Testing and Verification of Operating Systems and Information Security Issues

Prof. Alexander K. Petrenko,
petrenko@ispras.ru

12th TAROT Summer School on Software Testing, Verification & Validation
ISP RAS belongs to the Division of Mathematical Sciences of the RAS.

The Institute employs more than 200 highly qualified researchers and software engineers, including 12 doctors of science and 45 philosophy doctors.

Many employees of the Institute also work as professors in leading Moscow universities.
Software Engineering Department

- **SE Department staff:**
  - over 40 researchers and engineers, including 3 Doctors of Sc. and 13 Ph.D.

- **Major partners and customers**
  - Foreign partners: Microsoft Research, Intel Labs, Nokia, Google, ETRI, EADS Telecom, University of Passau, Fraunhofer FOKUS
  - Russian partners: NIISI RAS, GosNIIAS, VimpelCom, MCST(Elbrus)
  - International organizations: ISO/JTC 1, ETSI, The Linux Foundation
ISPRAS Research Model = ?

Software Engineering is that part of Computer Science which is too difficult for the Computer Scientist.

— Friedrich L. Bauer —

ISPRAS Research Model = Industrial Research
Application Domains
SE Department R&D Domains

- Verification techniques and tools (testing, software model checking, deductive verification)
- Trusted operating systems (Linux family, ARINC-653 Real-Time OS)
- Tool chains for critical software life cycle support
  - Requirements management tools
  - System modeling (AADL), simulation, risk analysis
  - Cyber-physical system integration (avionics)
- Telecom and operating systems API/ABI standards
- Hardware designs testing
- Model Based Testing foundations
Agenda

1. What is the “Operating System”?
2. Spectrum of OS testing and verification methods
3. State of the Art and ISPRAS’s experience
4. Information security specifics and OS verification
OS Verification Challenge

- Operating System is a base of software platform. Reliability and security of OS is ultimate prerequisite of information technologies quality.
- Critical software/systems need certification. OS certification is necessary part of certification process.
- IT domains requiring reliable, secure, trusted OSs:
  - Servers and work stations
  - Data centers
  - Avionics, other computing intensive systems
  - Mobile devices
  - SCADA, etc.
OS Architecture

- Libraries + Kernel
  - Libraries
  - Kernel
  - Drivers
  - Core kernel

- Monolithic Kernel

- Microkernel
  - Microkernel modules
OS Architecture. Scale

- Libraries + Kernel
  
  Libraries – ~1 million functions, ~ $10^5$ KLOC

  Kernel

- Monolithic Kernel
  
  Core kernel - ~ $5\cdot10^3$ KLOC

  Drivers - ~ 5-100 KLOC

- Microkernel
  
  Microkernel modules - 5-200 KLOC
Operating Systems Structure

User-space

Applications

System
Libraries

Utilities

System
Services

Operating
system

Special
File Systems

System
Calls

Signals,
Memory updates,
Scheduling,
...

Kernel-space

Kernel

Kernel
Threads

Device Drivers

Kernel Core (mmu, scheduler, IPC)

Kernel
Modules

Interrupts, DMA

IO Memory/IO Ports

Platform

Hardware
Spectrum of Testing/Verification Approaches

- Testing (dynamic analysis, monitoring, run-time verification, fault injection)
- Static analysis (*lightweight analysis*, software model checking)
- Static/dynamic analysis (DART, concolic testing)
- Deductive verification
Spectrum of Testing/Verification Approaches vs. Verification Aspects

- Testing (dynamic analysis, monitoring, run-time verification)
- Static analysis (*lightweight analysis*, software model checking)
- Static/dynamic analysis (DART, concolic testing)
- Deductive verification

Testing/Verification aspects:
- Functionality / Conformance / Reliability / Security / . . .
- *Usability testing*
- *Performance modeling and testing*
- . . .
<p>| Static Analysis       | Dynamic Analysis |</p>
<table>
<thead>
<tr>
<th>Static Analysis</th>
<th>Dynamic Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ All paths at once</td>
<td>- One path only</td>
</tr>
<tr>
<td>Static Analysis</td>
<td>Dynamic Analysis</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>+ All paths at once</td>
<td>− One path only</td>
</tr>
<tr>
<td>+ Hardware, test data and test environment is <strong>not</strong> required</td>
<td>− Hardware, test data and test environment is <strong>required</strong></td>
</tr>
<tr>
<td>Static Analysis</td>
<td>Dynamic Analysis</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>+ All paths at once</td>
<td>− One path only</td>
</tr>
<tr>
<td>+ Hardware, test data and test environment is not required</td>
<td>− Hardware, test data and test environment is required</td>
</tr>
<tr>
<td>− There are false positives</td>
<td>+ Almost no false positives</td>
</tr>
<tr>
<td>Static Analysis</td>
<td>Dynamic Analysis</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>+ All paths at once</td>
<td>- One path only</td>
</tr>
<tr>
<td>+ Hardware, test data and test environment is <strong>not</strong> required</td>
<td>- Hardware, test data and test environment is <strong>required</strong></td>
</tr>
<tr>
<td>- There are false positives</td>
<td>+ Almost no false positives</td>
</tr>
<tr>
<td>- Checks for predefined set of bugs only</td>
<td>+ The only way to show the code actually works</td>
</tr>
</tbody>
</table>
State of the Art. Methods and Tools. Testing

- 3 views on OS:
  - OS as API for applications
  - OS is an OS kernel
  - OS is a part of software/hardware platform

- OS as API for applications
  - Problems
    - Huge set of APIs (over 1 million functions)
    - Lack of specifications (poor quality of specifications)
State of the Art. Methods and Tools. Testing

• 3 views on OS:
  - OS as API for applications
  - OS is an OS kernel
  - OS is a part of software/hardware platform

• OS as API for applications.
  • Problems
    - Huge set of APIs (over 1 million functions)
    - Lack of specifications (poor quality of specifications)
  • Methods
    - Traditional (handmade) test suites
    - Specification/model based testing
  • Specification based testing tools
    - ADLT (Sun Microsystems, 1993)
    - KVEST (Nortel, ISPRAS, 1994-1999)
    - UniTESK/CTESK (ISPRAS, 2000-2007)
    - SpecExplorer (Microsoft, 2004-2009)
OLVER – Model Based Testing of Linux Basic Libraries(*)

(*) The project was supported by Russian Ministry of Education and Science and by The Linux Foundation
OLVER: Open Linux VERification

Linux Standard Base – LSB 3.1

- LSB Core 3.1 / ISO 23360
  - ABI
  - Utilities
  - ELF, RPM, ...
- LSB C++
- LSB Desktop

LSB Core ABI

- GLIBC
  - libc
  - libcrypt
  - libdl
  - libm
  - libpthread
  - librt
  - libutil
  - libpam
  - libz
  - libncurses

More 1500 interfaces
OLVER Process

- LSB Requirements
- Specifications
- Test Scenarios
- CTesK Automatic Generator
- Tests
- Test Suite
- Test Reports
- Testing Quality Goals
- Linux System

Test Oracles

Test stimuli → System under test → Test oracle

Specifications (pre- and postconditions)

System under test is a black box that provides API (functions, procedures etc.)


KVEST/UniTesK Workflow

Implementation

Specification
Model of coverage

Model of test
(test scenario)

Output analysis
Trace analysis

Test stimuli generator
So called “Implicit automata” or EFSM derived during on-the-fly test scenario execution.

