Uncovering the iceberg from the tip:

Generating API Specifications for Bug Detection via Specification Propagation Analysis

Miaoqian Lin , Kai Chen, Yi Yang, Jinghua Liu

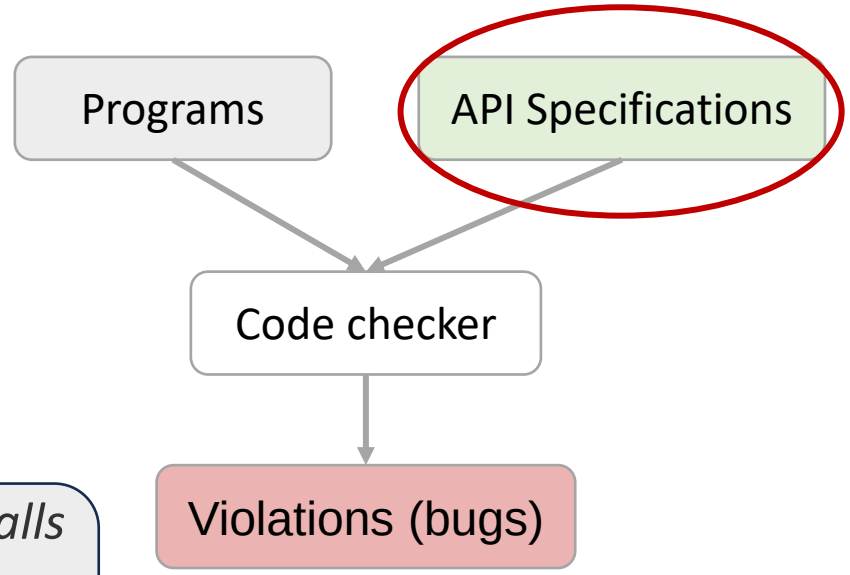
{linmiaoqian, chenkai, liujinghua, yangyi}@iie.ac.cn

Motivation: Bugs and Specifications

- Software is built with **various APIs** that encapsulate complex semantics.
- To use APIs safely, developers must adhere to **their specifications**.
- **Violations** of these specifications are API misuse, which can be further exploited by attackers.

```
01 static blk_status_t bsg_queue_rq(...) {
02     struct device *dev = q->queuedata;
03     ...
04     if (!get_device(dev))
05         return BLK_STS_IOERR;
06
07     if (!bsg_prepare_job(dev, req))
08 -       return BLK_STS_IOERR;
09 +       goto out;
10      ...
11 +  out:
12         put_device(dev);
13         return sts;
14 }
```

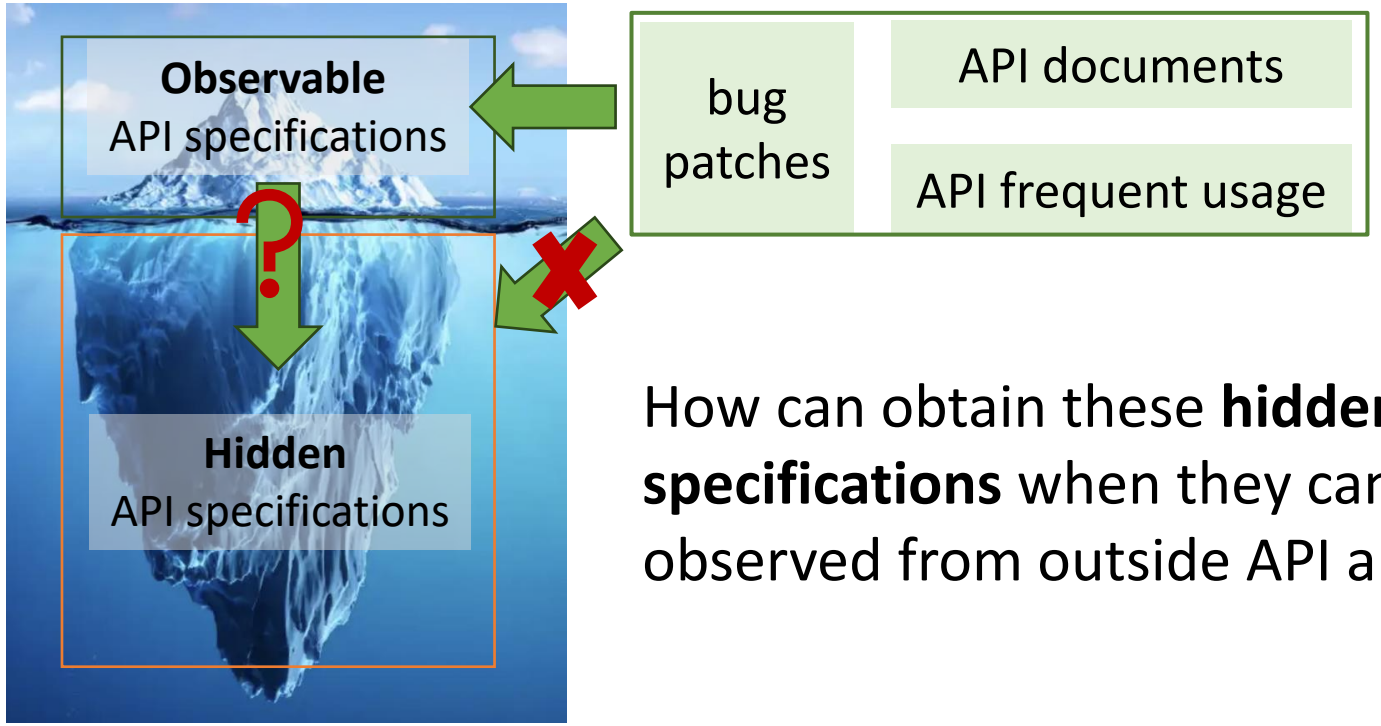
*"Make sure that bsg_queue_rq() calls **put_device()** if an error is encountered after **get_device()** was successful."*



