RTCON: Context-Adaptive Function-Level Fuzzing for RTOS Kernels

Eunkyu Lee, Junyoung Park, Insu Yun
KAIST, School of Electrical Engineering

KAIST

Agenda

01

Introduction of RTCon

- Challenges of Existing Works
- RTCon Design
- How It Works

02

Tutorial

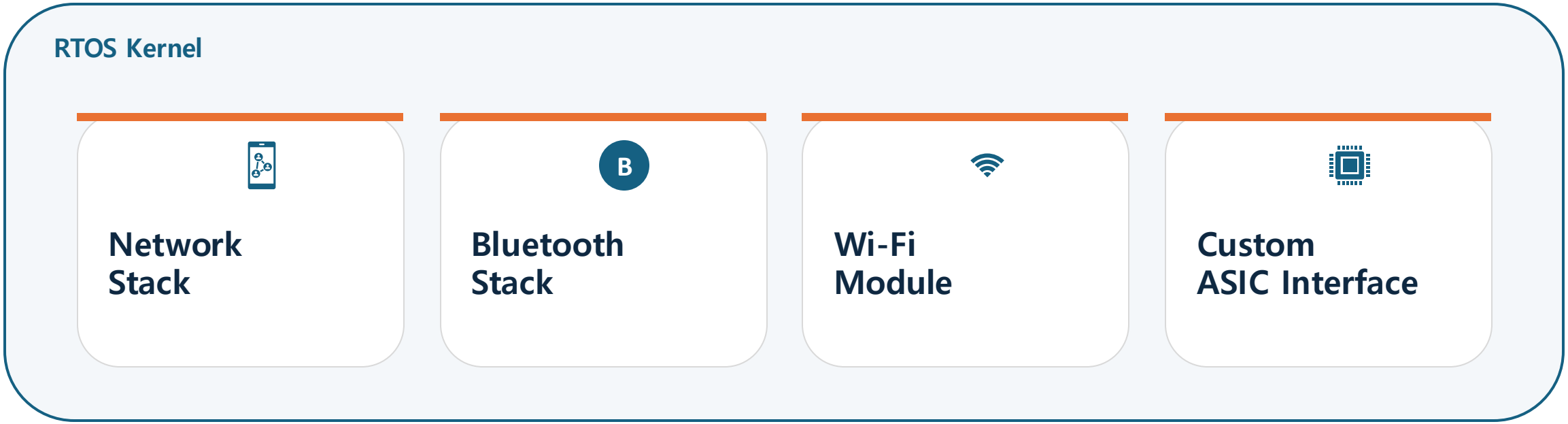
- Applying to RTOS
- As a General Library

Download Links for Docker Image

- Please download the RTCON images for the tutorial session.
- Building them from scratch takes a significant amount of time.
- Links
  - Zephyr Image : https://drive.google.com/file/d/1uUm5XVDYGkxpD0dWaairF2eQYKJgr0ga/view?usp=drive_link
  - FreeRTOS Image (11.74 GB) : https://drive.google.com/file/d/1cIKahrsSsH3m6iyAK51OT6JEMi9bcsaI/view?usp=drive_link
  - ThreadX Image (10.47 GB) : https://drive.google.com/file/d/1WLTclu5BevH59jM5saNX_yhdXDIPXaD2/view?usp=drive_link
  - **RIOT Image (will be used in tutorial session, 10.12 GB) :** https://drive.google.com/file/d/1lHtO9sCj2jLA3Mvy91S4LpPCzJTPHavB/view?usp=drive_link

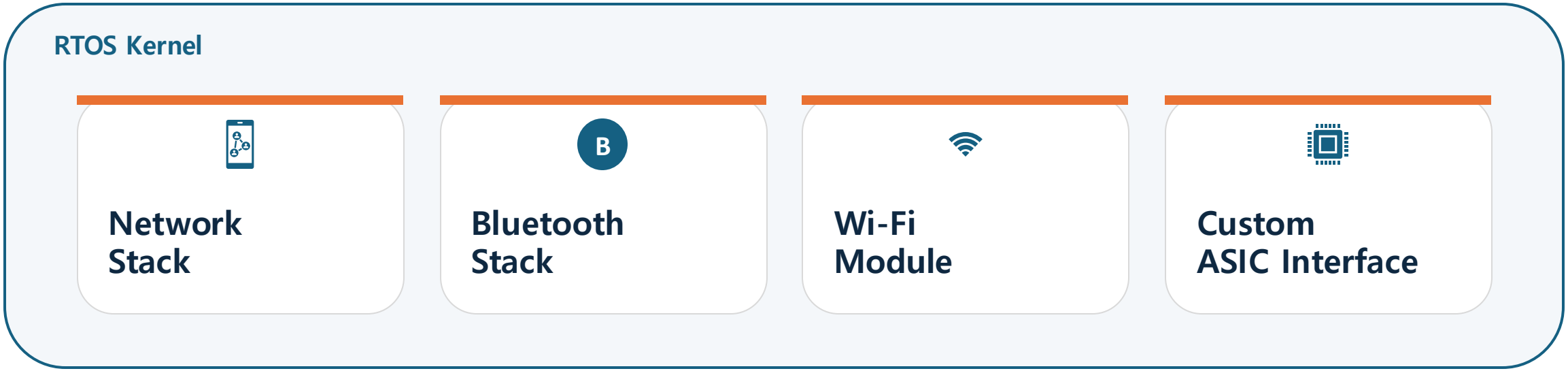
Security Challenges in RTOS kernels

- RTOS kernels expose many subsystems — Bluetooth, Wi-Fi, network stacks, custom peripherals.
- Diverse functionality **expands the external attack surface**.
- Even simple bugs can lead to **RCE or DoS** attacks.



Security Challenges in RTOS kernels

- RTOS kernels expose many subsystems — Bluetooth, Wi-Fi, network stacks, custom peripherals.
- Diverse functionality **expands the external attack surface**.
- Even simple bugs can lead to **RCE or DoS** attacks.



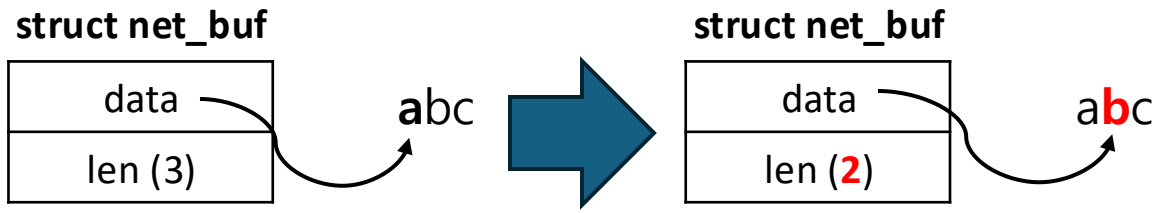
Even simple vulnerabilities are hard to find with existing fuzzing tools.

Motivating Example (CVE-2024-8798)

avdtp_process_configuration parses BT packet data **buf** using **net_buf_pull_u8**.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

Before `net_buf_pull_u8` **After `net_buf_pull_u8`**

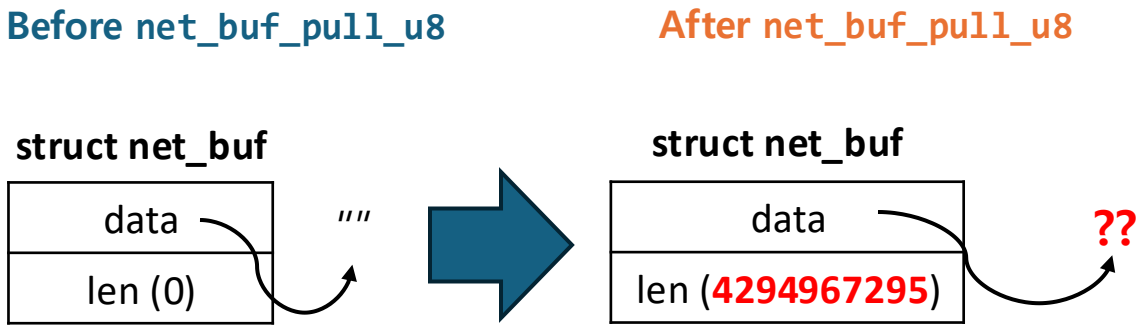


Pulls one byte from `buf` and decrements `buf->len`.

Motivating Example (CVE-2024-8798)

avdtp_process_configuration calls **net_buf_pull_u8** without checking **buf->len**.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

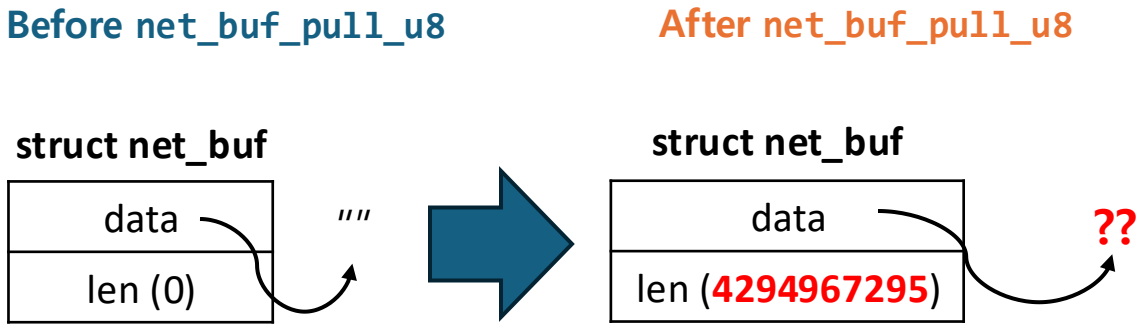


When `buf->len == 0`, pulling another byte reads past the buffer → OOB read.

Motivating Example (CVE-2024-8798)

avdtp_process_configuration calls **net_buf_pull_u8** without checking **buf->len**.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```



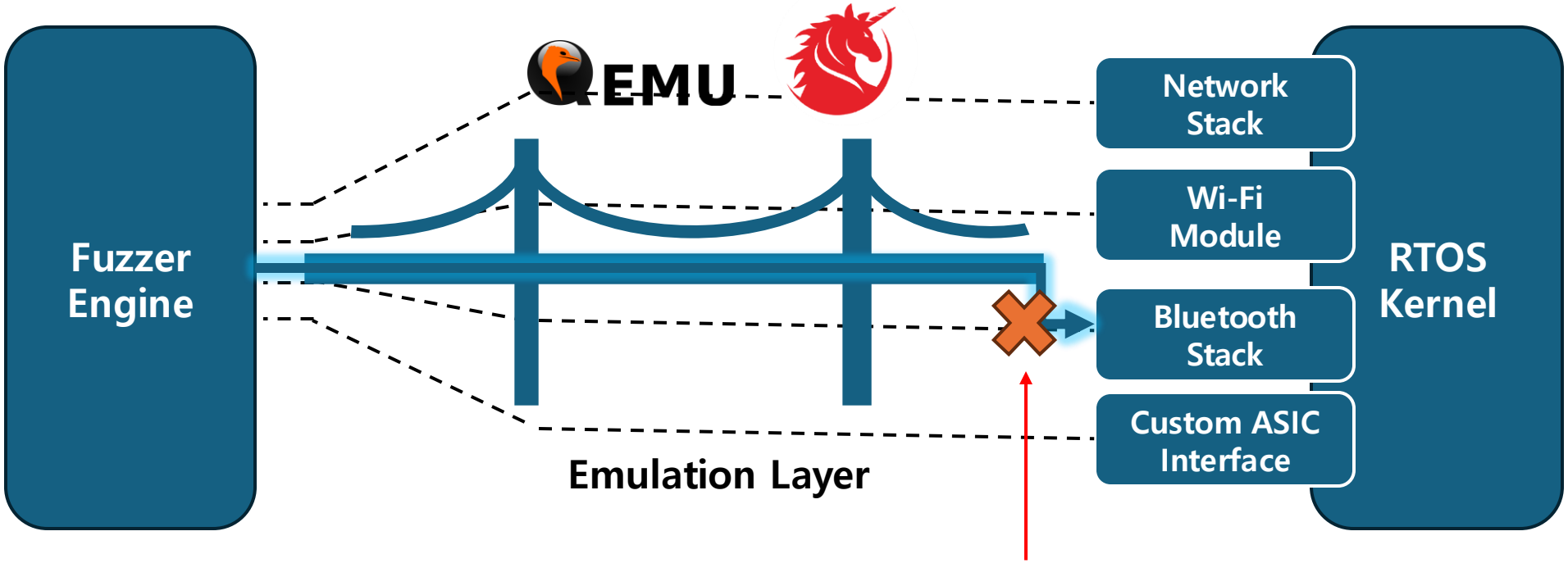
Simple OOB read

... yet hard to find with existing fuzzing tools.

