LinuxTesting.org

Proving sequential properties of unmodified Linux kernel code

Alexey Khoroshilov

khoroshilov@ispras.ru

ISP RAS

Ivannikov Institute for System Programming of the Russian Academy of Sciences

LinuxTesting.org

Linux Verification Center

Institute for System Programming of Russian Academy of Sciences

VERIFICATION CENTER OF THE OPERATING SYSTEM Linux

founded in 2005

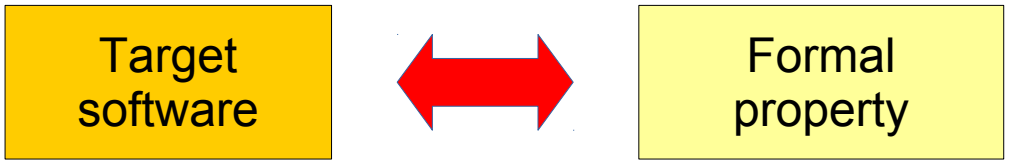
- OLVER Program
- Linux Standard Base Infrastructure Program
- Linux Driver Verification Program
- Linux File System Verification Program
- Linux Deductive Verification

More Secure Software?

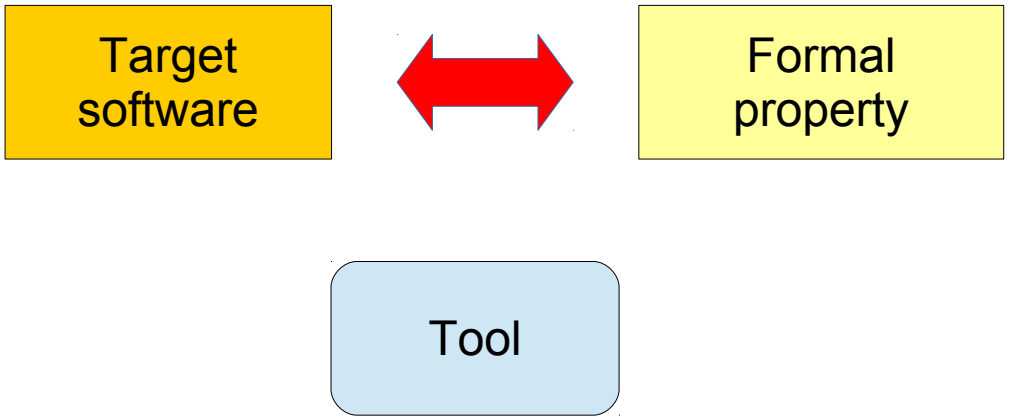
- More confidence

Program Analysis

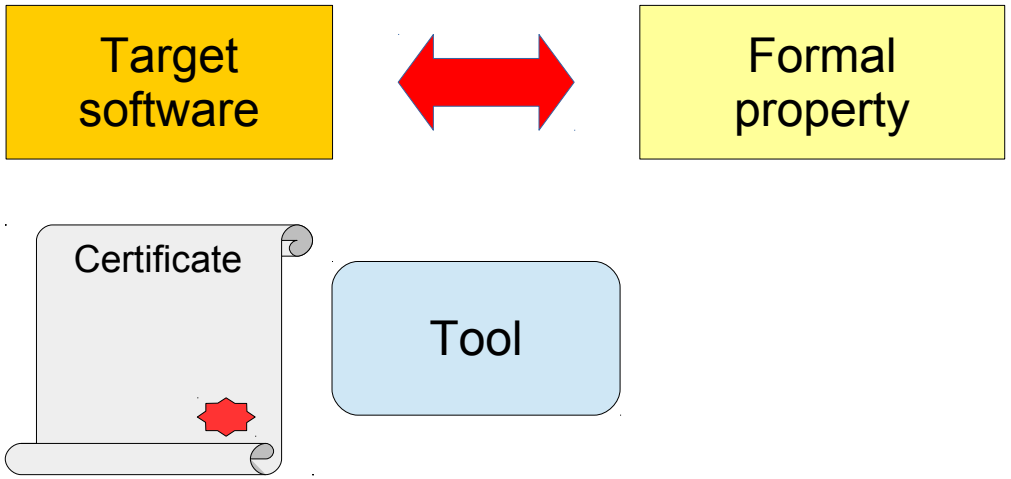
Program Analysis



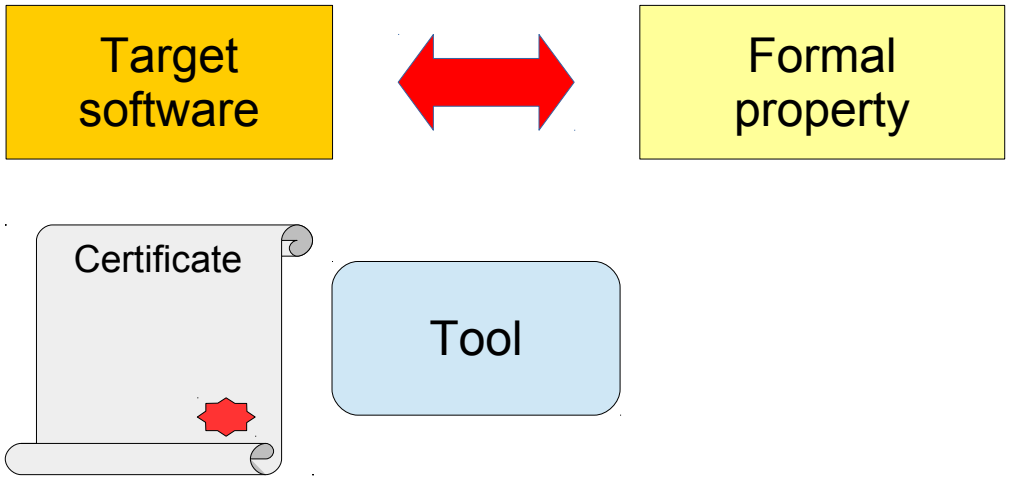
Program Analysis



Program Analysis

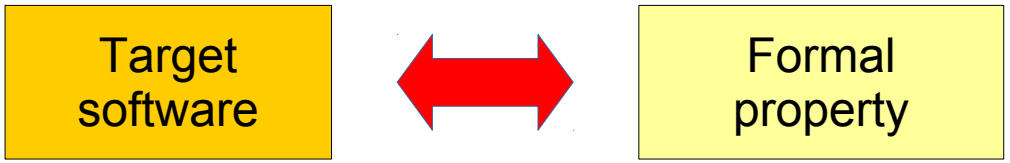


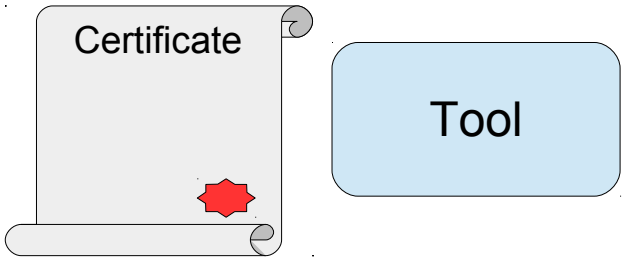
Program Analysis



- up to:
  - Formal property is valid
  - Tool is correct
  - Tool assumptions are held

