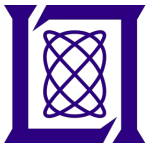
On the Effectiveness of Type-based Control Flow Integrity

Reza Mirzazade farkhani, Saman Jafari, Sajjad Arshad,
William Robertson, Engin Kirda, Hamed Okhravi

Northeastern University
College of Computer and Information Science

NEU SecLab



MIT
LINCOLN
LABORATORY

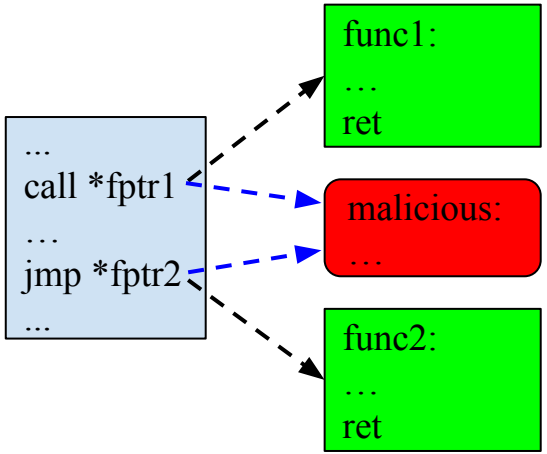
Outline

- Control Flow Hijacking
- Control Flow Integrity (CFI)
- Runtime Type Checking (RTC)
- Reuse Attack Protector (RAP)
- Typed ROP (TROP)
- PoC Exploit for Nginx
- Evaluation
- Conclusion

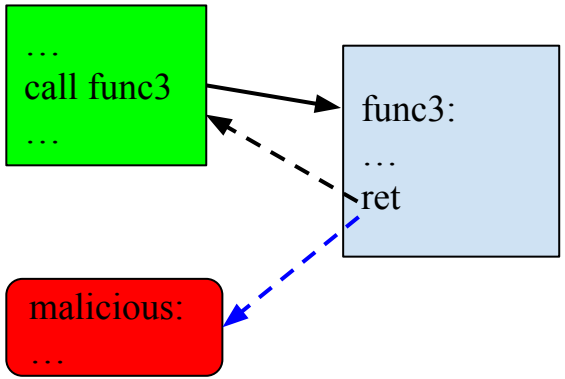
Control Flow Hijacking

- **Memory Corruption** can lead to violation of **Control Flow Graph (CFG)**

Forward Edges (calls, jumps**)**



Backward Edges (return addresses**)**



Control Flow Integrity (CFI)

- CFI prevents control flow hijacking by enforcing CFG at runtime
- CFG is usually generated ***statically*** using **Points-to Analysis**
  - **DSA**: Data Structure Analysis
  - **SVF**: Static Value-Flow Analysis
- Constructing **Sound** and **Precise** CFGs is **undecidable** and **impractical**

Runtime Type Checking (RTC)

- **Runtime Type Checking (RTC)** generates the CFG based on **Type Signature**
- RTC matches the type signature of each indirect control transfer with its target.
- Forward edge
  - The type of function pointer and the target are checked at each control transfer.
- Backward edge
  - The type of callee is checked during the function epilogue.
- Implementations
  - TypeArmor, KCFI (Kernel CFI), MCFI (Modular CFI), LLVM-CFI

Reuse Attack Protector (RAP)

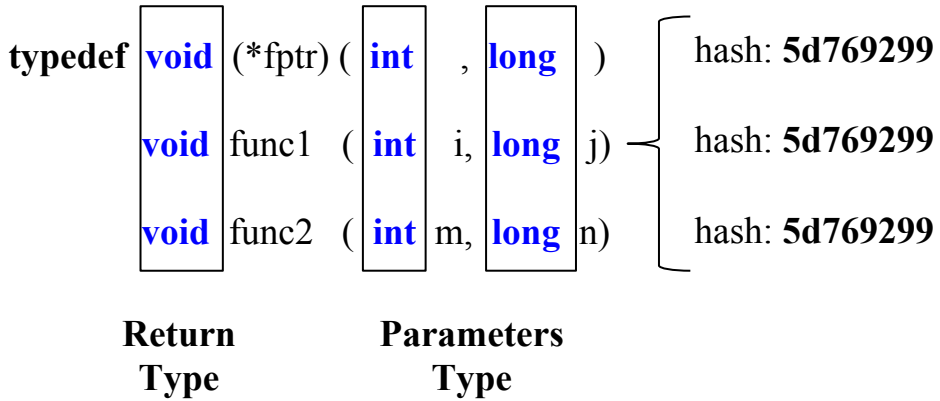
RAP™ is here. Public demo in 4.5 test patch and commercially available today!

April 28, 2016

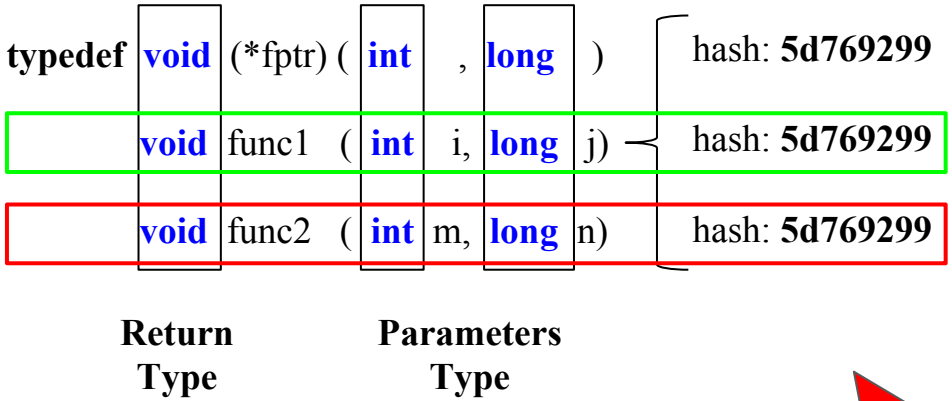
RAP™ Demonstrates World-First Fully CFI-Hardened OS Kernel

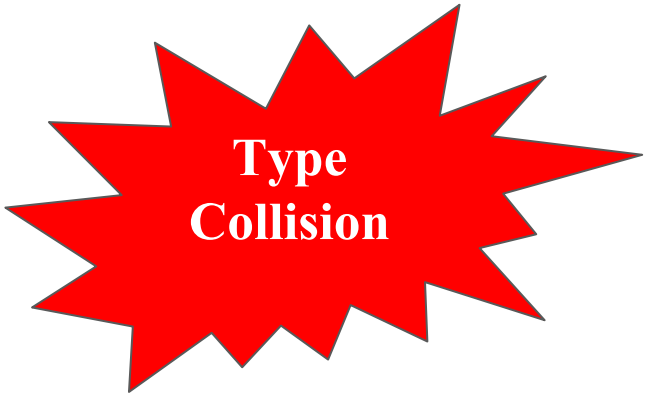
Type-based, high-performance, high-security, forward/backward-edge CFI
February 6, 2017