Limitations of Existing Fuzzing Tools

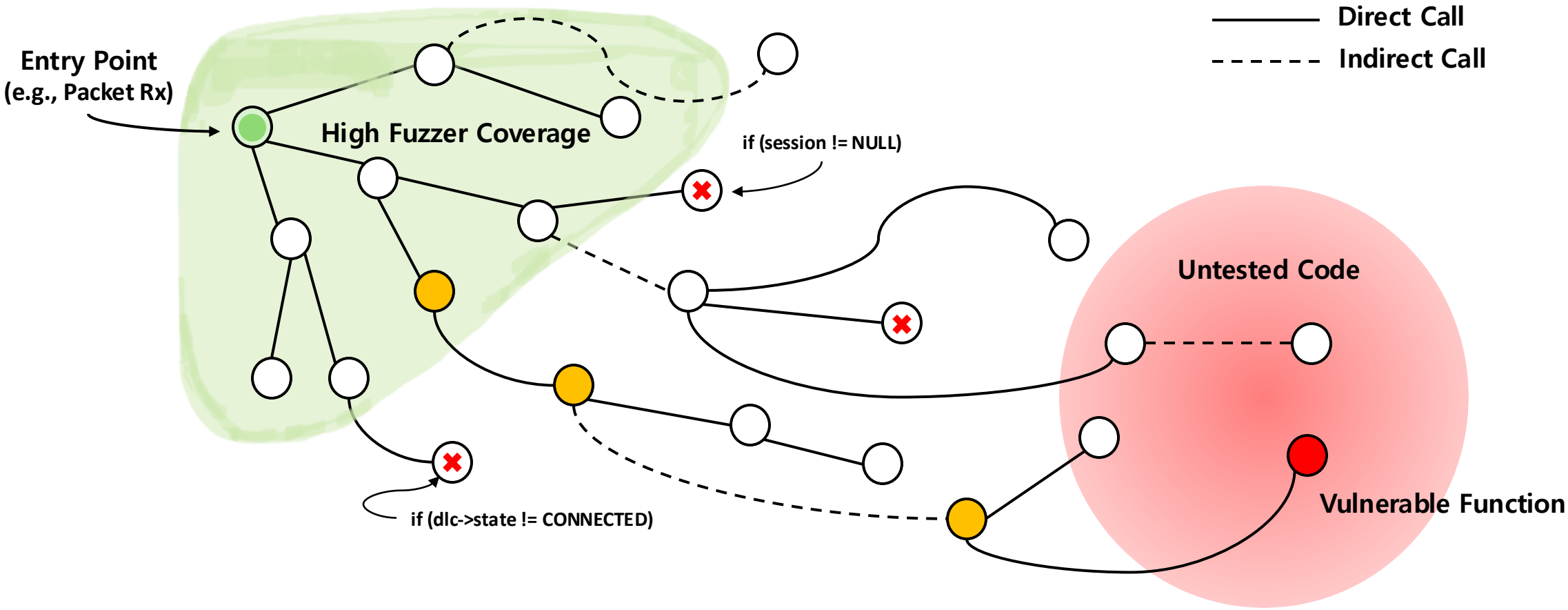
C1. Limitations of Embedded System Fuzzers

- Dynamic testing typically relies on **emulating the target system**.
- Coverage is restricted to a **partial subset** of RTOS subsystems.



Limitations of Existing Fuzzing Tools

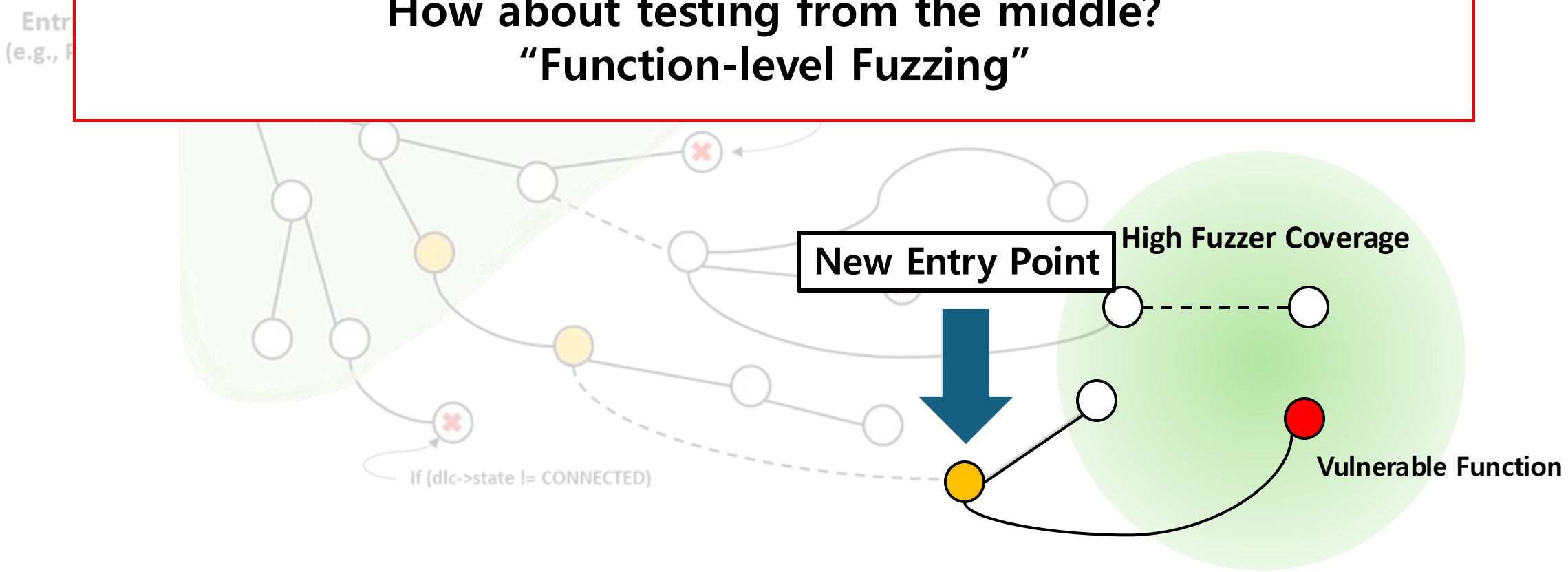
C2. Many vulnerabilities reside in deeply nested functions.



Function-level Fuzzing

Fuzzing deeply located functions without emulation.

How about testing from the middle?
“Function-level Fuzzing”



Limitations of Function-level Fuzzing

Function-level fuzzing requires parameters as inputs.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);  Vulnerable!!
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

buf is a user input

Can we reach the vulnerable code?

Limitations of Function-level Fuzzing

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);  Vulnerable!!
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

Function parameters as inputs?

PARAMETER	KIND	INPUT?
buf	user input	OK
session	context	?
msg_type	context	?
tid	context	?

How to handle context variables?

