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 Lefeuvre[1], Gaulthier Gain[2], Vlad-Andrei Bădoiu[3], Daniel Dinca[3], Vlad-Radu Schiller[1], Costin Raiciu[3], and Felipe Huici[4]
[1]The University of Manchester, [2]University of Liège, [3]Politehnica University of Bucharest, [4]Unikraft.io

Custom Oses & Compatibility

- We still need custom (research/prototype) Oses

Custom Oses & Compatibility

- 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?

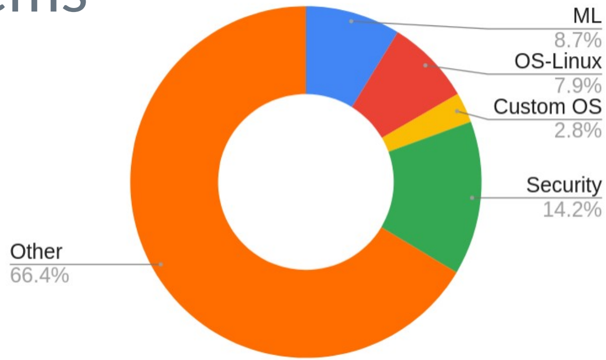
- Porting is not sustainable
- **Transparent compatibility:** emulate a popular OS e.g. Linux
  - Source level
  - Binary Libc level
  - **Binary system call level**

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, rince 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, implemente the needed OS feature, rince and repeat
- Most of that implementation is OS-specific
- How can we optimize it?

Static analysis?

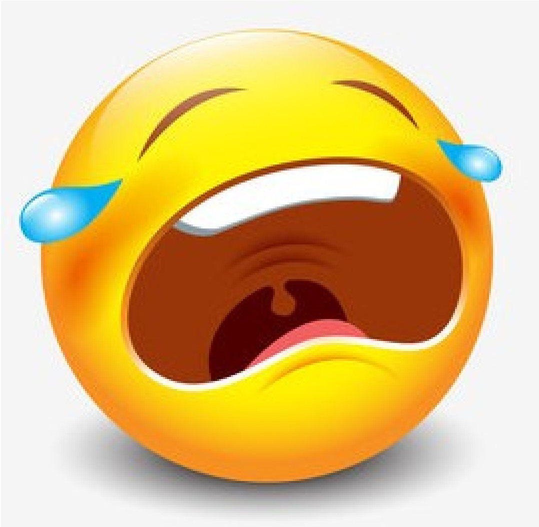
Ituitively a good solution because it is comprehensive

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

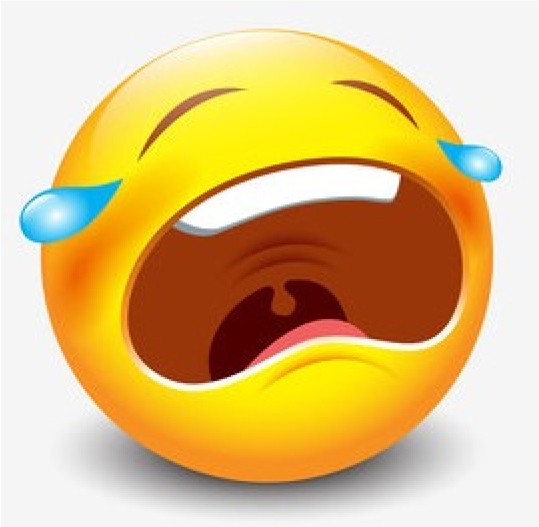


Static analysis?

