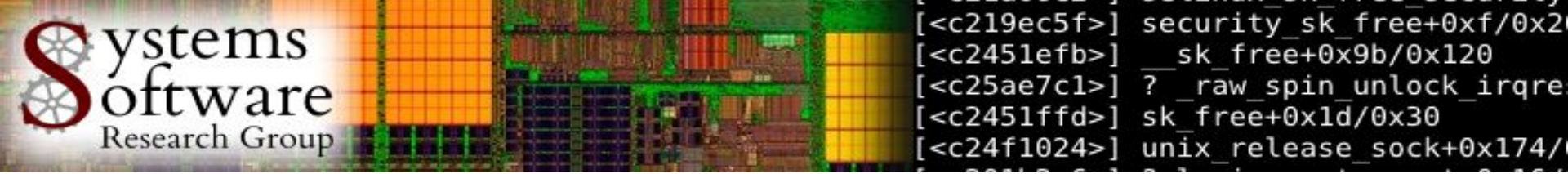


ece Bradley Department of Electrical & Computer Engineering

Renee 1.0

Scalable Translation Validation of Unverified Legacy OS Code

Amer Tahat, Sarang Joshi, Pronoy Gawsamy, Binoy Ravindran
Presenter: Amer Tahat
System Software Research Group-SSRG
Department of Electrical and Computer Engineering
Virginia Tech University

Acknowledgments: This is work is supported in part by ONR (Office of Naval Research) under grant number This work was supported in part by ONR under grant N00014-18-1-2665. We are also very grateful to Dr. Natarajan Shankar and Sam Owr from SRI for providing us with pvs7-dev and its patches that helped us to use PVS7 in our work.

Question

Is there any feasible methodology to produce a trustworthy **formal** model of a large OS? What about multiple OSes?

Grand Challenges

1. They may not have the source code available (only the binary).
2. They may not have the formal semantic of the high source code - possibly written in multiple languages (if the source is available).
3. Gap between formal model and the code. Expensive?
4. Large number of LOC, developers, and a complex life cycle.
5. Smaller number of formal verification engineers.

Related Work

1. **SeL4,** assumes that complete high-level source code of the OS is available to the verifier in a subset of the C language, called C0[1,2]
2. **CompCert,** presents the formal proof for a compiler, but restricts it to a subset of C called C-light[8].
3. **TAL**, presents a verification toolchain that targets a typed assembly language, which is transformed into a typed machine language to generate a safe binary.
4. **Hyperkernel***, an approach for designing a new OS kernel from scratch that is verifiable using SMT solvers, but the approach scopes out verifying legacy operating system [9].
5. [10] establishes that seL4's binary code is equivalent to its C 0 source, but is restricted to the already verified seL4's C0 code
6. **ARM in HOL** (2006-2010) [12,14], ARM in HOL [2011 - 2016] [13,15].

Hyperkernel* Best paper award in On Symposium on Operating Systems, Principles (SOSP17).

Related Work Cont'd

ASL : ARM Specification Language 2016 (Trustworthy and **Machine Readable**).

- A. Reid, "Trustworthy specifications of arm v8-a and v8-m system level architecture," in 2016 Formal Methods in Computer-Aided Design, FMCAD 2016.

Applications	Translation into many theorem provers, smt solvers other external specification languages ASL into SAIL [then into multiple theorem provers] [spisa19].

https://alastairreid.github.io/specification_languages/ (More about ASL)

