Collaborators/funders:
Systems and Software Security / FM Research Group
ARM Centre of Excellence
PPGEE, PPGI – UFAM
Centre for Digital Trust and Society
UKRI, EPSRC, EU Horizon and industrial partners

MANCHESTER 1824
The University of Manchester

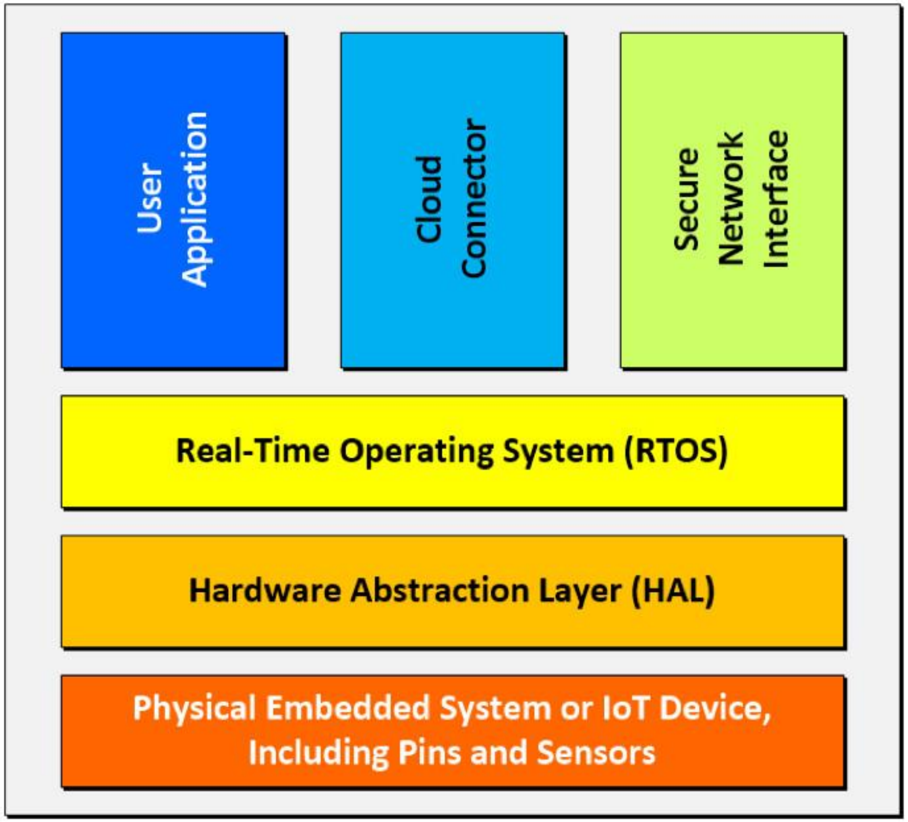
Security of Software Systems with Applications on the Internet of Things

Lucas Cordeiro
Department of Computer Science
lucas.cordeiro@manchester.ac.uk
https://ssvlab.github.io/lucasccordeiro/

Security in IoT Software

- Software security consists of **building programs** that continue to function **correctly** under **malicious attack**

Requirements	Definition
Availability	services are accessible if requested by authorized users
Integrity	data completeness and accuracy are preserved
Confidentiality	only authorized users can get access to the data



Basic software components in a secure embedded system or IoT device (Image source: Arm)

Memory Safety Vulnerabilities

Memory errors in **low-level software** written in **unsafe programming languages** represent one of the main problems in **computer security**

- The top 13 vulnerabilities in CWE include **five types of memory errors** (out of bounds and use after free)
- **Two out of the top three vulnerabilities** found in **GitHub** projects were memory safety issues
- **Microsoft** reports that around **70%** of all security updates in their products address **memory issues**
- **Google** reports a **similar number** for Chrome Browser

The CWE Top 13

#	ID	Name
1	**CWE-787**	**Out-of-bounds Write**
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
4	CWE-20	Improper Input Validation
5	**CWE-125**	**Out-of-bounds Read**
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
7	**CWE-416**	**Use After Free**
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
9	CWE-352	Cross-Site Request Forgery (CSRF)
10	CWE-434	Unrestricted Upload of File with Dangerous Type
11	**CWE-476**	**NULL Pointer Dereference**
12	CWE-502	Deserialization of Untrusted Data
13	**CWE-190**	**Integer Overflow or Wraparound**

Objective of this talk

Discuss **automated testing and formal verification** techniques that establish the **security of software systems**

- Define **standard notions of security** and **(software) security vulnerabilities** in **embedded and IoT applications**
- Explain **testing** and **verification** techniques to reason about the **system and software security**
- Present recent advancements towards a **hybrid approach** to protecting against **memory safety vulnerabilities**

Agenda

- **Define standard notions of security and (software) security vulnerabilities in real-world applications**
- Explain testing and verification techniques to reason about the system and software security
- Present recent advancements towards a hybrid approach to protect against memory safety vulnerabilities

What does it mean for software to be secure?

- A software system is secure if it **satisfies** a specified **security objective**

Example of Unmanned Aerial Vehicles (UAVs)

Vulnerability analysis

Remote accessibility (device authentication, access control)

Patch management

Attacks from physical world (GPS spoofing)

Boeing Unmanned Little Bird H-6U

Attacked by **rogue camera software** and by a **virus** delivered through a compromised USB stick

Klein et al., Formally verified software in the real world. Commun. ACM 61(10): 68-77 (2018)

Implementation Vulnerability

- We use the term *implementation vulnerability* (or *security bug*) both for bugs that
  - make it possible for an attacker to violate a **security objective**
  - for classes of bugs that enable **specific attack** techniques

Example of IoT: Message Queuing Telemetry Transport

- In 2021, we detected a data race vulnerability in the wolfMQTT library (messaging protocol)
  - Detected in function *MqttClient_WaitType*, which could lead to an information leak or data corruption

https://github.com/wolfSSL/wolfMQTT/issues/198
https://github.com/wolfSSL/wolfMQTT/pull/209

Aljaafari et al.: Combining BMC and Fuzzing Techniques for Finding Software Vulnerabilities in Concurrent Programs.

Critical Software Vulnerabilities

- Null pointer dereference

```
int main() {
 double *p = NULL;
  int n = 8;
 for(int i = 0; i < n; ++i )
   *(p+i) = i*2;
 return 0;
}
```

A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL

Scope	Impact
Availability	Crash, exit and restart
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands

Critical Software Vulnerabilities

- Null pointer dereference
- Double free

```
int main(){
 char* ptr = (char *)malloc(sizeof(char));
 if(ptr==NULL) return -1;
 *ptr = 'a';
 free(ptr);
  free(ptr);
 return 0;
}
```

The product calls *free()* twice on the same memory address, leading to modification of unexpected memory locations

Scope	Impact
Integrity Confidentiality Availability	Execute Unauthorized Code or Commands

Critical Software Vulnerabilities

- Null pointer dereference
- Double free
- Unchecked Return Value to NULL Pointer Dereference
- Division by zero
- Missing free
- Use after free
- APIs rule based checking

