Loupe: Designing Application-driven Compatibility Layers in Custom Operating Systems

Pierre Olivier, The University of Manchester
pierre.olivier@manchester.ac.uk

Joint work with Hugo Lefeuvre¹, Gaulthier Gain², Vlad-Andrei Bădoiu³, Daniel Dinca³, Vlad-Radu Schiller¹, Costin Raiciu³, and Felipe Huici⁴
¹The University of Manchester, ²University of Liège, ³Politehnica University of Bucharest, ⁴Unikraft.io
Custom Oses & Compatibility

► We still need custom (research/prototype) Oses
We still need custom (research/prototype) Oses
These are only as good/popular as the applications they can run
Custom Oses & Compatibility

- We still need custom (research/prototype) Oses
- These are only as good/popular as the applications they can run
- **Compatibility** with existing applications is key
  - To build a community
  - To attract potential sponsors/investors
  - To gather early numbers
  - etc.
How is Compatibility Achieved?

- Porting is not sustainable
How is Compatibility Achieved?

- Porting is not sustainable
- **Transparent compatibility:** emulate a popular OS e.g. Linux
  - Source level
  - Binary Libc level
  - Binary system call level
Compatibility Seemingly Takes Effort

- Linux has 360+ system calls
- Some are vectored (e.g. ioctl)
- Beyond system call: virtual filesystems (/proc, etc.)
- Hinders the development of custom Oses
Compatibility Seemingly Takes Effort

- Linux has 360+ system calls
- Some are vectored (e.g. `ioctl`)
- Beyond system call: virtual filesystems (`/proc`, etc.)
- Hinders the development of custom Oses

1000+ papers in SOSP/OSDI/ASPLOS/EuroSys over the last 10Y
Building Compatibility Layers is an Ad-hoc and Unoptimized Process

- Undertaken by several projects
  - OSv, Graphene, HermiTux, Unikraft, Zephyr, Fuchsia, Browsix, Kerla, etc.
Building Compatibility Layers is an Ad-hoc and Unoptimized Process

- Undertaken by several projects
  - Osv, Graphene, HermiTux, Unikraft, Zephyr, Fuchsia, Browsix, Kerla, etc.

- Application-driven, organic process:
  - Take an app, try to run it, it fails, implemente the needed OS feature, rinse and repeat
Building Compatibility Layers is an Ad-hoc and Unoptimized Process

- Undertaken by several projects
  - Osv, Graphene, HermiTux, Unikraft, Zephyr, Fuchsia, Browsix, Kerla, etc.

- Application-driven, organic process:
  - Take an app, try to run it, it fails, implemente the needed OS feature, rince and repeat

- Most of that implementation is OS-specific
Building Compatibility Layers is an Ad-hoc and Unoptimized Process

- Undertaken by several projects
  - Osv, Graphene, HermiTux, Unikraft, Zephyr, Fuchsia, Browsix, Kerla, etc.

- Application-driven, organic process:
  - Take an app, try to run it, it fails, implement the needed OS feature, rince and repeat

- Most of that implementation is OS-specific

- How can we optimize it?
Static analysis?

Intuitively a good solution because it is comprehensive
Static analysis?

Intuitively a good solution because it is comprehensive

This paper yields several insights for developers and researchers, which are useful for assessing the complexity and security of Linux APIs. For example, every Ubuntu installation requires 224 system calls, 208 ioctl, fcntl, and prctl codes and hundreds of pseudo files. For each API

Tsai et al., A Study of Modern Linux API Usage and Compatibility: What to Support When You’re Supporting, EuroSys’16 Best Paper Award
Static analysis?

Ituitively a good solution because it is comprehensive

This paper yields several insights for developers and researchers, which are useful for assessing the complexity and security of Linux APIs. For example, every Ubuntu installation requires 224 system calls, 208 ioctl, fcntl, and prctl codes and hundreds of pseudo files. For each API

Tsai et al., A Study of Modern Linux API Usage and Compatibility: What to Support When You’re Supporting, EuroSys’16 Best Paper Award

But do we need full compatilibility?
Or even 100% stability?
Dynamic analysis

strace(1) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | OPTIONS | DIAGNOSTICS | SETUID INSTALLATION | MULTIPLE PERSONALITIES SUPPORT | NOTES | BUGS | HISTORY | REPORTING BUGS | SEE ALSO | AUTHORS | COLOPHON