Limitations of Function-level Fuzzing

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);  Vulnerable!!
25
26
```

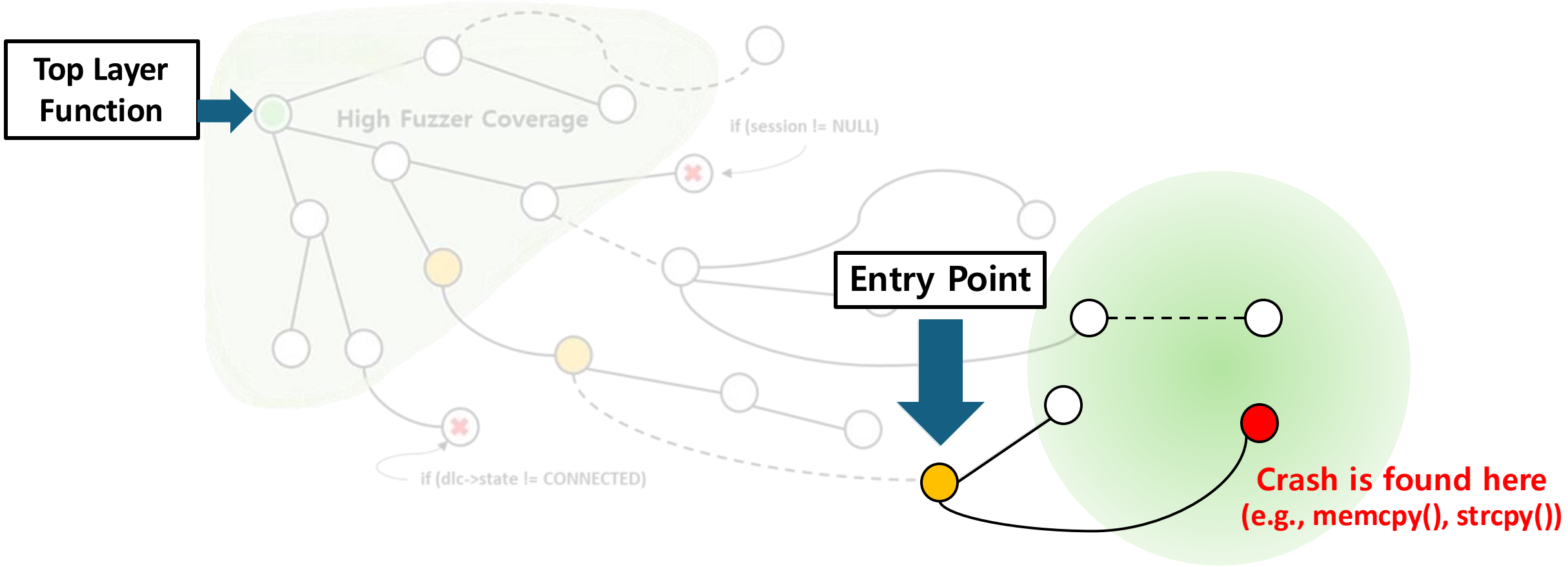
Function parameters as inputs?

PARAMETER	KIND	INPUT?
buf	user input	OK
session	context	?
msg_type	context	?
tid	context	?

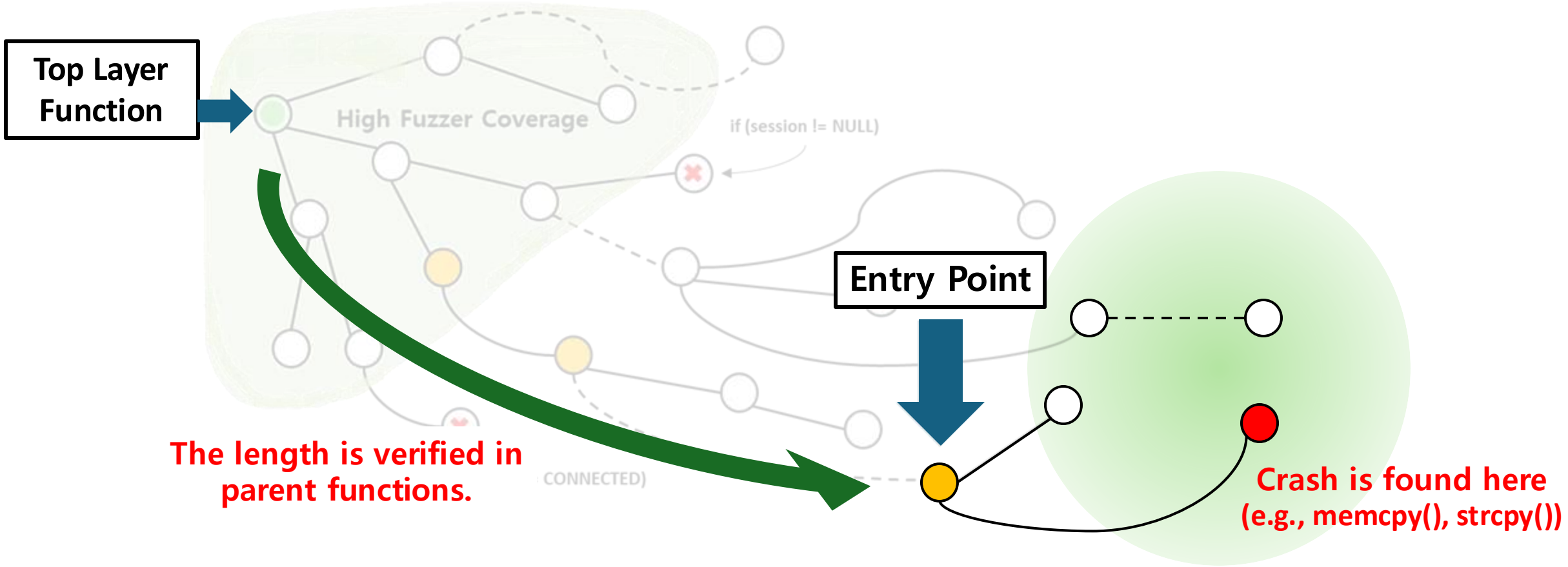
context variables!

Early termination before reaching the vulnerable code → False positives.

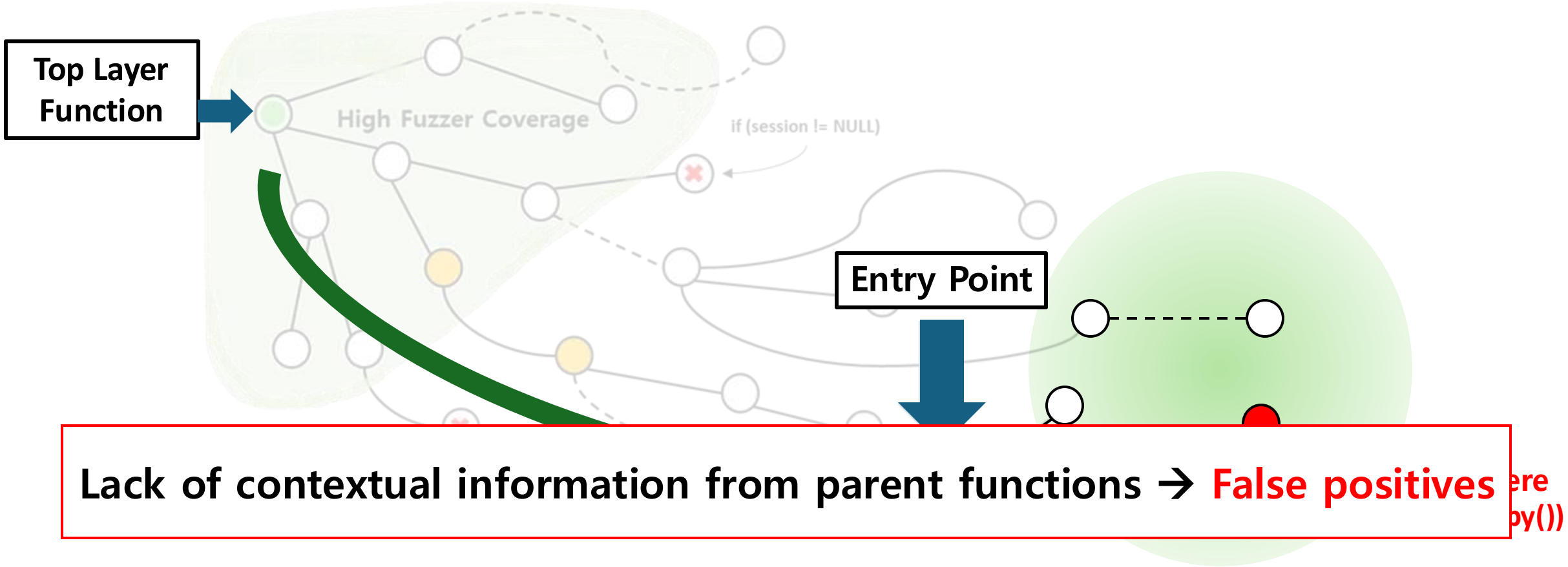
False Positives of Function-level Fuzzing



False Positives of Function-level Fuzzing



False Positives of Function-level Fuzzing



Our Approaches

C1. Reachability & Scalability

Traditional fuzzers **cannot reach** deep kernel functions.

A1. Function-level Fuzzing

Directly fuzz **any target function**.

C2. Missing Function Context

Calling a function directly **lacks the necessary contexts**.

A2. Adaptive Context Generation

Generate required context values **on-demand**.

C3. High False Positives

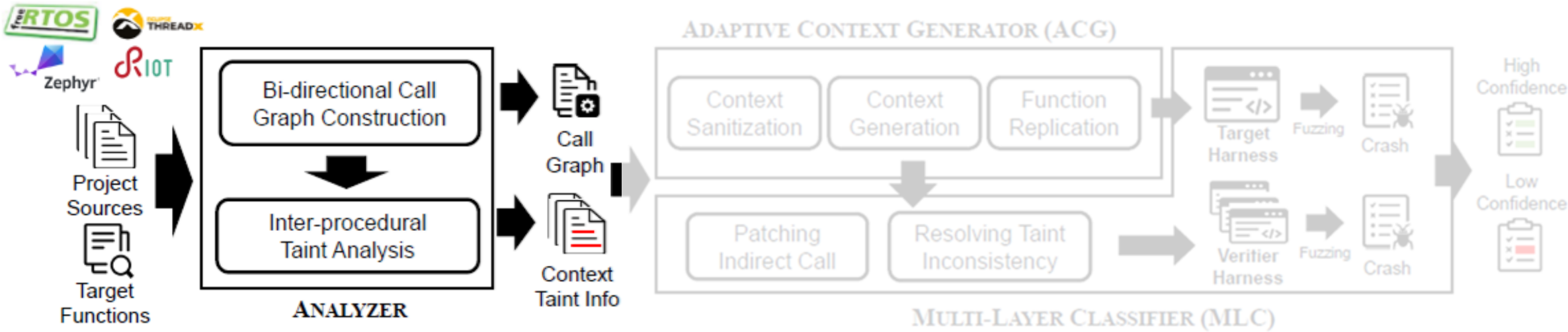
Function-level fuzzing creates many **invalid crashes**.

A3. Multi-layer Classification

Verify crashes by testing **higher-level functions**.

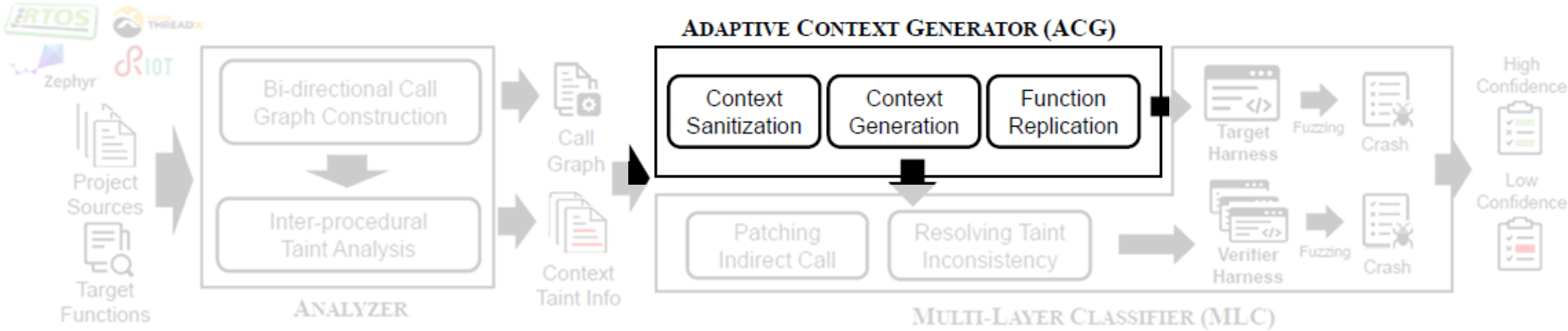
RTCON: Context-Adaptive Function-Level Fuzzing for RTOS Kernels

① RTCON performs call graph analysis & inter-procedural taint analysis.



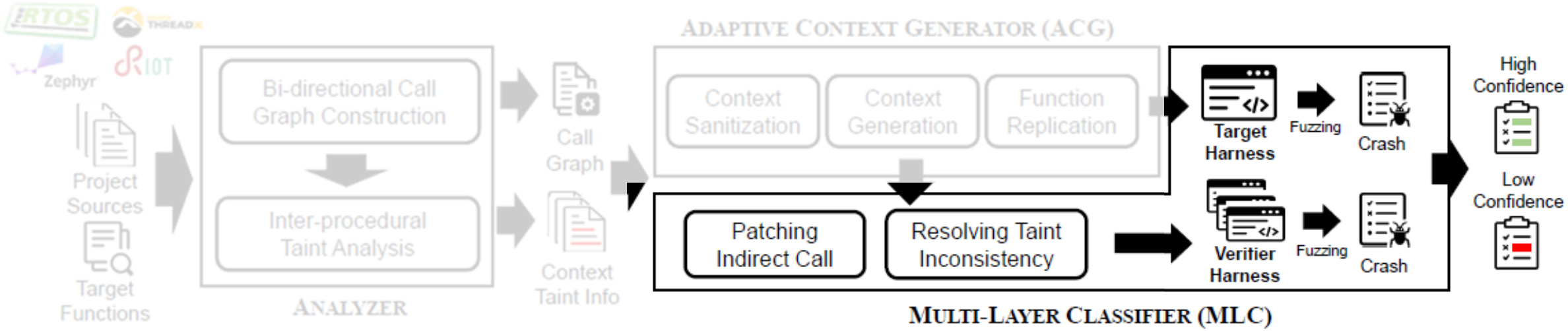
RTCON: Context-Adaptive Function-Level Fuzzing for RTOS Kernels

② **RTCON installs context sanitization & generation hooks on tainted (context-related) variable s.**



RTCON: Context-Adaptive Function-Level Fuzzing for RTOS Kernels

③ RTCON performs multi-layer classification by fuzzing both top-layer functions and the target function.



Design

① **Adaptive Context Generation**

② **Inter-procedural Taint Analysis**

③ **Multi-layer Classification**

Adaptive Context Generation: Challenges #1

Challenge: Dereferencing context variables is highly likely to cause a crash.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

Crash

Crash

`session` is a context variable.

Adaptive Context Generation: Our Approach

Instruments hook functions to catch potential faults and sanitize them.

It allocates the required memory on-the-fly.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14  if ((sep == NULL) ||
15      session = sanitizeLoad(session);
16      session->ops = sanitizeLoad(session->ops);
17       (session->ops->set_configuration_ind == NULL)) {
18      err = -ENOTSUP;
19    } else {
20       if (sep->state == AVDTP_STREAMING) {
21          err = -ENOTSUP;
22  }
23   ...
```

Hook function is inserted before dereferencing context variables.

