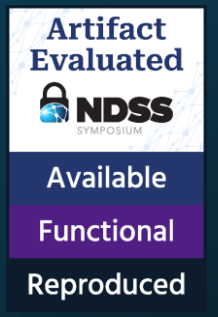


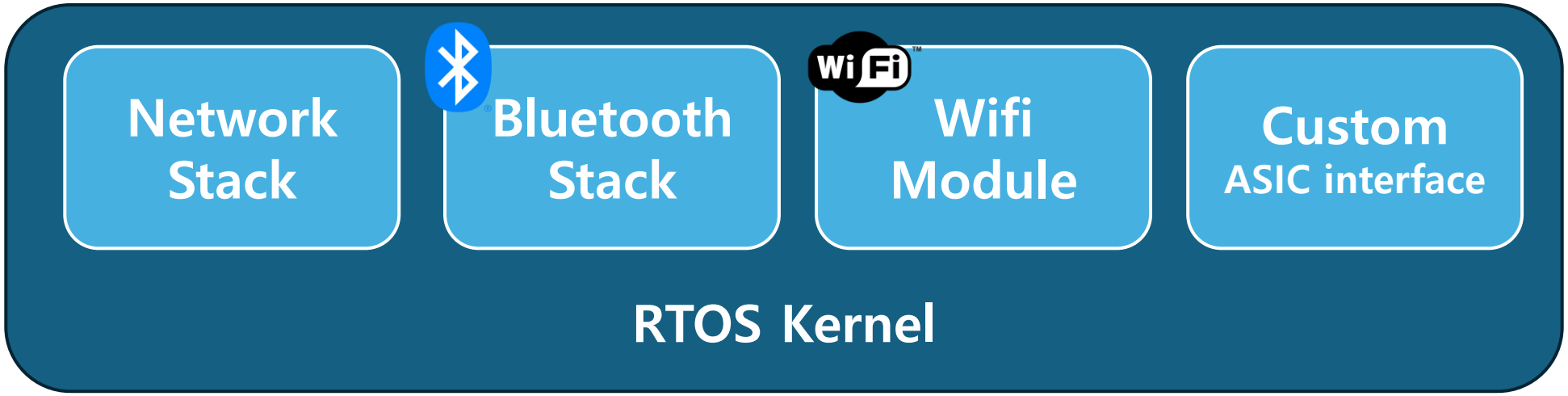
RTCON: Context-Adaptive Function-Level Fuzzing for RTOS Kernels

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

KAIST

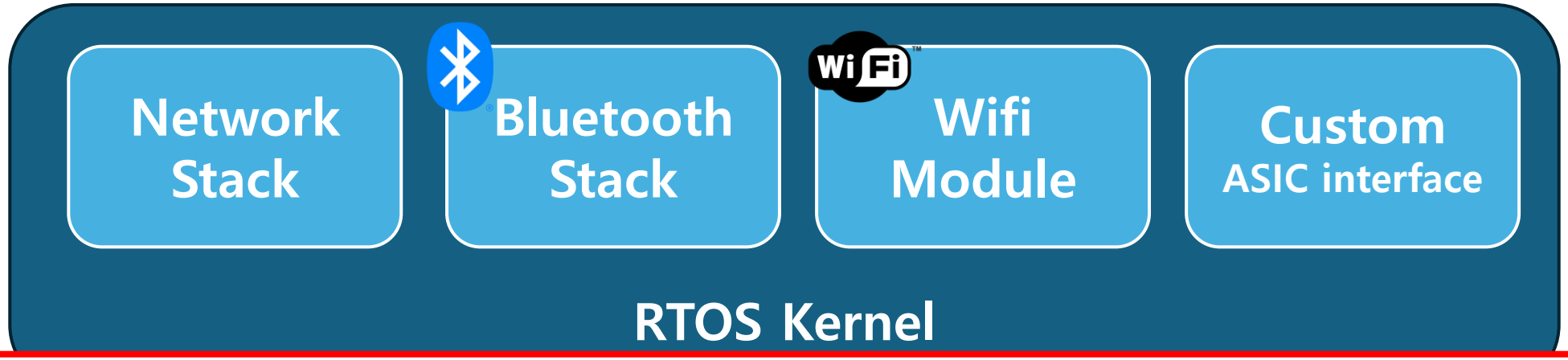
Security Challenges in RTOS kernels

- Provides various subsystems (Bluetooth, Wi-Fi, network stacks).
- Diverse functionalities expand external attack surface.
- Even simple vulnerabilities may lead to RCE or DoS attacks.



