Data-Oriented Programming

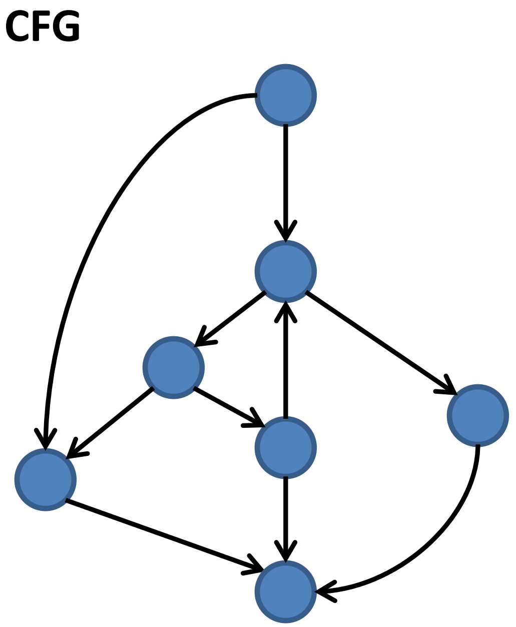
On the Expressiveness of Non-Control Data Attacks

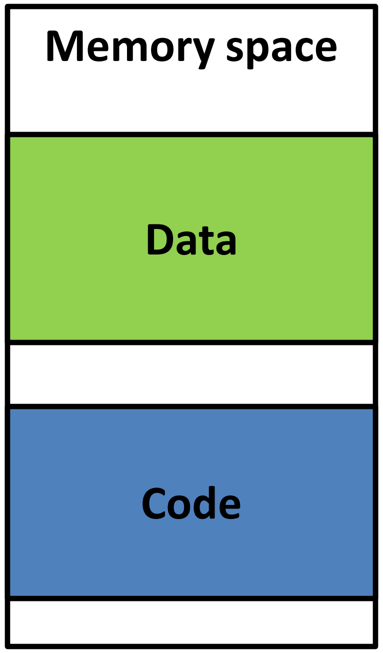
Hong Hu, Shweta Shinde,
Sendroiu Adrian, Zheng Leong Chua,
Prateek Saxena, Zhenkai Liang

Department of Computer Science
National University of Singapore

Control Attacks are Getting Harder

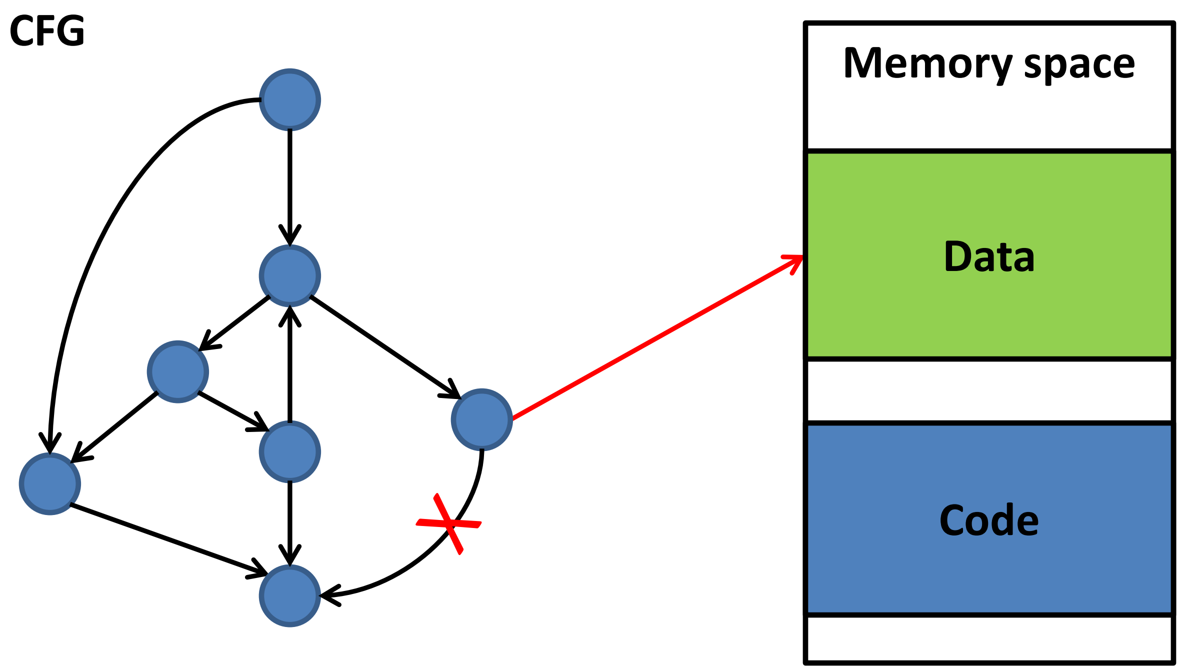
Control Attacks are Getting Harder





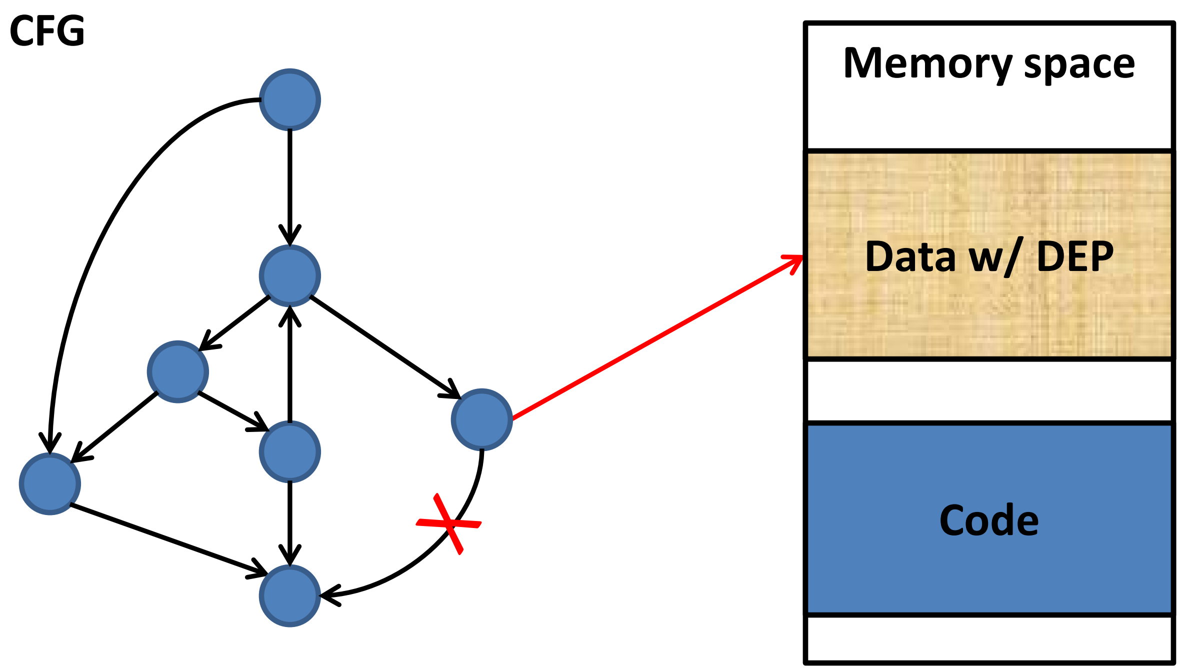
Control Attacks are Getting Harder

- Code injection



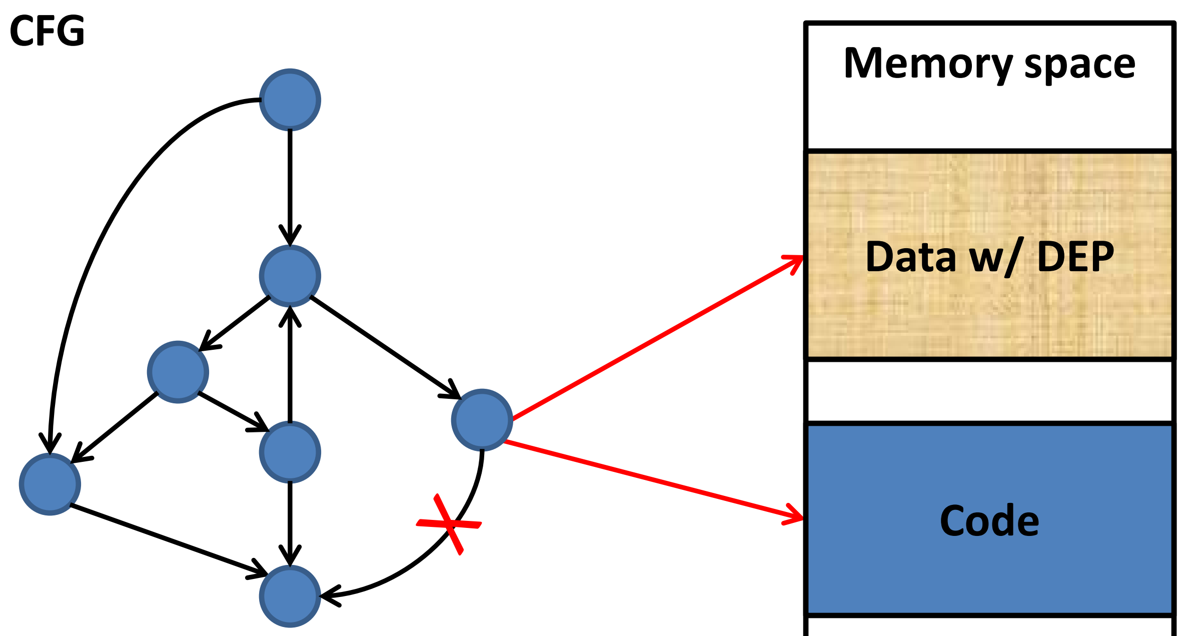
Control Attacks are Getting Harder

- Code injection ⇐ Data Execution Prevention



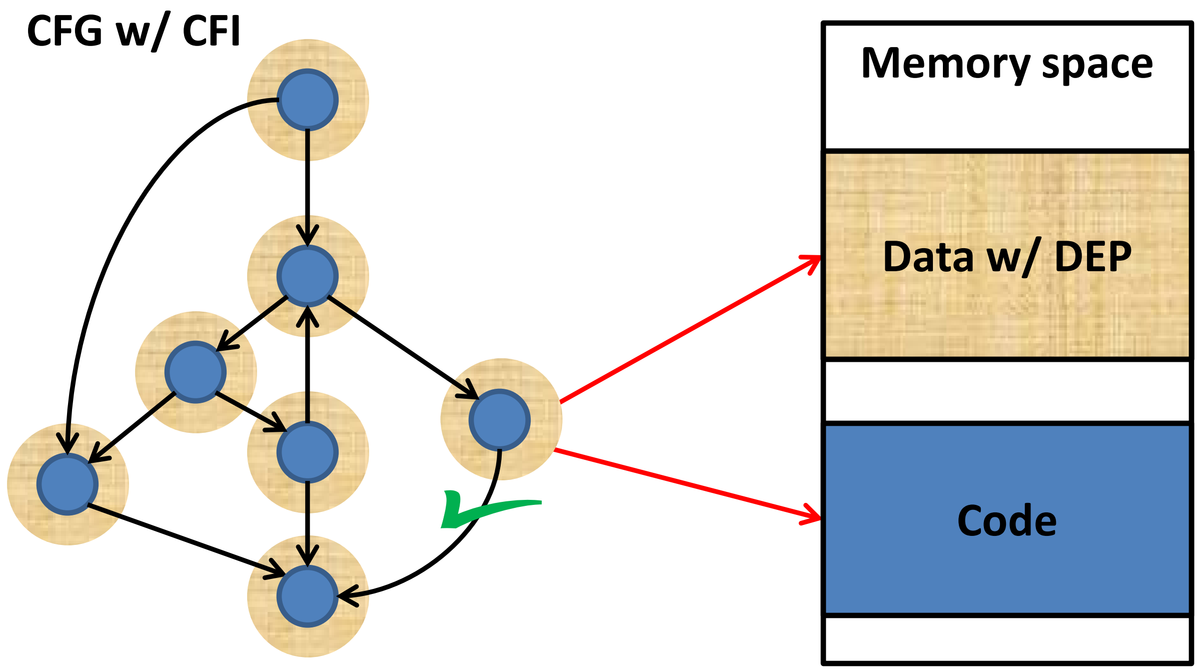
Control Attacks are Getting Harder

- Code injection ⇐ Data Execution Prevention
- Code reuse
  - return-to-libc
  - return-oriented programming (ROP)



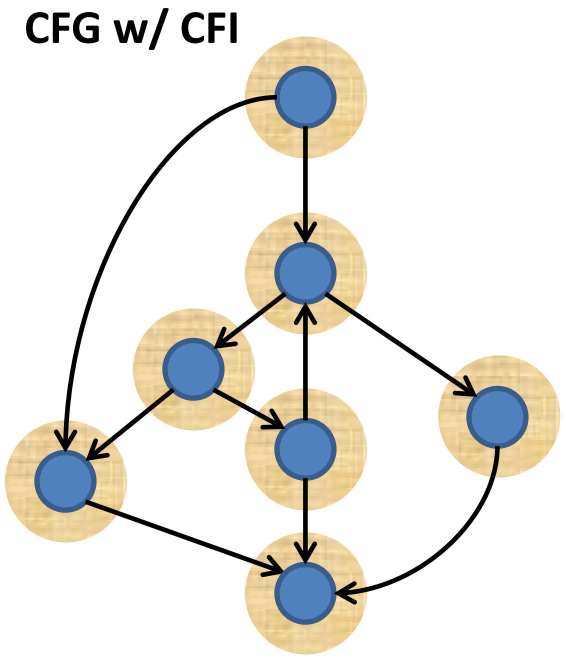
Control Attacks are Getting Harder

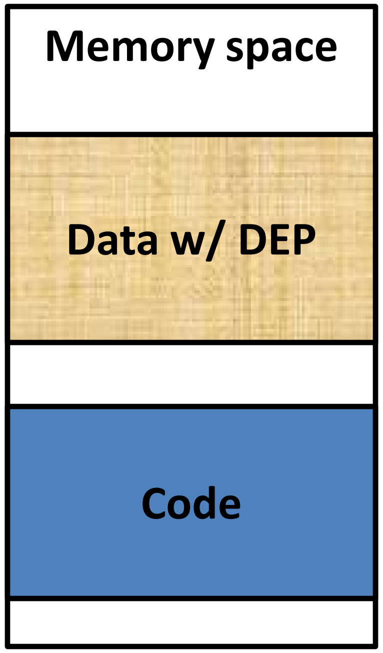
- Code injection ⟸ Data Execution Prevention
- Code reuse ⟸ Control Flow Integrity
  - return-to-libc
  - return-oriented programming (ROP)



A New Attack Class

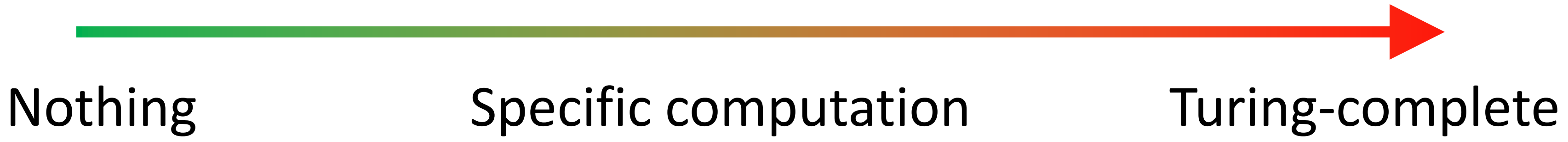
- Assume: conform to CFI & DEP

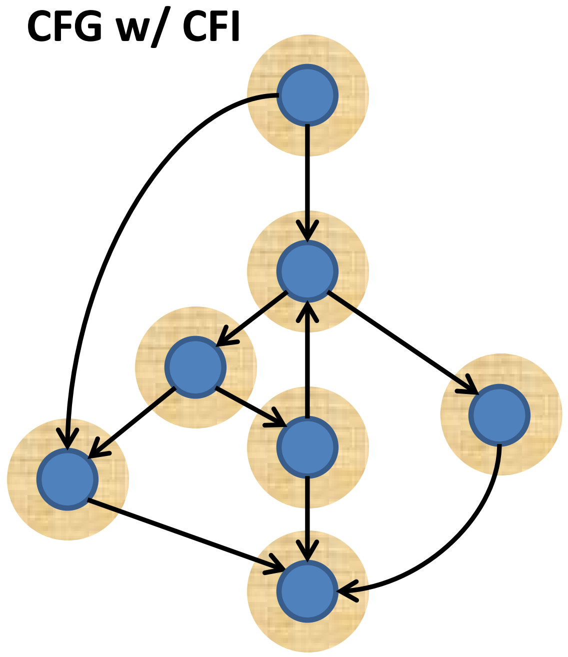


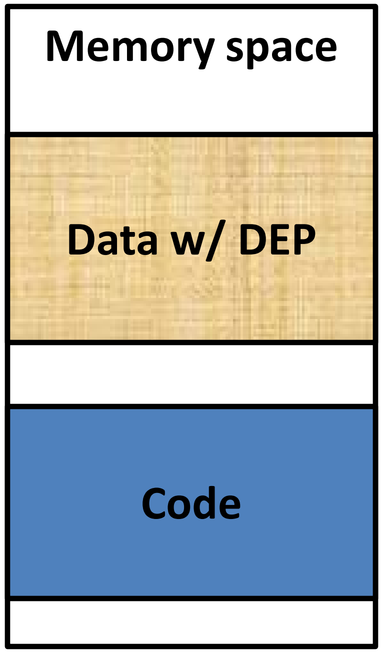


A New Attack Class

- Assume: conform to CFI & DEP
- Attackers' capability on arbitrary vul. programs?







Non-Control Data Attacks

- Corrupt/leak several bytes of security-critical data

Non-Control Data Attacks

- Corrupt/leak several bytes of security-critical data

```
//set root privilege   *
seteuid(0);
......
//set normal user privilege
seteuid(pw->pw_uid);
//execute user's command
```

```
//offset depends on IE version +
safemode = *(DWORD *)
             (jsobj + offset);
if(safemode & 0xB == 0) {
    Turn_on_God_Mode();
}
```

* Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-Control-Data Attacks Are Realistic Threats. In USENIX 2005.

+ Yang Yu. Write Once, Pwn Anywhere. In Black Hat USA 2014

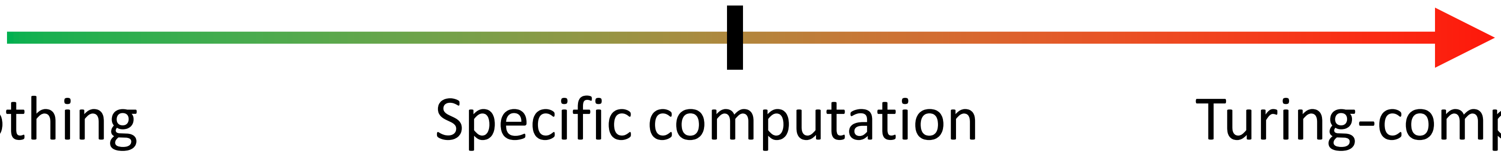
Non-Control Data Attacks

- Corrupt/leak several bytes of security-critical data

```
//set root privilege   *
seteuid(0);
......
//set normal user privilege
seteuid(pw->pw_uid);
//execute user's command
```

```
//offset depends on IE version +
safemode = *(DWORD *)
             (jsobj + offset);
if(safemode & 0xB == 0) {
    Turn_on_God_Mode();
}
```

- Special cases relying on particular data/functions
  - user id, safemode, private key, etc
  - interpreter – printf() (with "%n"), etc



* Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-Control-Data Attacks Are Realistic Threats. In USENIX 2005.

+ Yang Yu. Write Once, Pwn Anywhere. In Black Hat USA 2014

Contributions

- Non-control data attacks can be Turing-complete

Contributions

- Non-control data attacks can be Turing-complete
- Data-Oriented Programming (DOP)
  - build expressive non-control data attacks
  - independent of any specific data / functions

Contributions

- Non-control data attacks can be Turing-complete
- Data-Oriented Programming (DOP)
  - build expressive non-control data attacks
  - independent of any specific data / functions
- DOP builds attacks on real-world programs
  - bypass ASLR w/o address leakage
  - simulate a network bot
  - enable code injection

Motivating Example

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

Vulnerable Program

Motivating Example

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

Vulnerable Program

Motivating Example

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

Vulnerable Program

?

Motivating Example

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

Vulnerable Program

?

```
struct Obj {struct Obj *next; int prop;}

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

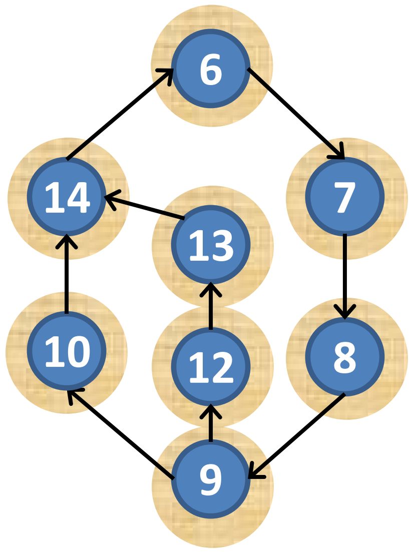
Malicious Computation

Motivating Example

```
1   struct server{int *cur_max, total, typ;} *srv;
2   int quota = MAXCONN; int *size, *type;
3   char buf[MAXLEN];
4   size = &buf[8]; type = &buf[12]
5   ...
6   while (quota--) {
7     readData(sockfd, buf);          // stack bof
8     if(*type == NONE ) break;
9     if(*type == STREAM)
10        *size = *(srv->cur_max);
11    else {
12        srv->typ = *type;
13        srv->total += *size;
14    } //...(following code skipped)...
15  }
```

Vulnerable Program

CFG w/ CFI



?