Implicit automata is an ADT with 2 operations:
- `recognise_node_ID () -> ({new, visited} x ID)`
- `next_call (next_input_stimulus) -> (...)`

The test engine step by step builds/explores all nodes (states) and all available function calls (transitions).
Requirements Catalogue

NAME

memcpy - copy bytes in memory

SYNOPSIS

#include <string.h>

void *memcpy(void *restrict s1, const void *restrict s2, size_t n);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The memcpy() function shall copy n bytes from the object pointed to by s2 into the object pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The memcpy() function shall return s2; no return value is reserved to indicate an error.
memcpy() specification template

```c
{
pre
{
   // If copying takes place between objects that overlap, the behavior is undefined.
   //REQ("app.memcpy.02", "Objects are not overlapped", TODO_REQ() );

   return true;
}
post
{
   /*The memcpy() function shall copy n bytes from the object
      pointed to by s2 into the object pointed to by s1. */
   //REQ("memcpy.01", "s1 contain n bytes from s2", TODO_REQ() );

   // The memcpy() function shall return s1; */
   //REQ("memcpy.03", "memcpy() function shall return s1", TODO_REQ() );

   return true;
}
}```
memcpy() precondition

specification
VoidTPtr memcpy_spec( CallContext context, VoidTPtr s1, VoidTPtr s2, SizeT n )
{
    pre
    {
        /* [Consistency of test suite] */
        REQ("", "Memory pointed to by s1 is available in the context",
            isValidPointer(context,s1) );
        REQ("", "Memory pointed to by s2 is available in the context",
            isValidPointer(context,s2) );

        /* [Implicit precondition] */
        REQ("", "Memory pointed to by s1 is enough",
            sizeWMemoryAvailable(s1) >= n );
        REQ("", "Memory pointed to by s2 is enough",
            sizeRMemoryAvailable(s2) >= n );

        // If copying takes place between objects that overlap, the behavior is undefined.
        REQ("app.memcpy.02", "Objects are not overlapped",
            !areObjectsOverlapped(s1,n,s2,n) );
    }
    return true;
}
memcpy() postcondition

specification
VoidTPtr memcpy_spec( CallContext context, VoidTPtr s1, VoidTPtr s2, SizeT n ) {
    post
    {
        /* The memcpy() function shall copy n bytes from the object
           pointed to by s2 into the object pointed to by s1. */
        REQ("memcpy.01", "s1 contain n bytes from s2",
            equals( readCByteArray_VoidTPtr(s1,n), @readCByteArray_VoidTPtr(s2,n) )
        );

        /* [The object pointed to by s2 shall not be changed] */
        REQ("", "s2 shall not be changed",
            equals( readCByteArray_VoidTPtr(s2,n), @readCByteArray_VoidTPtr(s2,n) )
        );

        /* The memcpy() function shall return s1; */
        REQ("memcpy.03", "memcpy() function shall return s1",
            equals_VoidTPtr(memcpy_spec,s1) );

        /* [Other memory shall not be changed] */
        REQ("", "Other memory shall not be changed",
            equals( readCByteArray_MemoryBlockExceptFor( getTopMemoryBlock(s1), s1, n ),
                @readCByteArray_MemoryBlockExceptFor( getTopMemoryBlock(s1), s1, n ) )
        );
    }
    return true;
}
### Failure report: requirement \{mvcur.04\} failed

**Postcondition failed**: If (newrow, newcol) is not a valid address for the terminal in use, mvcur() fails. 

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario</td>
<td>move_scenario</td>
</tr>
<tr>
<td>State</td>
<td>NULL</td>
</tr>
<tr>
<td>Transition</td>
<td>nCurses_mscvrs scen()</td>
</tr>
<tr>
<td>Specification</td>
<td>mvcur_spec()</td>
</tr>
<tr>
<td>Parameter value</td>
<td>struct ThreadId context = struct (0, 9546, 3066666666)</td>
</tr>
<tr>
<td>Parameter value</td>
<td>NCursesPosition * @cursorPosOld = struct (0, 0)</td>
</tr>
<tr>
<td>Parameter value</td>
<td>NCursesPosition * @cursorPosNew = struct (10000000, 10000000)</td>
</tr>
<tr>
<td>Parameter value</td>
<td>NCursesPosition * @cursorPosOld = struct (0, 0)</td>
</tr>
<tr>
<td>Parameter value</td>
<td>NCursesPosition * @cursorPosNew = struct (10000000, 10000000)</td>
</tr>
<tr>
<td>Return value</td>
<td>(int) 0</td>
</tr>
<tr>
<td>Coverage Branch</td>
<td>The only branch</td>
</tr>
<tr>
<td>Prime formula</td>
<td>invariant type NCursesPosition * (@cursorPosOld) = true</td>
</tr>
<tr>
<td>Prime formula</td>
<td>invariant type NCursesPosition * (@cursorPosNew) = true</td>
</tr>
<tr>
<td>Prime formula</td>
<td>invariant type NCursesPosition * (@cursorPosNew) = true</td>
</tr>
<tr>
<td>Prime formula</td>
<td>invariant type NCursesPosition * (@cursorPosNew) = true</td>
</tr>
</tbody>
</table>
Requirements Coverage Report

- [+] fs.glob (64 / 33 / 0)
- [+] fs.meta.access (123 / 56 / 0)
- [+] fs.meta.meta (111 / 44 / 0)
- [+] fs.meta.statvfs (45 / 12 / 0)
- [+] fs.name (24 / 8 / 1)
  o [+] realpath (15 / 6 / 0)
  o [+] dirname (5 / 2 / 0)
    o [+] dirname01
      The dirname() function shall return a pointer to a string that is the parent directory of path.
      dirname0101
      The dirname() function shall take a pointer to a character string that contains a pathname, and return a pointer to a string that is a pathname of the parent directory of that file.
      dirname02
      Trailing '' characters in the path are not counted as part of the path.
      dirname03
      If path does not contain a '/', then dirname() shall return a pointer to the string "".
      dirname04
      If path is a null pointer or points to an empty string, dirname() shall return a pointer to the string "".
      app_dirname05
      The dirname() function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.
      dirname07
      If path is a null pointer or points to an empty string, a pointer to a string "" is returned.
      app_dirname06
      The dirname() function may modify the string pointed to by path, and may return a pointer to static storage that may then be overwritten by subsequent calls to dirname().
    o [+] basename (4 / 0 / 1)
      o [+] basename01
        The basename() function shall return a pointer to the final component of path.
      o [+] basename0101 (FAILED)
        The basename() function shall take the pathname pointed to by path and return a pointer to the final component of the pathname, deleting any trailing '/' characters.
      o [+] basename02
        If the string pointed to by path consists entirely of the '/' character, basename() shall return a pointer to the string "".
      o [+] basename03
        If the string pointed to by path is exactly ""/, it is implementation-defined whether '/' or "" is returned.
      o [+] basename04
        If path is a null pointer or points to an empty string, basename() shall return a pointer to the string "".
      o [+] basename05
        The basename() function may modify the string pointed to by path, and may return a pointer to static storage that may then be overwritten by a subsequent call to basename().
      o [+] basename06
        The basename() function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.
- [+] fs.symlink (33 / 16 / 0)
- [+] fs.tmpfile (69 / 18 / 0)
- [+] fio.file (1151 / 375 / 0)
- [+] fio.fstream.buffer (21 / 1 / 0)
- [+] fio.fstream.fstream (747 / 37 / 0)
- [+] fio.fstream.lock (51 / 0 / 0)


#include <curses.h>

int mvcur(int oldrow, int oldcol, int newrow, int newcol);

DESCRIPTION