Security Challenges in RTOS kernels

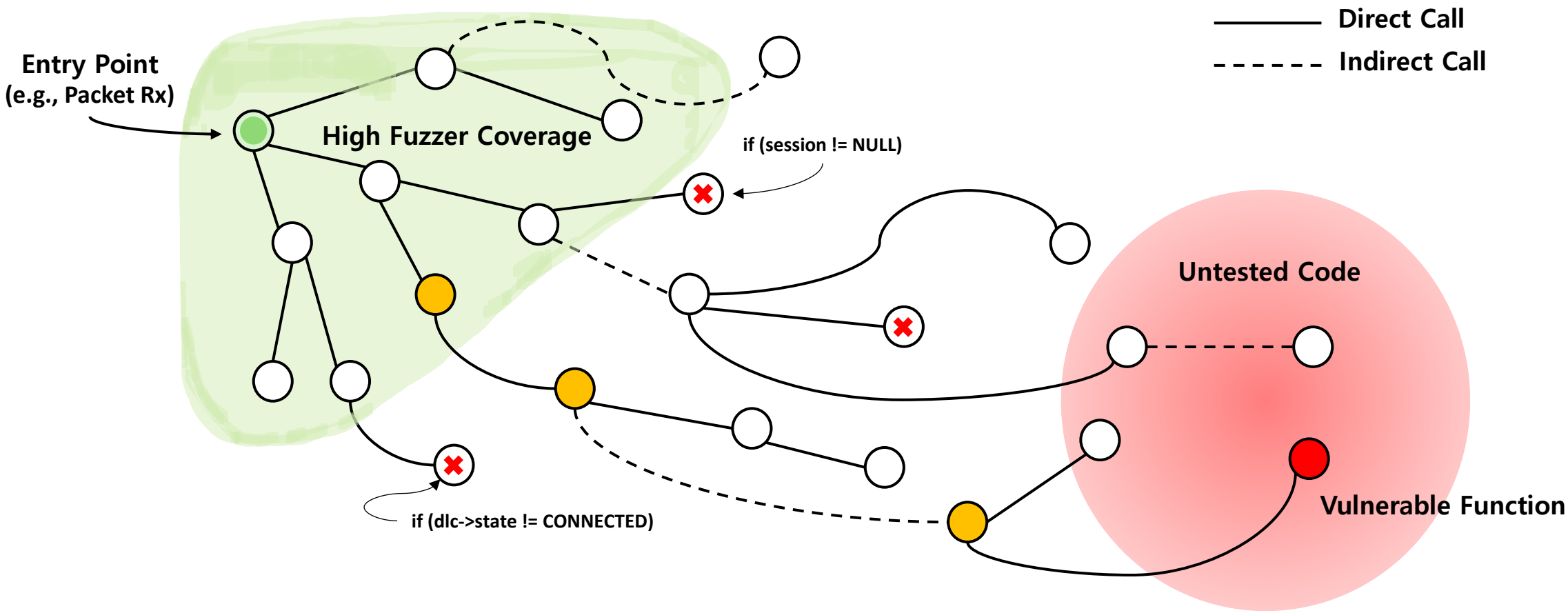
- Provides various subsystems (Bluetooth, Wi-Fi, network stacks).
- Diverse functionalities expand external attack surface.
- Even simple vulnerabilities may lead to RCE or DoS attacks.



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

Limitations of Existing Fuzzing Tools

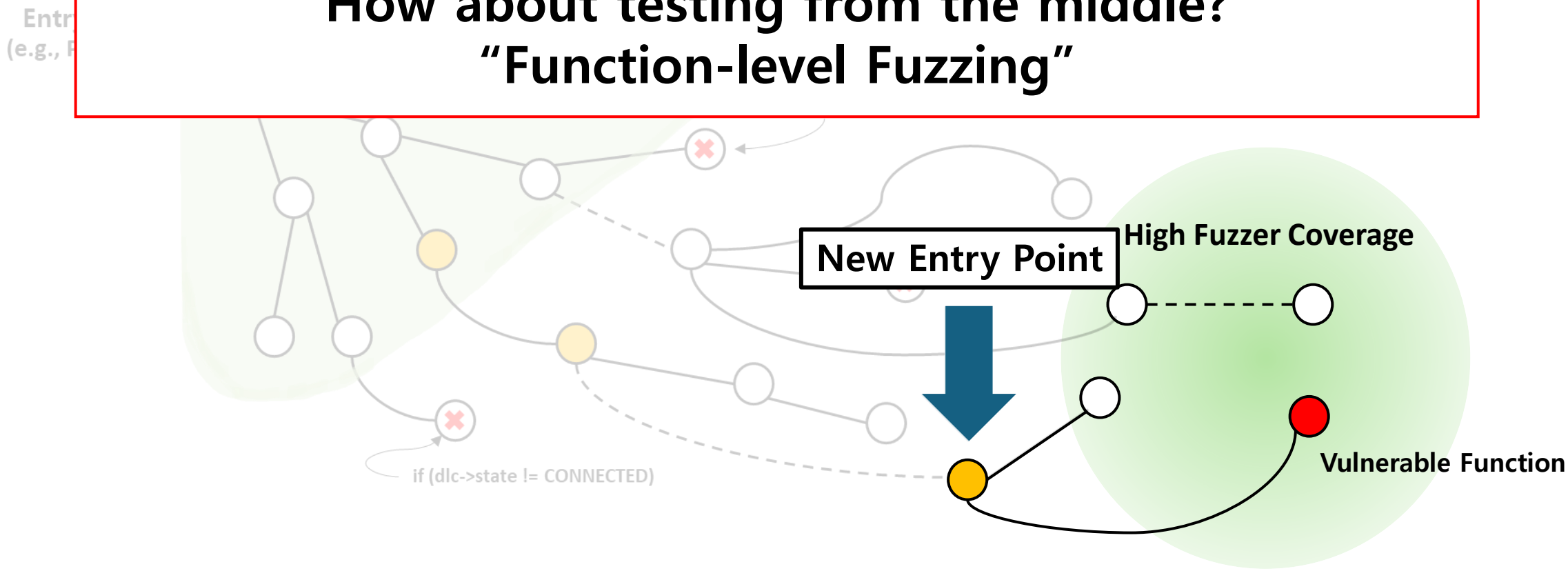
- Many vulnerabilities reside in deeply nested functions.



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);
```

buf → **OK** ✅

session → ❓

msg_type → ❓

tid → ❓

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 {
                                                  sep, int_seid, buf, &error_code);
26
```