NAME

strace - trace system calls and signals

SYNOPSIS

strace [ -ACdfhhikqrtttTvWwxxyyZ ] [ -I n ] [ -b execve ]
[ -e expr ] ... [ -O overhead ] [ -S sortby ] [ -U columns ]
[ -a column ] [ -o file ] [ -s ssize ] [ -X format ]
[ -P path ] ... [ -p pid ] ... [ --seccomp-bpf ]
[ --secontext=[format] ] { -p pid | [ -D DD ] [ -E var[=val] ] ... }
Dynamic analysis

strace(1) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | OPTIONS | DIAGNOSTICS | SETUID INSTALLATION | MULTIPLE PERSONALITIES SUPPORT | NOTES | BUGS | HISTORY | REPORTING BUGS | SEE ALSO | AUTHORS | COPHON

strace - trace system calls and signals

System Call Stubbing/Faking

```c
#include <hermit/syscall.h>
#include <hermit/stdlib.h>

/* TODO */

int sys_mincore(unsigned long start, size_t len, unsigned char *vec) {
    return -ENOSYS;
}
```
System Call Stubbing/Faking

```c
#include <hermit/sySCALL.h>
#include <hermit/stdio.h>

/* TODO */
int sys_mincore(unsigned long start, size_t len, unsigned char *vec) {
    return -ENOSYS;
}
```

```c
#include <hermit/sySCALL.h>
#include <hermit/logging.h>

int sys_chdir(const char *path) {
    LOG_WARNING("chdir not implemented, faking success\n");
    return 0;
}
```
System Call Support Landscape
System Call Support Landscape

Can we measure that?
Loupe

Super-strace measuring the system calls required to run an application, checking which ones can be faked/stubbed
Loupe

- Super-strace measuring the system calls required to run an application, checking which ones can be faked/stubbed
- Used to build a database of apps measurements
Loupe

- Super-strace measuring the system calls required to run an application, checking which ones can be faked/stubbed
- Used to build a database of apps measurements
- Can derive **support plans** for custom Oses
  - For a set of target apps to support and a set of already-implemented system calls, **what is the optimized order of system calls to implement to support as many apps as soon as possible**
Loupe from the user point of view

Dockerfile: how to build and run the app under test

Input workload (shell script)

Loupe
Loupe from the user point of view

- Dockerfile: how to build and run the app under test
- Input workload (shell script)
- App running
- Loupe
- Linux Kernel
Loupe from the user point of view

Dockerfile: how to build and run the app under test

Input workload (shell script)

App running

Loupe

Linux Kernel

Results: For each syscall s of the Linux API, does the app still work if s is stubbed/faked/both?
Dockerfile: how to build and run the app under test

Input workload (shell script)

Loupe

App running

Linux Kernel

Results:
For each syscall s of the Linux API, does the app still work if s is stubbed/faked/both

Support Plan

OS Profile

LoupeDB
How does it Works?

1) Determine all system calls done by the app processing the workload with a quick pass of strace
How does it Works?

1) Determine all system calls done by the app processing the workload with a quick pass of strace

2) For each system call identified, hook into system calls invocations with seccomp, emulate
   - Stubbing: return -ENOSYS
   - Faking: return 0

And check if the app/workload succeeds
How to check for success?

2 types of apps:

**Run-to-completion** (e.g. fio)

- Run the app instrumented with loupe, then check its exit code
- Optionally run a script after each run for additional checks (stdout, files created, etc.)
How to check for success?

2 types of apps:

Run-to-completion (e.g. fio)
- Run the app instrumented with loupe, then check its exit code
- Optionally run a script after each run for additional checks (stdout, files created, etc.)

Client/Server (e.g. nginx)
- Run the app and check that it does not crash
- Concurrently run a workload script (e.g. wrk) and check for its successful execution too
What Syscalls to (Really) Implement?
- Static analysis highly overestimate the engineering effort for supporting an app
- Naive (strace) dynamic analysis also measures much more syscalls that what is actually required
What Syscalls to (Really) Implement?

(a) Static analysis, binary.
What Syscalls to (Really) Implement?