(mvcur.01) The mvcur() function outputs one or more commands to the terminal that move the terminal's cursor to (newrow, newcol), an absolute position on the terminal screen. (mvcur.02) The (oldrow, oldcol) arguments specify the former cursor position. (mvcur.03.01) Specifying the former position is necessary on terminals that do not provide coordinate-based movement commands. (mvcur.03.02) On terminals that provide these commands, Curses may select a more efficient way to move the cursor based on the former position. (mvcur.04) If (newrow, newcol) is not a valid address for the terminal in use, mvcur() fails. (mvcur.05) If (oldrow, oldcol) is the same as (newrow, newcol), then mvcur() succeeds without taking any action. (mvcur.06) If mvcur() outputs a cursor movement command, it updates its information concerning the location of the cursor on the terminal.

RETURN VALUE

(mvcur.07.01) Upon successful completion, mvcur() returns OK. (mvcur.07.02) Otherwise, it returns ERR.

ERRORS

No errors are defined.

APPLICATION USAGE
OLVER Results

- Requirements catalogue built for LSB and POSIX
  - 1532 interfaces
  - 22663 elementary requirements
- 97 deficiencies in specification reported
- Formal specifications and tests developed for
  - 1270 interface (good quality)
  - + 260 interfaces (basic quality)
- 80+ bugs reported in modern distributions
- OLVER is a part of the official LSB Certification test suite
  http://ispras.linuxfoundation.org
OLFVER Conclusion

- model based testing allows to achieve better quality using less resources
- maintenance of MBT is cheaper
OLVER Conclusion

- model based testing allows to achieve better quality using less resources if you have smart test engineers
- maintenance of MBT is cheaper if you have smart test engineers
OLVER Conclusion

- model based testing allows to achieve better quality using less resources if you have smart test engineers
- maintenance of MBT is cheaper if you have smart test engineers
- traditional tests are more useful for typical test engineers and developers
model based testing allows to achieve better quality using less resources if you have smart test engineers
maintenance of MBT is cheaper if you have smart test engineers
traditional tests are more useful for typical test engineers and developers
so, long term efficiency is questionable
but...
Configuration Testing
Product Line Testing
State of the Art.
Methods and Tools. Testing

- 3 views on OS:
  - OS as API for applications
  - OS is an OS kernel
  - OS is a part of software/hardware platform

- OS is a part of software/hardware platform
  - **Problems**
    - Huge number of configurations
    - Unavailable hardware devices and lack of devices models
  - **Methods**
    - Ad-hoc ≡ proprietary know-how
    - Systematical reduction of target configurations

---


- **Tools**
  - No commercial or popular tool
- **Testing quality**
  - Not available
Linux Product Line Verification

• University of Waterloo

• University of Passau
OS Kernel
Testing/Verification
State of the Art.
Methods and Tools. Testing

• 3 views on OS:
  – OS as API for applications
  – **OS is an OS kernel**
  – OS is a part of software/hardware platform

• OS is a kernel
  • **Problems**
    – Event driven multithreading systems
    – Lack of specifications (poor quality of specifications, Microsoft Windows is an exclusion)
  • **Methods**
    – Run-time verification
    – Fault simulation


• **Tools**
  – No commercial or popular tool applicable in kernel mode

• **Testing quality**
  – Average test coverage lower 20%
Run-Time Verification
Sanitizer Tools Family.
Google research group of Konstantin Serebryany(*)

Run-time verification and compile-time code instrumentation.

Tools:

• MemorySanitizer: fast detector of uninitialized memory use in C++
• AddressSanitizer: A Fast Address Sanity Checker
• Dynamic Race Detection with LLVM Compiler
• ThreadSanitizer – data race detection
• KernelThreadSanitizer – data races in Linux Kernel

(*) http://research.google.com/pubs/KonstantinSerebryany.html
Robustness Testing
Fault Handling Code

- Is not so fun
- Is really hard to keep all details in mind
- Practically is not tested
- Is hard to test even if you want to
- Bugs seldom(never) occurs
  => low pressure to care
Why do we care?

- It beats someone time to time
- Safety critical systems
- Certification authorities
Operating Systems Structure

User-space
- Applications
  - System Libraries
  - Utilities
  - System Services

Kernel-space
- Kernel
  - Kernel Modules
  - Kernel Threads
  - Device Drivers
  - Signals, Memory updates, Scheduling, ...

Kernel Core (mmu, scheduler, IPC)

Operating system
- Interrupts, DMA
- IO Memory/IO Ports

Hardware
Run-Time Testing of Fault Handling

- Manually targeted test cases
  + The highest quality
  - Expensive to develop and to maintain
  - Not scalable

- Random fault injection on top of existing tests
  + Cheap
  - Oracle problem
  - No any guarantee
  - When to finish?
Systematic Approach

- **Hypothesis:**
  - Existing tests lead to more-or-less deterministic control flow in kernel code

- **Idea:**
  - Execute existing tests and collect all potential fault points in kernel code
  - *Systematically* enumerate the points and inject faults there
Fault Injection Implementation

- Based on KEDR framework*
  - intercept requests for memory allocation/bio requests
    - to collect information about potential fault points
    - to inject faults
  - also used to detect memory/resources leaks

(*) http://linuxtesting.org/project/kedr
KEDR Workflow

http://linuxtesting.org/project/kedr
**Systematic** vs. **Random**

- + 2 times more cost effective
- + Repeatable results
- – Requires more complex engine

- + Cover double faults
- – Unpredictable
- – Nondeterministic
<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Brief</th>
<th>Added on</th>
<th>Accepted</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0011</td>
<td>Crash</td>
<td>ext4: When mounted with backup superblock online resize leads to BUG_ON or causes filesystem corruption</td>
<td>2014-12-27</td>
<td><a href="http://www.spinics.net/lists/linux-ext4/msg46743.html">http://www.spinics.net/lists/linux-ext4/msg46743.html</a> commit</td>
<td>Fixed in kernel 3.19-rc4</td>
</tr>
<tr>
<td>F0009</td>
<td>Crash</td>
<td>ext4: Destruction of ext4_groupinfo_caches during one mount causes BUG_ON for other mounted ext4 filesystems</td>
<td>2014-05-12</td>
<td><a href="https://lkml.org/lkml/2014/5/12/147">https://lkml.org/lkml/2014/5/12/147</a> commit</td>
<td>Fixed in kernel 3.16-rc1</td>
</tr>
<tr>
<td>F0008</td>
<td>Crash</td>
<td>f2fs: BUG_ON() is triggered in recover_inode_page() when mount valid f2fs filesystem</td>
<td>2014-04-18</td>
<td><a href="https://lkml.org/lkml/2014/4/14/189">https://lkml.org/lkml/2014/4/14/189</a> commit</td>
<td>Fixed in kernel 3.17-rc1</td>
</tr>
<tr>
<td>F0007</td>
<td>Crash</td>
<td>f2fs: f2fs unmount hangs if f2fs_init_acl() fails during mkdir syscall</td>
<td>2014-02-17</td>
<td><a href="https://lkml.org/lkml/2014/2/6/18">https://lkml.org/lkml/2014/2/6/18</a> commit</td>
<td>Fixed in kernel 3.15-rc1</td>
</tr>
<tr>
<td>F0006</td>
<td>Deadlock</td>
<td>f2fs: a deadlock in mkdir if ACL is enabled</td>
<td>2013-10-28</td>
<td><a href="https://lkml.org/lkml/2013/10/26/163">https://lkml.org/lkml/2013/10/26/163</a> commit</td>
<td>Fixed in kernel 3.12-rc1</td>
</tr>
<tr>
<td>F0001</td>
<td>Crash</td>
<td>ext4: NULL pointer dereference in mount_fs() because of ext4_fill_super() wrongly reports</td>
<td>2012-11-08</td>
<td><a href="https://bugzilla.kernel.org/show_bug.cgi?id=48431">https://bugzilla.kernel.org/show_bug.cgi?id=48431</a> commit</td>