Motivation: Current Work and Limitations

- **Ideal way**: Understand each API's behavior and infer its specifications
  - Code complexity makes this difficult
- **Current work:** Extract specifications from **observable API artifacts**
  1. **API documents:** e.g., Advance (CCS'2020), APICAD (ICSE'2023)
  2. **API frequent usage:** e.g., APP-Miner (S&P'2024), APISan (Security'2016)
  3. **Bug patches:** e.g., APHP (Security'2023)
- **Problem: Limited by the availability of artifacts**
  - **For example:** The API **bus_find_device** requires certain specification to be followed after its call, but this is not documented.

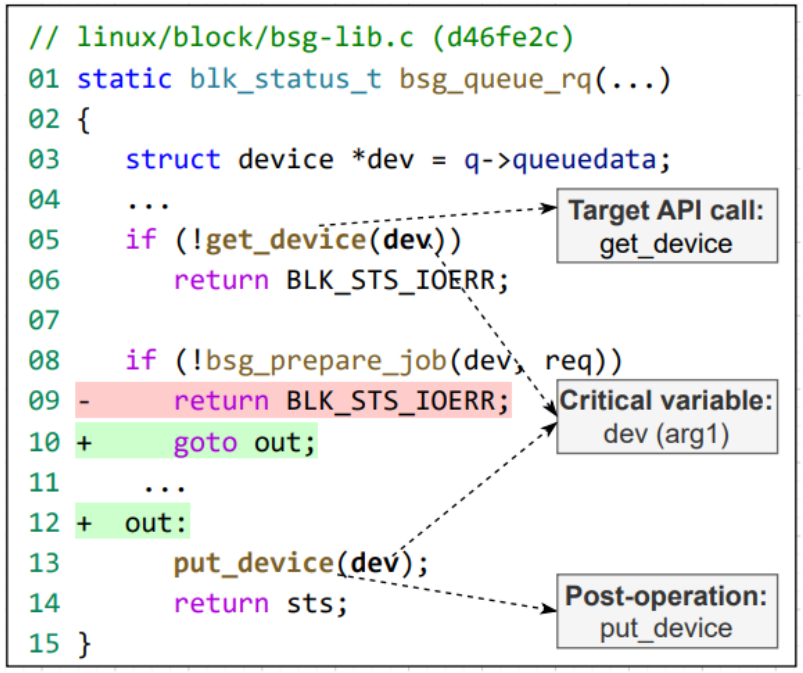
Motivation: From Known to Unknown



How can obtain these **hidden API specifications** when they cannot be observed from outside API artifacts?

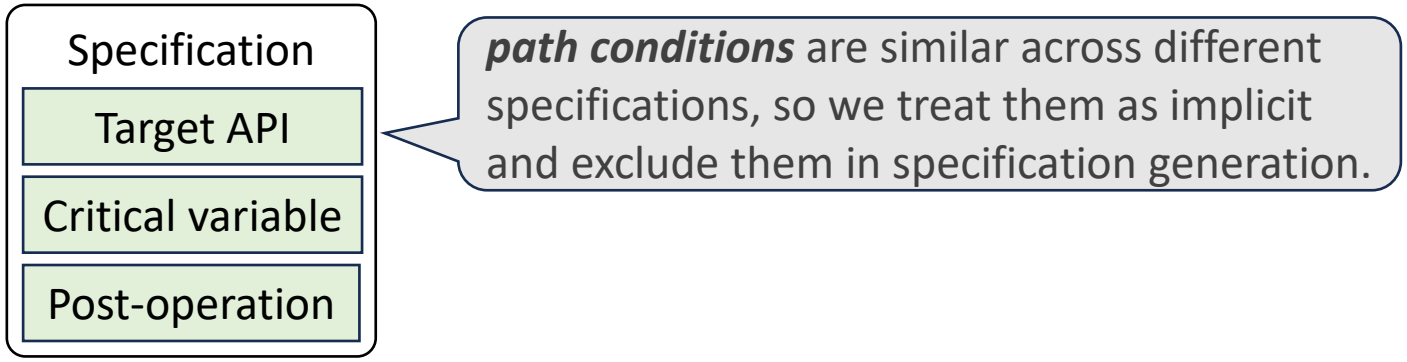
It is possible to infer **hidden specifications (unknown)** from **observable specifications (known)**?