```
struct Obj {struct Obj *next; int prop;}

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

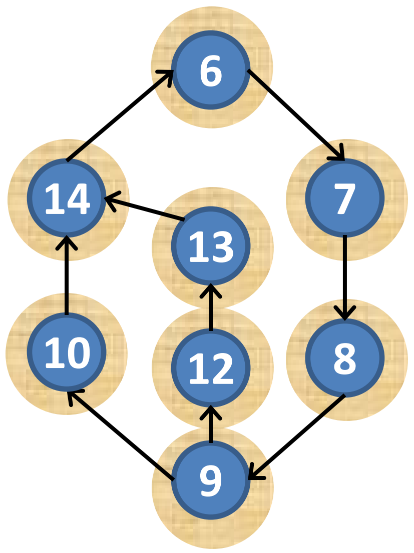
Malicious Computation

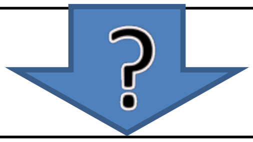
Motivating Example

```
1  struct server{int *cur_max, total, typ;} *srv;
2  int quota = MAXCONN; int *size, *type;
3  char buf[MAXLEN];
4  size = &buf[8]; type = &buf[12]
5  ...
6  while (quota--) {
7    readData(sockfd, buf);        // stack bof
8    if(*type == NONE ) break;
9    if(*type == STREAM)
10       *size = *(srv->cur_max);
11   else {
12       srv->typ = *type;
13       srv->total += *size;
14   } //...(following code skipped)...
15 }
```

Vulnerable Program

CFG w/ CFI





```
1  struct Obj {struct Obj *next; int prop;}
2
3  void updateList(struct Obj *list, int addend){
4   for(; list != NULL; list = list->next)
5      list->prop += addend;
6  }
```

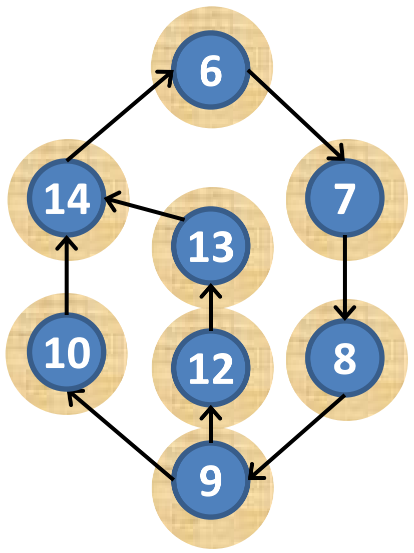
Malicious Computation

Motivating Example

```
1   struct server{int *cur_max, total, typ;} *srv;
2   int quota = MAXCONN; int *size, *type;
3   char buf[MAXLEN];
4   size = &buf[8]; type = &buf[12]
5   ...
6   while (quota--) {
7     readData(sockfd, buf);          // stack bof
8     if(*type == NONE ) break;
9     if(*type == STREAM)
10        *size = *(srv->cur_max);
11    else {
12        srv->typ = *type;
13        srv->total += *size;
14    } //...(following code skipped)...
15  }
```

Vulnerable Program

CFG w/ CFI



?

```
1   struct Obj {struct Obj *next; int prop;}
2
3   void updateList(struct Obj *list, int addend){
4    for(; list != NULL; list = list->next)
5       list->prop += addend;
6   }
```

Malicious Computation

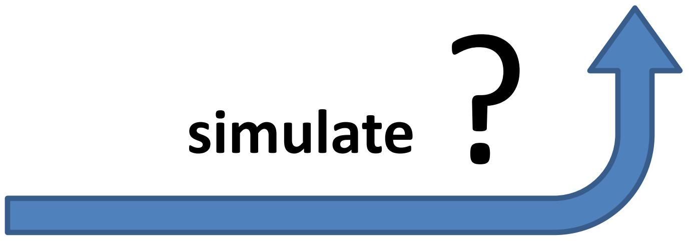
Motivating Example (cont.)

```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation



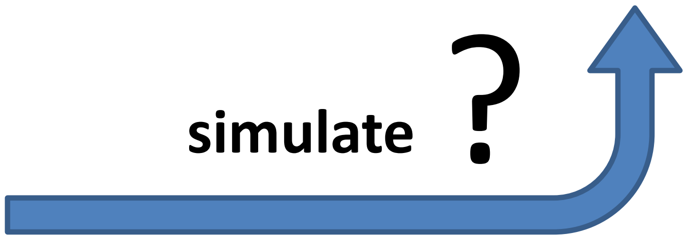
Motivating Example (cont.)

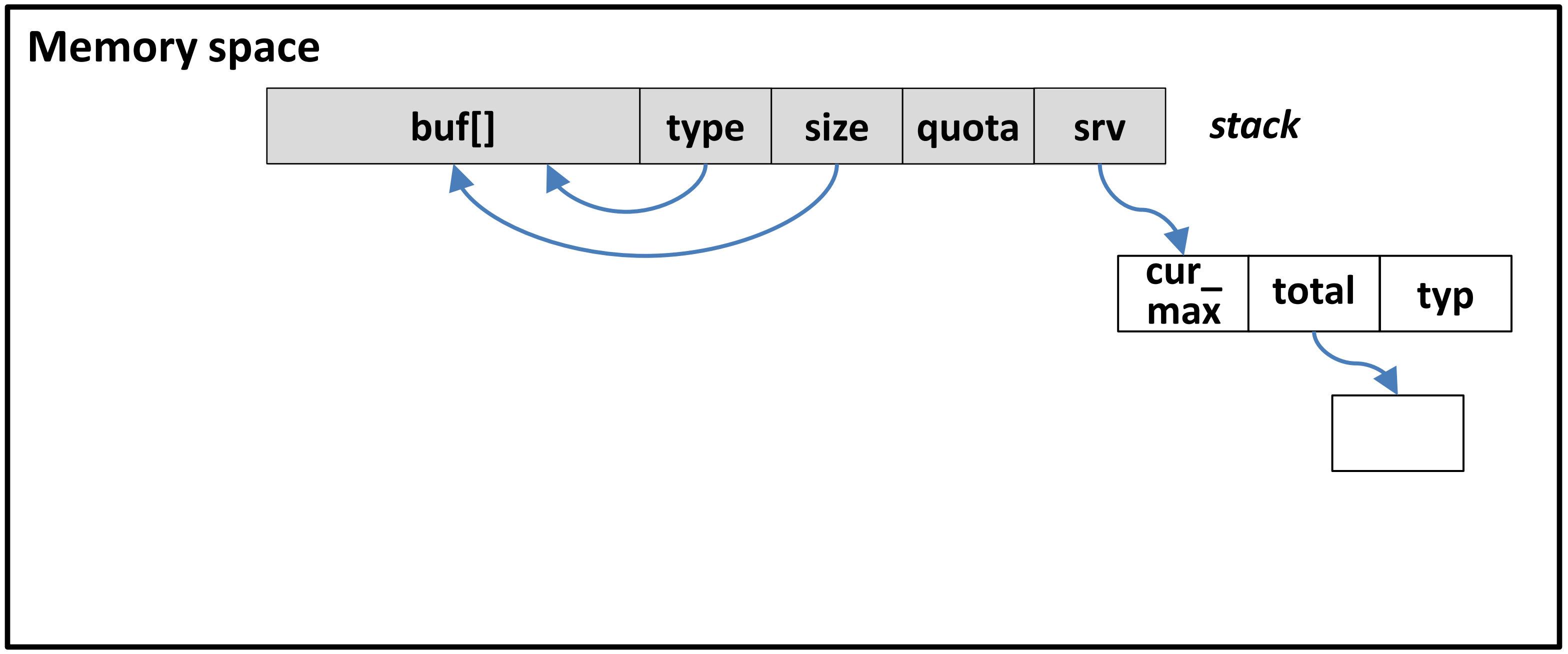
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation



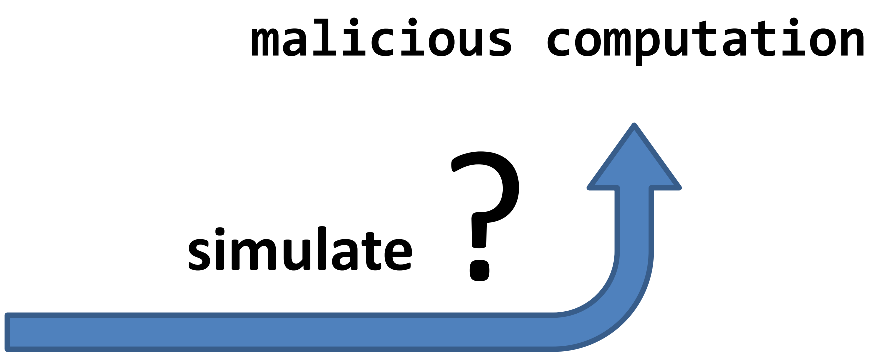


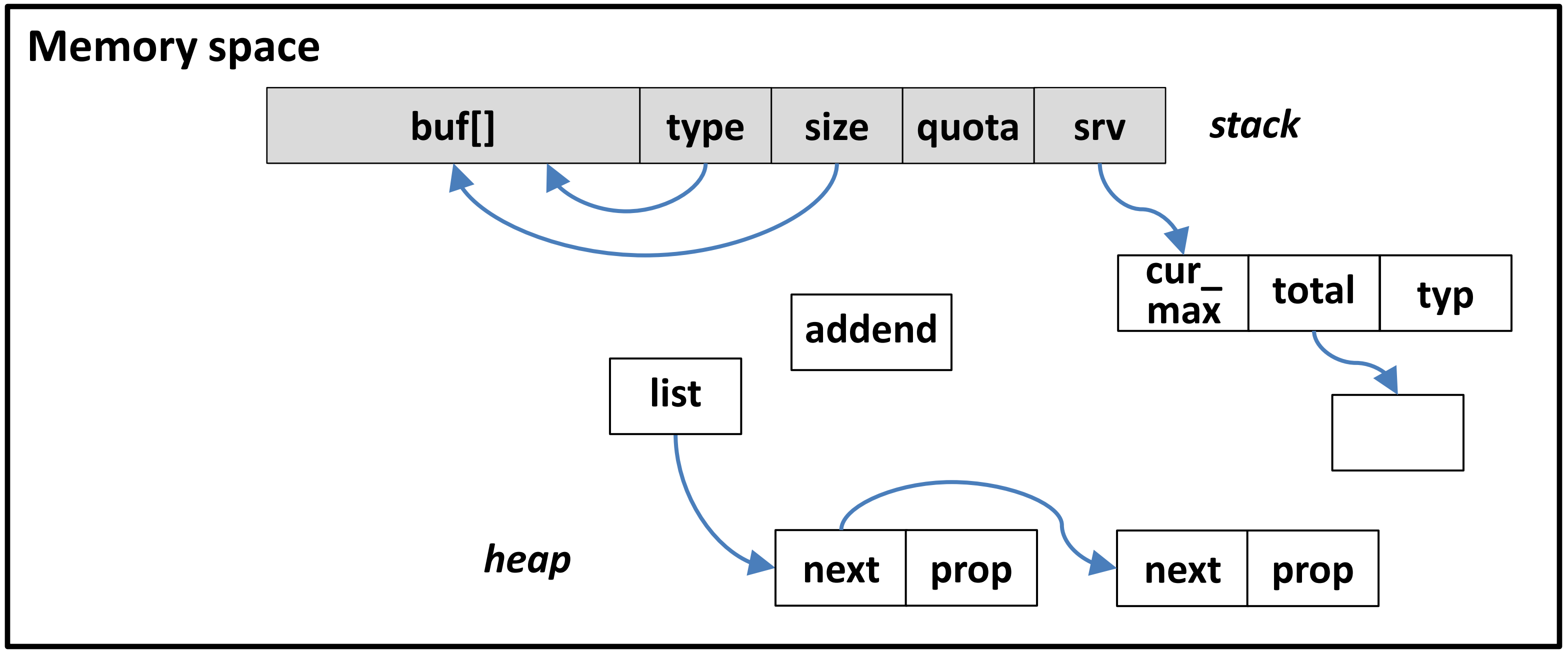
Motivating Example (cont.)

```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```





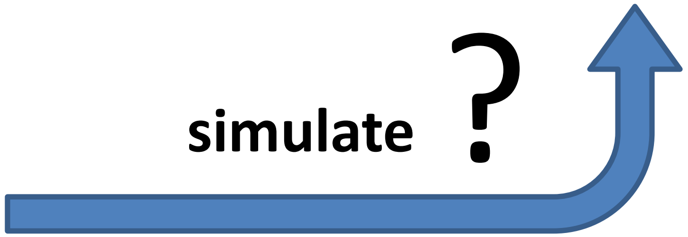
Motivating Example (cont.)

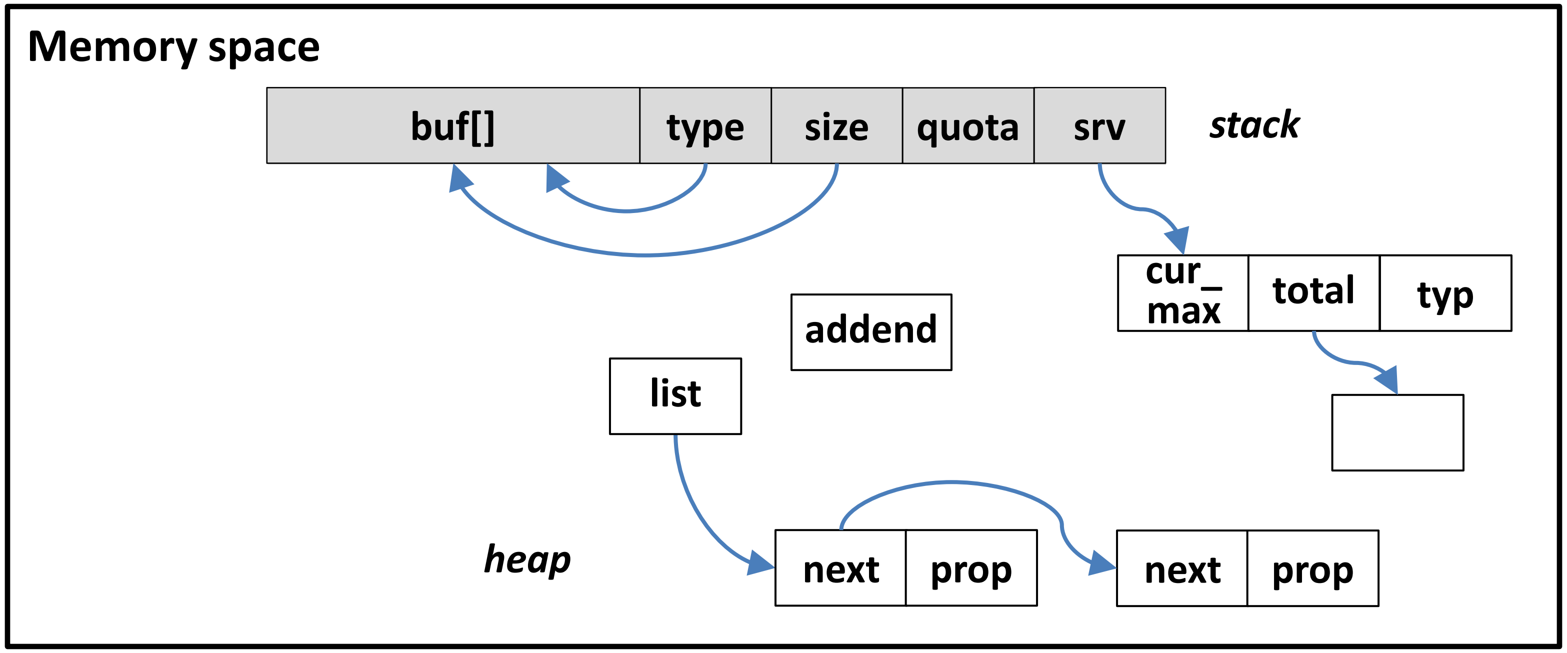
```
6  while (quota--) {
7     readData(sockfd, buf);
8     if(*type == NONE ) break;
9     if(*type == STREAM)
10        *size = *(srv->cur_max);
11    else {
12        srv->typ = *type;
13        srv->total += *size;
14    }
15 }
```

vulnerable program

```
4 for(; list != NULL; list = list->next)
5     list->prop += addend;
```

malicious computation





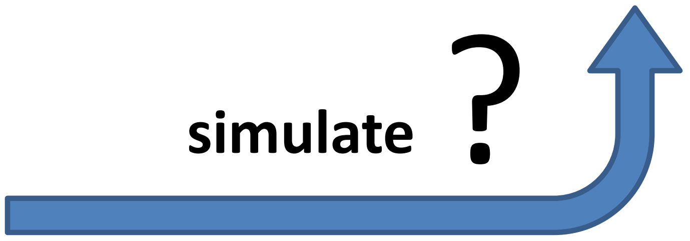
Motivating Example (cont.)

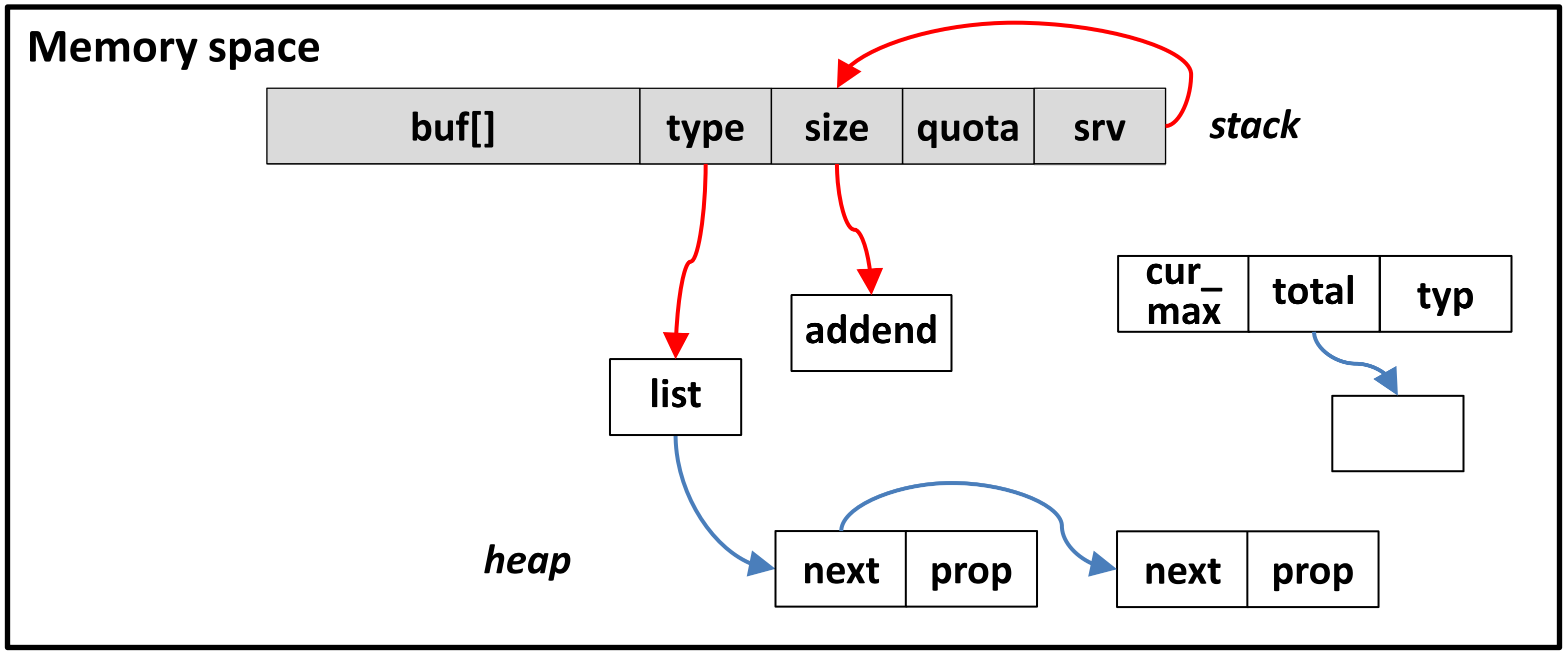
```
while (quota--) {
    readData(sockfd, buf);
    if(*type == NONE ) break;
    if(*type == STREAM)
        *size = *(srv->cur_max);
    else {
        srv->typ = *type;
        srv->total += *size;
    }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





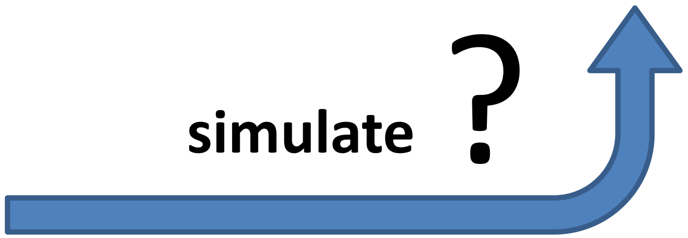
Motivating Example (cont.)

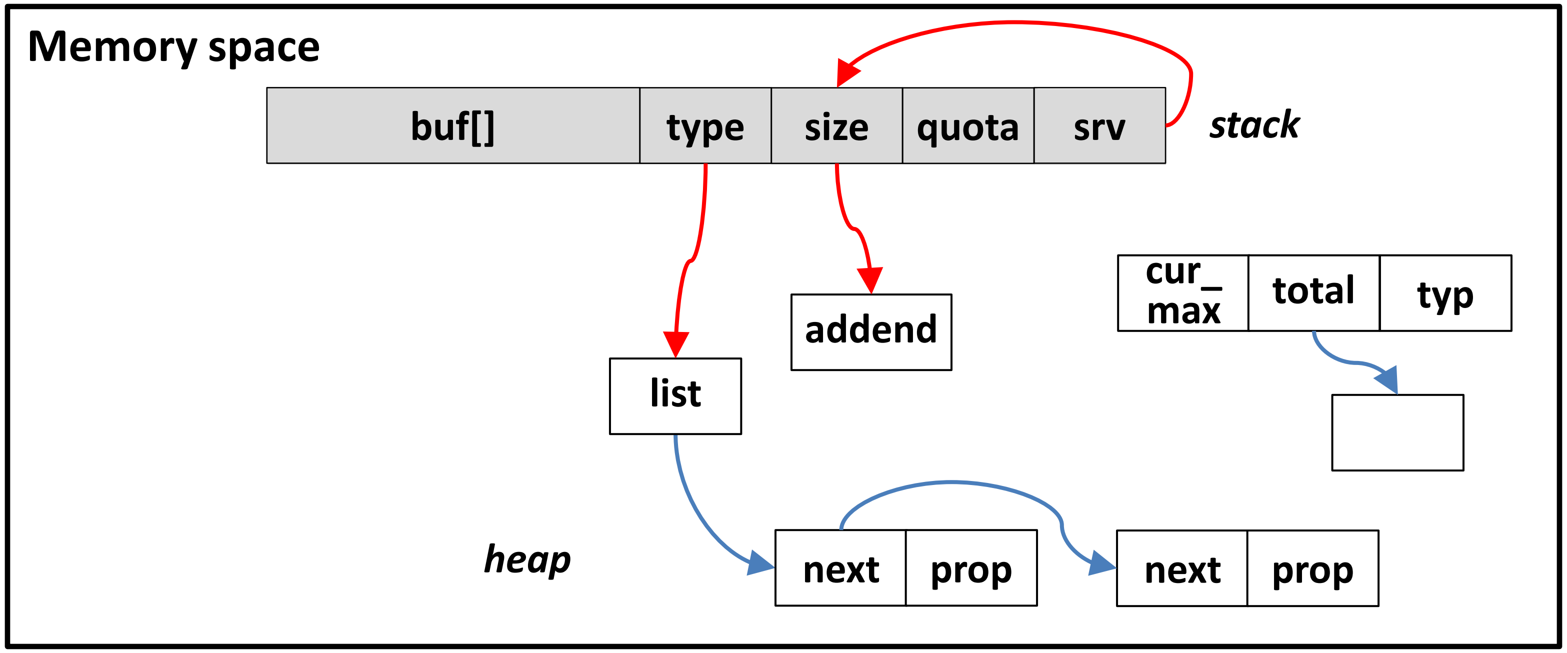
```
while (quota--) {
   readData(sockfd, buf);
   if(*type == NONE ) break;
   if(*type == STREAM)
       *size = *(srv->cur_max);
   else {
       srv->typ = *type;
       srv->total += *size;
   }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





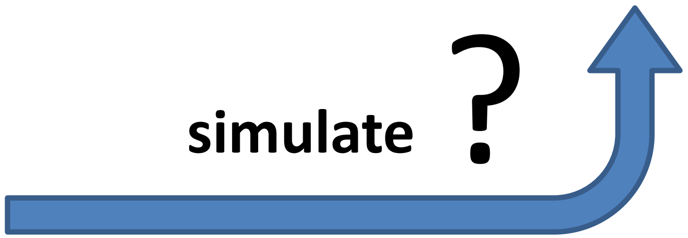
Motivating Example (cont.)

