软件安全与漏洞分析

3.4 返回导向编程的发展（3）

Previously in Software Security

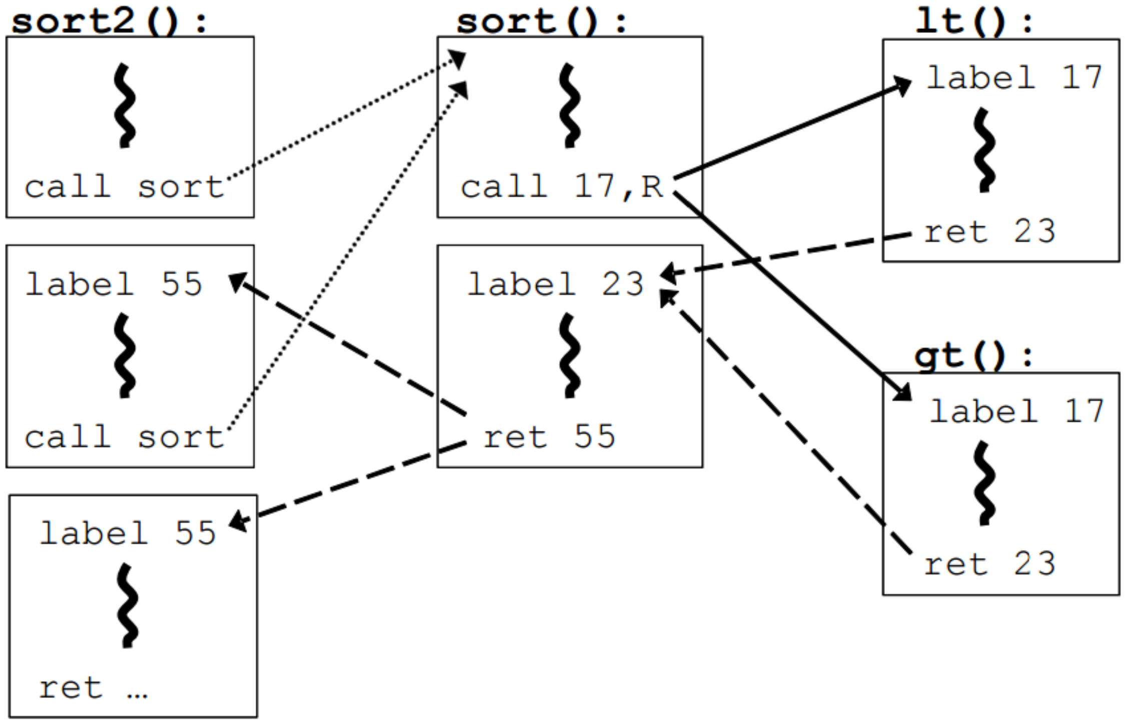
- 控制流完整性保护（CFI）及其几种主要实现思路
  - 基本思想
  - 实用（粗粒度）的CFI
  - 基于动态优化的CFI
  - 利用硬件特性和虚拟化技术的CFI

返回导向编程的发展（3）

- 本节主题　－ CFI的弱点及绕过
  - CFI的根本性弱点
  - 针对CFI的改进型返回导向编程编码方法
  - 返回导向编程的变种：数据导向编程

回顾：控制流完整性保护

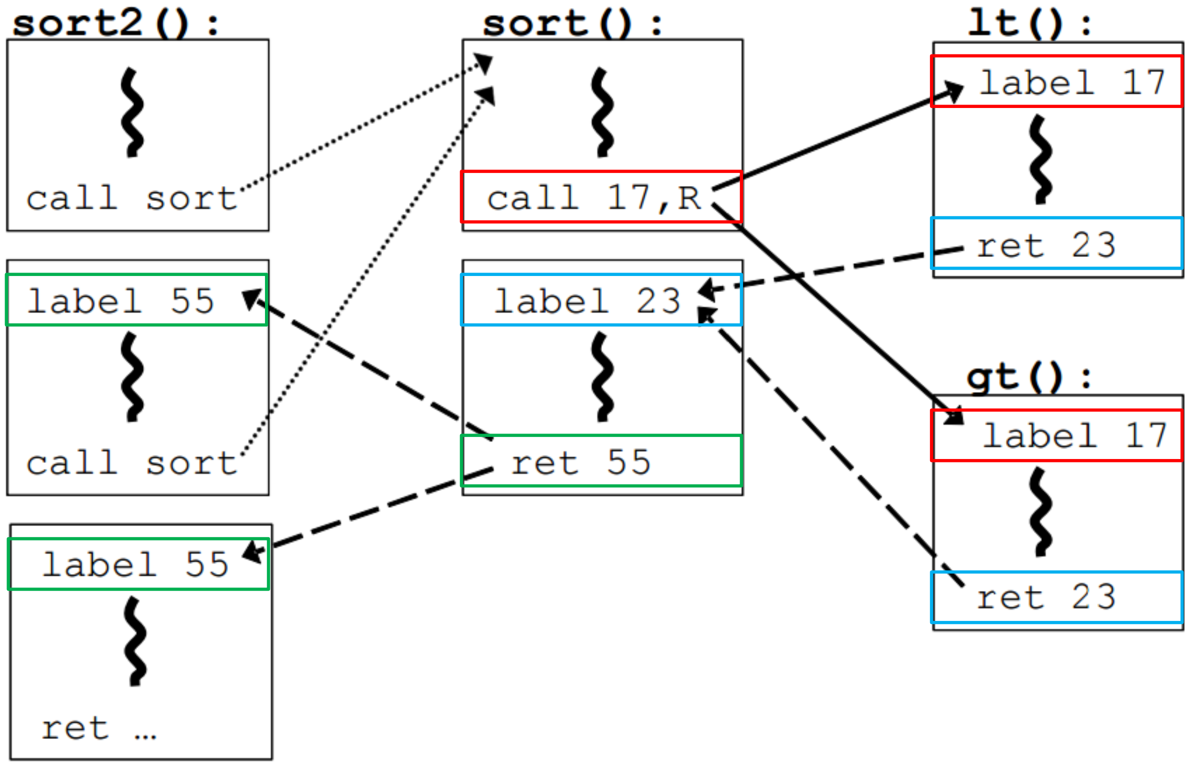
```
bool lt(int x, int y) {
    return x < y;
}
bool gt(int x, int y) {
    return x > y;
}

sort2(int a[], int b[], int len)
{
    sort( a, len, lt );
    sort( b, len, gt );
}
```