<td>Fixed in kernel 3.8-rc1</td>
</tr>
</tbody>
</table>
Concolic Testing
Concolic Testing

- Concolic = Symbolic + Concrete
  - SUT runs in concrete and in symbolic modes
  - Symbolic execution is used to collect conditions and branches of the current path
  - Collected data is used to generate new input data to cover more execution paths
# Concolic Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Language</th>
<th>Platform</th>
<th>Constraint Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>DART</td>
<td>C</td>
<td>NA</td>
<td>lp_solver</td>
</tr>
<tr>
<td>SMART</td>
<td>C</td>
<td>Linux</td>
<td>lp_solver</td>
</tr>
<tr>
<td>CUTE</td>
<td>C</td>
<td>Linux</td>
<td>lp_solver</td>
</tr>
<tr>
<td>CREST</td>
<td>C</td>
<td>Linux</td>
<td>Yices</td>
</tr>
<tr>
<td>EXE</td>
<td>C</td>
<td>Linux</td>
<td>STP</td>
</tr>
<tr>
<td>KLEE</td>
<td>C (LLVM bitcode)</td>
<td>Linux</td>
<td>STP</td>
</tr>
<tr>
<td>Rwset</td>
<td>C</td>
<td>Linux</td>
<td>STP</td>
</tr>
<tr>
<td>PathCrawler</td>
<td>C</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>SAGE</td>
<td>Machine code</td>
<td>Windows</td>
<td>Disolver</td>
</tr>
</tbody>
</table>
S2E for Kernel Testing

- based on KLEE
- uses patched Qemu
  - source code is not required
- supports plugins

(*) https://s2e.epfl.ch/
## Testing Aspects

<table>
<thead>
<tr>
<th>Monitoring Aspects</th>
<th>T2C</th>
<th>OLVER</th>
<th>Autotest</th>
<th>Cfg</th>
<th>FI</th>
<th>KEDR-LC</th>
<th>S2E</th>
<th>RH</th>
<th>KStrider</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kinds of Observable Events</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interface events</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>internal events</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Events Collection</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>internal</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>external</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>embedded</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Requirements Specification</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in-place (local, tabular)</td>
<td>+</td>
<td>+</td>
<td>If</td>
<td>Dis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>formal model (pre/post+invariants,...)</td>
<td>+</td>
<td></td>
<td>If</td>
<td>Co</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assertions/prohibited events</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Events Analysis</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>online</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in-place</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>outside</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

60 / 115
<table>
<thead>
<tr>
<th>Active Aspects</th>
<th>T2C</th>
<th>OLVER</th>
<th>Autotest</th>
<th>Cfg</th>
<th>FI</th>
<th>KEDR-LC</th>
<th>S2E</th>
<th>RH</th>
<th>KStrider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Test Situations Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>requirements coverage</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>class equivalence coverage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>model coverage (SUT/reqs)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>source code coverage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test Situations Setup/Set Gen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>passive</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fixed scenario</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>manual</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pre-generated</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>coverage driven</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>random</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>adapting scenario</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>coverage driven</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>source code coverage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>model/... coverage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>random</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>as option</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application interface</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HW interface</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>internal actions</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>inside</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>outside</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Software Model Checking
State of the Art. Methods and Tools. Software Model Checking

- Approaches:
  - Counterexample guided abstraction refinement (CEGAR) - Edmund Clarke et al.
  - Configurable Program Analysis – Dirk Beyer
  - Abstract interpretation - Patrick Cousot and Radhia Cousot
  - Bounded Model Checking – BMC – Edmund Clarke et al.

- Gold practices
  - Microsoft Research (SLAM)
  - LDV – Linux Driver Verification

- Problems
  - Lack of specs
  - Limitations on size and complexity of modules (no more 30-100KLine)

- Tools
  - Many but no commercial or popular tool

- Verification quality
<table>
<thead>
<tr>
<th>Competition candidate</th>
<th>BLAST 2.7</th>
<th>CPAchecker ABE 1.0.10</th>
<th>CPAchecker Memo 1.0.10</th>
<th>ESBMC 1.17</th>
<th>FShell 1.3</th>
<th>LLBMC 0.9</th>
<th>Predator 2011011</th>
<th>QARMC-HSF</th>
<th>SATabs 3.0</th>
<th>Wolverine 0.5c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affiliation</td>
<td>Moscow, Russia</td>
<td>Passau, Germany</td>
<td>Paderborn, Germany</td>
<td>Southampton, UK</td>
<td>Vienna, Austria</td>
<td>Karlsruhe, Germany</td>
<td>Brno, Czechia</td>
<td>Munich, Germany</td>
<td>Oxford, UK</td>
<td>Princeton, USA</td>
</tr>
<tr>
<td>ControlFlowInteger</td>
<td>71 / 9000 s</td>
<td>141 / 1000 s</td>
<td>140 / 3200 s</td>
<td>102 / 4500 s</td>
<td>28 / 580 s</td>
<td>100 / 2400 s</td>
<td>17 / 1100 s</td>
<td>140 / 4800 s</td>
<td>75 / 5400 s</td>
<td>39 / 580 s</td>
</tr>
<tr>
<td>DeviceDrivers</td>
<td>72 / 30 s</td>
<td>51 / 97 s</td>
<td>51 / 93 s</td>
<td>63 / 160 s</td>
<td>20 / 3.5 s</td>
<td>80 / 1.6 s</td>
<td>80 / 1.9 s</td>
<td>--</td>
<td>71 / 140 s</td>
<td>68 / 65 s</td>
</tr>
<tr>
<td>DeviceDrivers64</td>
<td>55 / 1400 s</td>
<td>26 / 1900 s</td>
<td>49 / 500 s</td>
<td>10 / 870 s</td>
<td>0 / 0 s</td>
<td>1 / 110 s</td>
<td>0 / 0 s</td>
<td>--</td>
<td>32 / 3200 s</td>
<td>16 / 1300 s</td>
</tr>
<tr>
<td>HeapManipulation</td>
<td>--</td>
<td>4 / 16 s</td>
<td>4 / 16 s</td>
<td>1 / 220 s</td>
<td>--</td>
<td>17 / 210 s</td>
<td>20 / 1.0 s</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>SystemC</td>
<td>33 / 4000 s</td>
<td>35 / 1000 s</td>
<td>36 / 450 s</td>
<td>67 / 760 s</td>
<td>--</td>
<td>8 / 2.4 s</td>
<td>21 / 630 s</td>
<td>8 / 820 s</td>
<td>57 / 5000 s</td>
<td>36 / 1900 s</td>
</tr>
<tr>
<td>Concurrency</td>
<td>--</td>
<td>0 / 0 s</td>
<td>0 / 0 s</td>
<td>6 / 270 s</td>
<td>--</td>
<td>0 / 0 s</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Overall</td>
<td>231 / 15000 s</td>
<td>267 / 1100 s</td>
<td>280 / 4300 s</td>
<td>249 / 6800 s</td>
<td>48 / 580 s</td>
<td>206 / 2700 s</td>
<td>138 / 1700 s</td>
<td>148 / 5600 s</td>
<td>236 / 14000 s</td>
<td>159 / 3800 s</td>
</tr>
</tbody>
</table>
# SVCOMP’2014 Results

<table>
<thead>
<tr>
<th>Competition candidate</th>
<th>BLAST</th>
<th>CBMC</th>
<th>CPAnumber</th>
<th>CPAlien</th>
<th>CSeq-Lazy</th>