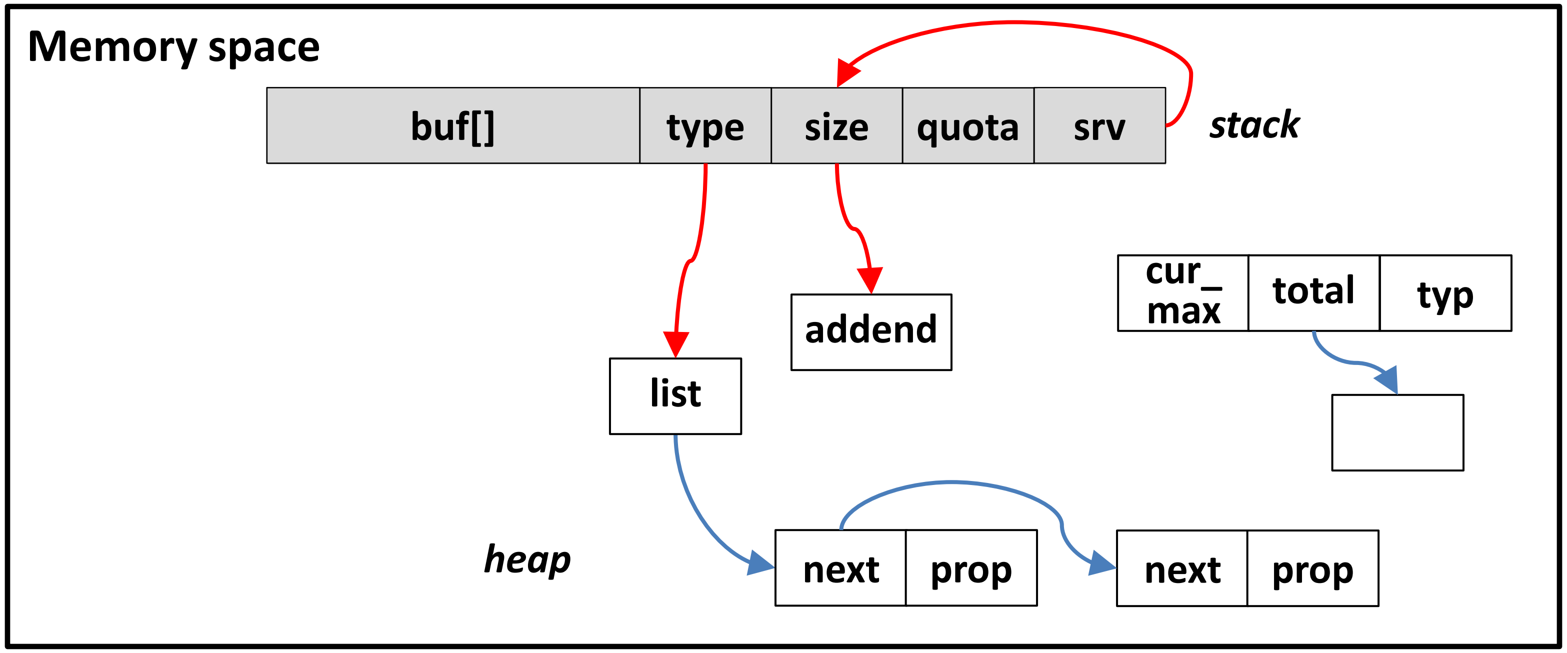
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





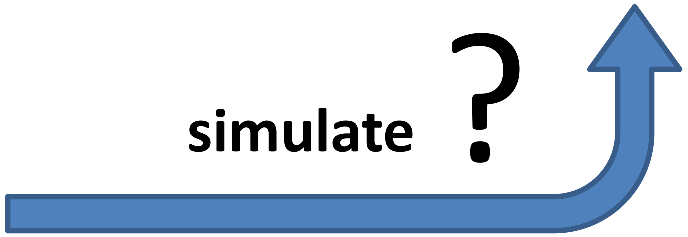
Motivating Example (cont.)

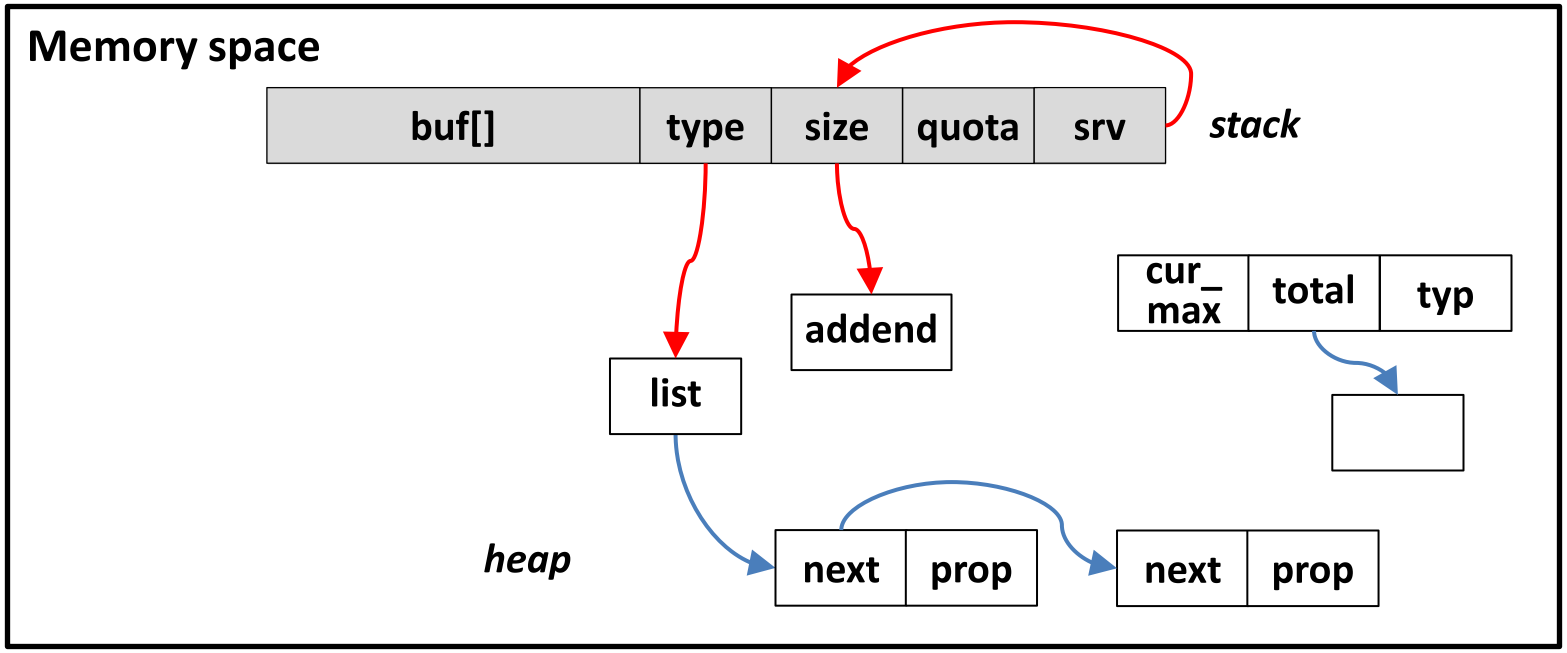
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





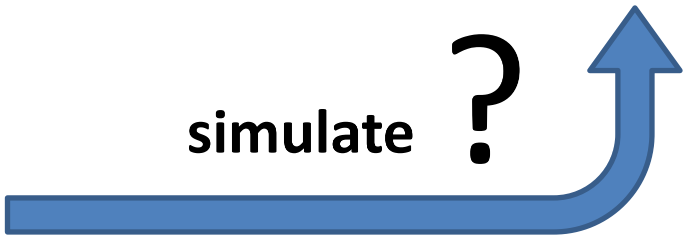
Motivating Example (cont.)

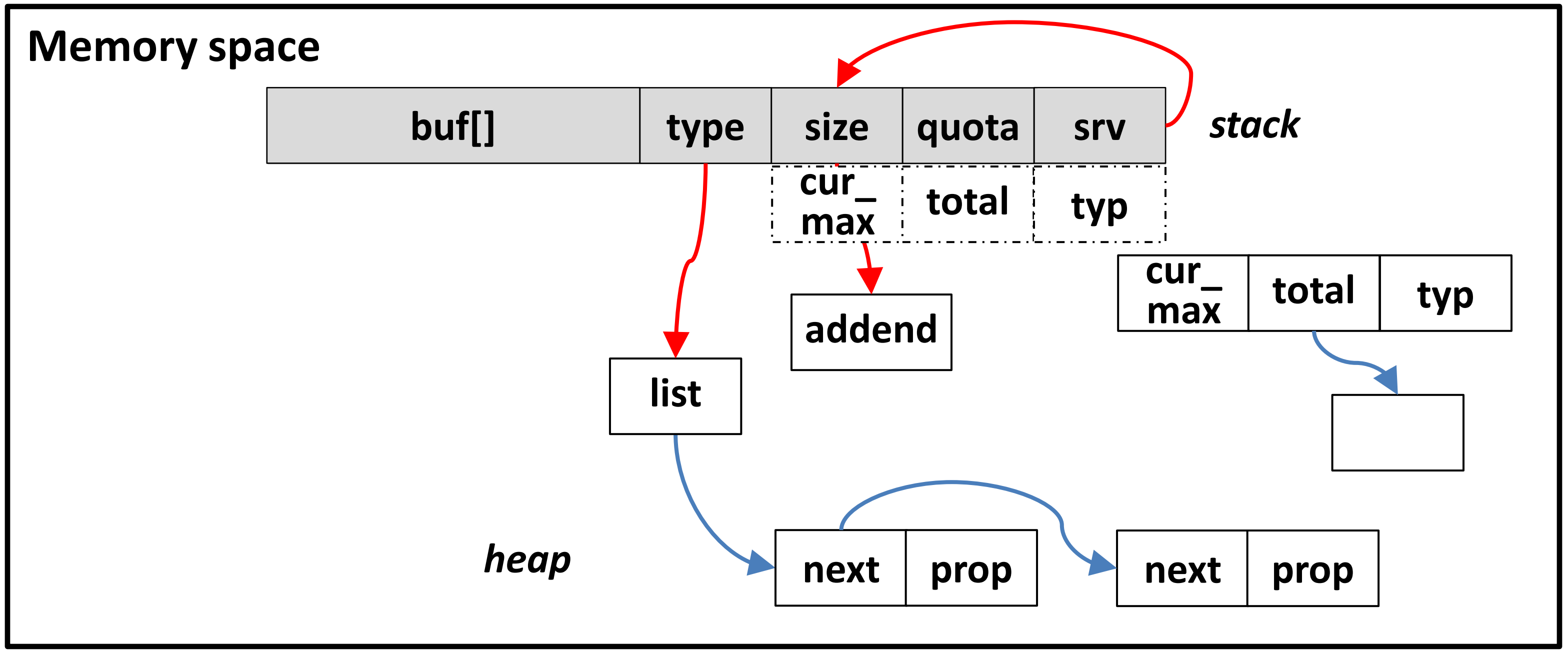
```
6  while (quota--) {
7    readData(sockfd, buf);
8    if(*type == NONE ) break;
9    if(*type == STREAM)
10       *size = *(srv->cur_max);
11   else {
12       srv->typ = *type;
13       srv->total += *size;
14   }
15 }
```

vulnerable program

```
4 for(; list != NULL; list = list->next)
5     list->prop += addend;
```

malicious computation





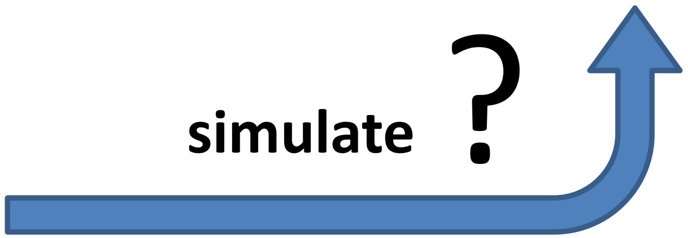
Motivating Example (cont.)

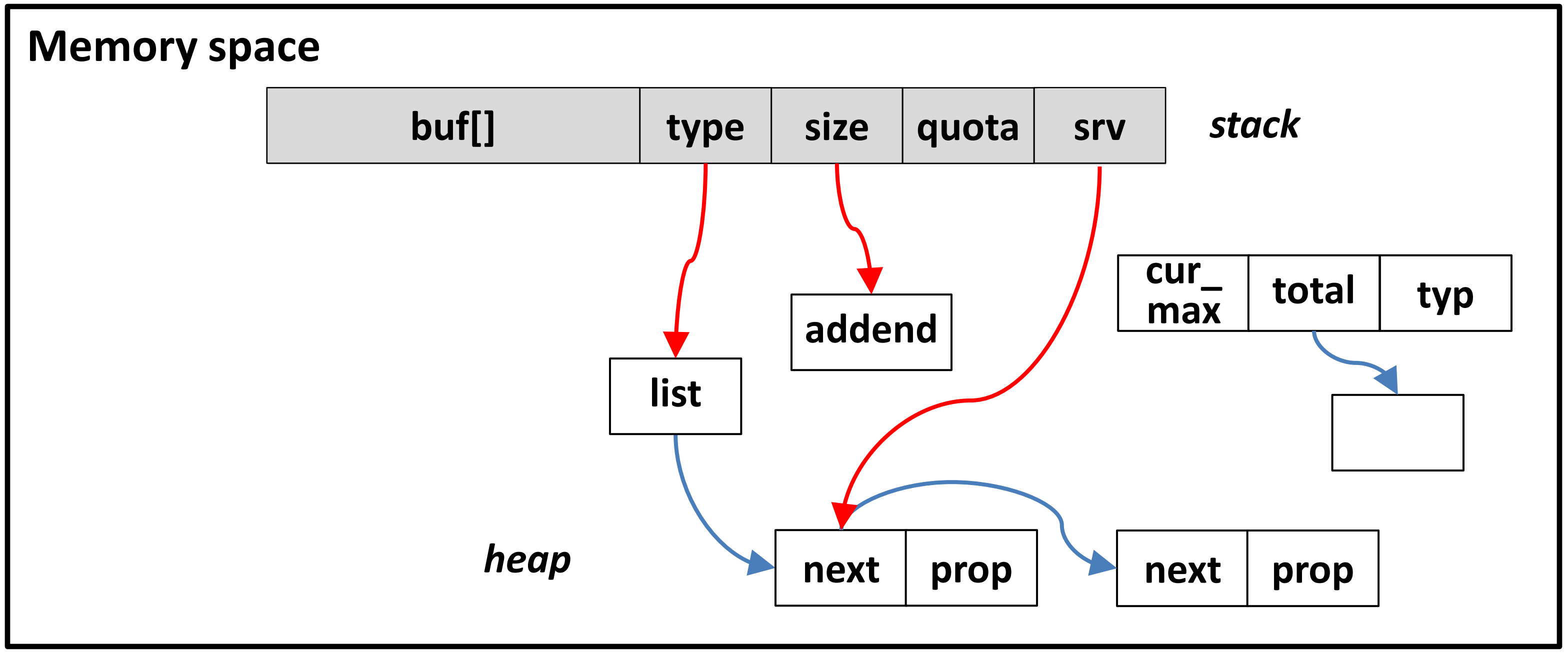
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





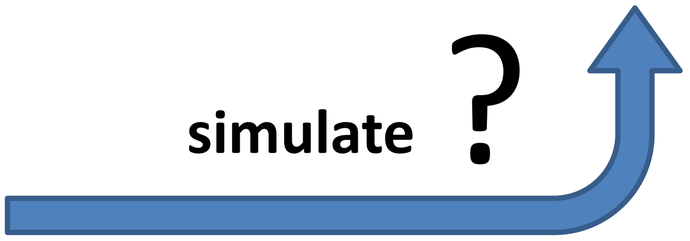
Motivating Example (cont.)

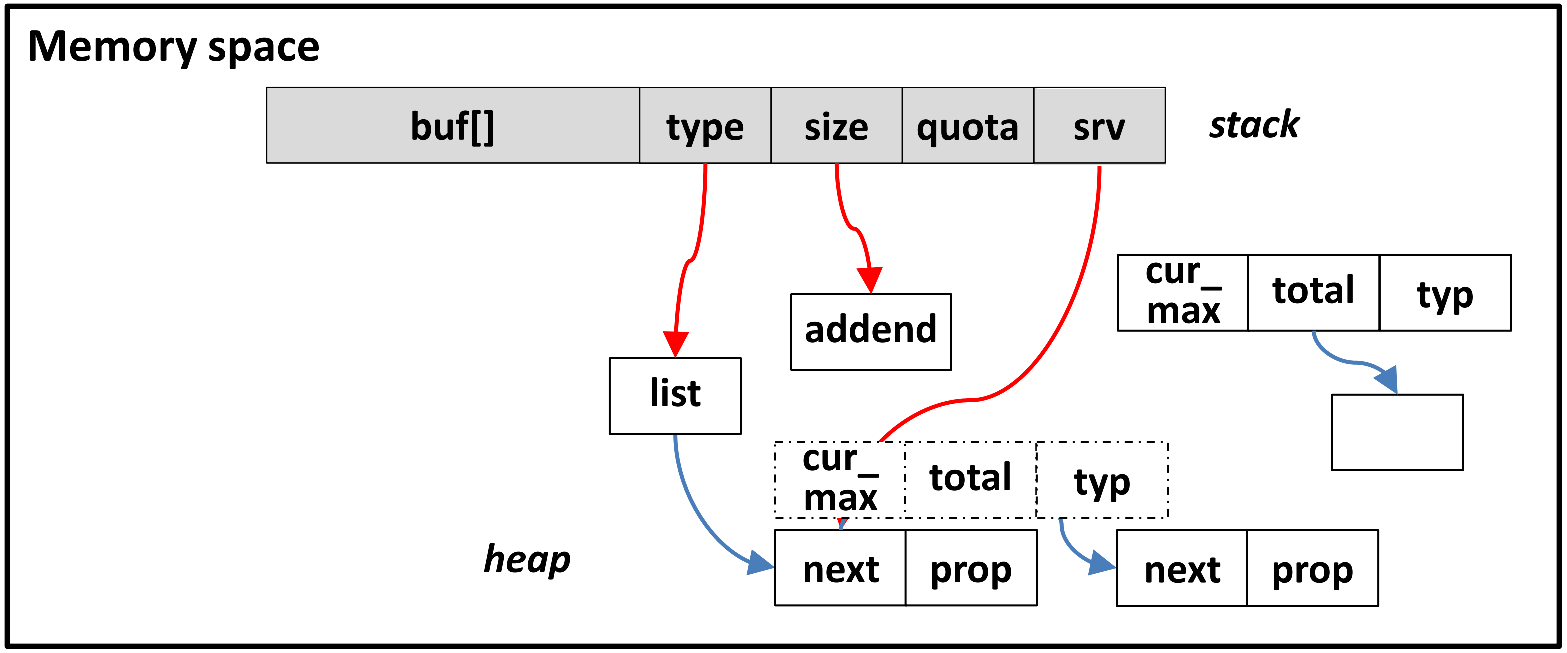
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





Motivating Example (cont.)

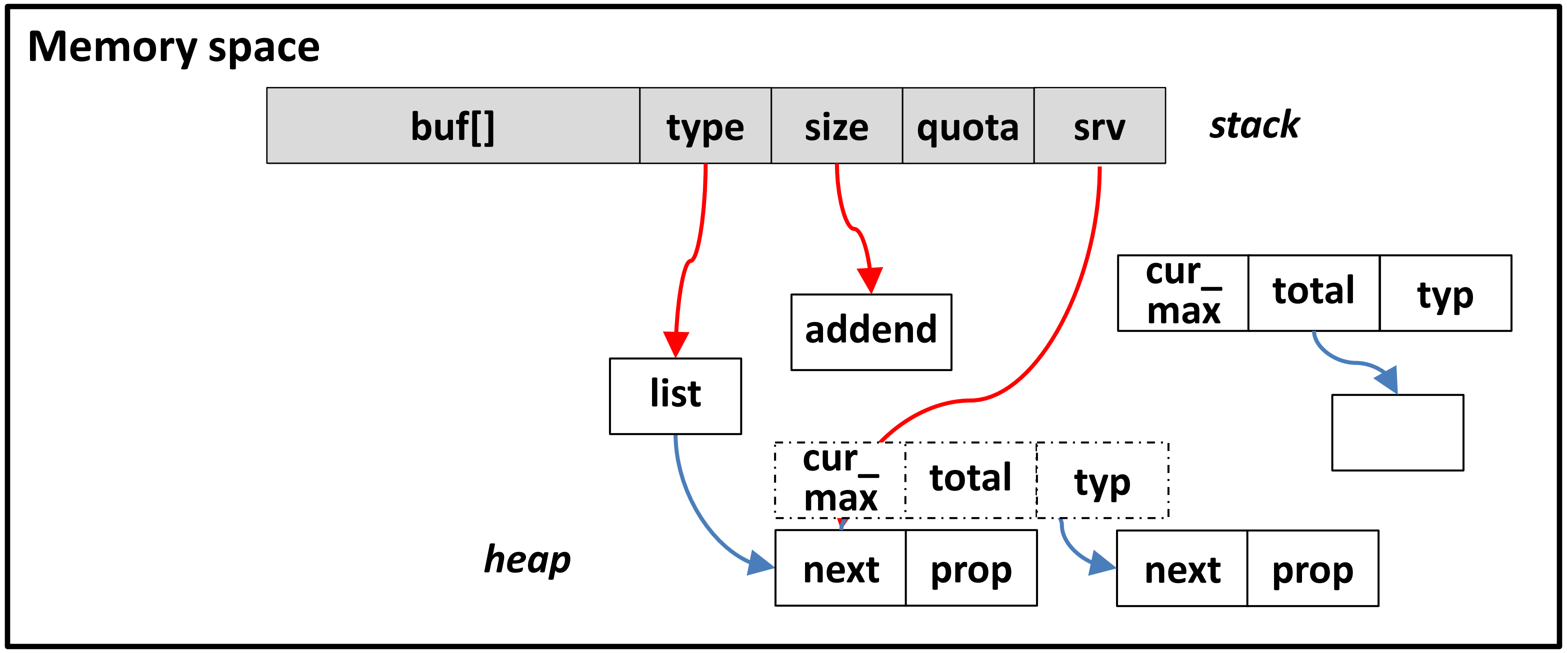
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation

simulate ?