Renee toolchain for the formalization of arm binary code

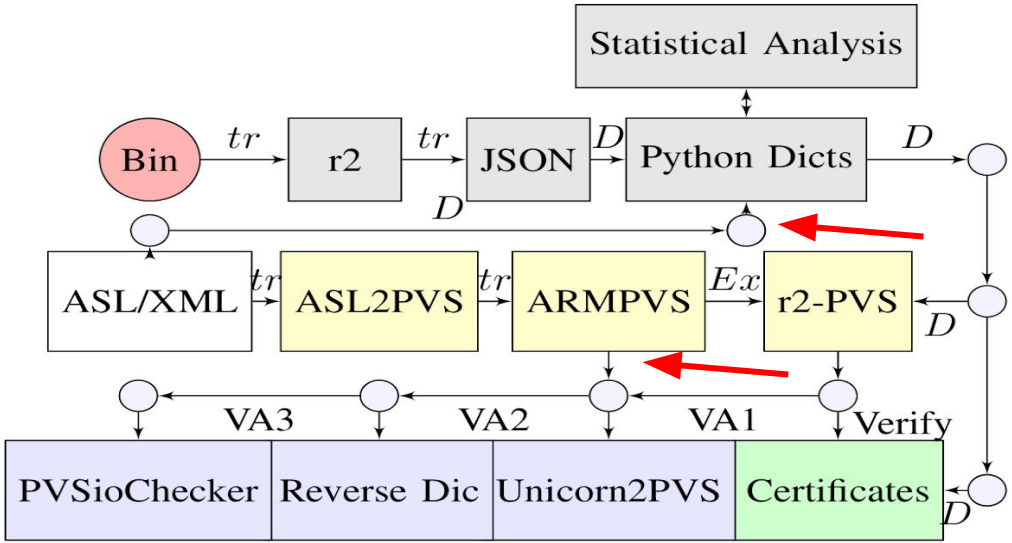


Fig. 1: Toolchain Workflow. $r2$= radare2, tr= Translation, D= Data transfer, Ex= Theories Export, VA= Validation.

ASL for Renee

Assisted us in many ways :

- Translating the instructions into PVS7,
- Generating Tests to validate ASl2PVS7 tr,
- Building a decoder, and an encoder from/to the theorem prover and radare2.

PVS7-Dev a game Changer

PVS7-Dev Background

- Theory parameters; e.g;:
  - bv: Theory [n : Nat] , n is visible in theory
- Dependant types
  - bvec[n] : Type = [below(n) -> bit]
- Generic Theories
  - (OOF- Object Oriented Formalization)

PVS7-Dev Theory Declaration

Ex: Let A be an abstract PVS theory with two bit vectors attributes; called a1 and b1.

We can declare:

B : Theory = A with {{ a1 := bv[2](0b01) }}

C : Theory = A with {{ a1 := bv[3](0b101),
b1:= bv[2](0b10)}}

Renee's Core Formalization Idea

- Every byte code in the target can be represented -in PVS7- as an instance of an abstract instruction's Theory (translated from ASL-XML file)!

From ASL to PVS7

RSL: PVS7 Instructions Theories

Ands_log_shift: Theory [(importing armstate) p : arm-state]

Works as a pre-state

BEGIN

Diag : bv[64] // will be instantiated by Translator with a bit vector

Decoding part

```
|31  |30:29|  28:24|  23:22|21|20:16|15:11|10:5|4:0|
|sf  |0  0 |  01010|  shift| 0|Rm    | imm6|  Rn |Rd |
      opc                    N
```

Listing 1: ASL-XML ands-log-shift bits diagram

```
v:diag=(#  sf:=bt(Diag,31),opc:=  bts(Diag,29,30),
Fixed1:=  bts(Diag,24,28),  shift:=  bts(Diag,22,23),
N:=  bt(Diag,21)            , Rm:=  bts(Diag,16,20),
imm6:=  bts(Diag,10,15)    , Rn:=  bts(Diag,5,9),
Rd:=  bts(Diag,0,4)  #)
```

Listing 2 PVS ands-log-shift diagram load

Addr : bv[64]

~ 1-1 Formalization ASL into PVS7

```
sts3: ASL(p)= sts2 with [operand1:= p.X(n)]
sts4: ASL(p)= sts3 with [operand2:=ShiftReg(64,p.X(
   m), shift_type, shift_amount)]
sts5: ASL(p)= if invert then sts4 with
                 [operand2:= NOT(sts4.operand2)]
              else sts4  endif
sts6: ASL(p) = Cond sts4.op = LogicalOp_AND -> sts5
    with [result := AND (sts5.operand1, sts5`
    operand2)],
                   sts4.op = LogicalOp_ORR ->
    sts5 with [result := OR  (sts5.operand1, sts5.
    operand2)],
                   sts4.op = LogicalOp_EOR ->
    sts5 with [result := XOR (sts5.operand1, sts5.
    operand2)] EndCond
% post state
p1:s = if sts6.setflags then p with [.PSTATE.NZCV:=
  let result_63 = field(64, sts6.result, 63 ,63)
  in bv[2](0b00) o IsZeroBit(64,sts6.result) o
    result_63]
       else p  endif
post: s = p1 with [.X(d) := sts6.result]
```