Program Analysis

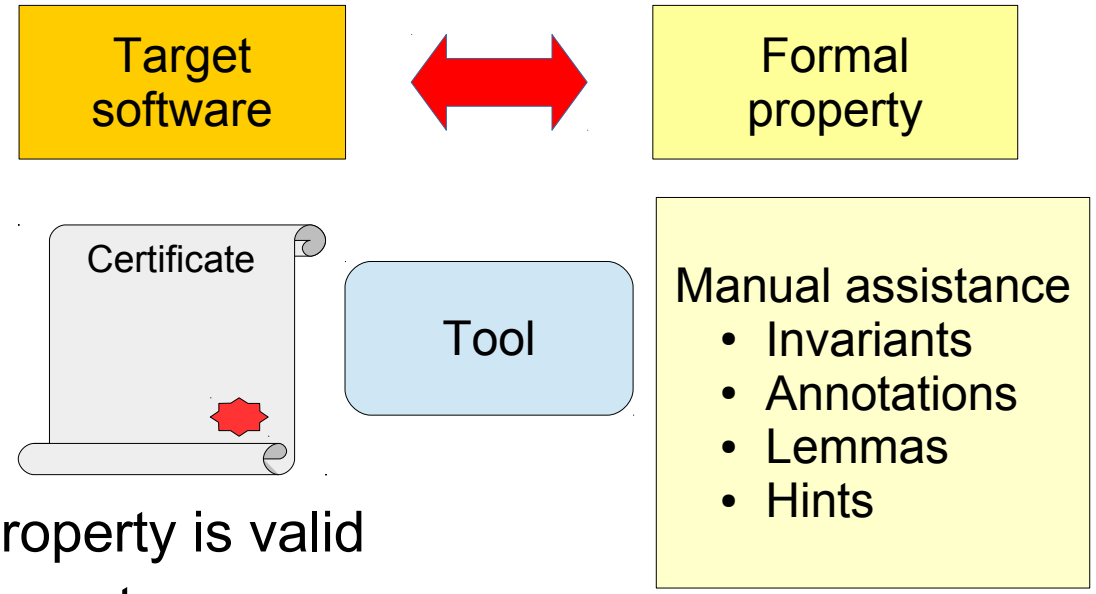




- Absence of typical errors (e.g. memory safety)
- Functional properties

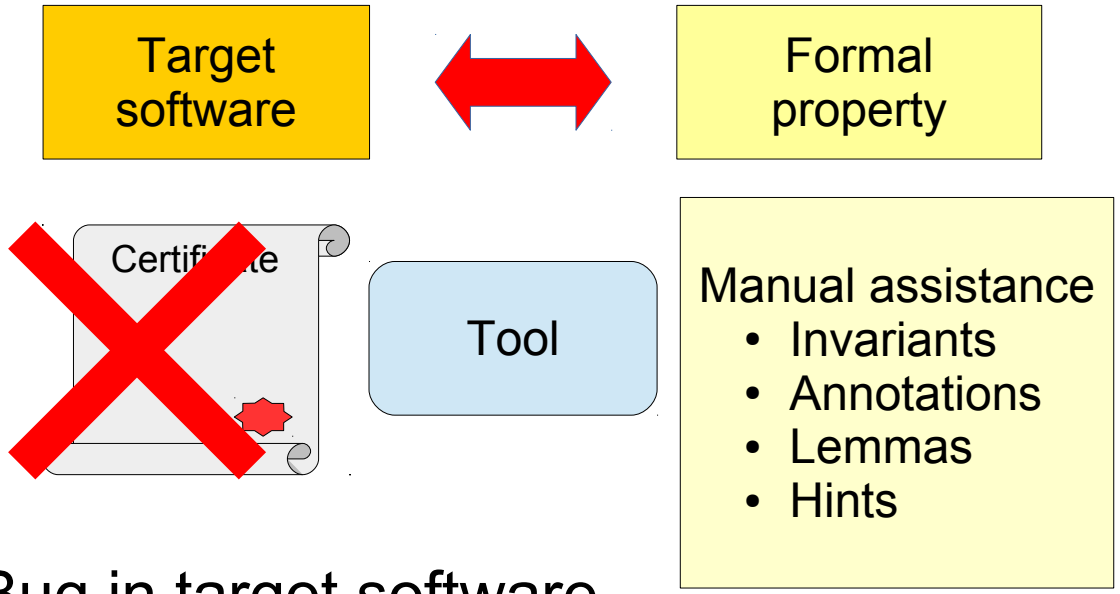
- up to:
  - Formal property is valid
  - Tool is correct
  - Tool assumptions are held

Program Analysis



- up to:
  - Formal property is valid
  - Tool is correct
  - Tool assumptions are held

Program Analysis



- Bug in target software
- Bug in formal property
- Problems with the tool

Legacy Code

Legacy Code

Target software

Formal property

- Absence of typical errors (e.g. memory safety)
- Functional properties

Legacy Code

Target software

Formal property

- Absence of typical errors (e.g. memory safety)
- **Functional properties**

- Informal specification (e.g. ISO/IEC 15408 or DO-178)
- Documentation
- Nothing

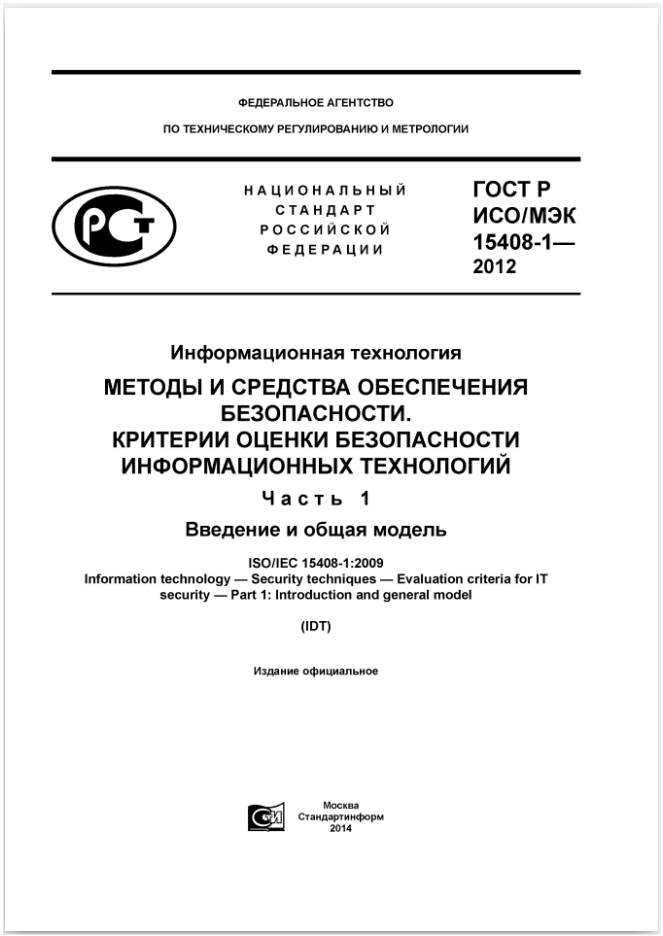
Verification of Linux kernel

- Absence of typical errors
  - Linux Driver Verification [out of scope]
- Functional properties (sequential)
  - Informal specification (ISO/IEC 15408)
  - Documentation
  - Nothing

Verification of Linux kernel

- Absence of typical errors
  - Linux Driver Verification [out of scope]
- Functional properties (sequential)
  - **Informal specification (ISO/IEC 15408)**
  - Documentation
  - Nothing

ISO/IEC 15408-2013 Common Criteria



ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р ИСО/МЭК 15408-1— 2012

Информационная технология

МЕТОДЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ. КРИТЕРИИ ОЦЕНКИ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Часть 1

Введение и общая модель

ISO/IEC 15408-1:2009
Information technology — Security techniques — Evaluation criteria for IT security — Part 1: Introduction and general model

(IDT)

Издание официальное

Москва
Стандартинформ
2014

ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р ИСО/МЭК 15408-2— 2013

Информационная технология

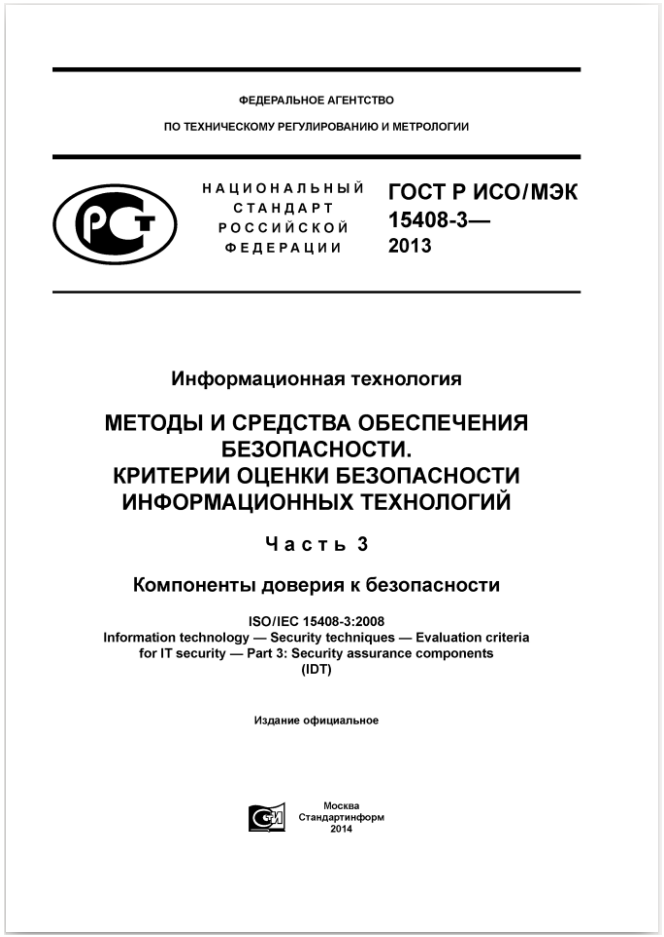
МЕТОДЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ. КРИТЕРИИ ОЦЕНКИ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Часть 2 Функциональные компоненты безопасности

ISO/IEC 15408-2:2008
Information technology — Security techniques — Evaluation criteria for IT security — Part 2. Security functional components
(IDT)

Издание официальное

Москва
Стандартинформ
2014



ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р ИСО/МЭК 15408-3— 2013

Информационная технология

МЕТОДЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ. КРИТЕРИИ ОЦЕНКИ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Часть 3

Компоненты доверия к безопасности

ISO/IEC 15408-3:2008
Information technology — Security techniques — Evaluation criteria for IT security — Part 3: Security assurance components
(IDT)

Издание официальное

Москва
Стандартинформ
2014

ISP RAS

Assurance components (ISO/IEC 15408-3-2013)

Security target

- **ASE_INT** Introduction
- **ASE_CCL** Conformance claim
- **ASE_SPD** Security problem definition
- **ASE_OBJ** Security objectives
- **ASE_ECD** Extended components definition
- **ASE_REQ** Security requirements
- **ASE_TSS** TOE summary specification

Development

- **ADV_SPM** Security policy model
- **ADV_FSP** Functional specification
- **ADV_TDS** TOE design
- **ADV_IMP** Implementation representation
- **ADV_ARC** Security architecture
- **ADV_INT** TOE internals

Life-cycle

- **ALC_CMC** Configuration management capabilities
- **ALC_CMS** Configuration management scope
- **ALC_DEL** Delivery
- **ALC_DVS** Development security
- **ALC_FLR** Flaw remediation
- **ALC_LCD** Life-cycle definition
- **ALC_TAT** Tools and techniques

Composition

- **ACO_COR** Composition rationale
- **ACO_DEV** Development evidence
- **ACO_REL** Reliance of dependent component
- **ACO_CTT** Composed TOE testing
- **ACO_VUL** Composition vulnerability analysis