Background: Bugs and Specifications

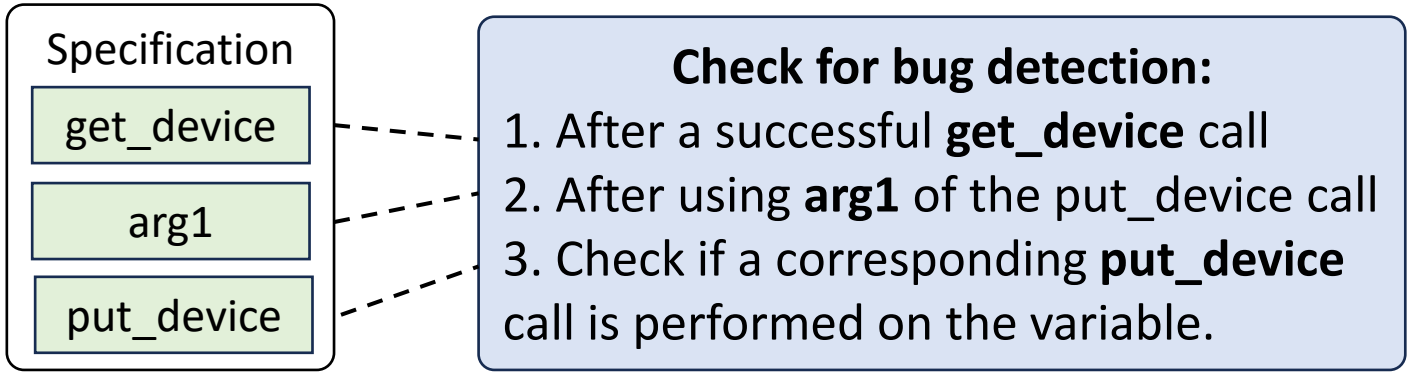


The API **get_device** obtain a reference to the device **dev**, which needs to released using **put_device**() after use.