<th>CSeq-MU</th>
<th>ESBMC 1.22</th>
<th>FrankenBit</th>
<th>LLBMC</th>
<th>Predator</th>
<th>Symbiotic 2</th>
<th>Threader</th>
<th>UFO</th>
<th>Ultimate Autonomizer</th>
<th>Ultimate Kojak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representing Jury Member</td>
<td>Mutilin</td>
<td>Tautschik</td>
<td>Stellenbosch, South Africa</td>
<td>Brno, Czechia</td>
<td>London, UK</td>
<td>Passau, Germany</td>
<td>Belgrade, Serbia</td>
<td>Cordeiro, Brazil</td>
<td>Pittsburgh, USA</td>
<td>Falke, Germany</td>
<td>Vojnar, Prague</td>
<td>Brno, Czechia</td>
<td>Brno, Czechia</td>
<td>Munich, Germany</td>
<td>Pittsburgh, USA</td>
</tr>
<tr>
<td>Affiliation</td>
<td>Moscow, Russia</td>
<td>London, UK</td>
<td>Stellenbosch, South Africa</td>
<td>London, UK</td>
<td>Belgrade, Serbia</td>
<td>Cordeiro, Brazil</td>
<td>Pittsburgh, USA</td>
<td>Falke, Germany</td>
<td>Vojnar, Prague</td>
<td>Brno, Czechia</td>
<td>Brno, Czechia</td>
<td>Munich, Germany</td>
<td>Pittsburgh, USA</td>
<td>Freiburg, Germany</td>
<td>Freiburg, Germany</td>
</tr>
<tr>
<td>BitVectors</td>
<td>68</td>
<td>2300 s</td>
<td>70</td>
<td>690 s</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>68</td>
<td>28 s</td>
<td>39</td>
<td>220 s</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Concurrency</td>
<td>78 tasks, max. score: 136</td>
<td>--</td>
<td>128</td>
<td>20000 s</td>
<td>0</td>
<td>136</td>
<td>1200 s</td>
<td>32</td>
<td>30000 s</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>--</td>
<td>100</td>
<td>--</td>
</tr>
<tr>
<td>ControlFlow</td>
<td>843 tasks, max. score: 1261</td>
<td>508</td>
<td>397</td>
<td>1009</td>
<td>9000 s</td>
<td>455</td>
<td>--</td>
<td>949</td>
<td>35000 s</td>
<td>986</td>
<td>8300 s</td>
<td>961</td>
<td>13000 s</td>
<td>511</td>
<td>3400 s</td>
</tr>
<tr>
<td>ControlFlow memeber</td>
<td>843 tasks, max. score: 255</td>
<td>64</td>
<td>788</td>
<td>178</td>
<td>121</td>
<td>--</td>
<td>85</td>
<td>148</td>
<td>10000 s</td>
<td>74</td>
<td>2000 s</td>
<td>--</td>
<td>184</td>
<td>5500 s</td>
<td>35</td>
</tr>
<tr>
<td>Loops</td>
<td>90 tasks, max. score: 90</td>
<td>25</td>
<td>99</td>
<td>68</td>
<td>16</td>
<td>--</td>
<td>88</td>
<td>76</td>
<td>95</td>
<td>27</td>
<td>16</td>
<td>347</td>
<td>44</td>
<td>20</td>
<td>29</td>
</tr>
<tr>
<td>Routines</td>
<td>97 tasks, max. score: 920</td>
<td>630</td>
<td>918</td>
<td>715</td>
<td>92</td>
<td>--</td>
<td>928</td>
<td>605</td>
<td>925</td>
<td>292</td>
<td>347</td>
<td>44</td>
<td>20</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>DeviceDrivers64</td>
<td>128 tasks, max. score: 2766</td>
<td>2682</td>
<td>2683</td>
<td>2631</td>
<td>28000 s</td>
<td>--</td>
<td>2358</td>
<td>14000 s</td>
<td>2639</td>
<td>3000 s</td>
<td>980</td>
<td>2200 s</td>
<td>2642</td>
<td>5700 s</td>
<td>--</td>
</tr>
<tr>
<td>Heap Manipulation</td>
<td>60 tasks, max. score: 135</td>
<td>--</td>
<td>132</td>
<td>12000 s</td>
<td>107</td>
<td>71</td>
<td>97</td>
<td>9070 s</td>
<td>--</td>
<td>107</td>
<td>130 s</td>
<td>111</td>
<td>95</td>
<td>105</td>
<td>--</td>
</tr>
<tr>
<td>Memory Safety</td>
<td>01 tasks, max. score: 98</td>
<td>--</td>
<td>4</td>
<td>11000 s</td>
<td>95</td>
<td>460 s</td>
<td>9</td>
<td>1300 s</td>
<td>--</td>
<td>39</td>
<td>170 s</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Recursive</td>
<td>23 tasks, max. score: 39</td>
<td>--</td>
<td>30</td>
<td>11000 s</td>
<td>0</td>
<td>590 s</td>
<td>--</td>
<td>53</td>
<td>4900 s</td>
<td>--</td>
<td>3</td>
<td>0.38 s</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Sequentialized Concurrent</td>
<td>261 tasks, max. score: 364</td>
<td>--</td>
<td>237</td>
<td>47000 s</td>
<td>97</td>
<td>--</td>
<td>244</td>
<td>38000 s</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>45 tasks, max. score: 67</td>
<td>30</td>
<td>66</td>
<td>15000 s</td>
<td>67</td>
<td>430 s</td>
<td>--</td>
<td>31</td>
<td>27000 s</td>
<td>37</td>
<td>830 s</td>
<td>0</td>
<td>0.0 s</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Overall</td>
<td>2568 tasks, max. score: 4718</td>
<td>--</td>
<td>3501</td>
<td>560000 s</td>
<td>2987</td>
<td>48000 s</td>
<td>--</td>
<td>975</td>
<td>280000 s</td>
<td>1843</td>
<td>24000 s</td>
<td>184</td>
<td>11000 s</td>
<td>220</td>
<td>42000 s</td>
</tr>
</tbody>
</table>

Note: The table entries include the tool names and their respective scores, times, and other performance metrics for various classes of benchmarks and competition categories.
SVCOMP‘2015 Results

LDV: Linux Driver Verification
Commit Analysis(*)

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
- 3101 patches overall

Raw data: http://linuxtesting.org/downloads/ldv-commits-analysis-2012.zip
Commit Analysis

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
- 3101 patches overall

<table>
<thead>
<tr>
<th>Unique commits to drivers</th>
<th>Support of a new functionality (321 ~ 20%)</th>
<th>Bug fixes (1182 ~ 80%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1503 ~ 50%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Commit Analysis

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
- 3101 patches overall

<table>
<thead>
<tr>
<th>Typical bug fixes</th>
<th>Generic bug fixes</th>
<th>Fixes of Linux kernel API misuse</th>
<th>Fixes of data races, deadlocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>(349 ~ 30%)</td>
<td>(102 ~ 30%)</td>
<td>(176 ~ 50%)</td>
<td>(71 ~ 20%)</td>
</tr>
<tr>
<td>Rule classes</td>
<td>Types</td>
<td>Number of bug fixes</td>
<td>Percents</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------</td>
<td>----------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Alloc/free resources</td>
<td></td>
<td>32</td>
<td>~18%</td>
</tr>
<tr>
<td>Check parameters</td>
<td></td>
<td>25</td>
<td>~14%</td>
</tr>
<tr>
<td>Work in atomic context</td>
<td></td>
<td>19</td>
<td>~11%</td>
</tr>
<tr>
<td>Uninitialized resources</td>
<td></td>
<td>17</td>
<td>~10%</td>
</tr>
<tr>
<td>Synchronization primitives in one thread</td>
<td></td>
<td>12</td>
<td>~7%</td>
</tr>
<tr>
<td>Style</td>
<td></td>
<td>10</td>
<td>~6%</td>
</tr>
<tr>
<td>Network subsystem</td>
<td></td>
<td>10</td>
<td>~6%</td>
</tr>
<tr>
<td>USB subsystem</td>
<td></td>
<td>9</td>
<td>~5%</td>
</tr>
<tr>
<td>Check return values</td>
<td></td>
<td>7</td>
<td>~4%</td>
</tr>
<tr>
<td>DMA subsystem</td>
<td></td>
<td>4</td>
<td>~2%</td>
</tr>
<tr>
<td>Core driver model</td>
<td></td>
<td>4</td>
<td>~2%</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td>27</td>
<td>~15%</td>
</tr>
<tr>
<td>NULL pointer dereferences</td>
<td></td>
<td>31</td>
<td>~30%</td>
</tr>
<tr>
<td>Alloc/free memory</td>
<td></td>
<td>24</td>
<td>~24%</td>
</tr>
<tr>
<td>Syntax</td>
<td></td>
<td>14</td>
<td>~14%</td>
</tr>
<tr>
<td>Integer overflows</td>
<td></td>
<td>8</td>
<td>~8%</td>
</tr>
<tr>
<td>Buffer overflows</td>
<td></td>
<td>8</td>
<td>~8%</td>
</tr>
<tr>
<td>Uninitialized memory</td>
<td></td>
<td>6</td>
<td>~6%</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td>11</td>
<td>~11%</td>
</tr>
<tr>
<td>Synchronization</td>
<td></td>
<td>60</td>
<td>~85%</td>
</tr>
<tr>
<td>Races</td>
<td></td>
<td>11</td>
<td>~15%</td>
</tr>
</tbody>
</table>

**Correct usage of the Linux kernel API**

(176 ~ 50%)

**Generic**

(102 ~ 30%)

**Synchronization**

(71 ~ 20%)
Software Model Checking

- Reachability problem
• **void other_func(int v)**
  
  ```
  assert( m != NULL);
  ```

• **void other_func(int v)**
  
  ```
  assert( x != NULL);
  ```
Device Driver World

