Korea University
College of Informatics

Center for Software Security and Assurance

CCSLAB Computer & Communication Security Laboratory

MOVERY: A Precise Approach for Modified Vulnerable Code Clone Discovery from Modified Open-Source Software Components

Seunghoon Woo, Hyunji Hong, Eunjin Choi, Heejo Lee

Korea University

USENIX security 2022

Motivation

Modified open-source software reuse is prevalent

- Reuse of open-source software (OSS) becomes a trend in software development
- Unmanaged OSS reuse can pose security threats (e.g., vulnerability propagation)
- Most developers reuse OSS projects with code/structural modifications*

🙁 Hard to discover propagated vulnerable codes with code changes

How can we precisely discover propagated vulnerable codes with various syntaxes?

* [CCS 2017] "Identifying Open-Source License Violation and 1-day Security Risk at Large Scale", Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee
* [ICSE 2021] "CENTRIS: A Precise and Scalable Approach for Identifying Modified Open-Source Software Reuse", Seunghoon Woo, Sunghan Park, Seulbae Kim, Heejo Lee, and Hakjoo Oh

Motivation

Addressing syntax diversity of vulnerable code

- Syntax diversity of vulnerable code
  - Internal modification of OSS
    - ❖ OSS source code frequently changes during **OSS updates**
    - ❖ Vulnerable code may exist in various syntax depending on the reused OSS version
  - External modification of OSS
    - ❖ Vulnerable code can be modified during the **OSS reuse process**

Motivation

Example: Syntax diversity caused by the internal OSS modification

- CVE-2014-5461 vulnerability in Lua (DoS vulnerability)
  - This vulnerable code existed in Redis (using Lua)
  - The syntax of the two vulnerable functions is quite different

```
int luaD_precall (lua_State *L, StkId func, int nresults) {
  lua_CFunction f;
  CallInfo *ci;
  int n;  /* number of arguments (Lua) or returns (C) */
  ptrdiff_t funcr = savestack(L, func);
  switch (ttype(func)) {
    …
    case LUA_TLCL: {  /* Lua function: prepare its call */
      StkId base;
      Proto *p = clLvalue(func)->p;
    - luaD_checkstack(L, p->maxstacksize);
    - func = restorestack(L, funcr);
      n = cast_int(L->top - func) - 1;
    + luaD_checkstack(L, p->maxstacksize);
```