- The specification triplet

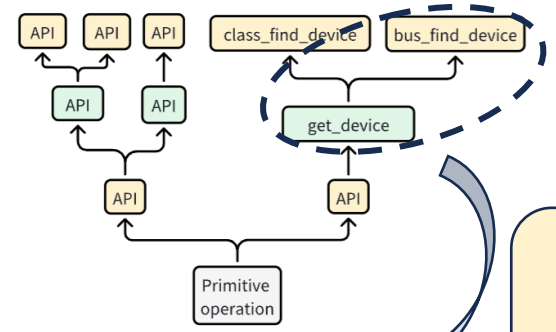


- The specification example



Insight: API Specification Propagation

- Specification propagation from get_device to bus_find_device



upper-level API: **bus_find_device**

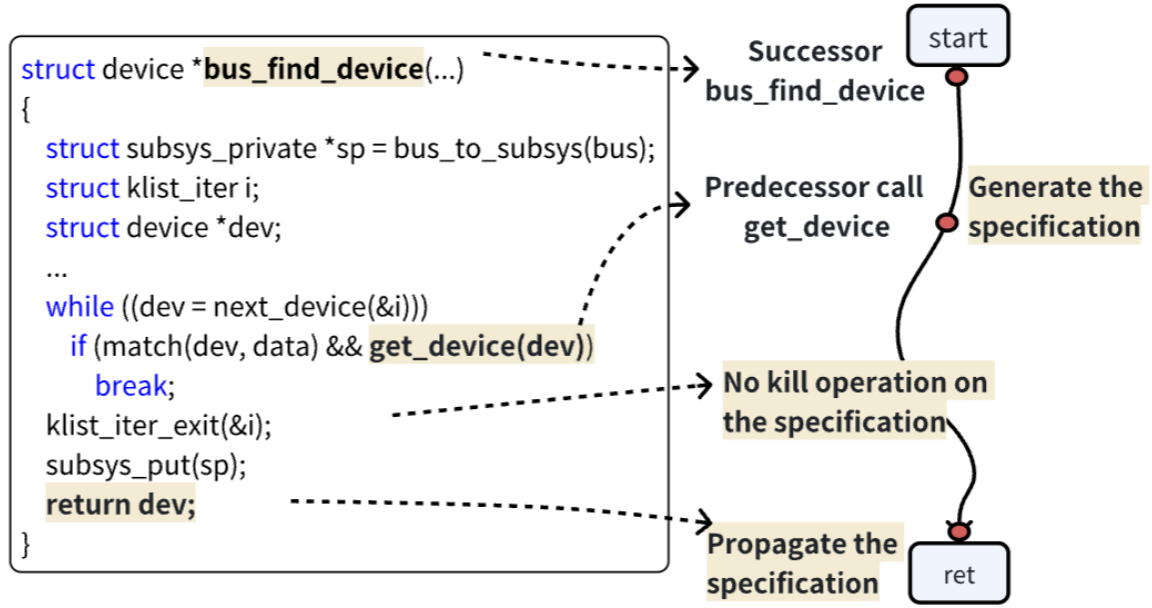
calls

lower-level API: **get_device**

Specification propagation from **get_device** to **bus_find_device**

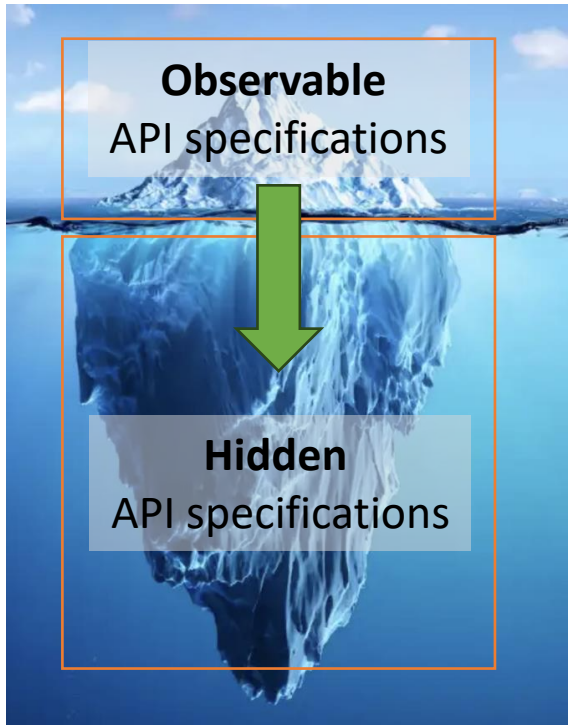
requires calling **put_device** to release the obtained reference count.

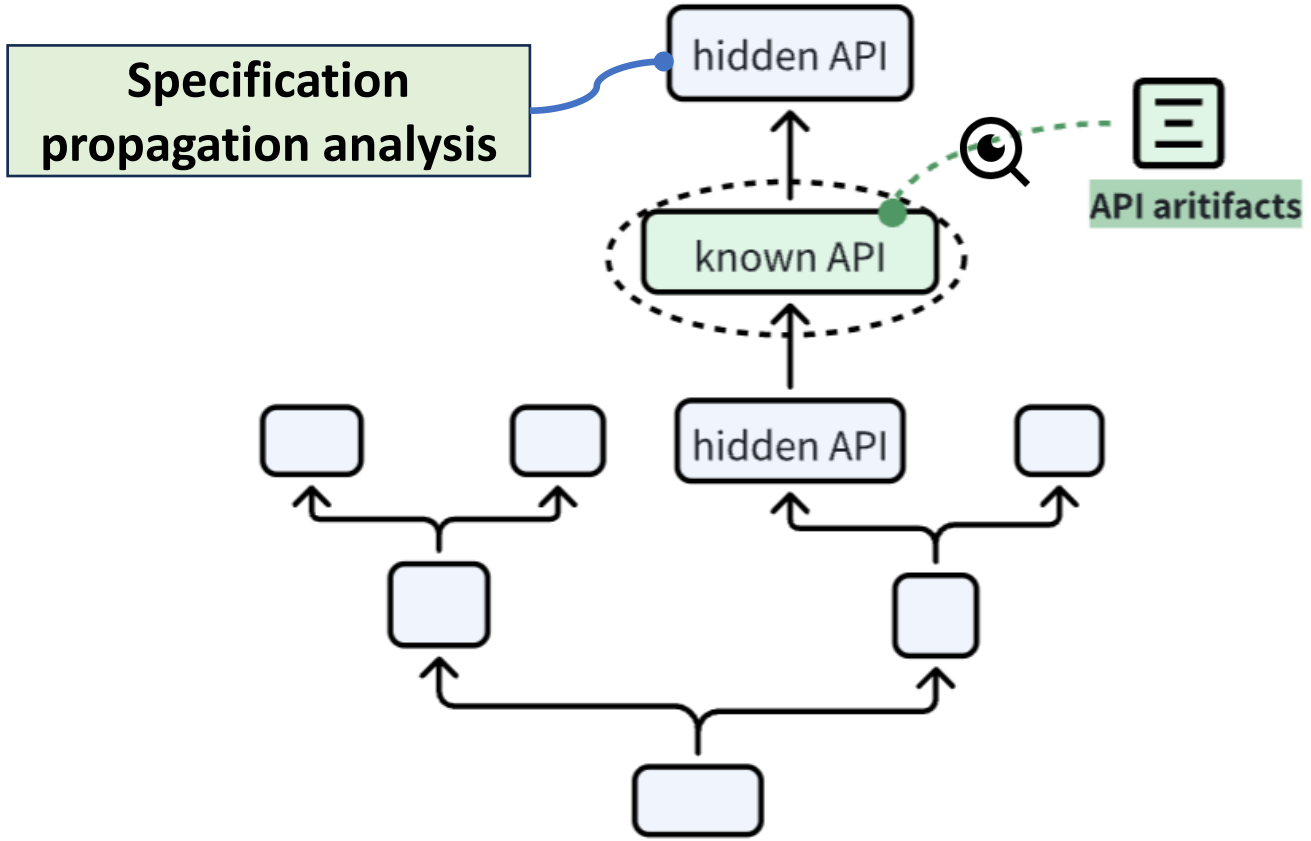
```
struct device *bus_find_device(...)
{
  struct subsys_private *sp = bus_to_subsys(bus);
  struct klist_iter i;
  struct device *dev;
  ...
  while ((dev = next_device(&i)))
    if (match(dev, data) && get_device(dev))
      break;
  klist_iter_exit(&i);
  subsys_put(sp);
  return dev;
}
```