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 compatilibity?
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

Search online pages

```
STRACE(1)                General Commands Manual               STRACE(1)
```

NAME top

```
       strace - trace system calls and signals
```

SYNOPSIS top

```
       strace [-ACdffhikqqrtttTvVwxxyyzZ] [-I n] [-b execve]
              [-e expr]... [-O overhead] [-S sortby] [-U columns]
              [-a column] [-o file] [-s strsize] [-X format]
              [-P path]... [-p pid]... [--seccomp-bpf]
              [--secontext[=format]] { -p pid | [-DDD] [-E var[=val]]...
              [-u username] command [args] }
```

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

Search online pages

```
STRACE(1)                General Commands Manual                STRACE(1)
```

NAME top

```
       strace - trace system calls and signals
```

SYNOPSIS top

```
       strace [-ACdffhikqqrtttTvVwxxyyzZ] [-I n] [-b execve]
              [-e expr]... [-O overhead] [-S sortby] [-U columns]
              [-a column] [-o file] [-s strsize] [-X format]
              [-P path]... [-p pid]... [--seccomp-bpf]
              [--secontext[=format]] { -p pid | [-DDD] [-E var[=val]]...
```

- strace is still not a panacea

System Call Stubbing/Faking

7 lines (6 sloc) | 157 Bytes

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

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

System Call Stubbing/Faking

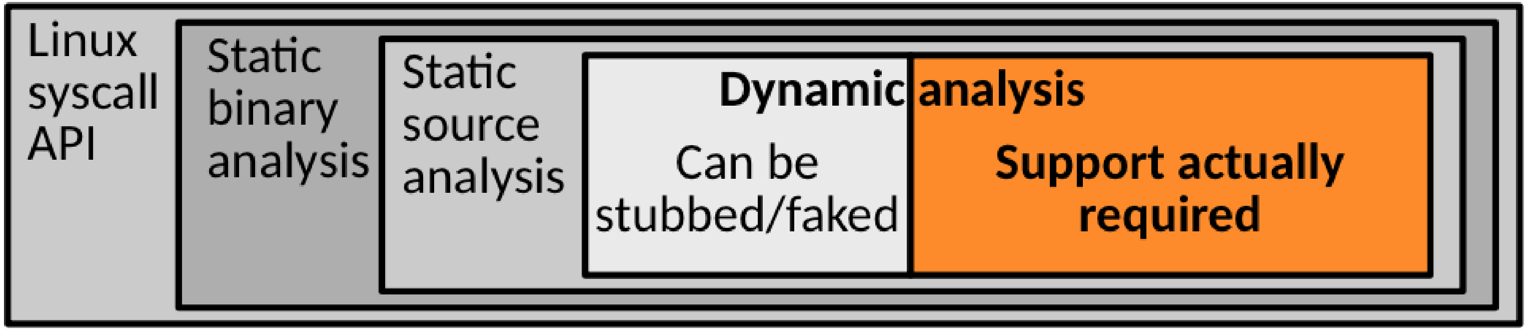
7 lines (6 sloc) | 157 Bytes

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

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

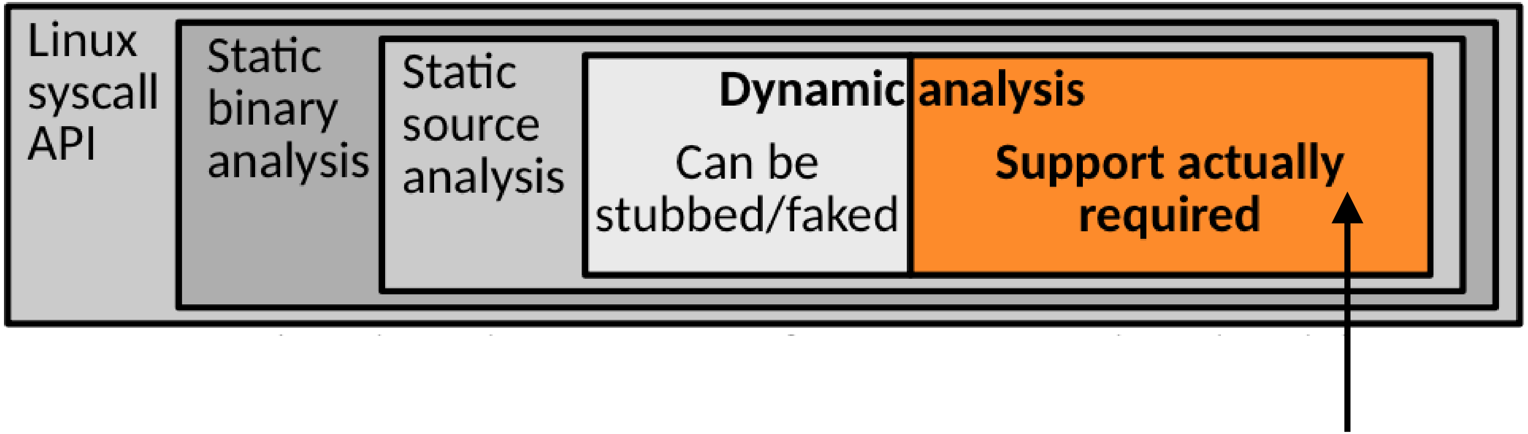
7 lines (6 sloc) | 161 Bytes

```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