Adaptive Context Generation: Hook Functions

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14  if ((sep == NULL) ||
15      session = sanitizeLoad(session);
16      session->ops = sanitizeLoad(session->ops);
17       (session->ops->set_configuration_ind == NULL)) {
18      err = -ENOTSUP;
19    } else {
20       if (sep->state == AVDTP_STREAMING) {
21          err = -ENOTSUP;
22  }
23   ...
```

In LLVM IR

```
%sess          = .... (parameter)
...
%ops           = Load %sess
%set_conf      = Load %ops
call %set_conf
```

```
...
%sess_new      = call sanitizeLoad
%ops           = Load %sess_new
%ops_new       = call sanitizeLoad
%set_conf      = Load %ops_new
```

Adaptive Context Generation: Hook Functions

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14  if ((sep == NULL) ||
15      session = sanitizeLoad(session);
16      session->ops = sanitizeLoad(session->ops);
17       (session->ops->set_configuration_ind == NULL)) {
18      err = -ENOTSUP;
19    } else {
20      if (sep->state == AVDTP_STREAMING) {
21        err = -ENOTSUP;
22  }
23   ...
```

In LLVM IR

```
%sess          = .... (parameter)
...
%ops           = Load %sess
%set_conf      = Load %ops
call %set_conf
```

```
...
%sess_new      = call sanitizeLoad
%ops           = Load %sess_new
```

On-Demand Allocation ✅

How does the fuzzer pass **complex conditional branches**?

Adaptive Context Generation: Challenges #2

Challenge: Context variables need to satisfy the branch conditions.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf); vulnerable!!
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

msg_type is also a context variable.

msg_type should be BT_AVDTP_CMD to reach the vulnerable code path.

Adaptive Context Generation: Our Approach

Assign values close to the operand values to the context variables.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

Before comparison, assign context values to `msg_type`.

Candidates:

- `BT_AVDTP_CMD`
- `BT_AVDTP_CMD - 1`
- `BT_AVDTP_CMD + 1`
- `Original Value`

Adaptive Context Generation: Hook Functions

Instruments hook functions to generate appropriate context values.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8     msg_type = generateCTX(BT_AVDTP_CMD);
9     if (msg_type == BT_AVDTP_CMD) {
10        int err = 0;
11        struct bt_avdtp_sep *sep;
12
13        // Get the stream endpoint from id
14        sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
15        if ((sep == NULL) ||
16            session = sanitizeLoad(session);
17            session->ops = sanitizeLoad(session->ops);
18            new_set_configuration_ind = generateCTX(NULL);
19            (new_set_configuration_ind == NULL)) {
20        } else {
21        ...
```

In LLVM IR

```
%msgtype        = .... (parameter)
...
cmpInst         %msgtype, %BT_AVDTP_CMD
```

```
%msgtype        = .... (parameter)
...
%newmsgtype = call generateCTX(%BT...)
cmpInst         %newmsgtype, %BT_...
```

Adaptive Context Generation: Hook Functions

Instruments hook functions to generate appropriate context values.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8     msg_type = generateCTX(BT_AVDTP_CMD);
9     if (msg_type == BT_AVDTP_CMD) {
10        int err = 0;
11        struct bt_avdtp_sep *sep;
12
13        // Get the stream endpoint from id
14        sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
15        if ((sep == NULL) ||
16            session = sanitizeLoad(session);
17
18
19
20
21
```

In LLVM IR

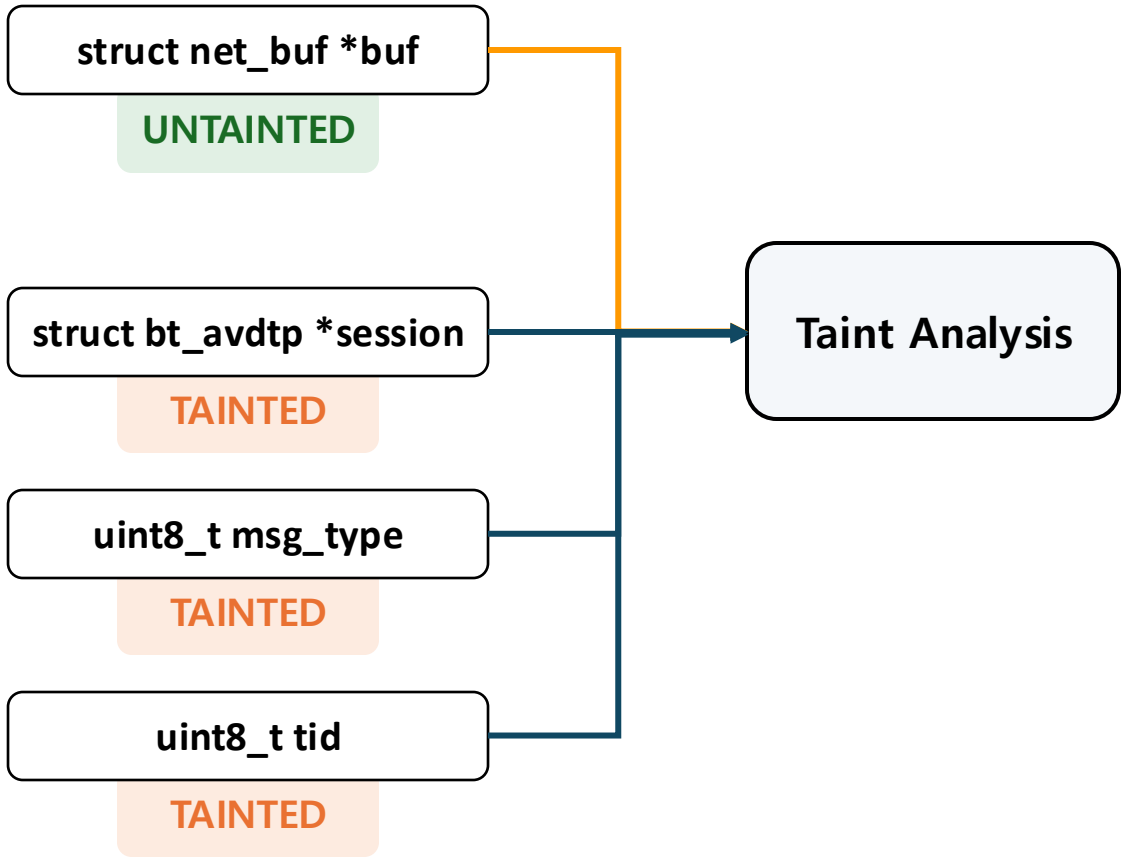
```
%msgtype        = .... (parameter)
...
cmpInst         %msgtype, %BT_AVDTP_CMD
```

On-Demand Allocation ✅
Generating Context Values Adaptively ✅
How do we know which variable is context-related?

Taint Analysis: Identify Context Variables

Goal: Identify all context variables within the function.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```



Taint Analysis: Identify Context Variables

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_code);
```

After iterative taint analysis, it can identify all context-related variables.

Tainted

Untainted

Hook Functions on Context Variables

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24        int_seid = net_buf_pull_u8(buf);
25        err = session->ops->set_configuration_ind(session,
26                                                  sep, int_seid, buf, &error_cod
```

Tainted

Untainted

Install context sanitization & generation hooks on tainted variables.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   new_msg_type = generateCTX(BT_AVDTP_CMD);
9   if (new_msg_type == BT_AVDTP_CMD) {
10    int err = 0;
11    struct bt_avdtp_sep *sep;
12
13    // Get the stream endpoint from id
14    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
15    new_sep = generateCTX(NULL);
16    if ((sep == NULL) ||
17        session = sanitizeLoad(session);
18        session->ops = sanitizeLoad(session->ops);
19        new_set_configuration_ind = generateCTX(NULL);
20        (new_set_configuration_ind == NULL)) {
21    } else {
22    ...
```

Hook Functions on Context Variables

Install context sanitization & generation hooks on tainted variables.

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   if (msg_type == BT_AVDTP_CMD) {
9     int err = 0;
10    struct bt_avdtp_sep *sep;
11
12    // Get the stream endpoint from id
13    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
14    if ((sep == NULL) ||
15        (session->ops->set_configuration_ind == NULL)) {
16      err = -ENOTSUP;
17    } else {
18      if (sep->state == AVDTP_STREAMING) {
19        err = -ENOTSUP;
20      } else {
21        uint8_t int_seid;
22        // No check for remaining buffer size
23        // Out-of-bounds read when it tries to pull 1 byte
24
25
26
```

```
6 static void avdtp_process_configuration(struct bt_avdtp *session,
7                                         struct net_buf *buf, uint8_t msg_type, uint8_t tid) {
8   new_msg_type = generateCTX(BT_AVDTP_CMD);
9   if (new_msg_type == BT_AVDTP_CMD) {
10    int err = 0;
11    struct bt_avdtp_sep *sep;
12
13    // Get the stream endpoint from id
14    sep = avdtp_get_sep(net_buf_pull_u8(buf) >> 2);
15    new_sep = generateCTX(NULL);
16    if ((sep == NULL) ||
17        session = sanitizeLoad(session);
18        session->ops = sanitizeLoad(session->ops);
```