Specifications propagate between APIs through the API call chain

Insight: Specification Propagation Analysis

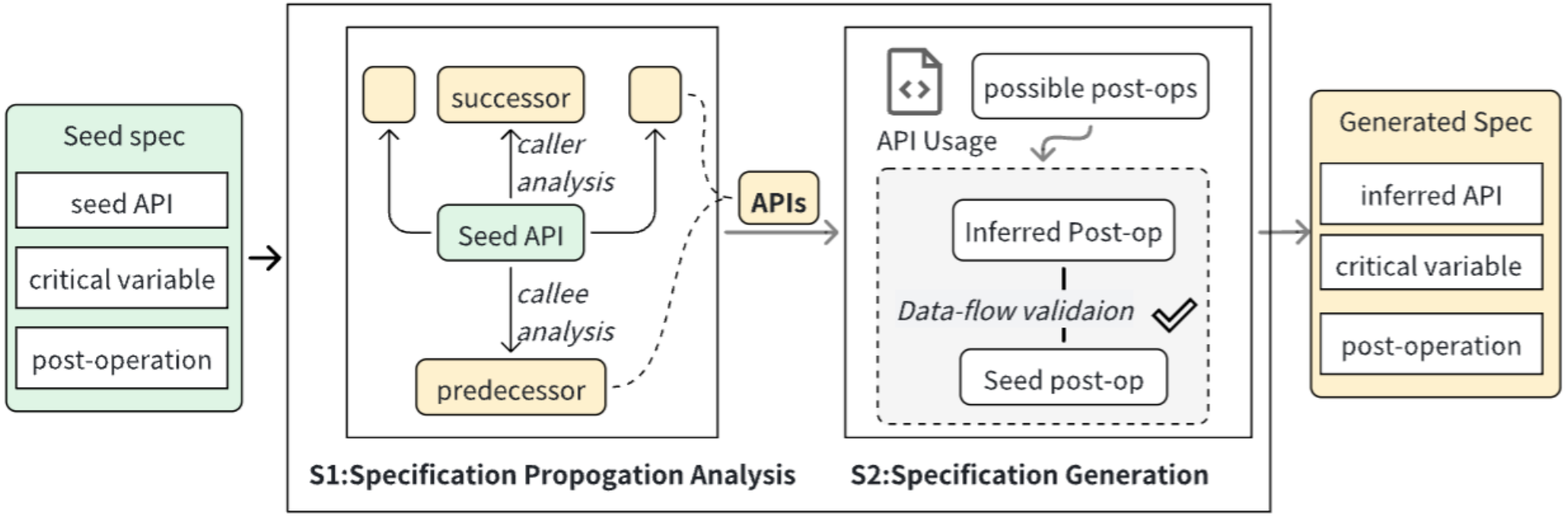




Use **specification propagation analysis** to reveal the hidden API specifications

Overview

- APISpecGen: generate new specifications using seed specifications
  - **Step1: Specification Propagation Analysis:** identifies which APIs the specifications may propagate to (successors) or originate from (predecessors).
  - **Step2: Specification Generation:** generate specific specifications for inferred APIs, i.e., determine their corresponding post-operations.

