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.
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
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]);
```

**HTTP ETag** hash value computation in `etag_mutate`
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
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]);
```

File (re)compression in `mod_compress_physical`

Bug diagnosed in issue tracker

`etag_mutate(con->physical.etag, srv->tmp_buf);`
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
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
A year ago in a city far far away...

Introducing novel approach for improving software updates:

Multi-version execution based approach
Relying on abundance of resources to improve reliability
Run the new version in parallel with the existing one
Synchronise the execution of the versions
Use output of correctly executing version
Synchronisation and fail-recovery mechanism
Synchronisation
Compare individual system calls and their arguments

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

Synchronisation and fail-recovery mechanism

GET /index.html HTTP/1.1
Host: srg.doc.ic.ac.uk
Accept-Encoding: gzip
Synchronisation
Compare individual system calls and their arguments

Checkpointing
Use clone to take a snapshot of a process

Synchronisation and fail-recovery mechanism
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

Synchronisation and fail-recovery mechanism
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

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

Synchronisation and fail-recovery mechanism

Crash

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

Synchronisation
Compare individual system calls and their arguments

for (h = 0, i = 0; i < etag->used - 1; ++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

Synchronisation and fail-recovery mechanism

Checkpointing
Use clone to take a snapshot of a process

Crash
Segmentation fault

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

Checkpointing

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]);
```

Synchronisation and fail-recovery mechanism

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 and fail-recovery mechanism

Synchronisation
Compare individual system calls and their arguments

\[ \text{for } (h = 0, i = 0; i < \text{etag-used}; ++i) \]
\[ h = (h << 5) \land (h >> 27) \land \text{etag.ptr}[i]; \]

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

\[ \text{for } (h = 0, i = 0; i < \text{etag-used}; ++i) \]
\[ h = (h << 5) \land (h >> 27) \land \text{etag.ptr}[i]; \]

Checkpointing
Use clone to take a snapshot of a process

\[ \text{for } (h = 0, i = 0; i < \text{etag-used - 1}; ++i) \]
\[ h = (h << 5) \land (h >> 27) \land \text{etag.ptr}[i]; \]

Crash
Segmentation fault

\[ \text{for } (h = 0, i = 0; i < \text{etag-used}; ++i) \]
\[ h = (h << 5) \land (h >> 27) \land \text{etag.ptr}[i]; \]

Reconvergence
Return to the original code and continue execution

\[ \text{for } (h = 0, i = 0; i < \text{etag-used}; ++i) \]
\[ h = (h << 5) \land (h >> 27) \land \text{etag.ptr}[i]; \]
Assumptions

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

- Assumes small bug propagation distance
- Crashes are the only type of observable divergences
- The non-crashing version used as an oracle
- If unrecoverable, continue with the non-crashing version
Synchronisation possible at multiple levels of abstraction

Total Synchronisation

Uncoordinated Execution
System Calls

Synchronisation possible at multiple levels of abstraction

Total Synchronisation

Uncoordinated Execution
Synchronisation possible at multiple levels of abstraction
Synchronisation possible at multiple levels of abstraction
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);
}
```

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

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

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

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

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

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

```c
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)
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
VERSION 1

0xdeadbeef <foo>:

f59: callq 0xdeadcafe <bar>

0xdeadcafe <bar>:

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

Snippet of instruction code

%rsp

0xdeadbf5e

Execution stack

VERSION 2

0xdeadbef3 <foo>:

f5e: callq 0xdeadcaff <bar>

0xdeadcaff <bar>:

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

Snippet of instruction code

%rsp

0xdeadbf64

Execution stack
### VERSION 1

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xdeadbeef &lt;foo&gt;:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f59:</td>
<td>callq 0xdeadcafe &lt;bar&gt;</td>
<td></td>
</tr>
<tr>
<td>0xdeadcafe &lt;bar&gt;:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b07:</td>
<td>mov -0x40(%rbp),%rax</td>
<td></td>
</tr>
<tr>
<td>b0a:</td>
<td>callq *%rax</td>
<td></td>
</tr>
</tbody>
</table>

#### Snippet of instruction code

- `%rsp`
- `0xdeadbf5e`

#### Execution stack

### VERSION 2'

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xdeadbeef &lt;foo&gt;:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f59:</td>
<td>callq 0xdeadcafe &lt;bar&gt;</td>
<td></td>
</tr>
<tr>
<td>0xdeadcafe &lt;bar&gt;:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b07:</td>
<td>mov -0x40(%rbp),%rax</td>
<td></td>
</tr>
<tr>
<td>b0a:</td>
<td>callq *%rax</td>
<td></td>
</tr>
</tbody>
</table>

#### Snippet of instruction code

- `%rsp`
- `0xdeadbf64`

#### Execution stack
VERSION 1

0xdeadbeef <foo>:

f59:  callq  0xdeadcafe <bar>

0xdeadcafe <bar>:

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

Snippet of instruction code

%rsp

0xdeadbf5e

Execution stack

VERSION 2'

0xdeadbeef <foo>:

f59:  callq  0xdeadcafe <bar>

0xdeadcafe <bar>:

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

Snippet of instruction code

%rsp

0xdeadbf5e

Execution stack
<table>
<thead>
<tr>
<th>VERSION 1</th>
<th>VERSION 2'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xdeadbeef &lt;foo&gt;:</td>
<td>0xdeadbeef &lt;foo&gt;:</td>
</tr>
<tr>
<td>f59: callq 0xdeadcafe &lt;bar&gt;</td>
<td>f59: callq 0xdeadcafe &lt;bar&gt;</td>
</tr>
<tr>
<td>0xdeadcafe &lt;bar&gt;:</td>
<td>0xdeadcafe &lt;bar&gt;:</td>
</tr>
<tr>
<td>b07: mov -0x40(%rbp),%rax</td>
<td>b07: mov -0x40(%rbp),%rax</td>
</tr>
<tr>
<td>b0a: callq *%rax</td>
<td>b0a: callq *%rax</td>
</tr>
</tbody>
</table>

Snippet of instruction code

%rsp

0xdeadbf5e

Execution stack

Snippet of instruction code

%rsp

0xdeadbf5e

Execution stack
**VERSION 1**

0xdeadbeef <foo>:

f59: callq 0xdeadcafe <bar>

0xdeadcafe <bar>:

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

**Snippet of instruction code**

%rsp

0xdeadbf5e

**Execution stack**

**VERSION 2’**

0xdeadbeef <foo>:

f59: callq 0xdeadcafe <bar>

0xdeadcafe <bar>:

aff: int $3

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

**Snippet of instruction code**

%rsp

0xdeadbf5e

**Execution stack**
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
Survived a number of crash bugs in several popular server applications

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 return 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></td>
<td></td>
</tr>
<tr>
<td>cut</td>
<td>&lt;1.10MB</td>
<td>&lt;100ms (imperceptible)</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
“The New Mx”

Better performance overhead:
   System call binary rewriting

Tolerance to system call divergences:
   Event streaming
read(6, “PING\r\n”, 1024)

Snippet of system call trace
### REDIS

--- SIGTRAP ---