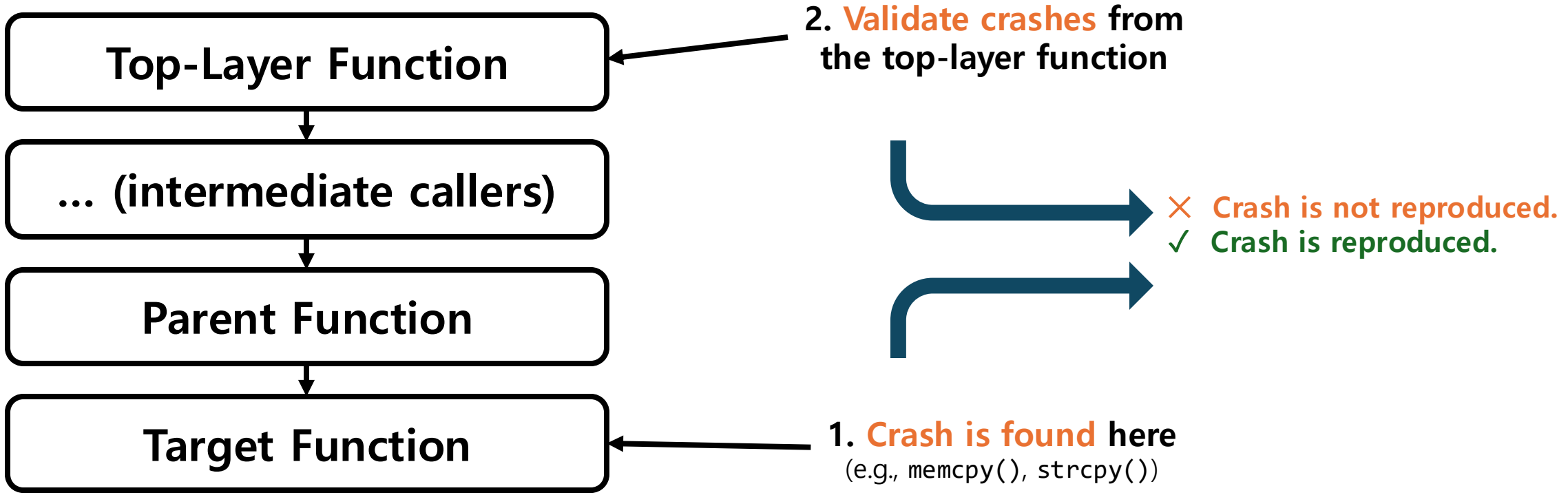
More details on the paper

Function call sanitization, function replication (1-KFA), indirect call.

Untainted

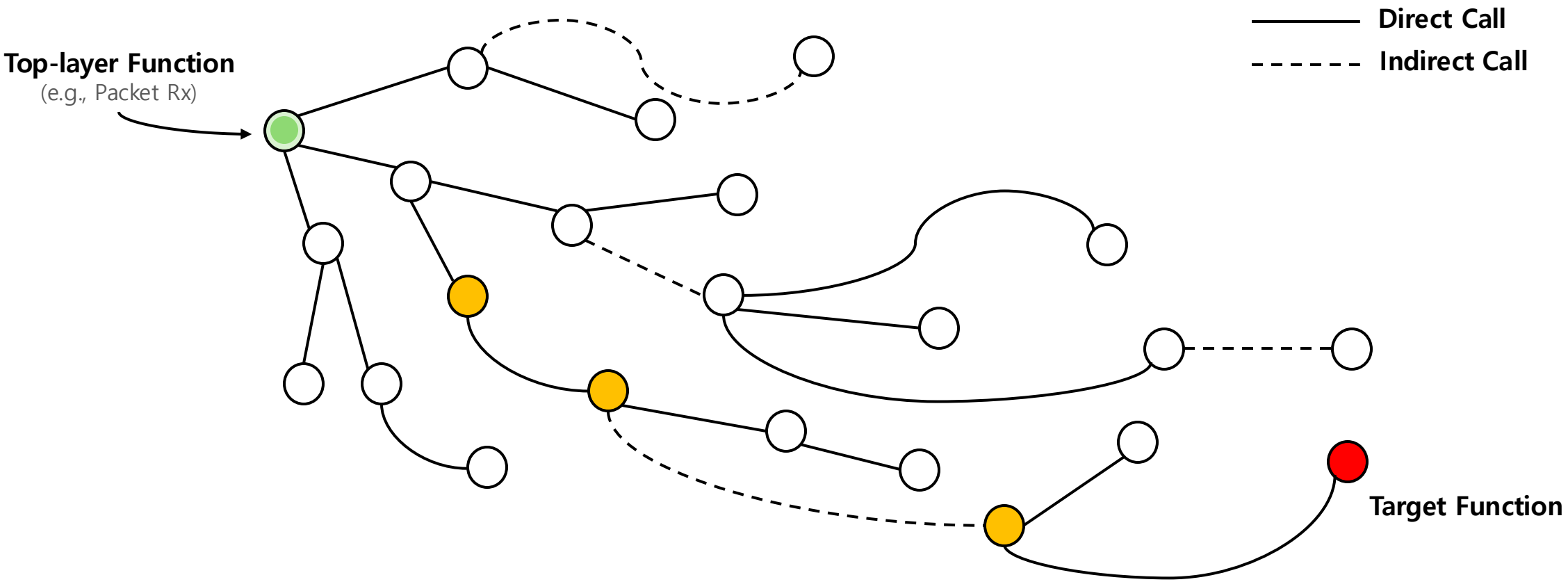
Multi-layer Classification (MLC)

Validate a crash from the top-layer function!



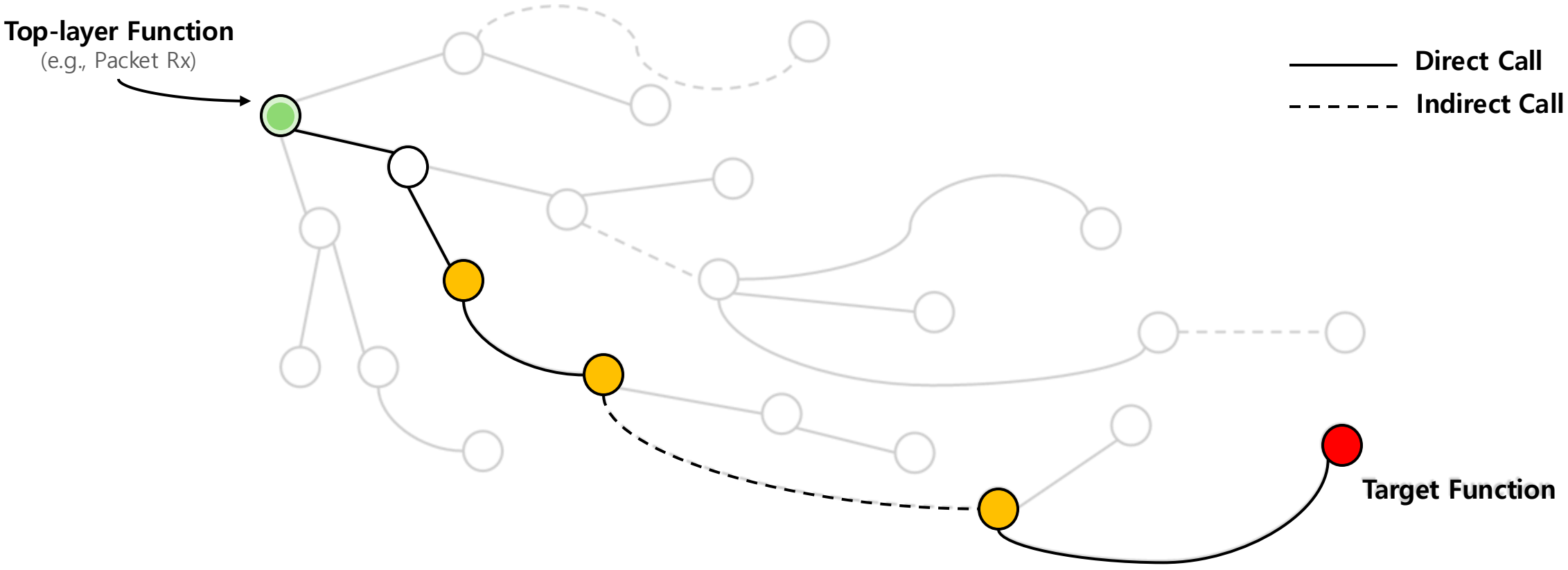
Multi-layer Classification (MLC)

Minimize the call graph and connect indirectly linked functions.



Multi-layer Classification (MLC)

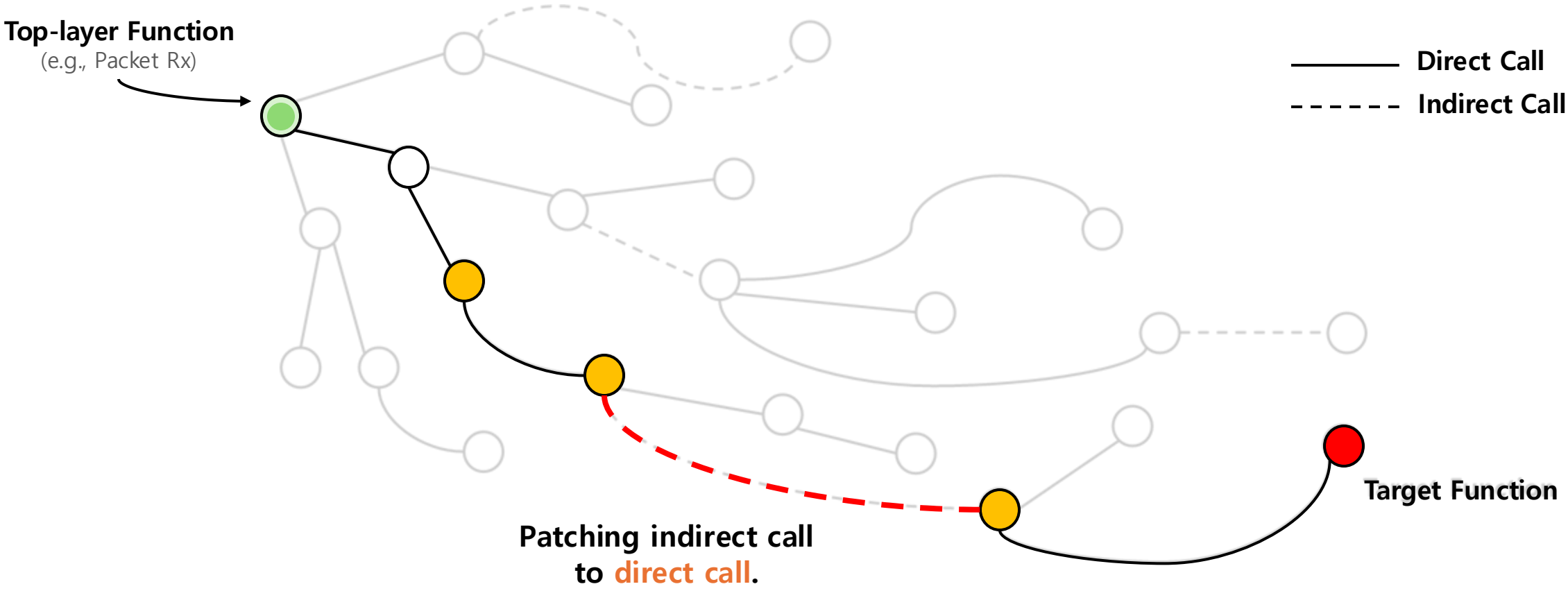
Minimize the call graph and connect indirectly linked functions.