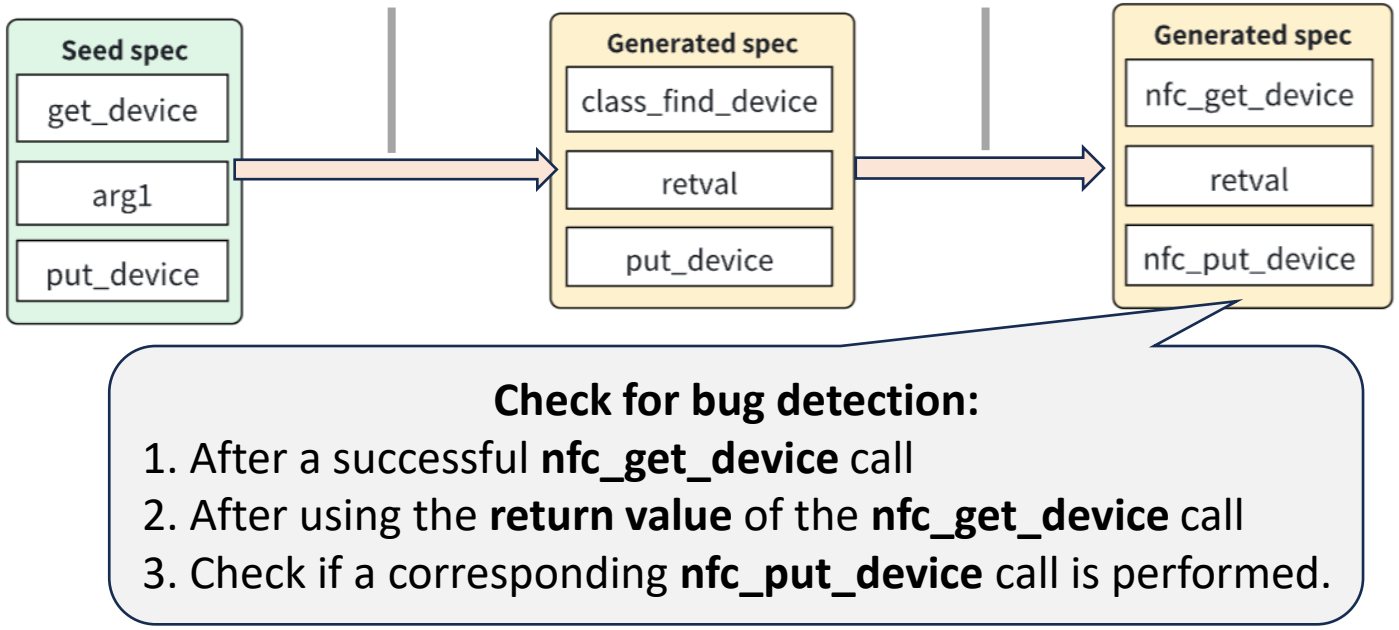


Method: Working Example

- Generated specifications

```
01 struct device *class_find_device(... *class, ...)
02 {
03   struct device *dev;
04   ...
05   while ((dev = class_dev_iter_next(&iter))) {
06     if (match(dev, data)) {
07       get_device(dev);
08       break;
09     }
10   }
11   return dev;
12 }
```

```
01 struct nfc_dev *nfc_get_device(unsigned int idx)
02 {
03   struct device *d;
04
05   d = class_find_device(&nfc_class, ...);
06   if (!d)
07     return NULL;
08
09   return to_nfc_dev(d);
10 }
```



Check for bug detection:

1. After a successful **nfc_get_device** call
2. After using the **return value** of the **nfc_get_device** call
3. Check if a corresponding **nfc_put_device** call is performed.

- Detected bug

```
01 static int nfc_genl_vendor_cmd(struct sk_buff *skb,
02                 struct genl_info *info)
03 {
04   struct nfc_dev *dev;
05   ...
06   dev = nfc_get_device(dev_idx);
07   if (!dev || !dev->vendor_cmds || !dev->n_vendor_cmds)
08     return -ENODEV;
09
10   if (info->attrs[NFC_ATTR_VENDOR_DATA]) {
11     ...
12     data_len = nla_len(info->attrs[NFC_ATTR_VENDOR_DATA]);
13     if (data_len == 0)
14       return -EINVAL;
15   }
16   ...
17   return -EOPNOTSUPP;
18 }
```

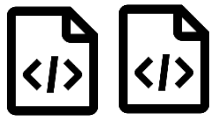
missing nfc_put_device

Detected API misuse in the Linux kernel: missing **nfc_put_device** after **nfc_get_device** after the critical variable **dev** usage, causing reference count leak.