A patch snippet for CVE-2014-5461 in Lua 5.2.3

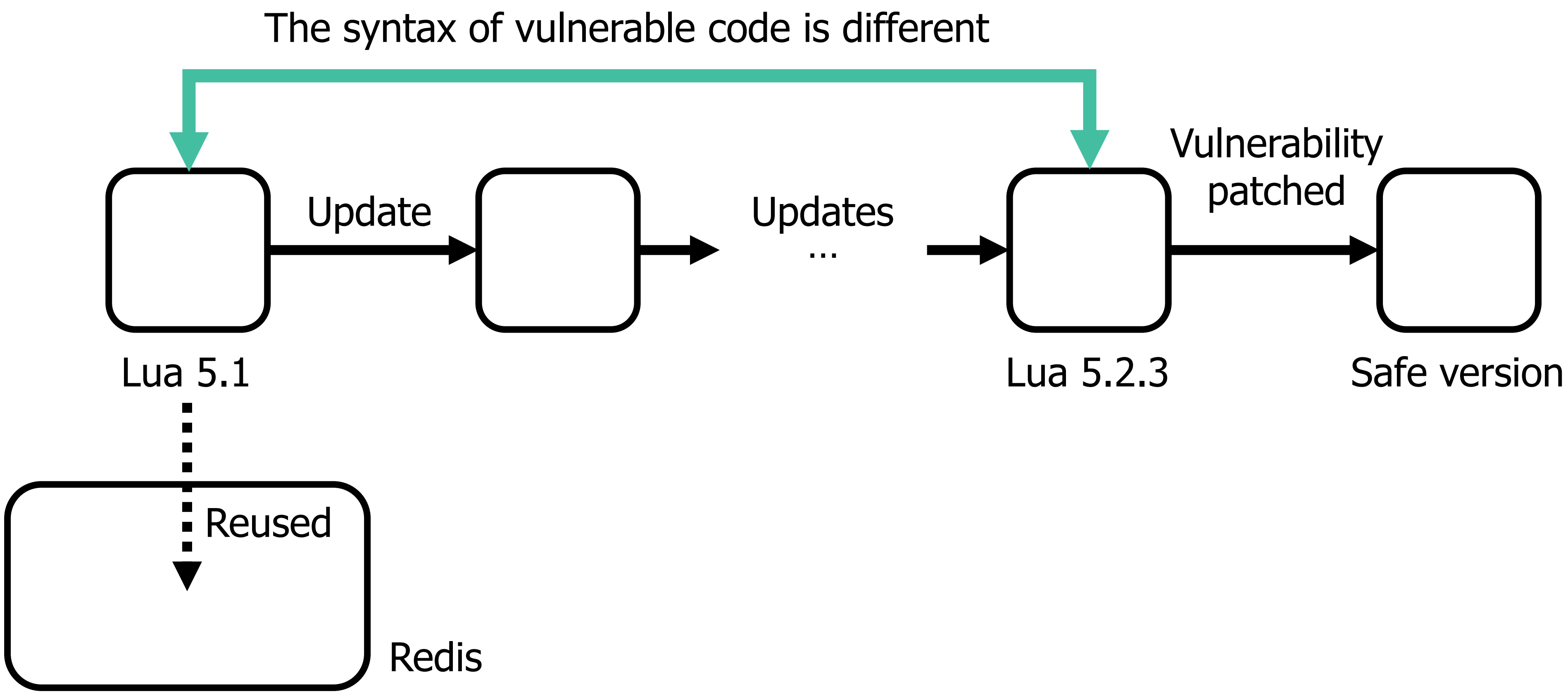
```
int luaD_precall (lua_State *L, StkId func, int nresults) {
  LClosure *cl;
  ptrdiff_t funcr;
  if (!ttisfunction(func)) /* `func' is not a function? */
    func = tryfuncTM(L, func);  /* check the `function' tag method */
  funcr = savestack(L, func);
  cl = &clvalue(func)->l;
  L->ci->savedpc = L->savedpc;
  if (!cl->isC) {  /* Lua function? prepare its call */
    CallInfo *ci;
    StkId st, base;
    Proto *p = cl->p;
  - luaD_checkstack(L, p->maxstacksize);
  + luaD_checkstack(L, p->maxstacksize + p->numparams);
```

A patch snippet for CVE-2014-5461 in Redis (using Lua 5.1)

Motivation

Example: Syntax diversity caused by the internal OSS modification

- CVE-2014-5461 vulnerability in Lua (DoS vulnerability)



Motivation

Example: Syntax diversity caused by the internal OSS modification

- Existing approaches fail to detect this propagated vulnerable code
  - ReDeBug [S&P 2012] ✕
    - ❖ Considering nearby three (by default) lines of deleted and added code lines from the patch

```
int luaD_precall (lua_State *L, StkId func, int nresults) {
  lua_CFunction f;
  CallInfo *ci;
  int n;  /* number of arguments (Lua) or returns (C) */
  ptrdiff_t funcr = savestack(L, func);
  switch (ttype(func)) {
    …
    case LUA_TLCL: {  /* Lua function: prepare its call */
      StkId base;
      Proto *p = clLvalue(func)->p;
    - luaD_checkstack(L, p->maxstacksize);
    - func = restorestack(L, funcr);
      n = cast_int(L->top - func) - 1;
    + luaD_checkstack(L, p->maxstacksize);
```

```
int luaD_precall (lua_State *L, StkId func, int nresults) {
  LClosure *cl;
  ptrdiff_t funcr;
  if (!ttisfunction(func)) /* `func' is not a function? */
    func = tryfuncTM(L, func);  /* check the `function' tag method */
  funcr = savestack(L, func);
  cl = &clvalue(func)->l;
  L->ci->savedpc = L->savedpc;
  if (!cl->isC) {  /* Lua function? prepare its call */
    CallInfo *ci;
    StkId st, base;
    Proto *p = cl->p;
  - luaD_checkstack(L, p->maxstacksize);
  + luaD_checkstack(L, p->maxstacksize + p->numparams);
```