Reuse Attack Protector (RAP)



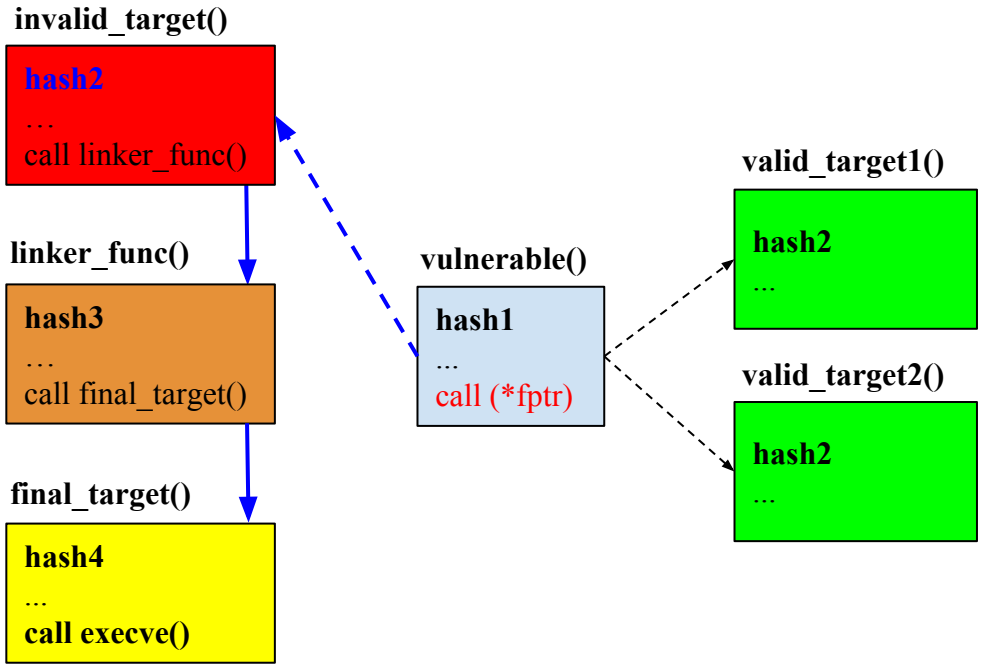
Reuse Attack Protector (RAP)





Sample Vulnerable Program

```
1  typedef void (*FunctionPointer)(void);
2  int flag = 0;
3  char *cmd;
4  void valid_target1(void) {
5      printf("Valid Target 1\n");
6  }
7  void valid_target2(void) {
8      printf("Valid Target 2\n");
9  }
10
11 int final_target(char *cmd) {
12     system(cmd);
13 }
14 int linker_func(void) {
15     if (flag ==1)
16         final_target(cmd);
17 }
18 void invalid_target(void) {
19     linker_func();
20 }
21 void vulnerable(char * input) {
22     FunctionPointer corruptible_fptr;
23     char buf[20];
24     if (strcmp(input, "1") == 0)
25         corruptible_fptr = &valid_target1;
26     else
27         corruptible_fptr = &valid_target2;
28     printf(input);
29     strcpy(buf, input);
30     corruptible_fptr();
31 }
```



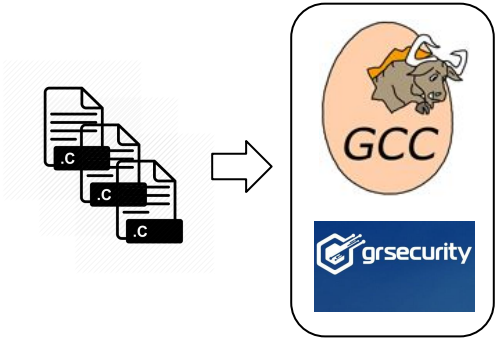
Research Questions

- Can RTC be practically bypassed using **type collisions**?
- Are there enough intermediate functions with satisfiable constraints in real-world applications?
- How prevalent are these constructs in real-world applications?

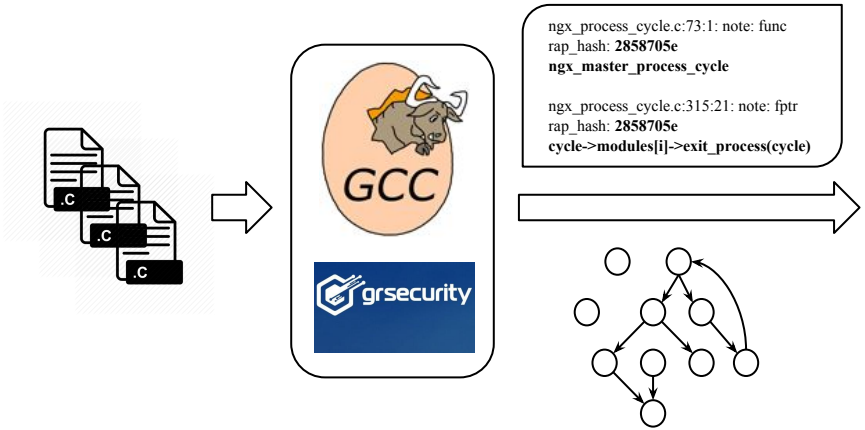
Threat Model

- The attacker has arbitrary **read** and **write** primitives to the memory
- The application contains **one strong** or **multiple limited** memory corruption vulnerability
- DEP and ASLR are enabled
- RAP is in place