Listing 4: PVS ands-log-shift Operational

```
bits(datasize) operand1 = X[n];
bits(datasize) operand2 = ShiftReg(m,
                    shift_type, shift_amount);
if invert then
            operand2 = NOT(operand2);
case op of when LogicalOp_AND
                    result = operand1 AND operand2;
          when LogicalOp_ORR
                    result = operand1 OR operand2;
          when LogicalOp_EOR
                    result = operand1 EOR operand2;
%Post state
if setflags then PSTATE.<N,Z,C,V> =
       result<datasize-1>:IsZeroBit(result):'00';
X[d] = result;
```

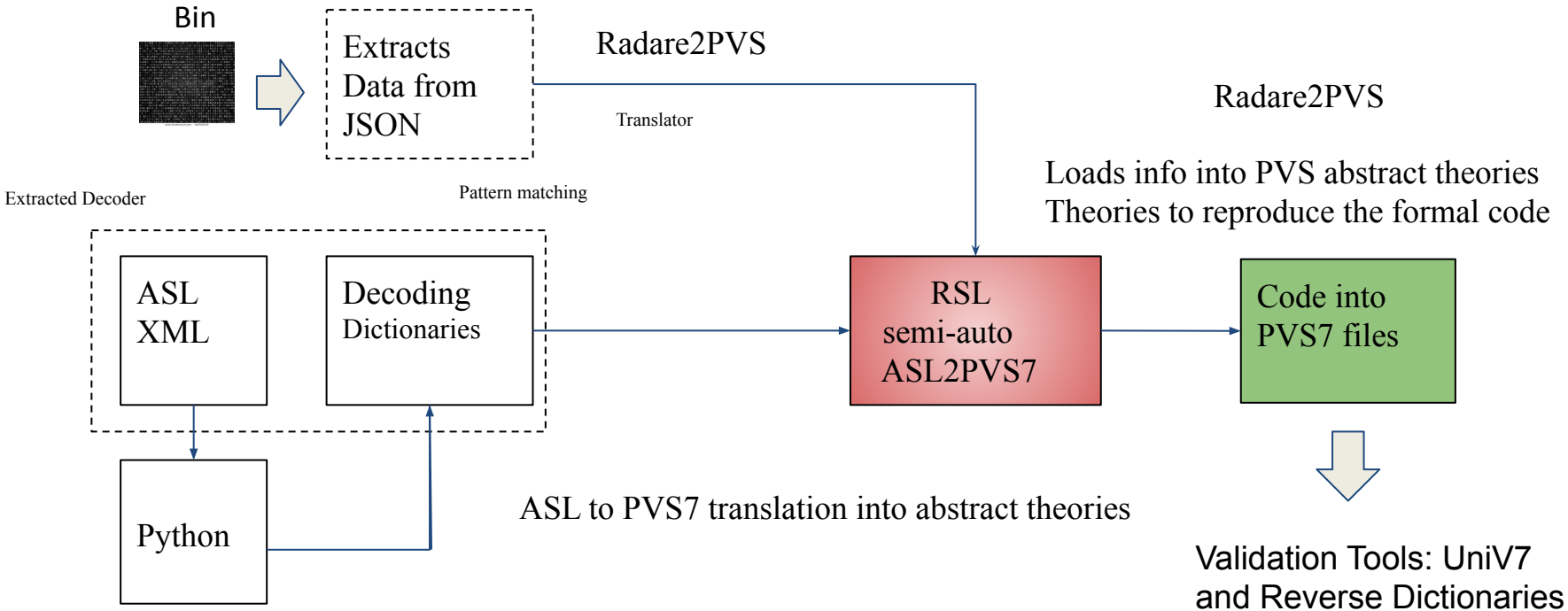
Listing 5: ASL ands-log-shift Operational