DIFFERENT

Motivation

Example: Syntax diversity caused by the internal OSS modification

- Existing approaches fail to detect this propagated vulnerable code
  - VUDDY [S&P 2017] ✗
    - ❖ Considering a whole vulnerable function

```
int luaD_precall (lua_State *L, StkId func, int nresults) {
  lua_CFunction f;
  CallInfo *ci;
  int n;  /* number of arguments (Lua) or returns (C) */
  ptrdiff_t funcr = savestack(L, func);
  switch (ttype(func)) {
    …
    case LUA_TLCL: {  /* Lua function: prepare its call */
      StkId base;
      Proto *p = clLvalue(func)->p;
    - luaD_checkstack(L, p->maxstacksize);
    - func = restorestack(L, funcr);
      n = cast_int(L->top - func) - 1;
    + luaD_checkstack(L, p->maxstacksize);
```

```
int luaD_precall (lua_State *L, StkId func, int nresults) {
  LClosure *cl;
  ptrdiff_t funcr;
  if (!ttisfunction(func)) /* `func' is not a function? */
    func = tryfuncTM(L, func);  /* check the `function' tag method */
  funcr = savestack(L, func);
  cl = &clvalue(func)->l;
  L->ci->savedpc = L->savedpc;
  if (!cl->isC) {  /* Lua function? prepare its call */
    CallInfo *ci;
    StkId st, base;
    Proto *p = cl->p;
  - luaD_checkstack(L, p->maxstacksize);
  + luaD_checkstack(L, p->maxstacksize + p->numparams);
```

DIFFERENT

MOVERY: A Precise Approach for Modified Vulnerable Code Clone Discovery from Modified Open-Source Software Components