Testing

- **ATE_COV** Coverage
- **ATE_FUN** Functional testing
- **ATE_DPT** Depth
- **ATE_IND** Independent testing

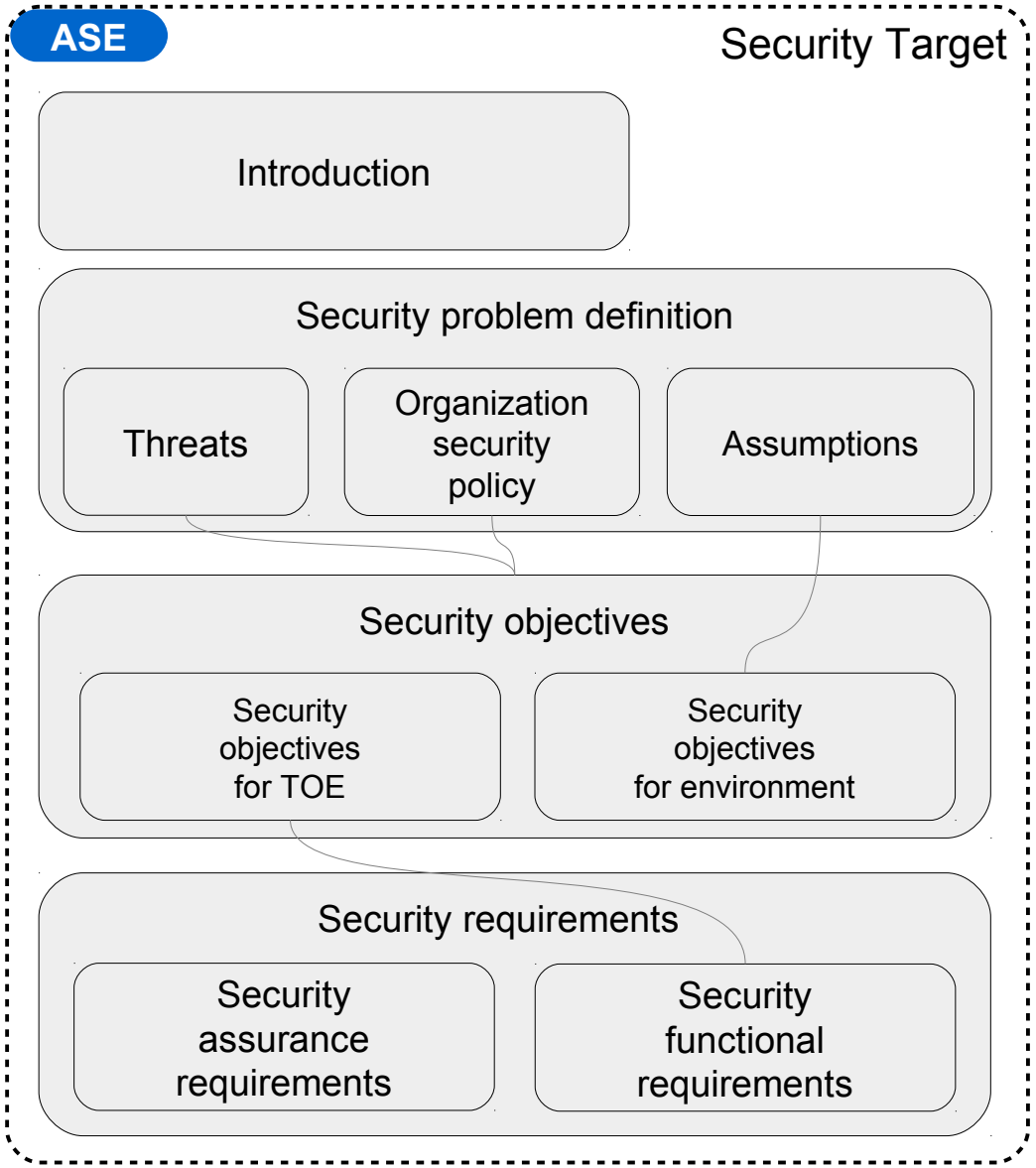
Vulnerability analysis

- **AVA_VAN** Vulnerability analysis

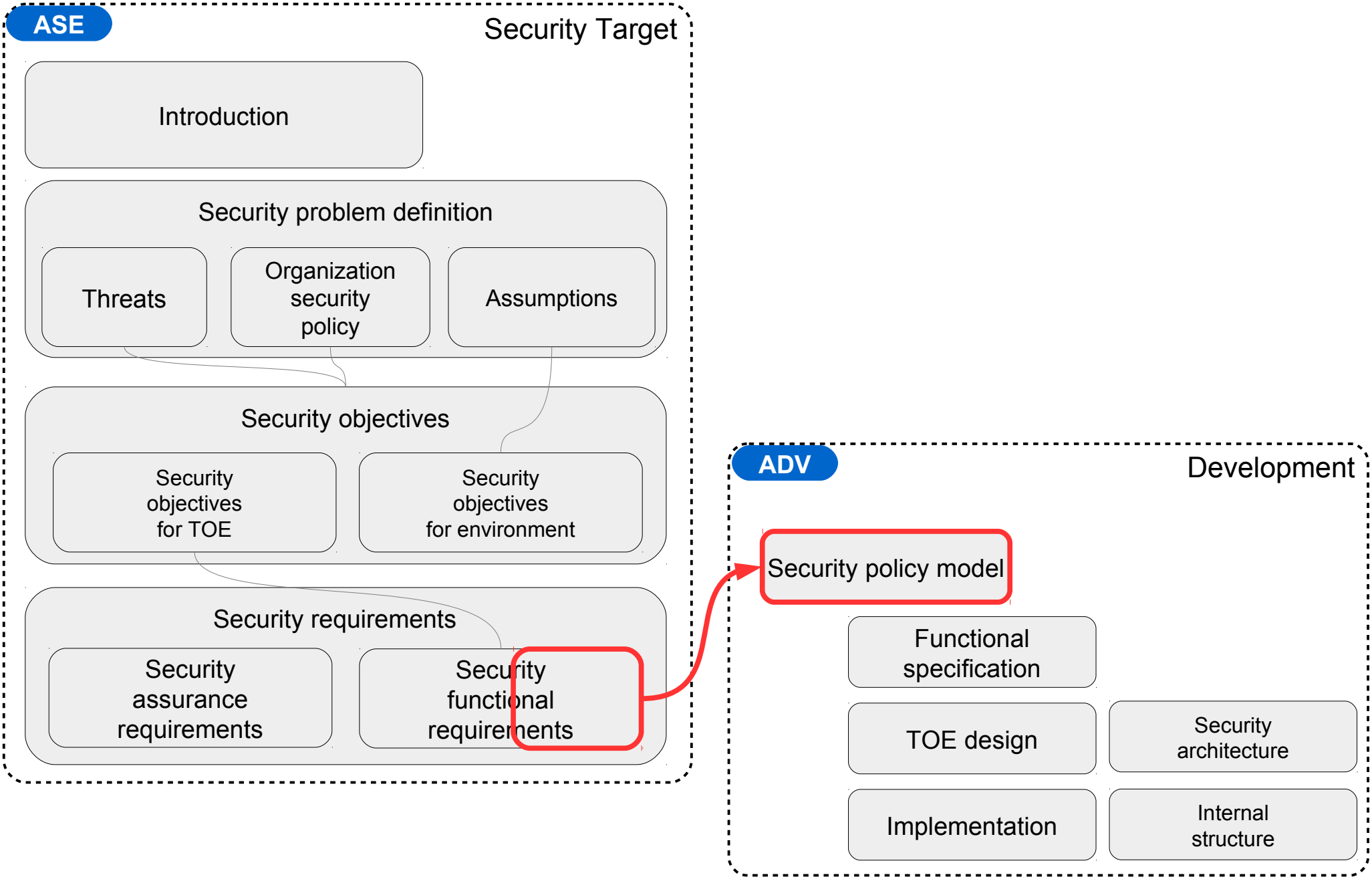
Guidance documents

- **AGD_OPE** Operational user guidance
- **AGD_PRE** Preparative procedures

ASE – Security Target



ADV_SPM — Security Policy Model



Astra Linux Special Edition

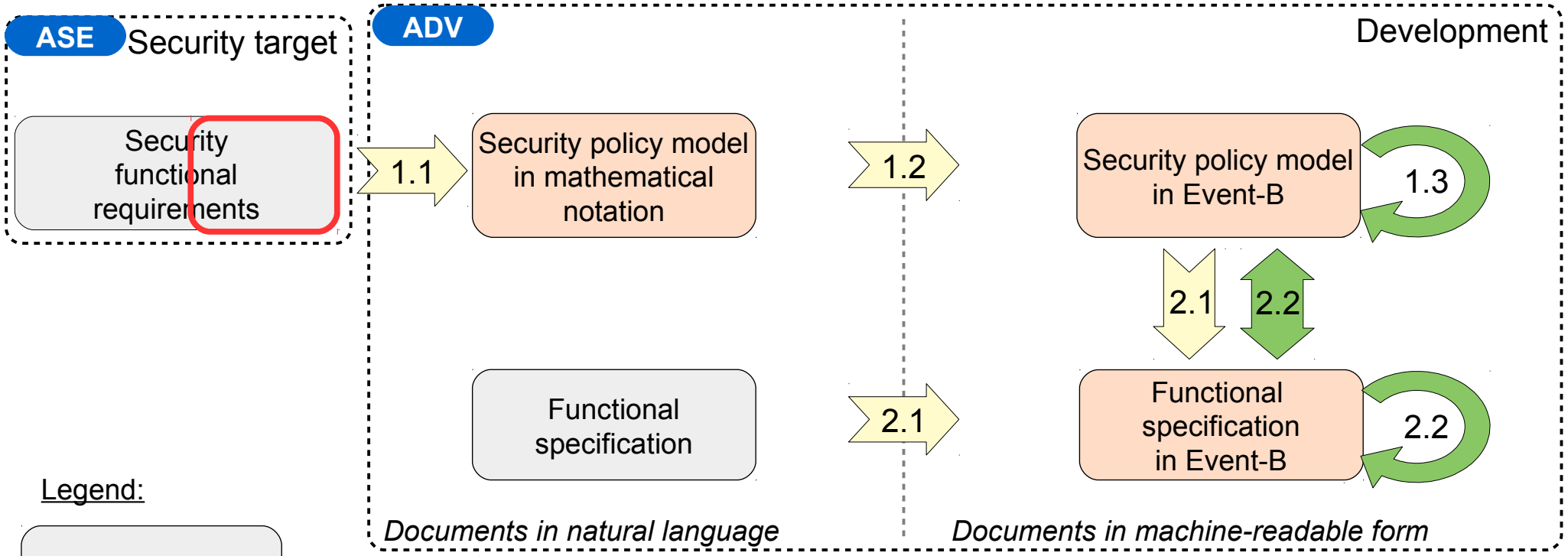
- Custom security policy model (MROSL-DP)
  - Lattice-based multi-level security (MLS)
  - Mandatory integrity control (MIC)
  - Role-based access control (RBAC)
- Custom Linux Security Module (LSM) implementation
  - parsec LSM

MROSL-DP Model

- Lattice-based multi-level security (MLS)