Post state

Operational part

From radare2 to PVS7

Translation Process

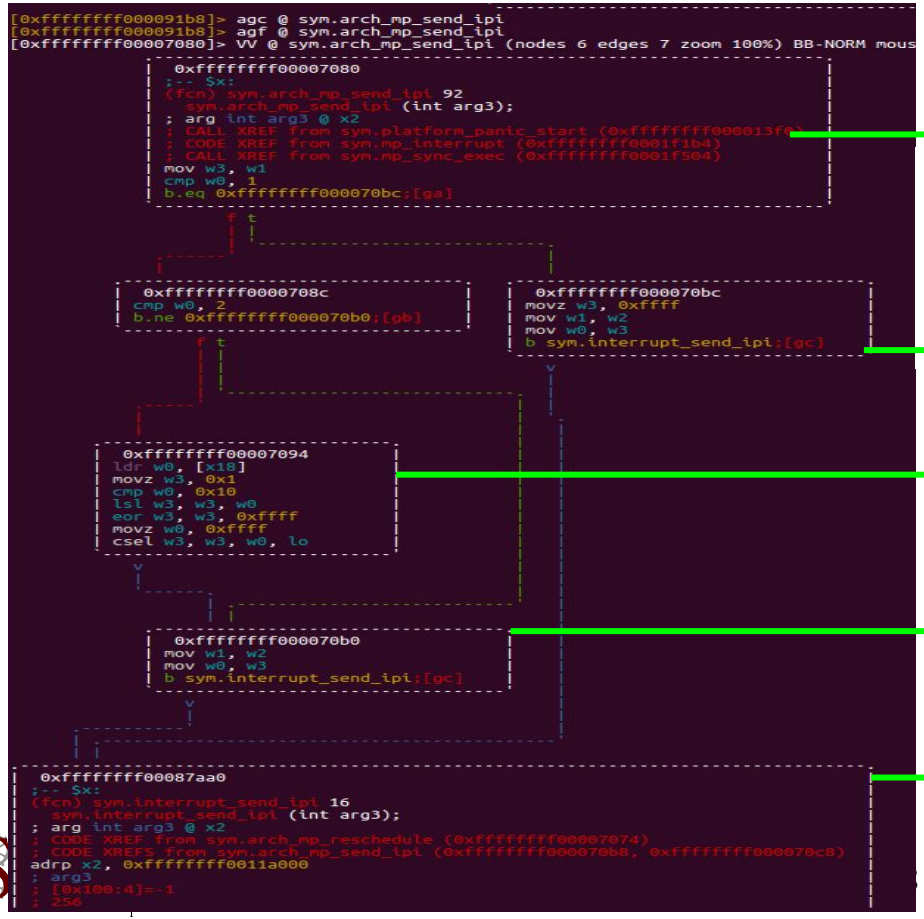


Radare2PVS7: Basic Block Tr

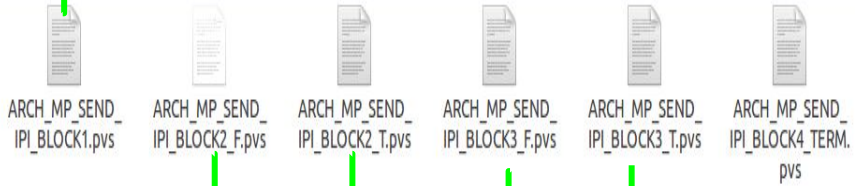
New Object of subs_addsub_imm

Original binary code-basic block stripped using radare2 analysis agf

Radare2PVS7: Basic Blocks CFG Tr



PVS working directory/zircon/terminals