Design of MOVERY

MOdified Vulnerable code clone discovERY

- A novel approach to precisely detect modified vulnerable code clones

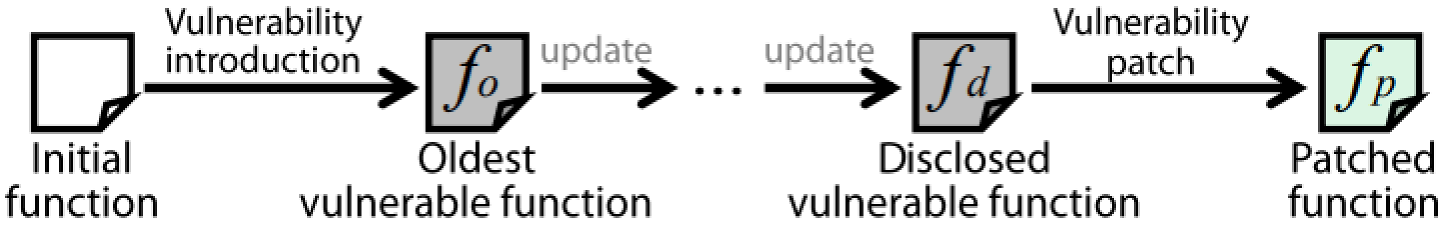
- Key techniques

  (1) Function collation

  (2) Core line extraction

  ❖ For addressing internal/external modifications of OSS

- Notations



Phase (1) Signature generation

Working example: Heap-buffer overflow vulnerability (CVE-2016-8654) in Jasper

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
    if (!(buf = jas_alloc2(bufsize, sizeof(jpc_fix_t)))) {
       abort();
    }
 }
 if (numrows >= 2) {
  - hstartcol = (numrows + 1 - parity) > 1;
  - // ORIGINAL (WRONG): m = (parity) ? hstartcol : (numrows - hstartcol);
  - m = numrows - hstartcol;
  + hstartrow = (numrows + 1 - parity) > 1;
  + // ORIGINAL (WRONG): m = (parity) ? hstartrow : (numrows - hstartrow);
  + m = numrows - hstartrow;
    n = m;
    dstptr = buf;
    srcptr = &a[(1 - parity) * stride]
    ...
```

A patch snippet for CVE-2016-8654 in Jasper

`hstartcol -> hstartrow`

Phase (1) Signature generation

Working example

- Function collation

f_o Oldest vulnerable function

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
    if (!(buf = jas_alloc(bufsize * sizeof(jpc_fix_t)))) {
       abort();
    }
 }
 if (numrows >= 2) {
    hstartcol = (numrows + 1 - parity) > 1;
    m = (parity) ? hstartcol : (numrows - hstartcol);


    n = m;
    dstptr = buf;
    srcptr = &a[(1 - parity) * stride]
    ...
```

f_d Disclosed vulnerable function

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
    if (!(buf = jas_alloc2(bufsize, sizeof(jpc_fix_t)))) {
       abort();
    }
 }
 if (numrows >= 2) {
  - hstartcol = (numrows + 1 - parity) > 1;
  - // ORIGINAL (WRONG): m = (parity) ?
                         hstartcol : (numrows - hstartcol);
  - m = numrows - hstartcol;
    n = m;
    dstptr = buf;
    srcptr = &a[(1 - parity) * stride]
    ...
```

- Highlighted areas indicate the code parts that differ from the disclosed vulnerable function

Phase (1) Signature generation

Definition of core lines

- Core lines in vulnerability signature generation

1. **Essential code lines**

   ❖ Code lines that were deleted from the patch and exist in both fo and fd

2. **Dependent code lines**

   ❖ Code lines that have control/data dependencies with the essential code lines

3. **Control flow code lines**

   ❖ Control statements that exist in both fo and fd

Phase (1) Signature generation

Working example

1) Extracting essential code lines (Ev)

- Code lines that were deleted from the patch (existing in both fo and fd)
- Essential code lines are closely related to the vulnerability manifestation

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
    if (!(buf = jas_alloc(bufsize * sizeof(jpc_fix_t)))) {
       abort();
    }
 }
 if (numrows >= 2) {
    hstartcol = (numrows + 1 - parity) > 1;
    m = (parity) ? hstartcol : (numrows - hstartcol);


    n = m;
    dstptr = buf;
    srcptr = &a[(1 - parity) * stride]
    ...
```

Oldest vulnerable function

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
    if (!(buf = jas_alloc2(bufsize, sizeof(jpc_fix_t)))) {
       abort();
    }
 }
 if (numrows >= 2) {
  - hstartcol = (numrows + 1 - parity) > 1;
  - // ORIGINAL (WRONG): m = (parity) ?
                            hstartcol : (numrows - hstartcol);
  - m = numrows - hstartcol;
    n = m;
    dstptr = buf;
    srcptr = &a[(1 - parity) * stride]
    ...
```

Disclosed vulnerable function

Phase (1) Signature generation

Working example

2) Extracting dependent code lines (Dv):

- Code lines that have control/data dependency with the essential code lines
- To determine whether the vulnerability trigger environment has propagated

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
    if (!(buf = jas_alloc(bufsize * sizeof(jpc_fix_t)))) {
       abort();
    }
 }
 if (numrows >= 2) {
    hstartcol = (numrows + 1 - parity) > 1;
    m = (parity) ? hstartcol : (numrows - hstartcol);

    n = m;
    dstptr = buf;
    srcptr = &a[(1 - parity) * stride]
    ...
```

