Model Checking Boot Code in AWS Data Centers

Kareem Khazem
We proved the memory safety of boot code running in AWS data centers (CAV 2018).

Byron Cook
Serdar Tasiran
Kareem Khazem
Michael Tautschnig
Daniel Kroening
Mark R. Tuttle
Today:
- C Bounded Model Checker
- Boot code & safety
- Challenges
- Solutions
CBMC
2017 SV-COMP (falsification)
gcc -c .c
 gcc -c .c
make

.text
.data
.debug_str

.text
.data
.debug_str
make CC=goto-cc
make CC=goto-cc

goto-cc -c

goto-cc -c

.text
.data
.debug_str

.text
.data
.debug_str
make CC=goto-cc

goto-cc -c

.text
.data
.debug_str
goto-cc
goto-cc
make CC=goto-cc

goto-cc -c

goto-cc -c

goto-cc -o

goto-cc -o
LLDB, GDB, ...

∀x, y. ∃p. p(x) ⇒ p(y) ∧ ...
int foo(int flag, int x) {
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag) {
        int z = a[y];
    } else {
        int z = a[x];
    }
}
int foo(int flag, int x)
{
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}
int foo(int flag, int x)
{
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}
int foo(int flag, int x) {
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}

flag_1 = \bot
∧ x_1 = \bot
∧ a_1 = [0, 0, 0]
∧ y_1 = 2
int foo(int flag, int x) {
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}

flag₁ = ⊥ ∧ x₁ = ⊥ ∧ a₁ = [0, 0, 0] ∧ y₁ = 2 ∧ x₂ = x₁ + y₁
int foo(int flag, int x)
{
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}

flag₁ = ⊥
∧ x₁ = ⊥
∧ a₁ = [0, 0, 0]
∧ y₁ = 2
∧ x₂ = x₁ + y₁
∧ (flag₁
    ∧ z₁ = a₁[y₁]
)}
int foo(int flag, int x)
{
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}

\[
\begin{align*}
\text{flag}_1 & = \perp \\
\land x_1 & = \perp \\
\land a_1 & = [0, 0, 0] \\
\land y_1 & = 2 \\
\land x_2 & = x_1 + y_1 \\
\land (\text{flag}_1 \\
\land z_1 & = a_1[y_1] \\
\lor \neg \text{flag}_1 \\
\land z_1 & = a_1[x_2])
\end{align*}
\]
int foo(int flag, int x) {
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}

flag₁ = ⊥ ∧ x₁ = ⊥

∧ a₁ = [0, 0, 0]
∧ y₁ = 2
∧ x₂ = x₁ + y₁
∧ (flag₁ ∧ z₁ = a₁[y₁])
∨ ¬flag₁
∧ z₁ = a₁[x₂])
∧ (bounds-violated
∨ arith-overflow)
∨ ...
int foo(int flag, int x)
{
    int a[3] = {0};
    int y = 2;
    x = x + y;
    if(flag)
        int z = a[y];
    else
        int z = a[x];
}

flag₁ = 1
∧ x₁ = 1
∧ a₁ = [0, 0, 0]
∧ y₁ = 2
∧ x₂ = x₁ + y₁
∧ (flag₁
    ∧ z₁ = a₁[y₁]
    ∨ ¬flag₁
    ∧ z₁ = a₁[x₂])
∧ (bounds-violated
    ∨ arith-overflow)
∧ ...

assert(¬G)
Manufacturer: Intel
Brand String: Intel(R) Core(TM) i7 CPU 870
Frequency: 2.93GHz
BCLK Speed: 133MHz
Cache L1: 256 KB
Cache L2: 1024 KB
Cache L3: 8192 KB
Ratio Status: Unlocked (Min: 09, Max: 22)
Ratio Actual Value: 22
CPUID: 106E5

CPU Ratio Setting: [22.0]
C1E Support: [Enabled]
Hardware Prefetcher: [Enabled]
Adjacent Cache Line Prefetch: [Enabled]
Max CPUID Value Limit: [Disabled]
Intel(R) Virtualization Tech: [Enabled]
CPU TM Function: [Enabled]
Execute-Disable Bit Capability: [Enabled]

When disabled, force the XD feature flag to always return 0.
GNU GRUB  version 1.99-21ubuntu3

Ubuntu, with Linux 3.2.0-23-generic
Ubuntu, with Linux 3.2.0-23-generic (recovery mode)
Memory test (memtest86+)
Memory test (memtest86+, serial console 115200)

Use the + and - keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the commands before booting or 'c' for a command-line.
[OK] Started Cryptography Setup for luks-a8d6998e-c92a-4347-9175-a0249dd72903.
[OK] Started Cryptography Setup for luks-a8d6998e-c92a-4347-9175-a0249dd72903.
[TIME] Timed out waiting for device dev-disk-by-x2duuid-b4b59673-x2de85b-x2d4529-x2d840f-x2da9043a27a121.device
[OK] Reached target System Initialization.
[OK] Reached target Basic System.
[OK] Found device SAMSUNG_HD753LJ.
    Starting Cryptography Setup for luks-a8d6998e-c92a-4...a0249dd72903...
[OK] Found device /dev/disk/by-uuid/a93109f2-0a41-4737-88c0-a2c726ed8a47.
    Starting Cryptography Setup for luks-a93109f2-0a41-4737-88e0-a2c726ed8a47...
[OK] Started Show Plymouth Boot Screen.
[OK] Reached target Paths.
    Starting Forward Password Requests to Plymouth...
[OK] Started Forward Password Requests to Plymouth.
[OK] Found device /dev/mapper/luks-a93109f2-0a41-4737-88c0-a2c726ed8a47.
[OK] Found device /dev/disk/by-uuid/1aaa80a3-5043-4214-9ff2-8908d1417f3f.
[OK] Started Cryptography Setup for luks-a93109f2-0a41-4737-88c0-a2c726ed8a47.
[OK] Started Cryptography Setup for luks-a8d6998e-c92a-4347-9175-a0249dd72903.
[TIME] Timed out waiting for device dev-disk-by-x2duuid-b4b59673-x2de85b-x2d4529-x2d840f-x2da9043a27a121.device
[OK] Reached target System Initialization.
[OK] Reached target Basic System.
[201.440708] dracut-initqueue[297]: Warning: crypto LUKS UUID b4b59673-e85b-4529-840f-a9043a27a121 not found
Starting Dracut Emergency Shell...
Warning: crypto LUKS UUID b4b59673-e85b-4529-840f-a9043a27a121 not found
Generating "/run/initramfs-rdsosreport.txt"