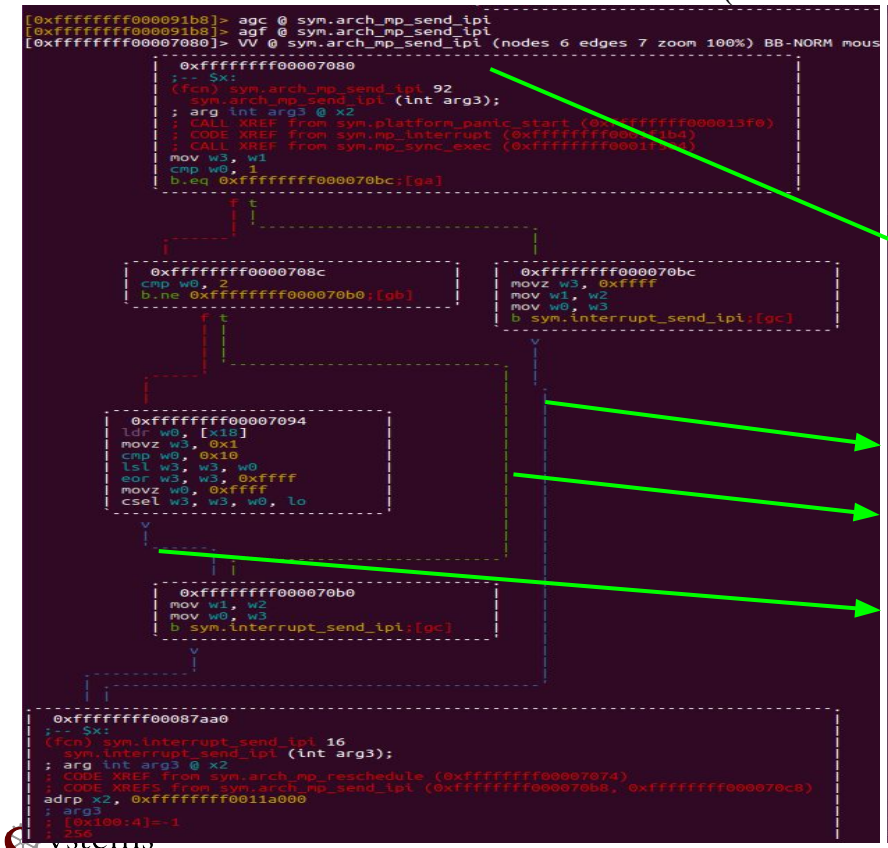


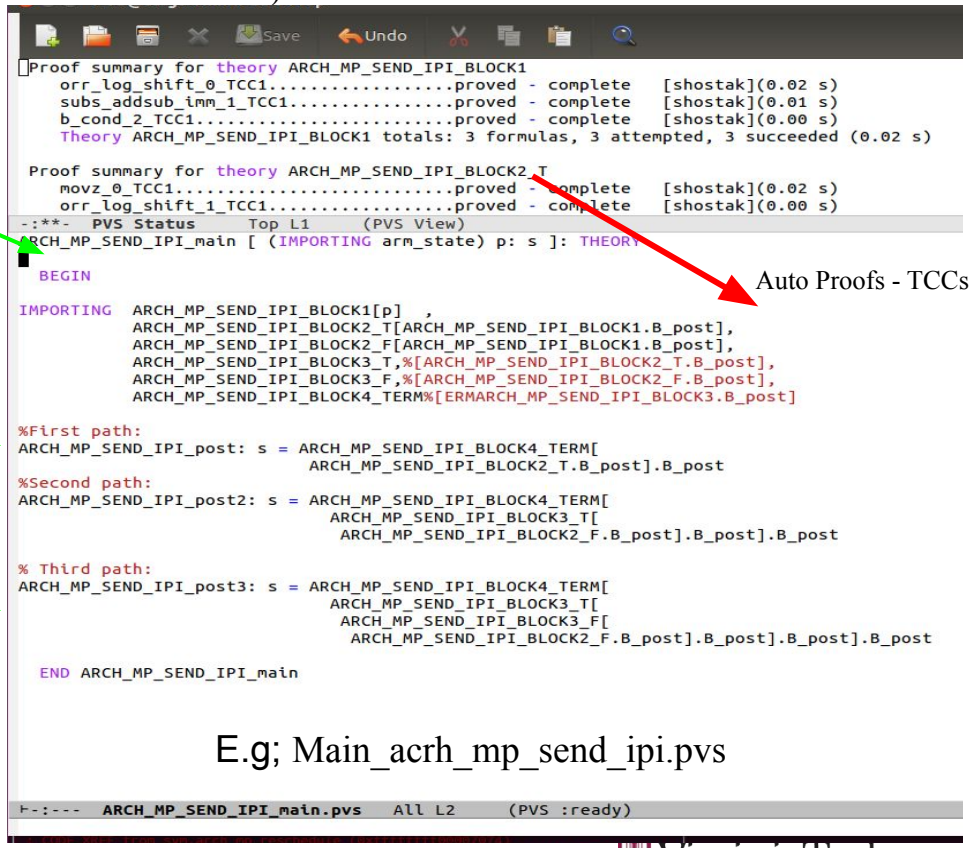
CFG: Control flow graph

VirginiaTech
Invent the Future

Functions Translation (CFG)

(Main file for each functions)





Auto Proofs - TCCs

Filling the Gap:
1- Unicorn 2 PVS7

UniVS7: Unicorn to PVS7 Validation Tool

Import Abstract model

Map pre-state unicorn state

Instantiate PVS7 model with the byte code!