  - No read access if !(seclabel(subj) ≥ seclabel(obj))
  - No write access if seclabel(subj) ≠ seclabel(obj)
- Mandatory integrity control (MIC)
  - No write access if integrity(subj) < integrity(obj)
- Role-based access control (RBAC)

~150 pages in mathematical notation

Process of Modeling and Verification of Access Policy Control



Legend:

Input data for the process

Result of the process

Manual development

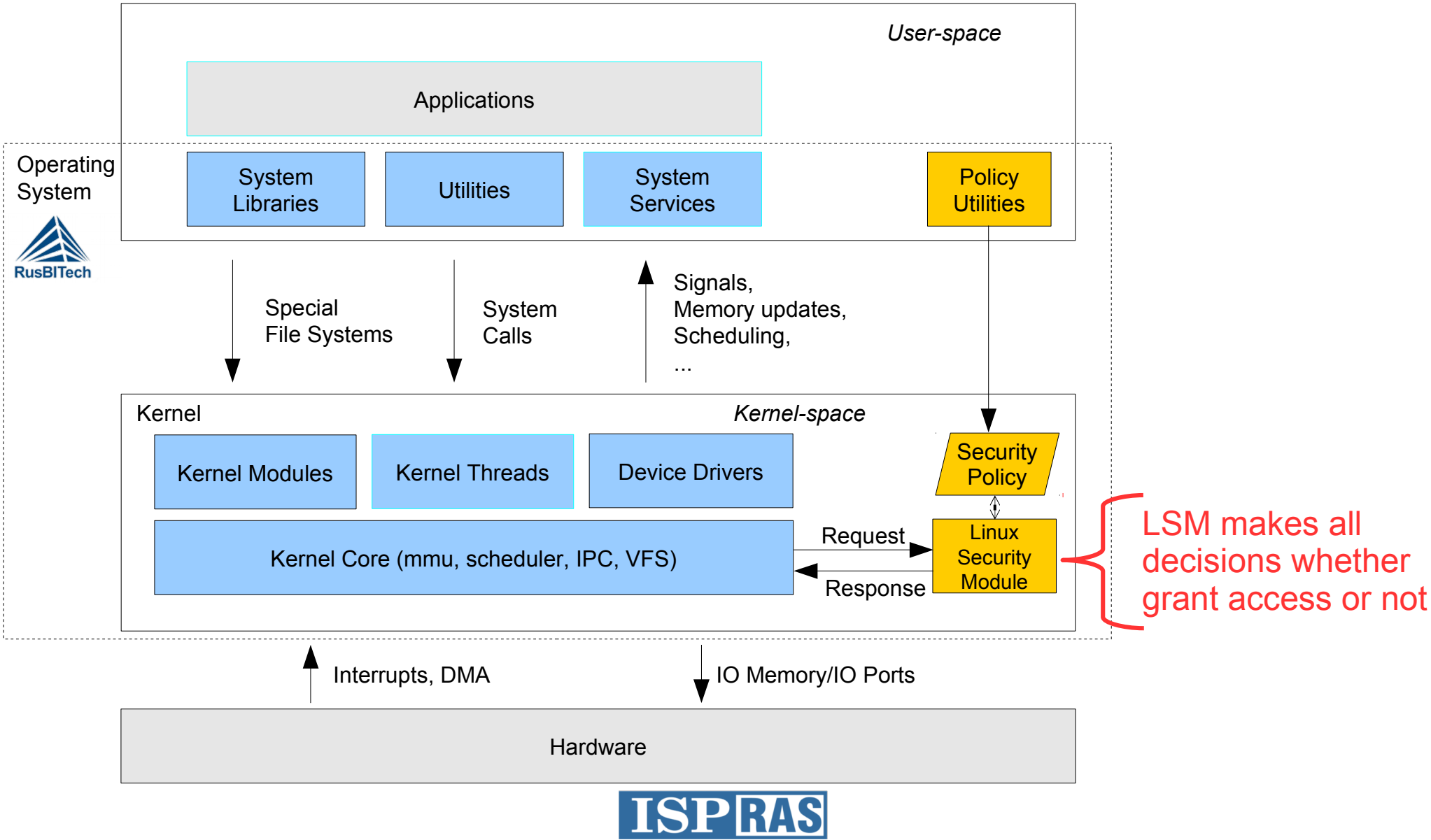
Automated verification

- 1. Security policy modeling
  - 1.1 Security policy modeling in mathematical notation
  - 1.2 Formalizing SPM in Event-B
  - 1.3 Verification of SPM in Event-B
- 2. Functional specification
  - 2.1 Formalization of FSP
  - 2.2 Verification of FSP

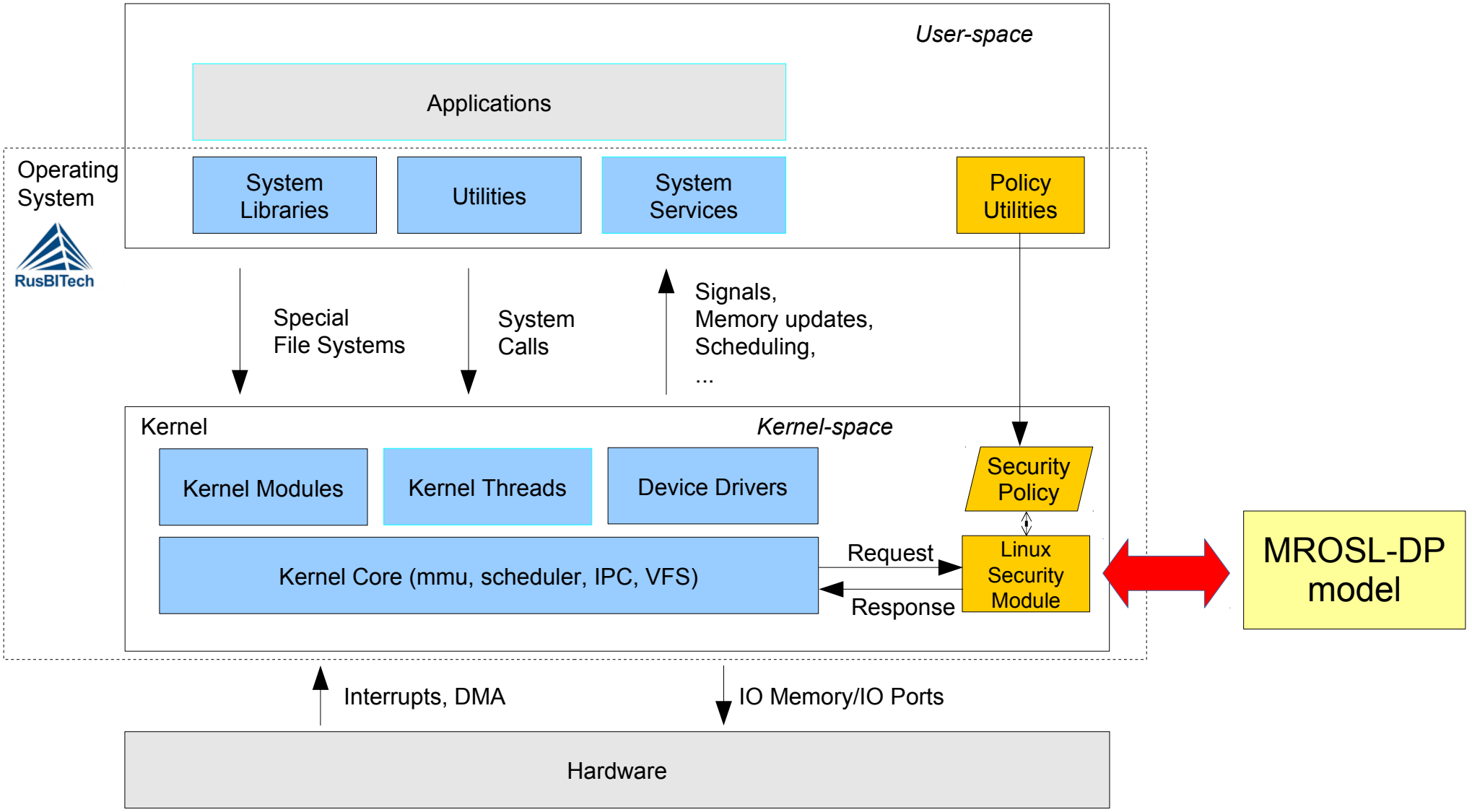
Formal MROSL-DP Model (Event-B)