Loupe

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

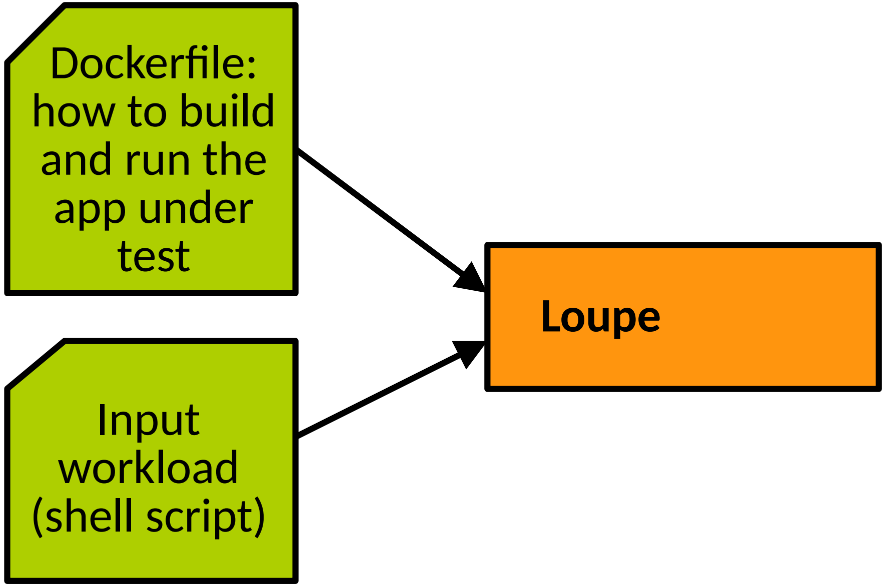
Loupe

- 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

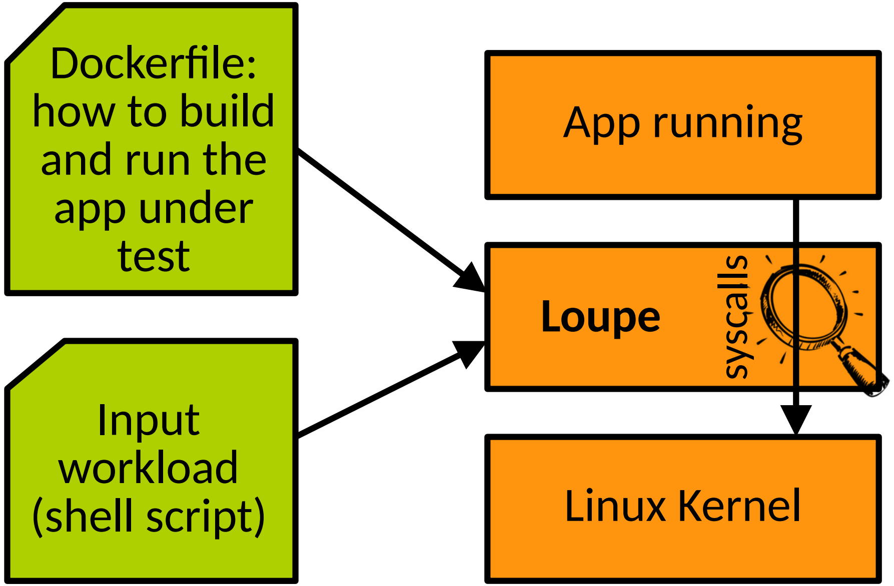
Loupe

- 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**

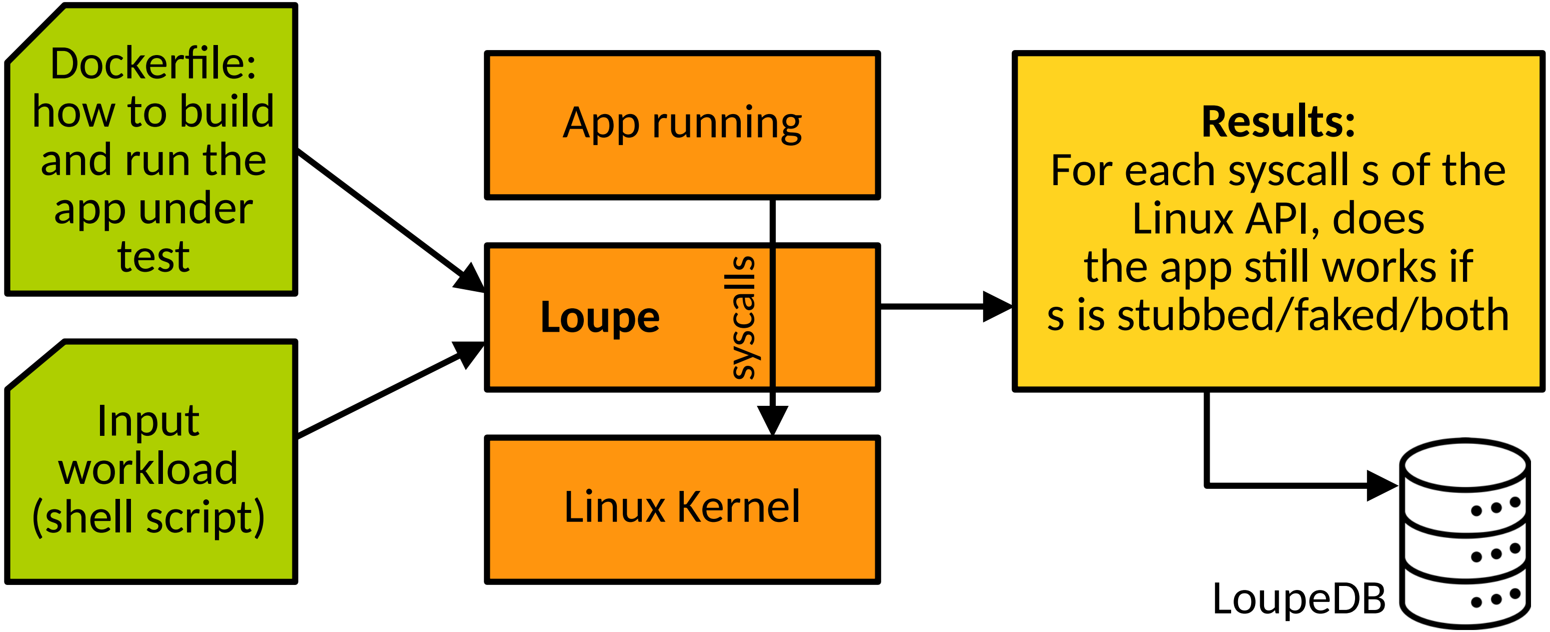
Loupe from the user point of view



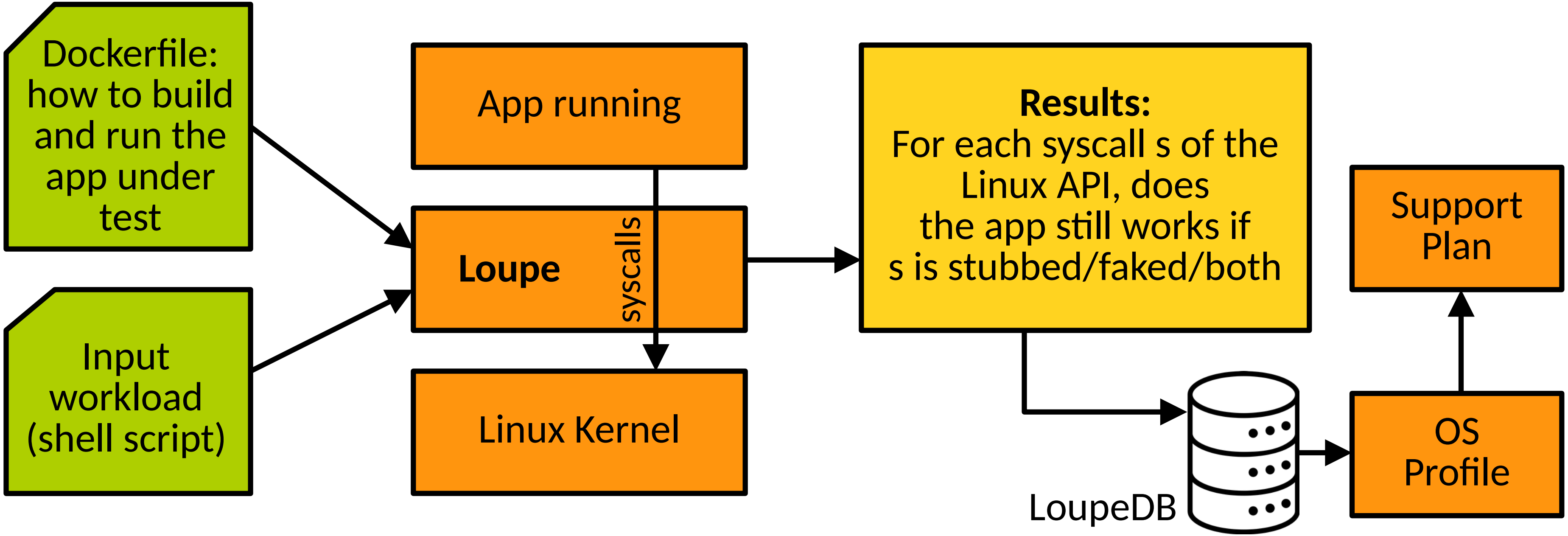
Loupe from the user point of view



Loupe from the user point of view



Loupe from the user point of view



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)