❌ Condition not met

❌ Crash

❌ Crash

buf → OK ✅

session → ❓

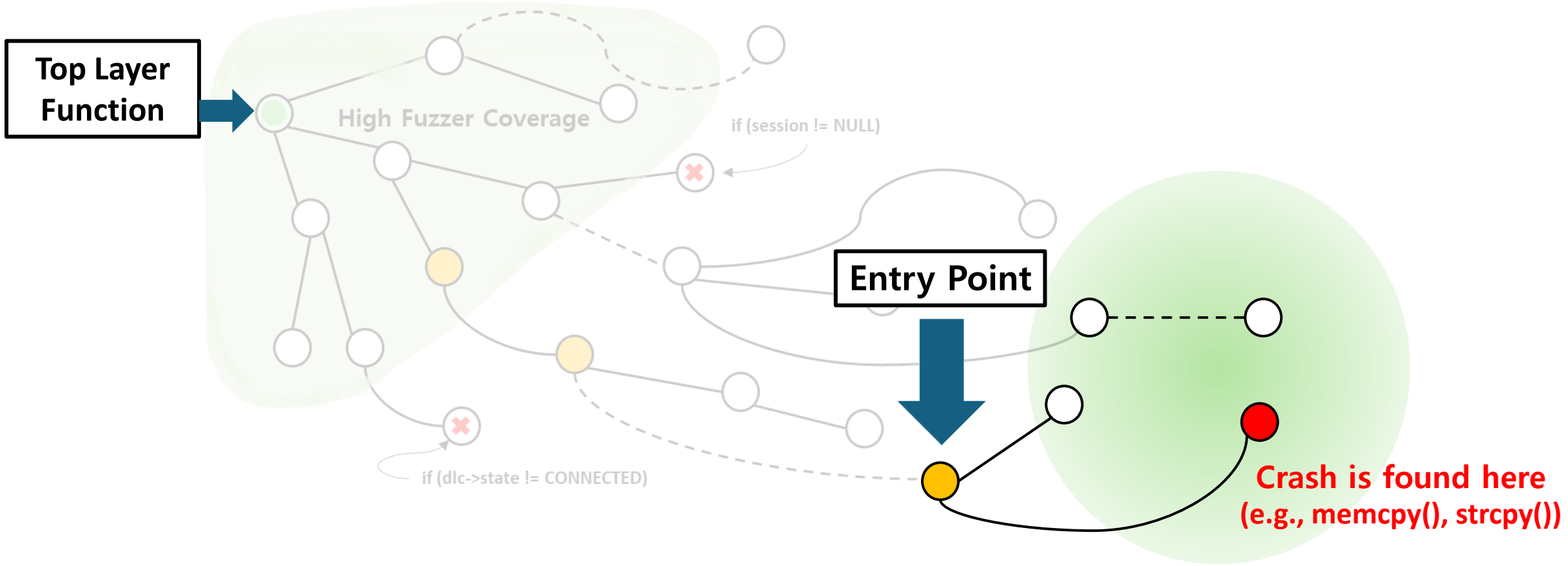
msg_type → ❓

tid → ❓

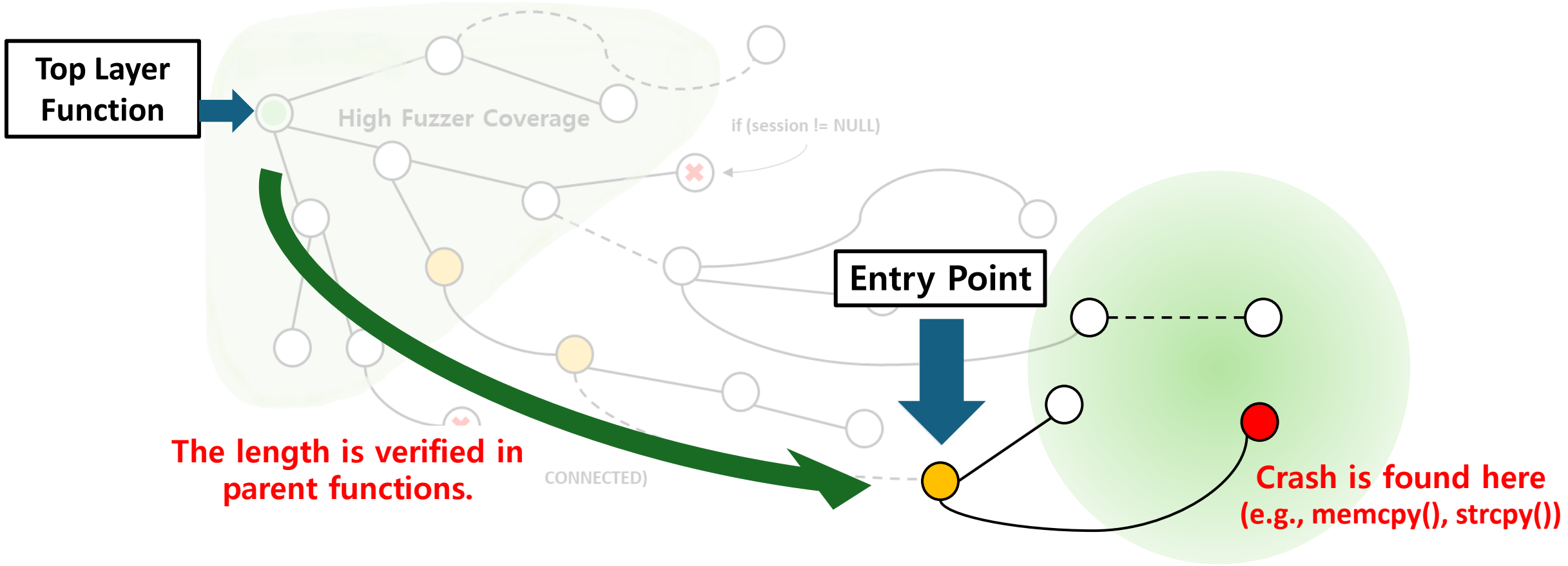
context variables!

Early termination before reaching the vulnerable code → False positives.