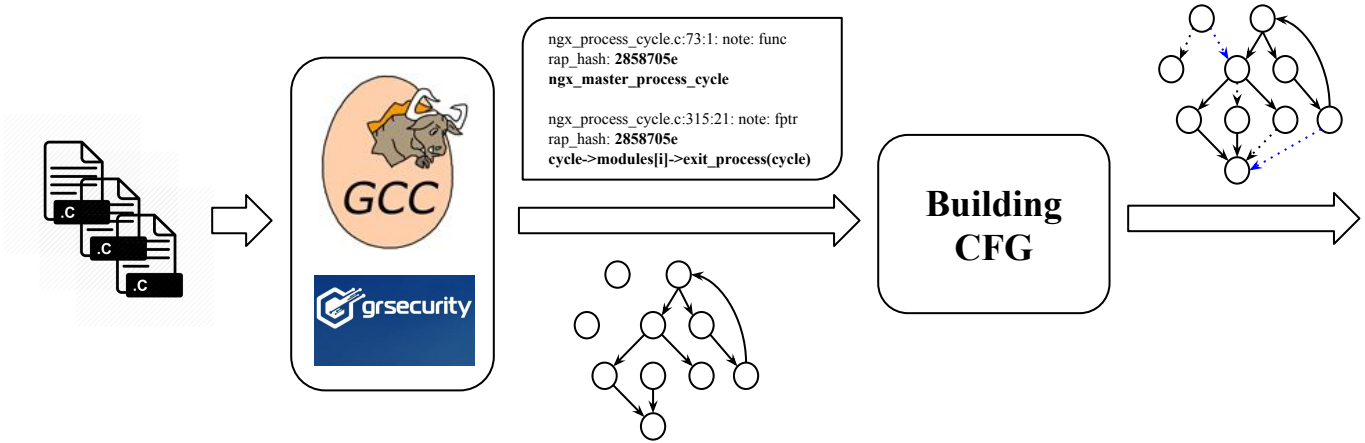
Typed ROP (TROP)



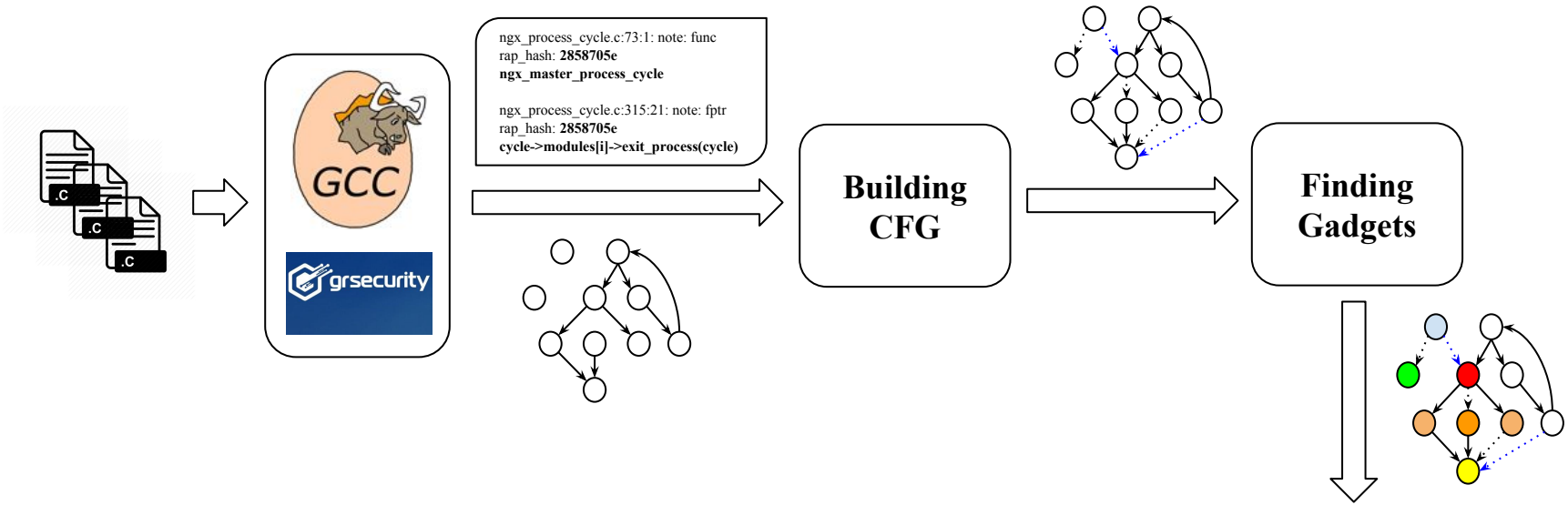
Typed ROP (TROP)



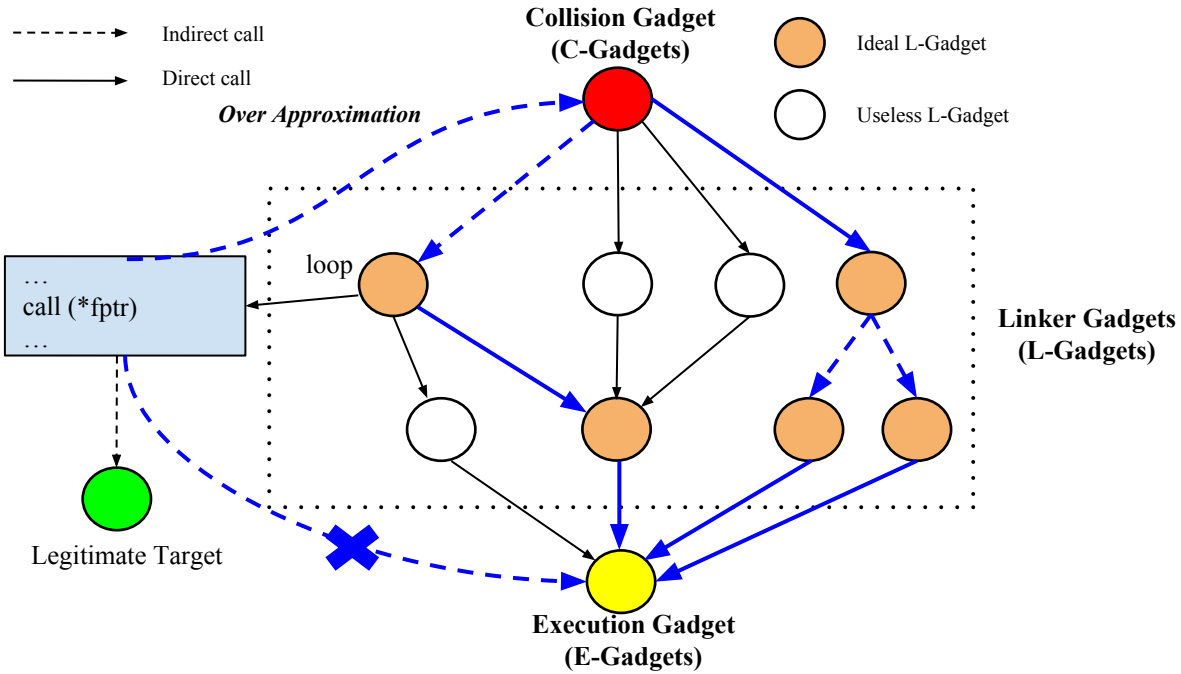
Typed ROP (TROP)



