FACULTY
OF INFORMATICS
Masaryk University

Formal verification of a Linux distribution

AUFOVER@Red Hat

Lukáš Zaoral

April 9, 2021

AUFOVER

- **AU**tomatizace **FO**rmální **VER**ifikace
- 1. 1. 2019 – 31. 12. 2021
- The goal is to support:
  1. development of university tools based on formal mathematical methods
  2. transfer of such tools to a commercial environment including integration with industrial partner's tools
- FI MU (Divine, Symbiotic), FIT VUT, Honeywell, Red Hat

Covscan

- Red Hat’s internal service for automated analysis (of updates) of RPM packages
- Initially just an easy-to-use wrapper for Coverity Scan
- Now supports also Google sanitizers, GCC Static Analyzer, Clang analyzer, ShellCheck, Cppcheck, Pylint, …
- Ongoing effort to support also CBMC, Divine, Facebook Infer, Symbiotic, Valgrind, …

Basic RPM lingo

- **Spec file** – defines all the actions to build an RPM package
- **Source RPM (SRPM)** – spec file + everything else necessary for a successful build
- **`rpmbuild`** – tool that builds a package according to its spec file
- **Binary RPM** – result of successful `rpmbuild`
- **`mock`** – containerised `rpmbuild`

Note

From now on, I will only consider software written in C/C++.

```
Name:           klee
Version:        2.2
Release:        1%{?dist}
Summary:        Symbolic Execution Engine
License:        NCSA
URL:            https://klee.github.io

Source0:        https://github.com/%{name}/%{name}-%{version}.tar.gz
Patch0:         use-python3.patch

BuildRequires:  cmake gcc-c++ make llvm-devel z3-devel

%description
Symbolic virtual machine built on top of the LLVM compiler infrastructure

%prep
%autosetup -p1

%build
%cmake -DENABLE_SOLVER_Z3:BOOL=ON
%cmake_build

%install
%cmake_install

%check
%cmake_build --target check

%files
%license LICENSE.TXT
%{_bindir}/*

%changelog
* Sat Mar 20 2021 Lukas Zaoral <lzaoral@redhat.com> - 2.2-1
- First release
```

Goal

Run the analyses with unmodified SRPMs!

I want to analyse my package!

```
# or cmake, meson, ...
$ ./configure CFLAGS='-O2 -g -fsanitize=address' LDFLAGS='-fsanitize=address'
...
$ make
gcc -O2 -g -fsanitize=address -D_GNU_SOURCE -c -o a.o a.c
gcc -O2 -g -D_GNU_SOURCE -c -o b.o b.c  # Where did it go?
gcc -o test1 c.o b.o  # WTF!?
/usr/bin/ld: c.o: in function `main':
c.c:(.text+0x16): undefined reference to '__asan_handle_no_return'
/usr/bin/ld: b.c:(.text+0x5d): undefined reference to '__asan_report_load8'
...
```

Possible problems:

- environment variables are (sometimes) ignored
- selected flags are discarded
- ...

cswrap

- generic compiler wrapper
- converts relative paths in diagnostic messages to absolute paths
- diagnostic messages are decorated by suffix <--[TOOL]
- aggregates diagnostic messages
- can reliably alter list of flags passed to a compiler

```
$ export PATH="$(cswrap --print-path-to-wrap):$PATH"
$ CSWRAP_ADD_CFLAGS='-O2 -g -fsanitize=address' make
gcc -D_GNU_SOURCE -c -o a.o a.c
gcc -D_GNU_SOURCE -c -o b.o b.c
gcc -o test1 c.o b.o
$ ./test1  # Everything compiled successfully!
=================================================================
==2307043==ERROR: AddressSanitizer: stack-buffer-overflow ...
...
```

https://github.com/kdudka/cswrap

cs{diff,grep,sort,linker}

- Command-line utilities for processing output from various analyzers to a unified format
- `csdiff` takes two lists of defects and output either added or fixed ones
- `csgrep` filters a list of defects by the specified regex-based predicates
- `cssort` sorts the given defect list by the selected key
- `cslinker` can extend the list of defects by CWE numbers