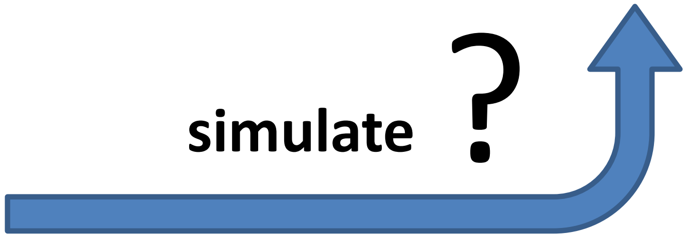
Motivating Example (cont.)

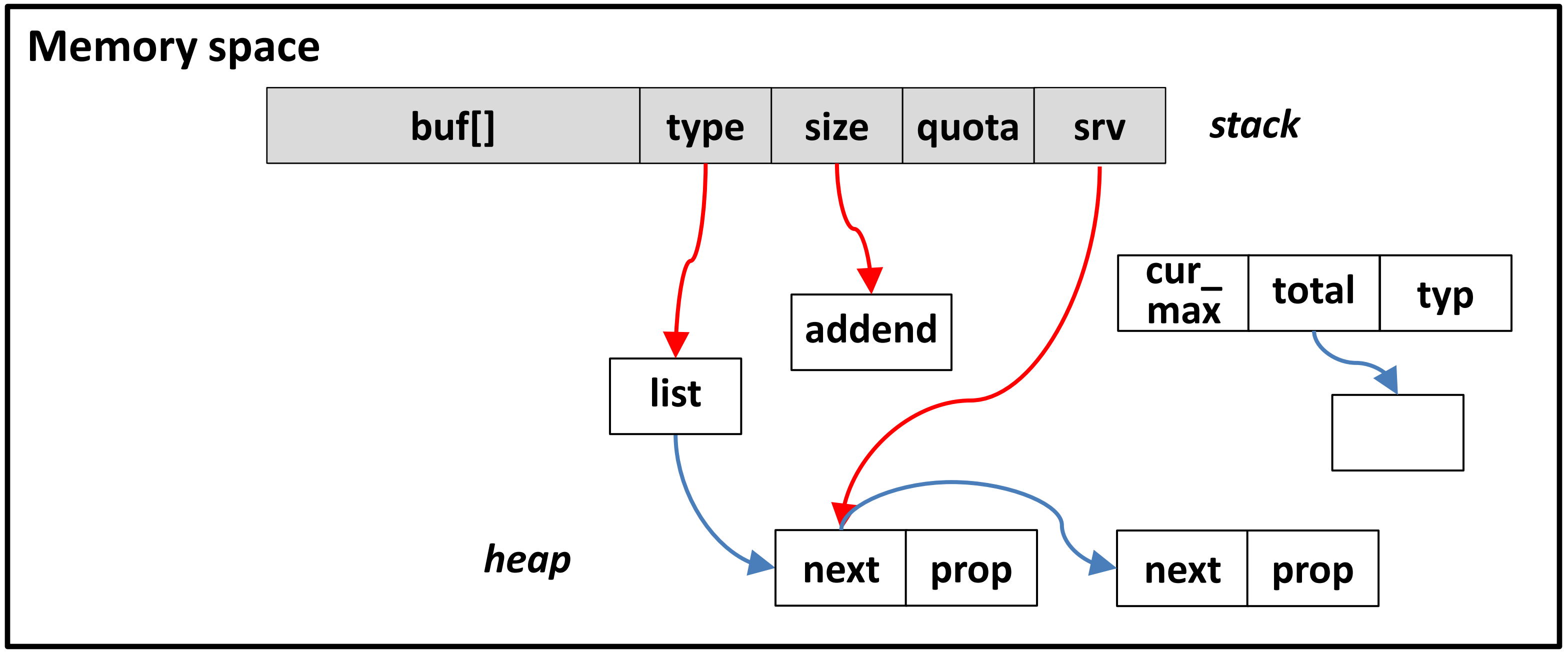
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





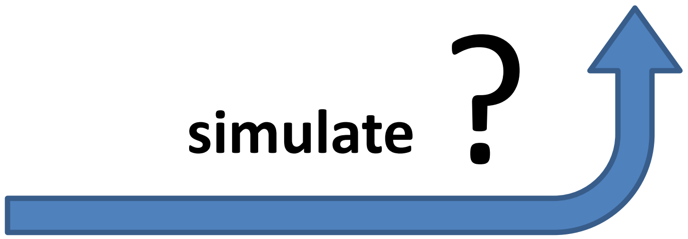
Motivating Example (cont.)

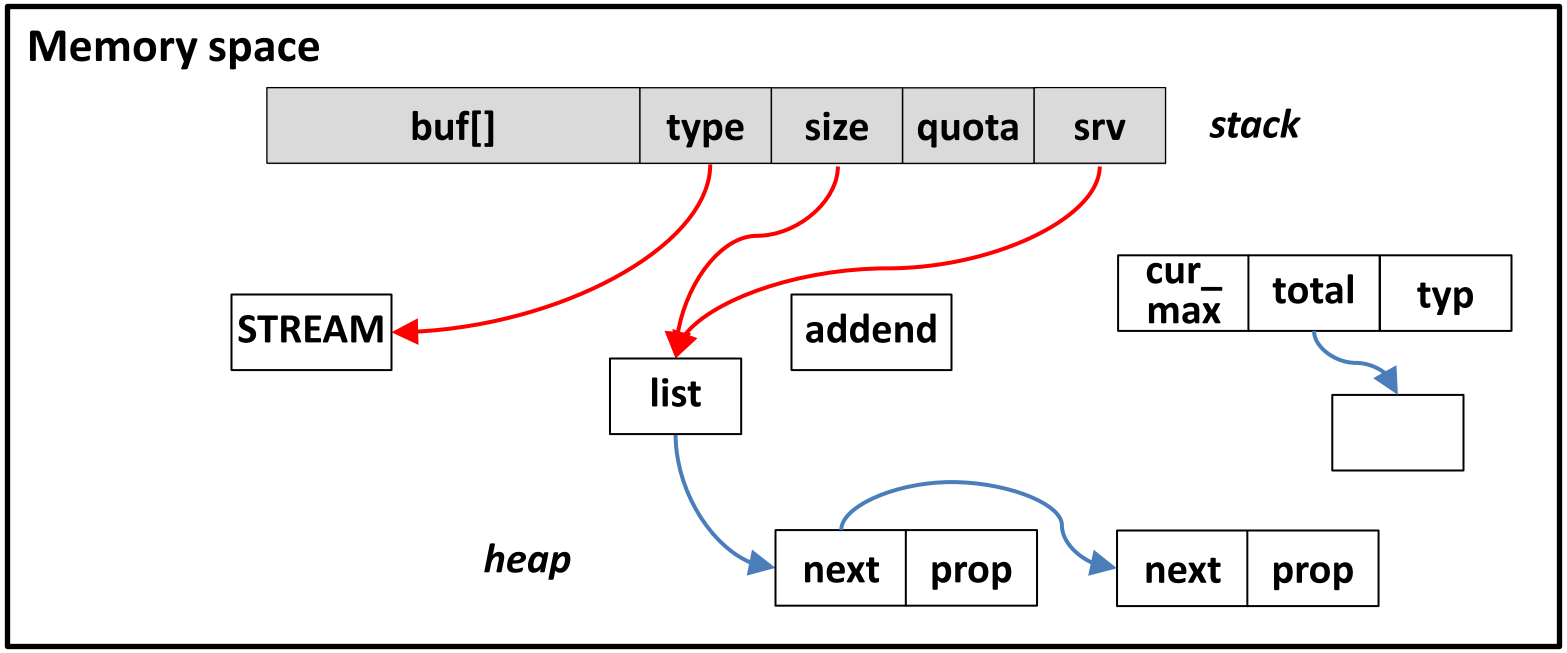
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





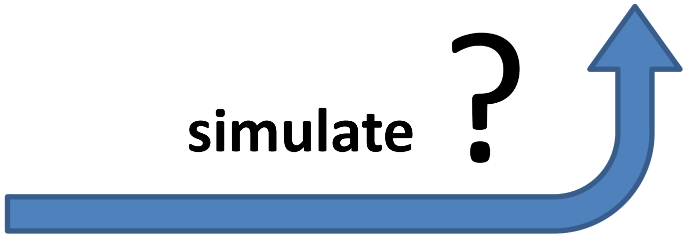
Motivating Example (cont.)

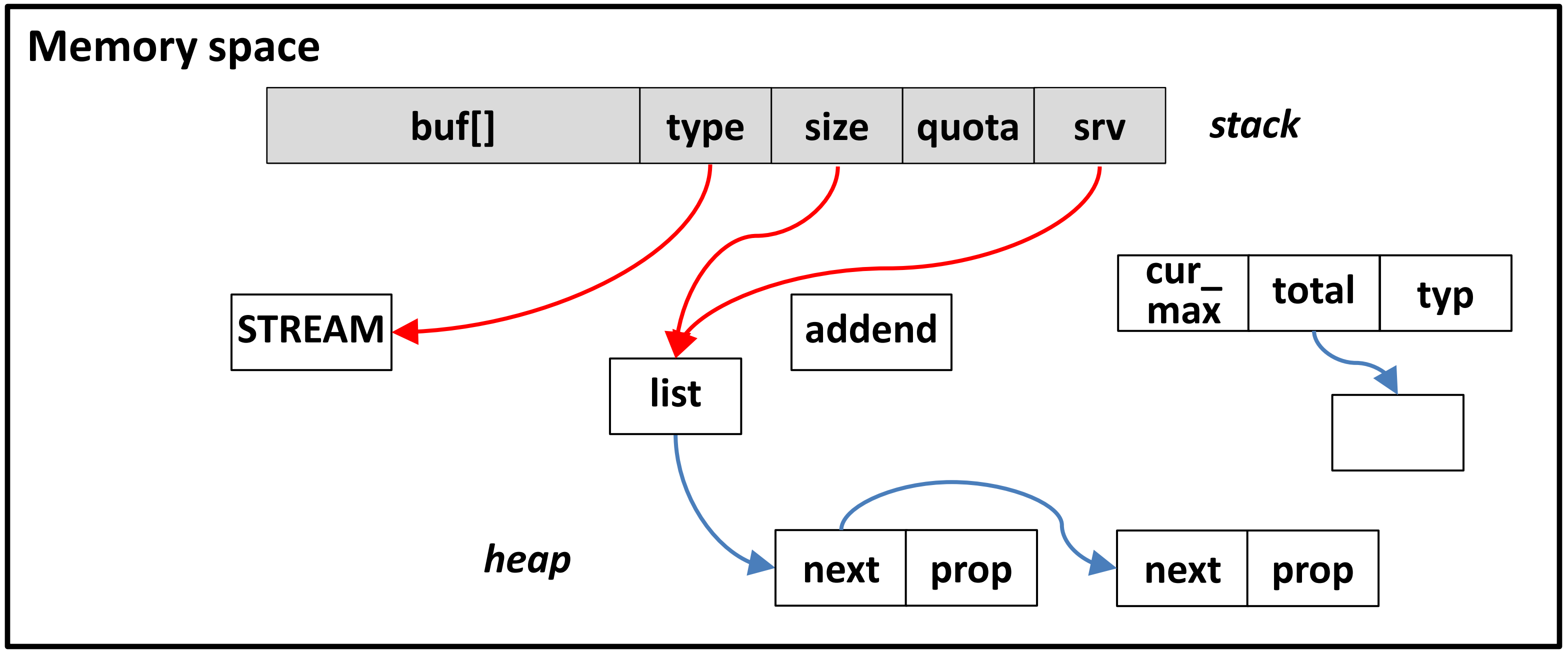
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





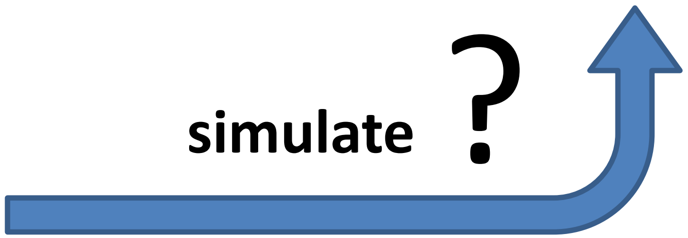
Motivating Example (cont.)

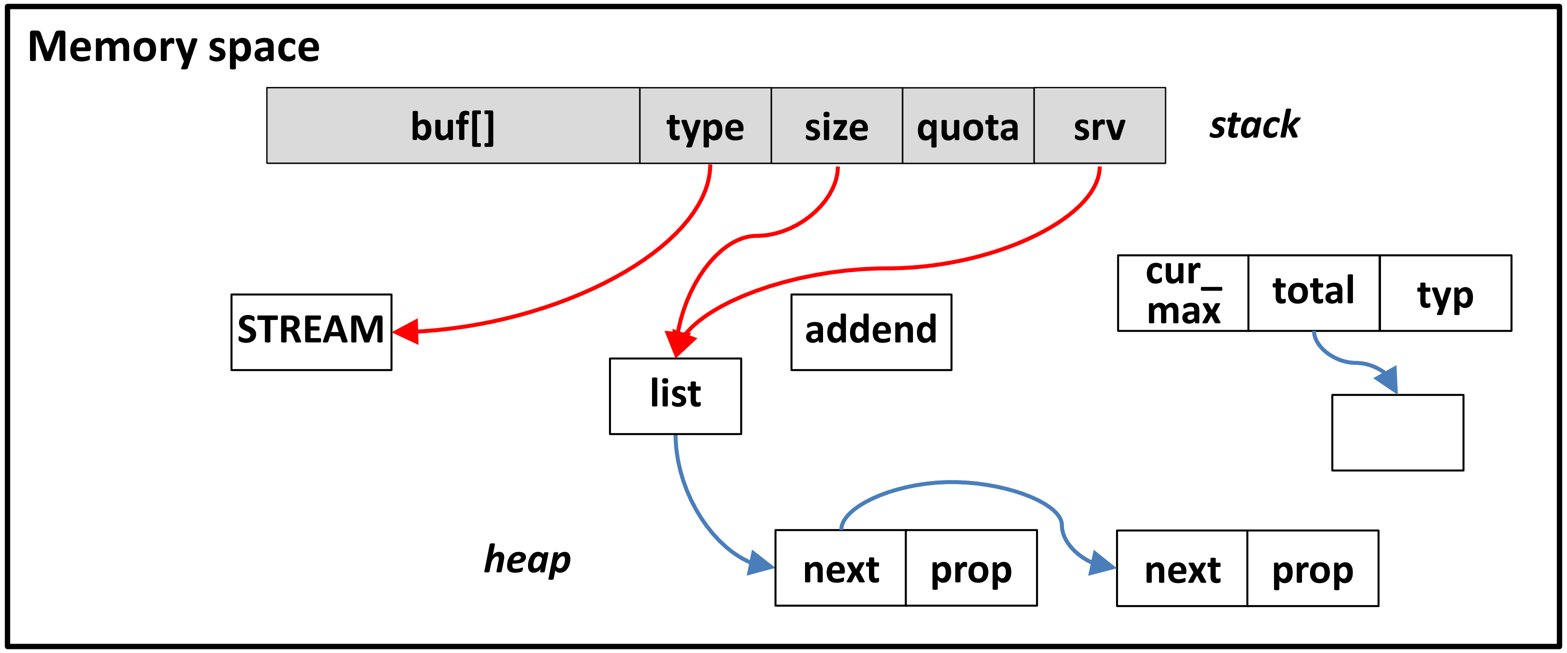
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





Motivating Example (cont.)

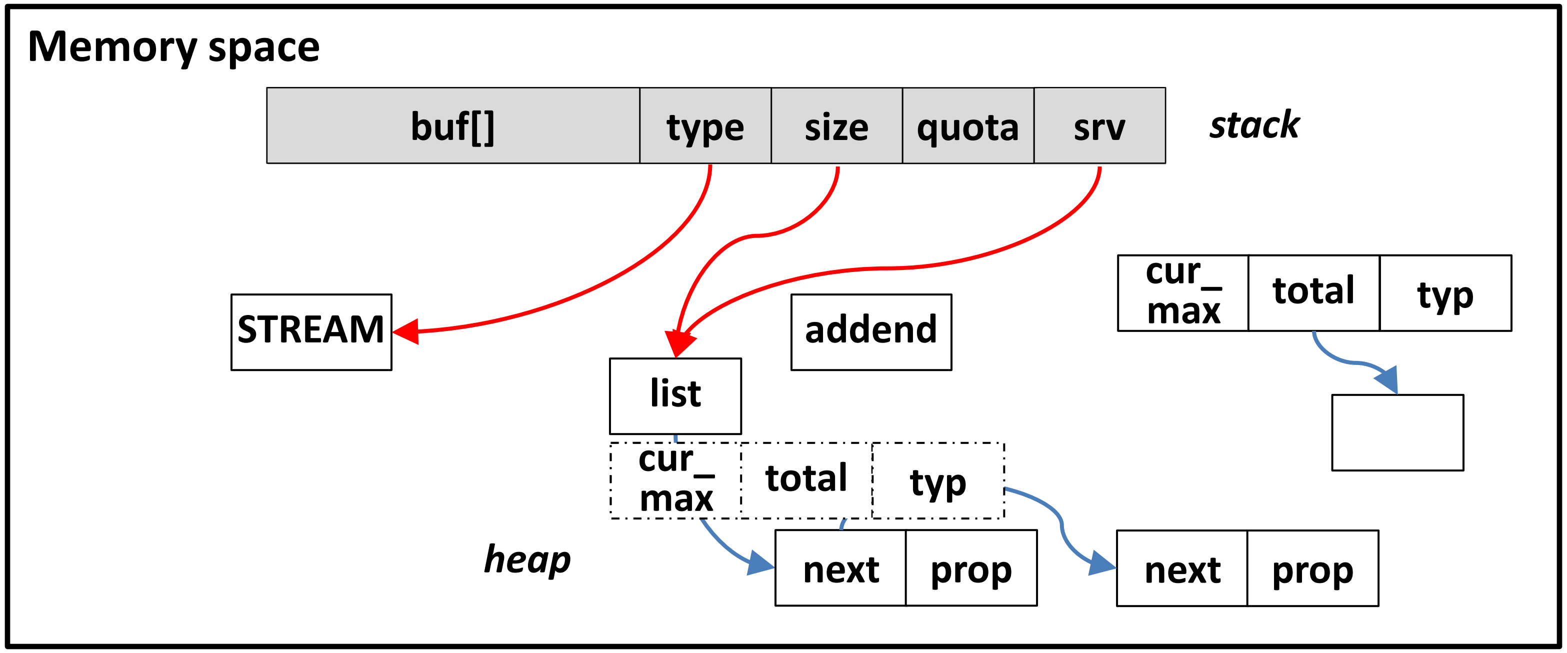
```
6   while (quota--) {
7     readData(sockfd, buf);
8     if(*type == NONE ) break;
9     if(*type == STREAM)
10        *size = *(srv->cur_max);
11    else {
12        srv->typ = *type;
13        srv->total += *size;
14    }
15 }
```

vulnerable program

```
4 for(; list != NULL; list = list->next)
5     list->prop += addend;
```

malicious computation

simulate ?



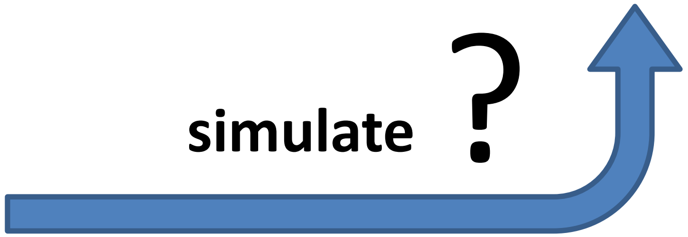
Motivating Example (cont.)

