VUDDY: A Scalable Approach for Vulnerable Code Clone Detection

Seulbae Kim, Seunghoon Woo, Heejo Lee, and Hakjoo Oh

May 23, 2017

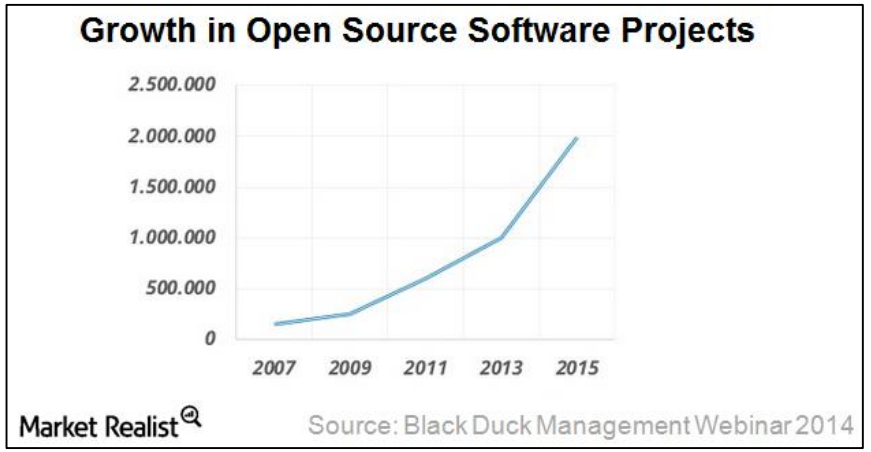
Question

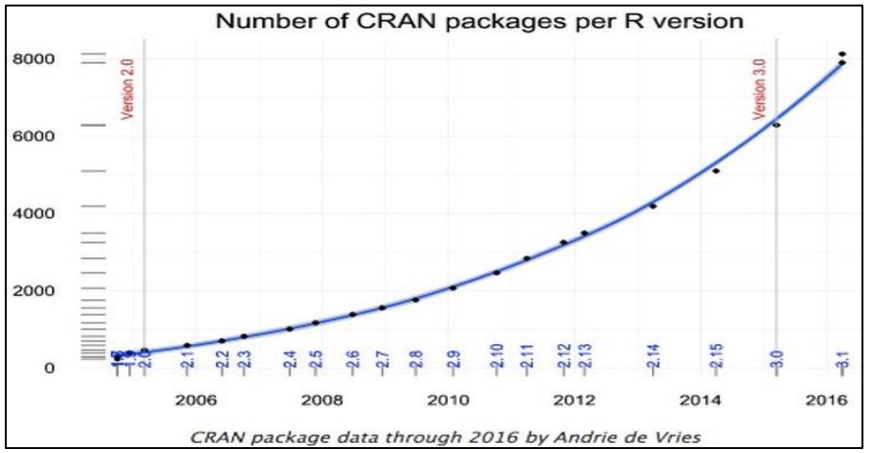
- Number of unpatched vulnerabilities in smartphone firmware's source code?

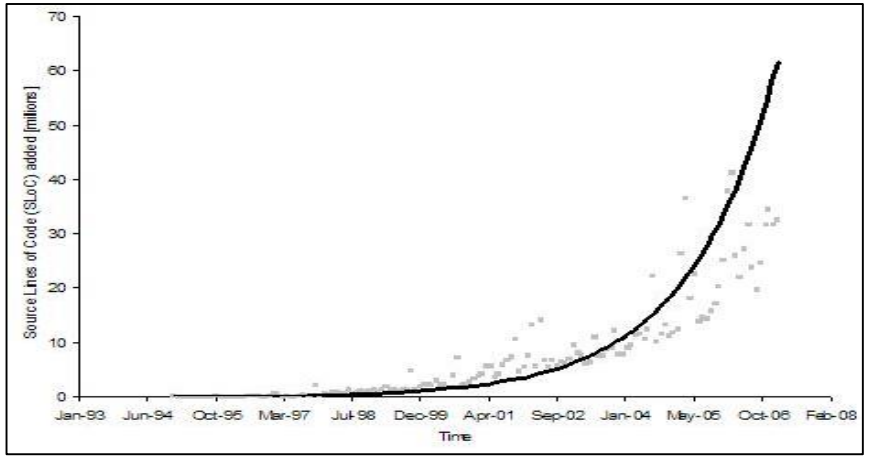
200+ unpatched vulnerable code clones detected!

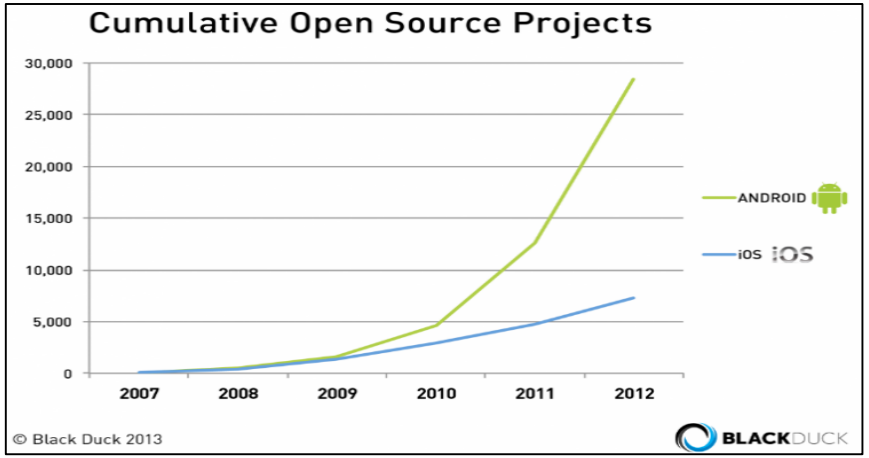
Motivation

- Number of open source software is increasing



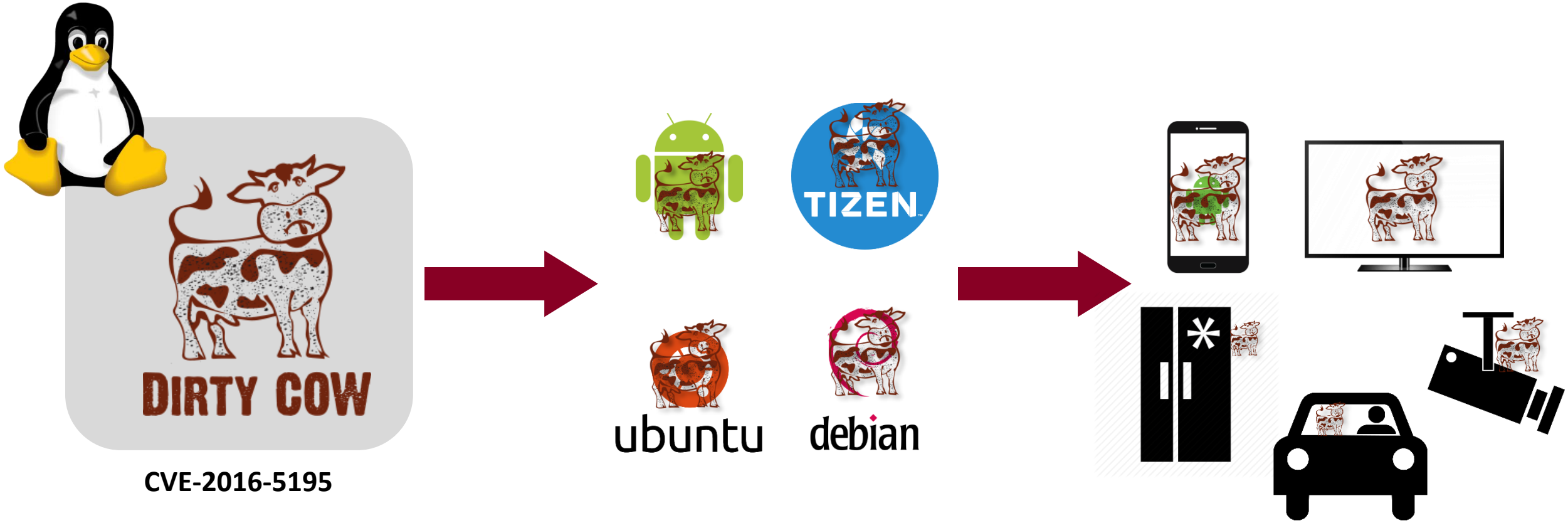






Motivation

- Code clones – reused code fragments
  - Major cause of vulnerability propagation