回顾：控制流完整性保护

- 通过预设的运行时校验，确保程序执行与预先定义的控制流图严格吻合
- 对程序中的每个间接控制转移，CFI需要指出其**合法跳转目标的范围**
- 需要兼顾执行开销问题
- 一些实现只面向对象程序的自身代码，另一些则面向进程空间内的所有代码

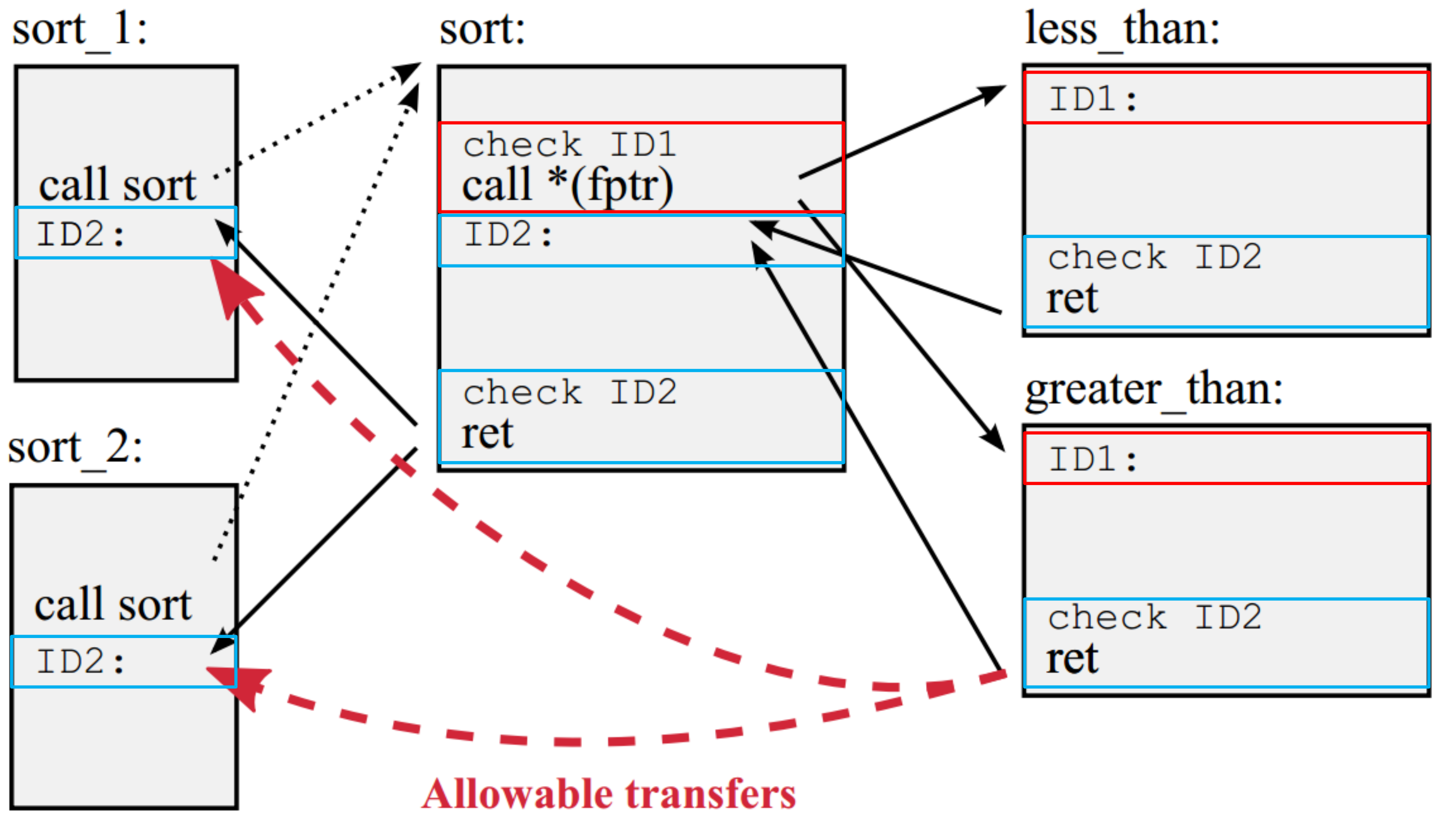
控制流完整性保护的根本性弱点



即使是在静态分析中，间接控制转移的合法跳转目标也既不是唯一的、也不是排他的

控制流完整性保护的根本性弱点