Research Questions

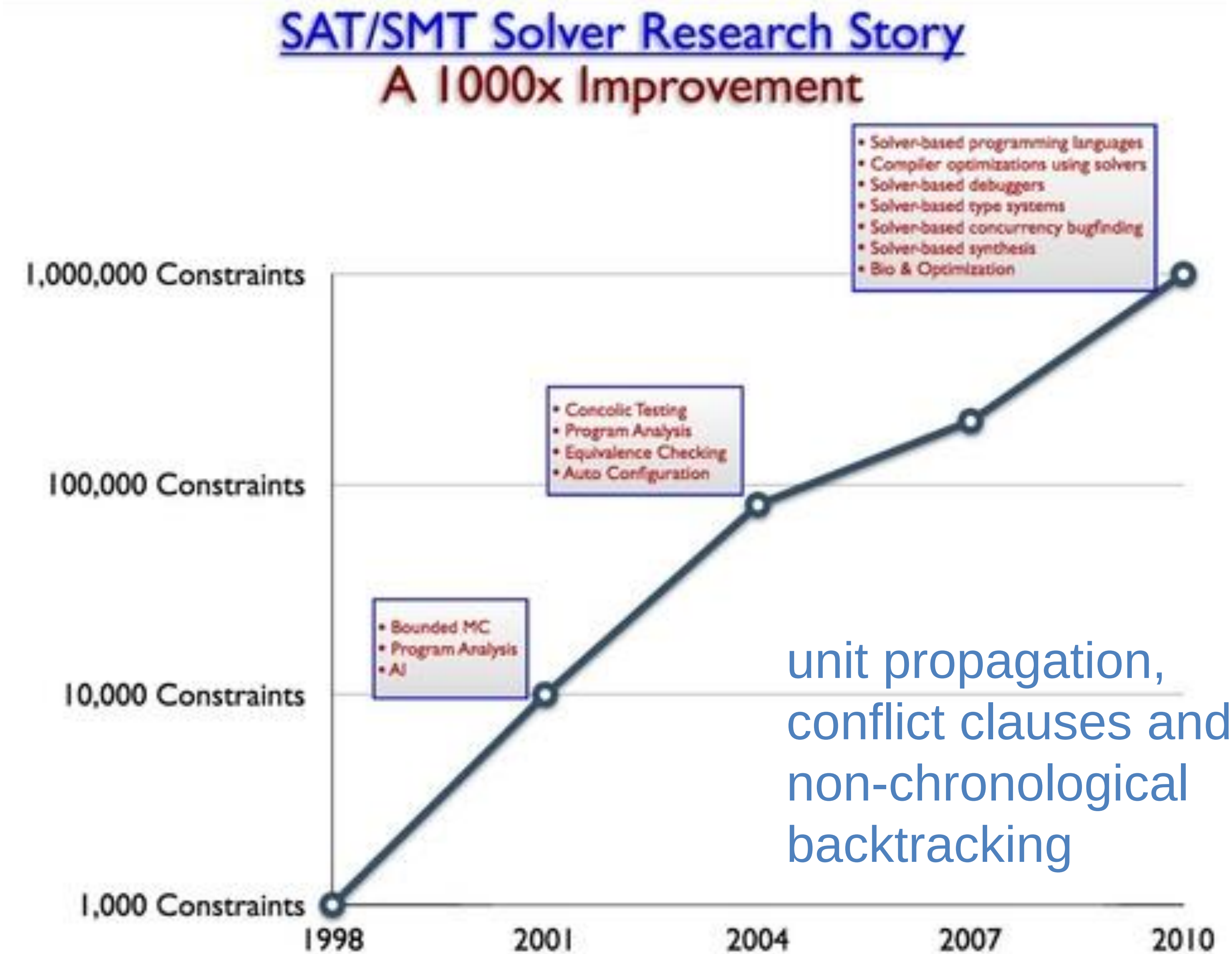
Given a **program** and a **security specification**, can we automatically **verify** that the **program performs as specified**?

Can we leverage **program analysis/synthesis** to discover more **software vulnerabilities** than existing state-of-the-art approaches?

Agenda

- Define standard notions of security and (software) security vulnerabilities in real-world applications
- **Explain testing and verification techniques to reason about the system and software security**
- Present recent advancements towards a hybrid approach to protect against memory safety vulnerabilities

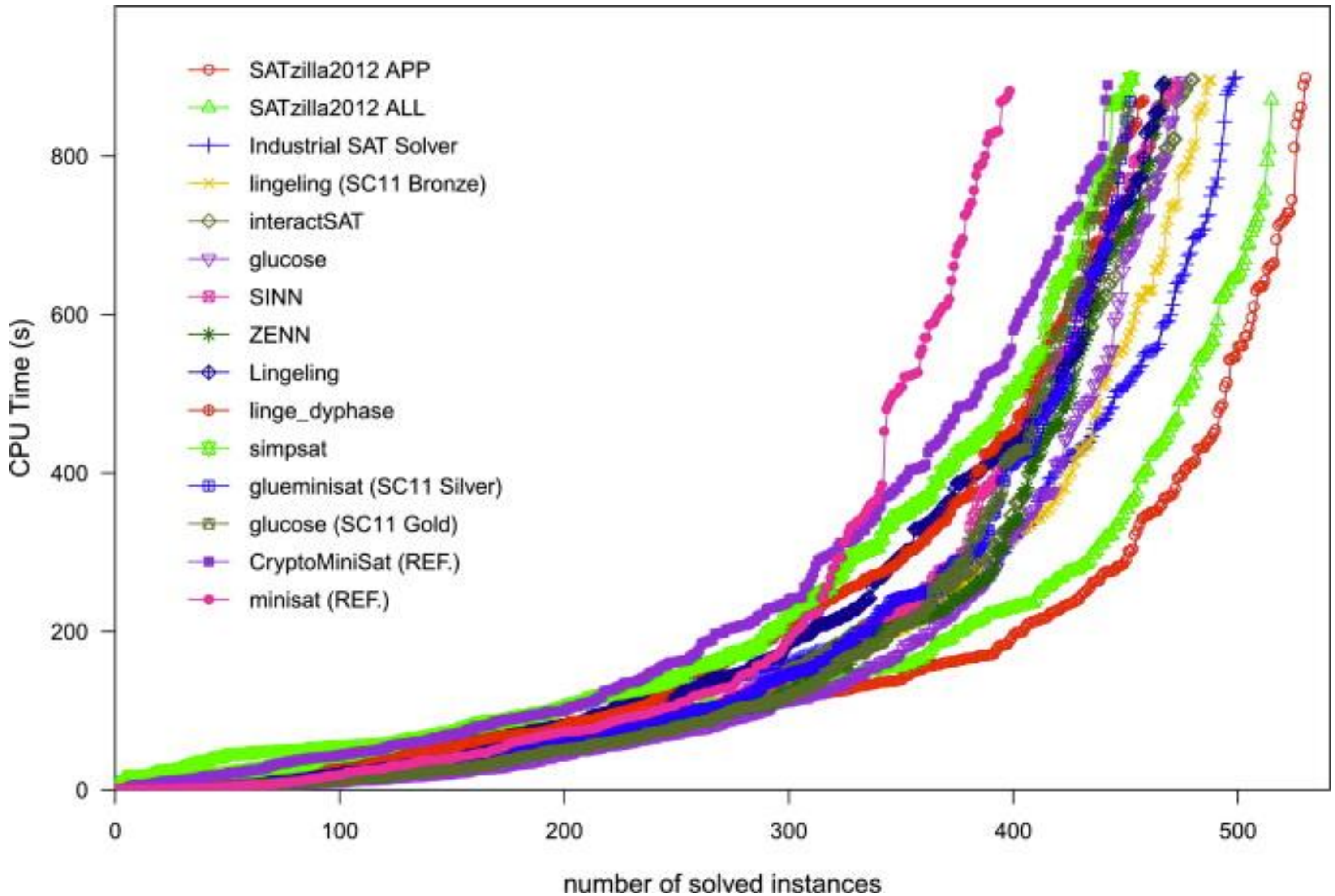
SAT solving as enabling technology



unit propagation, conflict clauses and non-chronological backtracking

Kroening, D., Strichman, O., Decision Procedures - An Algorithmic Point of View, Second Edition, Springer.