CVE-2016-5195

Problem: Scalable & Accurate Vulnerable Code Clone Discovery

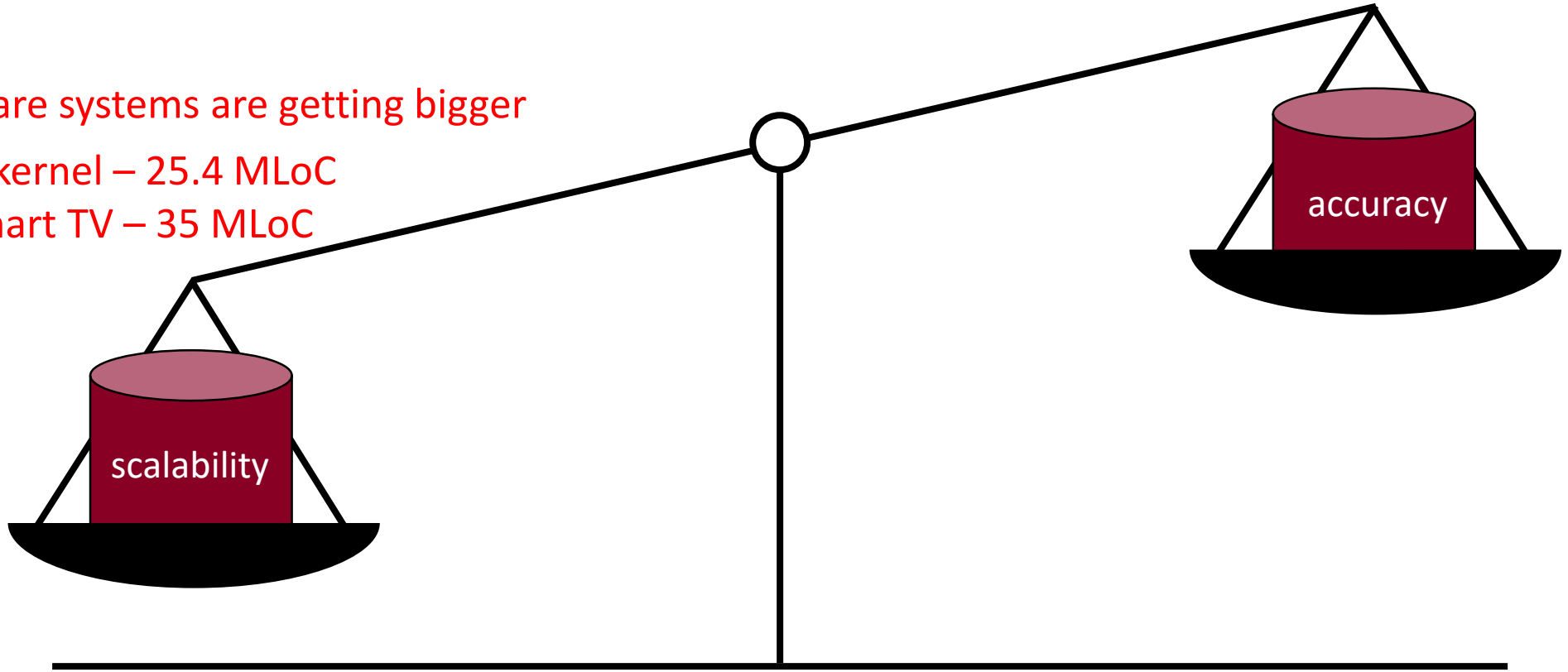
Scalable & Accurate Vulnerable Code Clone discovery

- Scalability

Software systems are getting bigger

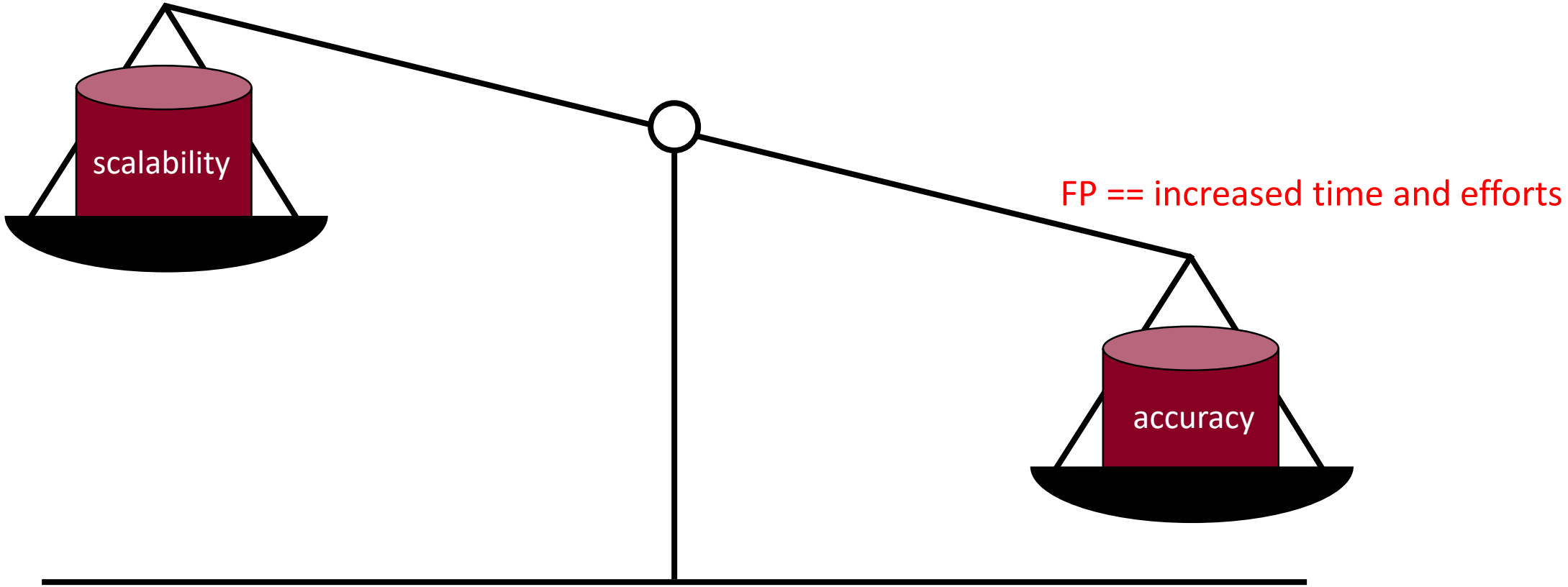
Linux kernel – 25.4 MLoC

“L” Smart TV – 35 MLoC



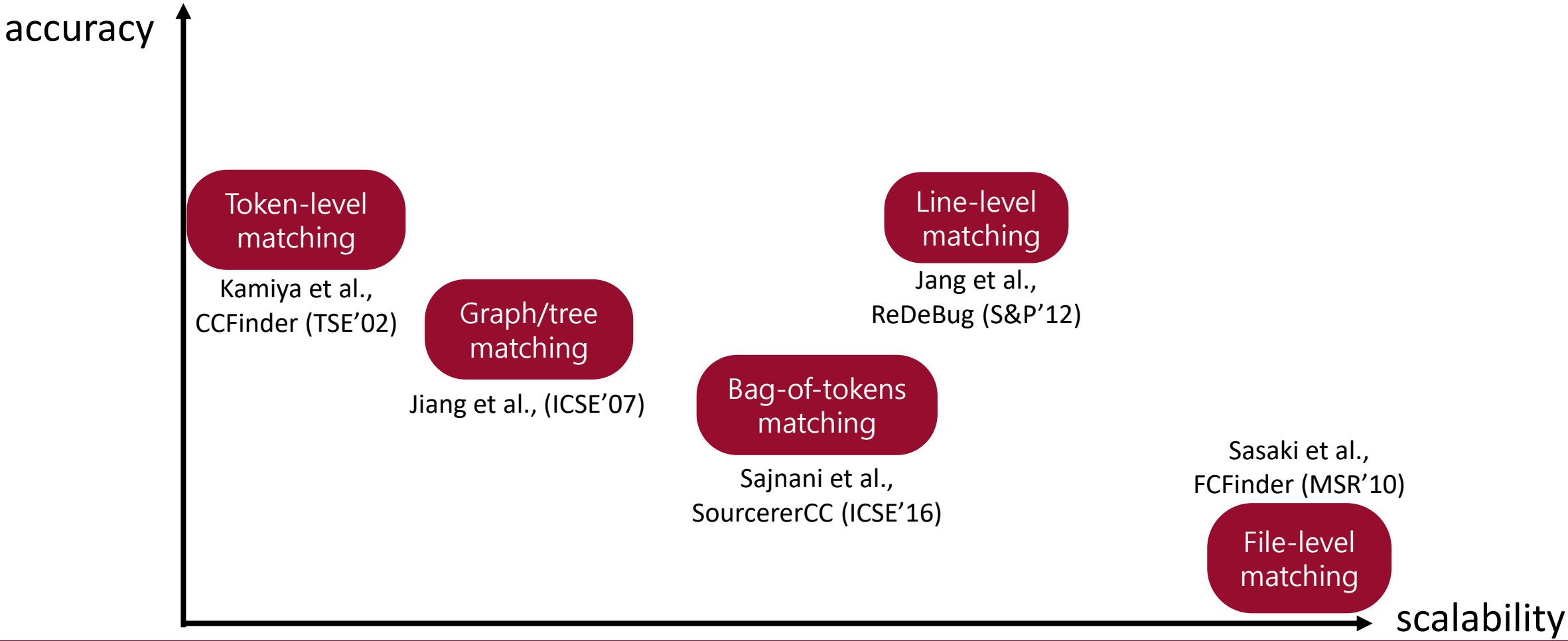
Scalable & Accurate Vulnerable Code Clone discovery