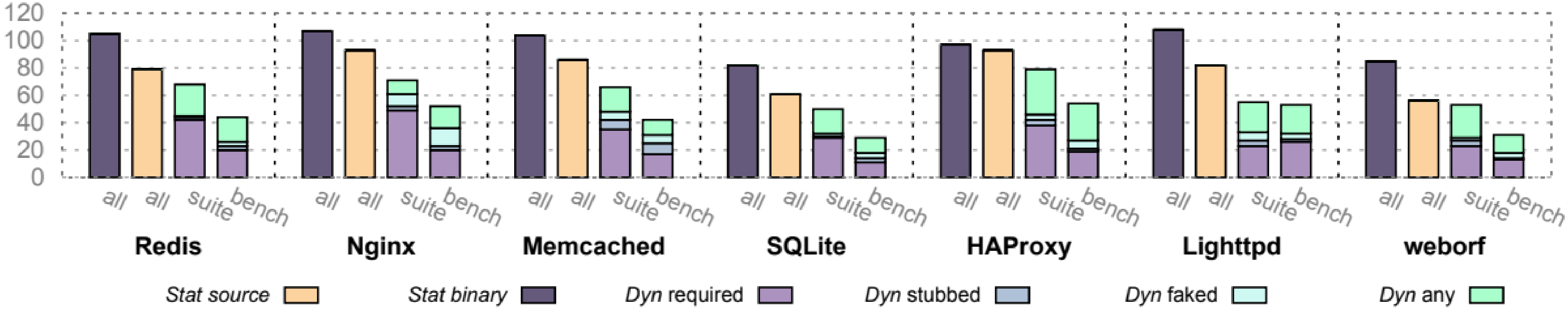
- Run the app instrumented with loupe, then check its exit code
- Optionally run a script after each run for additional checks (stdout, files created, etc.)

Client/Server (e.g. nginx)

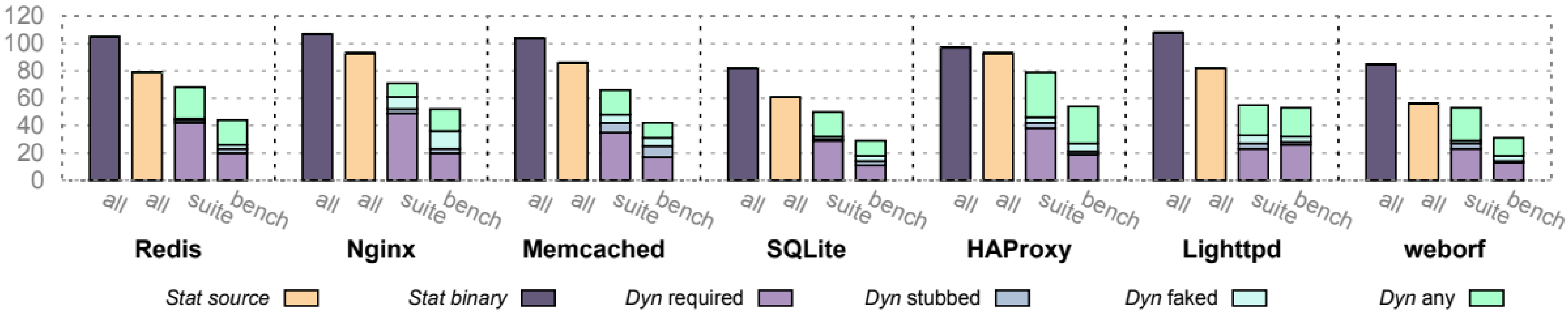
- Run the app and check that it does not crash
- Concurrently run a workload script (e.g. wrk) and check for its successful execution too

Results Analysis

What Syscalls to (Really) Implement?

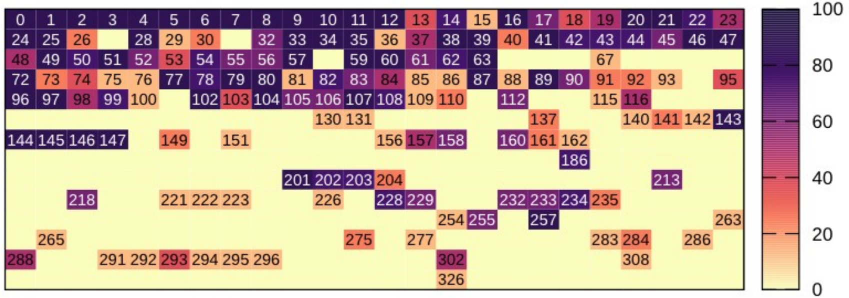


What Syscalls to (Really) Implement?



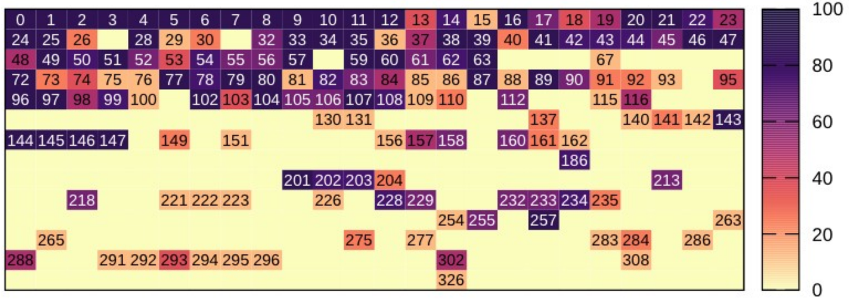