```c
int usbpn_open(struct net_device *dev) { ... };
int usbpn_close(struct net_device *dev) { ... };

struct net_device_ops usbpn_ops = {
    .ndo_open = usbpn_open, .ndo_stop = usbpn_close
};

int usbpn_probe(struct usb_interface *intf, const struct usb_device_id *id){
    dev->netdev_ops = &usbpn_ops;
    err = register_netdev(dev);
}

void usbpn_disconnect(struct usb_interface *intf){...}

struct usb_driver usbpn_struct = {
    .probe = usbpn_probe, .disconnect = usbpn_disconnect,
};

int __init usbpn_init(void){ return usb_register(&usbpn_struct);}
void __exit usbpn_exit(void){usb_deregister(&usbpn_struct );}
```

**Callback interface procedures registration**

**No explicit calls to init/exit procedures**
Driver Environment Model

```c
int main(int argc, char* argv[])
{
    usbpn_init();
    for(;;) {
        switch(*) {
            case 0: usbpn_probe(*,*,*);break;
            case 1: usbpn_open(*,*);break;
            ...
        }
    }
    usbpn_exit();
}
```
Driver Environment Model (2)

- Order limitation
  - `open()` after `probe()`, but before `remove()`
- Implicit limitations
  - `read()` only if `open()` succeed
  - and it is specific for each class of drivers
Model Checking and Linux Kernel

- Reachability problem

![Diagram showing a model checking process with nodes labeled L1#1, L2#1, L3#1, L4#1, L4#2, L5#1, ERR#1, L5#2, and ERR#2. The diagram illustrates the flow of expressions and conditions, including (x > y), !(x > y), z = x - y, z = y - x, !(z < 0), and (z < 0). The entry point is indicated as 'DONE', and the error location is marked as 'ERR#2'.]
Instrumentation

• **int f(int y)**
• **{**
• **struct urb **x;**
• **x = usb_alloc_urb(0,GFP_KERNEL);**
• **...**
• **usb_free_urb(x);**
• **return y;**
• **}"
Model Checking and Linux Kernel

- Reachability problem
Error Trace Visualizer

Rule: Mutex lock/unlock

Error trace

Function bodies

Source code

```
3182 LDV_IN_INTERRUPT = 1;
3191 +ldv_initialize_FOREACH();
3195 tmp_8 = nonet_int() /* The function body
3195 assert(tmp_8 != 0);
3198 tmp_7 = nonet_int() /* The function body
3200 assert(tmp_7 != 0);
3208 assert(tmp_7 != 1);
3300 assert(tmp_7 != 2);
3340 assert(tmp_7 != 3);
3350 assert(tmp_7 != 4);
3360 assert(tmp_7 != 5);
3370 assert(tmp_7 != 6);
3380 assert(tmp_7 != 7);
3390 assert(tmp_7 != 8);
3400 assert(tmp_7 != 9);
4000 assert(tmp_7 != 10);
4080 assert(tmp_7 != 11);
+carl9170_op_set_key(var_groupl /* hw */
1031 ar = *(hw).priv;
1035 err = 0;
1044 assert(!v if (ar->disable_offload || !_vif)
1045 return -EOPNOTSUPP;
1046 /* We have to fall back to software encryption, whenever
1047 the user choose to participates in an IBSS or is connected
1048 to more than one network.
1049 */
```

This is very unfortunate, because some machines cannot handle
the high throughpt speed in 802.11n networks.

/*
While the hardware supports *catch-all* key, for offloading
group-key en-/de-cryption. The way of how the hardware
decides which keyId maps to which key, remains a mystery...
Bugs Found (230 patches already applied)

Problems in Linux Kernel

This section contains information about problems in Linux kernel found within Linux Driver Verification program.

<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Brief</th>
<th>Added on</th>
<th>Accepted</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0212</td>
<td>Deadlock</td>
<td>nfit: acpi_nfit_notify(): Do not leave device locked</td>
<td>2015-12-11</td>
<td><a href="https://lkml.org/lkml/2015/12/11/781">https://lkml.org/lkml/2015/12/11/781</a> commit</td>
<td>Fixed in kernel 4.4-rc6</td>
</tr>
<tr>
<td>L0208</td>
<td>Crash</td>
<td>mcb: Do not return zero on error path in mcb_pci_probe()</td>
<td>2015-10-28</td>
<td><a href="https://lkml.org/lkml/2015/10/17/238">https://lkml.org/lkml/2015/10/17/238</a> commit</td>
<td>Fixed in kernel 4.4-rc1</td>
</tr>
<tr>
<td>L0207</td>
<td>Crash</td>
<td>staging: r8188eu: _enter_critical_mutex()</td>
<td>2015-10-28</td>
<td><a href="https://www.spinics.net/lists/kernel/msg2094451.html">https://www.spinics.net/lists/kernel/msg2094451.html</a> commit</td>
<td>Fixed in kernel 4.4-rc1</td>
</tr>
</tbody>
</table>
Deductive Verification

Deductive Verification

• Approaches:
  – Design and verify an ideal “perfect” OS
  – Verify a critical component of real-life OS

• Gold practices
  • L4 Kernel Verification
  • seL4
    – Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt. seL4: Formal Verification of an Operating-System Kernel
  • Verisoft OS
  • Verisoft + Microsoft Research – Pike OS, Hyper-V verification

• Problems
  – Tools limitations and lack of module specifications, no frozen interfaces in Linux Kernel

• Tools
  – Many but no commercial or common used tool
Astraver Project

- Deductive Verification of Linux Security Module
  - Joint project with NPO RusBITech
  - Formal security model MROSL-DP

- Assumptions
  - Linux kernel core conforms with its specifications
    - It is not target to prove

- Code under verification
  - Code is hardware independent
  - Verification unfriendly
MROSL DP

- Operating system access control model
  - Hierarchical Role-Based Access Control (RBAC)
  - Mandatory Access Control (MAC)
  - Mandatory Integrity Control (MIC)
- Implemented as Linux Security Module (LSM) for Astra Linux
- ~150 pages in mathematical notation
LSM Verification Project

*LSM stands for Linux Security Module*

Security requirements in math notation (MROSL DP model integrates of RBAC, MIC and MAC)

Implementation of LSM in Linux kernel
From Rigorous to Formal Security Model Requirements

