MoonShine: Optimizing OS Fuzzer Seed Selection with Trace Distillation

Shankara Pailoor, Andrew Aday, Suman Jana
Columbia University
OS Fuzzing

• Popular technique to find OS vulnerabilities
• Primarily tests system-call interface
  o Can be invoked by untrusted user programs
  o Large surface area for attack
OS Fuzzing - Overview

Seeds → Fuzzer → Synthetic Programs → OS

Feedback

fd = open(...) write(fd, ...) close(fd) ...
OS Fuzzing - Overview

Seeds

Fuzzer

Synthetic Programs

fd = open(...)
write(fd, ...)
close(fd)
...

Feedback

OS
Synthetic Program Generation

• Goal – Maximize code coverage
• Random generation alone is unlikely to succeed

```c
#include <unistd.h>
size_t write(int fd, const void *buf, size_t count);
```

Opened with write permissions

Valid userspace pointer
Synthetic Program Generation

• Goal – Maximize code coverage
• Random generation alone is unlikely to succeed
  o Fuzzer must track and maintain system-call dependencies
Synthetic Program Generation

• Goal – Maximize code coverage
• Random generation alone is unlikely to succeed
  o Fuzzer must track and maintain system-call dependencies
• State-of-the-art – Thousands of hardcoded rules!!

```c
resource fd[int32]
...
open(file ptr[in], ...) fd
write(f fd, buf buffer[in], count len[buf])
```
Synthetic Program Generation

• Goal – Maximize code coverage
• Random generation alone is unlikely to succeed
  o Fuzzer must track and maintain system-call dependencies
• State-of-the-art – Templates with *thousands* of manual rules
• Hard to scale
MoonShine

Real Program Traces

Distilled Seeds
Trace Distillation vs. User-Level Seed Selection

• MinSet (Sec’14)
  o Find smallest subset with most coverage

• Afl-tmin
  o Dynamically removes blocks of data while preserving coverage
  o Prohibitively slow with traces! Doesn’t scale
  o Needs to understand system call dependencies!
Distillation Challenges

• Minimize trace sizes

• Track dependencies
Why does trace size matter?

- Fuzzer performance tied to program size (# calls)
- 10 second trace of Chromium contains 462,225 calls!!
- Traces can’t be directly used as seeds
Trace Distillation

• Goal – Minimize the traces while preserving coverage

• Strategy – Select calls that contribute most coverage

```
strace /bin/ls
```

Total Calls: 359
New Coverage: 35

10x reduction
Dependencies

• Explicit Dependencies
  o Shared state passed through arguments

• Implicit Dependencies
  o Modify shared kernel data structure
Explicit Dependencies

• Call A is *explicitly dependent* on call B if B produces a result used by A

```python
3 = open("/tmp/file0.txt", O_WRONLY)
16 = write(3, "somerandomtext\n", 16)
```
Implicit Dependencies

• Call \textbf{A} is \textit{implicitly dependent} on Call \textbf{B} if \textbf{B} affects the execution of \textbf{A} by modifying a shared kernel data structure

\begin{verbatim}
int mlockall(...) {
    ...
    void mlock_fixup_lock {
        ...
        if (lock)
            vma->vm_flags = lock_flags
    }
}

int msync(void *addr, size_t length, int flags) {
    ...
    if (vma->vm_flags & VM_LOCKED)
        error = -EBUSY
    ...
}
\end{verbatim}
Tracking Explicit Dependencies

• Statically analyze trace

• **Return Cache**: Map<(Type, Ret-Val), List<Call>>
  
  o *data type* or *semantic type* (e.g., file descriptor)

• If (type, value) key in Return Cache, then every call that returned this key is marked as explicit dependency.
Tracking Implicit Dependencies

- Control and Data Flow Analysis
- Call $c$ uses shared variable $v$ in conditional $\Rightarrow c$ is *read dependent on* $v$
- Call $c$ writes to shared variable $v$ $\Rightarrow c$ is *write dependent on* $v$

If (Overlap != NULL) $\Rightarrow$ Implicit Dependency
Working Example

Trace Excerpt

mlockall(MCL_FUTURE)
3 = open(...)
0x7b2000 = mmap(NULL, ..., 3, 0)
0x7b3000 = mmap(NULL, ..., 3, 0)
0x7b4000 = mmap(NULL, ..., 3, 0)
-EBUSY = msync(0x7b2000, ..., MS_INVALIDATE)
5 = write(1, “Hello”, 5)
3 = write(1, “abc”, 3)

Distilled Trace
Working Example

Trace Excerpt

```c
mlockall(MCL_FUTURE)
3 = open(...)
0x7b2000 = mmap(NULL, ..., 3, 0)
0x7b3000 = mmap(NULL, ..., 3, 0)
0x7b4000 = mmap(NULL, ..., 3, 0)
-EBUSY = msync(0x7b2000, ..., MS_INVALIDATE)
5 = write(1, "Hello", 5)
3 = write(1, "abc", 3)
```
Working Example

