Hook Placement

Divya Muthukumaran, Nirupama Talele, and Trent Jaeger, Vinod Ganapathy and Gang Tan

1. Contribution

An **algorithm** for ***auto-hooking***, which:

- Minimal authorization hook placement
- Satisfy authorization constraints
    - Constraints satisfy specific access control policies

2. Motivation

Manual hook placement is *tedious* and *incorrect*.
Large codebases need retroactive security features.
Need ***systematic*** techniques to *retrofit legacy code* for ***security***

Goal of Authorization Hook Placement: completely mediate all **security sensitive operations** on shared resources.

This leads to two subgoals:
→ identifies *security sensitive operation*
→ placement of *minimal and effective hooks*

3.1 Related Works

There are some former works in:

(1) Manual Hooking:

1.1 X11 ~ proposed 2003, upstreamed 2007, changing to date. [Kilpatrick et al., ‘03]

1.2 Linux Security Modules ~ 2 years [Wright et al., ’02]

(2) Verifying Hook Consistency

2.1 For Kernels [Zhang et al.,2002, Edwards et al., 2002, Tan et al., 2008]

2.2 For Web Applications [Sun et al., 2011, RoleCast 2011, FixMeUp 2012]

(3) Placing Hooks Automatically (People also conduct auto-hooking process, with input of Sensitive Data types and hook code)

3.1 Server Applications [Ganapathy et al., 2005,2006, 2007];

3.2 Related Works

There are some ***limitations*** in former works:

(1) Manual Hooking:

Loads of work for programmers

(2) Verifying Hook Consistency

To identifying security sensitive operations, have to specify code patterns or/and security-sensitive data structures manually → former work of Trent, **SSO** only using sources of ***untrusted inputs*** and ***language specific lookup functions***

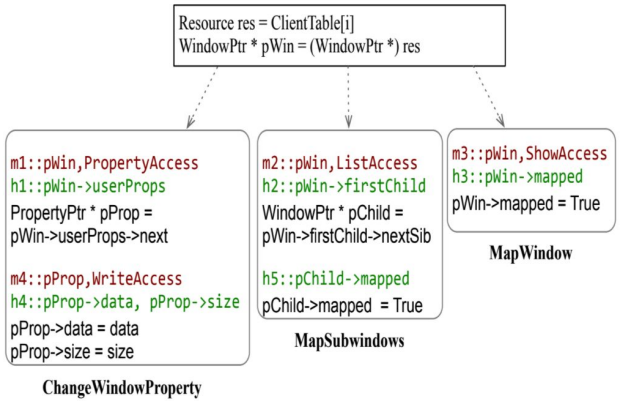
(3) Placing Hooks Automatically (People also conduct auto-hooking process, with input of Sensitive Data types and hook code)

→ use of low level representation of SSO, such as individual structure member accesses, result in hooks scattering across the whole program, which is hard to maintain and update

→ use of low level representation of SSO, leads to redundant authorization, and is not one to one mapping with placement of domain experts

4.1 Methodology: Authorization Hook Placement Problem

Two Main Problems: Find SSO ; Redundant Removal and Redundant Hoisting



Listing 1 Example of manual hoisting in the CopyGC function in the X server.

```
1   /*** gc.c ***/
2   int CopyGC(GC *pgcSrc, GC *pgcDst, BITS32 mask){
3    switch (index2)
4     {
5         result = dixLookupGC(&pGC, stuff->srcGC,
6                              client,DixGetAttrAccess);
7         if (result != Success)
8            return BadMatch;
9         case GCFunction:
10           /* Hook(pgcSrc, [read(GC->alu)]) */
11           pgcDst->alu = pgcSrc->alu;
12           break;
13        case GCPlaneMask:
14           /* Hook(pgcSrc, [read(GC->planemask)]) */
15           pgcDst->planemask = pgcSrc->planemask;
16           break;
17        case GCForeground:
18           /*  Hook(pgcSrc, [read(GC->fgPixel)]) */
19           pgcDst->fgPixel = pgcSrc->fgPixel;
20           break;
21        case GCBackground:
22           /* Hook(pgcSrc, [read(GC->bgPixel)]) */
23           pgcDst->bgPixel = pgcSrc->bgPixel;
24           break;
25          /* .... More similar cases */
26       }
27  }
```

4.2 Methodology: Authorization Hook Placement Problem

Two Main Problems:

- Granularity of SSO → balancing numbers of hooks placed and least privileges
- Redundant Removal and Redundant Hoisting

→ we need more control what can do and can't do, however not overkill

Idea in this paper: more policy specific auto-hooking strategy

4.3 Methodology: Authorization Hook Placement Problem

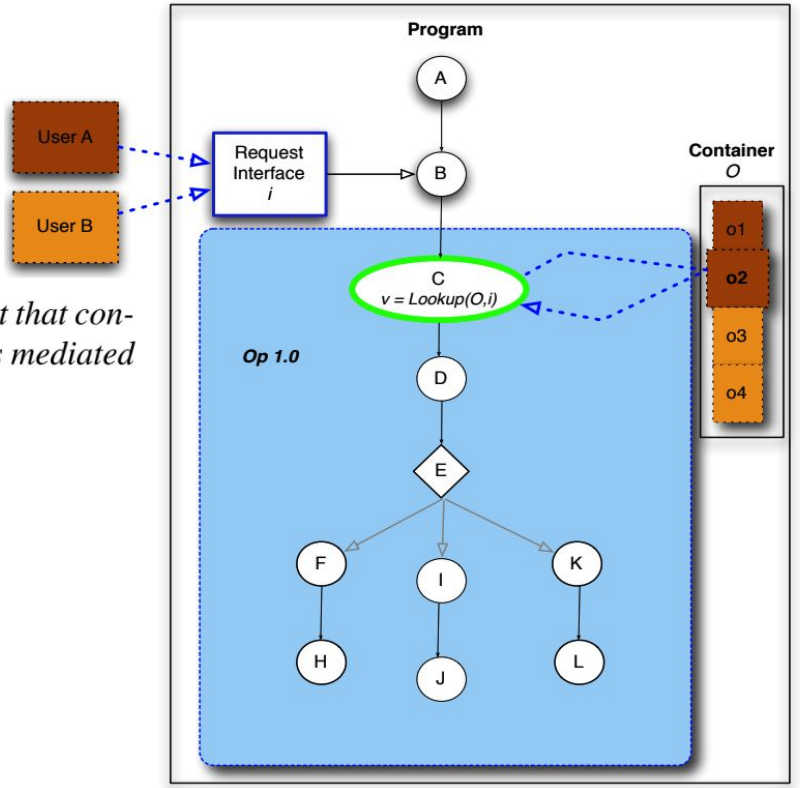
Static taint analysis
- Identify variables tainted by user request.
- Identify security-sensitive objects.

Control dependence analysis
- Identify security-sensitive operations.
- Hoist and remove redundant hooks.

4.3 Methodology: Static Taint Analysis

Static taint analysis

- Identify variables tainted by user request.
- Identify security-sensitive objects.



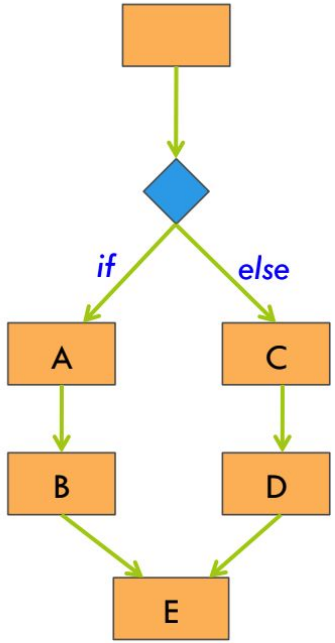
Definition 1 *An authorization hook is a tuple* (O_h, l_h) *where* l_h *is a statement that contains the hook and* $O_h \subseteq O_L$ *is a set of security-sensitive operation instances mediated by the hook.*

Stage 0: Get source code and user requests inputs(tainted)
Stage 1: identify **security sensitive objects**
Stage 2: identify **tainted** variables
Stage 3: identify **user choice operations**
Stage 4: identify authorization **hook placements**