Method: Specification Propagation Analysis

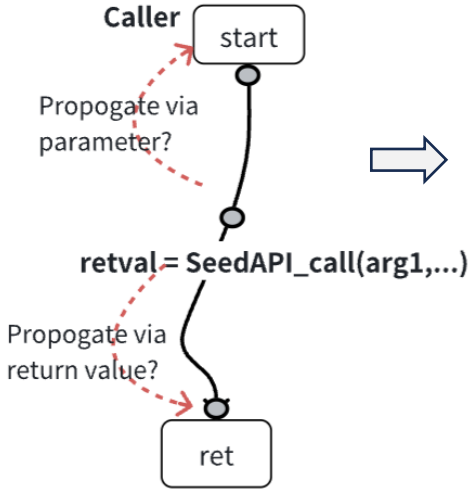
- **Caller Analysis to Get Successors**

❶ Get callers of the **seed API**



```
bus_find_device
class_find_device
tifm_device_probe
dpm_prepare
…
```

❷ Check for **critical variable**'s propagation

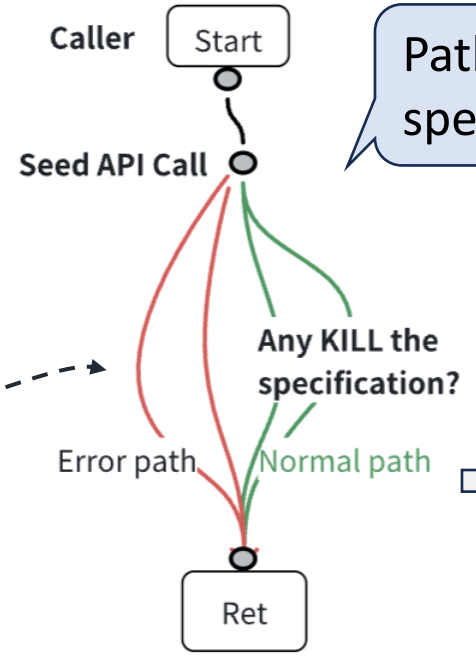



```
bus_find_device
class_find_device
tifm_device_probe
~~dpm_prepare~~
…
```

Seed

- get_device
- arg1
- put_device

❸ Check for **specification**'s propagation



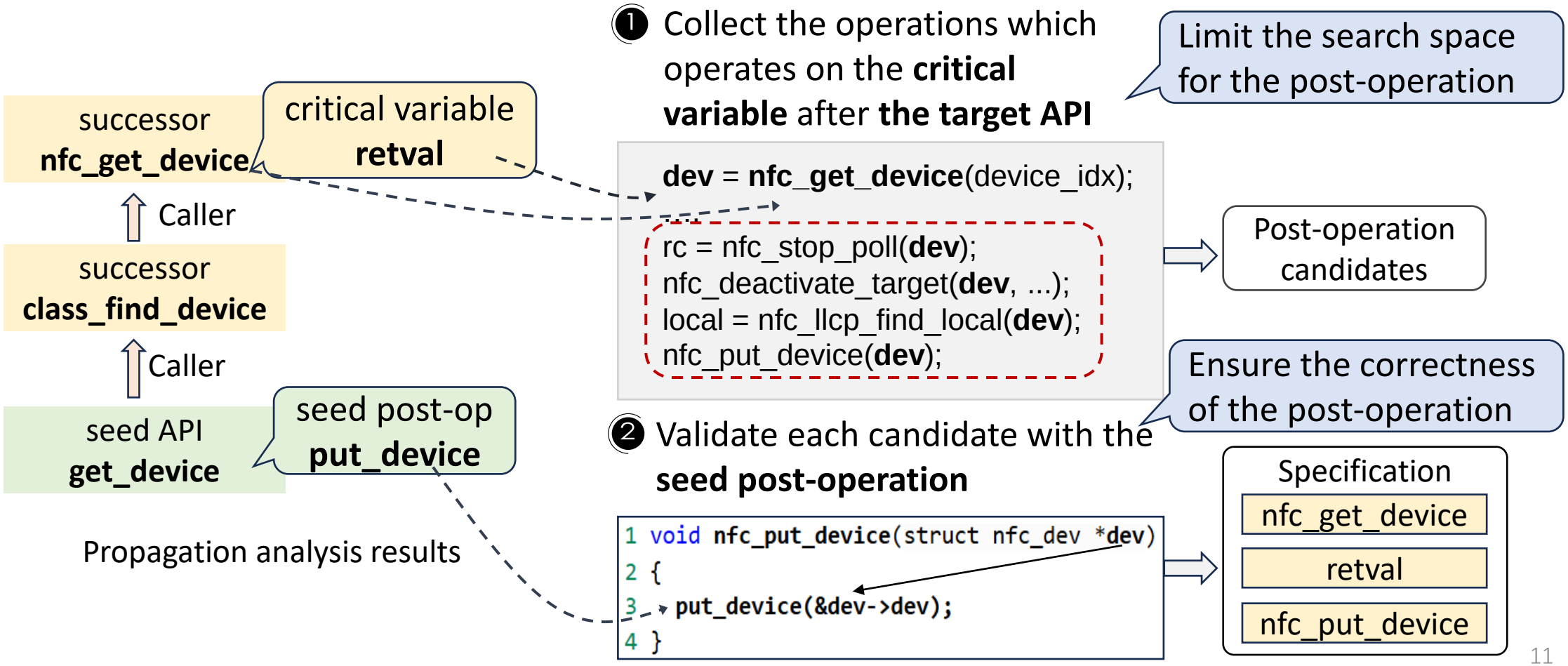
Path-sensitive strategy for specification propagation

```
bus_find_device
class_find_device
~~tifm_device_probe~~
~~dpm_prepare~~
…
```

Condition-by-condition check for propagation analysis, from coarse-grained to fine-grained analysis for efficiency.