- Constants: 34
- Axioms: 30
- Variables: 60
- Invariants: 248
- Events: 75
- Refinement levels: 4
- Size: 4393 LoC
- Proof obligations: 2962
- from ~150 pages in mathematical notation

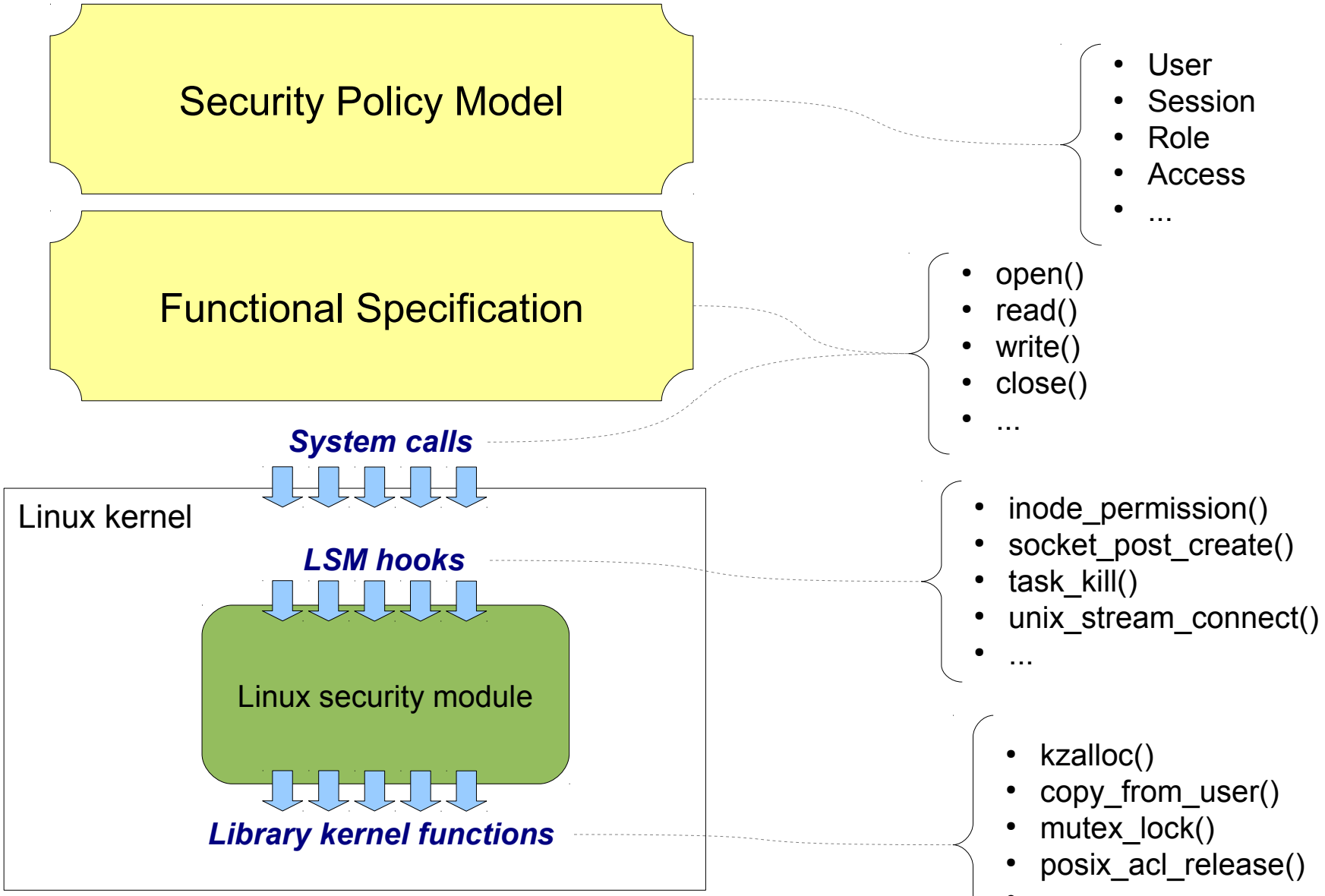
Key Component – Linux Security Module



Key Component – Linux Security Module



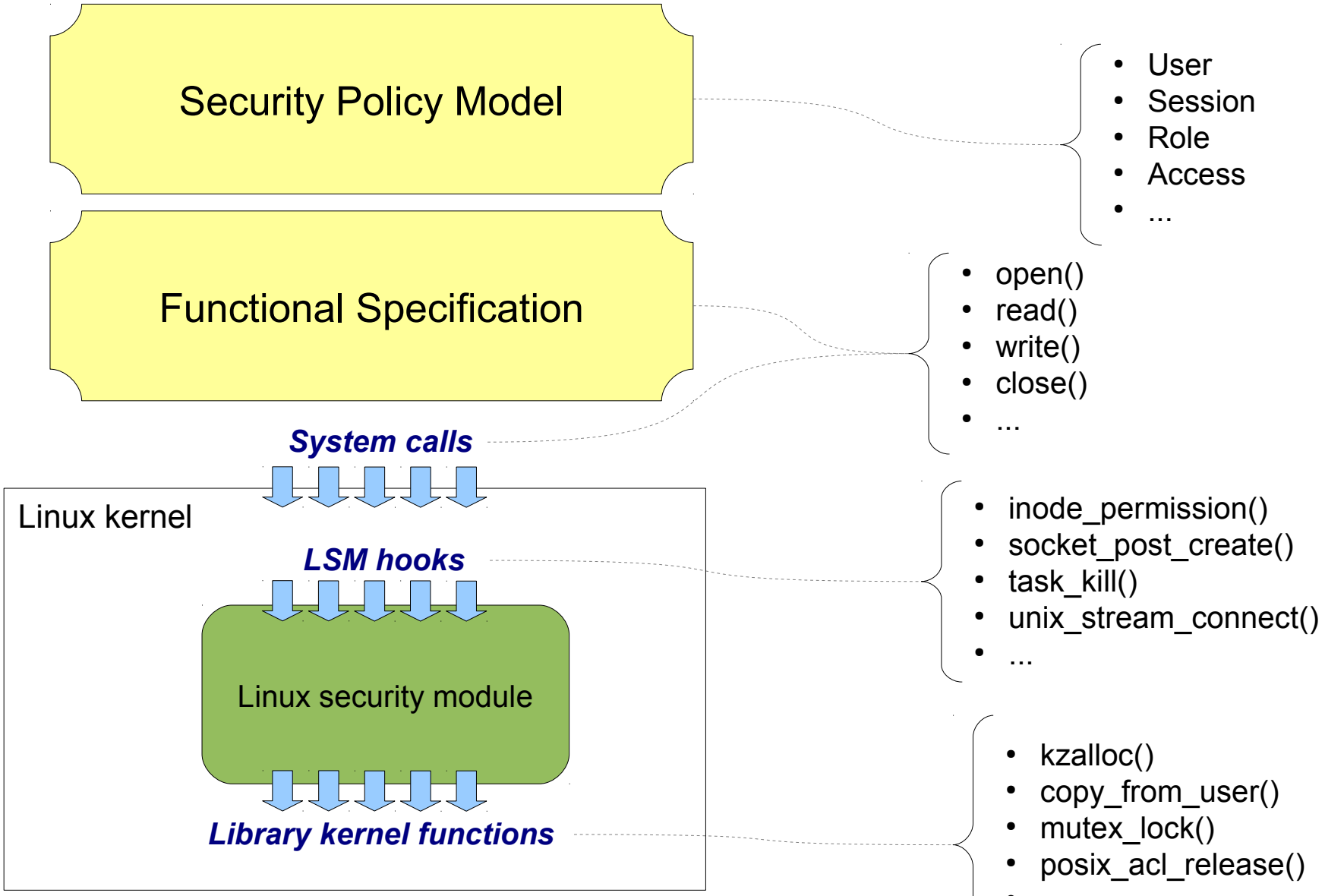
MROSL-DP and Linux Security Module – The Gap



MROSL-DP and Linux Security Module – The Gap

```
/*@ requires valid_seclabel(s);
    requires valid_seclabel(o);
    requires valid_mode(mode);
    assigns \nothing;
    behavior System:
       ensures \result == 0
            || \result == -EPERM;

    behavior Model:
       ensures \result == 0 <==> AccessIsPermitted(i2m_Label(s),
                                                   i2m_Label(o),
                                                   mode);
 */
int access_is_permitted(const seclabel_t *s, const seclabel_t *o, int mode)
```

MROSL-DP and Linux Security Module – The Gap



Project Settings

- Target code: Custom Linux Security Module
  - + Small: 3 KLoC
  - + Hardware independent
  - - Sometimes verification unfriendly
  - - Out of our control
- Properties to prove:
  - Absence of run-time errors
  - Compliance to MROSL-DP functional specifications
- Assumptions
  - Linux kernel core conforms with its specifications
    - It is not target to prove
  - No concurrent access to data