Entering emergency mode. Exit the shell to continue.
Type "journalctl" to view system logs.
You might want to save "/run/initramfs-rdsosreport.txt" to a USB stick or /boot after mounting them and attach it to a bug report.

dracut: #
Prove:
∀(  ,  ,  ,  ).
¬(buffer_overflow(  )
   ∨ null_dereference(  )
   ∨ unallocated_access(  )
);
Challenges:

- Memory-mapped I/O
- Device behaviour
- Efficiency
- Linker scripts
int x = 7;
int *y = malloc(...);
int x = 7;
int *y = malloc(...);
int x = 7;
int *y = malloc(...);

#define REG_BASE        (0x1000)
#define REG_BOOT_STRAP  (REG_BASE + 0x110)
int x = 7;
int *y = malloc(...);

#define REG_BASE        (0x1000)
#define REG_BOOT_STRAP  (REG_BASE + 0x110)
#define REG_BASE        (0x1000)
#define REG_BOOT_STRAP  (REG_BASE + 0x110)
#define REG_BOOT_CONF   (REG_BASE + 0x124)

__CPROVER_allocated_memory(
    REG_BOOT_STRAP, 0x14);
Device Behaviour

int x = 7;

... = x;
Device Behaviour

int x = 7;
... = x;
Device Behaviour

```java
int x = 7;
... = x;
```
Device Behaviour

int x = 7;
... = x;
Device Behaviour

int access_register(...)
{
    ...
}

Device Behaviour

int access_register(...) {

}
Device Behaviour

int access_register(...)
{

}

0150
Device Behaviour

__CPROVER_mm_io_r(addr, size)
__CPROVER_mm_io_w(addr, size, val)
Efficiency

- Byte-level memory access
- memcpy implementation
- Tightly-coupled memory
- Device access
- Machine code manipulation
FAST  SLOW  ROM
MEMORY {
    SLOW: ORIGIN = 0x2000, LENGTH = 24k
    FAST: ORIGIN = 0x8000, LENGTH = 4k
    ROM: ORIGIN = 0x9000, LENGTH = 28k
}

.text : { 
    *(.text*)
} > ROM
MEMORY {
    SLOW: ORIGIN = 0x2000, LENGTH = 24k
    FAST: ORIGIN = 0x8000, LENGTH = 4k
    ROM:  ORIGIN = 0x9000, LENGTH = 28k
}

.text: {
    *(.text*)
} > ROM
MEMORY {
    SLOW: ORIGIN = 0x2000, LENGTH = 24k
    FAST: ORIGIN = 0x8000, LENGTH = 4k
    ROM: ORIGIN = 0x9000, LENGTH = 28k
}

.text: {
    text_start = .;
    *(.text*)
    text_end = .;
} > ROM

text_size = SIZEOF(.text)
MEMORY {
  SLOW: ORIGIN = 0x2000, LENGTH = 24k
  FAST: ORIGIN = 0x8000, LENGTH = 4k
  ROM: ORIGIN = 0x9000, LENGTH = 28k
}

.text: {
  text_start = .;
  *(.text*)
  text_end = .;
} > ROM

text_size = SIZEOF(.text)
extern char text_size[];
extern char text_start[];
extern char text_end[];

int main() {
    assert(&text_size == (char *)0x400);
    assert(&text_start == (char *)0xb000);
}

{ 1 KiB 0x0400 B
    .text
    0xb400
    0xb000
    text_start
    text_end
    text_size
}
extern char text_size[];
extern char text_start[];
extern char text_end[];

int main() {
    memcpy(buf,
          (void *)&text_start,
          (size_t *)&text_size);
}
The problem

extern char text_size[];
extern char text_start[];
extern char text_end[];

int main(){
    memcpy(buf,
        (void *)&text_start,
        (size_t)&text_size);
}
The problem

extern char text_size[];
extern char text_start[];
extern char text_end[];

int main(){
    memcpy(buf,
        (void *)&text_start,
        (size_t)&text_size);
}

harness_1.c
char text_size[];
&text_size = 0x0400;
The problem

extern char text_size[];
extern char text_start[];
extern char text_end[];

int main(){
    memcpy(buf,
           (void *) text_start,
           (size_t) text_size);
}

---

unsigned text_size  = 0x0400u;
unsigned text_start = 0xb000u;
text_start = 0xb000;
text_size = 0x0400;
__CPROVER_allocated_memory(0xb000, 0x0400);
Summary

- Wanted to prove absence of bugs in boot code

- Several challenges unique to low-level code

- We enhanced CBMC to overcome these challenges
Model Checking Boot Code in AWS Data Centers

Kareem Khazem
Backup slides
Decision Procedure for Arrays

"Weakly Equivalent Arrays".

Jürgen Christ, Jochen Hoenicke
(FroCos 2015).
Nitro Architecture


https://youtu.be/LabltEXk0VQ