(a) Static analysis, binary.
(b) Static analysis, source.
(c) Dynamic analysis, executed.
(d) Dynamic analysis, required.
Why does Stubbing/Faking Work?

```c
if (getrlimit(RLIMIT_NOFILE,&limit) == -1) {
    serverLog
    (LL_WARNING,"Unable to obtain the current NOFILE" "limit
    (%s), assuming 1024 and setting the max clients" "configuration accordingly.", strerror(errno));
    server.maxclients = 1024-CONFIG_MIN_RESERVED_FDS;
}
```

getrlimit@Redis
Why does Stubbing/Faking Work?

Systems calls for which the return value is commonly not checked:
- close
- munmap
- sched_yield
- exit
- etc.

Figure 8. Apps checking system calls return values.
Long-Term Support?

**Figure 9.** System call usage and capacity to be stubbed/faked for recent (2021) and older (2005-2010) applications releases.
## Examples of Support Plans

<table>
<thead>
<tr>
<th>Step</th>
<th>Implement</th>
<th>Stub</th>
<th>Fake</th>
<th>Apps supported</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unikraft</strong> (commit 7d6797f, supports 174 syscalls)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(12 apps)</td>
</tr>
<tr>
<td>1</td>
<td>290</td>
<td>273, 218, 230</td>
<td>-</td>
<td>+ Memcached</td>
</tr>
<tr>
<td>2</td>
<td>218</td>
<td>-</td>
<td>-</td>
<td>+ H2O</td>
</tr>
<tr>
<td>3</td>
<td>283, 27</td>
<td>186</td>
<td>-</td>
<td>+ MongoDB</td>
</tr>
<tr>
<td><strong>Fuchsia</strong> (commit 5d20758, supports 152 syscalls)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(11 apps)</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>99, 222, 223</td>
<td>-</td>
<td>+ HAPrxy</td>
</tr>
<tr>
<td>2</td>
<td>302</td>
<td>273, 230, 105</td>
<td>-</td>
<td>+ Memcached</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>-</td>
<td>-</td>
<td>+ Lighttpd</td>
</tr>
<tr>
<td>4</td>
<td>128, 99, 27</td>
<td>-</td>
<td>-</td>
<td>+ MongoDB</td>
</tr>
<tr>
<td><strong>Kerla</strong> (commit 73a1873, supports 58 syscalls)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(4 apps)</td>
</tr>
<tr>
<td>1</td>
<td>56, 257, 54</td>
<td>(17 system calls)</td>
<td>47</td>
<td>+ Httpd</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>+ Weborf</td>
</tr>
<tr>
<td>3</td>
<td>232, 233, 302</td>
<td>(9 system calls)</td>
<td>213</td>
<td>+ HAPrxy</td>
</tr>
<tr>
<td>4</td>
<td>17, 18, 53</td>
<td>96, 40, 201, 105, 106, 116</td>
<td>290</td>
<td>+ Nginx</td>
</tr>
<tr>
<td>5</td>
<td>213, 262</td>
<td>95</td>
<td>-</td>
<td>+ Redis</td>
</tr>
<tr>
<td>6</td>
<td>291</td>
<td>293</td>
<td>-</td>
<td>+ Lighttpd</td>
</tr>
<tr>
<td>7</td>
<td>288, 290</td>
<td>32, 87</td>
<td>-</td>
<td>+ H2O</td>
</tr>
<tr>
<td>8</td>
<td>46</td>
<td>230</td>
<td>-</td>
<td>+ Memcached</td>
</tr>
<tr>
<td>9</td>
<td>8, 21, 87</td>
<td>-</td>
<td>25</td>
<td>+ SQLite</td>
</tr>
<tr>
<td>10</td>
<td>104, 107, 108, 102</td>
<td>-</td>
<td>-</td>
<td>+ Webfsd</td>
</tr>
<tr>
<td>11</td>
<td>128, 99, 229, 27, 73, 292, 283</td>
<td>131</td>
<td>137</td>
<td>+ MongoDB</td>
</tr>
</tbody>
</table>
Features in Development

▷ Fine-grained measurement
  - e.g. mmap's MAP_ANONYMOUS, IOCTLs
  - Virtual filesystems
    • /proc
    • /dev
Conclusion

▶ Building compatibility layers is important for many custom Oses
  - It is generally seen as a huge effort
▶ Ad-hoc, organic process that could be optimized
▶ Loupe streamline that process by measuring exactly what system calls need to be implemented for a given app/workload