- 受到性能的制约，粗粒度化的CFI对控制流的约束更加松散

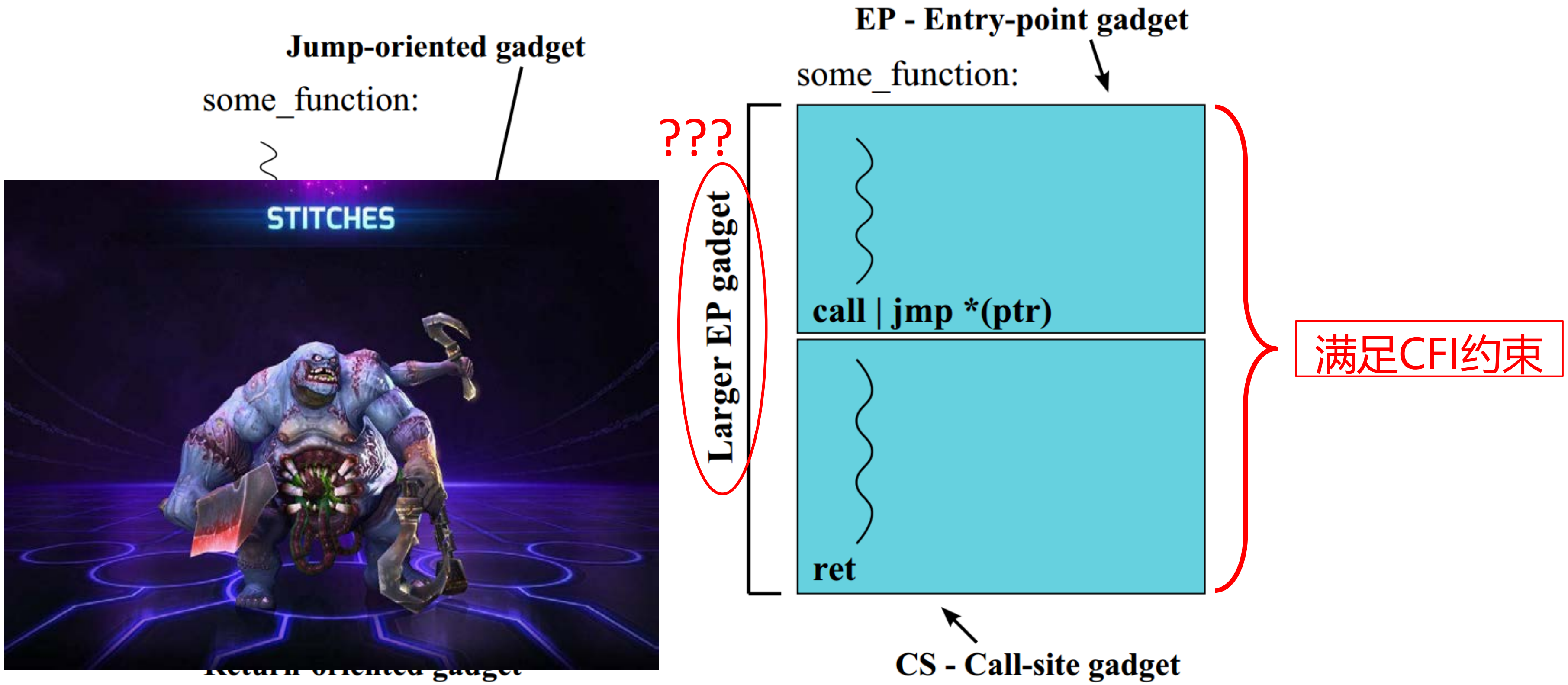


针对CFI的改进型返回导向编程

- CFI的弱点为返回导向编程提供了新类型的gadget来源
  - gadget的位置满足粗粒度CFI的约束规则
  - 新约束条件下的gadgets仍然有可能形成具备图灵完整性的攻击载荷
  - 副作用： gadget为了满足CFI约束而不可避免地增大了（无论是字节数还是指令数）