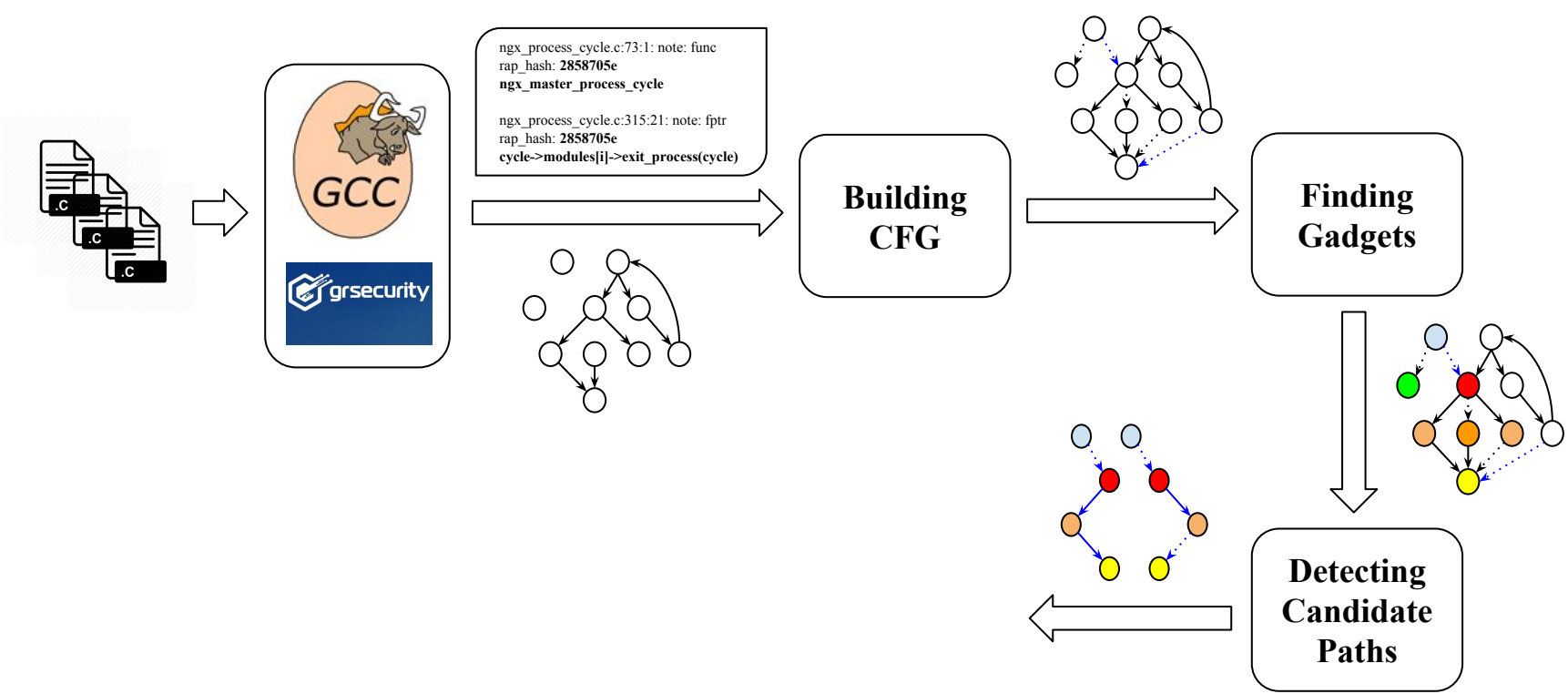
Typed ROP (TROP)



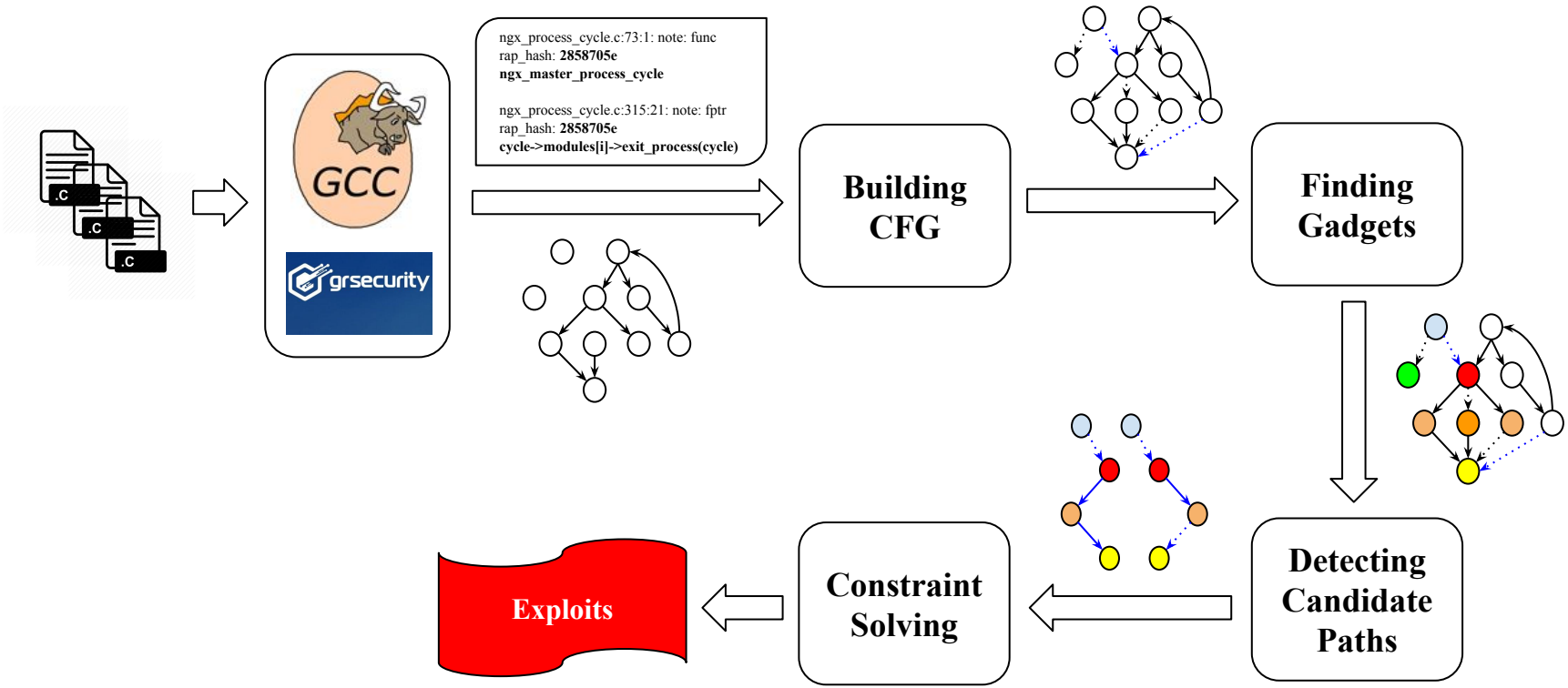
Gadgets


Indirect call
Direct call
Over Approximation
Collision Gadget
(C-Gadgets)
Ideal L-Gadget
Useless L-Gadget
loop
…
call (*fptr)
…
Linker Gadgets
(L-Gadgets)
Legitimate Target
Execution Gadget
(E-Gadgets)