Multi-layer Classification (MLC)

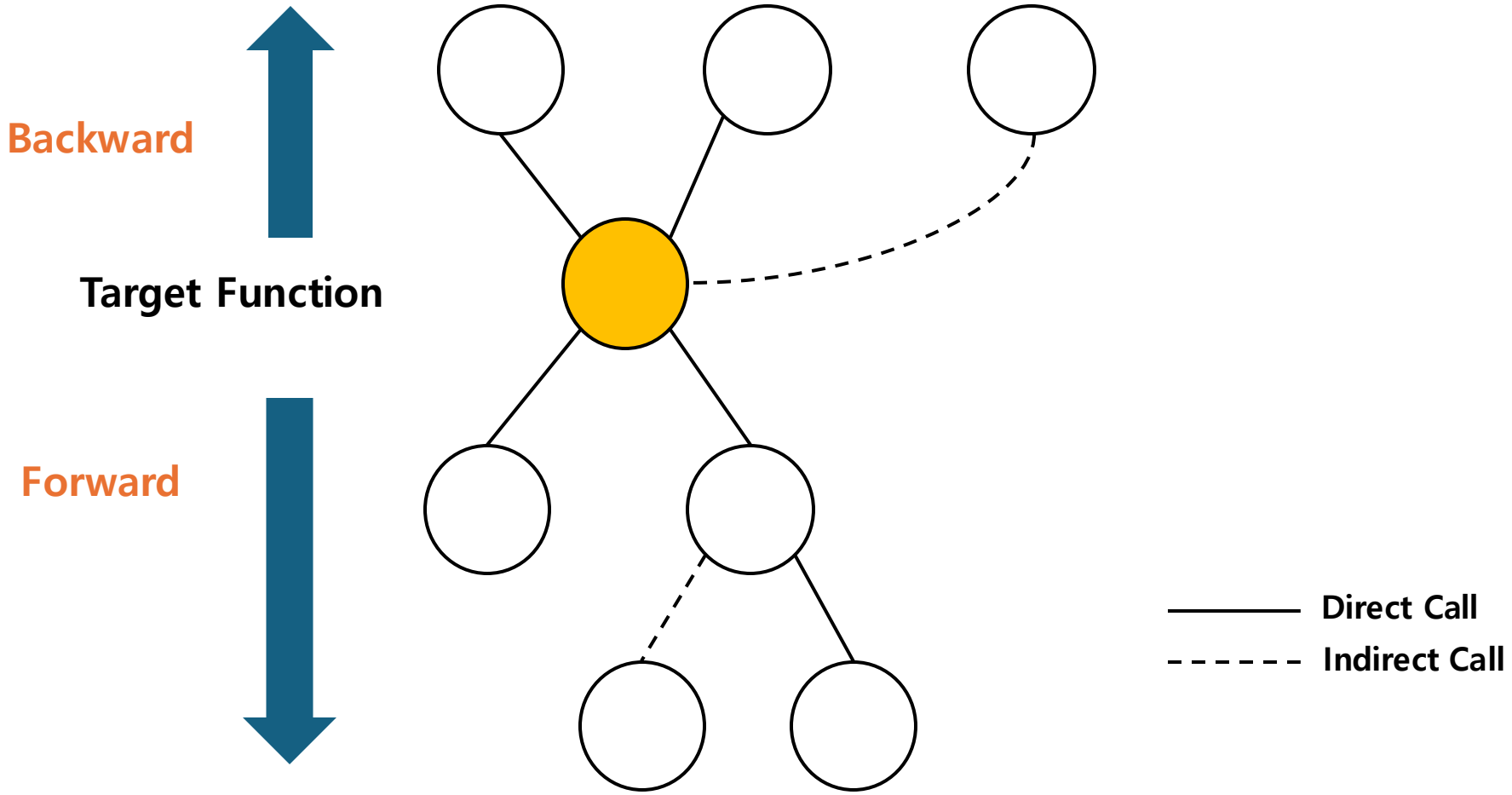
Minimize the call graph and connect indirectly linked functions.

Indirect functions can be linked **without initialization steps**.



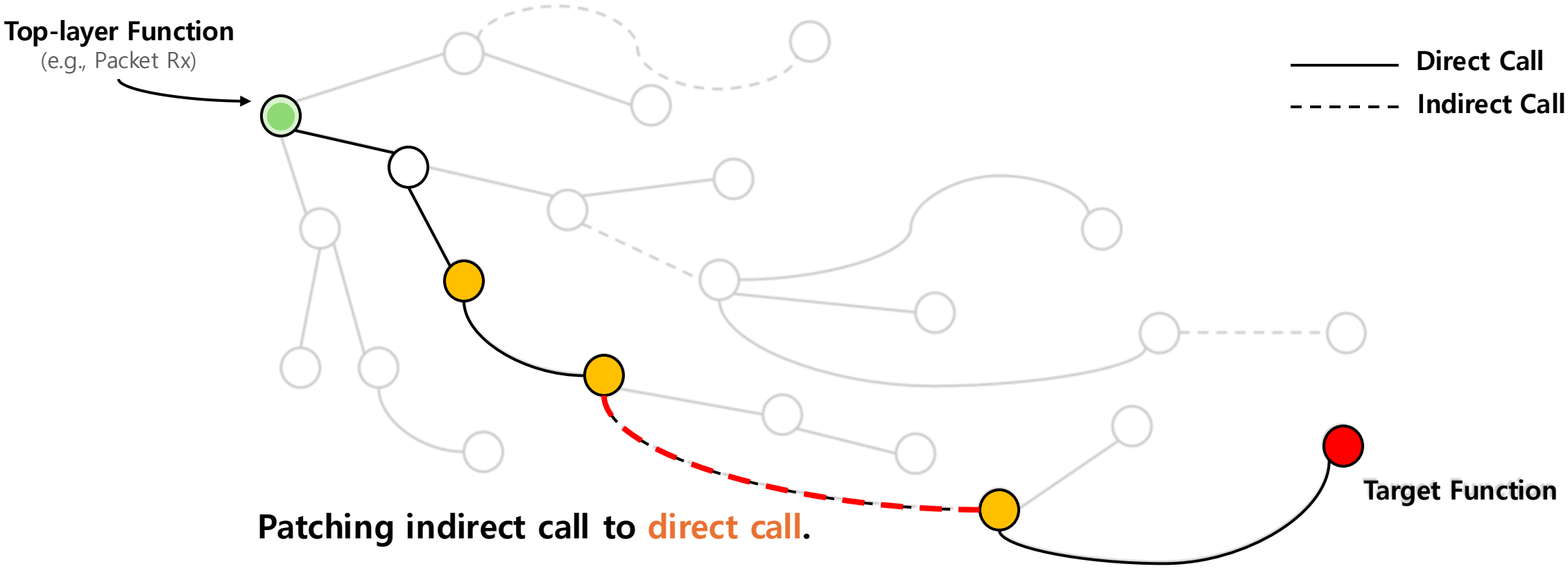
Bi-directional Call Graph Analysis

Call graph spans both backward (parents) and forward (children).



Multi-layer Classification (MLC)

Minimize the call graph and connect indirectly linked functions.



Evaluation

① RQ1

Can RTCON find bugs in real-world RTOSes?

② RQ2

How effective is RTCON compared to existing function-level fuzzers?

③ RQ3

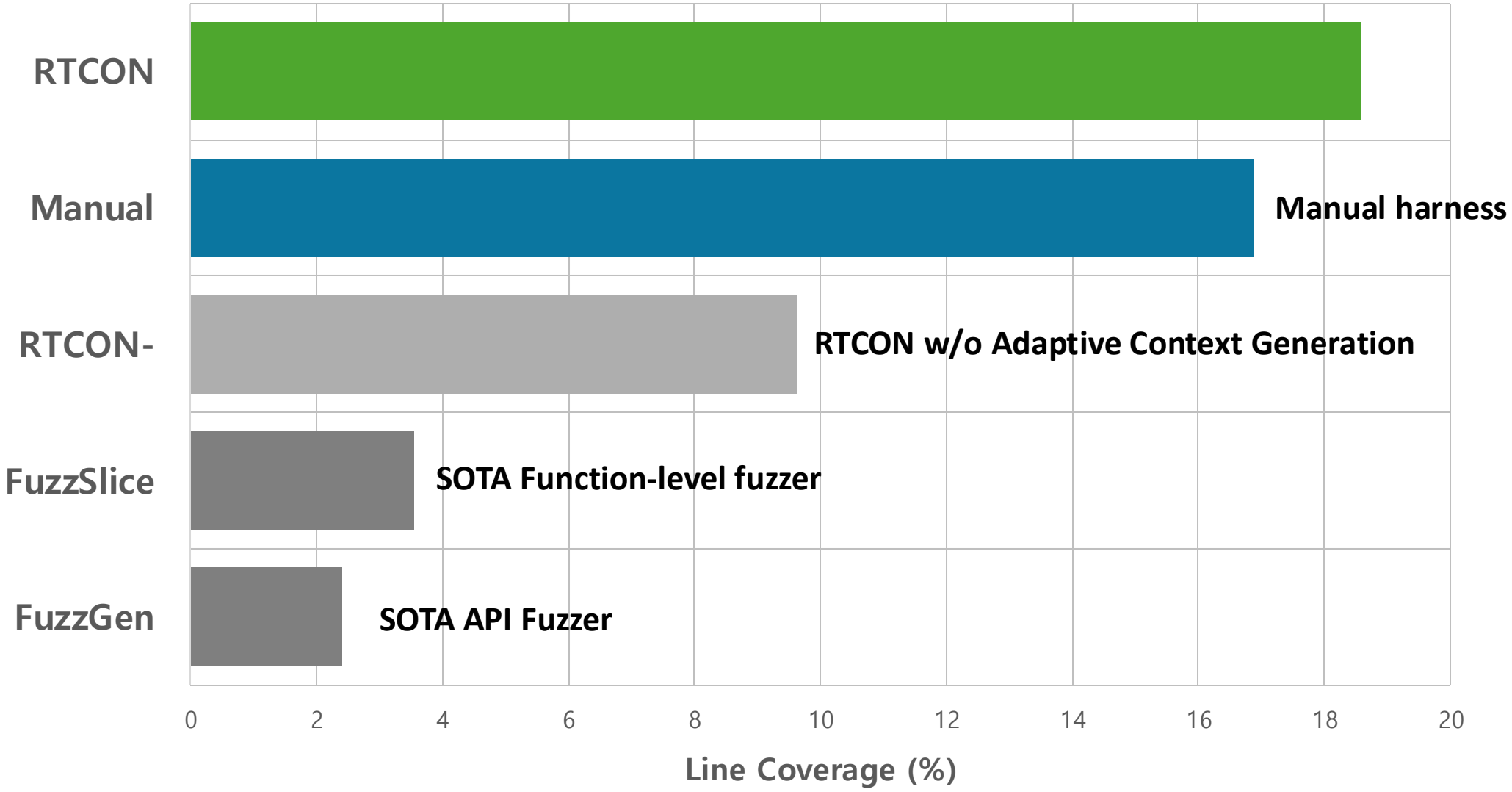
How effective is RTCON in classifying crashes?