`https://github.com/kdudka/csdiff`

```
Error: CLANG_WARNING
./0006-test.c:7:14: warning: Dereference of null pointer (loaded from variable 'ptr')
#        *ptr = 'A'; /* error */
#         ~~~ ^
./0006-test.c:5:5: note: 'ptr' initialized here
#    char *ptr = malloc(sizeof(char));
#    ^~~~~~~~~
./0006-test.c:6:9: note: Assuming 'ptr' is equal to NULL
#    if (ptr == NULL) {
#        ^~~~~~~~~~~
./0006-test.c:6:5: note: Taking true branch
#    if (ptr == NULL) {
#    ^
./0006-test.c:7:14: note: Dereference of null pointer (loaded from variable 'ptr')
#        *ptr = 'A'; /* error */
#         ~~~ ^
```

```
Error: DIVINE_WARNING
./0006-test.c: scope_hint: In function 'main':
./0006-test.c:7: error: null pointer dereference: [global*]
./0006-test.c:7: note: memory error in userspace
/opt/divine/include/dios/sys/fault.hpp:119: note: void __dios::FaultBase::handler<__dios::Context>(_VM_Fault,
_VM_Frame*, void (*)())
./0006-test.c:7: note: main
/opt/divine/include/dios/libc/sys/start.cpp:91: note: __dios_start
```

```
Error: COMPILER_WARNING
0002-test.c: scope_hint: In function 'main'
0002-test.c:7:5: warning[-Wfree-nonheap-object]: attempt to free a non-heap object 'a'
#    7 |     free(a); /* invalid free */
#      |     ^~~~~~~
```

csmock

- mock wrapper that adds additional functionality and makes subtle changes to the RPM build process for an unattended analysis of RPM packages
- uses tools described on previous slides
- plugin-based
- easy to use:
  1. Run `csmock -t cppcheck,clang zlib-1.2.11-25.fc35.src.rpm`
  2. Wait ...
  3. Profit!

Note

Quick demo at the end!

Is this enough to run dynamic analyzers? Almost.

- Static analyzers and sanitizers – `cswrap`
- Drawback – requires **`%check`** section and upstream test suite

Note

Recall that we do not want to modify the SRPMs!

- Analyse `rpmbuild` and all its sub-processes!
  - Possible for Valgrind – unnecessarily slow and verbose (`rpmbuild`, Bash, CMake, Make, gcc, ...)
  - CBMC, Divine, Symbiotic – compile with `cswrap`, run with ???
- Solution – `csexec`

ELF files and ELF interpreters

- Executable and Linkable Format
  - format used by executables, relocatables and shared libraries
  - divided into sections (`.text`, `.data`, `.dynamic`, ...)
- ELF interpreter (or dynamic linker/loader) – `LD.SO(8)`:
  - The program `ld-linux.so` finds and loads the shared objects (shared libraries) needed by a program, prepares the program to run, and then runs it.
  - Path stored in `.interp` section.

```
$ readelf -x .interp /bin/bash

Hex dump of section '.interp':
0x000002a8 2f6c6962 36342f6c 642d6c69 6e75782d /lib64/ld-linux-
0x000002b8 7838362d 36342e73 6f2e3200          x86-64.so.2.
```

csexec

- ‘`cswrap`’ for ELF binaries
- custom ELF interpreter + main binary
  - used to bootstrap the main binary; cannot not use `libc`; only x86_64 at the moment
  - main binary can use `libc`; appends a bell separated list in CSEXEC_WRAP_CMD to `argv` and executes it explicitly using system ELF interpreter

```
$ cat ./wrap.sh
#!/usr/bin/bash
echo "$1 I was executed through csexec!"
shift
exec "$@"
$ gcc -Wl,--dynamic-linker=/usr/bin/csexec-loader test.c
$ CSEXEC_WRAP_CMD=$'./wrap.sh\aWow!' ./a.out
Wow! I was executed through csexec!
...
```

https://github.com/kdudka/cswrap

csexec

Unfortunately, this approach did not work all the time:

- `argv[0]` could not be reconstructed[1]
- `readlink("/proc/self/exe", buf, bufsize)` returned the analyzer executable
- some `coreutils` tests rely on file descriptor counts

[1] https://sourceware.org/git/?p=glibc.git;a=commitdiff;h=c6702789

Demo!