```
rename_entity(x, x', y, name, z)

x, x' ∈ S, y, z ∈ E, y ∈ H_x(z), name ∈ NAME \ {""},
(x, z, write_o) ∈ A, [если shared_container(z) = true,
tо существует r ∈ R ∪ AR такая, что (x, r, read_o) ∈ 
AA и (y, own_o) ∈ PA(r)], либо (f(x) = f(x), если
CCR(y) = false илиCCRl(y) = false, то (x, i, entities_admin_role, read_o) ∈ AA), либо [f(x)
и (x, downgrade_admin_role, read_o) ∈ AA], либо f(x)
i(x), [если i(z) = i_high, то (x', f(x)_i_entity, write_o) ∈ A]
```
Example: \( \text{access\_write}(x, x', y) \) vs. Implementation

\[
x, x' \in S, \\
y \in E \cup R \cup AR
\]

существует \( r \in R \cup AR: (x, r, \text{read}_a) \in AA, \)

[если \( y \in E, \) то]

\[
i_e(y) \leq i_s(x)
\]

и (либо \( \text{execute\_container}(x, y) = true \))

и, если \( y \in E\_HOLE, \) то \( f_e(x) \leq f_e(y), \)
инчае \( f_e(y) = f_e(x), \)
либо \( (x, \text{downgrade\_admin\_role, read}_a) \in AA, \)

и \( (y, \text{write}_r) \in PA(r), \)

[если \( y \in R \cup AR, \) то \( (y, \text{write}_r) \in PA(x), \)]

\[
i_r(y) \leq i_s(x), \text{Constraint}_{AA}(AA') = true,
\]

(для \( e \in \{y\} \) либо \( (x, e, \text{read}_a) \in A, \) либо \( (x, e, \text{write}_a) \in A), \) либо \( f_r(y) = f_r(x), \)
либо \( (x, \text{downgrade\_admin\_role, read}_a) \in AA), \)

[если \( y \in E \) и \( i_s(y) = i_{\text{high}} \) или]

(\( y \in R \cup AR \) и \( i_r(y) = i_{\text{high}} \)) или

(\( x', f_s(x) \_\text{entity, write}_a) \in A \]

\[
\begin{alg}
\text{if} (o\_type \&\& PDP\_TYPE\_EHHOLE) \text{return} 0;
\text{if}\ (\text{mode} \&\& \_W\_OK) \{
\text{if}\ (s\_lev < o\_lev) \{ \text{If}\ ((s\_cat \& o\_cat) != o\_cat) \text{return} -1.\}
\text{if}\ (s\_lev > o\_lev) \{ \text{If}\ ((s\_cat < o\_cat) \text{return} -1.\}
\text{if}\ (s\_cat = o\_cat) \text{return} -1.\}
\\text{return} 0;
\end{alg}
LSM Verification Project

*LSM stands for Linux Security Module*

- Model in math notation (semiformal)
- Formal model
- LSM specs in ACSL
- LSM implementation in C

Semiformal to Formal

Abstract interfaces to implementation interfaces

Abstract model verification

LSM implementation verification
Verification Tool Chain

MROSL-DP model in math notation

\[ \text{access\_read}(x, x', y) \]

\[
\begin{align*}
x, x', y \in E \cup R \cup AR, \text{ существует } r \in R \cup AR: [x, r, y, read] \in AA, \\
& \begin{cases}
\text{если } y \in E, \text{ то } [y, read] \in \text{PA}(r) \text{ и либо } \text{execute\_container}(x, y) = \\
\text{true и } f(y) \leq f(x), \text{ либо } (x, downgrade\_admin\_role, \text{read}, y) \in AA], \\
\text{если } y \in R \cup AR, \text{ то } [y, read] \in \text{PA}(r), l(y) \leq l(x), \\
\text{Constraint}_{\text{PA}}(AA) = \text{true, для } e \in E \text{ либо } (x, e, read) \in A, \text{ либо } (x, e, write) \in A], \\
\text{либо } f(y) \leq f(x), \text{ либо } (x, downgrade\_admin\_role, \text{read}, y) \in AA], \\
\text{если } y \in R \cup AR \text{ и } l(y) = [\text{high}, \text{to } (x', f(x) \_entity, \text{write}, y) \in A] \\
\end{cases}
\end{align*}
\]

\( S' = S, E' = E, APA' = APA, PA' = PA, \\
\text{user}' = \text{user}, H' = H, F' = F, \\
\text{если } y \in E, \text{ то } [x' = A \cup \{x, y, \text{read}\}], \\
AA' = AA, \\
\text{если } y \in R \cup AR, \text{ то } \\
[AA' = AA \cup \{x, y, \text{read}\}], A' = A \]

Part of LSM in Astra Linux

Deductive verification of MROSL-DP model

Rodin (Event-B)

Frama-C, Why3

Deductive verification LSM in Astra Linux
LSM Verification Project

*LSM stands for Linux Security Module*

- Model in math notation
- Model in Event-B
- LSM specs in ACSL
- Rodin toolset
- Frama-C (Why2, Jessie)
- Abstract model verification
- LSM implementation verification
## Deductive Verification in C (*)

<table>
<thead>
<tr>
<th>Open source</th>
<th>Memory model</th>
<th>Already applied for OS low-level code verification</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>—</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Why3</td>
<td>+</td>
<td>+</td>
<td>—</td>
</tr>
<tr>
<td>Frama-C WP</td>
<td>+</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>VeriFast</td>
<td>—</td>
<td>+</td>
<td>—</td>
</tr>
<tr>
<td>C-to-Isabelle</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

(*) The research on deductive verification tools development was carried out with funding from the Ministry of Education and Science of Russia (the project unique identifier is RFMEFI60414X0051)
Frama-C–Jessie–Why3

Why2

Why3

C-progam with
ACSL annotations

CIL'

CIL with annotations

Jessie Plug-In

Program in Jessie

Jessie Engine

Why2 Generator

Why3 Generator

Program in WhyML

Why3 IDE

Verification Condition
Transformations

Verification conditions
in WhyML

Why3 VCG

Why3 Verification Results
Database

Formula Encoder

Theorem Encoder

SMT-LIB, etc.

Alt-Ergo, Z3, CVC4

Theorems Coq, PVS, Mizar

Coq, PVS
Problems with the tools

- Memory model limitations
  - Arithmetics with pointers to fields of structures (container_of)
  - Prefix structure casts
  - Reinterpret casts
- Integer model problems
- Limited code support
  - Functional pointers
  - String literals
- Scalability problems
- Usability problems
18-Feb-2015: The first public release of Astraver Toolset

Submit by Mikhail Mandrykin on Wed, 18/02/2015 - 14:30

We are happy to announce the first public release of Astraver Toolset 1.0 that is built on top of the Frama-C + Jessie + Why3 IDE deductive verification toolchain. The toolchain was adapted, so it can be used to specify and prove properties of Linux kernel code. The most of our modifications go to the Jessie plugin, while the Frama-C front-end and the Why3 platform have got just minor fixes or improvements. Some of our modifications were already applied upstream, while the rest is available in our public repositories.

The most important modifications are described below.

C Language Support

- Low-level reinterpret type casts between pointers to integral types. This feature required modification of the Jessie memory model as described in our paper "Extended High-Level C-Compatible Memory Model with Limited Low-Level Pointer Cast Support for Jessie Intermediate Language". The overall idea can be summarized as an ability to do certain ghost re-allocations of memory blocks in explicitly specified points in order to transform arrays of allocated objects (structures) from one type to another. WARNING. Discriminated unions support is not yet fully adapted to the modified memory model.