- Accuracy



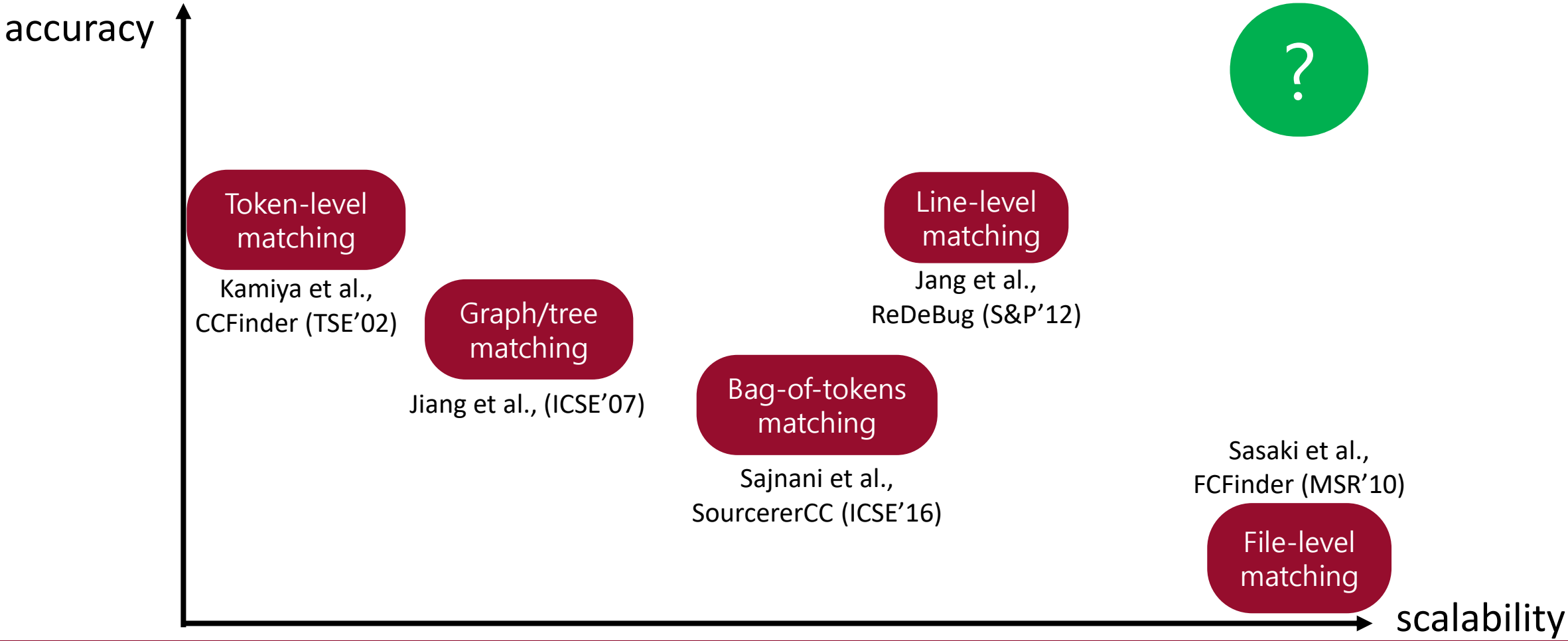
Scalable & Accurate Vulnerable Code Clone discovery

- Previous approaches



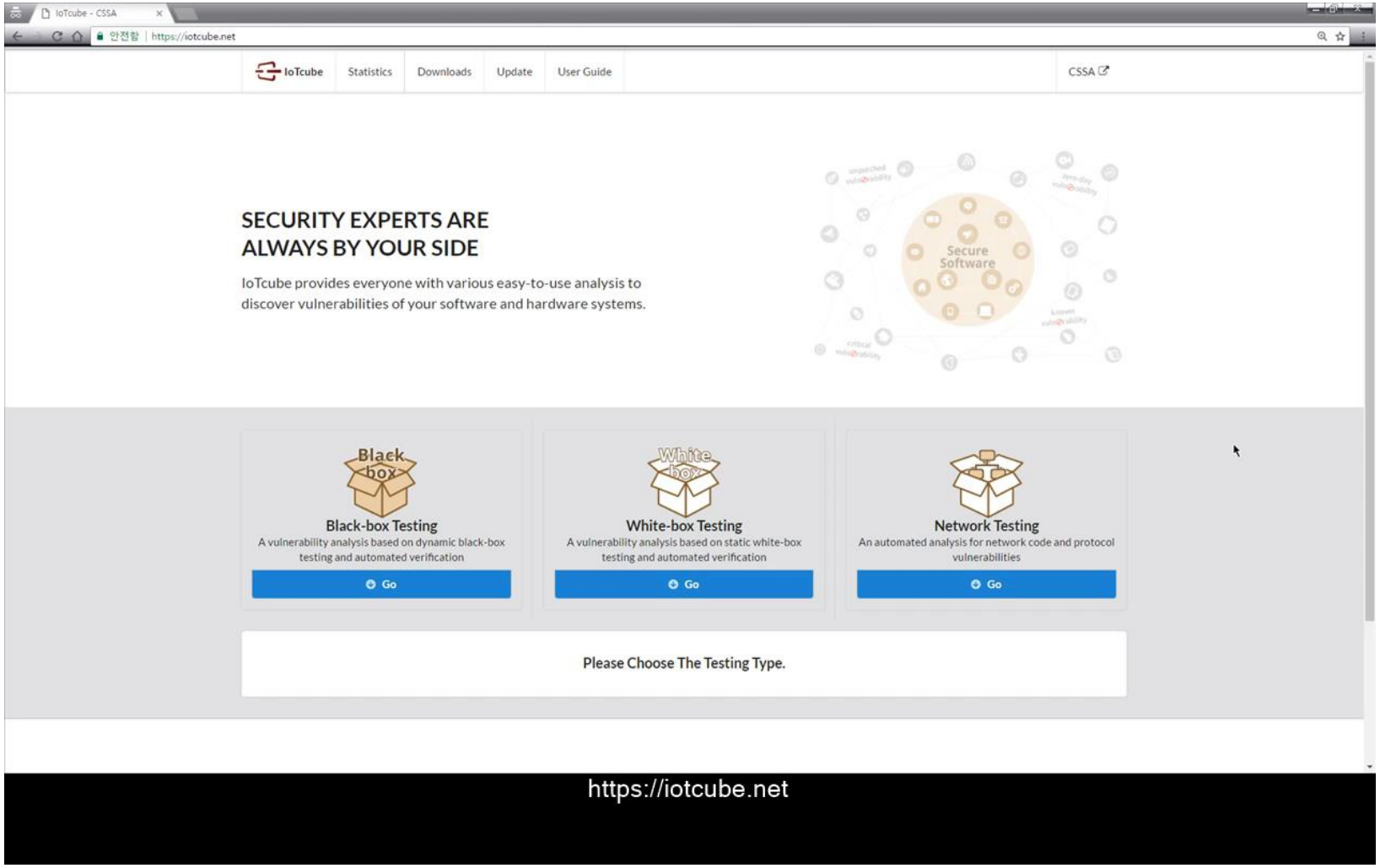
Scalable & Accurate Vulnerable Code Clone discovery

- Goal



Proposed Method: VUDDY

Demonstration of VUDDY



Proposed method: VUDDY

- VUDDY: VUlnerable coDe clone DiscoverY

Proposed method: VUDDY

- VUDDY: VUlnerable coDe clone DiscoverY
  - Searches for vulnerable code clones

Proposed method: VUDDY

- VUDDY: VUlnerable coDe clone DiscoverY
  - Searches for vulnerable code clones
  - Scales beyond **1 BLoC** target