Discovered Bugs by RTCON

No	System	Subsystem	Status	CVE	Remote	Context	Detail
1	Zephyr	BT AP	Fixed	CVE-2024-5931	✔		No sanitization for `num_subgroups` field in `parse_recv_state`
2		BT SDP	Fixed	CVE-2024-6135	✔	✔	Mishandling of truncated packets in `sdp_client_receive`
3		BT SDP	Fixed	CVE-2024-6137	✔	✔	Missing check for the maximum number of filters in `get_att_search_list`
4		BT L2CAP	Fixed		✔	✔	Mishandling of truncated packets in `l2cap_br_info_rsp`
5		BT AVDTP	Fixed	CVE-2024-8798	✔	✔	Mishandling of truncated packets in `bt_avdtp_l2cap_recv`
6		BT ASCS	Fixed	CVE-2024-6442	✔		Missing maximum ases bounds check in `ascs_cp_rsp_add`
7		BT RFCOMM	Fixed	CVE-2024-6258	✔	✔	Mishandling of truncated packets in `rfcomm_handle_data`
8		BT OTS	Fixed	CVE-2024-6444	✔		Mishandling of truncated packets in `olcp_ind_handler`
9		BT HCI	Fixed	CVE-2024-6259	✔		Mishandling multiple advertisements in `bt_hci_le_adv_ext_report`
10		BT HCI	Reported			✔	Missing num_bis validation in `hci_le_big_complete`
11		BT Shell	Confirmed				Missing nsig and nvnd check in `bt_mesh_comp_p1_elem_pull`
12		Utils	Fixed	CVE-2024-6443			Mishandling of null starting string in `utf8_trunc`
13		LoRaWAN	Reported		✔		Missing rx_pos bounds check in `frag_transport_package_callback`
14		LoRamac-node	Reported			✔	Missing fragCounter bounds check in `FragDecoderProcess`
15	RIOT	BT HCI	Fixed		✔	✔	Missing ad_len bounds check in `_on_scan_evt`
16		BT HCI	Confirmed		✔		Mishandling of truncated packets in `_filter_uuid`
17		BT HCI	Reported		✔		Mishandling multiple advertisements in `ble_hs_hci_evt_le_ext_adv_rpt`
18		BT HCI	Reported			✔	Missing adv_handle validation in `ble_hs_hci_evt_le_adv_set_terminated`
19		BT HCI	Fixed	CVE-2024-51569			Access header before length check in `ble_hs_hci_evt_num_completed_pkts`
20		LoRaWAN	Fixed		✔	✔	Missing header length check in `gnrc_lorawan_mic_is_valid`
21		DHCP Client	Fixed	CVE-2024-52802	✔		Mishandling of truncated packets in `_parse_advertise`
22		COAP	Duplicated	CVE-2021-41040			Use vulnerable version of 3rd party library
23	FreeRTOS	DNS	Duplicated	CVE-2024-38373	✔	✔	Missing domain length validation in `DNS_ParseDNSReply`
24	ThreadX	SNMPv3	Fixed	CVE-2025-55087	✔	✔	Mishandling of truncated packets in `_nx_snmp_version_3_process`
25							`_fields`
26							
27							

27 bugs discovered across four RTOS targets, including 14 new CVEs.

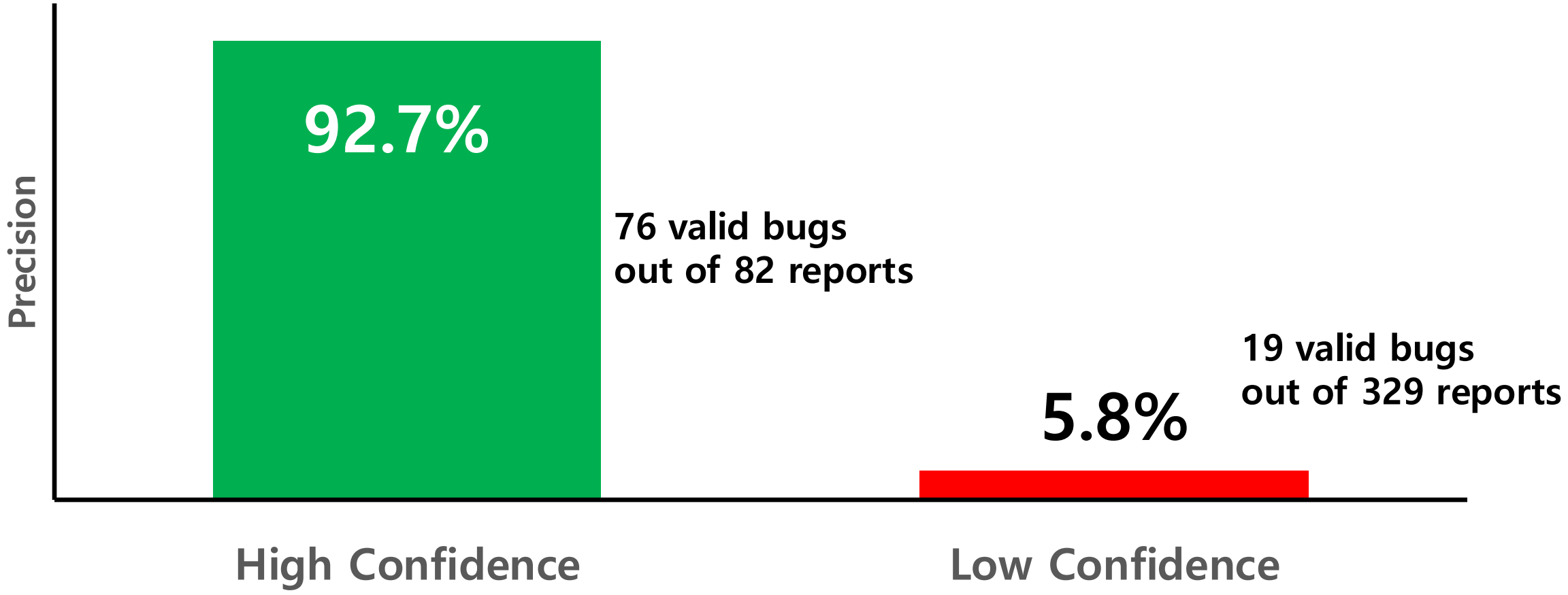
Line Coverage by RTCON



Bug Finding Capability

RTCON	27 / 27 (100%)	✅ Successful Detection
Manual Harness	20 / 27	✅ Successful Detection. — Requiring manual effort.
RTCON- (w/o ACG)	13 / 27	— Failed to find context-related bugs
FuzzSlice & FuzzGen	0 / 27	❌ Harness Generation Failure ❌ Structure Inference Failure

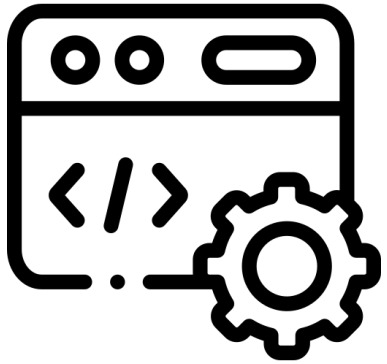
Effectiveness of MLC



Analysts focusing only on high-confidence reports would have
76 true positives while only looking at 6 false positives.

Conclusion

RTCON effectively overcomes the limitations of function-level fuzzing for complex targets like RTOS kernels.



1. Adaptive Context Generation (ACG)

Enables fuzzing of **any target function** without requiring manual setup.

2. Multi-Layer Classification (MLC)

Provides an effective method to focus on **high-confidence vulnerabilities**.

Tutorial

① Apply to RTOS (RIOT)

Walkthrough using a real RTOS target.

② As a General Library

Use RTCON to fuzz any C/C++ project.

Apply to RIOT · Step 1: Clone

STEP 1 / 5

Clone the Zephyr source tree and the RTCON repository side by side.

```
# Clone Zephyr
$ git clone https:///github.com/kaist-hacking/RTCON
```

Apply to RIOT · Step 2: Build

STEP 2 / 5

Build the RTCON-instrumented RIOT binary using the provided wrapper.

```
# Install docker
$ sudo apt update && sudo apt install -y docker.io

# Build the RTCon base image. (This may take some time because it downloads and compiles the LLVM proje
ct. ~ 20 min)
$ docker compose build base

# Build the RTCon RIOT from the base image. (~ 20 min)
$ docker compose build riot
```

Apply to RIOT · Step 3a: Configure Target Function

STEP 3a / 5

Specify the **target function** that RTCON should fuzz inside RIOT.
We already provide an example function list in **`eval/config/riot-func-list-reduced.txt`**.

```
# eval/config/riot-func-list-reduced.txt

Reduced Config
-------------------------------------------------
/source/projects/RIOT/riot_latest/pkg/nimble/rpble/nimble_rpble.c,_on_scan_evt,3,4,0
/source/projects/RIOT/riot_latest/pkg/nimble/autoconn/nimble_autoconn.c,_on_scan_evt,3,4,0
/source/projects/RIOT/riot_latest/pkg/nimble/netif/nimble_netif.c,_netdev_set,2,3,0
/source/projects/RIOT/riot_latest/pkg/nimble/netif/nimble_netif.c,nimble_netif_accept,0,1,0
/source/projects/RIOT/riot_latest/pkg/nimble/scanlist/nimble_scanlist.c,nimble_scanlist_update,3,4,0
/source/projects/RIOT/riot_latest/sys/net/gnrc/link_layer/lorawan/gnrc_lorawan_mlme.c,gnrc_lorawan_mlme_process_join,1,2,0
```

Apply to RIOT · Step 3b: Configure Source File

STEP 3b / 5

Tell RTCON which **source file** contains the target function.
It is important to specify the **source file** because **static functions may have identical names**.

```
# eval/config/riot-func-list-reduced.txt

Reduced Config
-------------------------------------------------
/source/projects/RIOT/riot_latest/pkg/nimble/rpble/nimble_rpble.c,_on_scan_evt,3,4,0
/source/projects/RIOT/riot_latest/pkg/nimble/autoconn/nimble_autoconn.c,_on_scan_evt,3,4,0
/source/projects/RIOT/riot_latest/pkg/nimble/netif/nimble_netif.c,_netdev_set,2,3,0
/source/projects/RIOT/riot_latest/pkg/nimble/netif/nimble_netif.c,nimble_netif_accept,0,1,0
/source/projects/RIOT/riot_latest/pkg/nimble/scanlist/nimble_scanlist.c,nimble_scanlist_update,3,4,0
/source/projects/RIOT/riot_latest/sys/net/gnrc/link_layer/lorawan/gnrc_lorawan_mlme.c,gnrc_lorawan_mlme_process_join,1,2,
0
```

Apply to RIOT · Step 3c: Mark User Input

STEP 3c / 5

Mark which parameter is the **user input** by index — RTCON taints it as the fuzz source.
User buffer input, user buffer length, and user input type (to be discussed later, -1 means none)

```
# eval/config/riot-func-list-reduced.txt

Reduced Config
------------------------------------------------
/source/projects/RIOT/riot_latest/pkg/nimble/rpble/nimble_rpble.c,_on_scan_evt,3,4,0
/source/projects/RIOT/riot_latest/pkg/nimble/autoconn/nimble_autoconn.c,_on_scan_evt,3,4,0
/source/projects/RIOT/riot_latest/pkg/nimble/netif/nimble_netif.c,_netdev_set,2,3,0
/source/projects/RIOT/riot_latest/pkg/nimble/netif/nimble_netif.c,nimble_netif_accept,0,1,0
/source/projects/RIOT/riot_latest/pkg/nimble/scanlist/nimble_scanlist.c,nimble_scanlist_update,3,4,0
/source/projects/RIOT/riot_latest/sys/net/gnrc/link_layer/lorawan/gnrc_lorawan_mlme.c,gnrc_lorawan_mlme_process_join,1,2,0
```