Are we there yet?

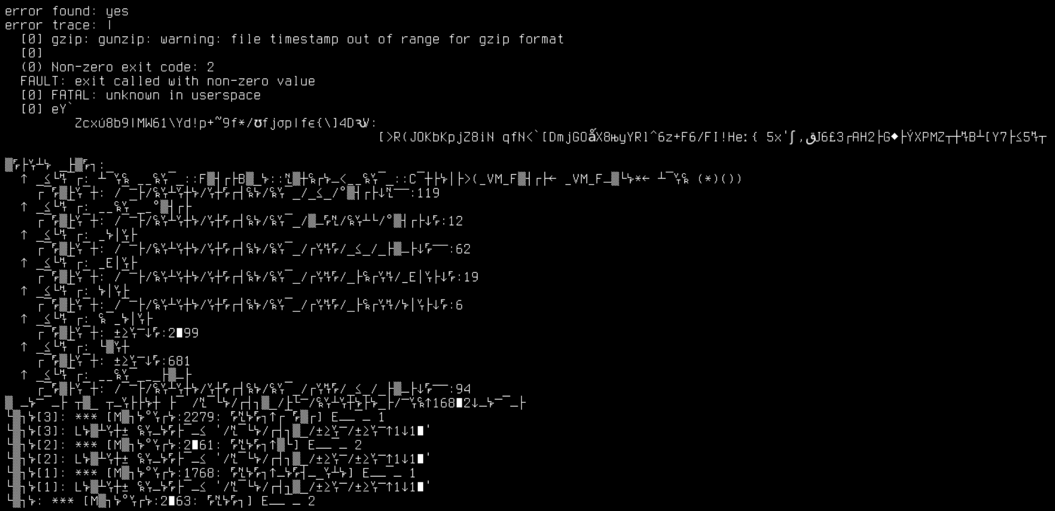
- Compilation failures of tools
- Excessive number of dependencies
- Compilation failures of verified programs
- Regressions – `aufover-benchmark` [2]
- Imprecise or just completely wrong models

```c
#include <assert.h>

int main(void)
{
#ifdef __x86_64__
    assert(0);
#endif
}
```