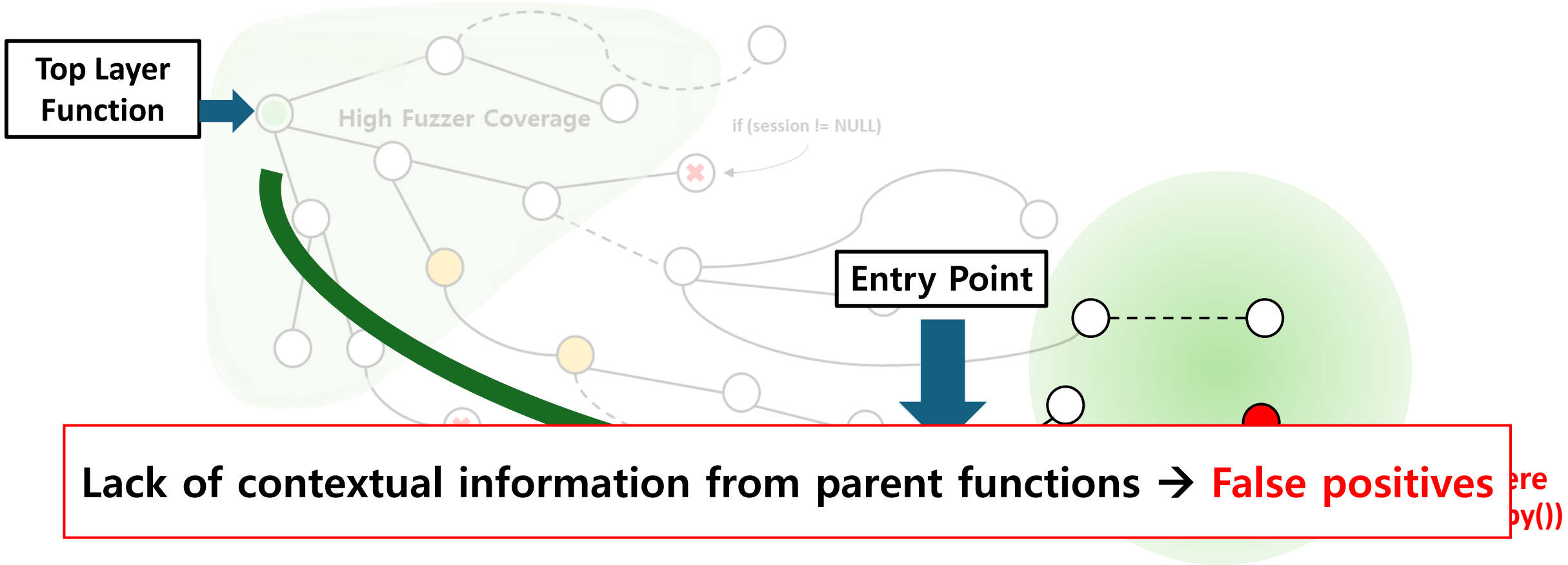
False Positives of Function-level Fuzzing



False Positives of Function-level Fuzzing



False Positives of Function-level Fuzzing

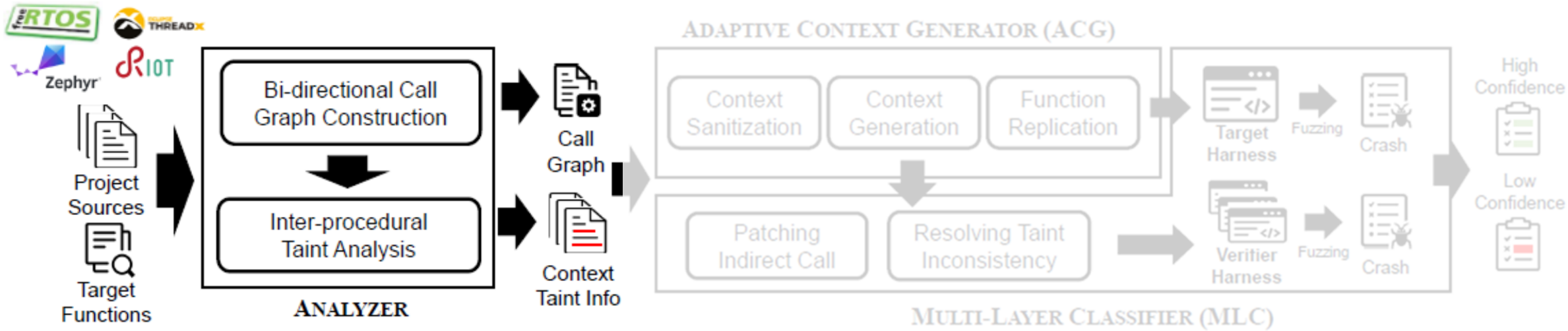


Our Approaches

C1. Reachability & Scalability Traditional fuzzers **cannot reach deep** kernel functions.	**A1. Function-level Fuzzing** Directly fuzz **any target function.**
C2. Missing Function Context Directly calling a function **lacks the necessary contexts**.	**A2. Adaptive Context Generation** **Generating** required context values **on-demand.**
C3. High False Positives Function-level fuzzing creates many **invalid crashes**.	**A3. Multi-layer Classification** **Verifies** crashes by testing **higher-level functions**.

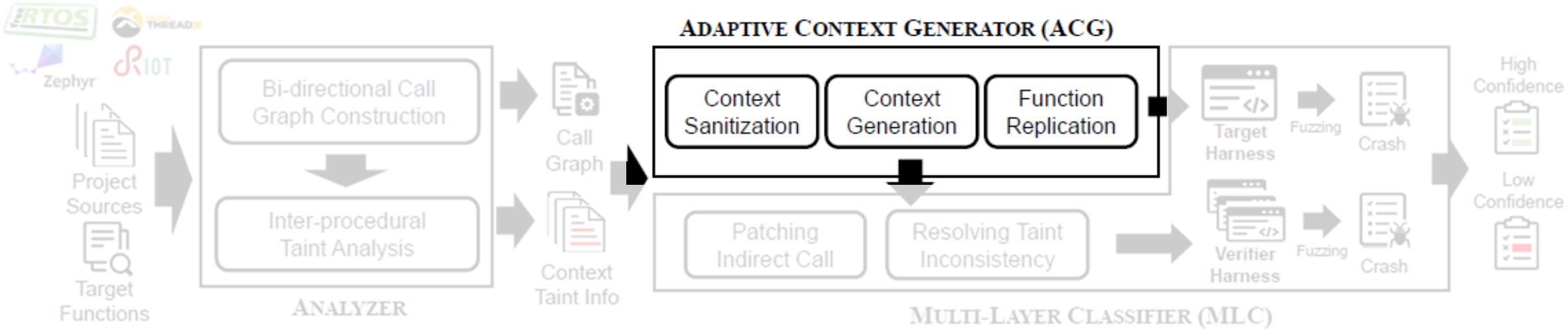
RTCON: Context-Adaptive Function-Level Fuzzing for RTOS Kernels

