Analyzing System Software Components Using API Model Guided Analysis

Tuba Yavuz and Ken (Yihang) Bai
tuba@ece.ufl.edu, baiyihang@ufl.edu
University of Florida

KLEE Workshop 2022
September 15th-16th
Imperial College, London
Motivation

• Analyzing system code using symbolic execution is challenging partly due to the complexity of Application Programming Interface (API)
• Modeling the API of complex system code is key to a scalable and precise analysis
• Techniques that automatically learn the behavior of APIs are not mature yet
  • Automated synthesis: the state space is often huge, guided by some syntax and/or input/output samples
  • Machine learning: still exists a big semantic gap between program representations and those used by the learning algorithms
Motivation

• Under-constrained symbolic execution as implemented in UC-KLEE can support component-level analysis
  • UC-KLEE is not open-source
  • The goal of modeling is UC-KLEE to filter out false positives

• PROMPT approach for component-level analysis for system code
  • API modeling and API model guided symbolic execution
  • Better control for scalability and precision
Solution: PROMPT & PROSE

Component under analysis

PROMPT (extends KLEE)

Memory errors (and other analysis results)

PROSE API models

https://github.com/sysrel/PROMPT

PROMPT: API Model Guided Symbolic Execution

- Performs under-constrained symbolic execution
  - No need for a test driver
- Performs lazily initialization according to the specified API model
- Extends the KLEE symbolic execution engine to incorporate API modeling rules during handling of various instructions
  - Memory access instructions
  - Function calls
  - Return instructions
- Has been applied to real-world system code
  - Linux device drivers, cryptographic libraries, BlueZ
Modeling with PROSE

• PROSE is the API specification language for PROMPT
• Consists of four parts:
  • Global settings
  • Data modeling
  • Function modeling
  • Lifecycle modeling
Modeling Choices for Functions

- Should we model `foo` by abstracting away the function body and symbolizing the return value?
- If return value is a pointer type, should we also create NULL returning cases when using a model for `foo`?
- If `foo` is an entry point, should we use a constant value or a symbolic value for an argument?
- If the argument is a pointer type and to be symbolized, what should be the size of the array? A constant value or an expression involving another argument of the same function?
- If `foo` has inline assembly should we model `foo` automatically?
- Should we execute `foo`’s body and symbolize the return value of inline assembly?
- Should `foo` be modeled as a memory (de)allocator?
- Should we model `foo` with another function definition?
- Should entry-point `foo` be followed by `bar` on a specific return value?
- Should we havoc the arguments to model the side-effect of `foo`?
Modeling Choices for Lazy Initialization of Struct Types

```c
struct TypeA {
    primitiveType field_m;
    ...
    functionPointerType field_j;
    ...
    pointerType field_i;
    ...
    struct TypeB *field_k;
}
```

Is `TypeA` a singleton type?

How should we constrain the primitive type field(s) of `TypeA`?

How should we constrain the size of the array for the pointer field?

Should we initialize the function pointer field in `TypeA` to NULL or to a specific function?

Should we assume a `TypeB` object is embedded inside a `TypeA` object when lazy initializing `field_k`?
Pointer arithmetic using a negative offset

Memory out of bounds error when dereferenced!

Just tell PROMPT that type A embeds type B

(False positive due to imprecise context!)
Specifying type embedding relationships

PROSE API MODEL

global settings:
data models:
  type A embeds type B;
function models:
lifecycle model:
  entry-point foo

COMPONENT UNDER ANALYSIS

```c
#define cont_of(ptr, type, member) ({
    // container_of
    void *__mptr = (void *)(ptr);
    ((type *)__mptr - offsetof(type, member));})

struct A {
    int a;
    struct B b;
    char c;
};

int foo(int x, struct B *b) { struct A *ep;
    L1: ep = cont_of(b, struct A,b);
    L2: if (x > 0)
    L3: ep->a = x;
    L4: else
    L5: ep->a = -x;
    L6: return ep->a;
}
```

STEPS:

```
$ export PROMPT=/home/prompt/prompt_build_dir/bin/klee
$ cd PROMPT_examples
$ cd data_modeling/embeds
$ ./run.sh foo.bc 2>&1 | tee o.txt
$ ls -l klee-last/
```