Oldest vulnerable function

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
    if (!(buf = jas_alloc2(bufsize, sizeof(jpc_fix_t)))) {
       abort();
    }
 }
 if (numrows >= 2) {
    hstartcol = (numrows + 1 - parity) > 1;
    // ORIGINAL (WRONG): m = (parity) ?
                           hstartcol : (numrows - hstartcol);
    m = numrows - hstartcol;
    n = m;
    dstptr = buf;
    srcptr = &a[(1 - parity) * stride]
    ...
```

Disclosed vulnerable function

Phase (1) Signature generation

Working example

3) Extracting control flow code lines (Fv)

- To determine whether the essential code line has still reachable with the same conditions

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
     if (!(buf = jas_alloc(bufsize * sizeof(jpc_fix_t)))) {
        abort();
     }
 }
 if (numrows >= 2) {
     hstartcol = (numrows + 1 - parity) > 1;
     m = (parity) ? hstartcol : (numrows - hstartcol);

     n = m;
     dstptr = buf;
     srcptr = &a[(1 - parity) * stride]
     ...
```

Oldest vulnerable function

```
void jpc_qmfb_split_col (...) {
...
 if (bufsize > QMFB_SPLITBUFSIZE) {
     if (!(buf = jas_alloc2(bufsize, sizeof(jpc_fix_t)))) {
        abort();
     }
 }
 if (numrows >= 2) {
     hstartcol = (numrows + 1 - parity) > 1;
     // ORIGINAL (WRONG): m = (parity) ?
                              hstartcol : (numrows - hstartcol);
     m = numrows - hstartcol;
     n = m;
     dstptr = buf;
     srcptr = &a[(1 - parity) * stride]
     ...
```

Disclosed vulnerable function

Phase (1) Signature generation

Gathering code lines and generating signatures

4) Generating signatures

- Vulnerability signature (Sv)
- Patch signature (Sp)
  - ❖ An approach similar to generating a vulnerability signature is performed (deleted -> added)
  - ❖ Control flow lines (Fp) that exist only in the patch function are already included in Ep

$$S_v = (E_v, D_v, F_v)$$

$$S_p = (E_p, D_p)$$

Phase (2) Vulnerable code clone discovery

Detecting vulnerable code clones in the target program (T)

- A function f in T is a vulnerable code clone if it satisfies:

- ***Cond 1)*** f should contain all code lines in S_v.

$$\forall_{l \in S_v}.(l \in f)$$

- ***Cond 2)*** f should not contain any code lines in S_p.

$$\forall_{l \in S_p}.(l \notin f)$$

- ***Cond 3)*** The syntax of f should be similar to f_o or f_d.

$$(\mathtt{Sim}(f, f_o) \geq \theta) \vee (\mathtt{Sim}(f, f_d) \geq \theta)$$

Phase (2) Vulnerable code clone discovery

Detecting vulnerable code clones in the target program (T)

- A function f in T is a vulnerable code clone if it satisfies:

- ***Cond 1)*** f should contain all code lines in S_v.

$$\forall_{l \in S_v}.(l \in f)$$

- ***Cond 2)*** f should not contain any code lines in S_p.

$$\forall_{l \in S_p}.(l \notin f)$$

- ***Cond 3)*** The syntax of f should be similar to f_o or f_d.

$$(\mathtt{Sim}(f, f_o) \geq \theta) \vee (\mathtt{Sim}(f, f_d) \geq \theta)$$

* Using the Jaccard index by considering the function as a set of code lines

EVALUATION

Dataset and parameter setting

- CVE dataset
  - 4,219 C/C++ CVE vulnerabilities (patches)
    - ❖ Collected from NVD
    - ❖ 7,762 vulnerable/patched function pairs
    - ❖ 5,936 oldest vulnerable functions

Target program overview

IDX	Name	Version	#Line*	#Comp†	Domain
T1	FreeBSD	v12.2.0	14,489,534	47	Operating system
T2	ReactOS	v0.4.13	6,419,855	23	Operating system
T3	ArangoDB	v3.7.9	3,064,973	22	Database
T4	FFmpeg	n4.3.2	1,230,520	4	Multimedia processing
T5	OpenCV	v4.5.1	1,092,317	15	Computer vision
T6	Emscripten	v2.0.15	759,020	11	Compiler
T7	Crown	v0.42.0	723,372	20	Game engine
T8	Git	v2.31.0	293,467	5	Version control system
T9	OpenMVG	v1.6	262,610	8	Image processing
T10	Redis	v5.0.12	212,672	8	Database
Total	-	-	**28,548,340**	**190**	-

*: Counting only C/C++ code lines, †: The number of modified OSS components.