Verification of Linux kernel

- Absence of typical errors
  - Linux Driver Verification [out of scope]
- Functional properties (sequential)
  - Informal specification (ISO/IEC 15408)
  - **Documentation**
  - **Nothing**

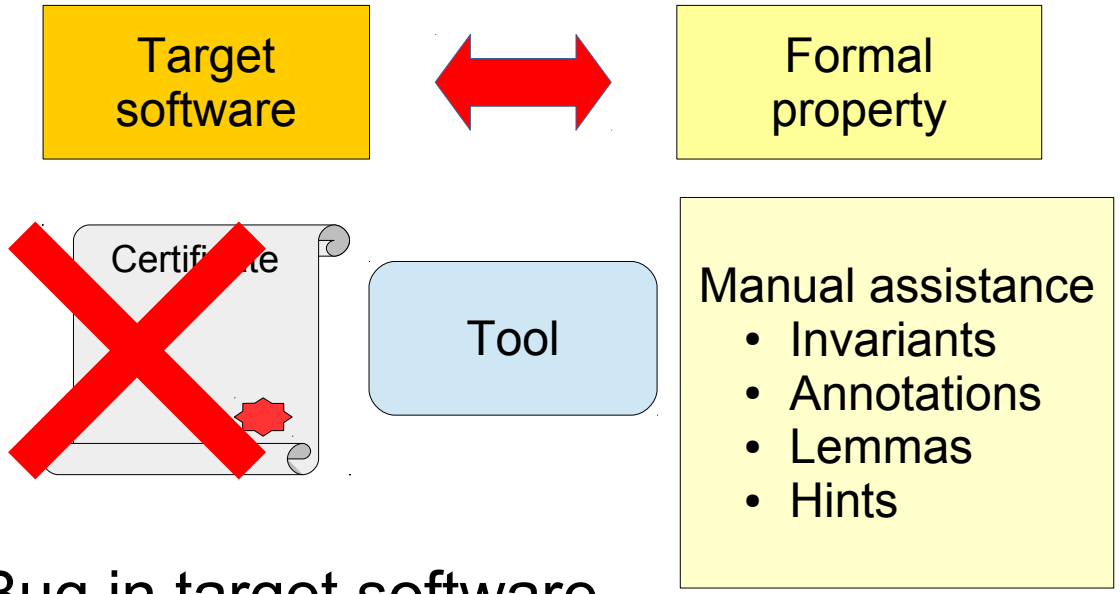
VerKer - Linux kernel library functions

- Target code: Linux kernel library functions
  - + Small and «simple» functions
  - + Hardware independent
  - - Sometimes verification unfriendly
  - - Out of our control
- Properties to prove:
  - Absence of run-time errors
  - Compliance to functional specifications (as strict as possible)
- Assumptions
  - No concurrent access to data
- Public repository
  - https://forge.ispras.ru/projects/verker
  - Lead by Denis Efremov

Tools

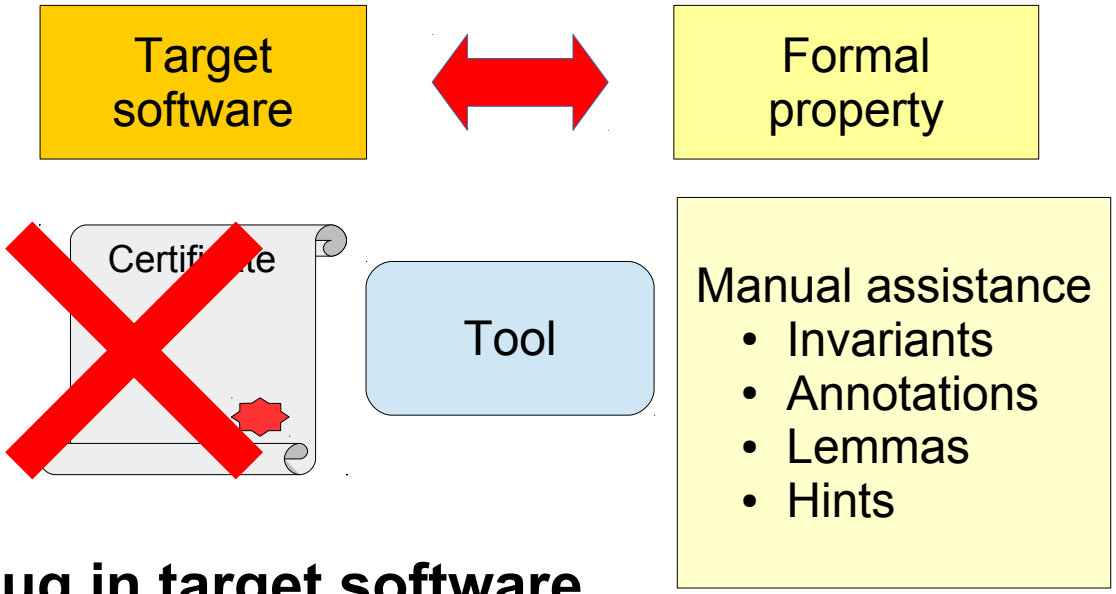
- Frama-C + Jessie2 + Why3

Program Analysis



- Bug in target software
- Bug in formal property
- Problems with the tool

Program Analysis



- **Bug in target software**
- Bug in formal property
- Problems with the tool

Bugs in target software

// returns a number of nonempty elements of range array

```
int clean_sort_range(struct range *range, int az)
{
        int i, j, k = az - 1, nr_range = 0;

        for (i = 0; i < k; i++) {
                if (range[i].end)
                        continue;
                for (j = k; j > i; j--) {
                        if (range[j].end) {
                                k = j;
                                break;
                        }
                }
                if (j == i)
                        break;
                range[i].start = range[k].start;
                range[i].end   = range[k].end;
                range[k].start = 0;
                range[k].end   = 0;
                k--;
        }
        /* count it */
        for (i = 0; i < az; i++) {
                if (!range[i].end) {
                        nr_range = i;
                        break;
                }
        }

        /* sort them */
        sort(range, nr_range, sizeof(struct range), cmp_range, NULL);

        return nr_range;
}
```

// number of nonempty elements is evaluated by
// finding the first empty element of the array

Bugs in target software

commit 834b40380e93e36f1c9b48ec1d280cebe3d7bd8c
Author: Alexey Khoroshilov <khoroshilov@ispras.ru>
Date: Thu Nov 11 14:05:14 2010 -0800

kernel/range.c: fix clean_sort_range() for the case of full array

clean_sort_range() should return a number of nonempty elements of range array, but if the array is full clean_sort_range() returns 0.

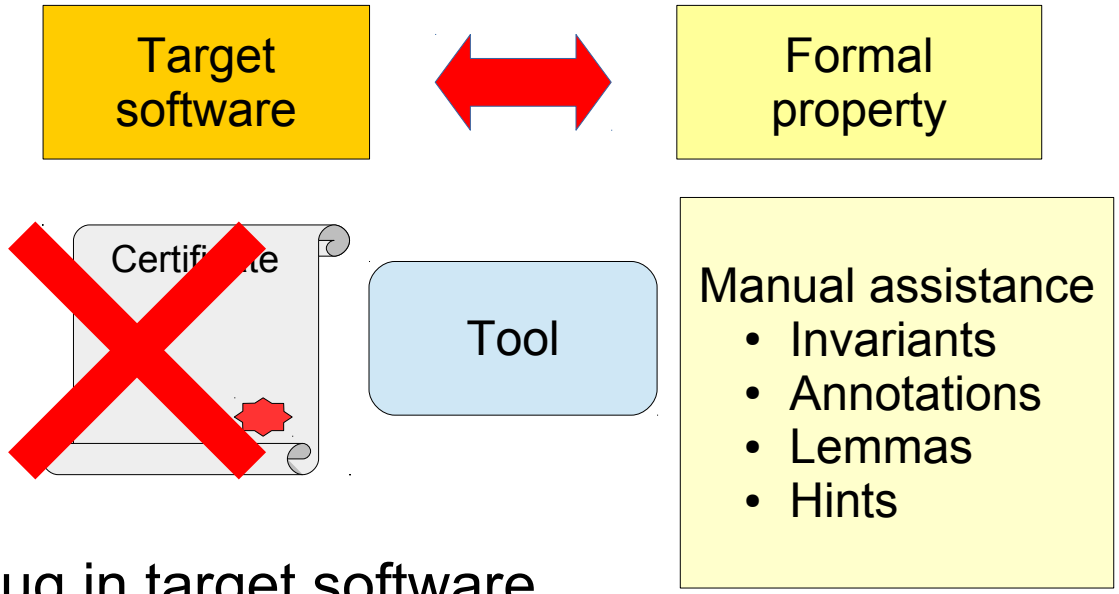
The problem is that the number of nonempty elements is evaluated by finding the first empty element of the array. If there is no such element it returns an initial value of local variable nr_range that is zero.

The fix is trivial: it changes initial value of nr_range to size of the array.

The bug can lead to loss of information regarding all ranges, since typically returned value of clean_sort_range() is considered as an actual number of ranges in the array after a series of add/subtract operations.

