tr>
<tr>
<td>mkfifo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mknod</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cut</td>
<td>Buffer overflow</td>
<td>1,201 revs. (2 years 3 months)</td>
</tr>
</tbody>
</table>

## Server applications:

<table>
<thead>
<tr>
<th>APPLICATION/ISSUE</th>
<th>BUG</th>
<th>TIME SPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>lighttpd #2169</td>
<td>Loop index underflow</td>
<td>87 revs. (2 months 2 days)</td>
</tr>
<tr>
<td>lighttpd #2140</td>
<td>Off-by-one error</td>
<td>12 revs. (2 months 1 day)</td>
</tr>
<tr>
<td>redis #344</td>
<td>Missing <strong>return</strong> statement</td>
<td>27 revs. (6 days)</td>
</tr>
</tbody>
</table>
17.91% overhead on SPEC CPU2006 over single version despite 2x utilisation cost

Measured using SPEC CPU2006 1.2
Taken on 3.50 GHz Intel Xeon E3 1280 with 16 GB of RAM, Linux kernel 3.1.9
Interactive applications:

<table>
<thead>
<tr>
<th>UTILITY</th>
<th>INPUT SIZE</th>
<th>OVERHEAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5sum</td>
<td>&lt;1.25MB</td>
<td>&lt;100ms (imperceptible)</td>
</tr>
<tr>
<td>sha1sum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mkdir</td>
<td>&lt;115 nested directories</td>
<td>&lt;100ms (imperceptible)</td>
</tr>
<tr>
<td>mkfifo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mknod</td>
<td>&lt;1.10MB</td>
<td>&lt;100ms (imperceptible)</td>
</tr>
<tr>
<td>cut</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Measured using Coreutils 6.10
Taken on 3.50 GHz Intel Xeon E3 1280 with 16 GB of RAM, Linux kernel 3.1.9

Server applications:

<table>
<thead>
<tr>
<th>APPLICATION</th>
<th>SCENARIO</th>
<th>OVERHEAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>lighttpd</td>
<td>localhost/network</td>
<td>2.60x – 3.49x</td>
</tr>
<tr>
<td></td>
<td>distant networks</td>
<td>1.01x – 1.04x</td>
</tr>
<tr>
<td>redis</td>
<td>localhost/network</td>
<td>3.74x – 16.72x</td>
</tr>
<tr>
<td></td>
<td>distant networks</td>
<td>1.00x – 1.05x</td>
</tr>
</tbody>
</table>

Measured using redis-benchmark and http_load
Taken on 3.50 GHz Intel Xeon E3 1280 with 16 GB of RAM, Linux kernel 3.1.9
Summary

Novel approach for improving software updates:
  Based on multi-version execution
  Mx can survive crash bugs in real apps

Many opportunities for future work:
  Better performance overhead
  Tolerance to system call divergencies
  Support for more complex code changes
  Support for non-crashing type of divergences