Method: Specification Generation

- **Problem:** the post-operations may change during specification propagation
- **Solution:** utilizes **API usage** and **data-flow validation**



Evaluations

- Setup
  - **Tested program:** the Linux Kernel, with more 23 million LoC
  - **Seeds:** six specifications collected by previous work

- Evaluations
  - Specification Generation
  - Bug Detection
  - Compared with SOTAs
  - Quality of API artifacts

Target API	Post-Operation	Var	Commit
get_device	put_device	arg1	d46fe2cb2dc
device_initialize	put_device	arg1	a5808add9a6
try_module_get	module_put	arg1	44f8baaf230
kmalloc	kfree	retval	493ff661d43
kstrdup	kfree	retval	e629e7b525a
PTR_ERR	IS_ERR	arg1	59715cffce1

Six seed specification for evaluations

Results

- Specification Generation and Bug Detection

Seed API	#Spec	# New Bug	Bug Type
get_device	760	137	Refcount leak
device_initialize	91	6	Refcount leak
try_module_get	58	1	Refcount leak
kmalloc	1202	30	Memory leak
kstrdup	14	1	Memory leak
ERR_PTR	5207	11	Null-pointer-deref
Total	**7332**	**186**	-

Efficiency:
Generate specifications **in 2 hours**, averaging **1 second** per specification.

Accuracy:
Generated specifications are **100%** accurate.

Effectiveness:
APISpecGen generated **7,332** specifications with from just six seed specifications, with **186** new bugs detected.

Results

- **SOTAs and the quality of API artifacts for specification extraction**

Note: evaluated SOTAs on the specifications and bugs identified by APISpecGen

Method	Coverage of specifications	Coverage of Bugs
Advance	14%	10%
IPPO	-	0%
SinkFinder	11%	10%
APHP	21%	17%

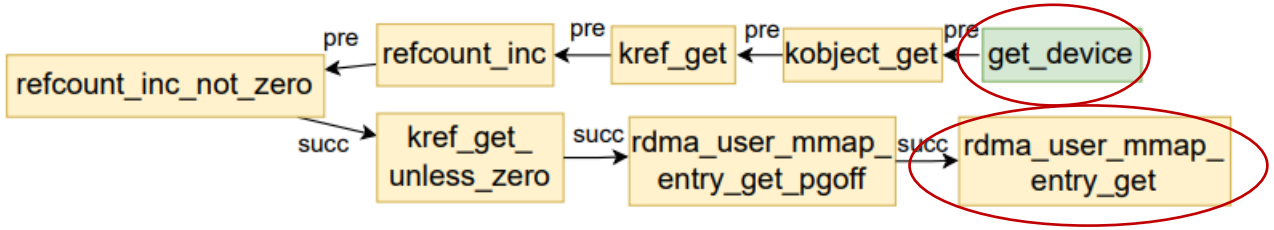
SOTAs fail to extract most specifications and miss corresponding bugs.

API Document	**87%** of the specifications are not mentioned in document
API Frequent Usage	**93%** of the specifications are not frequently followed in usage code
API Name	**12.71%** of APIs lack informative subwords in their name

Artifact-based methods are inherently limited by the quality of API artifacts.
(e.g. **87%** of specifications are not documented)

Results

- **Characteristic of Generated Specifications**
  - The inferred API can differ significantly from the original seed API.
  - APIs evolve from the same primitive APIs and share similar specifications



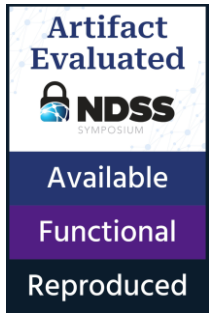
from seed API get device to rdma_user_mmap_entry_get

- **Generalizability**
  - APISpecGen successfully generated **39** specifications for OpenSSL and **76** for FFmpeg from just **one seed each**

Summary

- **Previous:** Observe API specifications externally, treating APIs individually.
- **Ours:** Generate specifications from internal facts, using connections between APIs.
- ***Idea: API Specification Propagation***
- **Framework**: APISpecGen, generates specifications using seed specification with **iterative bidirectional specification propagation analysis**.
- **Results**: Generate **7332** specifications using only **6** seed specifications. **87%** of these specifications are not documented; Detect **186** new bugs in Linux kernel.

Open-source
https://github.com/Yuuoniy/APISpecGen



Thanks for your attention!

Q&A

Paper

Code