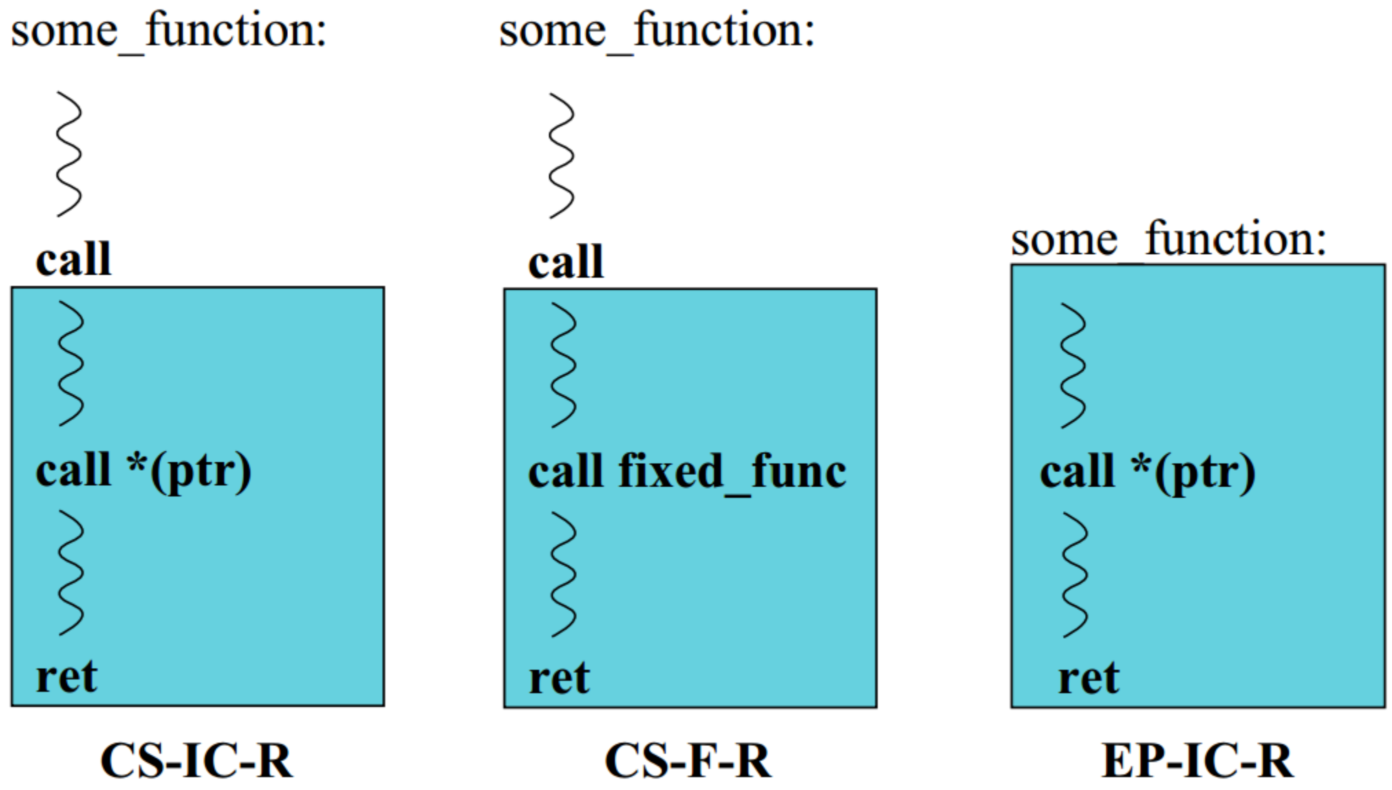
针对CFI的改进型返回导向编程

- 返回导向编程改进型1：利用以函数入口和函数调用点为起始的gadget



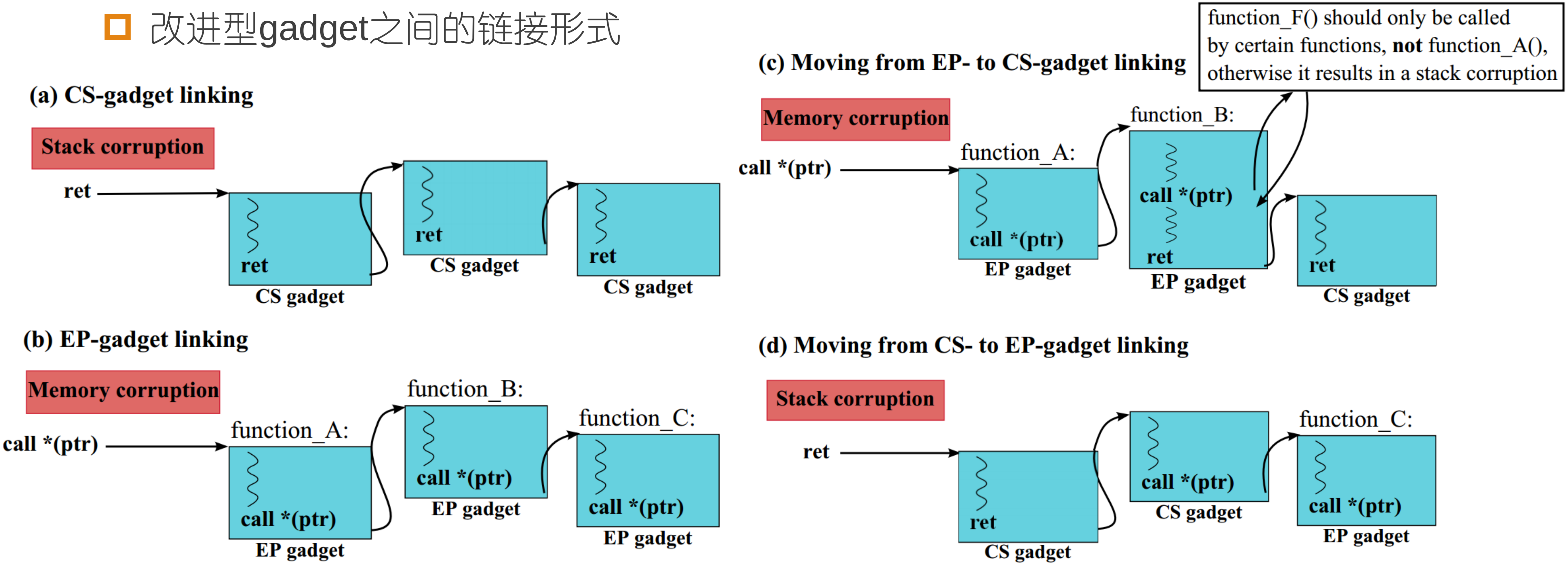
针对CFI的改进型返回导向编程

- 利用改进型的gadget实现为ROP服务的函数调用



针对CFI的改进型返回导向编程

- 改进型gadget之间的链接形式



针对CFI的改进型返回导向编程

- 参考文献：Göktas E, Athanasopoulos E, Bos H, et al. Out of control: Overcoming control-flow integrity[C]//Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014: 575-589.
- 扩展阅读：改进型ROP对粗粒度CFI方案CCFIR的概念验证攻击
  - 查阅论文附录所列出的改进型gadget，并在相应库文件中寻找它们的位置
  - 自学论文第IV章中所给出的攻击案例，思考攻击载荷如何利用附录中的gadget（即，在实现其功能的同时，避免其副作用）

针对CFI的改进型返回导向编程

- 返回导向编程改进型2：图灵完全的改进型gadget集对抗现存最严格的CFI实践
- 主要依据：实用型的CFI总体上采用了以下一些（或其中的一部分）防御策略
  - 制定针对间接控制转移指令的约束规则
  - 基于行为的启发式规则
  - 有条件的检查时机

针对CFI的改进型返回导向编程

Control-Flow Integrity (CFI) Policies	CFI for COTS [46]	kBouncer [31]	ROPecker [13]	ROPGuard [20]	EMET 4.1 [29]	*Combined Policy*
CFI_{RET} : destination has to be call-preceded	✓	✓	◯	✓	✓	✓
CFI_{RET} : destination can be taken from a code pointer	✓	✗	◯	✗	✗	✗
CFI_{JMP} : destination has to be call-preceded	✓	◯	◯	◯	◯	✓