SAT Competition

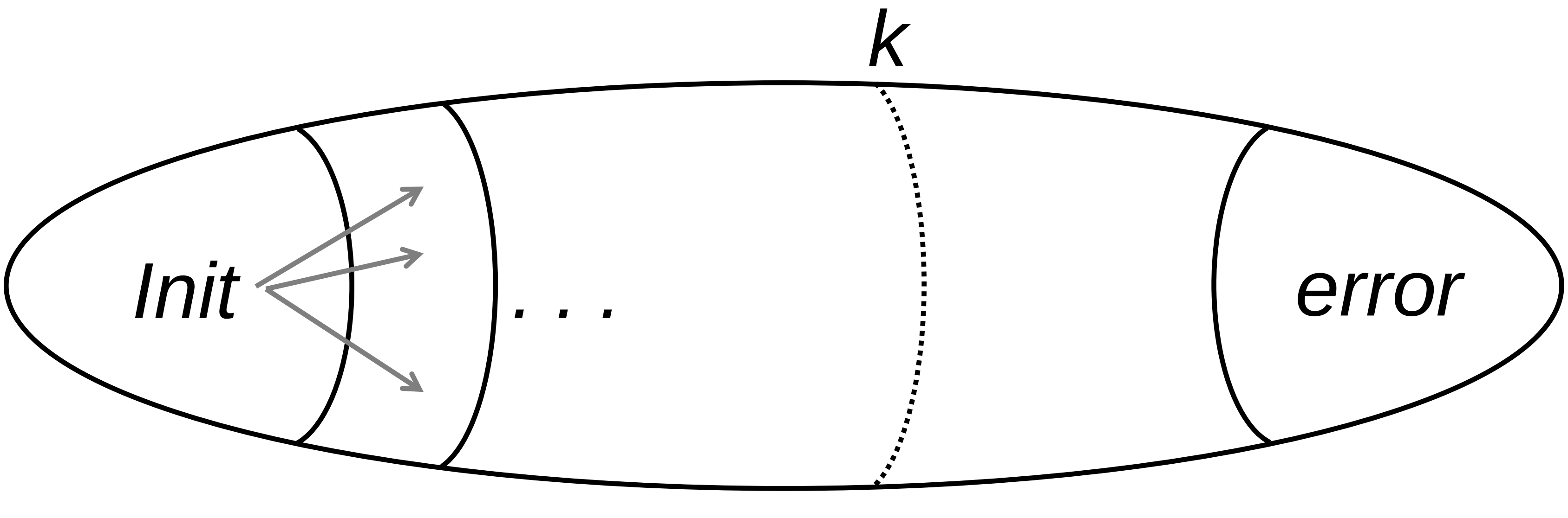


http://www.satcompetition.org/

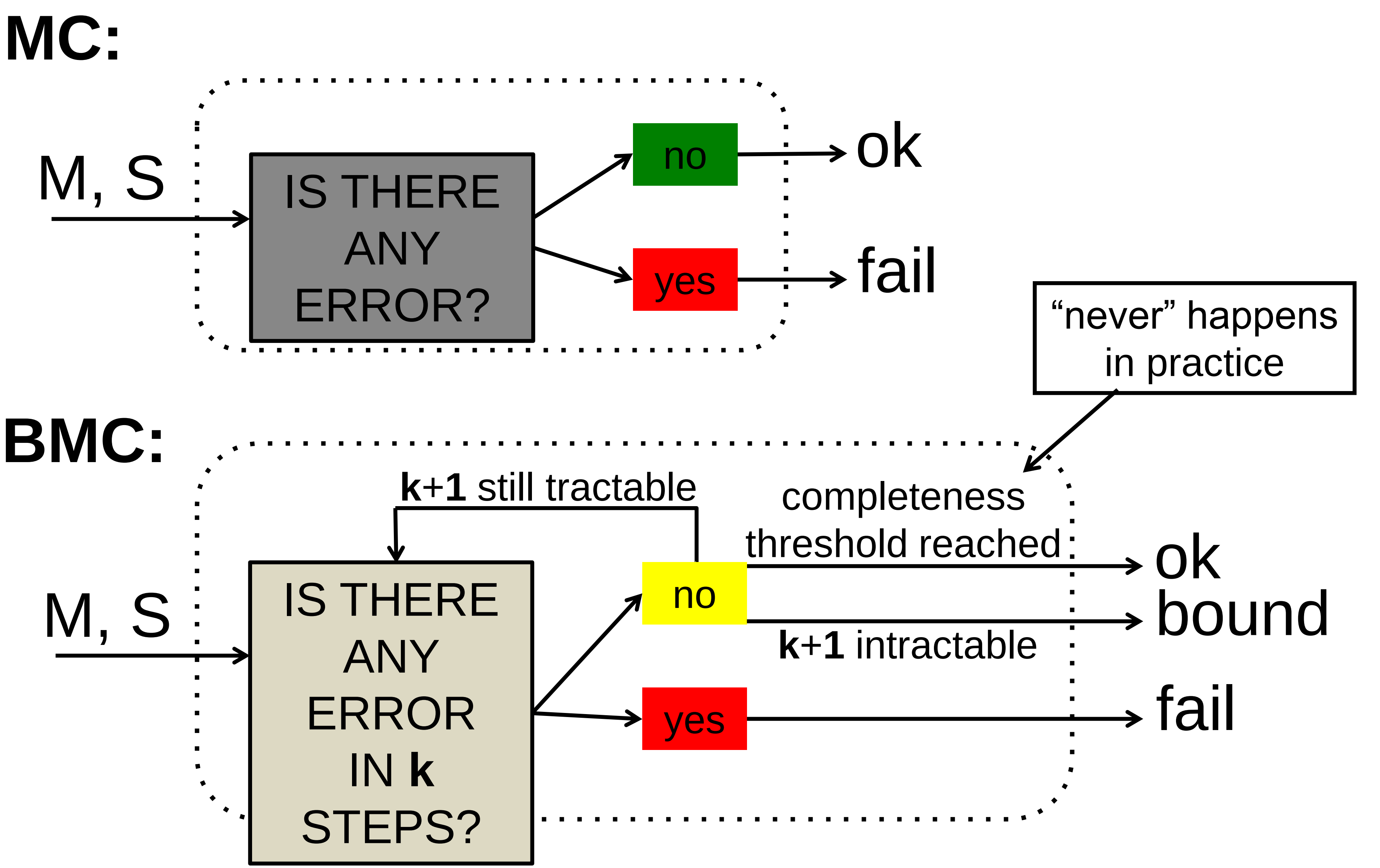
Bounded Model Checking (BMC)

MC: check if a property holds for all states

BMC: check if a property holds for a subset of states



Bounded Model Checking (BMC)

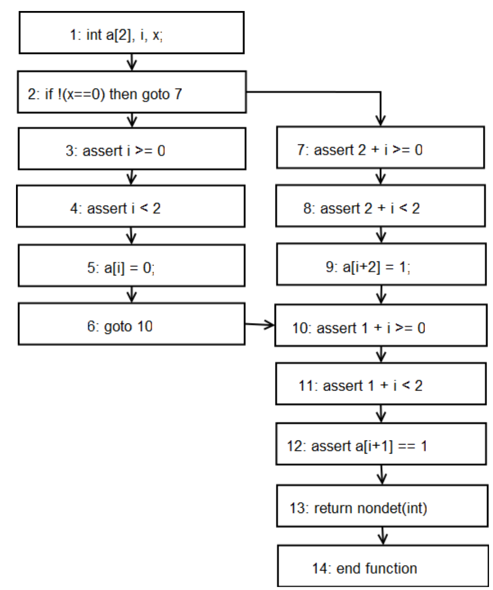


Software BMC

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, ”ML”);
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf(“Access Denied\n”);
    exit(0);
  }
  printf(“Access Granted\n”);
}
```

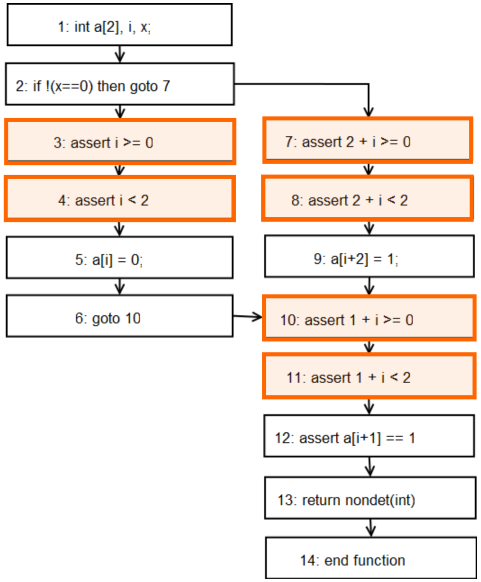


Software BMC

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

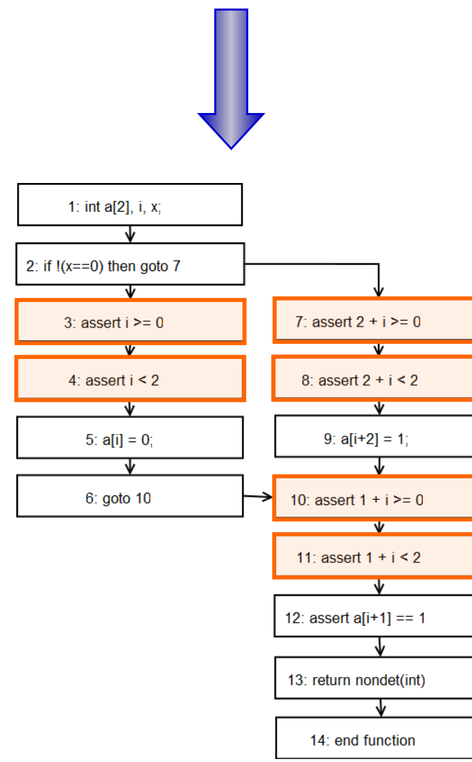


Software BMC

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, ”ML”);
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf(“Access Denied\n”);
    exit(0);
  }
  printf(“Access Granted\n”);
}
```