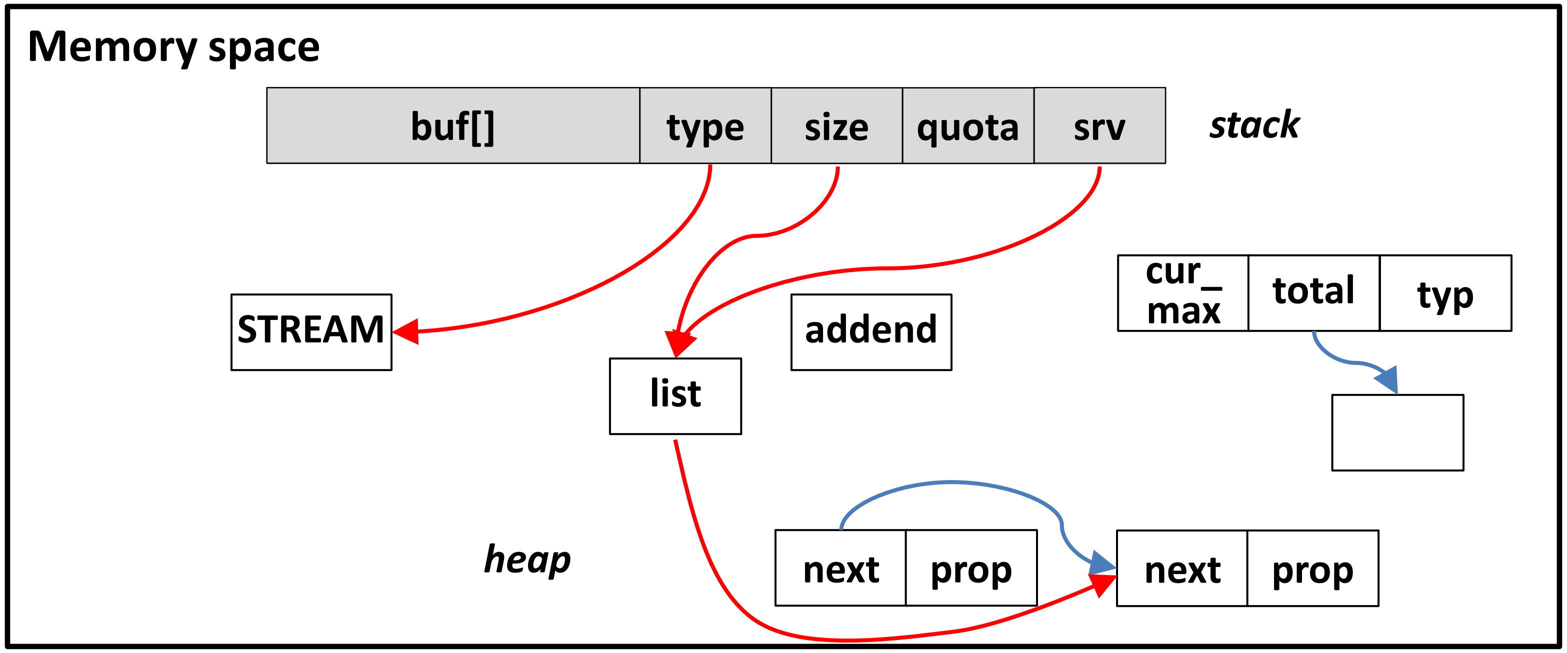
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
      list->prop += addend;
```

malicious computation





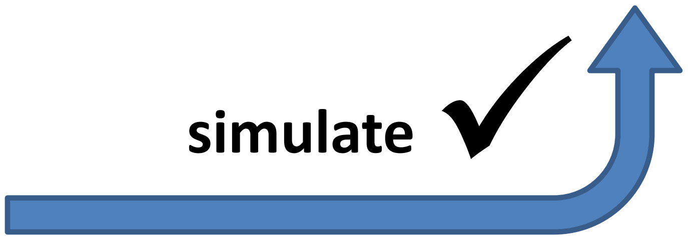
Motivating Example (cont.)

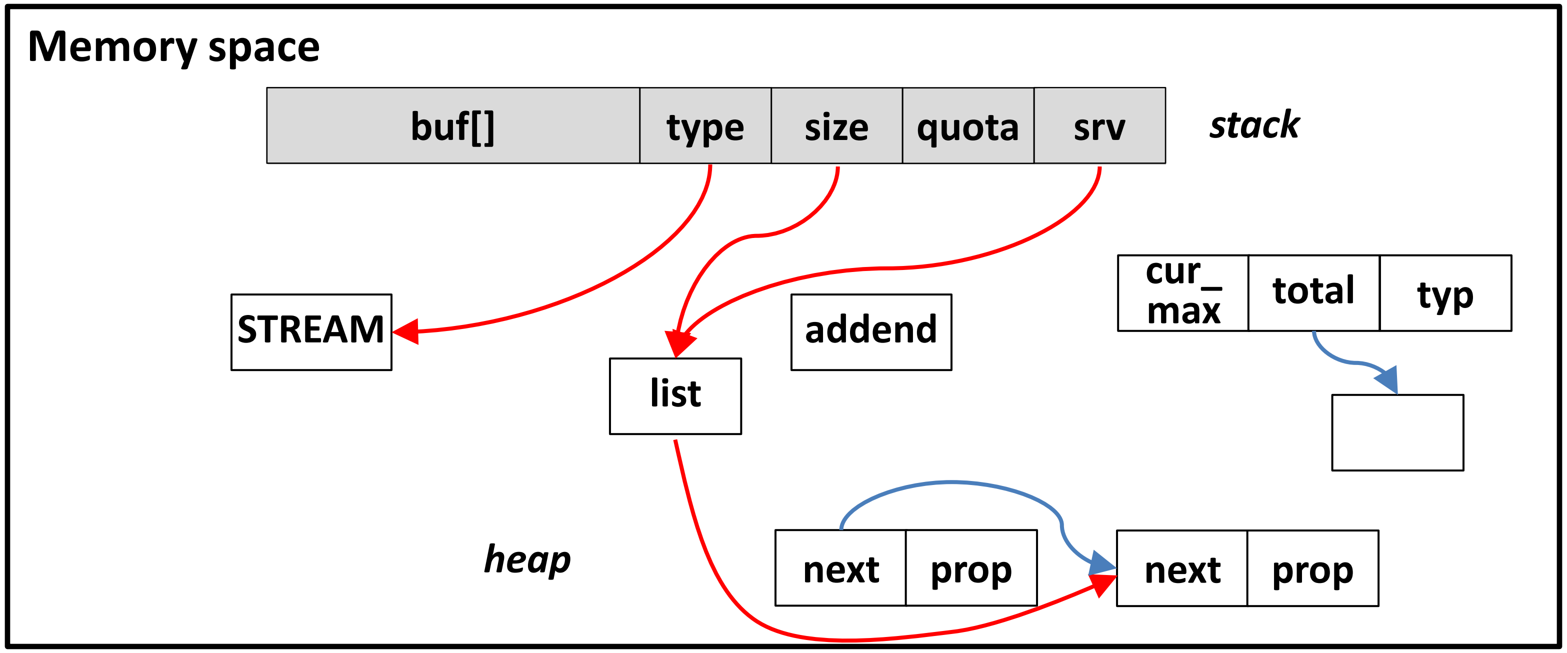
```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
  }
}
```

vulnerable program

```
for(; list != NULL; list = list->next)
    list->prop += addend;
```

malicious computation





Data-Oriented Programming

A Generic Technique

Data-Oriented Programming (DOP)

- General construction
  - w/o dependency on specific data / functions

Data-Oriented Programming (DOP)

- General construction
  - w/o dependency on specific data / functions
- Expressive attacks
  - towards Turing-complete computation

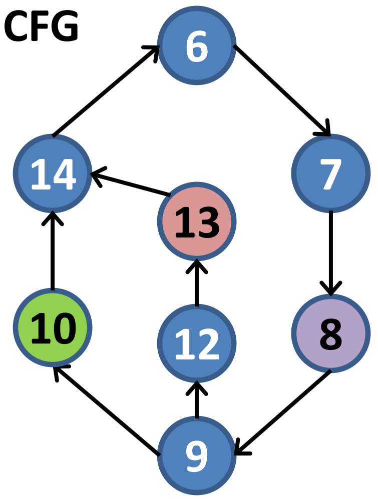
Data-Oriented Programming (DOP)

- General construction
  - w/o dependency on specific data / functions
- Expressive attacks
  - towards Turing-complete computation
- Elements
  - data-oriented gadgets
  - gadget dispatchers

Data-Oriented Gadgets

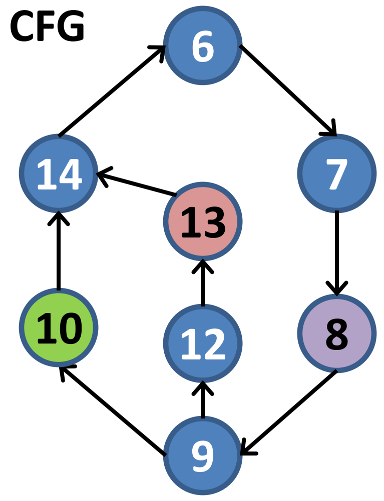
- x86 instruction sequence
  - show in normal execution (CFI)

CFG


$$\begin{tikzcd}
 & 6 \arrow[dr] & \\
14 \arrow[ur] & 13 \arrow[l] & 7 \arrow[d] \\
10 \arrow[u] & 12 \arrow[u] & 8 \arrow[dl] \\
 & 9 \arrow[ul] \arrow[u] &
\end{tikzcd}$$

Data-Oriented Gadgets

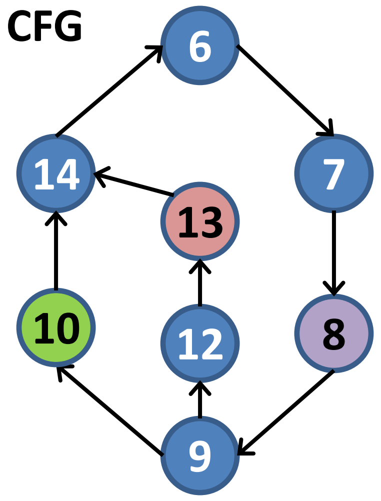
- x86 instruction sequence
  - show in normal execution (CFI)



```
Addition: srv->total += *size;

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

Data-Oriented Gadgets

- x86 instruction sequence
  - show in normal execution (CFI)
  - save results in memory
  - ***load*** *micro-op -->* ***semantics*** *micro-op -->* ***store*** *micro-op*

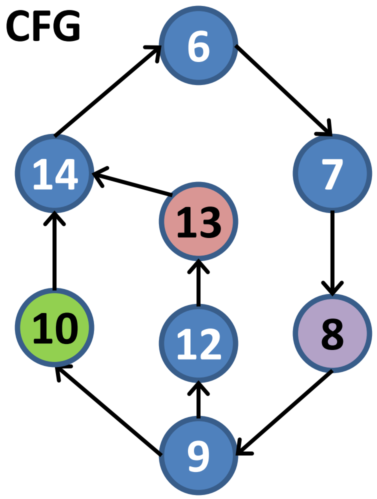


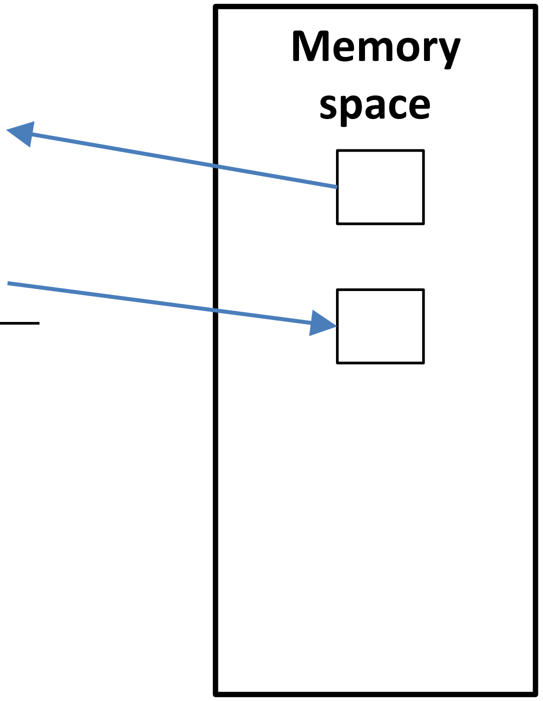
```
Addition: srv->total += *size;

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

Data-Oriented Gadgets

- x86 instruction sequence
  - show in normal execution (CFI)
  - save results in memory
  - ***load*** *micro-op --> **semantics** micro-op --> **store** micro-op*

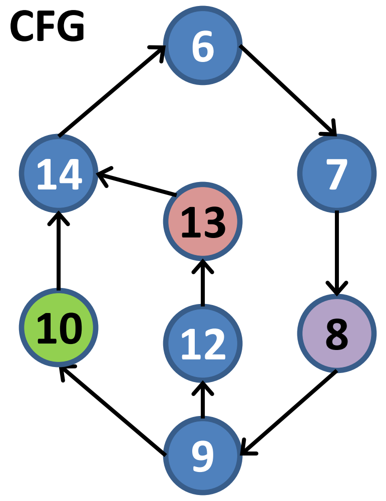
```
Addition: srv->total += *size;

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





Data-Oriented Gadgets

- x86 instruction sequence
  - show in normal execution (CFI)
  - save results in memory
  - ***load*** *micro-op --> **semantics** micro-op --> **store** micro-op*

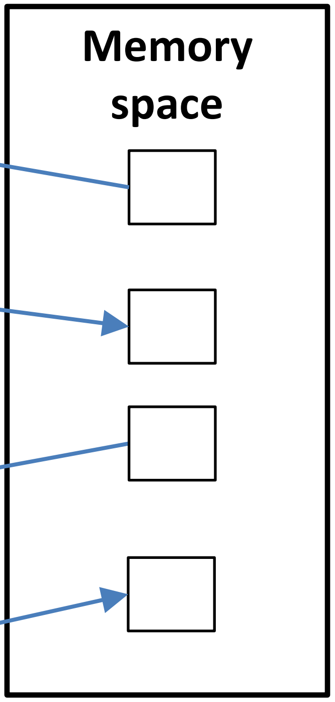


Addition: `srv->total += *size;`

```
mov (%esi), %ebx     //load micro-op
mov (%edi), %eax     //load micro-op
add %ebx, %eax       //addition
mov %eax, (%edi)     //store micro-op
```

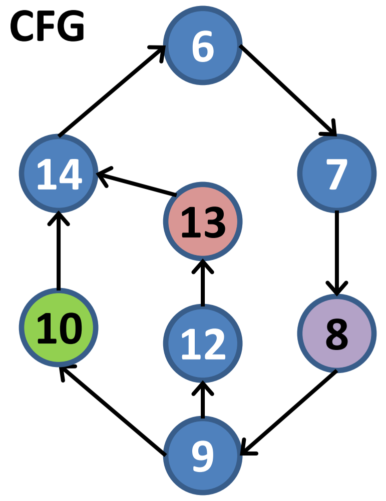
Load: `*size = *(srv ->cur_max);`

```
mov (%esi), %ebx     //load micro-op
mov (%edi), %eax     //load micro-op
mov 0xb(%ebx), %eax  //load
mov %eax, (%edx)     //store micro-op
```



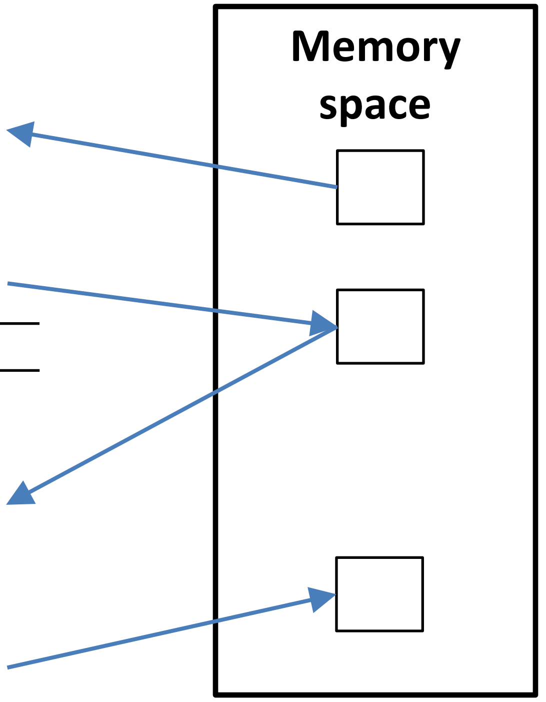
Data-Oriented Gadgets

- x86 instruction sequence
  - show in normal execution (CFI)
  - save results in memory
  - ***load*** *micro-op --> **semantics** micro-op --> **store** micro-op*

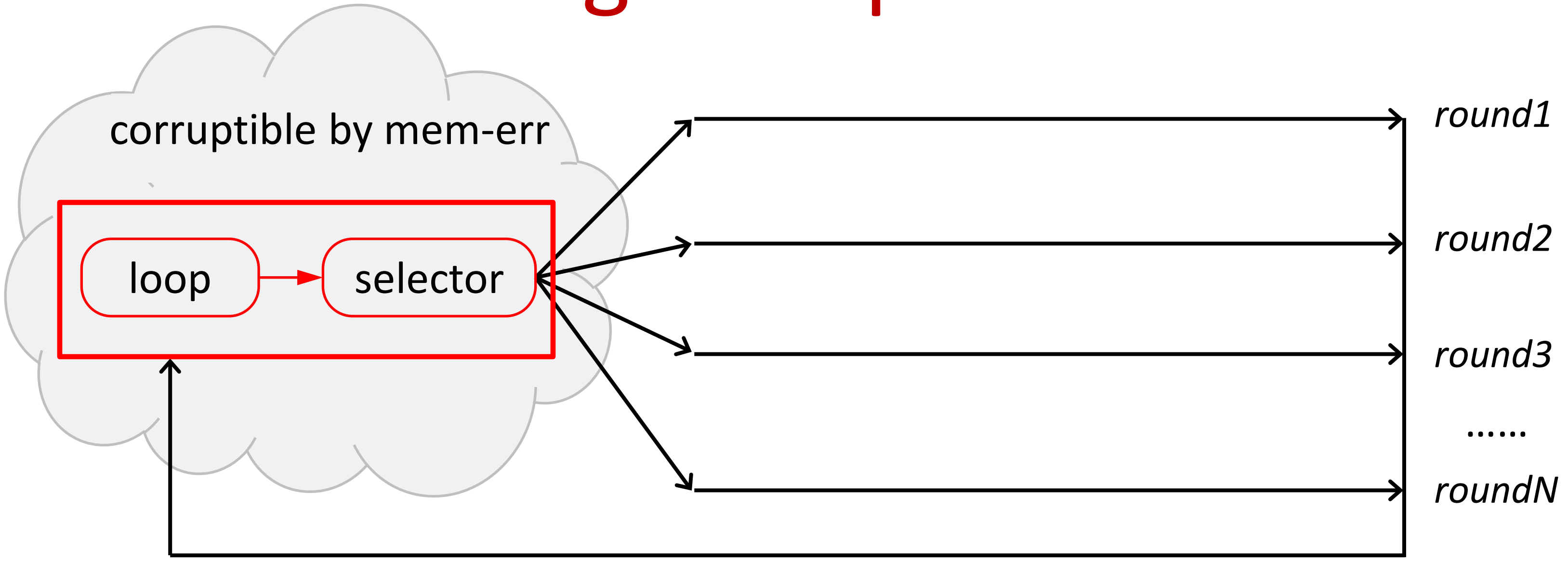


```
Addition: srv->total += *size;

mov (%esi), %ebx      //load micro-op
mov (%edi), %eax      //load micro-op
add %ebx, %eax        //addition
mov %eax, (%edi)      //store micro-op
```

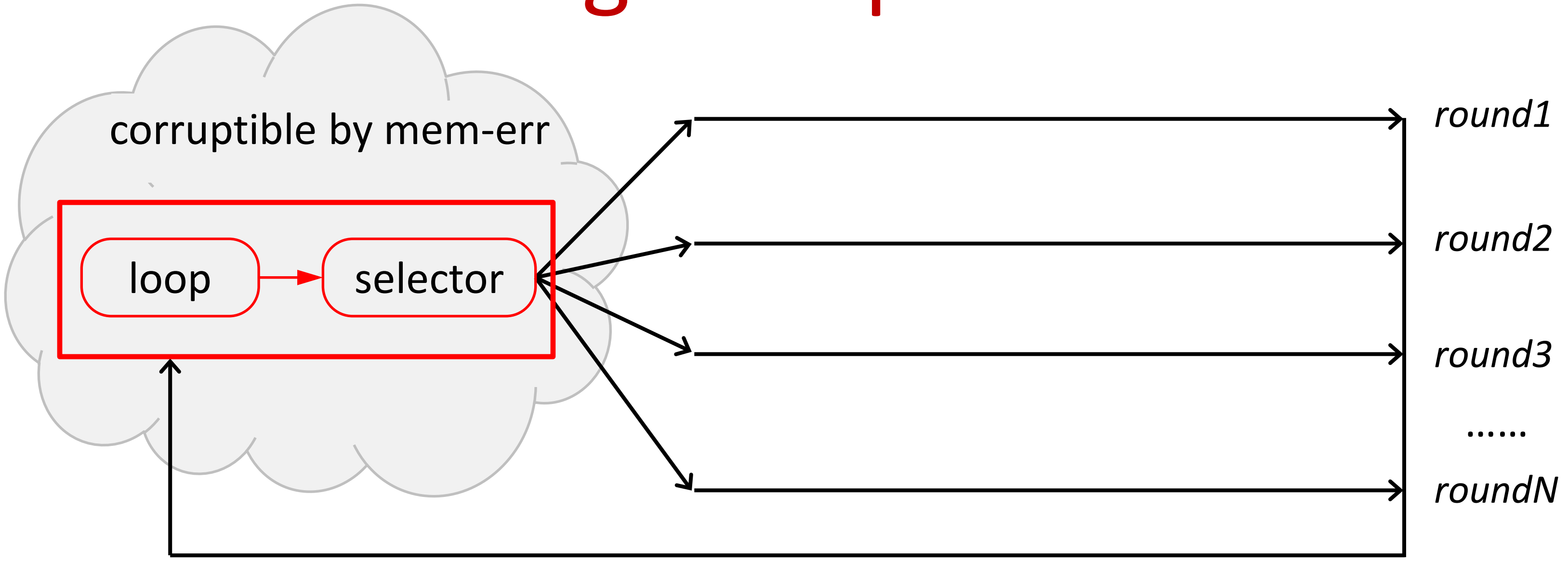
```
Load:     *size = *(srv ->cur_max);

mov (%esi), %ebx      //load micro-op
mov (%edi), %eax      //load micro-op
mov 0xb(%ebx), %eax //load
mov %eax, (%edx)      //store micro-op
```



Gadget Dispatcher

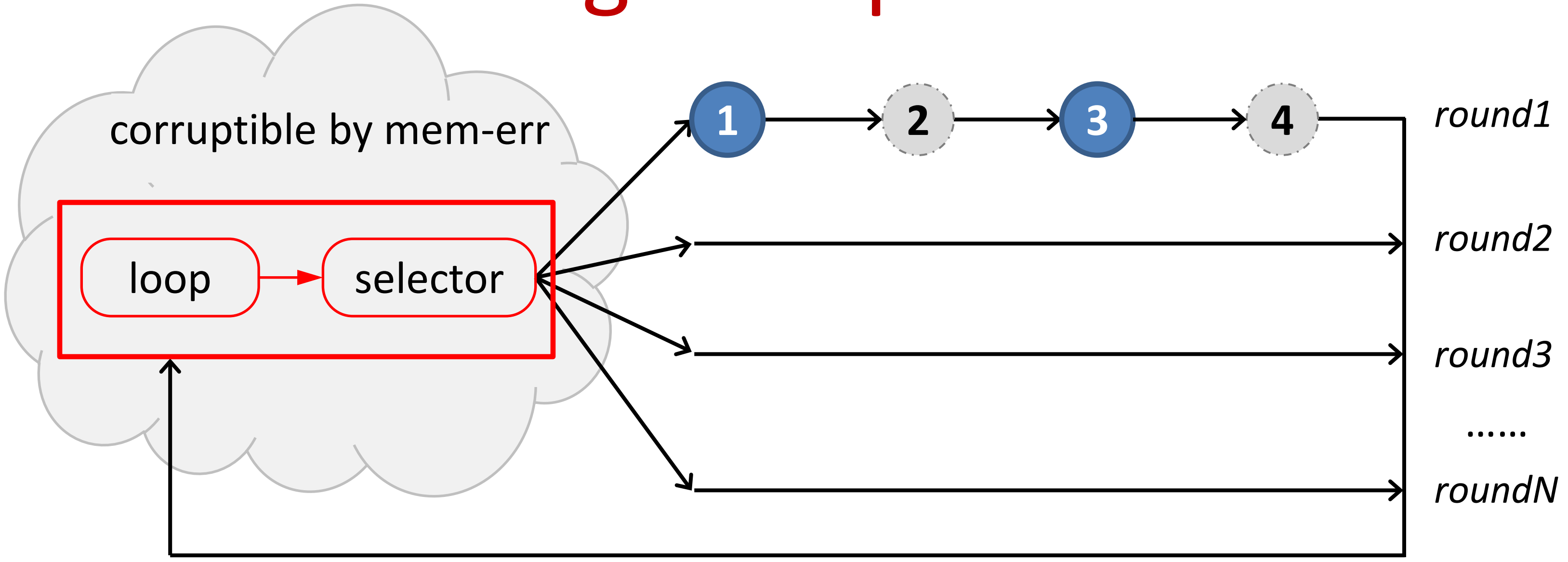


Gadget Dispatcher



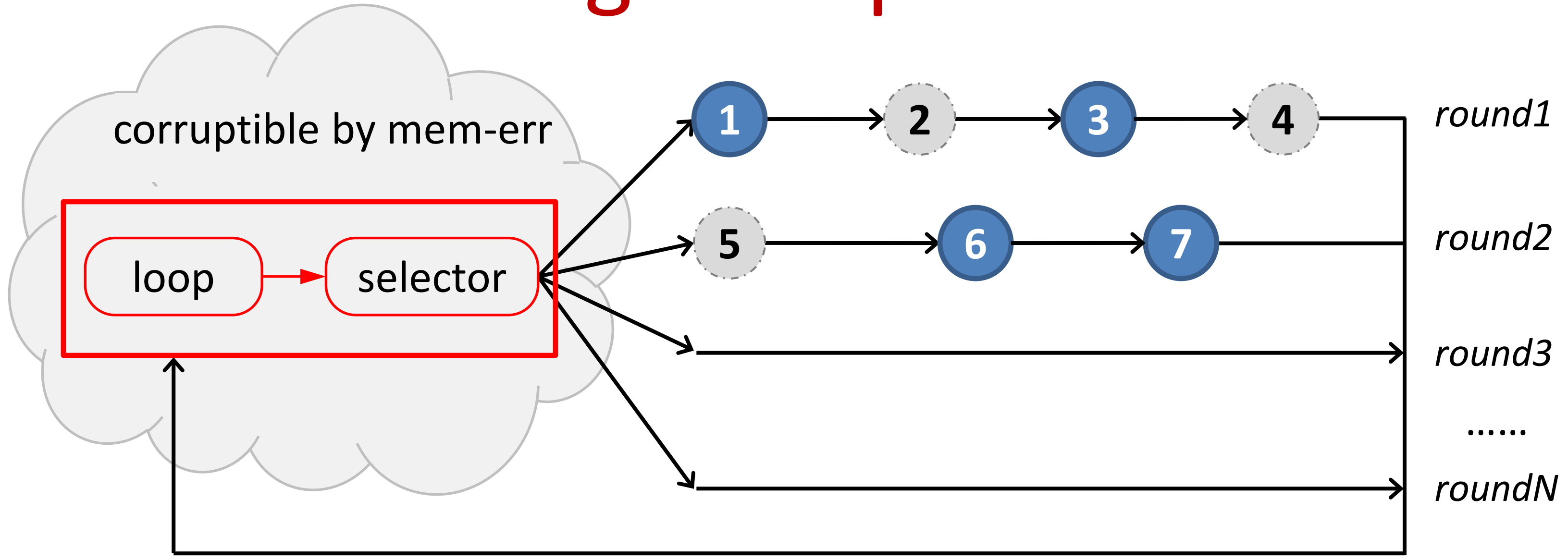
- Chain data-oriented gadgets ***"legitimately"***
  - **loop** ---> repeatedly invoke gadgets
  - **selector** ---> selectively activate gadgets