CFI_{JMP} : destination can be taken from a code pointer	✓	◯	◯	◯	◯	✓
CFI_{CALL} : destination can be taken from an exported symbol	✓	◯	◯	◯	◯	✓
CFI_{CALL} : destination can be taken from a code pointer	✓	◯	◯	◯	◯	✓
CFI_{HEU} : allow only s consecutive short sequences,	◯	$s <= 7$	$s <= 10$	◯	◯	$s <= 7$
CFI_{HEU} : where *short* is defined as n instructions	◯	$n <= 20$	$n <= 6$	◯	◯	$n <= 20$
CFI_{TOC} : check at every indirect branch	✓	◯	◯	◯	◯	Always observed
CFI_{TOC} : check at critical API functions or system calls	◯	✓	✓	✓	✓	
CFI_{TOC} : check when leaving sliding code window	◯	◯	✓	◯	◯	

针对CFI的改进型返回导向编程

- 改进型gadget的类型（来源为kernel32.dll）
  - 具有call指令前缀的gadget（主要）
  - call-ret配对的gadget组合
  - “长nop” gadget
- 主要意义
  - 方案1没有回答的问题：**绕过CFI是否将导致返回导向编程的功能性有所缺损？**
  - 方案2给出了答案：**不会**

针对CFI的改进型返回导向编程

- 寄存器装载用gadget

Register	Call-Preceded Sequence (ending in `ret`)
EBP	`pop ebp`
ESI	`pop esi; pop ebp`
EDI	`pop edi; leave`
ECX	`pop ecx; leave`
EBX	`pop edi; pop esi; pop ebx; pop ebp`
EAX	`mov eax,edi; pop edi; leave`
EDX	`mov eax,[ebp-8]; mov edx,[ebp-4];` `pop edi; leave`