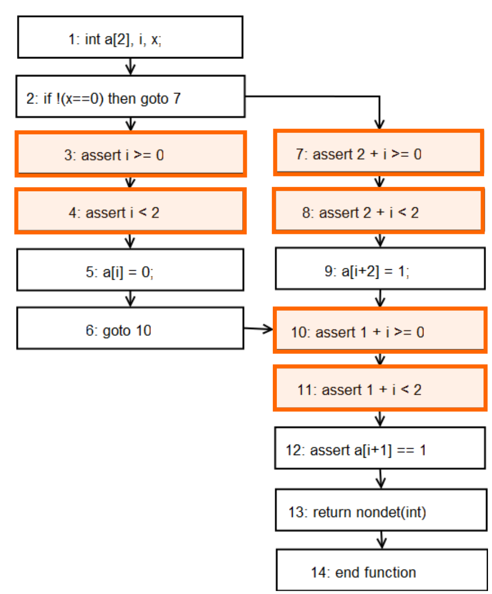
Software BMC

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions
  - unreachable code

} crucial



Software BMC

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions
  - unreachable code

  } crucial
- front-end converts unrolled and **optimized program into SSA**

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

$$g_1 = x_1 == 0$$
$$a_1 = a_0 \text{ WITH } [i_0 := 0]$$
$$a_2 = a_0$$
$$a_3 = a_2 \text{ WITH } [2+i_0 := 1]$$
$$a_4 = g_1 \ ?\ a_1 : a_3$$
$$t_1 = a_4[1+i_0] == 1$$

Software BMC

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions
  - unreachable code

  } crucial
- front-end converts unrolled and **optimized program into SSA**
- extraction of *constraints C* and *properties P*

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

$$C := \begin{bmatrix} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2 + i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{bmatrix}$$

$$P := \begin{bmatrix} i_0 \ge 0 \wedge i_0 < 2 \\ \wedge\, 2 + i_0 \ge 0 \wedge 2 + i_0 < 2 \\ \wedge\, 1 + i_0 \ge 0 \wedge 1 + i_0 < 2 \\ \wedge\, select(a_4, i_0 + 1) = 1 \end{bmatrix}$$

Software BMC

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions
  - unreachable code

  } crucial
- front-end converts unrolled and **optimized program into SSA**
- extraction of *constraints C* and *properties P*
  - specific to selected SMT solver, uses theories

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

$$C := \left[\begin{array}{l} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2 + i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{array}\right]$$

$$P := \left[\begin{array}{l} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2 + i_0 \geq 0 \wedge 2 + i_0 < 2 \\ \wedge\, 1 + i_0 \geq 0 \wedge 1 + i_0 < 2 \\ \wedge\, select(a_4, i_0 + 1) = 1 \end{array}\right]$$

Software BMC

```
int getPassword() {
  char buf[2];
  gets(buf);
  return strcmp(buf, "ML");
}
```

```
void main(){
  int x=getPassword();
  if(x){
    printf("Access Denied\n");
    exit(0);
  }
  printf("Access Granted\n");
}
```

- program modelled as transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions
  - unreachable code

  } crucial
- front-end converts unrolled and **optimized program into SSA**
- extraction of *constraints C* and *properties P*
  - specific to selected SMT solver, uses theories
- satisfiability check of $C \wedge \neg P$

$$C := \begin{bmatrix} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2 + i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{bmatrix}$$

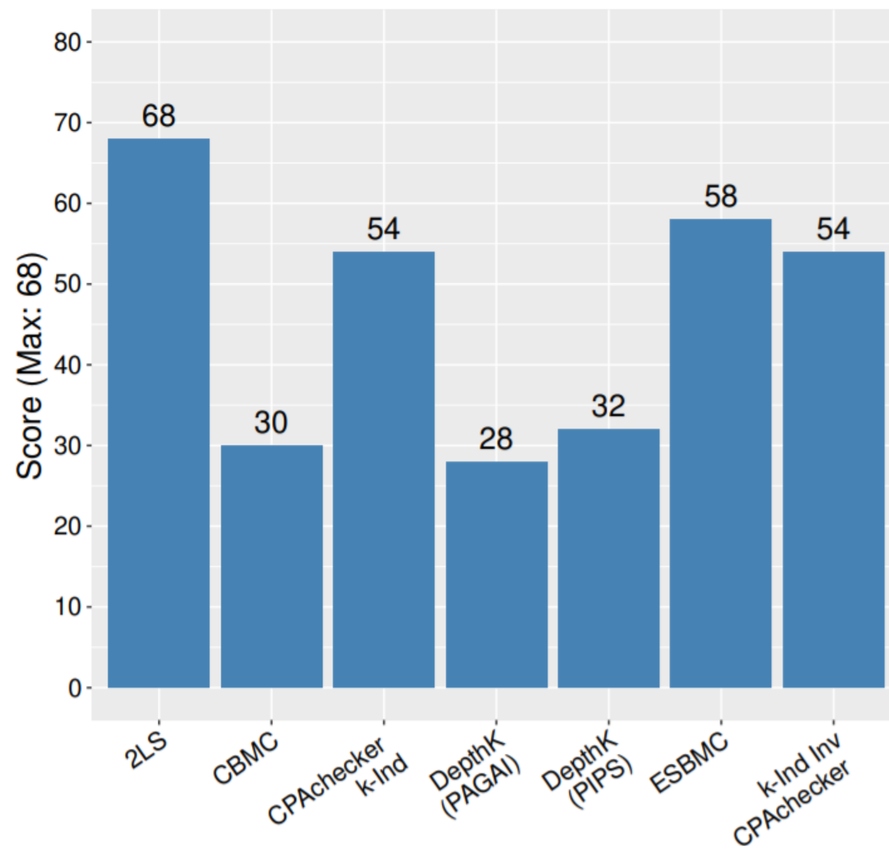
$$P := \begin{bmatrix} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2 + i_0 \geq 0 \wedge 2 + i_0 < 2 \\ \wedge\, 1 + i_0 \geq 0 \wedge 1 + i_0 < 2 \\ \wedge\, select(a_4, i_0 + 1) = 1 \end{bmatrix}$$

Cordeiro et al.: SMT-Based Bounded Model Checking for Embedded ANSI-C Software. IEEE TSE, 2012

Embedded Software Verification

- **Powerstone:** automotive-control and fax applications
- **Real-Time SNU:** matrix handling and signal processing, cyclic-redundancy check, Fourier transform, and JPEG encoding
- **WCET:** a set of programs for executing worst-case time analysis