- Target programs
  - 10 software programs that are popular (based on GitHub stars) and contain a sufficient number of OSS components
- Parameter
  - $\theta = 0.5$

EVALUATION

Accuracy measurement

- Comparison targets
  - Two existing vulnerable code clone detection tools: VUDDY [S&P 2017] and ReDeBug [S&P 2012]
  - MOVERY significantly outperformed existing approaches

Target software	#Discovered VCCs*	ReDeBug					VUDDY					MOVERY				
		#TP	#FP	#FN	Precision	Recall	#TP	#FP	#FN	Precision	Recall	#TP	#FP	#FN	Precision	Recall
ReactOS	210	31	9	179	0.78	0.15	8	0	202	1.00	0.04	207	3	3	0.99	0.99
OpenCV	72	38	15	34	0.72	0.53	26	2	46	0.93	0.36	72	3	0	0.96	1.00
Emscripten	56	22	8	34	0.73	0.39	9	1	47	0.90	0.16	50	4	6	0.93	0.89
FreeBSD	33	25	44	8	0.36	0.76	6	16	27	0.27	0.18	27	4	6	0.87	0.82
Crown	23	22	2	1	0.92	0.96	14	2	9	0.88	0.61	23	2	0	0.92	1.00
OpenMVG	23	15	5	8	0.75	0.65	4	0	19	1.00	0.17	19	0	4	1.00	0.83
ArangoDB	6	4	1	2	0.80	0.67	2	0	4	1.00	0.33	6	2	0	0.75	1.00
FFmpeg	5	2	2	3	0.50	0.40	0	1	5	0.00	0.00	5	1	0	0.83	1.00
Redis	5	3	0	2	1.00	0.60	3	0	2	1.00	0.60	5	0	0	1.00	1.00
Git	1	1	1	0	0.50	1.00	0	0	1	N/A	0.00	1	0	0	1.00	1.00
Total	434	163	87	271	0.65	0.38	72	22	362	0.77	0.17	**415**	**19**	**19**	**0.96**	**0.96**

*VCCs: Vulnerable Code Clones

EVALUATION

Accuracy measurement

- Comparison targets
  - Two existing vulnerable code clone detection tools: VUDDY [S&P 2017] and ReDeBug [S&P 2012]
  - MOVERY significantly outperformed existing approaches

Target software	#Discovered VCCs*	ReDeBug					VUDDY					MOVERY				
		#TP	#FP	#FN	Precision	Recall	#TP	#FP	#FN	Precision	Recall	#TP	#FP	#FN	Precision	Recall
ReactOS	210	31	9	179	0.78	0.15	8	0	202	1.00	0.04	207	3	3	0.99	0.99
OpenCV	72	38	15	34	0.72	0.53	26	2	46	0.93	0.36	72	3	0	0.96	1.00
Emscripten	56	22	8	34	0.73	0.39	9	1	47	0.90	0.16	50	4	6	0.93	0.89
FreeBSD	33	25	44	8	0.36	0.76	6	16	27	0.27	0.18	27	4	6	0.87	0.82
Crown	23	22	2	1	0.92	0.96	14	2	9	0.88	0.61	23	2	0	0.92	1.00
OpenMVG	23	15	5	8	0.75	0.65	4	0	19	1.00	0.17	19	0	4	1.00	0.83
ArangoDB	6	4	1	2	0.80	0.67	2	0	4	1.00	0.33	6	2	0	0.75	1.00
FFmpeg	5	2	2	3	0.50	0.40	0	1	5	0.00	0.00	5	1	0	0.83	1.00
Redis	5	3	0	2	1.00	0.60	3	0	2	1.00	0.60	5	0	0	1.00	1.00
Git	1	1	1	0	0.50	1.00	0	0	1	N/A	0.00	1	0	0	1.00	1.00
Total	434	163	87	271	0.65	0.38	72	22	362	0.77	0.17	**415**	**19**	**19**	**0.96**	**0.96**