针对CFI的改进型返回导向编程

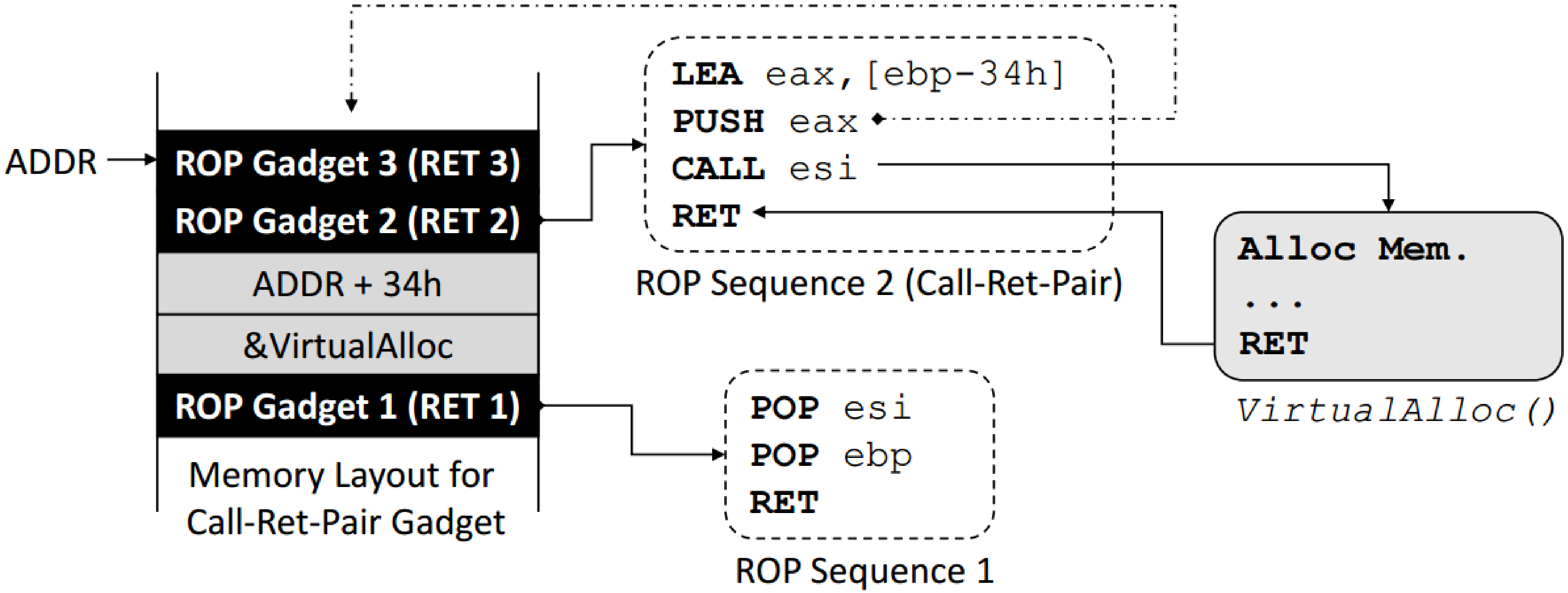
- 内存读写/算术逻辑运算/控制转移用gadget

Type	Call-Preceded Sequence (ending in `ret`)
LOAD (eax)	`mov eax, [ebp+8]; pop ebp`
STORE (eax)	`mov [esi],eax; xor eax,eax; pop esi; pop ebp`
STORE (esi)	`mov [ebp-20h],esi`
STORE (edi)	`mov [ebp-20h],edi`
ADD/SUB	`sub eax,esi; pop esi; pop ebp`
XOR	`xor eax,edi; pop edi; pop esi; pop ebp`

Type	Call-Preceded Sequence (ending in `ret`)
unconditional branch 1	`leave`
unconditional branch 2	`add esp,0Ch; pop ebp`
conditional LOAD(eax)	`neg eax; sbb eax,eax; and eax,[ebp-4];leave`