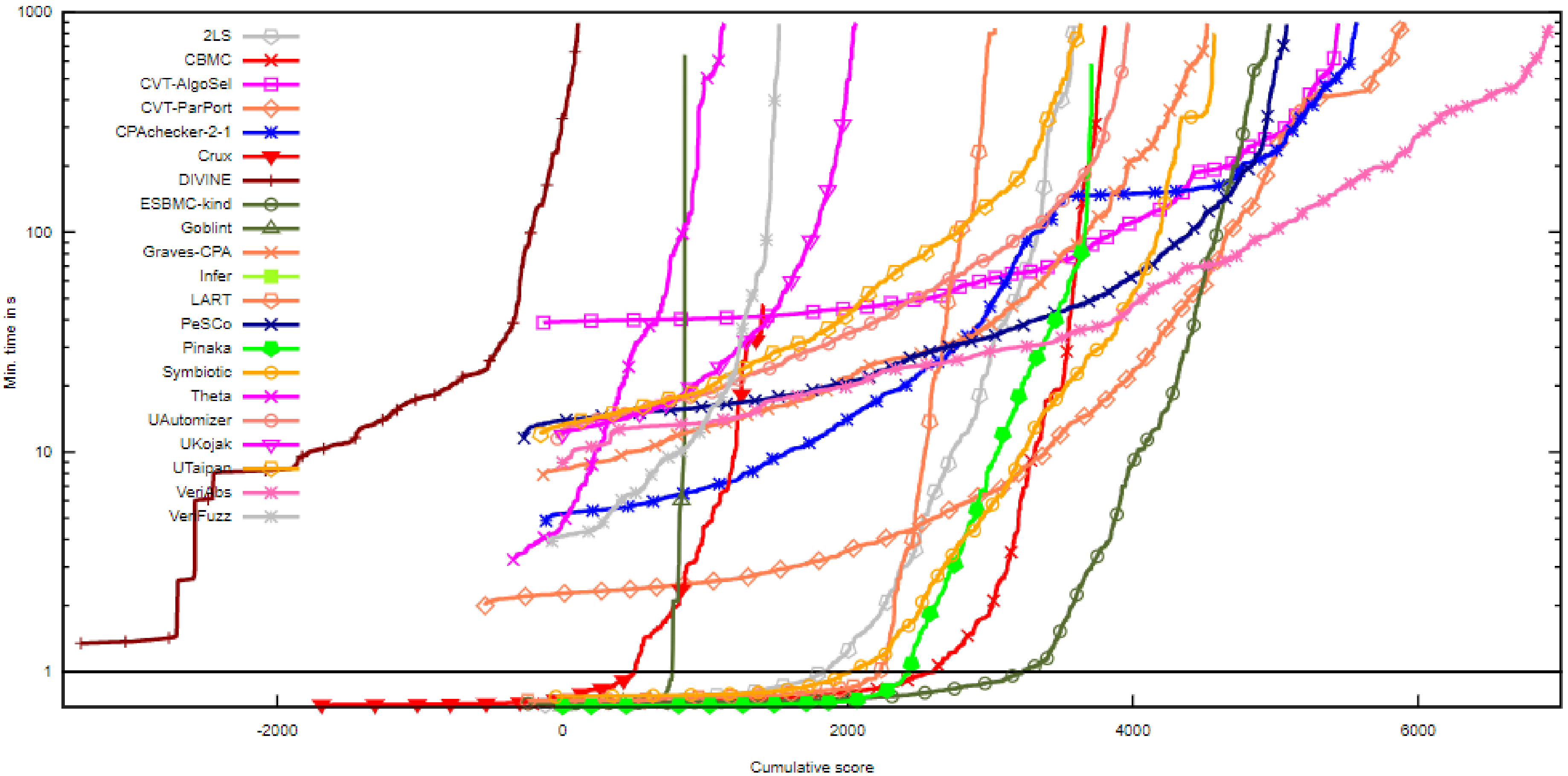
34 tasks; 900s, 15GB
ESBMC achieved the 2nd place



Alhawi et al.: Verification and refutation of C programs based on k-induction and invariant inference. STTT, 2021

Verification of the `Reach-Safety` Category

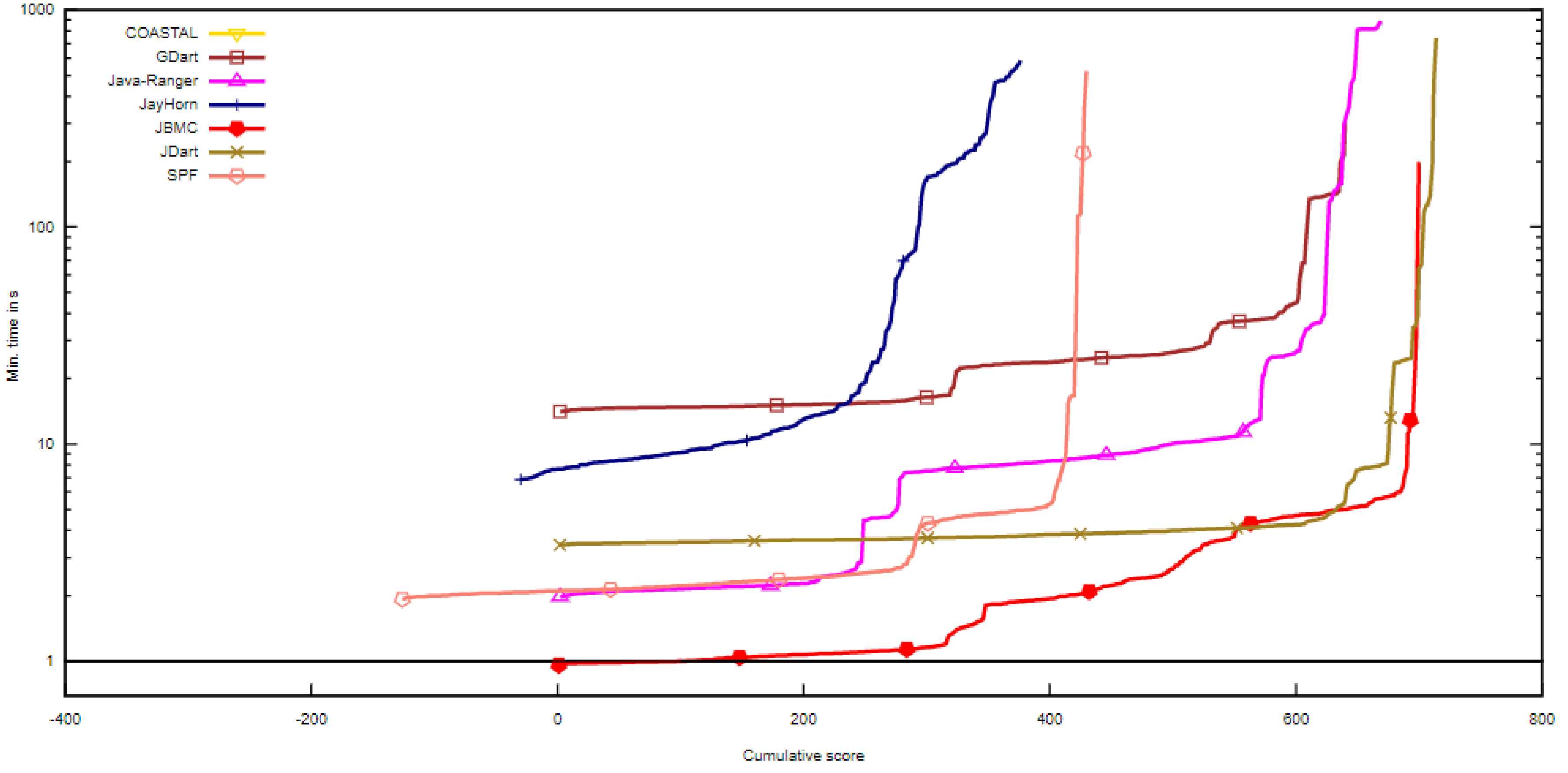
- SV-COMP 2022, 5400 verification tasks, max. score: 8631
- ESBMC achieved the 6th place



https://sv-comp.sosy-lab.org/2022/

Verification of the `Java` Category

- SV-COMP 2022, 586 verification tasks, max. score: 828
- JBMC achieved the 2th place