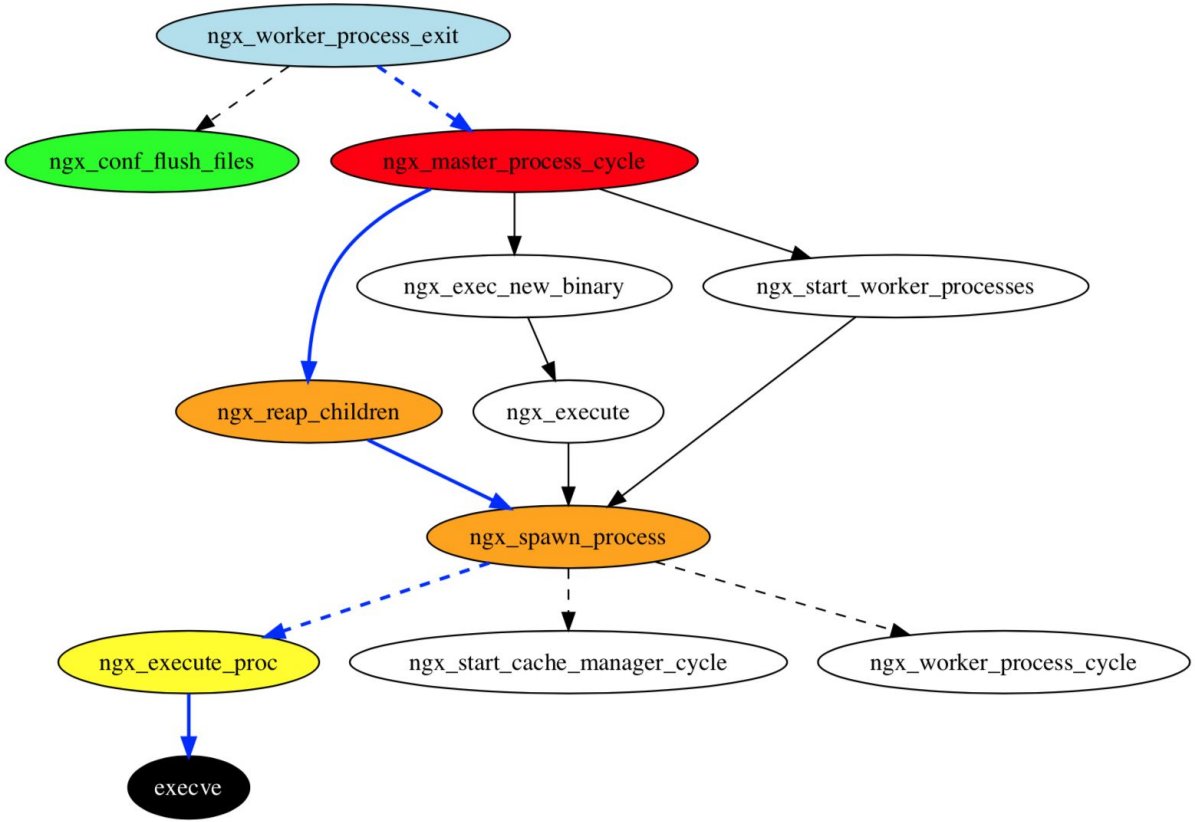
Typed ROP (TROP)



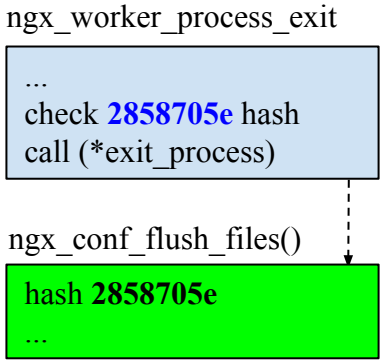
Typed ROP (TROP)



PoC Exploits (Nginx)

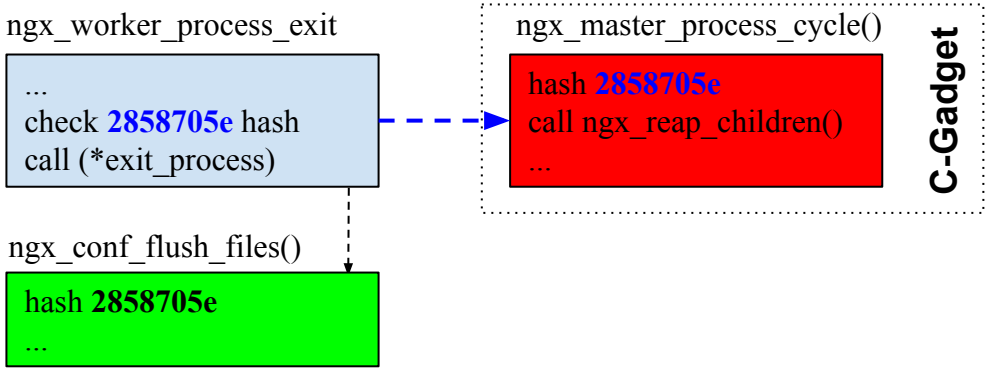


PoC Exploits (Nginx)