Check the value emulated in PVS vs unicorn's

Validate it!

```
1  ANDS_LOG_SHIFT_8A0A1C93: THEORY BEGIN
2          IMPORTING rs1@log_shift
3  p:   s = init with [ .X:= unicorn_pre_state ]
4
5  new_test_obj: Theory = log_shift[p]{{
6              Diag:= 0b11001001001110000101000001010001,
7              addr:= 0x10000 }}
8
9  test1: lemma let X_post= p.X with [.X:=
       unicorn_post_state ] in
10                 let p2=  p with [.X:= X_post] in
11                 new_test_obj.post = p2.post
12
13 %UniVS7's auto generated Proof-Lite scripts:
14 %|- X_sts_TCC*          : PROOF
15 %|- test1_TCC1          : PROOF (eval-formula)  QED
16 %|- test1:PROOF (log-shift) QED
17 END ANDS_LOG_SHIFT_8A0A1C93
```

Listing 6: ands_log_shift UniVS7 generic test format

Filling the Gap:
2- Reverse Dictionaries

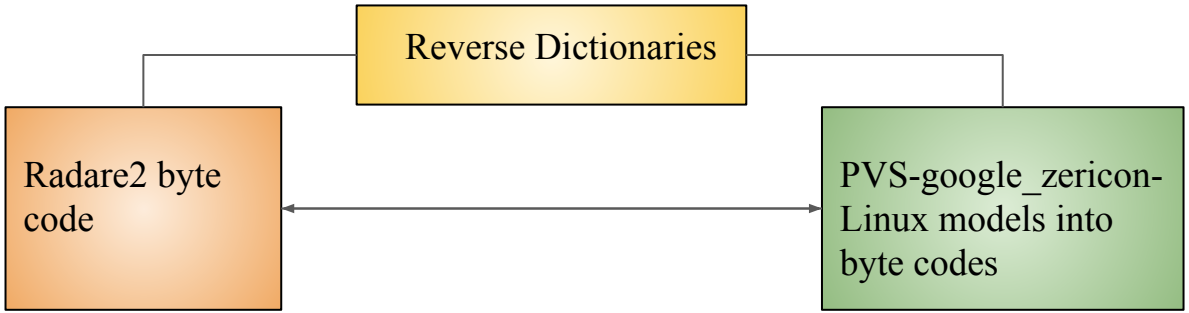
Radare2PVS Validation via Reverse Dictionaries

Decoder:
Byte code1 -- > decoded into ands_log_shift_0.pvs with Diag0

Reverse Dic:
Encode ands_log_shift_0.pvs with Diag0 into **Byte code2**

Then it Checks :

code1 = code2



We encode PVS instructions back to ARM binary using a reversed algorithm of the decoder and compare the outputs with radare's code

Renee on Google's Zircon & Linux

Simple demo

Click here: Renee_v1 tr from r2pvs7

Statistics & Results

Fig. 2: Instructions classes usage in Zircon and Linux.

Table 1: Toolchain statistics on Zircon microkernel and Linux Kernel.

Properties	Zircon	Linux
Languages Used	C++, C, asm, Py, Shell	C,asm
Size of High Source	71K	100K
# Byte Code Analyzed	200K	300K
# Functions Formalized	127	243
Target CFG&FCG	Tree,Term	Tree,Term
Target Byte Code	383	665
# PVS7&Proof-Lite LOC	2253	6950
Overall Tests & TCCs	150K	150K
Reverse Dic Tests & TCCs	3056	5320
Translator TCB LOC	1430	1430
Time for Tr	10m	15m
Time for Tr VA	30m	45m

Limitations

1. We formalized a subset of ARMv8.v3-A64 instructions (used in our targets' selected functions).
2. We are also restricted to Linear-terminal functions (essential to formalizing almost all other functions).
3. We supported sequential deterministic code.