Proposed method: VUDDY

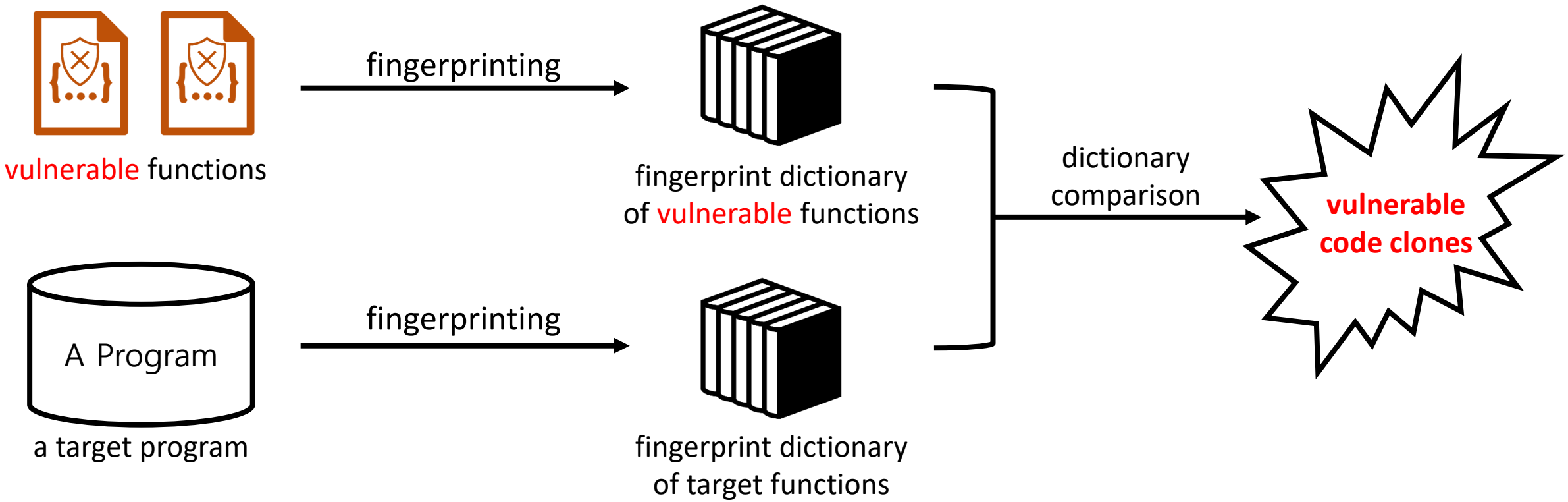
- VUDDY: VUlnerable coDe clone DiscoverY
  - Searches for vulnerable code clones
  - Scales beyond **1 BLoC** target
  - Detects both known & **unknown** vulnerability

Proposed method: VUDDY

- VUDDY: VUlnerable coDe clone DiscoverY
  - Searches for vulnerable code clones
  - Scales beyond **1 BLoC** target
  - Detects both known & **unknown** vulnerability
  - Low false positive rate