Apply to RIOT · Step 3d: Mark User Input

STEP 3d / 5

Once you update the target list file, you need to rebuild the Docker image.

```
# Build the RTCon RIOT from the base image. (~ 30 sec)
$ docker compose build riot
```

Apply to RIOT · Step 4: Run

STEP 4 / 5

Run RTCON to test the **target function in the target list** by specifying the **file name, function name, and test type**.
In this tutorial, since we do not use MLC, the last argument should be set to `false`.

```
# It will build the fuzzing harness for ble_hs_hci_evt_le_ext_adv_rpt.
$ docker compose run riot /source/projects/RIOT/riot_latest/build/pkg/nimble/nimble/host/src/ble_hs_hci
_evt.c ble_hs_hci_evt_le_ext_adv_rpt 0 false
```

Apply to RIOT · Step 4: Run

STEP 4 / 5

RTCON will start by analyzing the call graph from the target function.

```
[INFO] Replacing test function with 1
[INFO] Analyzing CFG for ble_hs_hci_evt_le_ext_adv_rpt
[+] Read tcg.dot
[+] Read tcg.dot Done. - 0:00:18.834144
[+] Read whole_project.ll
[+] Read whole_project.ll Done. - 0:00:19.306947
[+] Read /configs/riot-func-list-reduced.txt
[+] Read /configs/riot-func-list-reduced.txt Done. - 0:00:19.869816
[+] Valid indirect call: ble_hs_hci_evt_le_meta -> ble_hs_hci_evt_le_ext_adv_rpt
[+]     Parent params: [None, None, None]
[+]     Child params: [None, None, None]
[*] Add parent ble_hs_hci_evt_le_meta to analysis target
[+] Valid indirect call: ble_hs_hci_evt_process -> ble_hs_hci_evt_le_meta
[+]     Parent params: [None, None, None]
[+]     Child params: [None, None, None]
=======================================
```

Apply to RIOT · Step 4: Run

STEP 4 / 5

The target harness will be located in the **host_bin** directory.
In the RIOT case, it will be located at **host_bin/riot-[func_name]-fuzz** .

```
func: /source/projects/RIOT/riot_latest/build_ble_hs_hci_evt_le_ext_adv_rpt/pkg/nimble/nimble/host/src/b
le_hs_adv.c,replicate_ble_hs_adv_find_field_1
func: /source/projects/RIOT/riot_latest/build_ble_hs_hci_evt_le_ext_adv_rpt/pkg/nimble/nimble/host/src/b
le_hs_adv.c,replicate_ble_hs_adv_find_field_0
[+] Target ble_hs_hci_evt_le_ext_adv_rpt built successfully. Total elapsed time: 78.77 seconds. Total cr
eated functions: 238
[+] Finished building target ble_hs_hci_evt_le_ext_adv_rpt. Total elapsed time: 78.79 seconds
root@5caa7cbba299:/source/projects/RIOT# mkdir corpus
root@5caa7cbba299:/source/projects/RIOT# ./host_bin/riot-ble_hs_hci_evt_le_ext_adv_rpt-fuzz corpus/
```

Apply to RIOT · Step 5: Run

STEP 5 / 5

RTCON starts fuzzing on the target function.

```
root@5caa7cbba299:/source/projects/RIOT# ./host_bin/riot-ble_hs_hci_evt_le_ext_adv_rpt-fuzz corpus/
RIOT native interrupts/signals initialized.

INFO:        2 files found in corpus/
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 2 min: 133b max: 134b total: 267b rss: 35Mb
==2991==WARNING: ASan is ignoring requested __asan_handle_no_return: stack type: default top: 0x7fffd228
7000; bottom 0x0000011d1000; size: 0x7fffd10b6000 (140736700571648)
False positive error reports may follow
For details see https://github.com/google/sanitizers/issues/189
#4    INITED cov: 42 ft: 46 corp: 2/267b exec/s: 0 rss: 36Mb
#10    NEW    cov: 42 ft: 47 corp: 3/335b lim: 134 exec/s: 0 rss: 36Mb L: 68/134 MS: 1 EraseBytes-
#121    NEW    cov: 43 ft: 48 corp: 4/464b lim: 134 exec/s: 0 rss: 37Mb L: 129/134 MS: 1 EraseBytes-
#167    REDUCE cov: 43 ft: 48 corp: 4/459b lim: 134 exec/s: 0 rss: 37Mb L: 63/134 MS: 1 EraseBytes-
```

Apply to RIOT · Example of Crash

```
0x5020000539b9 is located 121 bytes after 16-byte region [0x502000053930,0x502000053940)
allocated by thread T0 here:
    #0 0x51a42f in malloc (/source/projects/RIOT/host_bin/riot-ble_hs_hci_evt_le_ext_adv_rpt-fuzz+0x5
1a42f) (BuildId: 95aa149648452069fcce0078e7e7f81616937d14)

SUMMARY: AddressSanitizer: heap-buffer-overflow /source/projects/RIOT/riot_latest/build_ble_hs_hci_ev
t_le_ext_adv_rpt/pkg/nimble/nimble/host/src/ble_hs_hci_evt.c:595:55 in ble_hs_hci_evt_le_ext_adv_rpt
Shadow bytes around the buggy address:
  0x502000053700: fa fa 00 00 fa fa 00 00 fa fa 00 00 fa fa 00 00
  0x502000053780: fa fa 00 00 fa fa 00 00 fa fa 00 00 fa fa 00 00
  0x502000053800: fa fa 00 00 fa fa 00 00 fa fa 00 00 fa fa 00 00
  0x502000053880: fa fa 00 00 fa fa 00 00 fa fa fd fa fa fa fd fd
  0x502000053900: fa fa 07 fa fa fa 00 00 fa fa fa fa fa fa fa fa
=>0x502000053980: fa fa fa fa fa fa fa[fa]fa fa fa fa fa fa fa fa
  0x502000053a00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x502000053a80: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x502000053b00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x502000053b80: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x502000053c00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
```

Try Yourself!

1. Change the configuration
2. Target the function / user input index
3. Build the harness
4. Run the harness

General Library

RTCON can also be used to fuzz general libraries.

⚠ Experimental feature

- It may not work correctly.

General Library · Step 1: Clone

STEP 1 / 6

Clone RTCON and checkout to general branch.

```
# Clone your project
$ git clone https://github.com/kaist-hacking/RTCON.git

# Checkout to general branch
$ cd RTCON && git fetch origin && git checkout -b general origin/general
```

General Library · Step 2: Clone Project & Configure

STEP 2 / 6

Clone your C/C++ project to the **project** directory inside the RTCON project.

```
# Clone your project in the project directory
$ cd project && git clone https://github.com/ittiam-systems/libhevc.git

# Configure the project specific
[config.yml]
  libhevc:
    subdir:        libhevc  # project name in the project directory
    build_command: "rm -rf build && mkdir build && cd build && cmake .. && make" # Build command for th
e project
    build_dir:     build # Build directory for the project
    binary_name:   hevcenc # Final binary name
```

General Library · Step 3: Build & Run RTCON

STEP 3 / 6

Build the RTCON (It will take some time) and Run RTCON

```
# Build with RTCON wrapper
$ docker compose build

# Run RTCON
$ docker compose run –rm rtcon
```

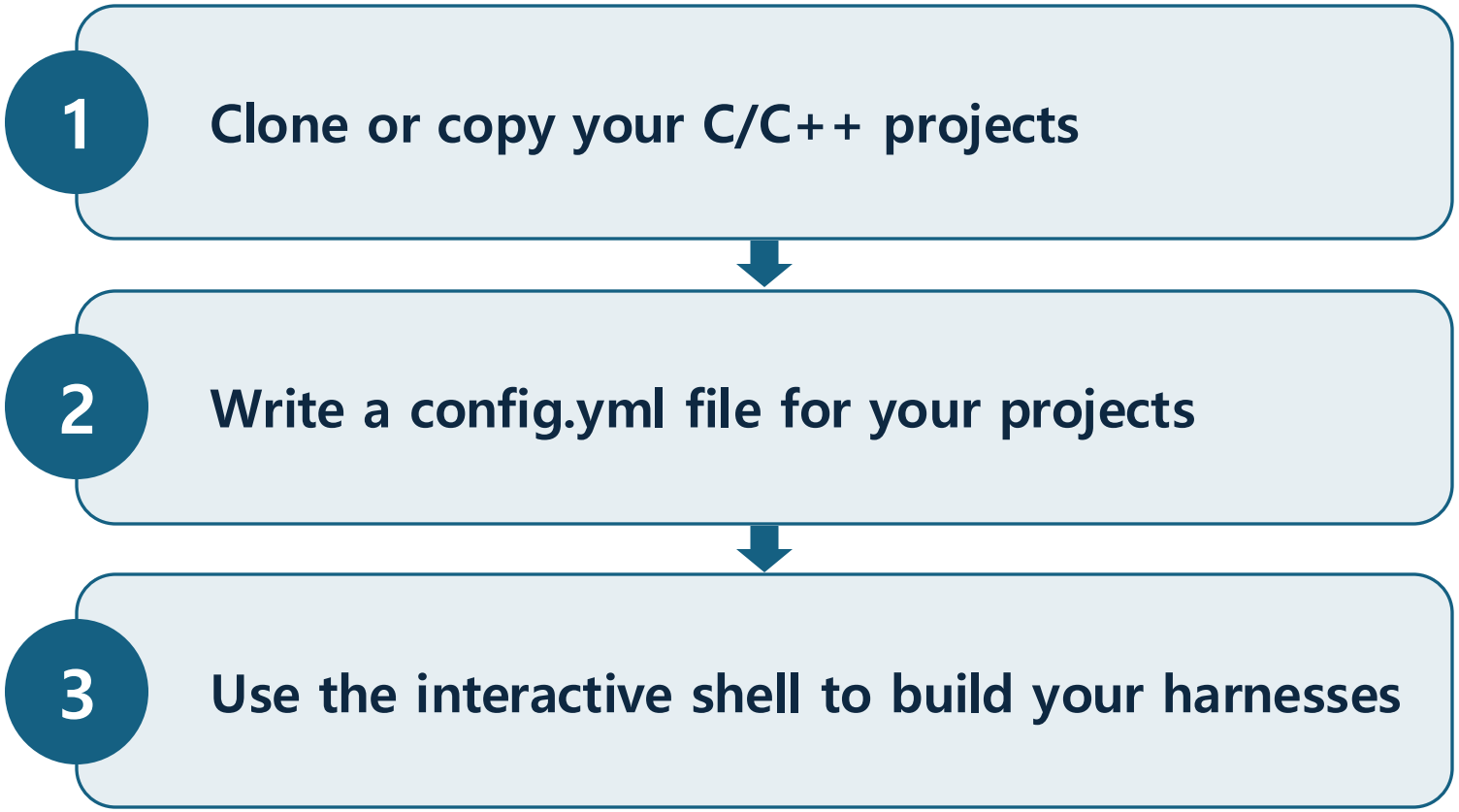
General Library · Step 4: Interactive Shell

STEP 4 / 6

Choose the project and the function you want to fuzz

You can search for keywords to select targ et functions.

Try Yourself!



Thank you

Contact:

Eunkyu Lee (ekleezg@kaist.ac.kr)