Gadget Dispatcher



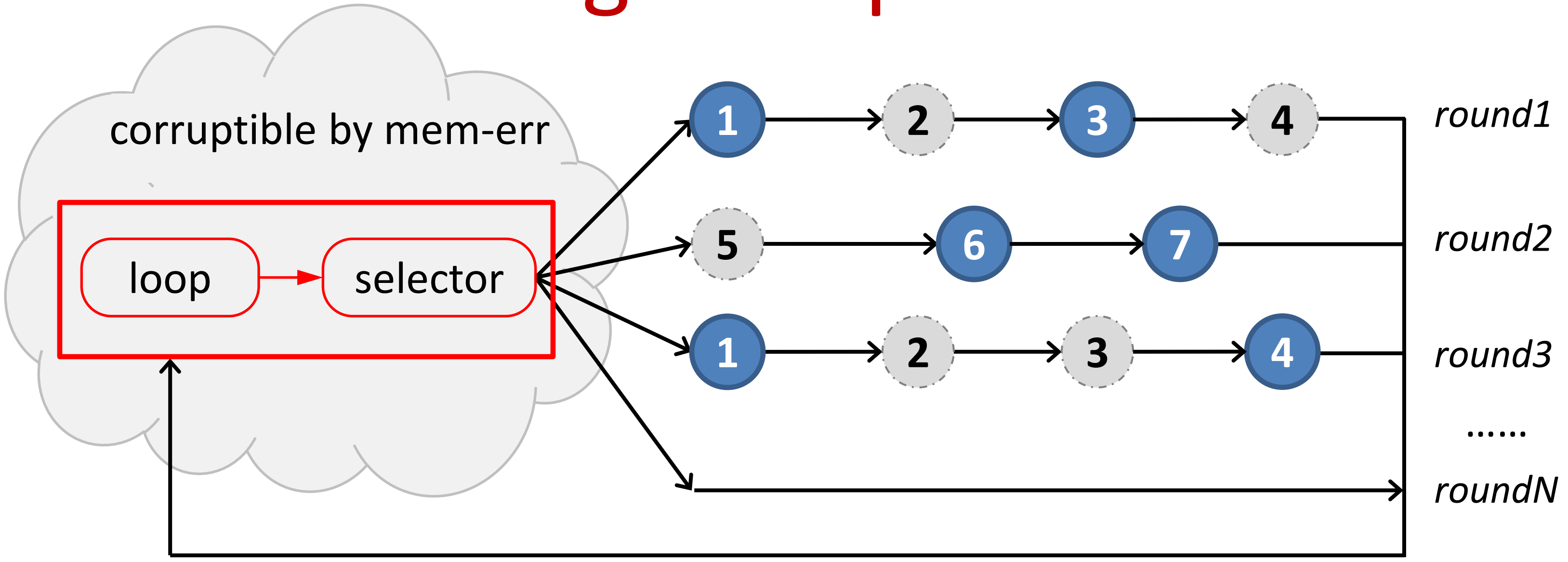
- Chain data-oriented gadgets “***legitimately***”
  - **loop** ---> repeatedly invoke gadgets
  - **selector** ---> selectively activate gadgets

Gadget Dispatcher



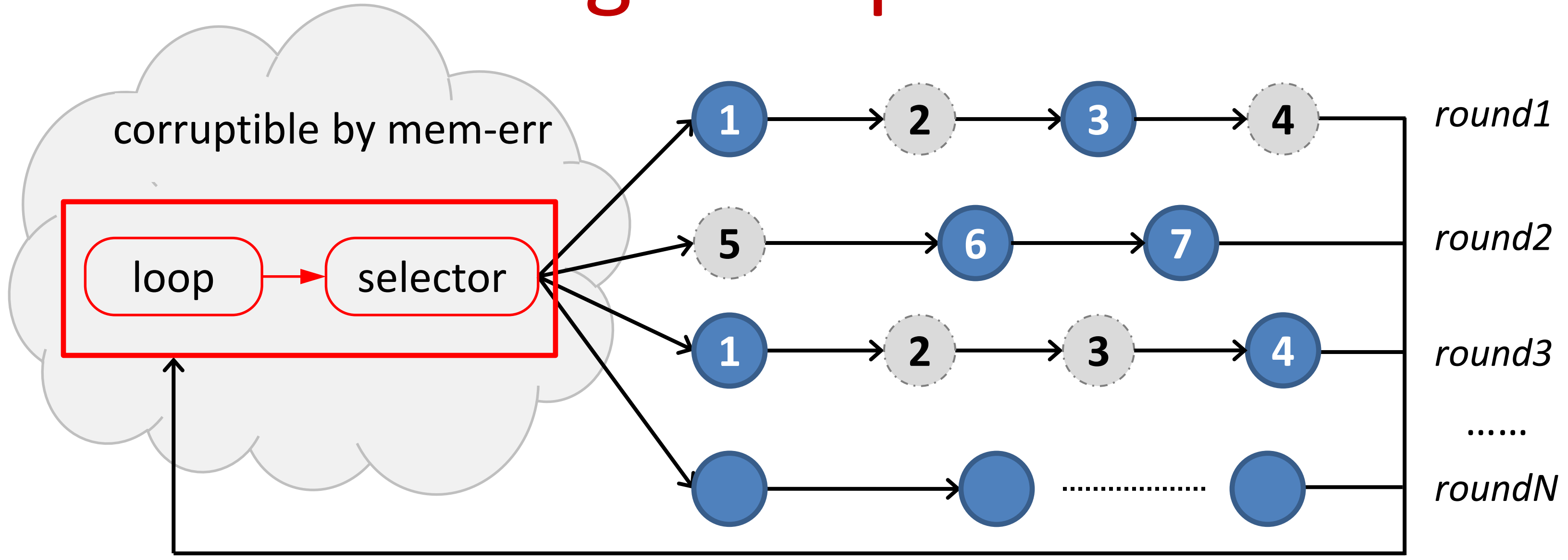
- Chain data-oriented gadgets “***legitimately***”
  - **loop** ---> repeatedly invoke gadgets
  - **selector** ---> selectively activate gadgets

Gadget Dispatcher



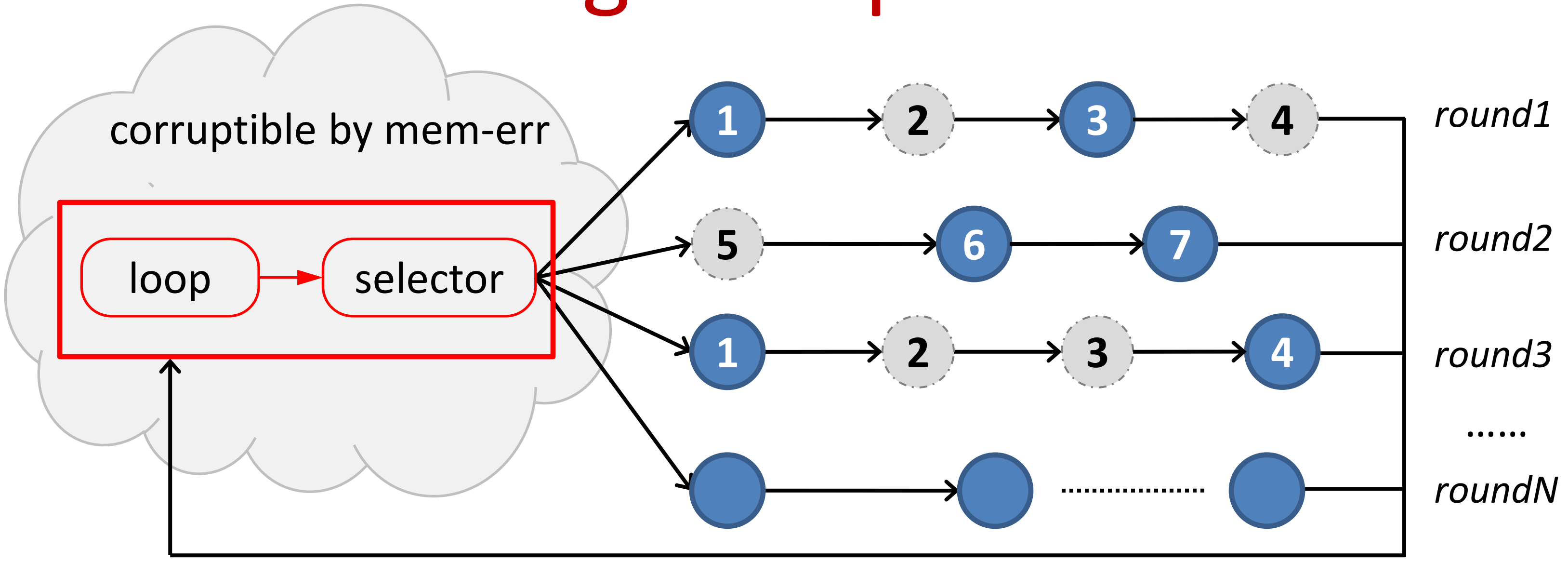
- Chain data-oriented gadgets ***"legitimately"***
  - **loop** ---> repeatedly invoke gadgets
  - **selector** ---> selectively activate gadgets

Gadget Dispatcher



- Chain data-oriented gadgets ***"legitimately"***
  - **loop** ---> repeatedly invoke gadgets
  - **selector** ---> selectively activate gadgets

Gadget Dispatcher



- Chain data-oriented gadgets ***"legitimately"***
  - **loop** ---> repeatedly invoke gadgets
  - **selector** ---> selectively activate gadgets

```
6  while (quota--) {                      // loop
7    readData(sockfd, buf);               // selector
8    if(*type == NONE ) break;
9    if(*type == STREAM) *size = *(srv->cur_max);
10    else{ srv->typ = *type;  srv->total += *size; }
14 }
```

Turing-completeness

- DOP emulates a minimal language *MINDOP*
  - *MINDOP* is Turing-complete

Semantics	Statements In C	Data-Oriented Gadgets in DOP
arithmetic / logical	a op b	*p op *q
assignment	a = b	*p = *q
load	a = *b	*p = **q
store	*a = b	**p = *q
jump	goto L	vpc = &input
conditional jump	if (a) goto L	vpc = &input if *p
p – &a; q – &b; op – any arithmetic / logical operation		

Attack Construction

```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
   } //...(code skipped)...
}
```

Attack Construction

```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
   } //...(code skipped)...
}
```

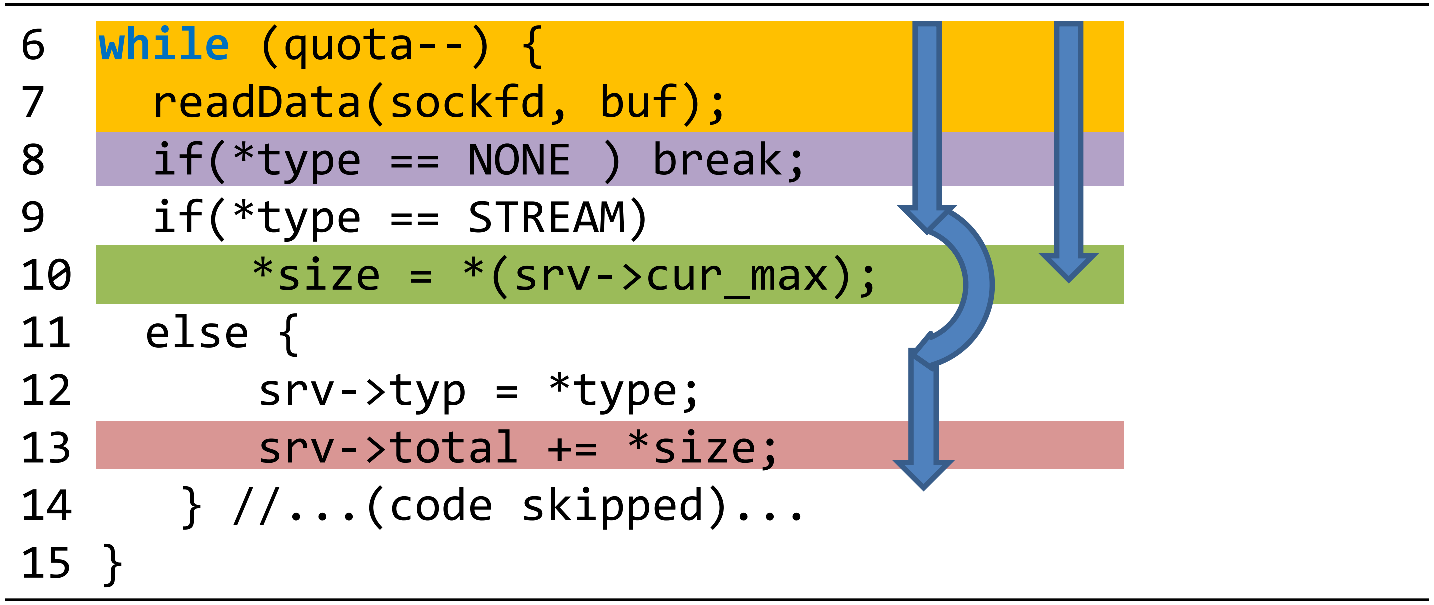
- Gadget identification
  - statically identify load-semantics-store chain from LLVM IR

Attack Construction

```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
  else {
      srv->typ = *type;
      srv->total += *size;
   } //...(code skipped)...
}
```

- Gadget identification
  - statically identify load-semantics-store chain from LLVM IR
- Dispatcher identification
  - static identify loops with gadgets from LLVM IR

Attack Construction



```
while (quota--) {
  readData(sockfd, buf);
  if(*type == NONE ) break;
  if(*type == STREAM)
      *size = *(srv->cur_max);
 else {
      srv->typ = *type;
      srv->total += *size;
   } //...(code skipped)...
}
```

- Gadget identification
  - statically identify load-semantics-store chain from LLVM IR
- Dispatcher identification
  - static identify loops with gadgets from LLVM IR
- Gadget stitching
  - select gadgets and dispatchers (manual)
  - check stitchability (manual)

Evaluation

Evaluation – Feasibility

9 x86 programs with 9 vulnerabilities

- Nginx, ProFTPD, Wu-FTPD, sshd, Bitcoind,
- Wireshark, sudo, musl libc, mcrypt

Evaluation – Feasibility

9 x86 programs with 9 vulnerabilities

- Nginx, ProFTPD, Wu-FTPD, sshd, Bitcoind,
- Wireshark, sudo, musl libc, mcrypt

- x86 Gadgets
  - 7518 in total, 1273 reachable via selected CVEs
  - 8 programs can simulate all MINDOP operations
- x86 Dispatchers
  - 1443 in total, 110 reachable from selected CVEs

Evaluation – Feasibility

9 x86 programs with 9 vulnerabilities

- Nginx, ProFTPD, Wu-FTPD, sshd, Bitcoind,
- Wireshark, sudo, musl libc, mcrypt

- x86 Gadgets
  - 7518 in total, 1273 reachable via selected CVEs
  - 8 programs can simulate all MINDOP operations
- x86 Dispatchers
  - 1443 in total, 110 reachable from selected CVEs
- 2 programs can build Turing-complete attack
- 3 end-to-end attacks

Case Study: Bypassing Randomization

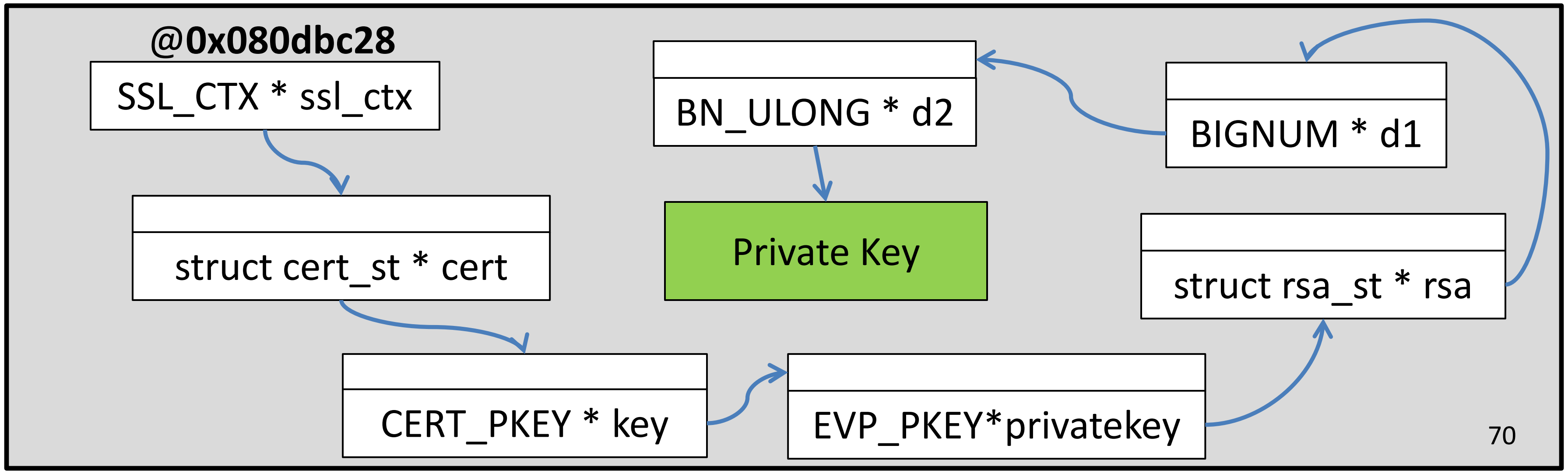
- Previous methods
  - information leakage to network
- Defeat ASLR w/o address leakage to network?