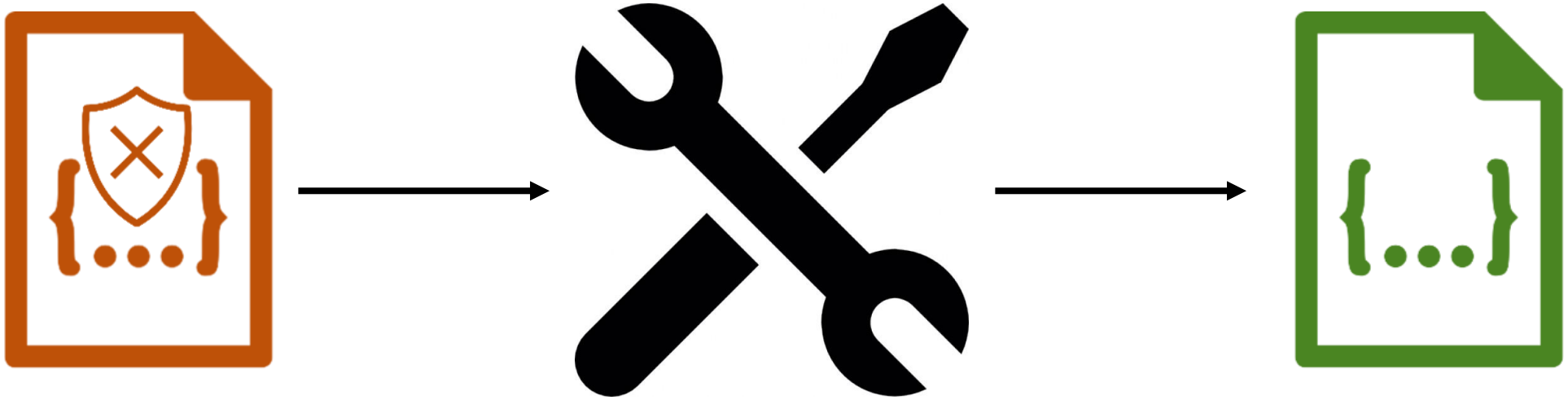
Proposed method: VUDDY

- Overview



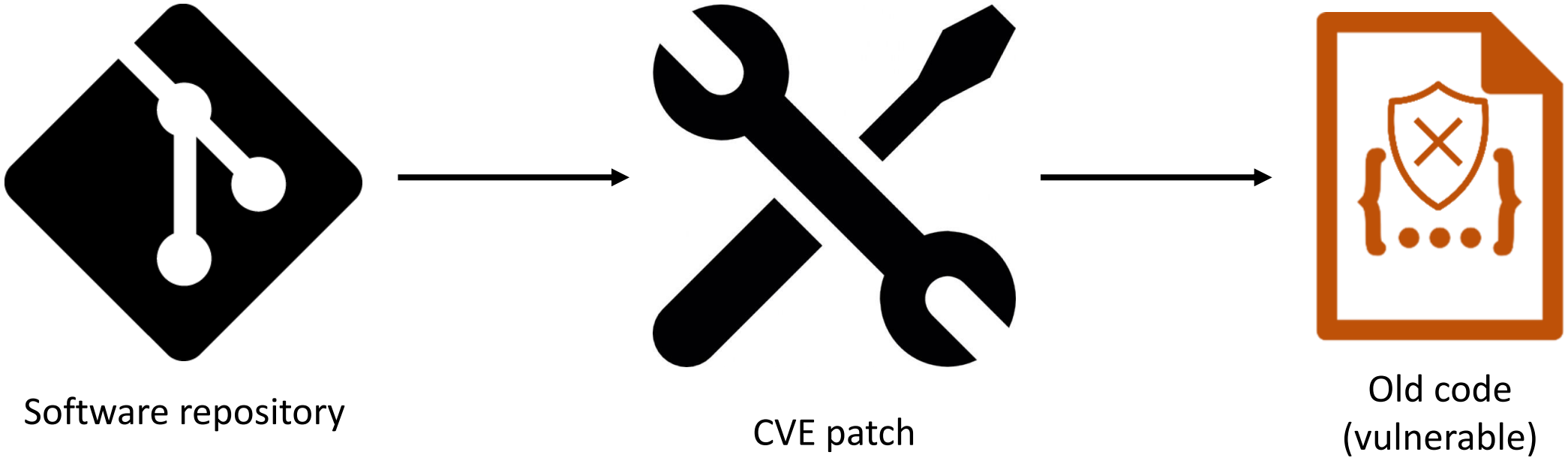
Collecting vulnerable code

- Vulnerability patching

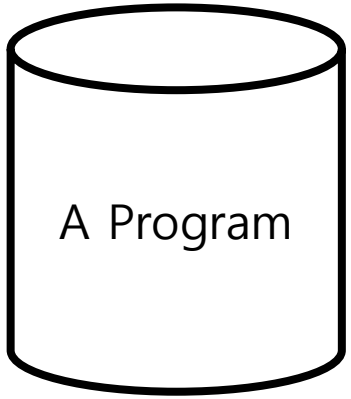


Collecting vulnerable code

- Reconstructing vulnerability from security patch

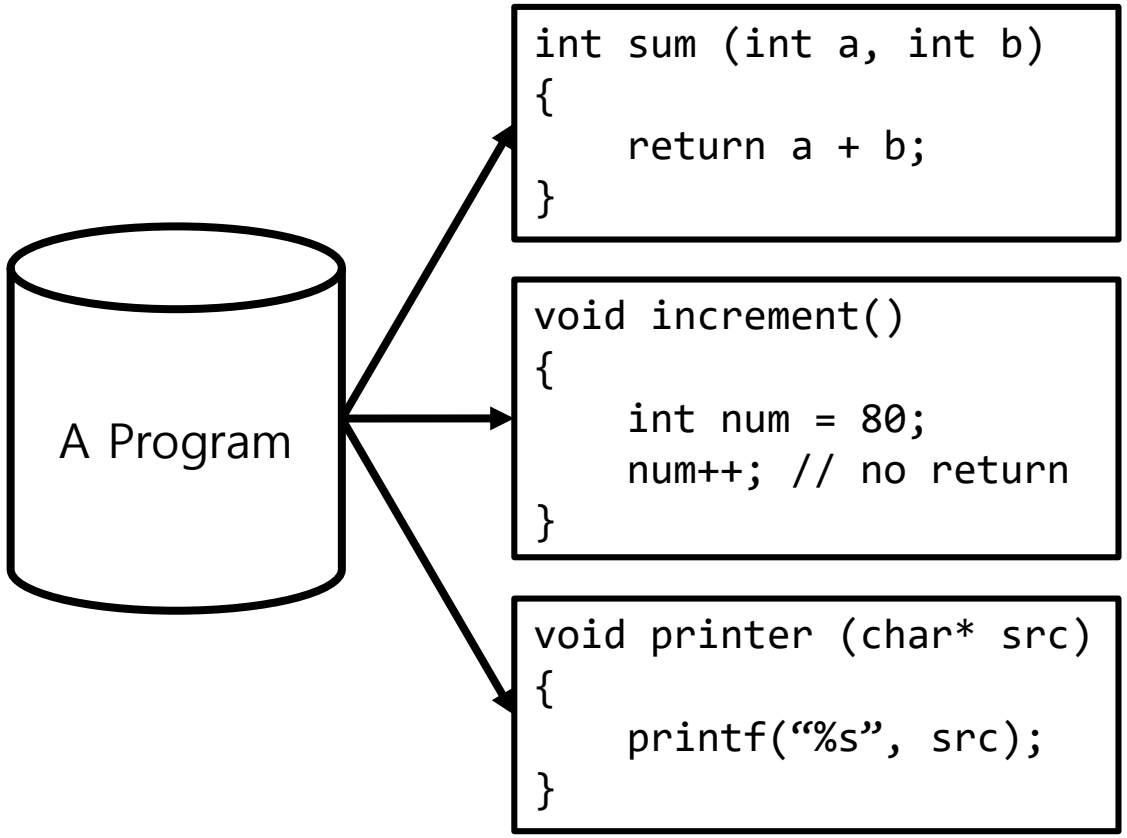


Fingerprinting a program



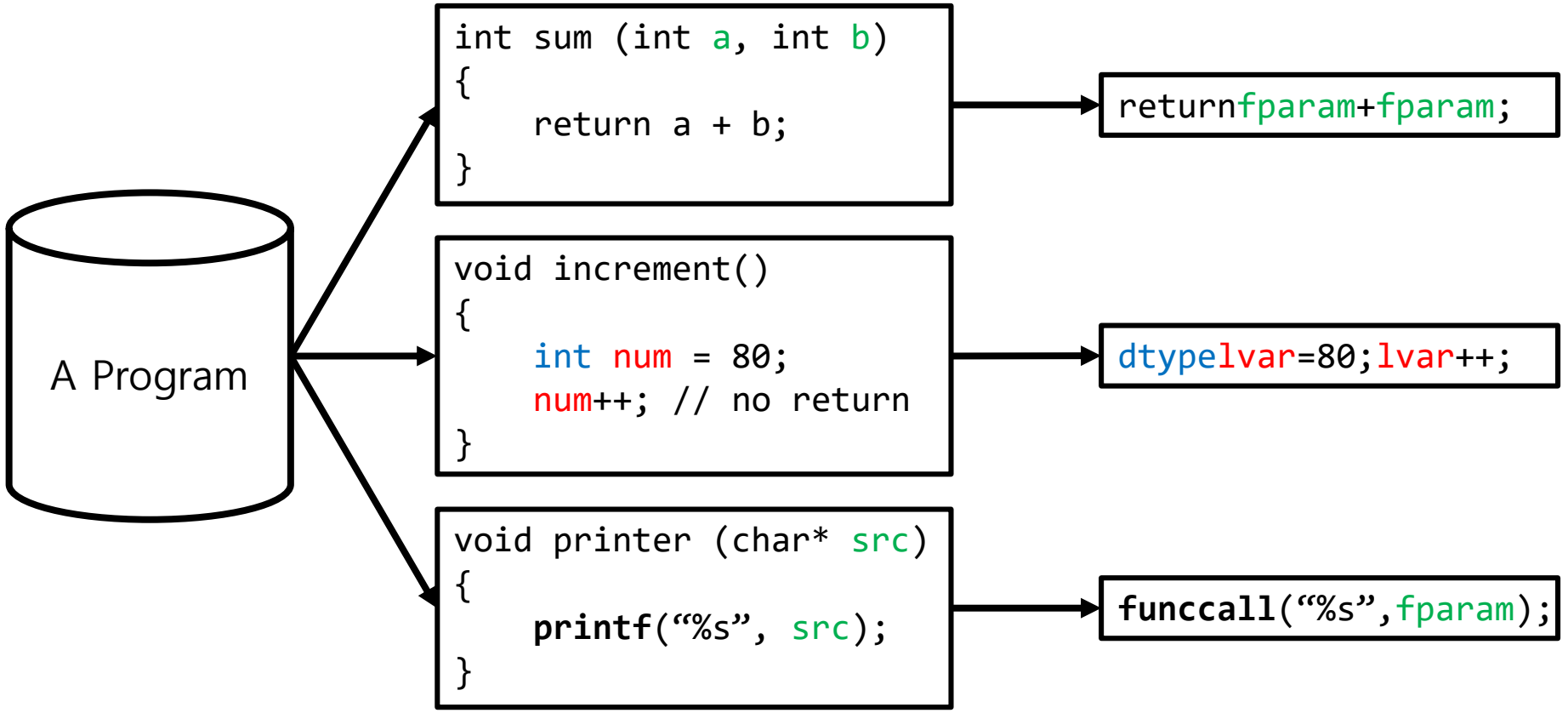
Fingerprinting a program

1. Retrieve all functions from a program



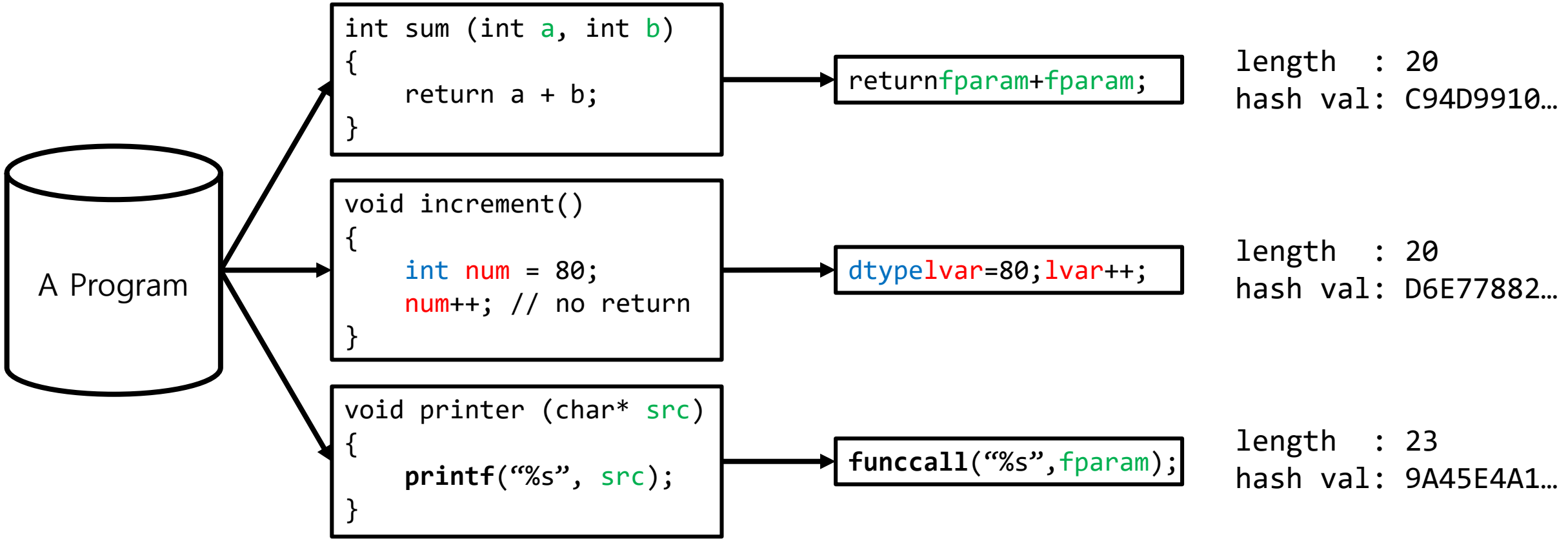
Fingerprinting a program

2. Apply abstraction and normalization to functions



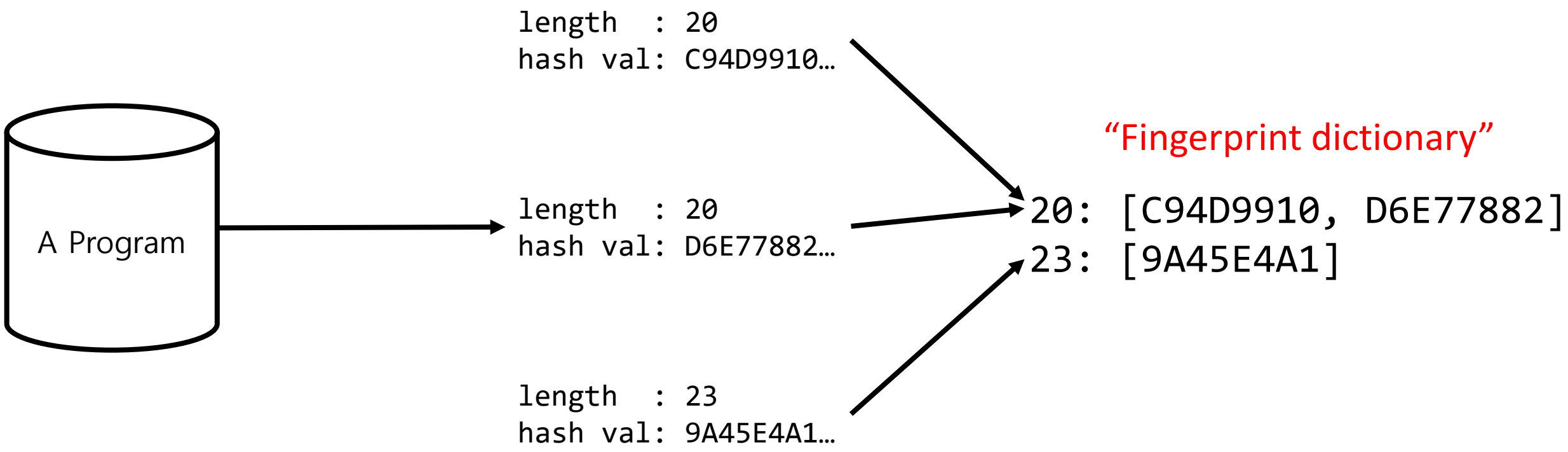
Fingerprinting a program

3. Compute length and hash value



Fingerprinting a program

4. Store in a dictionary



Abstraction

- Transform function by replacing
  - Formal parameters
  - Data types
  - Local variables
  - **Function names**

Level 0: No abstraction

```
1 void avg (float arr[], int len) {
2     static float sum = 0;
3     unsigned int i;
4
5     for (i = 0; i < len; i++) {
6         sum += arr[i];
7     }
8
9     printf("%f %d\n", sum/len, validate(sum));
10 }
```

Abstraction

- Transform function by replacing
  - Formal parameters
  - Data types
  - Local variables
  - **Function names**

Level 1: Formal parameter abstraction

```
1 void avg (float FPARAM[], int FPARAM) {
2     static float sum = 0;
3     unsigned int i;
4
5     for (i = 0; i < FPARAM; i++) {
6         sum += FPARAM[i];
7     }
8
9     printf("%f %d\n", sum/FPARAM, validate(sum));
10 }
```

Abstraction

- Transform function by replacing
  - Formal parameters
  - Data types
  - Local variables
  - **Function names**

Level 2: Local variable name abstraction

```
1 void avg (float FPARAM[], int FPARAM) {
2     static float LVAR = 0;
3     unsigned int LVAR;
4
5     for (LVAR = 0; LVAR < FPARAM; LVAR ++) {
6         LVAR += FPARAM[LVAR];
7     }
8
9     printf("%f %d\n", LVAR/FPARAM, validate(LVAR));
10 }
```

Abstraction

- Transform function by replacing
  - Formal parameters