Trace Excerpt

mlockall(MCL_FUTURE)
3 = open(…)
0x7b2000 = mmap(NULL, …, 3, 0)
0x7b3000 = mmap(NULL, …, 3, 0)
0x7b4000 = mmap(NULL, …, 3, 0)
-EBUSY = msync(0x7b2000, …,
               MS_INVALIDATE)
5 = write(1, “Hello”, 5)
3 = write(1, “abc”, 3)
Working Example – Explicit Dependencies

Trace Excerpt

mlockall(MCL_FUTURE)
3 = open(…)

0x7b3000 = mmap(NULL, …, 3, 0)
0x7b4000 = mmap(NULL, …, 3, 0)

3 = write(1, “abc”, 3)

Distilled Trace

0x7b2000 = mmap(NULL, …, 3, 0)
-EBUSY = msync(0x7b2000, …, MS_INVALIDATE)
5 = write(1, “Hello”, 5)

Explicit Dependencies
Working Example – Implicit Dependencies

Trace Excerpt

```
mlockall(MCL_FUTURE)
0x7b2000 = mmap(NULL, ..., 3, 0)
0x7b3000 = mmap(NULL, ..., 3, 0)
0x7b4000 = mmap(NULL, ..., 3, 0)
```

Distilled Trace

```
3 = open(...)
0x7b2000 = mmap(NULL, ..., 3, 0)
-EBUSY = msync(0x7b2000, ..., MS_INVALIDATE)
5 = write(1, “Hello”, 5)
```

Implicit Dependencies
Implementation

• Linux Kernel
• Syzkaller — OS Fuzzer (Google)
• Strace — System-call traces
• Kcov — Coverage
• Smatch — Static analysis framework
  o Read deps. with Condition Hook
  o Write deps. with Unary Op and Assign. Hooks
• 2580 lines of Golang and 640 lines of C
# Evaluation - Setup

<table>
<thead>
<tr>
<th>Seed Source</th>
<th>Number of Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glibc Testsuite</td>
<td>1120</td>
</tr>
<tr>
<td>Linux Kernel Selftests</td>
<td>55</td>
</tr>
<tr>
<td>Linux Testing Project (LTP)</td>
<td>390</td>
</tr>
<tr>
<td>Open Posix Testsuite</td>
<td>1630</td>
</tr>
</tbody>
</table>
## New Vulnerabilities

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Module</th>
<th>Operation</th>
<th>Impact</th>
<th>Version Introduced</th>
<th>Distill. Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPF</td>
<td>bpf/devmap.c</td>
<td>dev_map_alloc()</td>
<td>Illegal allocation size</td>
<td>4.0</td>
<td>(I+E) &amp; (E)</td>
</tr>
<tr>
<td>BTRFS</td>
<td>fs/btrfs/file.c</td>
<td>btrfs_fallocate()</td>
<td>Assert Failure</td>
<td>4.14</td>
<td>(I+E)</td>
</tr>
<tr>
<td>Ext4</td>
<td>fs/fs-writeback.c</td>
<td>move_expired_inodes()</td>
<td>Use After Free</td>
<td>4.6</td>
<td>(I+E)</td>
</tr>
<tr>
<td>JFS</td>
<td>fs/jfs/xattr.c</td>
<td>__jfs_setxattr()</td>
<td>Memory Corruption</td>
<td>2.6</td>
<td>(I+E) &amp; (E)</td>
</tr>
<tr>
<td>Network</td>
<td>net/ipv4/inet_connection_sock.c</td>
<td>inet_child_forget()</td>
<td>Use After Free</td>
<td>4.4</td>
<td>(I+E)</td>
</tr>
<tr>
<td>Network</td>
<td>net/core/stream.c</td>
<td>sk_kill_stream_queues()</td>
<td>Memory Corruption</td>
<td>4.4</td>
<td>(I+E)</td>
</tr>
<tr>
<td>Network</td>
<td>net/core/dst.c</td>
<td>dst_release()</td>
<td>NULL Pointer Deref</td>
<td>4.15-rc8</td>
<td>(I+E)</td>
</tr>
<tr>
<td>Network</td>
<td>net/netfilter/nf_conntrack_core.c</td>
<td>init_conntrack()</td>
<td>Memory Leak</td>
<td>4.6</td>
<td>(I+E)</td>
</tr>
<tr>
<td>Network</td>
<td>net/nfc/nfc.h</td>
<td>nfc_device_iter_exit()</td>
<td>NULL Pointer Deref</td>
<td>4.17-rc4</td>
<td>(I+E)</td>
</tr>
<tr>
<td>Network</td>
<td>net/socket.c</td>
<td>socket_setattr()</td>
<td>NULL Pointer Deref</td>
<td>4.10</td>
<td>(I+E) &amp; (E)</td>
</tr>
<tr>
<td>Posix-timers</td>
<td>kernel/time/posix-cpu-timers.c</td>
<td>posix_cpu_timer_set()</td>
<td>Integer Overflow</td>
<td>4.4</td>
<td>(I+E) &amp; (E)</td>
</tr>
<tr>
<td>Reiserfs</td>
<td>fs/reiserfs/inode.c, fs/reiserfs/ioctl.c, fs/direct-io.c</td>
<td>Multiple</td>
<td>Deadlock</td>
<td>4.10</td>
<td>(I+E)</td>
</tr>
<tr>
<td>TTY</td>
<td>tty/serial/8250/8250_port.c</td>
<td>serial8250_console_putchar()</td>
<td>Kernel Hangs Indefinitely</td>
<td>4.14-rc4</td>
<td>(I+E)</td>
</tr>
<tr>
<td>VFS</td>
<td>fs/omap.c</td>
<td>omap_dio_rw()</td>
<td>Data Corruption</td>
<td>3.10</td>
<td>(I+E) &amp; (E)</td>
</tr>
<tr>
<td>VFS</td>
<td>lib/iov_iter.c</td>
<td>iov_iter_pipe()</td>
<td>Data Corruption</td>
<td>3.10</td>
<td>(I+E) &amp; (E)</td>
</tr>
<tr>
<td>VFS</td>
<td>fs/pipe.c</td>
<td>pipe_set_size()</td>
<td>Integer Overflow</td>
<td>4.9</td>
<td>(I+E) &amp; (E)</td>
</tr>
<tr>
<td>VFS</td>
<td>inotify_fsnotify.c</td>
<td>inotify_handle_event()</td>
<td>Memory Corruption</td>
<td>3.14</td>
<td>(I+E)</td>
</tr>
</tbody>
</table>
Coverage Improvement