PROMPT RESULTS:

```
# of paths: 2
No memory errors at lines L3, L5, and L6
```

10
BlueBorne CVE-2017-1000251

• Bluetooth is a very popular wireless short range communication protocol
• One of the vulnerabilities dubbed as BlueBorne is in BlueZ
  • BlueZ is the Bluetooth stack used in the Linux kernel
  • The vulnerability can be exploited to perform remote code execution when the extended flow specification (EFS) feature is utilized in L2CAP
• https://github.com/sysrel/PROMPT/tree/master/JASE_benchmarks/bluez/l2cap_config_rsp
• Directory on the VM
  • $ cd /home/prompt/PROMPT/JASE_benchmarks/bluez/l2cap_config_rsp
• How to run
  • $ export PROMPT=/home/prompt/prompt_build_dir/bin/klee
  • $ ./run.sh 2>/dev/null 1>/dev/null
• Check the result
  • $ more klee-last/test000069.ptr.error
BlueBorne CVE-2017-1000251 in a nutshell

```c
static inline int l2cap_config_rsp(struct l2cap_conn *conn, struct l2cap_cmd_hdr *cmd, u8 *data)
{ ...
    switch (result) { ...
    case L2CAP CONF PENDING:
        set_bit(CONF REM CONF PEND, &chan->conf_state);
        if (test_bit(CONF LOC CONF PEND, &chan->conf_state)) {
            char buf[64]; // the buffer involved in the overflow
            len = l2cap_parse_conf_rsp(chan, rsp->data, len, buf, &result);
        }
    }
    ...
}
```

// No checks on the buffer size!

```c
static void l2cap_add_conf_opt(void **ptr, u8 type, u8 len, unsigned long val)
{ ...
    struct l2cap_conf_opt *opt = *ptr;
    ...// write to the buffer
    *ptr += L2CAP CONF OPT SIZE + len;
}
```
global settings:

array size 128;

model funcs with asm off;

symbolize inline asm on;

data models: ...

function models: ...

lifecycle model: ...

static inline int l2cap_config_rsp(
    struct l2cap_conn *conn,
    struct l2cap_cmd_hdr *cmd,
    u8 *data)
{
    ...  
    switch (result) { ...
    case L2CAP_CONF_PENDING:
        char buf[64]; // the buffer involved in the overflow
        len = l2cap_parse_conf_rsp(chan, rsp->data,
            len, buf, &result);
    ...
}
Pointer arithmetic using a negative offset

Just tell PROMPT that type l2cap_chan embeds type list_head

```c
static inline int l2cap_config_rsp(
    struct l2cap_conn *conn,
    struct l2cap_cmd_hdr *cmd,
    u8 *data)
{
    ... chan = l2cap_get_chan_by_scid(conn, scid);
    if (!chan) return 0;
    switch (result) { ... case L2CAP_CONF_PENDING:
        char buf[64]; // the buffer involved in the overflow
        len = l2cap_parse_conf_rsp(chan, rsp->data,
            len, buf, &result);
    ... }
```
COMPONENT UNDER ANALYSIS

static inline int l2cap_config_rsp(
    struct l2cap_conn *conn,
    struct l2cap_cmd_hdr *cmd,
    u8 *data)
{
    ... 
    chan = l2cap_get_chan_by_scid(conn, scid);
    if (!chan) return 0;
    switch (result) {
    case L2CAP_CONF_PENDING:
        char buf[64]; // the buffer involved in the overflow
        len = l2cap_parse_conf_rsp(chan, rsp->data,
                                    len, buf, &result);
    ...

PROSE API MODEL

data models:

    type l2cap_chan embeds type list_head;

    (x = l2cap_sock_state_change_cb) where l2cap_sock_state_change_cb
    is function,
    x    is l2cap_ops field  5;

    (x = l2cap_sock_ready_cb) where l2cap_sock_ready_cb is function,
    x    is l2cap_ops field  6;

    (x = l2cap_sock_suspend_cb) where l2cap_sock_suspend_cb is
    function,
    x    is l2cap_ops field  9;
function models:

 alloc usb_alloc_coherent sizearg 1 initzero true symbolize false;
 alloc __kmalloc sizearg 0 initzero true symbolize false;
 alloc vzalloc sizearg 0 initzero true symbolize false;
 free kfree memarg 0;
 free vfree memarg 0;

returnonly skb_clone;
returnonly kfree_skb;
returnonly l2cap_send_cmd;
returnonly hci_send_cmd;
returnonly l2cap_clear_timer;
...