- Static analysis highly overestimate the engineering effort for supporting an app
- Naive (`strace`) dynamic analysis also measures much more syscalls that what is actually required

What Syscalls to (Really) Implement?

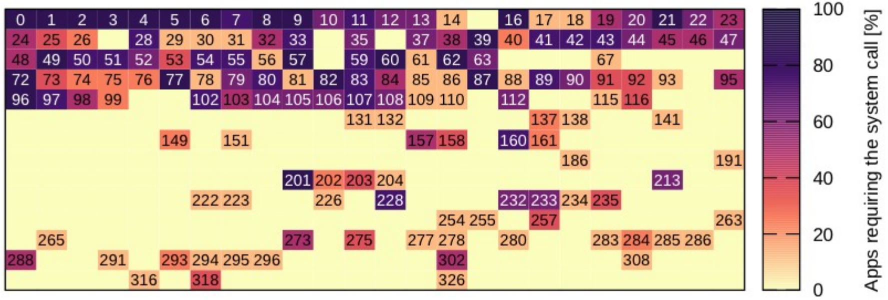


(a) Static analysis, binary.

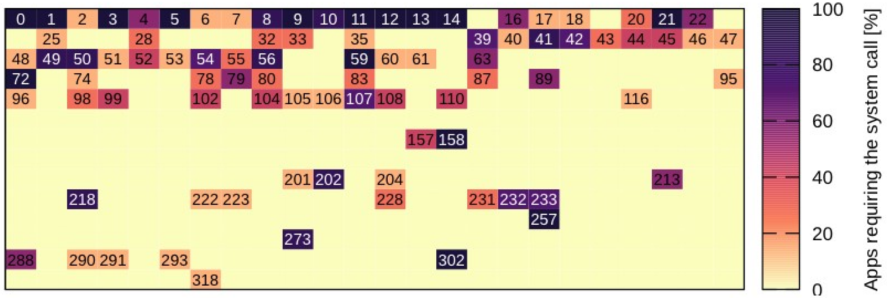
What Syscalls to (Really) Implement?



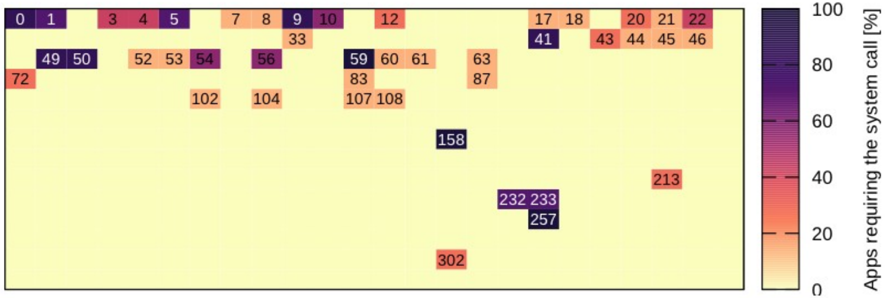
(a) Static analysis, binary.



(b) Static analysis, source.



(c) Dynamic analysis, executed.



(d) Dynamic analysis, required.

Why does Stubbing/Faking Work?