*VCCs: Vulnerable Code Clones

MOVERY could discover 2.5x and 5.8x more vulnerable codes than ReDeBug and VUDDY

EVALUATION

Accuracy measurement

- MOVERY could discover more VCCs than VUDDY and ReDeBug

VCCs that are hardly discovered by existing techniques

Types	**Description**	**#Discovered VCCs**
T1	VCCs without code lines deleted in security patches.	**32**
T2	VCCs with various syntaxes derived from the oldest vulnerable function (fo).	**221** (221 VCCs closer to fo than fd)
T3	VCCs with heavy syntax change.	**166** (166 VCCS: $\texttt{Sim}(f, f_d)$ ≤ 0.5)

Conclusion

Conclusion

- Many vulnerable codes are propagated with syntax modifications
  - 396 (91%) out of 434 VCCs existed in a different syntax to the disclosed vulnerable function
- MOVERY
  - A precise approach for discovering modified VCCs from modified components
  - MOVERY significantly outperformed existing approaches in vulnerable code clone discovery
    - ❖ High vulnerability discovery accuracy: 96% precision and 96% recall
- Equipped with VCC discovery results from MOVERY,
  - Developers can address threats caused by propagated vulnerabilities in modified components

Thank you for your attention!

- MOVERY repository (https://github.com/wooseunghoon/MOVERY-public)

CONTACT

- Seunghoon Woo (seunghoonwoo@korea.ac.kr, https://wooseunghoon.github.io)
- Computer & Communication Security Lab (https://ccs.korea.ac.kr)
- Center for Software Security and Assurance (https://cssa.korea.ac.kr)

APPENDIX I

Preprocessing

- Abstraction
  - Replacing every occurrence of parameters, variable names, variable types, and callee function names in each function with symbols PARAM, DNAME, DTYPE, and FCALL

```
if (bufsize > QMFB_SPLITBUFSIZE) {
if (numrows >= 2) {
    hstartcol = (numrows + 1 - parity) > 1;
    srcptr = &a[(1 - parity) * stride]
```

→

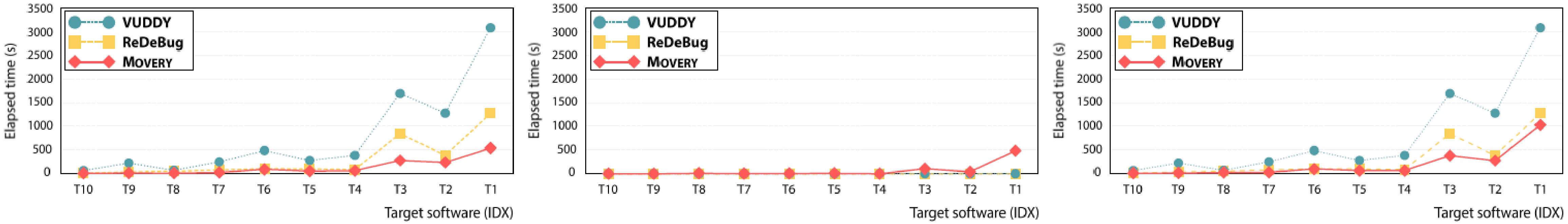
```
if (DVAL > QMFB_SPLITBUFSIZE) {
if (PARAM >= 2) {
    DVAL = (PARAM + 1 - PARAM) > 1;
    DVAL = &PARAM[(1 - PARAM) * PARAM]
```

- Selective abstraction
  - Abstraction is applied only when the abstraction code before and after the patch is <u>different</u>

APPENDIX II

Speed and scalability measurement

- MOVERY requires the least amount of time in the vulnerability discovery
- MOVERY discovers VCCs from the target programs varied from 213 K to 14.5 M LoC
  - The required time is not significantly increased



(a) Target preprocessing times. (b) Matching times. (c) Total times (preprocessing + matching).