- Unhelpful verification reports

[2] `https://github.com/aufover/aufover-benchmark`



```
error found: yes
error trace: |
  [0] gzip: gunzip: warning: file timestamp out of range for gzip format
  [0]
  (0) Non-zero exit code: 2
  FAULT: exit called with non-zero value
  [0] FATAL: unknown in userspace
  [0] eY`
         Zcxú8b9|MW61\Yd!p+~9f*/ʊfjσp|fϵ{\]4DݍV:
                                                        [>R(JOKbKpjZ8iN qfN<`[DmjGOẳX8 њyYRl^6z+F6/FI!He:{ 5x'ʃ,ق J6£3┌AH2├G◆├ÝXPMZ┬┼␤B┴[Y7├≤5␤┬

▒␌├␋┴␊ ⎽├▒␌┐:
  ↑ ⎽≤└␉⎺┌: ┴⎺␋␍ ⎽⎽␍␋⎺⎽::F▒┤┌├B▒⎽␊::␤▒┼␍┌␊⎽<⎽⎽␍␋⎺⎽::C⎺┼├␊│├>(⎽VM⎽F▒┤┌├← ⎽VM⎽F⎽▒└␊*← ┴⎺␋␍ (*)())
     ┌⎺␌▒├␋⎺┼: /⎺⎻├/␍␋┴␋┼␊/␋┼␌┌┤␍␊/␍␋⎺⎽/⎽≤⎽/°▒┤┌├↓␤⎻⎻:119
  ↑ ⎽≤└␉⎺┌: ⎽⎽␍␋⎺⎽⎽°▒┤┌├
     ┌⎺␌▒├␋⎺┼: /⎺⎻├/␍␋┴␋┼␊/␋┼␌┌┤␍␊/␍␋⎺⎽/▒⎼␌␤/␍␋┴└/°▒┤┌├↓␌:12
  ↑ ⎽≤└␉⎺┌: ⎽␊│␋├
     ┌⎺␌▒├␋⎺┼: /⎺⎻├/␍␋┴␋┼␊/␋┼␌┌┤␍␊/┌␋␉␌/⎽≤⎽/⎽├▒⎼├↓␌⎻⎻:62
  ↑ ⎽≤└␉⎺┌: ⎽E│␋├
     ┌⎺␌▒├␋⎺┼: /⎺⎻├/␍␋┴␋┼␊/␋┼␌┌┤␍␊/┌␋␉␌/⎽├␍┌␋␉/⎽E│␋├↓␌:19
  ↑ ⎽≤└␉⎺┌: ␊│␋├
     ┌⎺␌▒├␋⎺┼:⎽/⎺⎻├/␍␋┴␋┼␊/␋┼␌┌┤␍␊/┌␋␉␌/⎽├␍┌␋␉/␊│␋├↓␌:6
  ↑ ⎽≤└␉⎺┌: ␍⎺⎽␊│␋├
     ┌⎺␌▒├␋⎺┼: ±≥␋⎻↓␌:2█99
  ↑ ⎽≤└␉⎺┌: └▒␋┼
     ┌⎺␌▒├␋⎺┼: ±≥␋⎻↓␌:681
  ↑ ⎽≤└␉⎺┌: ⎽⎽␍␋⎺⎽⎽⎽├▒⎼├
     ┌⎽␌▒├␋⎺┼: /⎺⎻├/␍␋┴␋┼␊/␋┼␌┌┤␍␊/┌␋␉␌/⎽≤⎽/⎽├▒⎼├↓␌⎻⎻:94
▒ ⎼␊⎻⎺⎼├ ┬▒⎽ ┬⎼␋├├␊┼ ├⎺ /␤⎺└␊/┌┤┐▒⎽/├␊└⎻/␍␋┴␋┼␊⎼␤±≥␋⎻↑168█2↓⎼␊⎻⎺⎼├
└▒┐␊[3]: *** [M▒┐␊°␋┌␊:2279: ␌␤␊␌┐↑┌⎺␌▒┌] E⎼⎼⎺⎼ 1
└▒┐␊[3]: L␊▒┴␋┼± ␍␋⎼␊␌├⎺⎼≤ '/␤⎺└␊/┌┤┐▒⎽/±≥␋⎻/±≥␋⎻↑1↓1█'
└▒┐␊[2]: *** [M▒┐␊°␋┌␊:2█61: ␌␤␊␌┐↑▒└] E⎼⎼⎺⎼ 2
└▒┐␊[2]: L␊▒┴␋┼± ␍␋⎼␊␌├⎺⎼≤ '/␤⎺└␊/┌┤┐▒⎽/±≥␋⎻/±≥␋⎻↑1↓1█'
└▒┐␊[1]: *** [M▒┐␊°␋┌␊:1768: ␌␤␊␌┐↑⎼␊␌┤⎼⎽␋┴␊] E⎼⎼⎺⎼ 1
└▒┐␊[1]: L␊▒┴␋┼± ␍␋⎼␊␌├⎺⎼≤ '/␤⎺└␊/┌┤┐▒⎽/±≥␋⎻/±≥␋⎻↑1↓1█'
└▒┐␊: *** [M▒┐␊°␋┌␊:2█63: ␌␤␊␌┐] E⎼⎼⎺⎼ 2
```