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



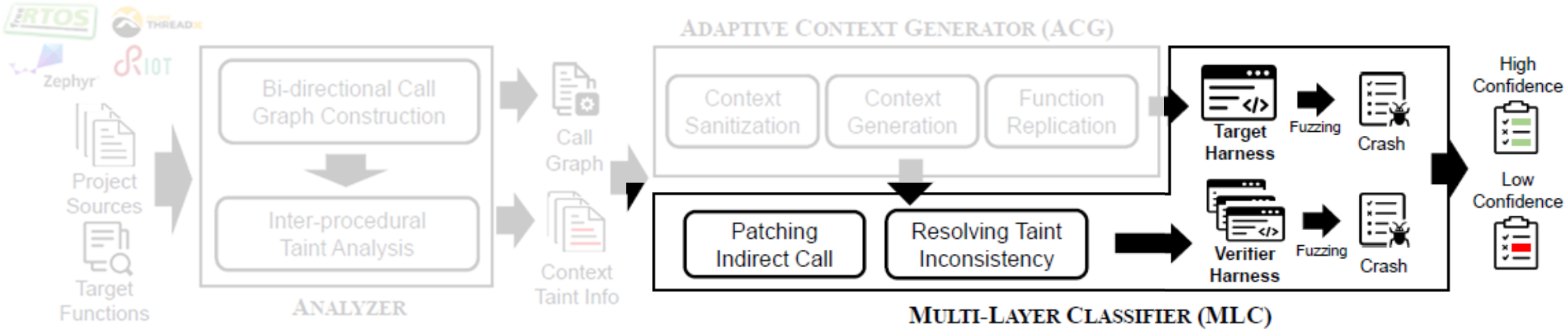
RTCON: Context-Adaptive Function-Level Fuzzing for RTOS Kernels

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



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

- Adaptive Context Generation
- Inter-procedural Taint Analysis
- 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);
```

session is a context variable.

❌ **Crash**

❌ **Crash**

Adaptive Context Generation: Our Approach

- Instruments **hook functions to catch potential faults and sanitizes 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** in order 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

```
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
19
20
21
```

In LLVM IR

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

```
%msgtype        = .... (parameter)
```

On-Demand Allocation ✅

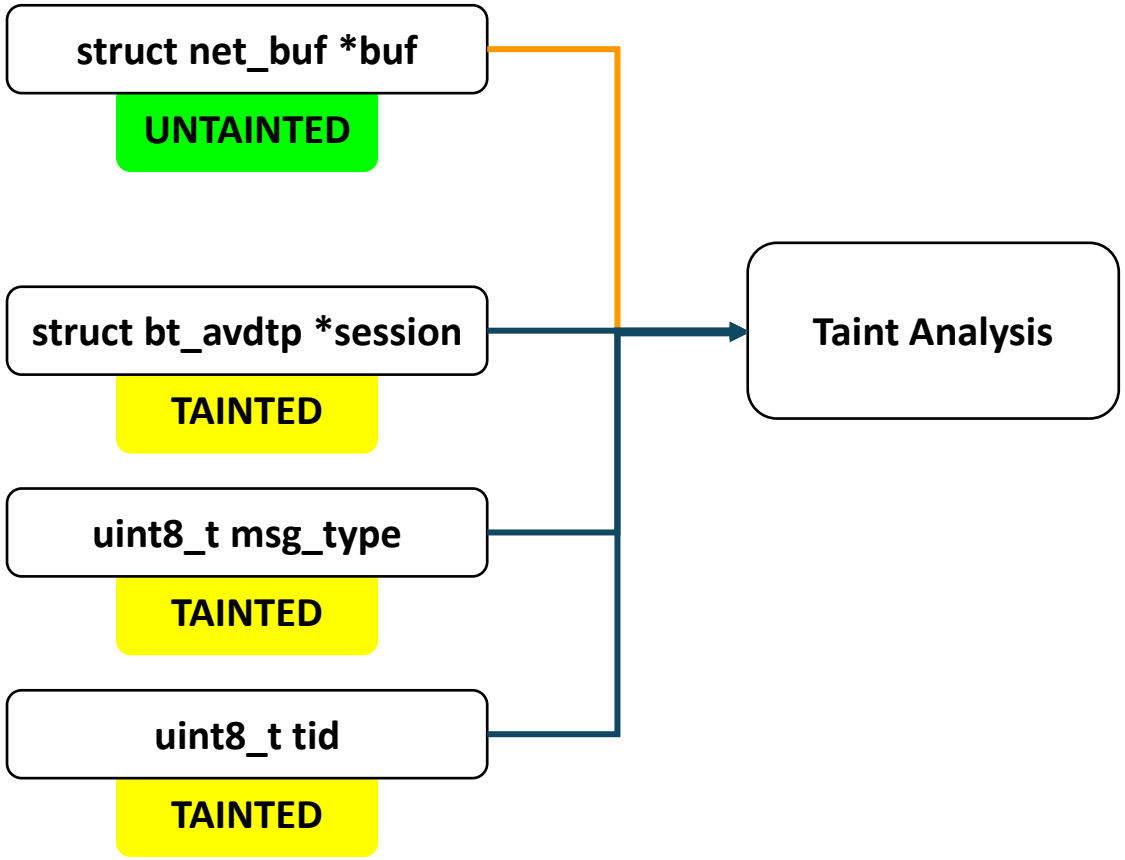
Generating Context Values Adaptively ✅

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

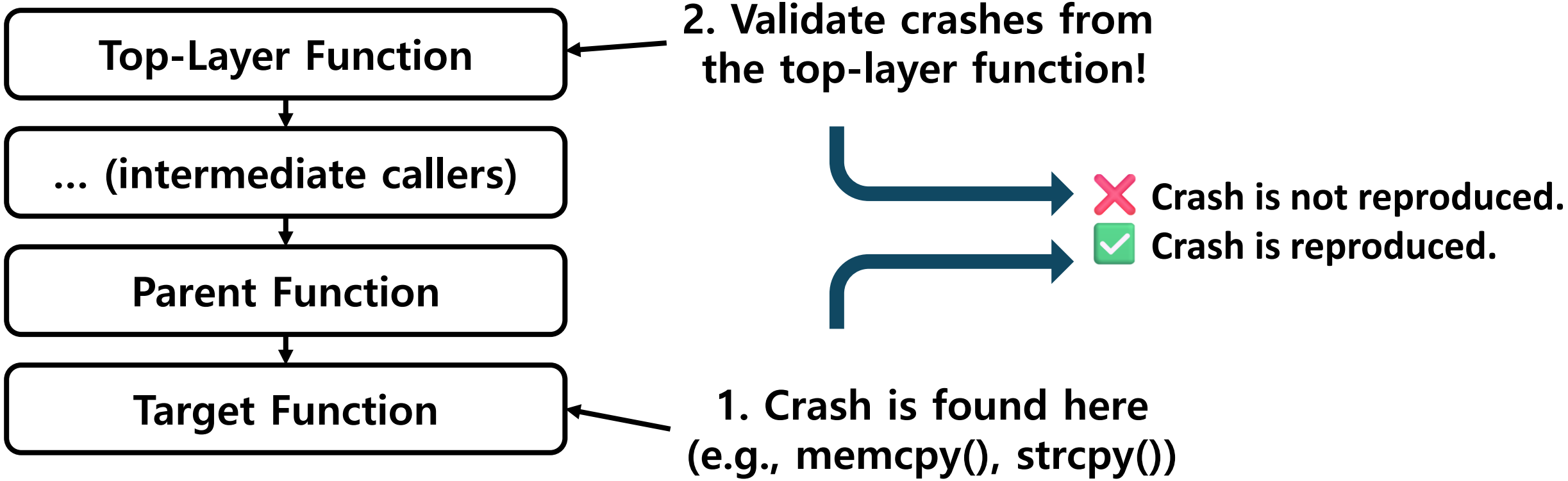
Untainted

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    ...
```