Work in progress

- Adding more A64 instructions classes (more coverage),
- Adding more 32bits-instructions (back compatibility),
- Functions with loops,
- Proving security properties: Adding formal assurance against (DOP, JOP, ROP attacks).

Questions?

The End!

THANK YOU!

REFERENCES

[1] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," in *ACM Symposium on Operating Systems Principles*. ACM, 2009, pp. 207–220.

[2] G. Heiser and K. Elphinstone, "L4 microkernels: The lessons from 20 years of research and deployment," *ACM Transactions on Computer Systems (TOCS)*, vol. 34, no. 1, p. 1, 2016.

[3] X. Leroy, "Formal verification of a realistic compiler," *Commun. ACM*, vol. 52, no. 7, pp. 107–115, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1538788.1538814

[4] A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang, "Semantic foundations for typed assembly languages," *ACM Trans. Program. Lang. Syst.*, vol. 32, no. 3, pp. 7:1–7:67, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1709093.1709094

[5] T. Sewell, F. Kam, and G. Heiser, "Complete, high-assurance determination of loop bounds and infeasible paths for wcet analysis," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*. IEEE, 2016, pp. 1–11.

[6] N. Shankar, "Combining theorem proving and model checking through symbolic analysis," in *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, 2000, pp. 1–16. [Online]. Available: https://doi.org/10.1007/3-540-44618-4_1

[7] S. Owre, J. Rushby, N. Shankar, and D. Stringer-Calvert, "PVS: an experience report," in *Applied Formal Methods—FM-Trends 98*, ser. Lecture Notes in Computer Science, D. Hutter, W. Stephan, P. Traverso, and M. Ullman, Eds., vol. 1641. Boppard, Germany: Springer-Verlag, oct 1998, pp. 338–345. [Online]. Available: http://www.csl.sri.com/papers/fmtrends98/

[8] J. Ševčík, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan, and P. Sewell, "Compcerttso: A verified compiler for relaxed-memory concurrency," *J. ACM*, vol. 60, no. 3, pp. 22:1–22:50, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2487241.2487248

[9] L. Nelson, H. Sigurbjarnarson, K. Zhang, D. Johnson, J. Bornholt, E. Torlak, and X. Wang, "Hyperkernel: Push-button verification of an os kernel," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 252–269. [Online]. Available: http://doi.acm.org/10.1145/3132747.3132748

[10] T. A. L. Sewell, M. O. Myreen, and G. Klein, "Translation validation for a verified OS kernel," in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, 2013, pp. 471–482. [Online]. Available: https://doi.org/10.1145/2491956.2462183

[11] S. Flur, K. E. Gray, C. Pulte, S. Sarkar, A. Sezgin, L. Maranget, W. Deacon, and P. Sewell, "Modelling the armv8 architecture, operationally: Concurrency and isa," *SIGPLAN Not.*, vol. 51, no. 1, pp. 608–621, Jan. 2016. [Online]. Available: http://doi.acm.org/10.1145/2914770.2837615

[12] A. C. J. Fox and M. O. Myreen, "A trustworthy monadic formalization of the armv7 instruction set architecture," in *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, 2010, pp. 243–258. [Online]. Available: https://doi.org/10.1007/978-3-642-14052-5_18

[13] A. C. J. Fox, "Improved tool support for machine-code decompilation in HOL4," in *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, 2015, pp. 187–202. [Online]. Available: https://doi.org/10.1007/978-3-319-22102-1_12

[14] M. O. Myreen, A. C. J. Fox, and M. J. C. Gordon, "Hoare logic for ARM machine code," in *International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings*, 2007, pp. 272–286. [Online]. Available: https://doi.org/10.1007/978-3-540-75698-9_18

[15] D. P. Mulligan, S. Owens, K. E. Gray, T. Ridge, and P. Sewell, "Lem: Reusable engineering of real-world semantics," *SIGPLAN Not.*, vol. 49, no. 9, pp. 175–188, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2692915.2628143