  - Data types
  - Local variables
  - **Function names**

Level 3: Data type abstraction

```
1 DTYPE avg (DTYPE FPARAM[], DTYPE FPARAM) {
2     DTYPE LVAR = 0;
3     unsigned DTYPE LVAR;
4
5     for (LVAR = 0; LVAR < FPARAM; LVAR ++) {
6         LVAR += FPARAM[LVAR];
7     }
8
9     printf("%f %d\n", LVAR/FPARAM, validate(LVAR));
10 }
```

Abstraction

- Transform function by replacing
  - Formal parameters
  - Data types
  - Local variables
  - **Function names**

Level 4: Function call abstraction

```
1 DTYPE avg (DTYPE FPARAM[], DTYPE FPARAM) {
2     DTYPE LVAR = 0;
3     unsigned DTYPE LVAR;
4
5     for (LVAR = 0; LVAR < FPARAM; LVAR ++) {
6         LVAR += FPARAM[LVAR];
7     }
8
9     FUNCCALL("%f %d\n", LVAR/FPARAM, FUNCCALL(LVAR));
10 }
```

Normalization

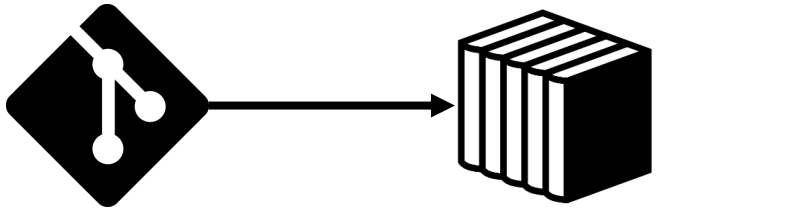
- Remove
  - comments
  - tabs
  - white spaces
  - CRLF
- Convert into lowercase

```
1 DTYPE avg (DTYPE FPARAM[], DTYPE FPARAM) {
2   DTYPE LVAR = 0;
3   unsigned DTYPE LVAR;
4
5   for (LVAR = 0; LVAR < FPARAM; LVAR ++) {
6      LVAR += FPARAM[LVAR];
7   }
8
9   FUNCCALL("%f %d\n", LVAR/FPARAM, FUNCCALL(LVAR));
10 }
```

```
dtypelvar=0;unsigneddtypelvar;for(lvar=0;lvar<fparam;lvar++){lvar+=fparam[lvar];}funccall("%f
%d\n",lvar/fparam,funccall(lvar));
```

Vulnerable code clone detection

- By comparing two fingerprint dictionaries

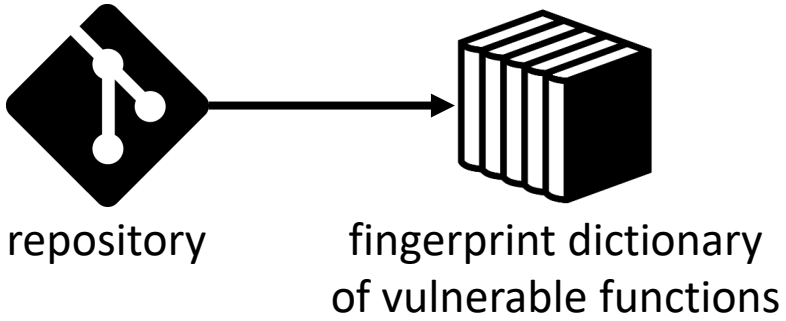


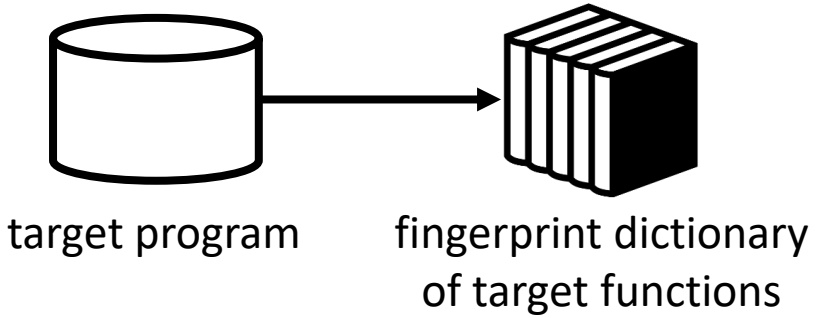
repository

fingerprint dictionary
of vulnerable functions

Vulnerable code clone detection

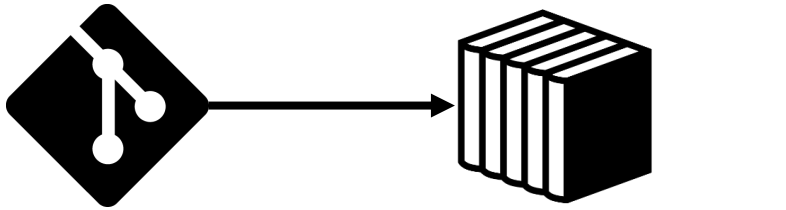
- By comparing two fingerprint dictionaries





Vulnerable code clone detection

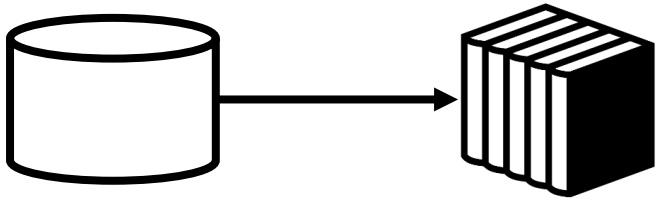
- By comparing two fingerprint dictionaries



repository

fingerprint dictionary
of vulnerable functions

```
20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]
```



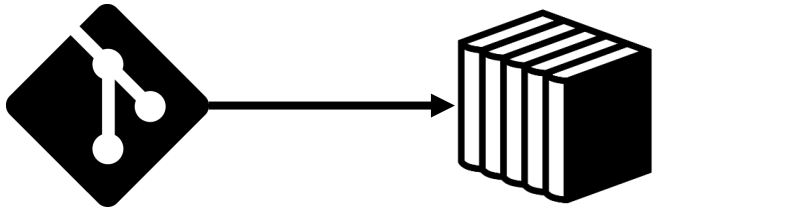
target program

fingerprint dictionary
of target functions

```
20: [C94D9910, D6E77882]
23: [9A45E4A1]
```