```c
getpid()
```

--- SIGTRAP ---

**Snippet of system call trace**

### MX

```c
ptrace(PTRACE_GETREGS, 7, {...}, NULL)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
```

```c
read(8, “PING\r\n”, 1024)
```

```c
ptrace(PTRACE_GETREGS, 7, {...}, NULL)
process_vm_writev(7, {...}, 1, {...}, 1, 0)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
```

**Snippet of system call trace**
Snippet of system call trace

VMA

--- SIGTRAP ---

getpid()

--- SIGTRAP ---

Snippet of system call trace

VMA

REDIS

--- SIGTRAP ---

MX

ptrace(PTRACE_GETREGS, 7, {...}, NULL)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
read(8, "PING\r\n", 1024)
ptrace(PTRACE_GETREGS, 7, {...}, NULL)
process_vm_writev(7, {...}, 1, {...}, 1, 0)
ptrace(PTRACE_SETREGS, 7, {...}, {...})
ptrace(PTRACE_SYSCALL, 7, {...}, NULL)
## Snippet of instruction code

### GLIBC

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xdeadbeef</td>
<td><code>mov  $0x0,%eax</code></td>
<td></td>
</tr>
<tr>
<td>0xdeadbeef</td>
<td><code>syscall</code></td>
<td></td>
</tr>
</tbody>
</table>

### REDIS

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4050f0</td>
<td><code>callq &lt;read@plt&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>
Snippet of instruction code

### GLIBC

0xdeadbeef <__libc_read>:

2a:  jmpq  $0x13cd0

### VMA

0x4050f0 <anetRead>:

405130:  callq  <read@plt>

### NX

0x13cd0 <syscall_enter>:

13d31:  cmp  $0x1,%r10

13d3a:  callq  *%r10

Snippet of instruction code
System call synchronisation possible at different phases
Every system call

System call synchronisation possible at different phases
System call synchronisation possible at different phases
Every system call

System call synchronisation possible at different phases

Record-replay

Event streaming
APPLICATION

LEADER

FOLLOWER

Event log

En
En-1
En-2
En-3
En-4
En-5
En-6
En-7
En-8
En-9
En-10
En-11
En-12
En-13
En-14

APPLICATION

Event log
Event log
Event log

APPLICATION

FOLLOWER

APPLICATION

FOLLOWER

APPLICATION

LEADER
Future Work

Support for more complex code changes:
- Data structure inference & excavation
- Control flow graph isomorphisms
- Call stack reconstruction

Support for non-crashing type of divergences:
- Infinite loops and deadlocks
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
Distinct code bases, manually-generated

N-version programming: A fault-tolerance approach to reliability of software operation

Chen, L., and Avizienis, A. FTCS’78

Using replicated execution for a more secure and reliable web browser

Xue, H., Dautenhahn, N., and King, S. T. NDSS’12

Variants of the same code, automatically generated

N-variant systems: a secretless framework for security through diversity

Cox, B., Evans, D., Filipi, A., Rowanhill, J., Hu, W., Davidson, J., Knight, J., Nguyen-Tuong, A., and Hiser, J. USENIX Security’06

Run-time defense against code injection attacks using replicated execution


Online validation of different manually-evolved versions

Efficient online validation with delta execution

Tucek, J., Xiong, W., Zhou, Y. ASPLOS’09

Tachyon: Tandem Execution for Efficient Live Patch Testing

Maurer, M., Brumley, D. USENIX Security’12