Case Study: Bypassing Randomization

- Previous methods
  - information leakage to network
- Defeat ASLR w/o address leakage to network?
- Vulnerable *ProFTPD*
  - use OpenSSL for authentication
  - a dereference chain to the private key

Case Study: Bypassing Randomization

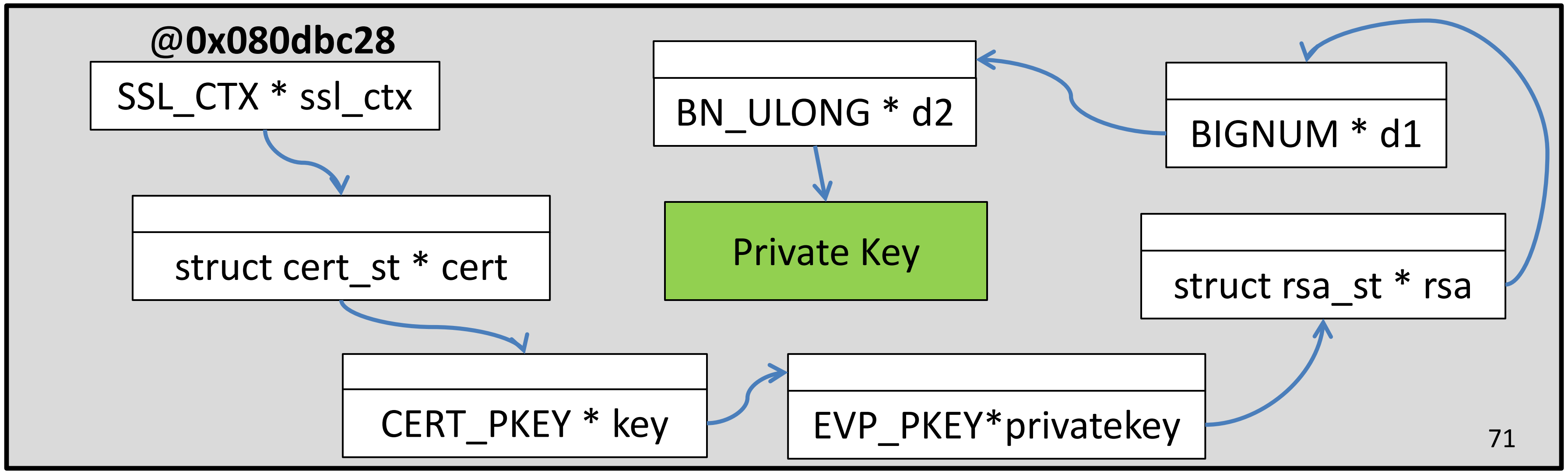
- Previous methods
  - information leakage to network
- Defeat ASLR w/o address leakage to network?
- Vulnerable *ProFTPD*
  - use OpenSSL for authentication
  - a dereference chain to the private key



Case Study: Bypassing Randomization

- Gadgets

MOV	*p = *q
ADD	*X = *X + offset
LOAD	*Z = **Y



Case Study: Bypassing Randomization

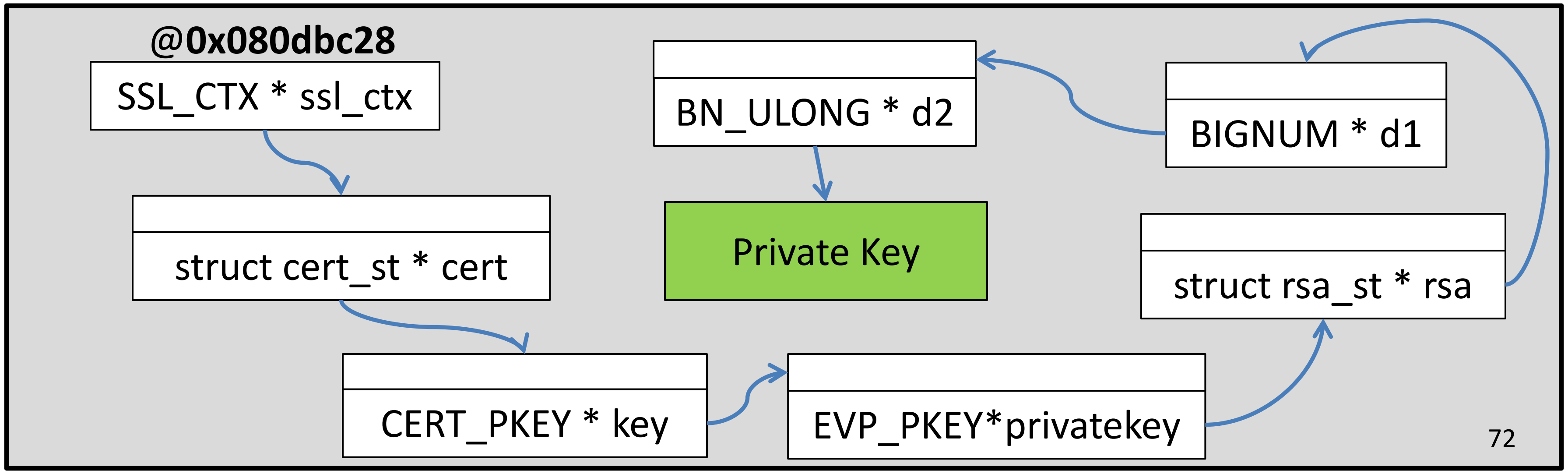
- Gadgets

MOV	*p = *q
ADD	*X = *X + offset
LOAD	*Z = **Y

- Dispatcher

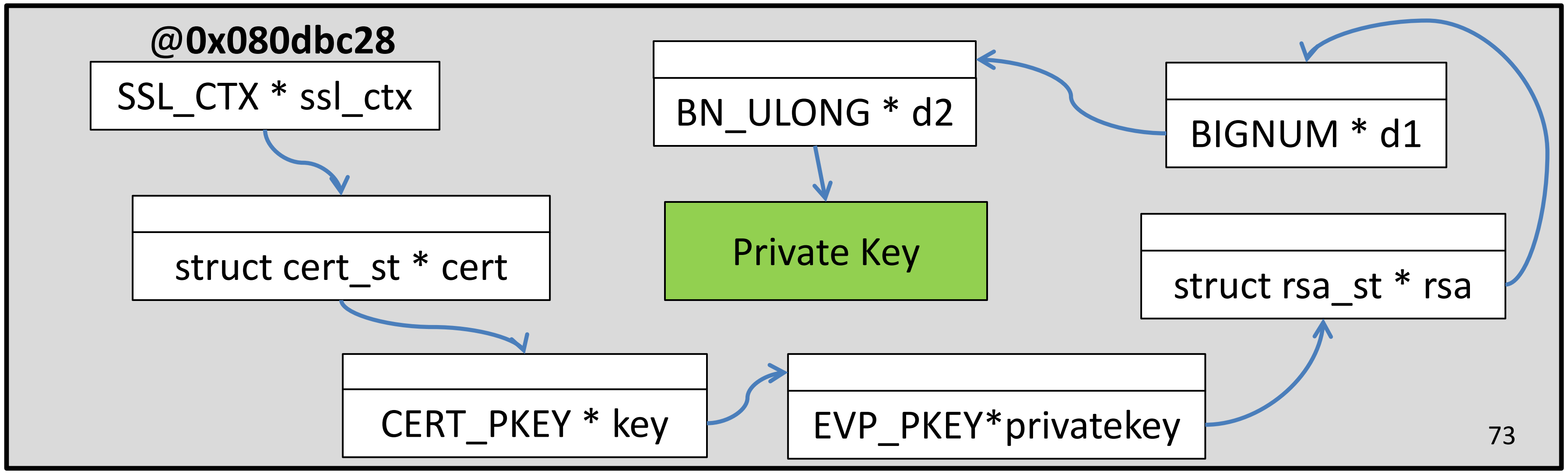
```
while (1) {
    user_request =
        get_user_request();
    dispatch(user_request);
}
```

```
func1() { memory_error; MOV;}
func2() { ADD; }
func3() { LOAD; }
```



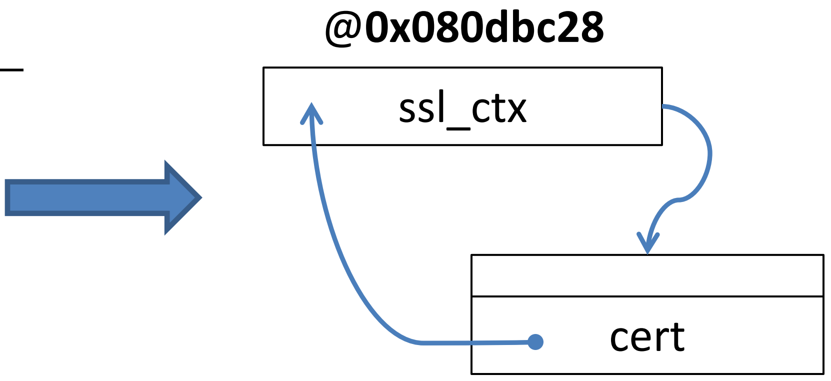
Case Study: Bypassing Randomization

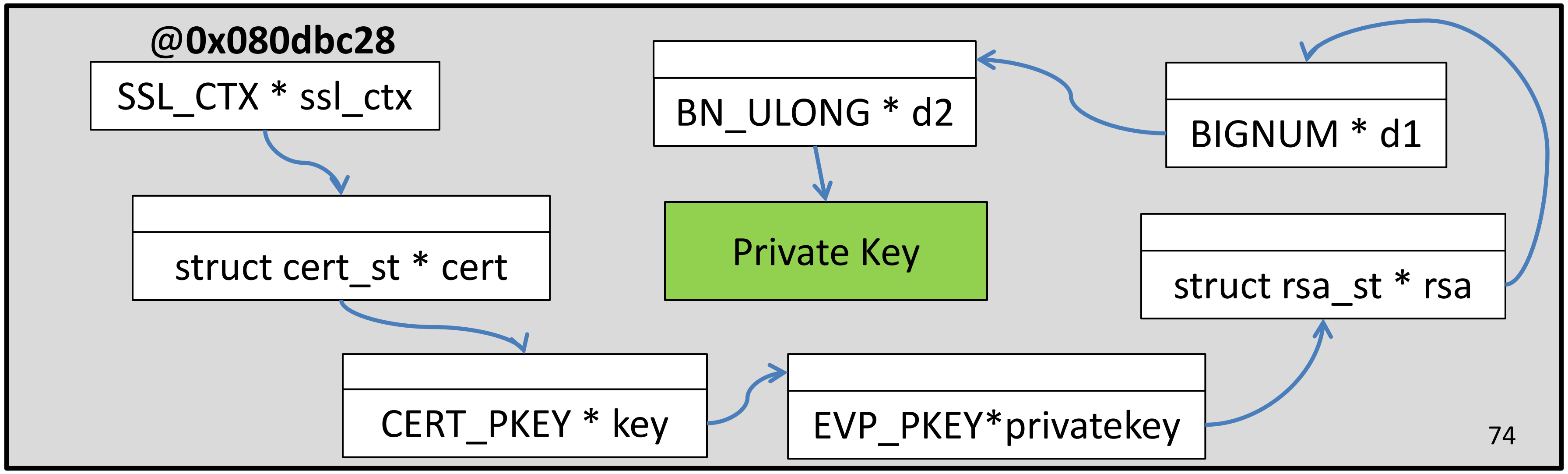
MOV	*X = *0x080dbc28 (**ssl_ctx**)
ADD	*X = *X + offset1
MOV	*Y = *X
LOAD	*Z = **Y
MOV	*0x080dbc28 = *Z (**cert**)



Case Study: Bypassing Randomization

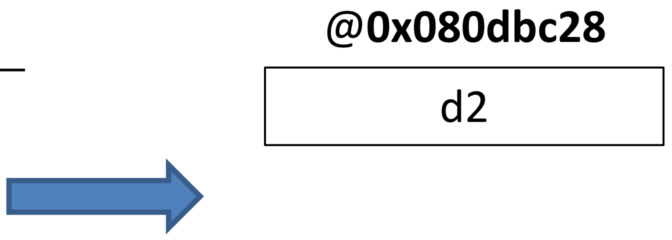
MOV	*X = *0x080dbc28 (**ssl_ctx**)
ADD	*X = *X + offset1
MOV	*Y = *X
LOAD	*Z = **Y
MOV	*0x080dbc28 = *Z (**cert**)

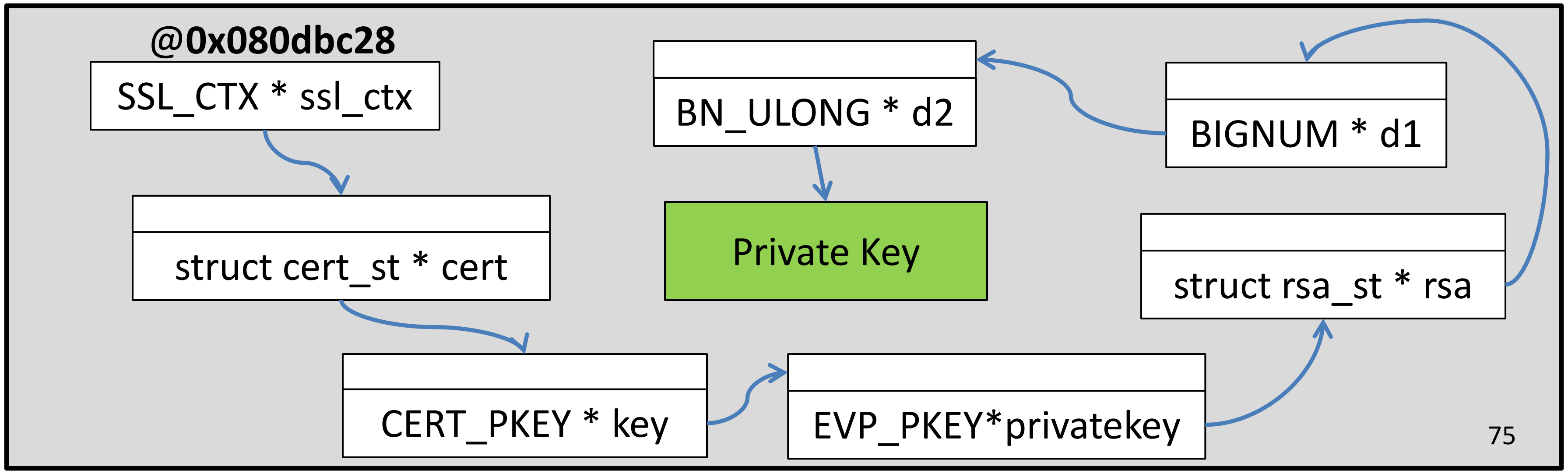




Case Study: Bypassing Randomization

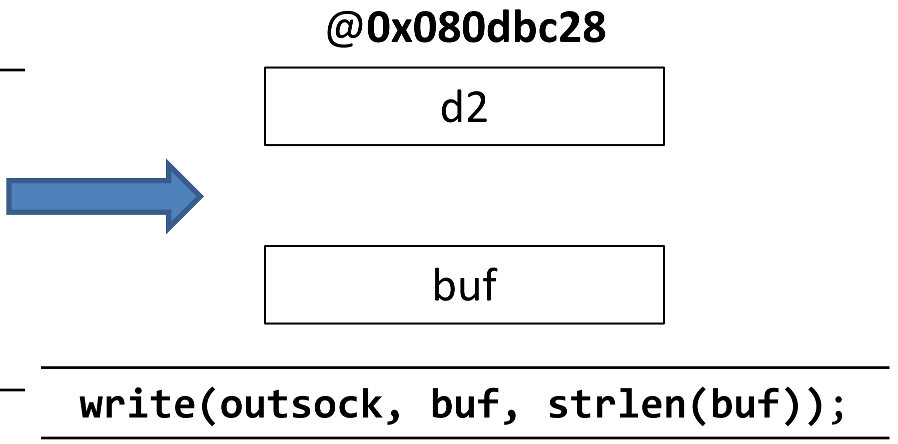
MOV	*X = *0x080dbc28 (**ssl_ctx**)
ADD	*X = *X + offset1
MOV	*Y = *X
LOAD	*Z = **Y
MOV	*0x080dbc28 = *Z (**cert**)



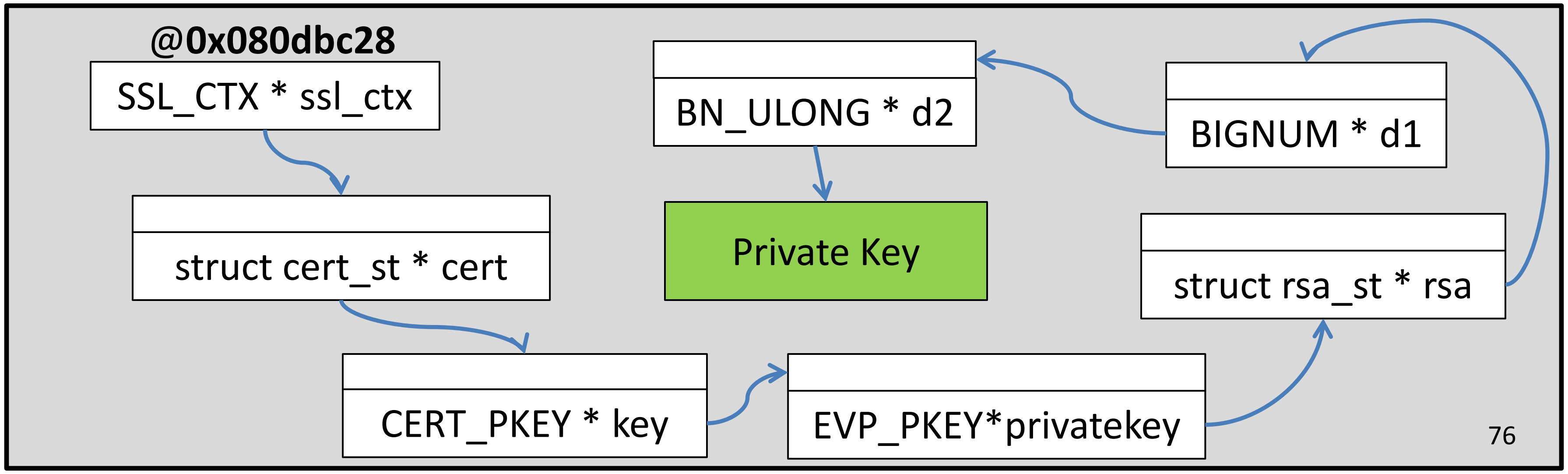


Case Study: Bypassing Randomization

MOV	*X = *0x080dbc28 (**ssl_ctx**)
ADD	*X = *X + offset1
MOV	*Y = *X
LOAD	*Z = **Y
MOV	*0x080dbc28 = *Z (**cert**)

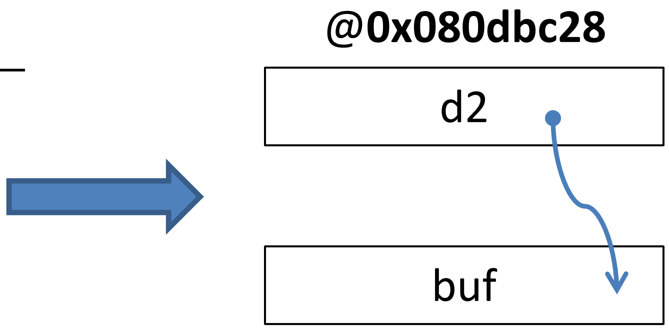


```
write(outsock, buf, strlen(buf));
```

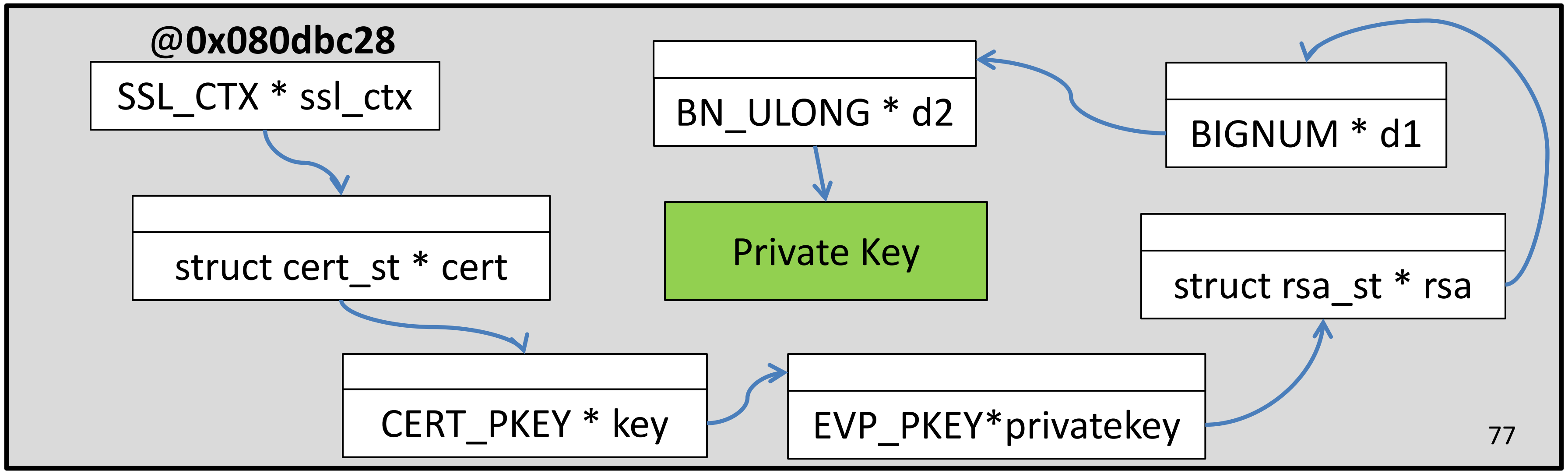


Case Study: Bypassing Randomization

MOV	*X = *0x080dbc28 (**ssl_ctx**)
ADD	*X = *X + offset1
MOV	*Y = *X
LOAD	*Z = **Y
MOV	*0x080dbc28 = *Z (**cert**)

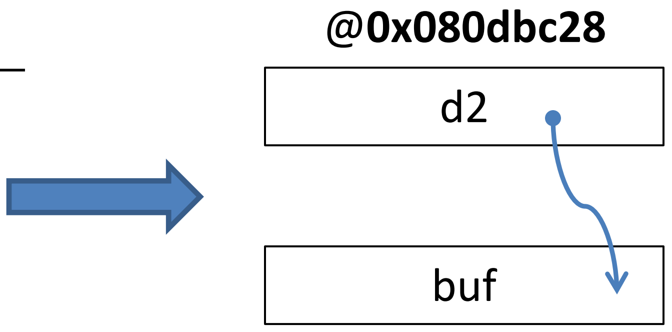


```
write(outsock, buf, strlen(buf));
```



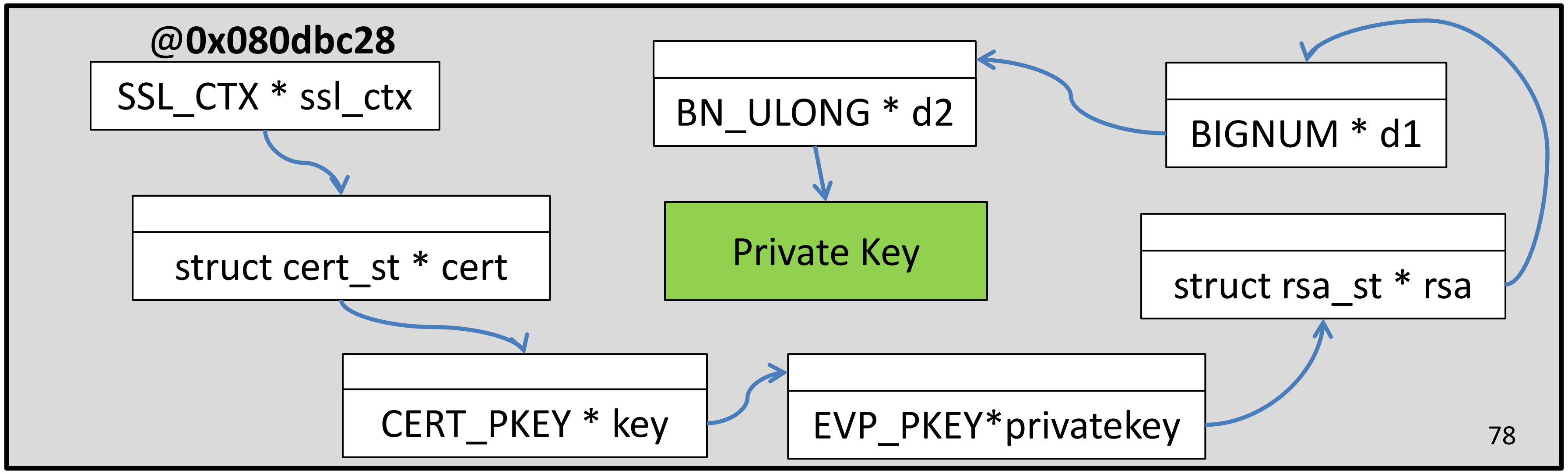
Case Study: Bypassing Randomization

MOV	*X = *0x080dbc28 (**ssl_ctx**)
ADD	*X = *X + offset1
MOV	*Y = *X
LOAD	*Z = **Y
MOV	*0x080dbc28 = *Z (**cert**)



```
write(outsock, buf, strlen(buf));
```

leak private key to network

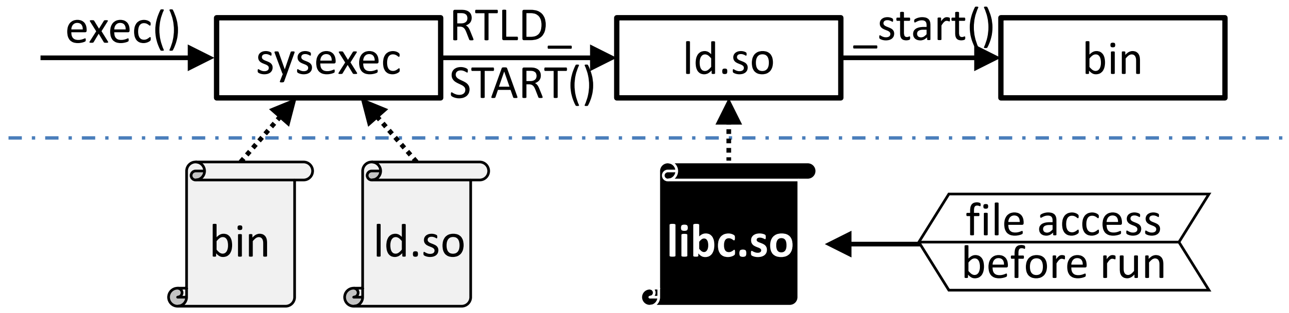


dlopen() – Dynamic Linking Interface

- Load the dynamic library into memory space
  - resolve symbols based on binary metadata
  - patch program due to relocation
  - like *LoadLibrary*() on Windows

dlopen() – Dynamic Linking Interface

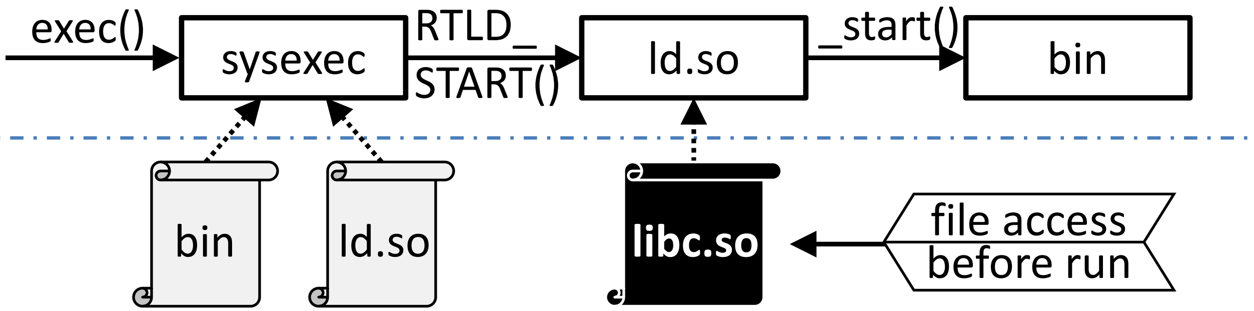
- Load the dynamic library into memory space
  - – resolve symbols based on binary metadata
  - – patch program due to relocation
  - – like *LoadLibrary*() on Windows
- Dynamic loader can do arbitrary computation[*]