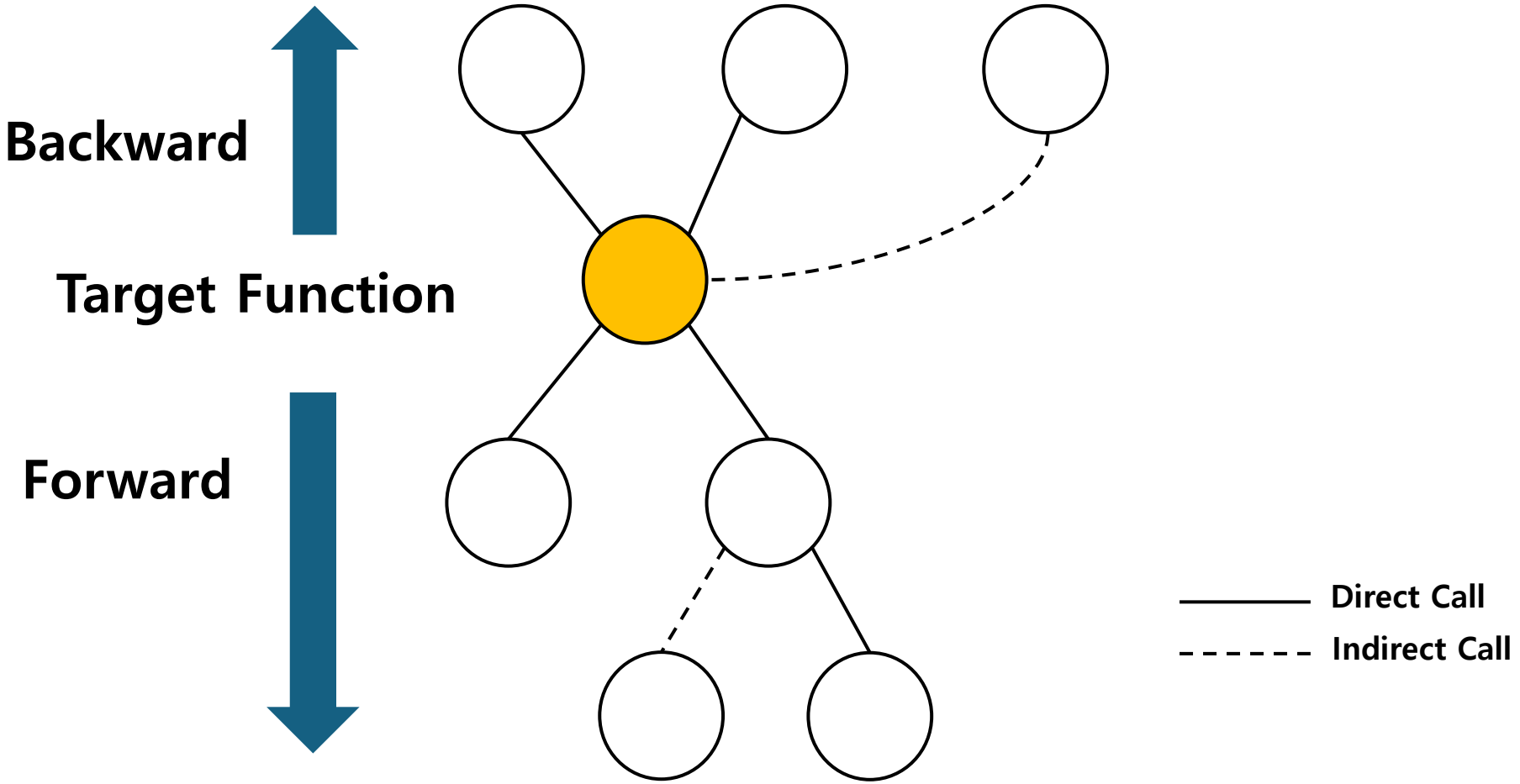
Multi-layer Classification (MLC)

Validate a crash from the top-layer function!