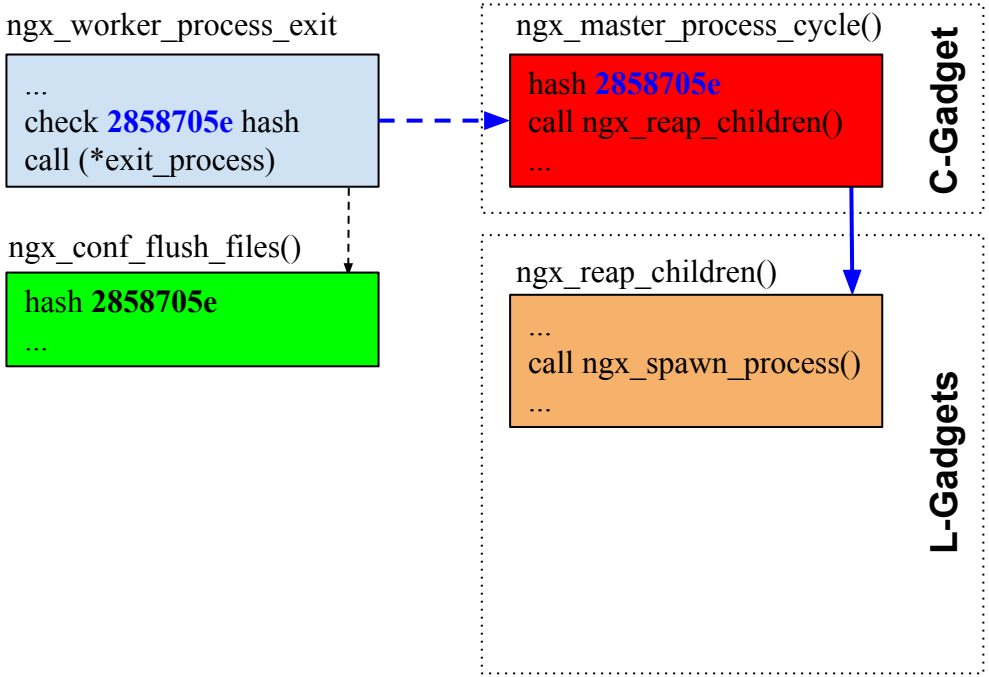

```
static void
ngx_worker_process_exit(ngx_cycle_t *cycle) {
    ngx_uint_t i;
    ngx_connection_t *c;

    for (i = 0; cycle->modules[i]; i++) {
        if (cycle->modules[i]->exit_process) {
            cycle->modules[i]->exit_process(cycle);
        }
    }
    …
}
```

PoC Exploits (Nginx)



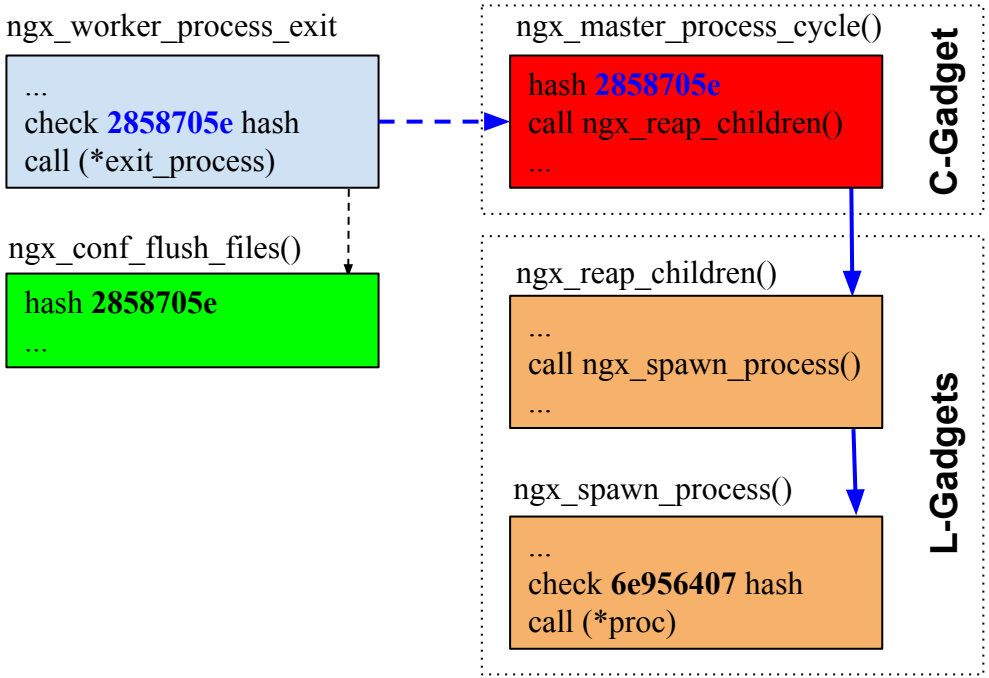

```
void
ngx_master_process_cycle(ngx_cycle_t * cycle) {
   …
   /* By setting this condition to true, the attacker can
    * reach to the next gadget which is ngx_reap_children()
    */
   if (ngx_reap) {
      ngx_reap = 0;
      ngx_log_debug0(NGX_LOG_DEBUG_EVENT,
                     cycle, log, 0, "reap children");
      live = ngx_reap_children(cycle);
     …
     …
}
```

PoC Exploits (Nginx)



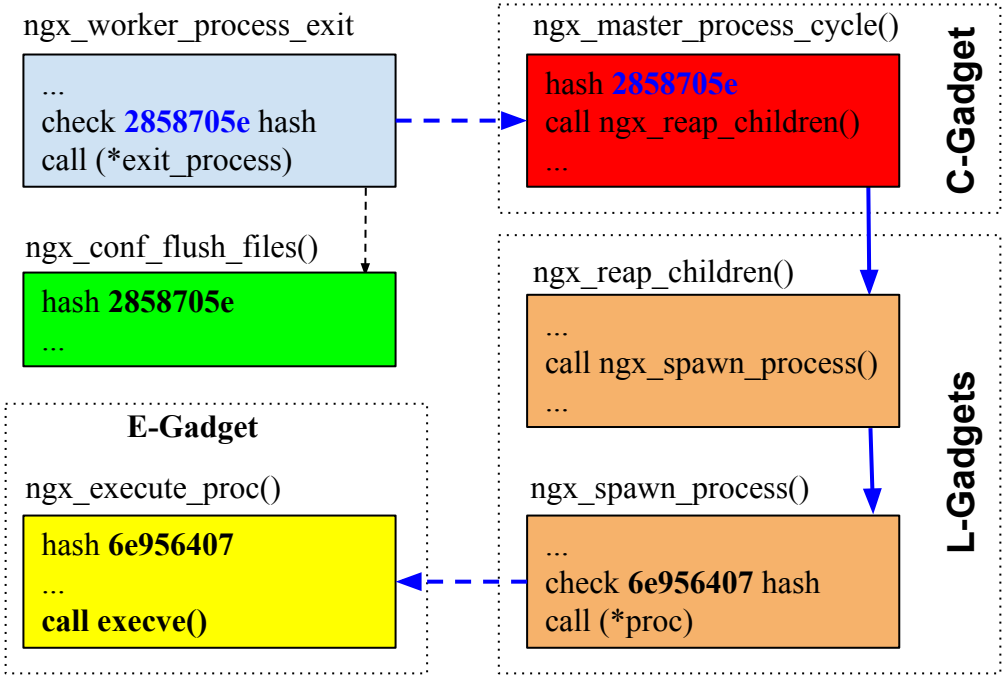

```
static ngx_uint_t
ngx_reap_children(ngx_cycle_t * cycle) {
   …
   for (i = 0; i < ngx_last_process; i++) {
      …
      if (ngx_processes[i].respawn &&
          !ngx_processes[i].exiting &&
          !ngx_terminate &&
          !ngx_quit) {
         if (ngx_spawn_process(cycle,
             ngx_processes[i].proc,
             ngx_processes[i].data,
             ngx_processes[i].name, i)
             == NGX_INVALID_PID) {
             …
```