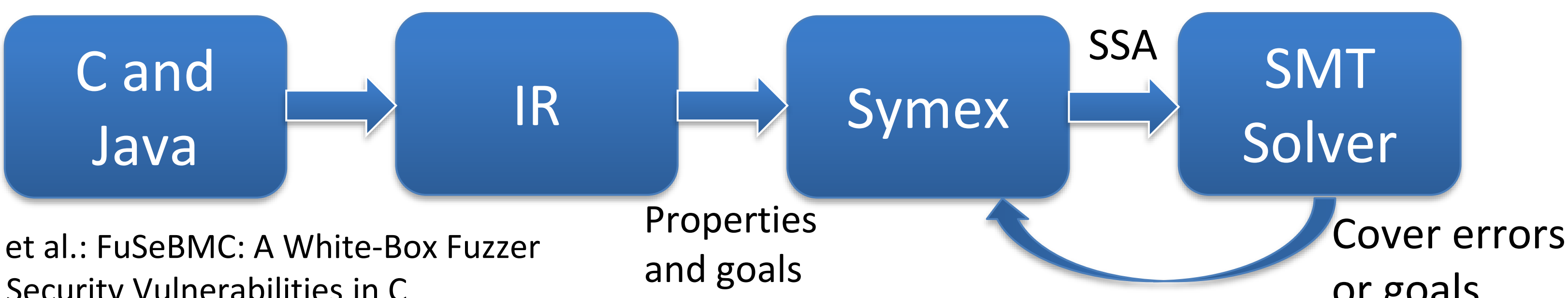


Cordeiro et al.: JBMC: A Bounded Model Checking Tool
for Verifying Java Bytecode. CAV (1) 2018: 183-190

https://sv-comp.sosy-lab.org/2022/

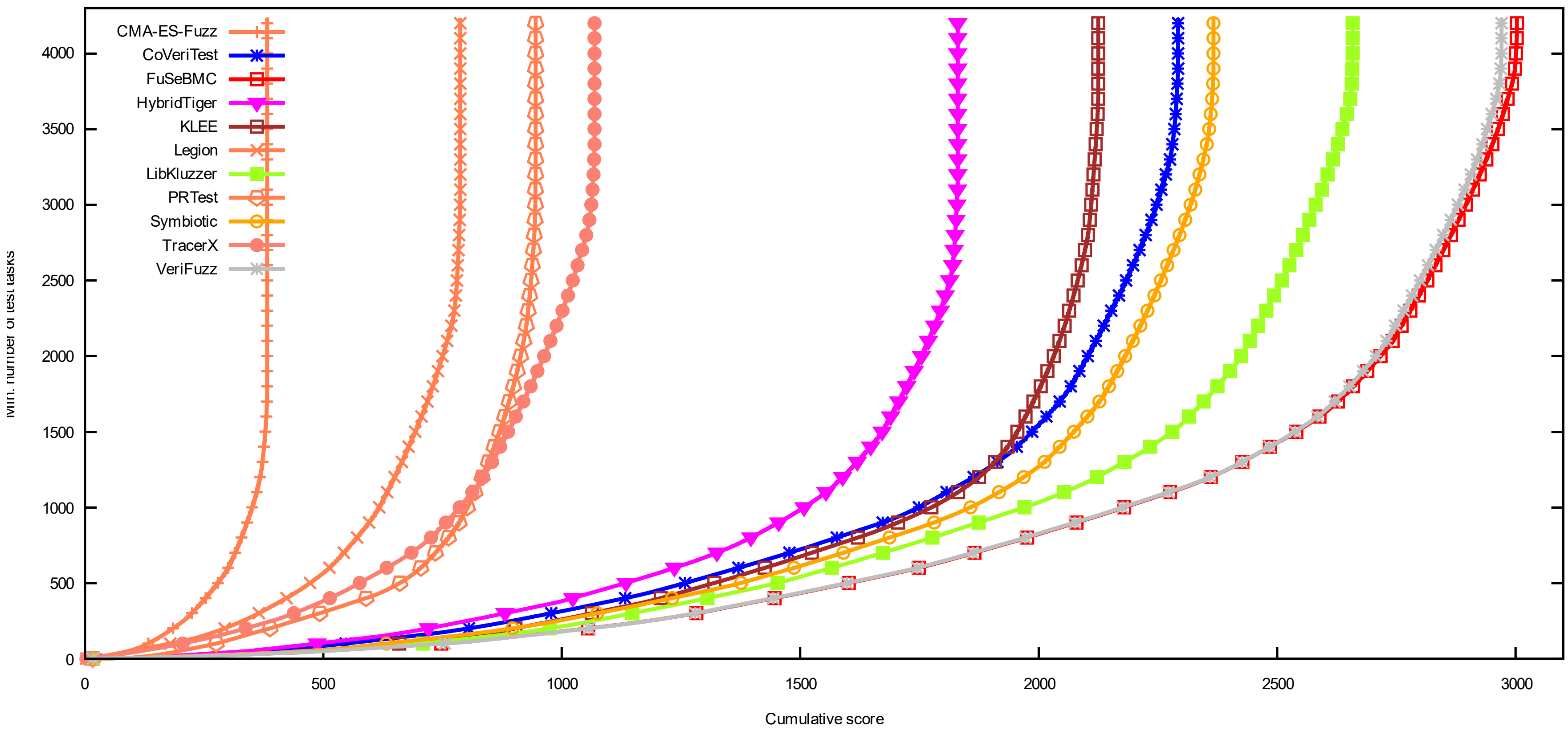
White-box Fuzzing:
Bug Finding and Code Coverage

- Translate the program to an **intermediate representation** (IR)
- Add properties to check **errors** or goals to check **coverage**
- **Symbolically** execute IR to produce an SSA program
- Translate the resulting SSA program into a **logical formula**
- Solve the formula iteratively to cover errors and goals
- Interpret the solution to figure out the **input conditions**
- Spit those input conditions out as a test case



Alshmrany et al.: FuSeBMC: A White-Box Fuzzer for Finding Security Vulnerabilities in C Programs. FASE, 2021

Competition on Software Testing 2022: Results of the `Cover-Error` Category



FuSeBMC achieved 3 awards: 1st place in `Cover-Error`, 1st place in `Cover-Branches`, and 1st place in `Overall`

https://test-comp.sosy-lab.org/2022/

WolfMQTT Verification

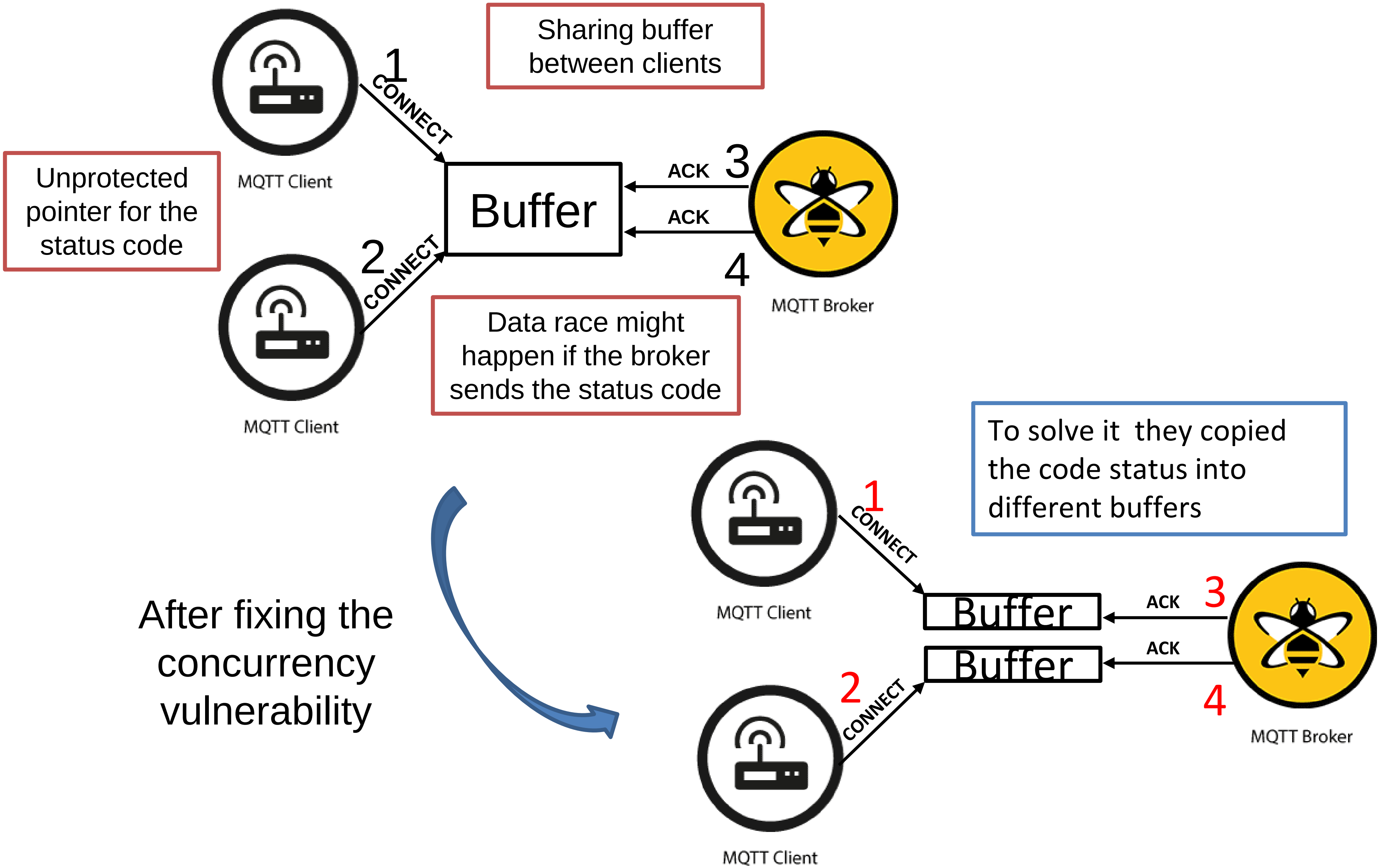
- **wolfMQTT** library is a client implementation of the MQTT protocol written in C for **IoT devices**