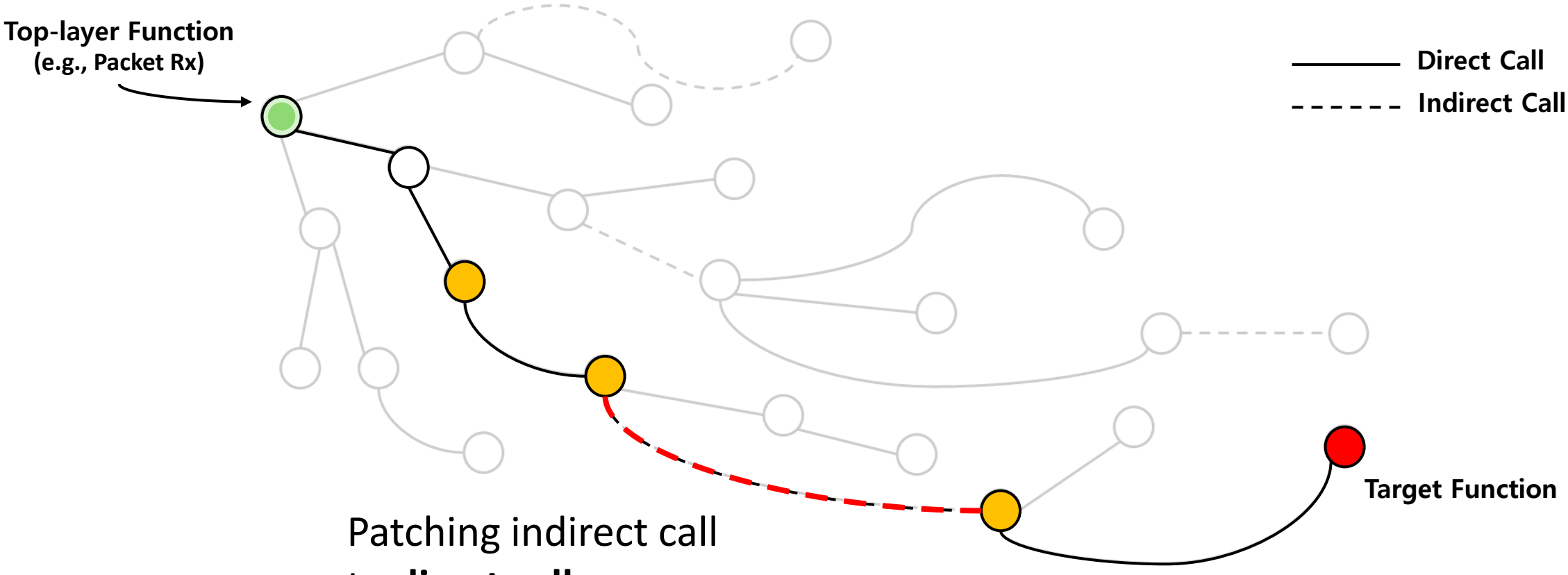
Bi-directional Call Graph Analysis

Call graph both **backward** (toward parent functions) and **forward** (toward child functions).



Multi-layer Classification (MLC)

Minimize the call graph and connect indirectly linked functions.



Patching indirect call
to **direct call**.