```
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.

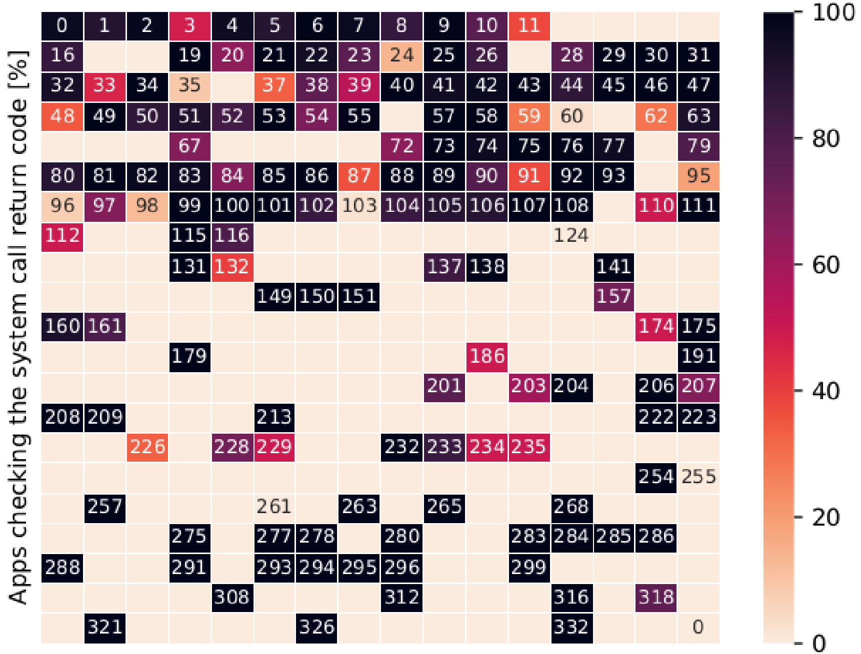


Figure 8. Apps checking system calls return values.

Long-Term Support?

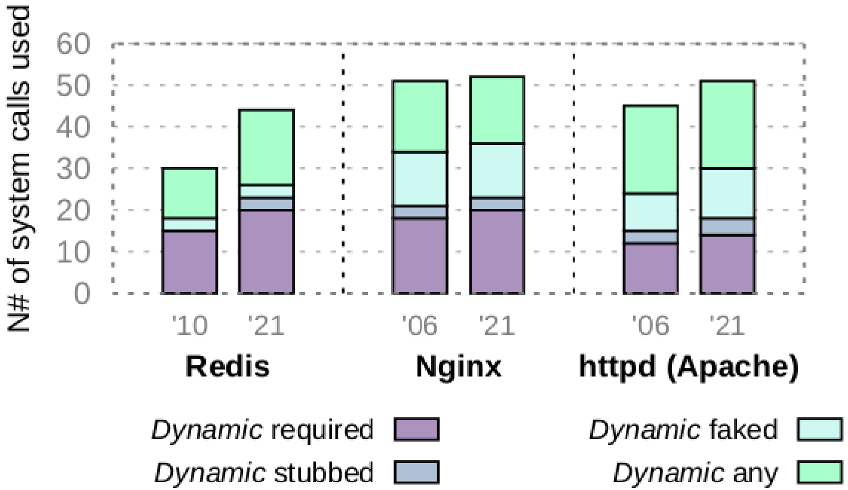


Figure 9. System call usage and capacity to be stubbed/faked for recent (2021) and older (2005-2010) applications releases.

Examples of Support Plans

Step	Implement	Stub	Fake	Apps supported
Unikraft (commit `7d6707f`, supports 174 syscalls)				
0	-	-	-	(12 apps)
1	290	273, 218, 230	-	+ Memcached
2	218	-	-	+ H2O
3	283, 27	186	-	+ MongoDB
Fuchsia (commit `5d20758`, supports 152 syscalls)				
0	-	-	-	(11 apps)
1	-	99, 222, 223	-	+ HAProxy
2	302	273, 230, 105	-	+ Memcached
3	33	-	-	+ Lighttpd
4	128, 99, 27	-	-	+ MongoDB
Kerla (commit `73a1873`, supports 58 syscalls)				
0	-	-	-	(4 apps)
1	56, 257, 54	(17 system calls)	47	+ Httpd
2	10	-	-	+ Weborf
3	232, 233, 302	(9 system calls)	213	+ HAProxy
4	17, 18, 53	96, 40, 201, 105, 106, 116	290	+ Nginx
5	213, 262	95	-	+ Redis
6	291	293	-	+ Lighttpd
7	288, 290	32, 87	-	+ H2O
8	46	230	-	+ Memcached
9	8, 21, 87	-	25	+ SQLite
10	104, 107, 108, 102	-	-	+ Webfsd
11	128, 99, 229, 27, 73, 202, 283	131	137	+ MongoDB

Demo

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