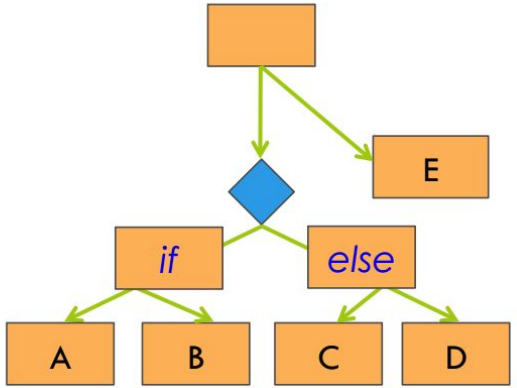
4.4 Methodology: CDG

CDG graph generating

→ dominance tree

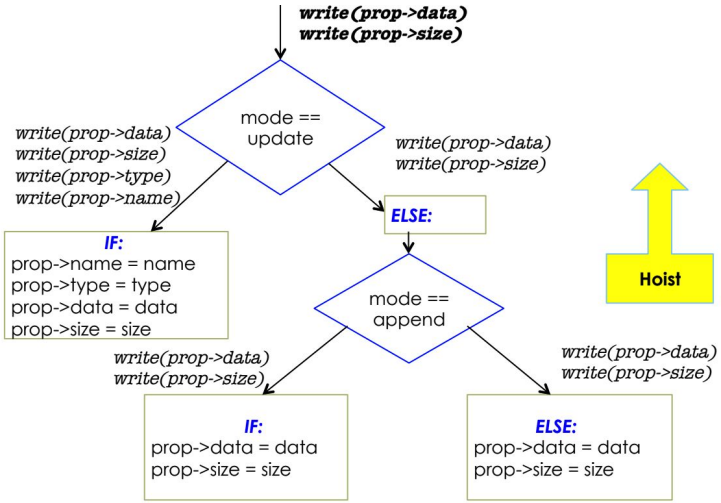


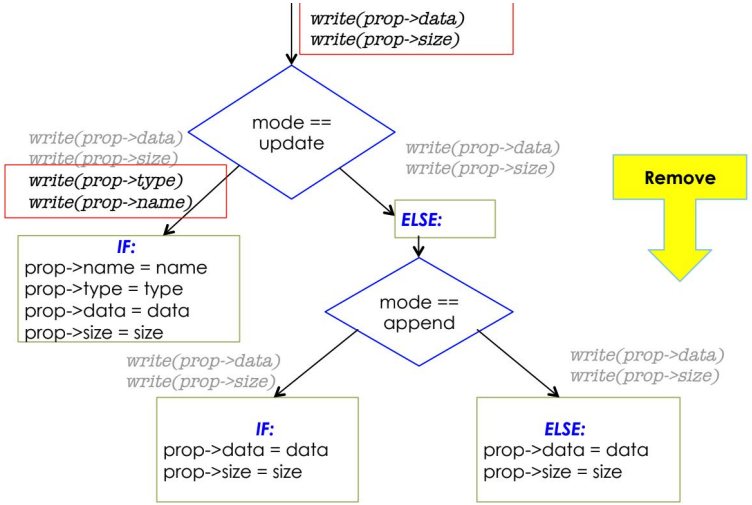
Control Flow Graph (CFG)



Control Dependence Graph (CDG)

4.5 Methodology: Hoisting and Removal





Constraints Selector:
Equivalence *of accesses to SSOs*

Definition 2 *A set of authorization constraints $\mathcal{P}$ is a pair (S, Q) of relationships between SSOs in the program, where Q stands for* **equivalence** *and S stands for* **subsumption**.

Constraints Selector:
E.g. MLS, if a subject read a field of a variable also permits read all fields of the variable

5. Results

a) Does the approach produce placements that are closer to manually placed hooks?

b) Does the approach reduce programmer effort necessary to place authorization hooks?

constraint selectors v.s. Trent former work

(1) Hook number reduced by 30%, as shown below

(2) Constraint selectors reduces the gap between manual and automated placements by 58%

(3) Constraint selectors reduces the programmer effort by 58%

Program	DEFAULT		MLS	
	REMOVE	*HOIST*	*REMOVE*	*HOIST*
X Server 1.13	237	55	113	10
Postgres 9.1.9	208	42	146	21
Linux VFS 2.6.38.8	53	4	49	3
Memcached	8	1	6	0

Table 2: Table showing the hoisting (HOIST) and removal (REMOVE) suggestions in the default placement (DEFAULT) and placements generated using the constraint selectors (MLS).

6. Take Away

Pros:

User friendly:

- The programmers only have to specify high-level security goals.
- Static and Dynamic Analysis helps programmers with constraint selection.

Cons:

This paper does not consider **alias analysis, polymorphic type or path sensitive analysis**

Thanks!

Q & A