Vulnerable code clone detection

- By comparing two fingerprint dictionaries

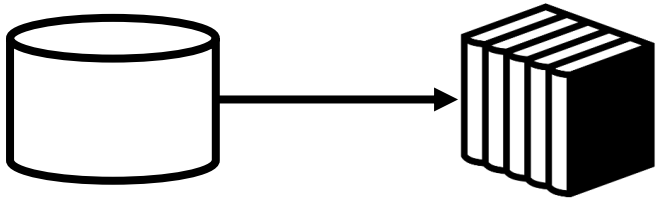


repository

fingerprint dictionary
of vulnerable functions

```
20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]
```

key_lookup(**20**) hit



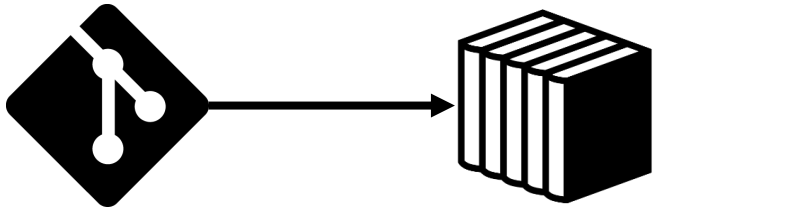
target program

fingerprint dictionary
of target functions

```
20: [C94D9910, D6E77882]
23: [9A45E4A1]
```

Vulnerable code clone detection

- By comparing two fingerprint dictionaries

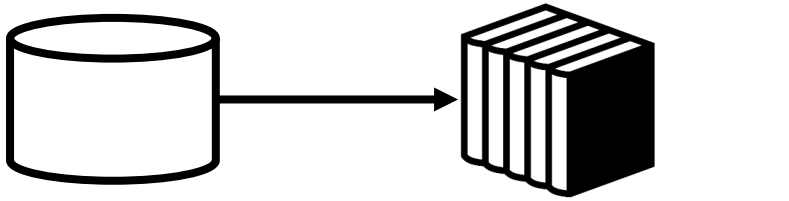


repository

fingerprint dictionary
of vulnerable functions

```
20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]
```

key_lookup(20) hit → have **C94D9910** in common (CLONE!)



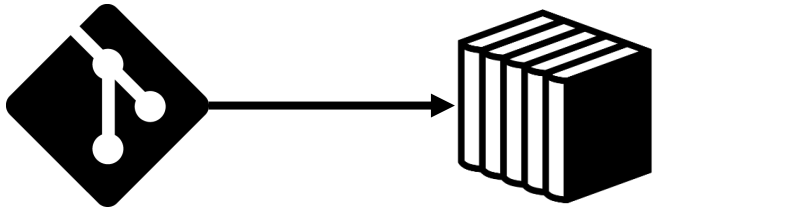
target program

fingerprint dictionary
of target functions

```
20: [C94D9910, D6E77882]
23: [9A45E4A1]
```

Vulnerable code clone detection

- By comparing two fingerprint dictionaries

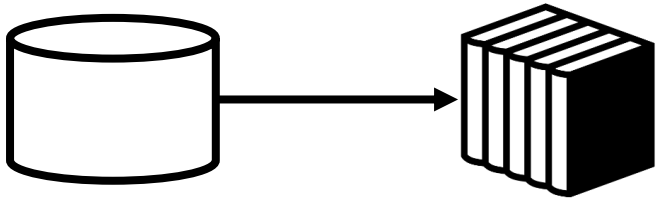


repository

fingerprint dictionary
of vulnerable functions

```
20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]
```

key_lookup(21) fail



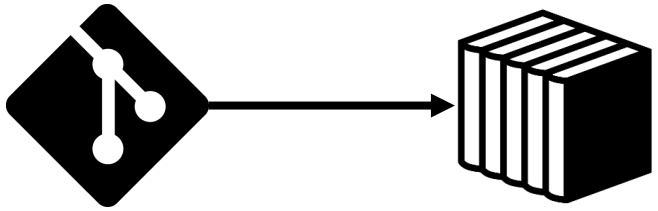
target program

fingerprint dictionary
of target functions

```
20: [C94D9910, D6E77882]
23: [9A45E4A1]
```

Vulnerable code clone detection

- By comparing two fingerprint dictionaries

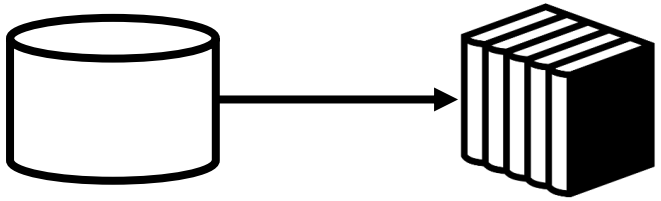


repository

fingerprint dictionary
of vulnerable functions

```
20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]
```

key_lookup(22) fail



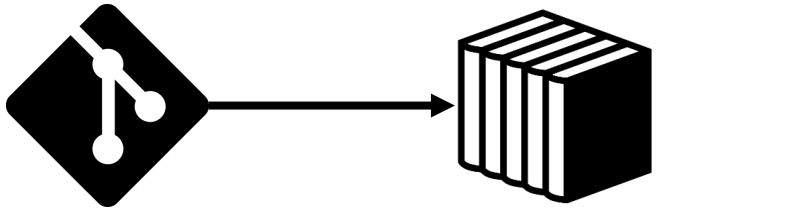
target program

fingerprint dictionary
of target functions

```
20: [C94D9910, D6E77882]
23: [9A45E4A1]
```

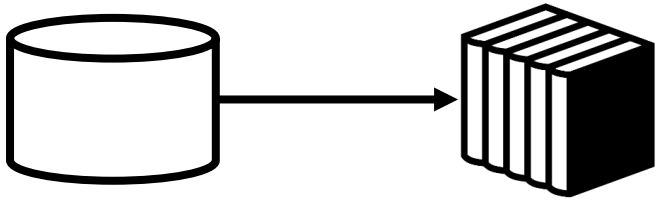
Vulnerable code clone detection

- By comparing two fingerprint dictionaries



repository

fingerprint dictionary
of vulnerable functions



target program

fingerprint dictionary
of target functions

```
20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]
```

clone: C94D9910

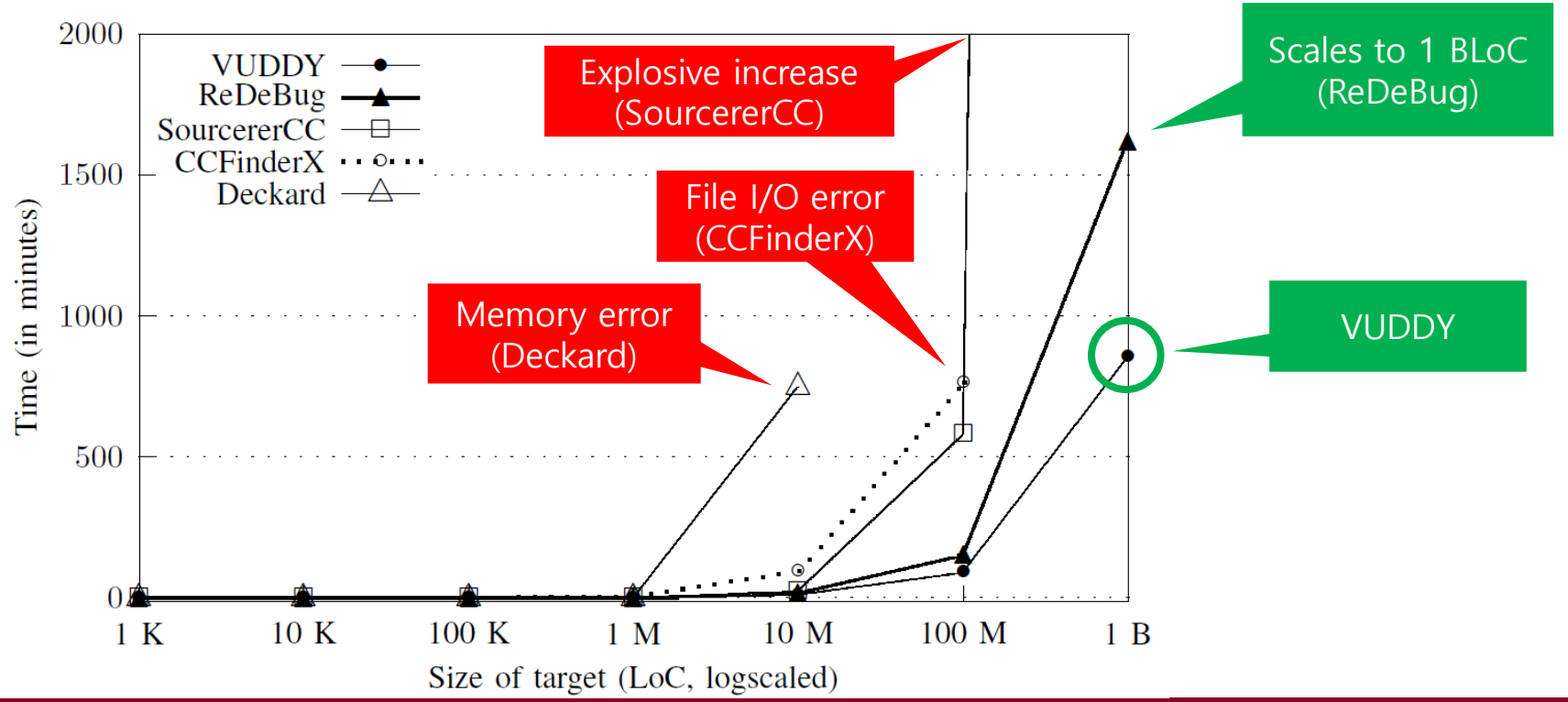
```
int sum (int a, int b)
{
    return a + b;
}
```

```
20: [C94D9910, D6E77882]
23: [9A45E4A1]
```