Signed-off-by: Alexey Khoroshilov <khoroshilov@ispras.ru>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>

Program Analysis



- Bug in target software
- **Bug in formal property**
- Problems with the tool

Bugs in formal property

"ACSL By Example Version 14.1.0", Listing 3.32 (The logic function Count):

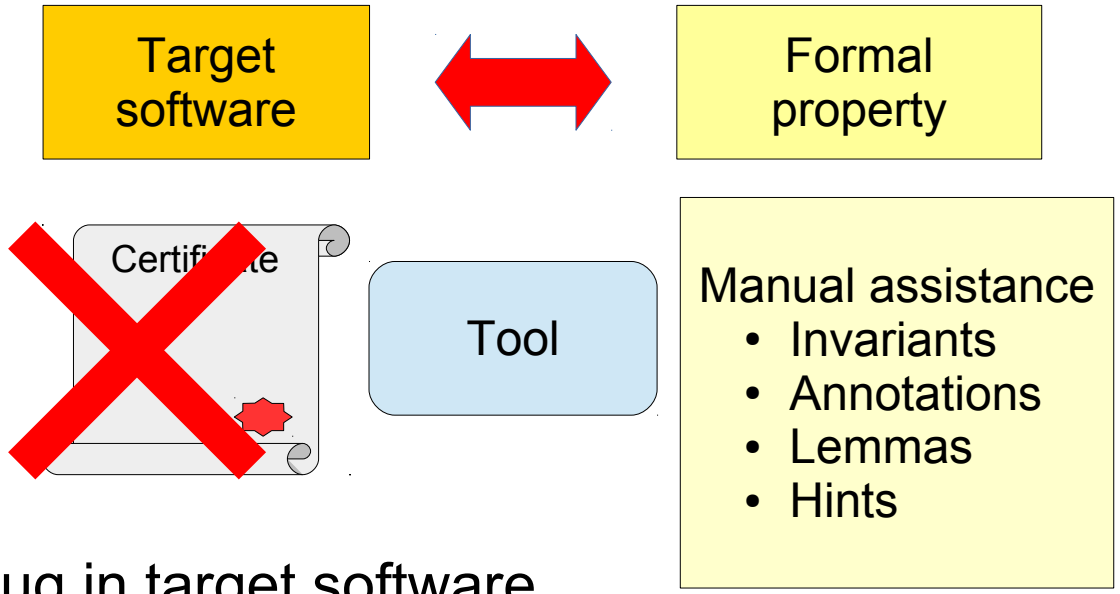
```
axiomatic Count {
  logic integer Count{L}(value_type *a, integer m, integer n, value_type v) reads a[m..n-1];

axiom CountSectionEmpty:
   \forall value_type *a, v, integer m, n; n <= m ==> Count(a, m, n, v) == 0;

axiom CountSectionHit:
   \forall value_type *a, v, integer n, m;
      a[n] == v ==> Count(a, m, n + 1, v) == Count(a, m, n, v) + 1;
...
}
```

Contradiction:

```
value_type a = 5;
assert Count(&a + 1, 0, -1, (value_type) 5) == 0;
assert Count(&a + 1, 0, 0, (value_type) 5) == 0;
assert Count(&a + 1, 0, 0, (value_type) 5) == Count(&a + 1, 0, -1, (value_type) 5) + 1);
assert 0 == 1;
```

Found by Denis Efremov, Mikhail Mandrykin

Program Analysis



- Bug in target software
- Bug in formal property
- **Problems with the tool**

Unmodified Linux kernel code

```
36  /**
37   * strncasecmp - Case insensitive, length-limited string comparison
38   * @s1: One string
39   * @s2: The other string
40   * @len: the maximum number of characters to compare
41   */
42  int strncasecmp(const char *s1, const char *s2, size_t len)
43  {
44          /* Yes, Virginia, it had better be unsigned */
45          unsigned char c1, c2;
46
47          if (!len)
48                  return 0;
49
50          do {
51                  c1 = *s1++;
52                  c2 = *s2++;
53                  if (!c1 || !c2)
54                          break;
55                  if (c1 == c2)
56                          continue;
57                  c1 = tolower(c1);
58                  c2 = tolower(c2);
59                  if (c1 != c2)
60                          break;
61          } while (--len);
62          return (int)c1 - (int)c2;
63  }
```

Unmodified Linux kernel code (2)

```
489  #ifndef __HAVE_ARCH_STRNLEN
490  /**
491   * strnlen - Find the length of a length-limited string
492   * @s: The string to be sized
493   * @count: The maximum number of bytes to search
494   */
495  size_t strnlen(const char *s, size_t count)
496  {
497          const char *sc;
498
499          for (sc = s; count-- && *sc != '\0'; ++sc)
500                  /* nothing */;
501          return sc - s;
502  }
503  EXPORT_SYMBOL(strnlen);
504  #endif
```

Problems with the tool

- Low level memory operations
  - Arithmetics with pointers to fields of structures (container_of)
  - Prefix structure casts
  - Reinterpret casts
- Integer overflows and bit operations
- Complex functionality requires manual proof
  - Lemma functions
- Limited code support
  - Functional pointers
  - String literals
- Scalability problems
- Usability problems

AstraVer Plugin *(Mikhail Mandrykin)*

- Reinterpretation support for pointers to integral types,
  - merging array reinterpretation is only supported for divisible sizes
- Jessie theory/module split
  - Automatic theory/module dependency computation per code function
- New model (theories and modules) for integral types
  - Better support for bitwise and wrap-around operations
- Three-staged typing of annotations
  - Arbitrary order of logic definitions, mutual recursion
- A number of small extra features
  - Relevant code extraction (annotated functions with dependencies)
  - Function pointer support through exhaustive check for may-aliases
  - Rewriting of variadic functions through additional array argument
  - Template annotations for memcpy(), memmove(), memcmp(), …

Done (current plugin)

- No support for input languages beyond C+ACSL (Java, OCaml)
- No annotation inference
- No bitvector memory regions
- No automatic frame condition generation for logic functions

Dropped

- Reimplementation of the plugin based on dynamic frames and interpreted finite sets
  - Customized bounded instantiation strategy for lemmas and frame axioms
  - Theory of finite sets
  - Translation to formulas in stratified sort fragment
  - Counterexample model reconstruction

Imple-mented

- New path-sensitive region, effect and frame inference
- Translation to new intermediate representation (new Frama-C plugin)
- ACSL extensions: lemma functions, logic context management, region annotations,...

TODO (new plugin)

Recent Experiments

Lemma Functions

```
/*@ ghost
  @ /@ ensures \result == a + b;
  @   @ ensures \result == (a ^ b) + (a & b) * 2;
  @   @ lemmafn \true;
  @   @/
  @ unsigned long long sum_as_xor_plus_and(unsigned a, unsigned b)
  @ {
  @   unsigned long long result = (a ^ b) + ((unsigned long long) (a & b) <<
   ↪ 1ULL);
  @   unsigned long long result_2 = (unsigned long long) a + b;
  @   return result;
  @ }
  @*/
```

Recent Experiments

Abstract Axiomatics

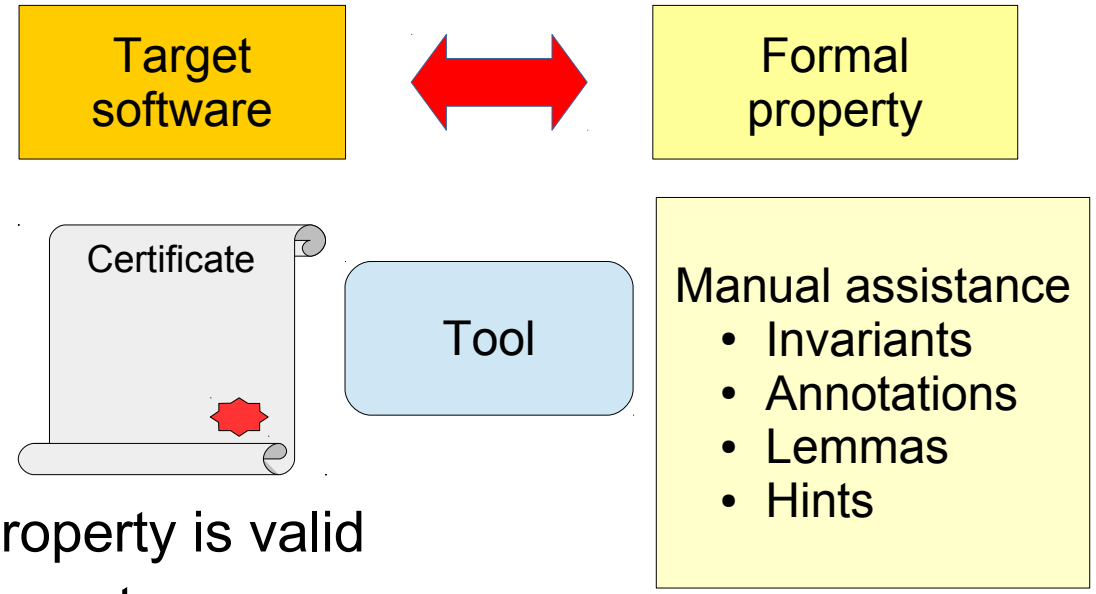
```
/*@
axiomatic Count {
  type index;
  type elem;
  type map;
  logic elem get(map m, index i);
  logic map c_set(map m, index i, elem e);
  logic map empty;
  logic boolean p(elem e);
  logic integer count_p(map p);
  axiom Count_emp: count_p(empty) == 0;
  axiom Count_more: \forall map m, index i, elem e;
    !p(get(m, i)) && p(e) ==> count_p(c_set(m,i,e)) == count_p(m) + 1;
  axiom Count_less: \forall map m, index i, elem e;
    p(get(m, i)) && !p(e) ==> count_p(c_set(m,i,e)) == count_p(m) - 1;
  axiom Count_same: \forall map m, index i, elem e;
    p(get(m, i)) == p(e) ==> count_p(c_set(m,i,e)) == count_p(m);
  abstract axiom Get_emp: \forall index i; !p(get(empty, i));
} */
```