- Prefix type casts between outer structures and their corresponding first substructures (through field inlining and structure inheritance relation in Jessie).
- Kernel memory (de)allocating functions kmalloc()/kzalloc(), kfree().
- Built-in C99 _Bool type.
- Standard library functions memcpy(), memmove(), memcmp() and memset(). The support for these functions is implemented through type-based specialization of several pre-defined pattern specifications.
- Function pointers (through exhaustive may-aliases checking).
- Variadic functions (through additional array argument).
- Inline assembly (through undefined function calls).

(*) The main purpose of implementing support for these features was the ability to use the tools on our target code without the need for its significant preliminary modification. As a result the support is not complete enough to be...
LSM Verification Project

*LSM stands for Linux Security Module*

- **Model in math notation**
- **Model in Event-B**
- **LSM specs in ACSL**

**Semiformal to Formal**

- Abstract interfaces to implementation interfaces

**Rodin toolset**

- Abstract model verification
- LSM implementation verification

**Handmade > 10 pages**

**Comments > 100 pages**

**Event-B ~ 3000 lines**

**C Source code ~ 5 Kline**

**ACSL code > 15 Kline**

**Frama-C (Why2, Jessie)**
Hierarchical MROSL DP Model
(decomposition of Event-B model)

1. RBAC – Role-based access control

2. Model 1. with MAC (Mandatory access control)

3.1. Model 2 with MAC and information flow in memory control

3.2. Model 2 for hypervisors

4.1. Model 3.1 with MAC and information flow in time control

4.2. Model 3.1 for distributed systems
LSM Verification Conclusion

- InfoSec requirements are essentially non-functional, they are not decomposed as the functional requirements and
- the direct correspondence between the formal security model entities implementation entities of such a complex system as the operating system (?) can not be built
- What to do?
Final Discussion
OS Scale

- Libraries + Kernel
  - Libraries – ~1 million functions, ~ $10^5$ KLOC
  - Kernel

- Monolithic Kernel
  - Core kernel - ~ $5 \cdot 10^3$ KLOC
  - Drivers - ~ 5-100 KLOC

- Microkernel
  - Microkernel modules - 5-200 KLOC
OS Scale - Verification Approaches

- Libraries + Kernel ~ $10^6$ KLOC
- Monolithic Kernel (~ $10^4$ KLOC, Linux, Windows)
- Hypervisors (Hyper-V) < 300 KLOC
- Drivers/modules < 100 KLOC
- Microkernel (< 10 KLOC, L4, PikeOS)

Testing

Software Model Checking

Deductive Verification
Verification Approaches and Development Processes

- **Static verification**
  - High quality test suite
  - Deductive verification

- **Static analysis**
  - One kind of bugs in 1 execution
  - Lightweight development processes

- **Static**
  - All kinds of bugs in all executions
  - Heavyweight development processes
What is “Heavyweight processes”?
Verification Approaches and Development Processes

- Static analysis
- Static verification
- High quality test suite
- Deductive verification

- 1 kind of bugs in 1 execution
- All kinds of bugs in all executions

- Lightweight development processes
- Heavyweight development processes
Verification Approaches and Development Processes

- **Static analysis**
- **Static verification**
- **High quality test suite**
- **Deductive verification**

1. **One test** in 1 execution
   - **Lightweight development processes**

2. **Static analysis** in 1 execution

3. **High quality test suite** verification in all executions
   - **Heavyweight development processes**

**all kinds of bugs**

**1 kind of bugs**
Conclusion on Practical Verification

Trivial conclusions:

• No silver bullet

• We are seeing remarkable progress in the use of formal and other sophisticated software analysis techniques.

Other ones:

• However deep testing and verification require a deep knowledge of the system under analysis and it is not clear how such a situation may change in the near future

• The axiom that testing should be done by an independent testers group in the case of very complex systems is not valid.
Conclusion on Practical Verification

- Dines Bjørner: Each development team must include at least one mathematician.
- In practice, Intel and Microsoft have integrated development team and testers.
- seL4 & PikeOS verification experience shows that such projects joint designers and mathematicians-verifiers.

Dines Bjørner
Conclusion OS Information Security

Trivial conclusion:

- Safety & security strongly intersect, one without the other cannot be provided.
- Deep verification easier to perform for a small and simple OS than for large and complex one.

Other ones:

- Programmers try to ensure safety without linking the design decisions with security issues - to some extent it is possible.
- But sometimes we cannot follow this way, for example, we cannot pass certification process.
- A high level of confidence requires heavyweight processes, in particular, careful work with the requirements specification - this is the most difficult moment - pointed out by Alan Perlis.
Conclusion OS Information Security

This is not surprising since computers can compute so much more than we yet know how to specify

— Alan Perlis —

A high level of confidence requires heavyweight processes, in particular, careful work with the requirements specification - this is the most difficult moment - pointed out by Alan Perlis
Conclusion OS Information Security

• We have to establish the problem of conformance of security model with protection mechanisms of a trusted operating system informally (or formally in part).

• Shura-Bura noted that the transition from the informal to the formal is essentially informal.

• This thesis leads to the conclusion that in addition to the verification tasks we have establish and solve the validation task.

• Open problem: How to combine and reuse the techniques, tools, and verification&validation artifacts?
Acknowledgements:

- TAROT organizers
- Antoine Rollet
- Alexey Khoroshilov, Victor Kuliamin, Petr Devyanin
- Sponsors and industrial partners
Merci!
Welcome to SYRCoSE-2017 in Kazan(Innopolis) May 29-31, 2017
http://syrcose.isprras.ru
Read More . . .

– [http://www.ispras.ru/groups/se/](http://www.ispras.ru/groups/se/)

• Publications

• Open projects: UniTESK, OLVER, LDV, BLAST, CPAchecker, MASIW, Requality, Frama-C/Why3/Jessie
  – [http://unitesk.ru](http://unitesk.ru)
  – [http://hardware.ispras.ru](http://hardware.ispras.ru)
  – [http://linuxtesting.org](http://linuxtesting.org)
Merci!