Performance Evaluation
&
Case Study

Performance

- Scalability evaluation
  - Dataset: 25 K GitHub projects (>1 push, >1 star during Jan 1~July 28, 2016)
  - Execution time when varying size of target programs are given to VUDDY, CCFinderX, DECKARD, ReDeBug, and SourcererCC



Performance

- Accuracy evaluation
  - Vulnerability database VS Apache HTTPD 2.4.23 (350 KLoC)
    - TP: CCFinderX > VUDDY > DECKARD > SourcererCC (the greater, the better)
    - FP: VUDDY < SourcererCC < CCFinderX < DECKARD (the lower, the better)

	Time	TP	FP	FN	Precision
VUDDY	22 s	9	0	3	1.000
SourcererCC	125 s	2	54	10	0.036
DECKARD	234 s	4	458	8	0.009
CCFinderX	1201 s	11	63	1	0.147

TABLE I: Accuracy of VUDDY, SourcererCC, DECKARD, and CCFinderX
when detecting clones between the vulnerability database and Apache HTTPD 2.4.23

Performance

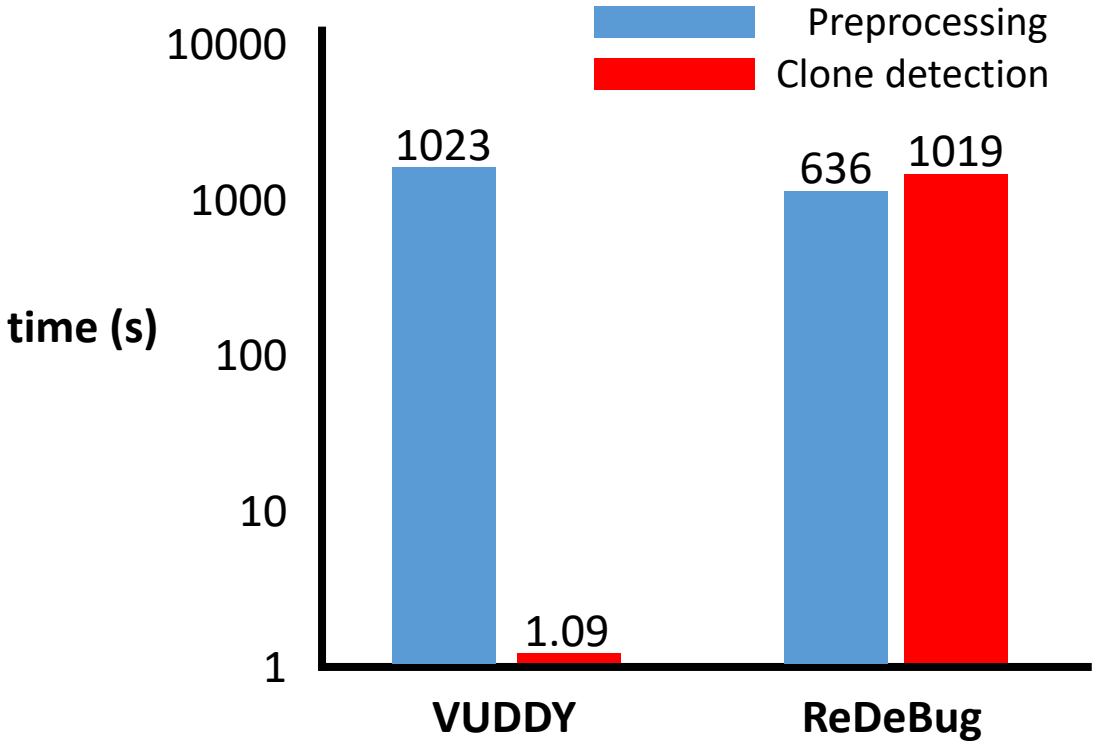
- VUDDY vs ReDeBug (CMU, S&P'12)
  - Detecting vulnerable code clones in an Android smartphone's firmware (15 MLoC)

	VUDDY	ReDeBug
Preprocessing time	17 m 3 s	11 m 16 s
Clone detection time	**1.09 s**	**16 m 59 s**
# initial reports	206	2,090
# true positives	206	202
# false positives	**0**	**1,888**

TABLE II: Comparison of VUDDY and ReDeBug, targeting Android firmware

Performance

- VUDDY vs ReDeBug (CMU, S&P'12)
  - Detecting vulnerable code clones in an Android smartphone's firmware (15 MLoC)



Generated fingerprints can be reused

Actual detection in practice: 1000x faster

Case study

- Unknown vulnerability detected in Linux kernel (even in 4.11.1)

Original patch for CVE-2008-3528 targeting ext2 file system

```
1    struct ext2_dir_entry_2 * ext2_dotdot (struct inode *dir, struct page **p)
2    {
3 -    struct page *page = ext2_get_page(dir, 0);
4 +    struct page *page = ext2_get_page(dir, 0, 0);
5      ext2_dirent *de = NULL;
6
7      if (!IS_ERR(page)) {
```

Could trigger "printk flood" & DoS in CentOS 7, and Ubuntu14.04

Patched function in ext2 file system

```
1  struct ext2_dir_entry *ext2_dotdot (struct
                    inode * dir, struct page **p)
2    {
3      struct page *page = ext2_get_page(dir, 0, 0);
4      struct ext2_dir_entry *de = NULL;
5
6      if (!IS_ERR(page)) {
```

Vulnerable function in nilfs2 file system

```
1  struct nilfs_dir_entry *nilfs_dotdot (struct
                    inode * dir, struct page **p)
2    {
3      struct page *page = nilfs_get_page(dir, 0);
4      struct nilfs_dir_entry *de = NULL;
5
6      if (!IS_ERR(page)) {
```

Case study

- Zero-day in Apache HTTPD 2.4.23 (2.4.20 through 2.4.25)
  - HTTPD uses unpatched Expat library for parsing XML
    - vulnerable to CVE-2012-0876
  - Hash DoS attack triggered by sending a crafted packet!

```
// Vulnerable function in httpd/srclib/apr-util/xml/expat/lib/xmlparse.c, lines 5429-5433.
for (i = 0; i < table->size; i++){
  if (table->v[i]) {
    unsigned long newHash = hash(table->v[i]->name);
    size_t j = newHash & newMask;
    step = 0;
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4730	daemon	20	0	435504	8968	2952	S	100.1	0.2	0:04.92	httpd
634	root	20	0	191960	10648	9444	S	0.3	0.3	0:02.54	vmtoolsd
1442	unused	20	0	1571620	114444	68224	S	0.3	2.8	0:26.20	compiz
1	root	20	0	119676	5800	3944	S	0.0	0.1	0:01.86	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.03	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H

Summary

Summary

- VUDDY is an approach capable of detecting **software vulnerability** using a database of previously security-patched functions

Summary

- VUDDY is an approach capable of detecting **software vulnerability** using a database of previously security-patched functions
- Applying **abstraction** to the functions enable identifying **unknown vulnerable functions** while still maintaining a low margin of errors

Summary

- VUDDY is an approach capable of detecting **software vulnerability** using a database of previously security-patched functions
- Applying **abstraction** to the functions enable identifying **unknown vulnerable functions** while still maintaining a low margin of errors
- Function-level granularity and length-based filtering reduces the number of signature comparisons, guaranteeing **high scalability**

Summary

- VUDDY is an approach capable of detecting software vulnerability using a database of previously security-patched functions
- Applying abstraction to the functions enable identifying unknown vulnerable functions while still maintaining a low margin of errors
- Function-level granularity and length-based filtering reduces the number of signature comparisons, guaranteeing high scalability
- Open web service
  - Implementation and testing available at https://iotcube.net