lifecycle model:
entry-point l2cap_config_rsp

static inline int l2cap_config_rsp(
    struct l2cap_conn *conn,
    struct l2cap_cmd_hdr *cmd,
    u8 *data)
{
    ...  
    chan = l2cap_get_chan_by_scid(conn, scid);
    if (!chan) return 0;
    switch (result) {
    case L2CAP_CONF_PENDING:
        char buf[64]; // the buffer involved in the overflow
        len = l2cap_parse_conf_rsp(chan, rsp->data,
            len, buf, &result);
    ...
static inline int l2cap_config_rsp(struct l2cap_conn *conn, struct l2cap_cmd_hdr *cmd, u8 *data)
{
    chan = l2cap_get_chan_by_scid(conn, scid);
    if (!chan) return 0;
    switch (result) {
    case L2CAP_CONF_PENDING:
        char buf[64];          // the buffer involved in the overflow
        len = l2cap_parse_conf_rsp(chan, rsp->data, len, buf, &result);
        ...

int l2cap_get_conf_opt_PROSE(void **ptr, int *type, int *olen, unsigned long *val)
{
    struct l2cap_conf_opt *opt = *ptr;
    int len;

    opt->type = 1;  // L2CAP_CONF_MTU
    opt->len = 2;

    len = L2CAP_CONF_OPT_SIZE + opt->len;
    *ptr += len;

    *type = opt->type;
    *olen = opt->len;

    return len;
}
Error: memory error: out of bound pointer
Line: 2996
assembly.ll line: 43986
Stack:
    #000043986 in l2cap_add_conf_opt (ptr=142165200, type=1, len=2, val=43947) at /home/tuba/Documents/tools/clang-kernel-build/linux-stable/net/bluetooth/l2cap_core.c:2996
    #100048020 in l2cap_parse_conf_rsp (chan=92475136, rsp=141640710, len, data=141847760, result=141548608) at /home/tuba/Documents/tools/clang-kernel-build/linux-stable/net/bluetooth/l2cap_core.c:3551
    #200047685 in l2cap_config_rsp (conn=141657120, cmd=141631360, cmd_len, data=141640704) at /home/tuba/Documents/tools/clang-kernel-build/linux-stable/net/bluetooth/l2cap_core.c:4186

PROMPT detects the BlueBorne vulnerability within 5 minutes!
PROMPT Evaluation

• Developed registration and deregistration API models for the video, sound, and network subsystems of the Linux kernel

• Analyzed the probe and disconnect entry points of 57 drivers from the video, sound, and network subsystems

• Compared Programming Model Guided Execution with Lazy Initialization only in terms of coverage and error rate
PROMPT (PMGSE) vs Lazy Init on Coverage

- **PMGSE**: API Model Guided Symbolic Execution
- PMGSE-RED eliminates some of the manual modeling effort compared to PMGSE-FULL

Linux device drivers from video, sound, and network subsystems
PROMPT (PGMSE) vs Lazy Init on Error Rate

Error Rate Comparison

- Percentage of Error Paths
- Error Rate Comparison for video, sound, and network categories
- Three categories: PROMPT-FULL, PROMPT-RED, LAZY INIT ONLY
New Bugs Found with PROMPT

NULL pointer dereference in the cs46xx sound driver

--- a/sound/pci/cs46xx/dsp_spos.c
+++ b/sound/pci/cs46xx/dsp_spos.c
@@ -899,6 +899,9 @@ int cs46xx_dsp_proc_done (struct snd_cs4
 struct dsp_spos_instance * ins = chip->dsp_spos_instance;
 int i;
 +if (!ins)
 +return 0;
 +snd_info_free_entry(ins->proc_sym_info_entry);
 ins->proc_sym_info_entry = NULL;

Double-free in the hso network driver

--- a/drivers/net/usb/hso.c.orig
+++ b/drivers/net/usb/hso.c
@@ -2377,7 +2377,9 @@ static void hso_free_net_device(struct h
 remove_net_device(hso_net->parent);
 -if (hso_net->net)
 +if (hso_net->net &
 +hso_net->net->reg_state == NETREG_REGISTERED)
 unregister_netdev(hso_net->net);

API Misuse: Do not unregister devices that are not registered yet!
Conclusions

• PROMPT supports a variety of API modeling to support component-level analysis
  • Check out https://github.com/sysrel/PROMPT

• If you use PROMPT for your work, please cite our paper

• This work has been partially funded by
  • National Science Foundation (NSF) awards CNS-1815883 and CNS-1942235
  • Semiconductor Research Corporation (SRC)
Thank you - Questions?