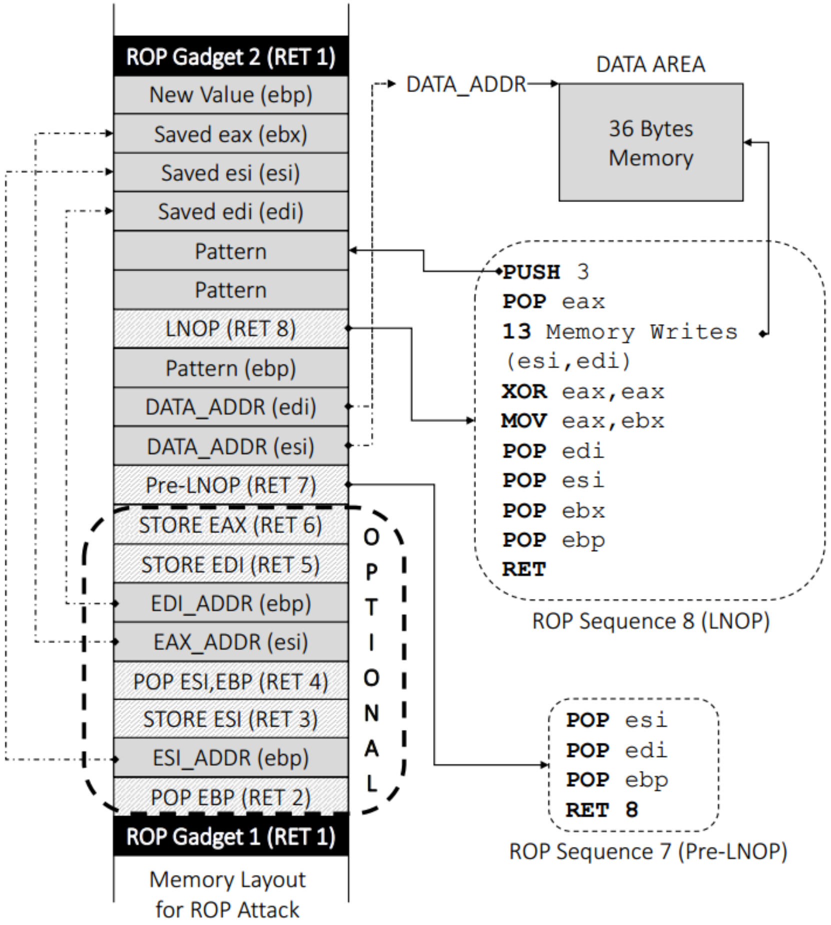
针对CFI的改进型返回导向编程

- call-ret配对的gadget组合



针对CFI的改进型返回导向编程

- “长nop” gadget
  - 动机：启发式规则不允许过多短指令组连续执行
  - 构造：含有>20条指令（但执行效果相当于nop）
  - 用途：间歇地布置于攻击载荷中，骗过启发式规则



针对CFI的改进型返回导向编程

- 参考文献：Davi L, Sadeghi A R, Lehmann D, et al. Stitching the Gadgets: On the Ineffectiveness of Coarse-Grained Control-Flow Integrity Protection[C]//USENIX Security. 2014, 14.
- 扩展阅读：查阅论文附录所列出的“长nop”gadget的具体实现
- 作业：将上述“长nop”gadget实现中的每条汇编指令翻译成对应的二进制串
  - 要求：查手册，手动做（是的，我知道有这种自动译码器）

返回导向编程的变种：数据导向编程

- 回顾Return-to-Libc攻击
  - 篡改返回地址指向指定函数入口
  - 通过溢出伪造输入参数等栈数据结构
  - 诱使函数在执行中使用伪造的参数数据，实现恶意目的
- 从Return-to-Libc攻击可以看出：**数据污染可以成为代码重用类攻击的重要成分**

返回导向编程的变种：数据导向编程

```
struct server{ int *cur_max, total, typ;} *srv;
int connect_limit = MAXCONN; int *size, *type;
char buf[MAXLEN];
size = &buf[8]; type = &buf[12];
...
while(connect_limit--) {
  readData(sockfd, buf);          // stack bof
  if(*type == NONE  )  break;
  if(*type == STREAM)             // condition
     *size = *(srv->cur_max);     // dereference
  else {
     srv->typ = *type;            // assignment
     srv->total += *size;         // addition
  } ... (following code skipped) ...
}
```

FTP server

|| ?

```
struct Obj{struct Obj *next; unsigned int prop;}
void updateList(struct Obj *list, int addend) {
   for(; list != NULL; list = list->next)
      list->prop += addend;
}
```

list updating

返回导向编程的变种：数据导向编程

stack growth ←

stack layout	buf[]	type	size	connect_limit	srv
malicious input for one round	**“AAA…AAA”**	**p**	**q**	**0x100**	**n-8**
	“AAA…AAA”	**m**	**p**	**0x100**	**p**

Overflow	Executed Instr. (Code 2)	Simulated Instr. (Code 3)

type ← p
size ← q
srv ← n-8
	if(*type == NONE) break;
	srv->typ = *type;
	srv->total += *size;
		if(list == NULL) break;
		srv = list;
		list->prop += addend;

type ← m
size ← p
srv ← p
	if(*type == NONE) break;
	if(*type == STREAM)
	*size = *(srv->cur_max);
		if(list == NULL) break;
		if(list == STREAM)
		list = list->next;

p – &list; q – &addend; m – &STREAM; n – &srv

返回导向编程的变种：数据导向编程

- 特点
  - 通过污染数据（特别是指针）**改变被重用代码的实际语义**
  - **借助循环不断注入新的攻击载荷**，使得被重用代码的实际执行效果随之改变

- 因此，**数据导向编程所重用的代码资源主要来自串操作（或同类执行行为）**

C Code	`srv->total += *size;`

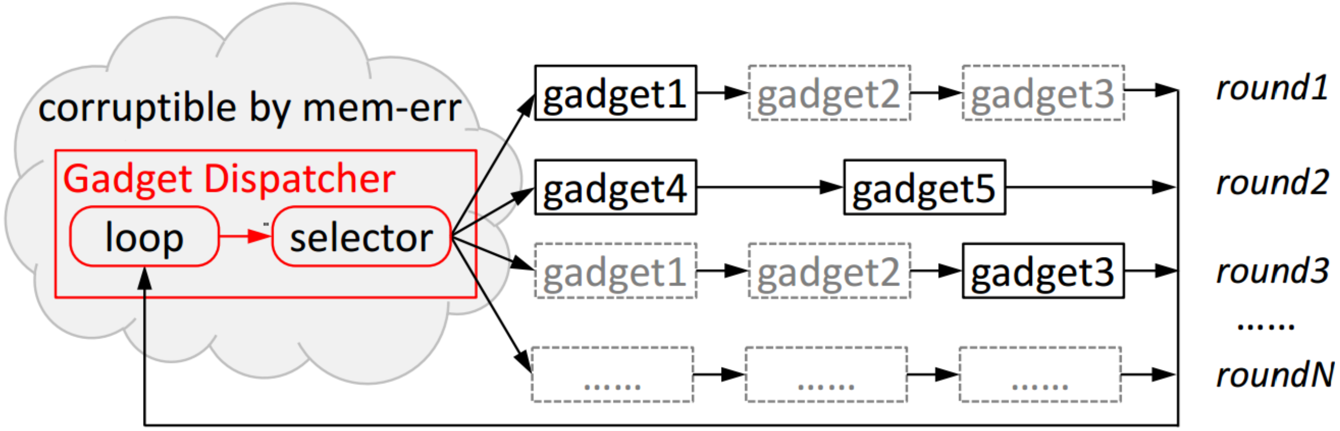
ASM Code	`1 mov (%esi), %ebx      //load micro-op`
	`2 mov 0x4(%edi), %eax //load micro-op`
	`3 add %ebx, %eax       //addition`
	`4 mov %eax, 0x4(%edi) //store micro-op`