● 13.1% coverage increase over default Syzkaller with implicit + explicit

● 9.7% coverage increase over default Syzkaller with only explicit
## Effectiveness of Distillation

<table>
<thead>
<tr>
<th>Total Calls</th>
<th>After Distillation</th>
<th>Comparison</th>
<th>Coverage Preserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,900,000</td>
<td>16,400</td>
<td>176x reduction</td>
<td>86%</td>
</tr>
</tbody>
</table>
Vulnerability Discovered By MoonShine
Exhibit: Buffer Overflow in inotify (CVE-2017-7533)

```
inotify_handle_event(..., file_name)
{
    //file_name is currently HelloWorld
    len = strlen(file_name);
    alloc_len += len + 1;
    event = kmalloc(alloc_len, GFP_KERNEL);
}
```

```
sys_rename(..., new_name)
{
    //new_name is LongFileName
    ...
    copy_name(file_name, new_name)
    //file_name changed to LongFileName
}
```

```
strcpy(event->name, file_name);
//strcpy will now overflow event
```
Exhibit: Buffer Overflow in inotify (CVE-2017-7533)

Seed Distilled by MoonShine

1: mmap(...)  
2: r0 = inotify_init ()  
3: r1 = \texttt{inotify\_add\_watch}(r0, ".", 0xffff)  
4: chmod(".", 0x1ed)  
5: r2 = creat("short1", 0x1ed)  
6: close(r2)  
7: \texttt{rename}("short1", "short2")  
8: close(r0)
Exhibit: Buffer Overflow in inotify (CVE-2017-7533)

Seed Distilled by MoonShine

1. mmap(...)  
2. r0 = inotify_init ()  
3. r1 = inotify_add_watch(r0, ".", 0xffff)  
4. chmod(".", 0x1ed)  
5. r2 = creat("short1", 0x1ed)  
6. close(r2)  
7. rename("short1", "short2")  
8. close(r0)
Exhibit: Buffer Overflow in inotify (CVE-2017-7533)

Crash-inducing mutation

1: mmap(...)  
2: r0 = inotify_init ()  
3: r1 = inotify_add_watch(r0, ".", 0xffff)  
4: chmod(".", 0x1ed)  
5: r2 = creat("short1", 0x1ed)  
6: close(r2)  
7: rename("short1", "long_name")  
8: close(r0)
Conclusion

• State-of-the-art OS fuzzers rely on manual rules
  o Hard to scale
• MoonShine scalably generates seeds from traces of real-world programs
  o Lightweight static analysis to track explicit and implicit dependencies
• Discovered 17 new vulnerabilities in Linux kernel

https://github.com/shankarapailoor/moonshine

Getting integrated into syzkaller
Backup Slides
Limitations/Future Work

- Support more OS/Fuzzers
- No multithreaded dependency tracking
- Inter-procedural dependencies
  - Infer that a file must be created from trace
- Multiple distillation strategies
  - Distillation without code coverage?
Static Analysis False Positives/Negatives

• False Positives
  o Imprecise pointer analysis

• False Negatives
  o Incomplete AST traversal - function pointers
  o Shared state is not global variable or struct/union field
  o Aliased struct fields get modified
    • char *p = a->v; p[0] = 1