Recent Experiments

Abstract Axiomatics - Use

```
/*@ axiomatic Bitcount {
  logic unsigned zeros = 0;
  logic boolean get_bit (unsigned m, int i) =
     (m & (1 << (unsigned)i)) > 0;
  logic unsigned set_bit (unsigned m, int i, boolean e) =
    e ? m | (1 << (unsigned)i) : m & ~(1 << (unsigned)i);
  logic boolean is_set (boolean e) = e;
  logic integer bitcount(unsigned x);
  include Count \with
    type index = int, type elem = boolean, type map = unsigned,
    function empty = zeros, function get = get_bit,
    function c_set = set_bit, function p = is_set, function count_p = bitcount;
} */
```

Program Analysis



- up to:
  - Formal property is valid
  - Tool is correct
  - Tool assumptions are held

VerKer - Linux kernel library functions

- Status:
  - 25 of 37 functions are proved
    - some lemmas for logic functions requires manual hints
  - specifications with proof protocols are available

- check_bytes8
- memchr
- memcmp
- memscan
- skip_spaces
- strcasecmp
- strcat
- strchr
- strchrnul
- strcmp
- strcpy
- strcspn
- strlen
- strnchr
- strnlen
- strpbrk
- strrchr
- strsep
- strspn
- strlcpy
- memmove(*)
- memcpy
- memset
- kstrtobool
- _parse_integer_fixup_radix

Open Problems

- Specification

Open Problems

```
489  #ifndef __HAVE_ARCH_STRNLEN
490  /**
491   * strnlen - Find the length of a length-limited string
492   * @s: The string to be sized
493   * @count: The maximum number of bytes to search
494   */
495  size_t strnlen(const char *s, size_t count)
496  {
497          const char *sc;
498
499          for (sc = s; count-- && *sc != '\0'; ++sc)
500                  /* nothing */;
501          return sc - s;
502  }
503  EXPORT_SYMBOL(strnlen);
504  #endif
```

Open Problems

```
 2 /**
 3  * strnlen - Find the length of a length-limited string
 4  * @s: The string to be sized
 5  * @count: The maximum number of bytes to search
 6  */
 7
 8 /*@ requires valid_strn(s, count);
 9     assigns \nothing;
10     ensures \result == strnlen(s, count);
11     behavior null_byte:
12        assumes \exists integer i; 0 <= i <= count && s[i] == '\0';
13        ensures s[\result] == '\0';
14        ensures \forall integer i; 0 <= i < \result ==> s[i] != '\0';
15     behavior count_len:
16        assumes \forall integer i; 0 <= i <= count ==> s[i] != '\0';
17        ensures \result == count;
18     complete behaviors;
19     disjoint behaviors;
20  */
21 size_t strnlen(const char *s, size_t count)
22 {
23         const char *sc;
24         /*@ loop invariant 0 <= count <= \at(count,Pre);
25             loop invariant s <= sc <= s + strnlen(s,\at(count,Pre));
26             loop invariant sc - s == (\at(count,Pre) - count);
27             loop invariant valid_strn(sc, count);
28             loop invariant strnlen(s,\at(count,Pre)) == strnlen(sc, count) + (sc - s);
29             loop invariant \forall integer i; 0 <= i < sc - s ==> s[i] != '\0';
30             loop variant count;
31          */
32         for (sc = s; count--/*@%*/ && *sc != '\0'; ++sc)
33                 /* nothing */;
34
35         return sc - s;
36 }
```

Conclusions

- Legacy software
  - All possible code constructs
  - Reconstruct requirements
- Open source tools is crucial
- New assurance components for program analysis

Assurance components (ISO/IEC 15408-3-2013)

Security target

- **ASE_INT** Introduction
- **ASE_CCL** Conformance claim
- **ASE_SPD** Security problem definition
- **ASE_OBJ** Security objectives
- **ASE_ECD** Extended components definition
- **ASE_REQ** Security requirements
- **ASE_TSS** TOE summary specification

Development

- **ADV_SPM** Security policy model
- **ADV_FSP** Functional specification
- **ADV_TDS** TOE design
- **ADV_IMP** Implementation representation
- **ADV_ARC** Security architecture
- **ADV_INT** TOE internals

Life-cycle

- **ALC_CMC** Configuration management capabilities
- **ALC_CMS** Configuration management scope
- **ALC_DEL** Delivery
- **ALC_DVS** Development security
- **ALC_FLR** Flaw remediation
- **ALC_LCD** Life-cycle definition
- **ALC_TAT** Tools and techniques

Composition

- **ACO_COR** Composition rationale
- **ACO_DEV** Development evidence
- **ACO_REL** Reliance of dependent component
- **ACO_CTT** Composed TOE testing
- **ACO_VUL** Composition vulnerability analysis

Testing

- **ATE_COV** Coverage
- **ATE_FUN** Functional testing
- **ATE_DPT** Depth
- **ATE_IND** Independent testing

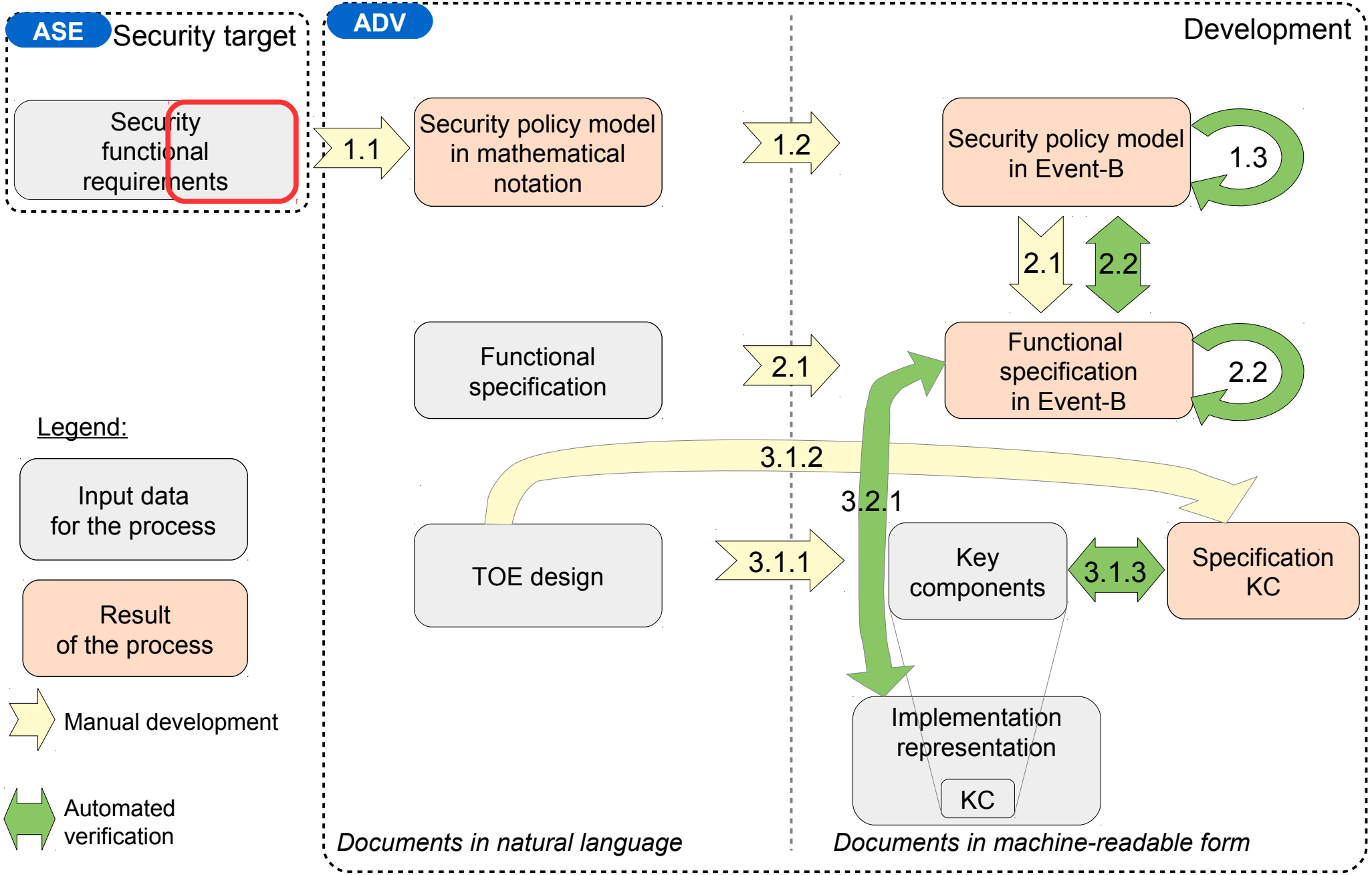
Vulnerability analysis

- **AVA_VAN** Vulnerability analysis

Guidance documents

- **AGD_OPE** Operational user guidance
- **AGD_PRE** Preparative procedures

Process of Modeling and Verification of Access Policy Control



LinuxTesting.org

Thank you!

http://linuxtesting.org/astraver

ISP RAS

Ivannikov Institute for System Programming of the Russian Academy of Sciences