C Code	`srv->typ = *type;`

ASM Code	`1 mov (%esi), %ebx     // load micro-op`
	`2 mov %ebx, %eax       // move`
	`3 mov %eax, 0x8(%edi) // store micro-op`

返回导向编程的变种：数据导向编程

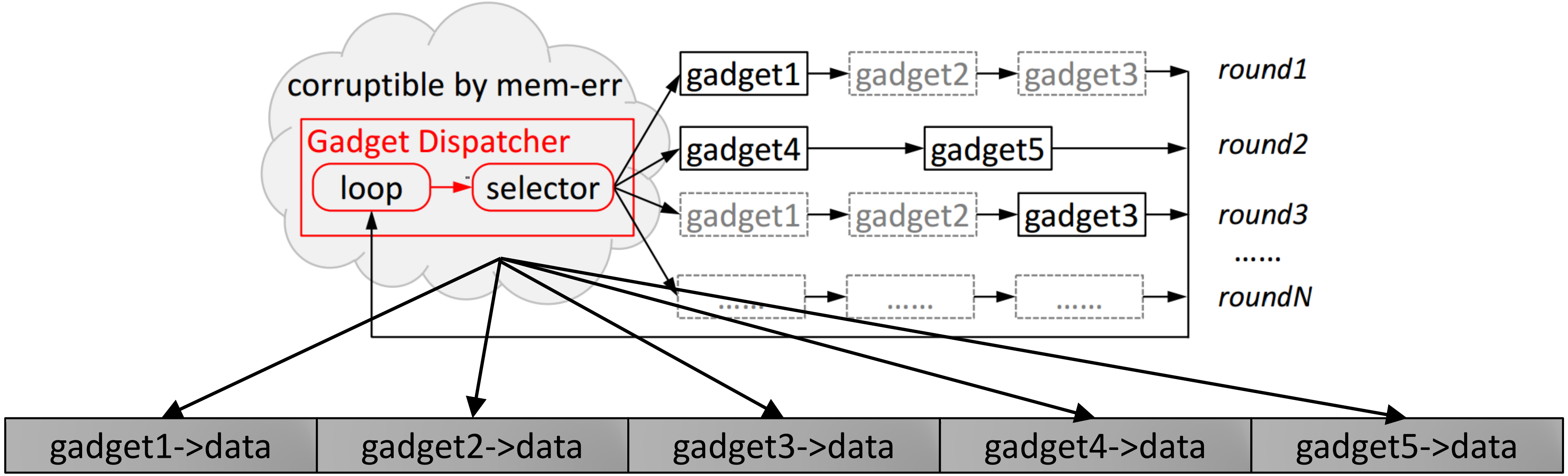
- 数据导向编程的循环调度



  - **交互式攻击** – 允许攻击者在每轮循环中输入不同的载荷以激活不同的gadget
  - **非交互式攻击** – 攻击者必须一次性输入整个攻击载荷

返回导向编程的变种：数据导向编程

- 非交互式攻击需要额外的指令元素以支持虚拟的跳转操作



返回导向编程的变种：数据导向编程

- 非交互式攻击需要额外的指令元素以支持虚拟的跳转操作

```
void cmd_loop(server_rec *server, conn_t *c) {
  while (TRUE) {
    pr_netio_telnet_gets(buf, ..);
    cmd = make_ftp_cmd(buf, ...);
    pr_cmd_dispatch(cmd); // dispatcher
  }
}
char *pr_netio_telnet_gets(char * buf, ...) {
  while(*pbuf->current!='\n' && toread>0)
    *buf++ = *pbuf->current++;
}
```

- 数据导向编程同样也是图灵完全的

针对CFI的改进型返回导向编程

- 参考文献：Hu H, Shinde S, Adrian S, et al. Data-oriented programming: On the expressiveness of non-control data attacks[C]//Security and Privacy (SP), 2016 IEEE Symposium on. IEEE, 2016: 969-986.
- 扩展阅读：学习论文V-B章节中所描述的图灵完全的DOP攻击实例

What’s next?

- ASLR（已经广泛应用的粗粒度型/更复杂的细粒度型）
- 令ASLR无效化的ROP攻击：just-in-time代码重用