`subscribe_task` and `waitMessage_task` are called through different threads accessing `packet_ret`, causing a data race in `MqttClient_WaitType`

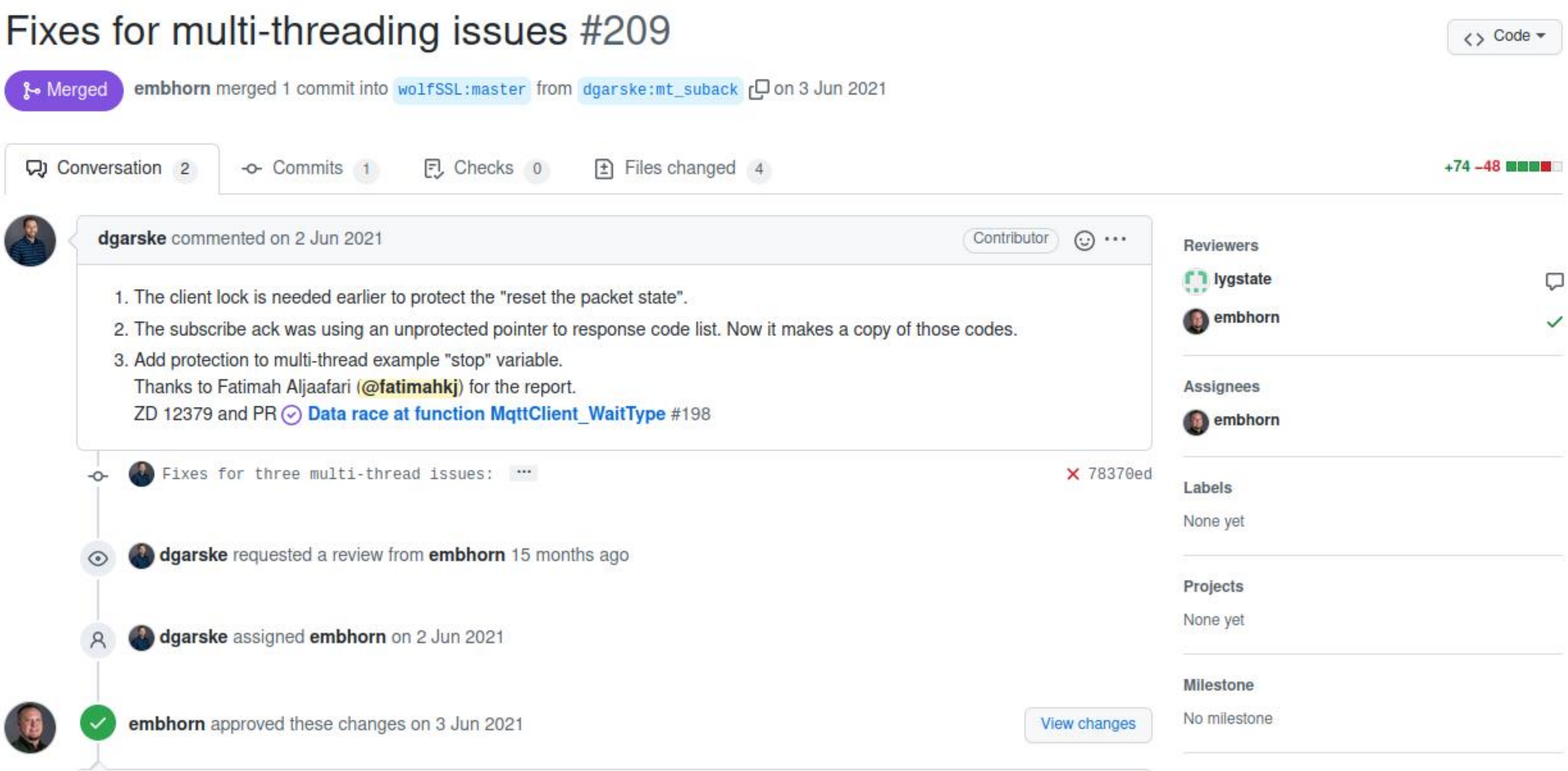
```
Int main(){
Pthread_t th1, th2;
static MQTTCtx mqttCtx;
pthread_create(&th1, subscribe_task, &mqttCtx))
pthread_create(&th2, waitMessage_task, &mqttCtx))}

static void *subscribe_task(void *client){
.....
MqttClient_WaitType(client,msg,MQTT_PACKET_TYPE_ANY,
0,timeout_ms);
.....}
static void *waitMessage_task(void *client){
…
MqttClient_WaitType(client, msg, MQTT_PACKET_TYPE_ANY,
0,timeout_ms);
.....}
static int MqttClient_WaitType(MqttClient *client,
void *packet_obj,
   byte wait_type, word16 wait_packet_id, int timeout_ms)
{
.....
           rc = wm_SemLock(&client->lockClient);
           if (rc == 0) {
               if (MqttClient_RespList_Find(client,
(MqttPacketType)wait_type,
                       wait_packet_id, &pendResp)) {
                   if (pendResp->packetDone) {
                       rc = pendResp->packet_ret;
.....}
```

Here is where the data race might happen! Unprotected pointer

WolfMQTT Verification



Bug Report



Fixes for multi-threading issues #209

Merged **embhorn** merged 1 commit into `wolfSSL:master` from `dgarske:mt_suback` on 3 Jun 2021

Code

Conversation 2 | Commits 1 | Checks 0 | Files changed 4 | +74 −48

dgarske commented on 2 Jun 2021 — Contributor

1. The client lock is needed earlier to protect the "reset the packet state".
2. The subscribe ack was using an unprotected pointer to response code list. Now it makes a copy of those codes.
3. Add protection to multi-thread example "stop" variable.
   Thanks to Fatimah Aljaafari (@fatimahkj) for the report.
   ZD 12379 and PR **Data race at function MqttClient_WaitType** #198

`Fixes for three multi-thread issues: …` ✕ 78370ed

dgarske requested a review from **embhorn** 15 months ago

dgarske assigned **embhorn** on 2 Jun 2021

embhorn approved these changes on 3 Jun 2021 — View changes

Reviewers
lygstate
embhorn ✓

Assignees
embhorn

Labels
None yet

Projects
None yet

Milestone
No milestone

https://github.com/wolfSSL/wolfMQTT

Agenda

- Define standard notions of security and (software) security vulnerabilities in real-world applications
- **Explain testing and verification techniques to reason about the system and software security**
- **Present recent advancements towards a hybrid approach to protect against memory safety vulnerabilities**

Capability Hardware Enhanced RISC Instructions (CHERI)

63		0
permissions (15 bits)	reserved	base and bounds (41 bits)
pointer address (64 bits)		