PoC Exploits (Nginx)




```
ngx_pid_t ngx_spawn_process(ngx_cycle_t *cycle,
      ngx_spawn_proc_pt proc, void *data, char *name,
      ngx_int_t respawn) {
   …
   switch (pid) {
      case -1:
         …
      case 0:
         ngx_pid = ngx_getpid();
         proc(cycle, data);
         …
}
```

PoC Exploits (Nginx)




```
static void
ngx_execute_proc(ngx_cycle_t *cycle, void *data) {
    ngx_exec_ctx_t * ctx = data;

    if (execve(ctx->path, ctx->argv, ctx->envp) == -1) {
        ngx_log_error(NGX_LOG_ALERT,
                cycle -> log,
                ngx_errno,
                "execve() failed while executing %s\"%s\"",
                 ctx -> name, ctx -> path);
    }

    exit(1);
}
```

Evaluation

- Type Collisions
- Gadget Distribution
- Libc
- Type Checking vs. Points-to Analysis
- Type Diversification
- Practical Challenges

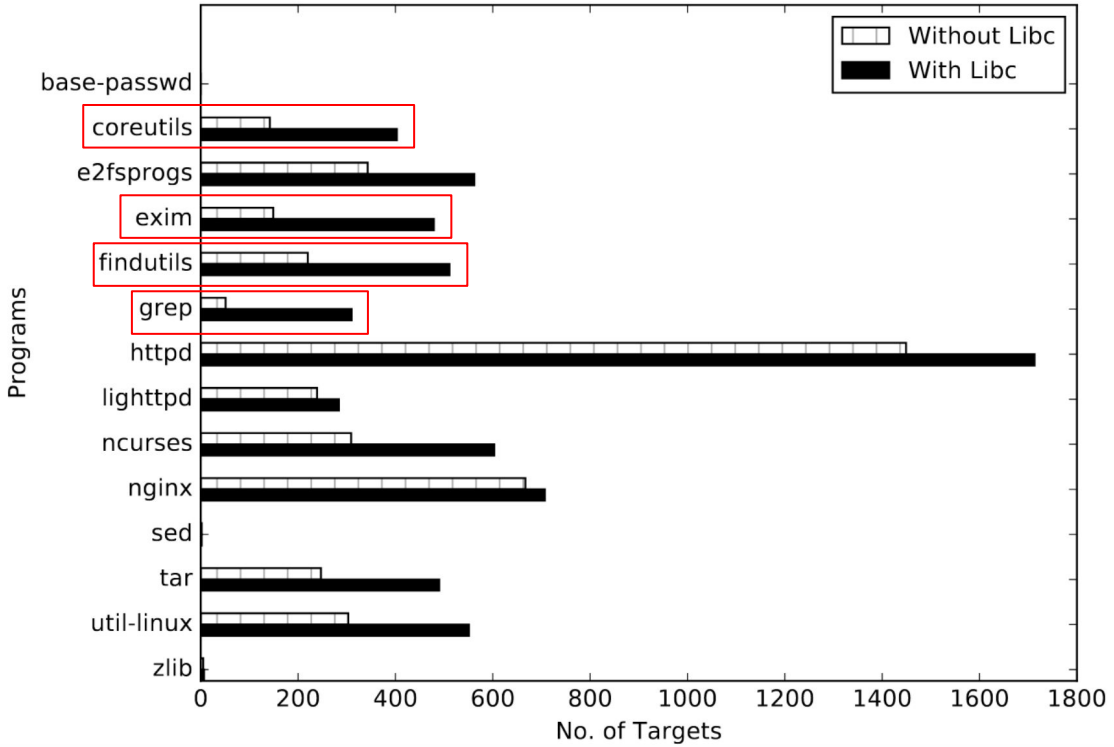
Type Collisions

App	Version	Function Pointer	Call Sites	Functions	Functions w/ Hash	Function Targets		Indirect Calls	
						All	Invalid	All	Invalid
base-passwd	3.5.39	6	6	45	45 (100.0%)	0	0	0	0 (0.0%)
coreutils	8.2	42	80	1,789	682 (38.1%)	116	43	416	110 (26.4%)
e2fsprogs	1.42.13	97	264	1,964	1,243 (63.3%)	251	176	1,383	400 (28.9%)
exim	4.89	43	93	968	607 (62.7%)	88	121	359	165 (46.0%)
findutils	4.6.0	28	52	821	554 (67.5%)	200	89	326	65 (19.9%)
grep	2.25	19	28	460	264 (57.4%)	38	19	113	52 (46.0%)
httpd	2.4.25	248	546	2,800	2,338 (83.5%)	1,332	483	3,915	794 (20.3%)
lighttpd	1.4.45	27	108	899	524 (58.3%)	228	40	830	221 (26.6%)
ncurses	6.0	46	77	1,835	1,045 (56.9%)	156	273	969	397 (41.0%)
nginx	1.10.1	84	290	1,299	977 (75.2%)	610	319	5,977	3,512 (58.8%)
sed	4.2.2	1	1	213	140 (65.7%)	2	0	2	0 (0.0%)
tar	1.28	46	86	1,166	730 (62.6%)	141	166	1,008	754 (74.8%)
util-linux	2.27.1	53	75	3,143	1,681 (53.5%)	211	177	1,060	643 (60.7%)
zlib	1.2.8	5	14	152	108 (71.1%)	5	0	13	0 (0.0%)