Evaluation

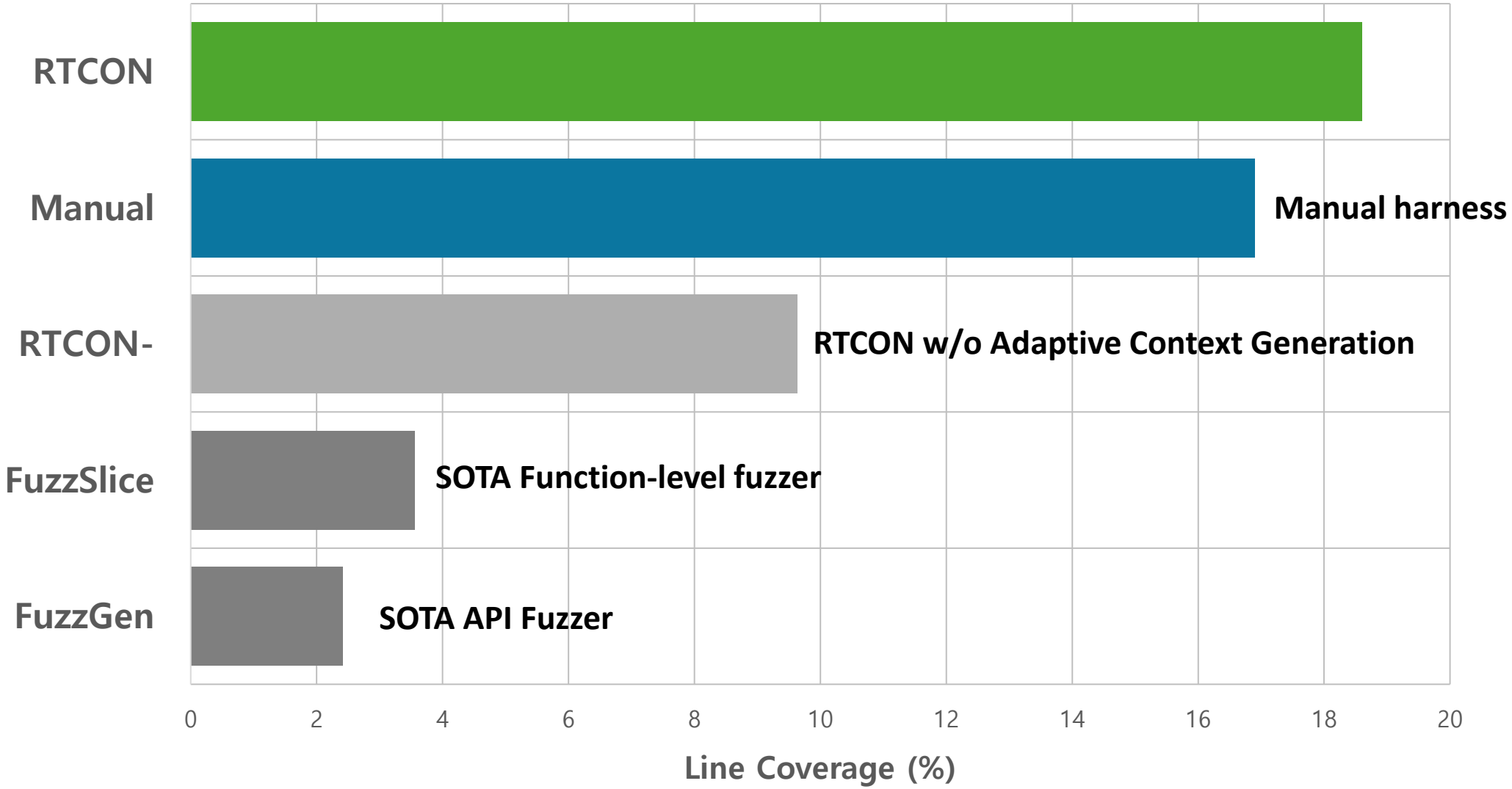
- RQ1. Can RTCON find bugs in real-world RTOSes?
- RQ2. How effective is RTCON compared to existing function-level fuzzers?
- RQ3. How effective is RTCON in classifying crashes?

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.

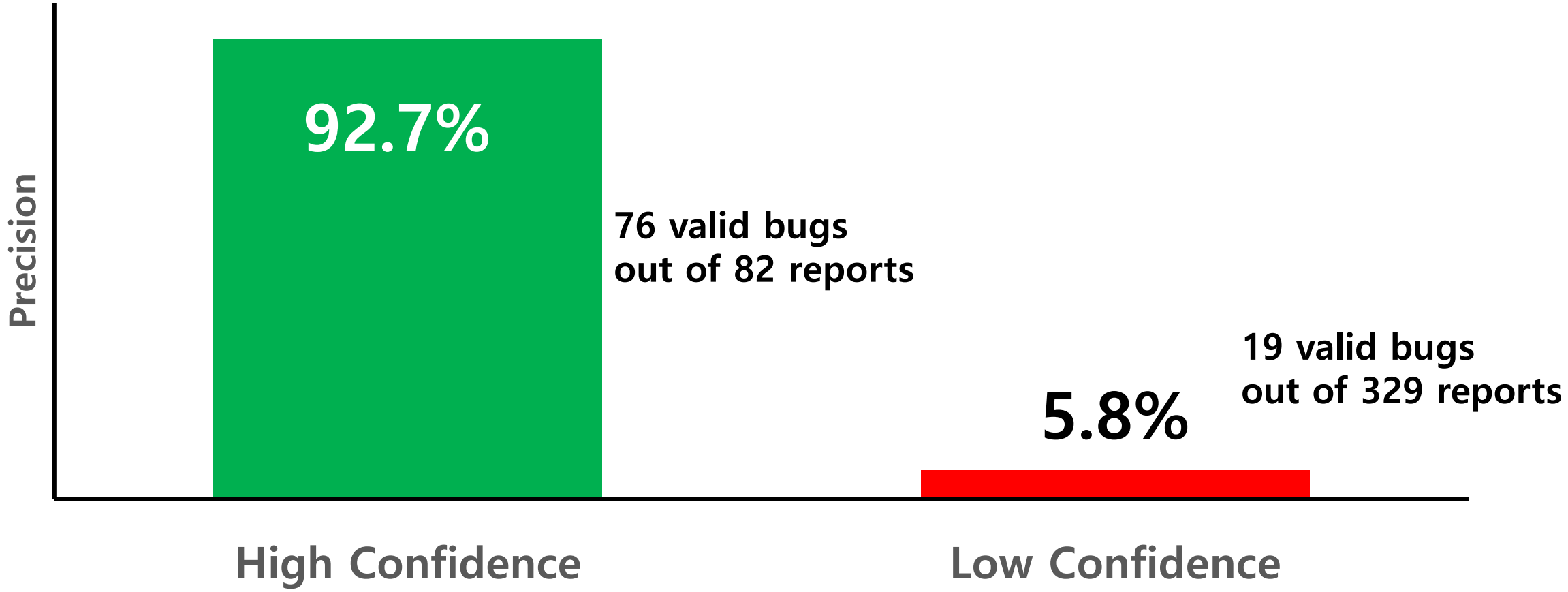
Line Coverage by RTCON



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

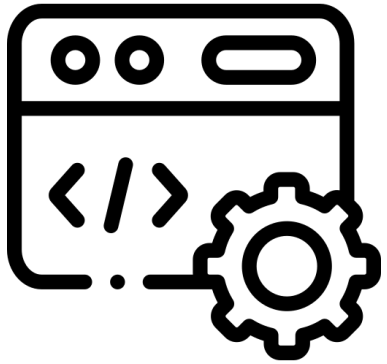
Effectiveness of MLC



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.

Thank you

Contact:
Eunkyu Lee (ekleezg@kaist.ac.kr)