CHERI 128-bit capability

Mnemonic	Description
CGetBase	Move base to a GPR
CGetLen	Move length to a GPR
CGetTag	Move tag bit to a GPR
CGetPerm	Move permissions to a GPR
CGetPCC	Move the PCC and PC to GPRs
CIncBase	Increase base and decrease length
CSetLen	Set (reduce) length
CClearTag	Invalidate a capability register
CAndPerm	Restrict permissions
CToPtr	Generate **C0**-based integer pointer from a capability
CFromPtr	CIncBase with support for NULL casts
CBTU	Branch if capability tag is unset
CBTS	Branch if capability tag is set
CLC	Load capability register

CHERI Clang/LLVM and **LLD**[1] - compiler and linker for CHERI ISAs

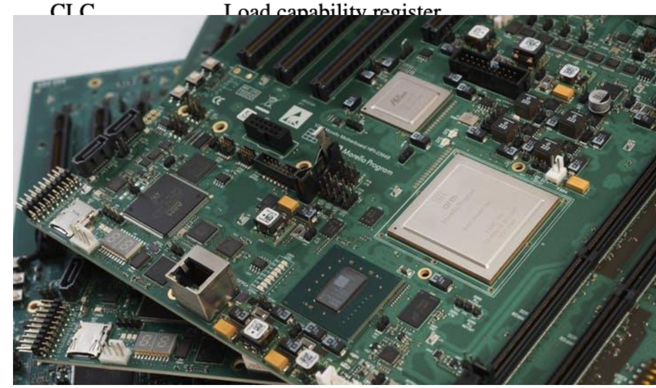
[1]https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-llvm.html

CheriBSD[2] - adaptation of FreeBSD to support CHERI ISAs

[2]https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheribsd.html

ARM Morello[3] - SoC development board with a CHERI-extended ARMv8-A processor

[3]https://www.arm.com/architecture/cpu/morello



CHERI-C program

```
#include <stdlib.h>
#include <string.h>
#include <cheri/cheric.h>

void main() {
   int n = nondet_uint() % 1024;                            /* models arbitrary user input */
   char a[n+1], *__capability b = cheri_ptr(a, n+1);
   b[n] = 17;                                               /* succeeds */
   char *__capability c = cheri_setbounds(b-1, n);          /* fails: not the same object */
   /* ... */                                                /* more CHERI-C API checks */
   memset_c(c, 42, n);                                      /* setting memory through a capability */
}
```

CHERI-C API

New capability types

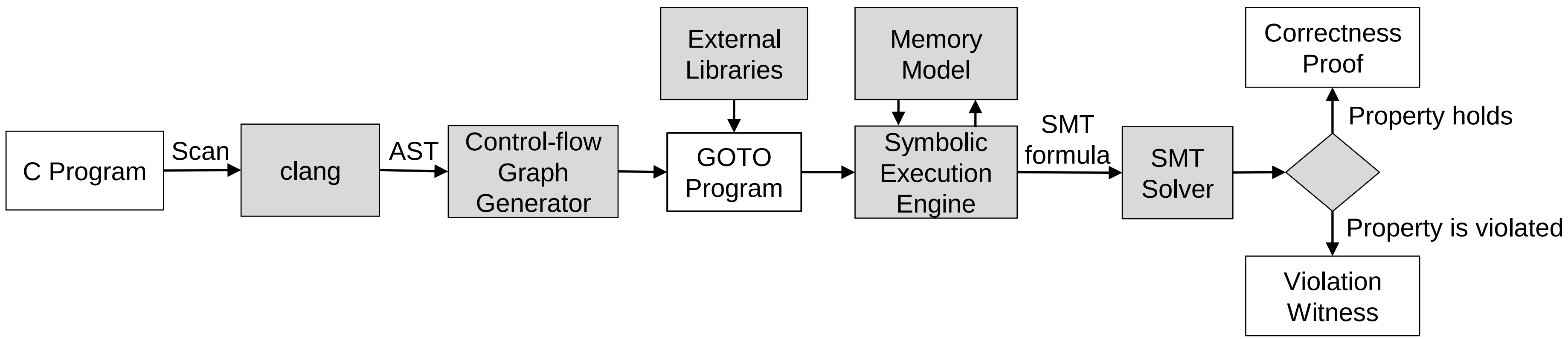
Pure-capability CHERI-C model

```
#include <stdlib.h>
#include <string.h>
#include <cheri/cheric.h>

void main() {
   int n = nondet_uint() % 1024;
   char a[n+1], *__capability b = cheri_ptr(a, n+1);
   b[n] = 17;
   char *__capability c = cheri_setbounds(b-1, n);
   /* ... */
   memset_c(c, 42, n);
}
```

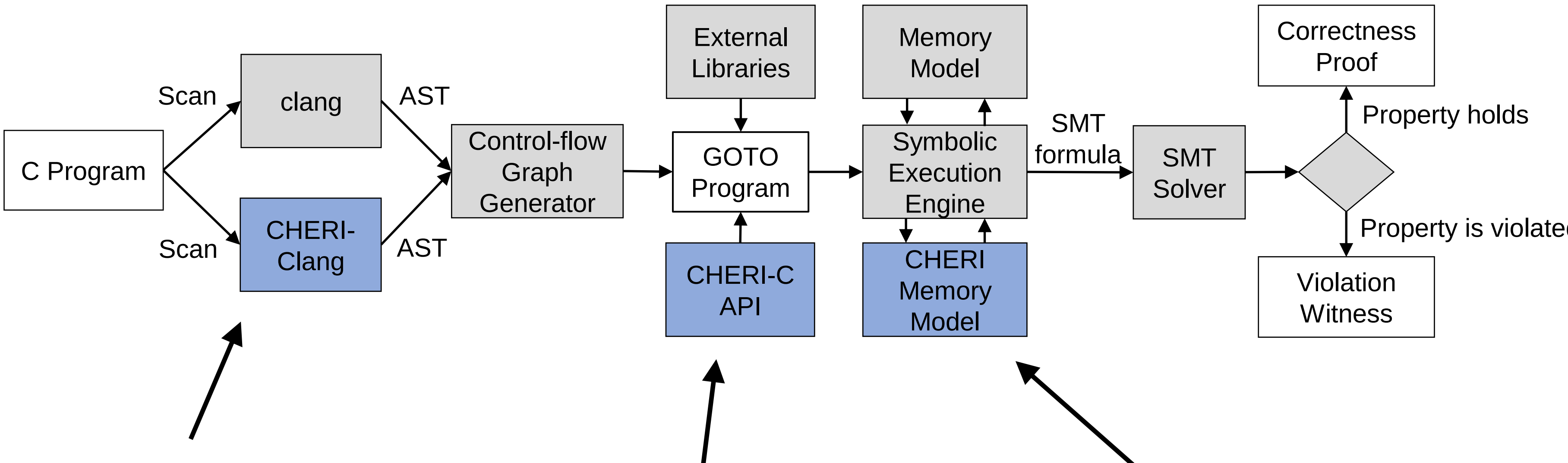
```
#include <string.h>
#include <stdio.h>

void main(void) {
   int n = nondet_uint() % 1024;
   char a[n+1], *b = a;
   b[n] = 17;
   char *c = b-1;
   memset(c, 42, n);
}
```

All pointers are automatically replaced with capabilities by the **CHERI Clang/LLVM** compiler

The Efficient SMT-based Bounded Model Checker (ESBMC)



ESBMC-CHERI



CHERI Clang/LLVM compiler

Implement computational model for CHERI-C API functions inside ESBMC (e.g., ***cheri_setbounds***)

- New capability types
- Tagged memory
- Capability dereferencing

Brauße et al.: ESBMC-CHERI: towards verification of C programs for CHERI platforms with ESBMC. ISSTA 2022: 773-776

Achievements

- **Distinguished Paper Award** at ACM ICSE’11 (acceptance rate 14%)
- **32 awards** from the international competitions on software verification (SV-COMP) and testing (Test-Comp) 2012-2022 at **TACAS/FASE**
  - Bug finding
  - Cover error
- **Intel** deploys **ESBMC** in production as one of its verification engines for **verifying firmware in C**
- **Nokia** has found **security vulnerabilities** in **telecommunication software** written in **C++**

Research Mission

Automated **testing, verification** and **synthesis** to ensure the **security** in **embedded and IoT software**

↓

Methods, algorithms, and tools to write software with respect to security