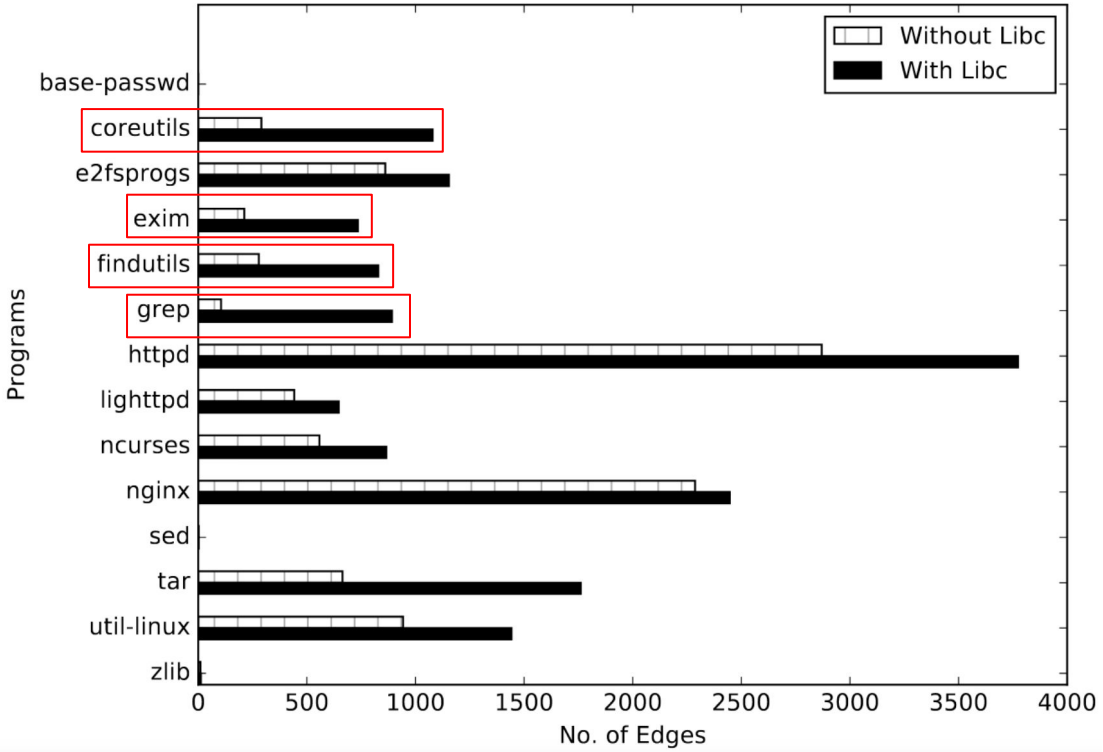
Gadget Distribution

App	Version	C-GADGET	L-GADGET	E-GADGET
nginx	1.10.1	8	6	1
httpd	2.4.25	40	19	5
lighttpd	1.4.45	8	29	6
exim	4.90	16	32	7

Libc (Targets)



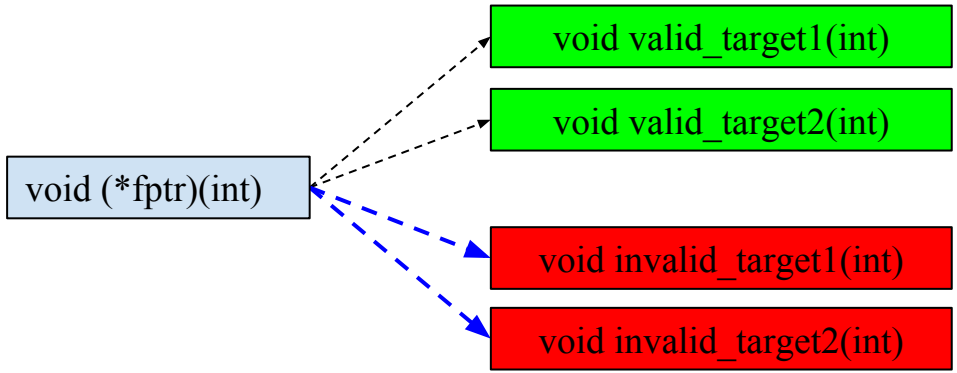
Libc (Edges)


Without Libc
With Libc
base-passwd
coreutils
e2fsprogs
exim
findutils
grep
httpd
lighttpd
ncurses
nginx
sed
tar
util-linux
zlib
Programs
0
500
1000
1500
2000
2500
3000
3500
4000
No. of Edges

Type Checking vs. Points-to Analysis

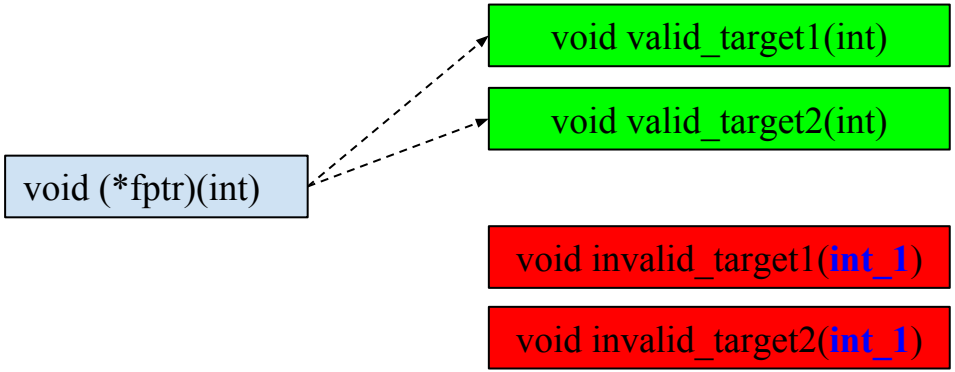
App	Base	Type Checking		Points-to Analysis	
		Total	Invalid	Total	Invalid
base-passwd	0	0	0 (0.0%)	0	0 (0.0%)
coreutils	213	291	78 (26.8%)	308	198 (64.3%)
e2fsprogs	557	861	304 (35.3%)	42	15 (35.7%)
exim	107	212	105 (49.5%)	169	99 (58.6%)
findutils	237	279	42 (15.1%)	448	231 (51.6%)
grep	54	105	51 (48.6%)	108	60 (55.6%)
httpd	2,126	2,870	744 (25.9%)	-	-
lighttpd	327	442	115 (26.0%)	1,096	938 (85.6%)
ncurses	291	558	267 (47.8%)	507	238 (46.9%)
nginx	1,276	2,287	1,011 (44.2%)	-	-
sed	2	2	0 (0.0%)	2	0 (0.0%)
tar	208	664	456 (68.7%)	360	167 (46.4%)
util-linux	311	943	632 (67.0%)	596	465 (78.0%)
zlib	10	10	0 (0.0%)	10	4 (40.0%)

Type Diversification



Type Diversification

- Complicates separate compilation



Practical Challenges

- Mismatch Types
  - **void *** can point to any pointer (e.g., **int ***)
- Support for Assembly Code

Conclusion

- Evaluated RTC from security and practicality perspectives
- Type collisions between function pointers and E-Gadgets are rare
- TROP showed collisions with other functions in a nested fashion can be exploited
- Gadgets for mounting TROP are **abundant** in real-world applications
- RTC is a practical defense but **not sufficient** to prevent control flow hijacking

Questions?

Reza Mirzazade farkhani

reza699@ccs.neu.edu