* R. Shapiro, S. Bratus, and S. W. Smith, ""Weird Machines" in ELF: A Spotlight on the Underappreciated Metadata," in WOOT 2013.

dlopen() – Dynamic Linking Interface

- Load the dynamic library into memory space
  - resolve symbols based on binary metadata
  - patch program due to relocation
  - like *LoadLibrary*() on Windows
- Dynamic loader can do arbitrary computation*

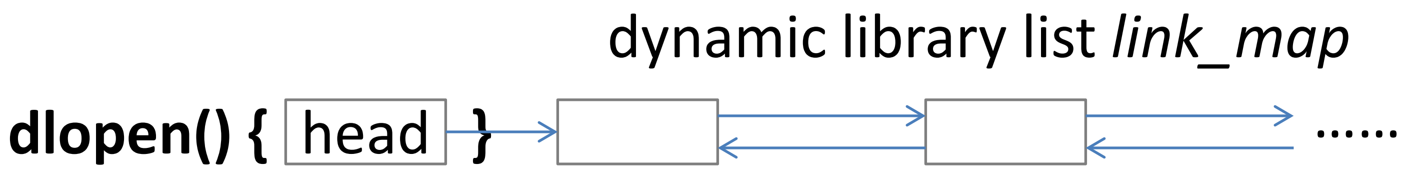


- The same to *dlopen*()

* R. Shapiro, S. Bratus, and S. W. Smith, ""Weird Machines" in ELF: A Spotlight on the Underappreciated Metadata," in WOOT 2013.

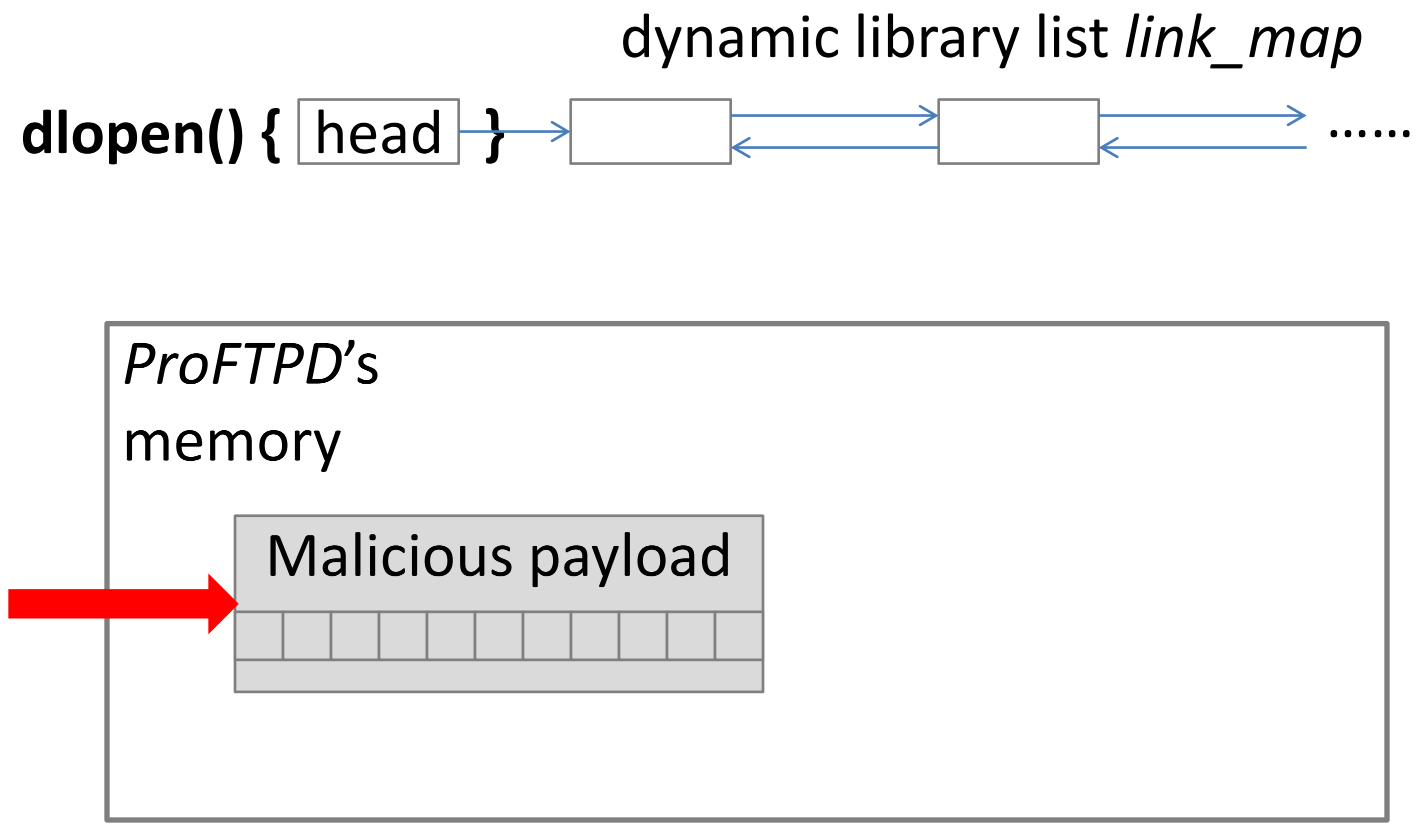
Case Study: Simulating A Network Bot

- Attacks with *dlopen*



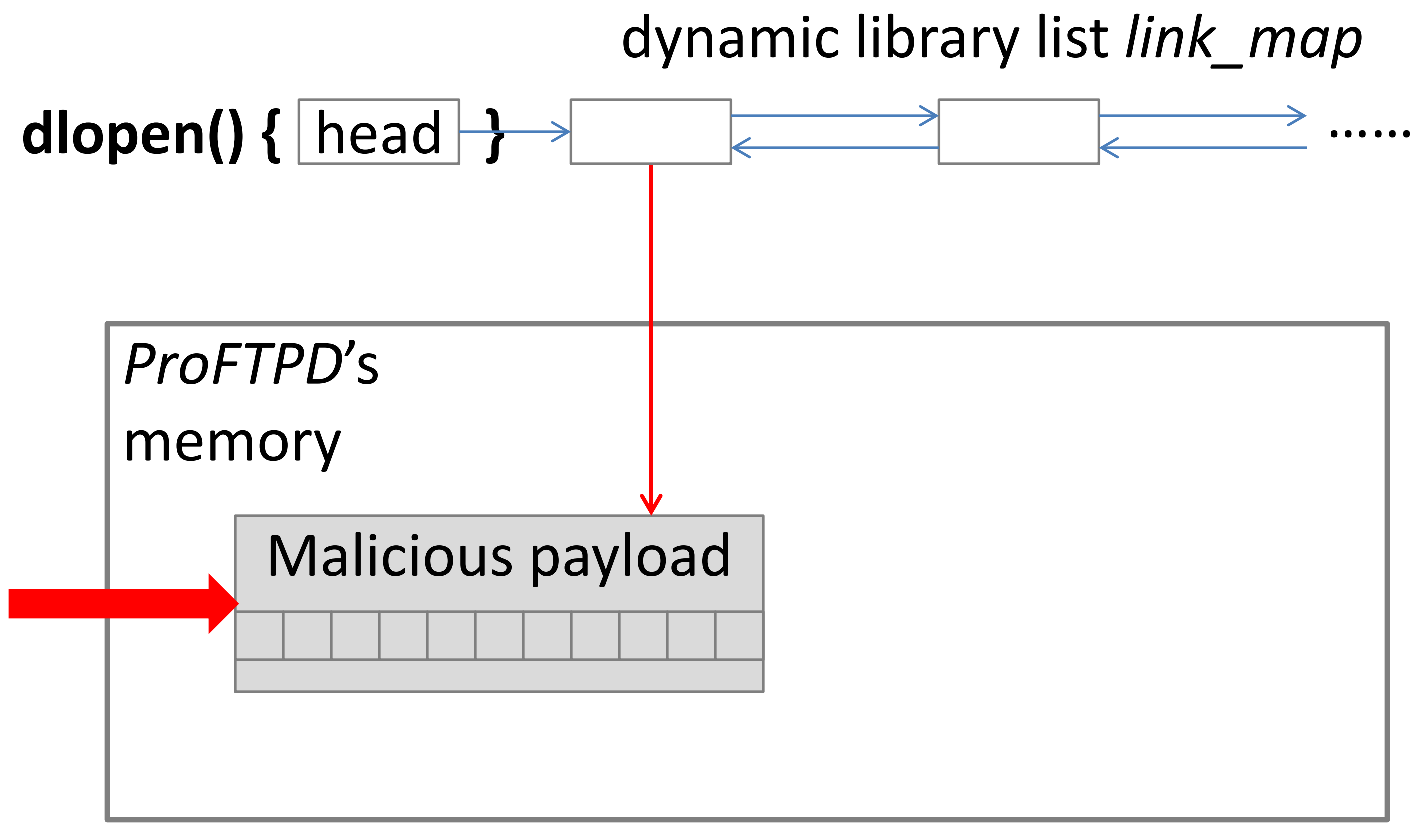
Case Study: Simulating A Network Bot

- Attacks with *dlopen*
  - send malicious payload



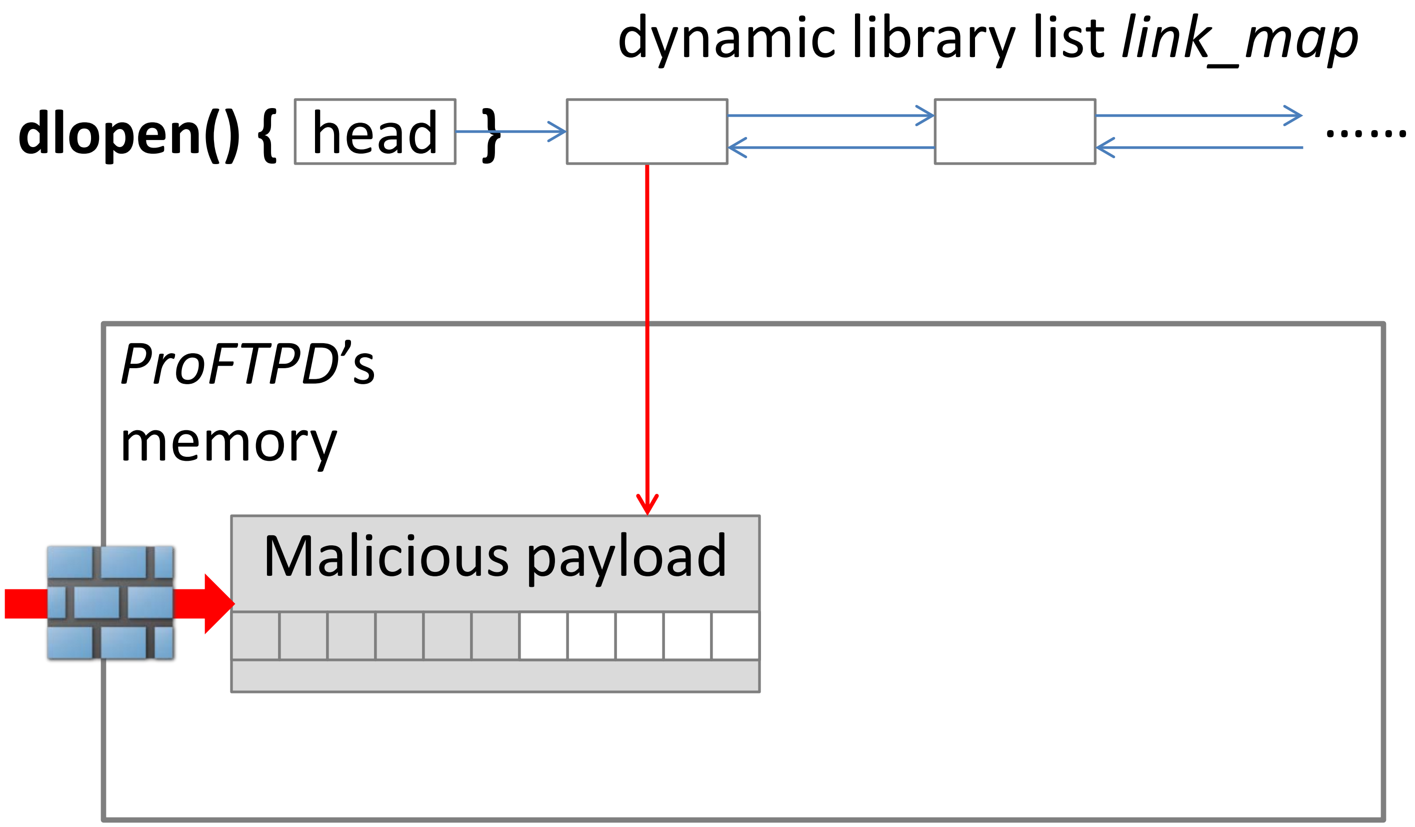
Case Study: Simulating A Network Bot

- Attacks with *dlopen*
  - send malicious payload
  - corrupt link list & call *dlopen*



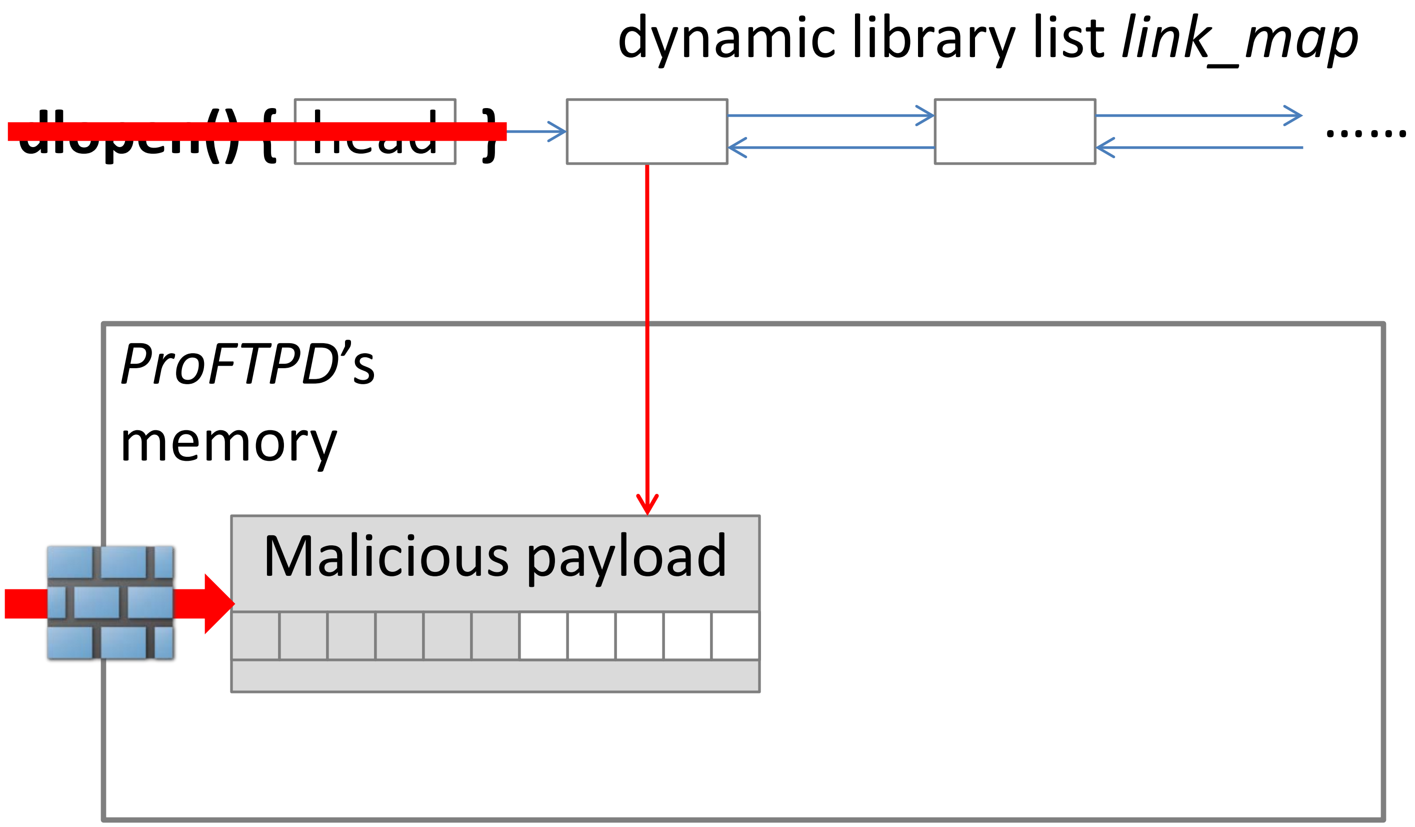
Case Study: Simulating A Network Bot

- Attacks with *dlopen*
  - send malicious payload   invalid input
  - corrupt link list & call *dlopen*



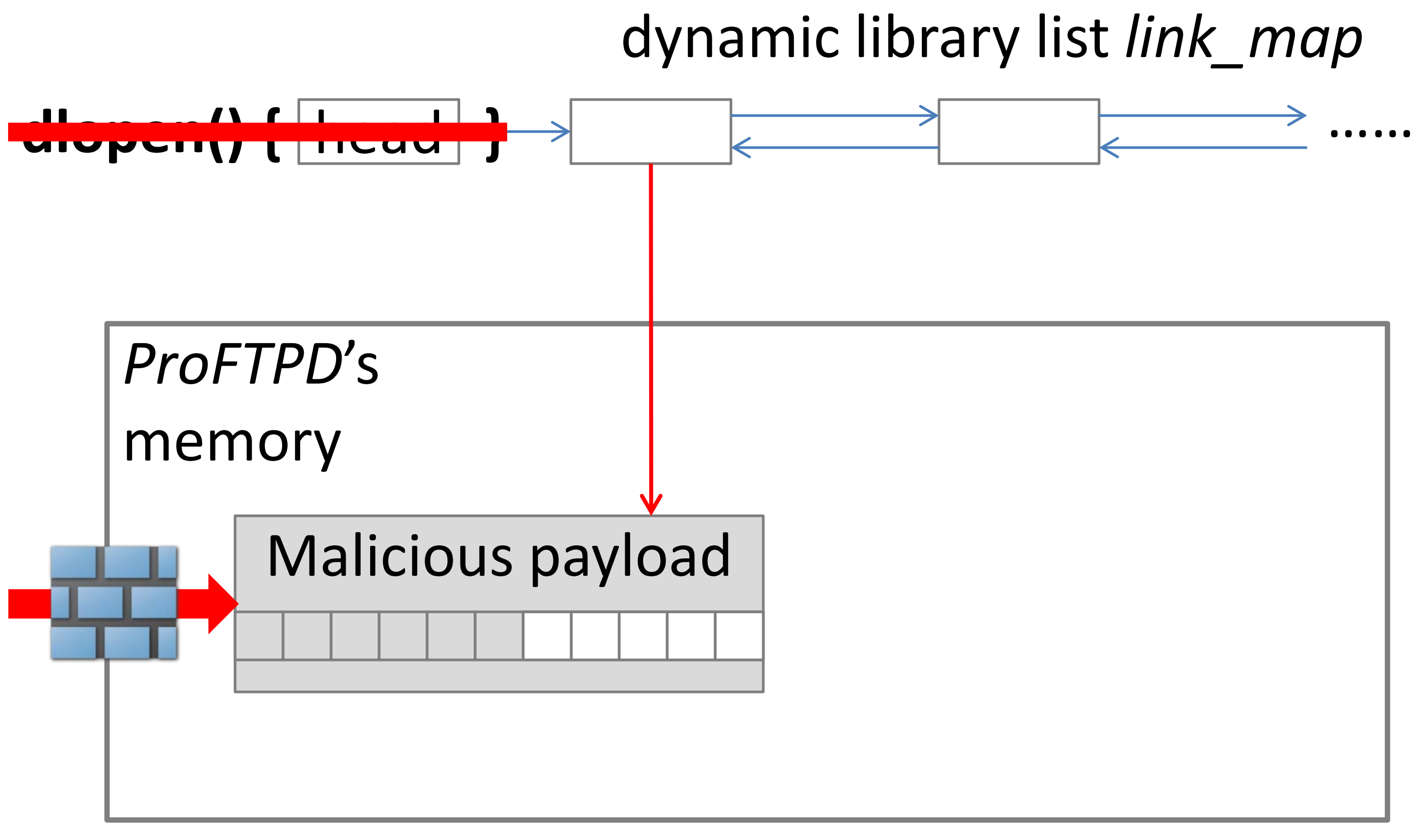
Case Study: Simulating A Network Bot

- Attacks with *dlopen*
  - send malicious payload — invalid input
  - corrupt link list & call *dlopen* — no call to *dlopen*



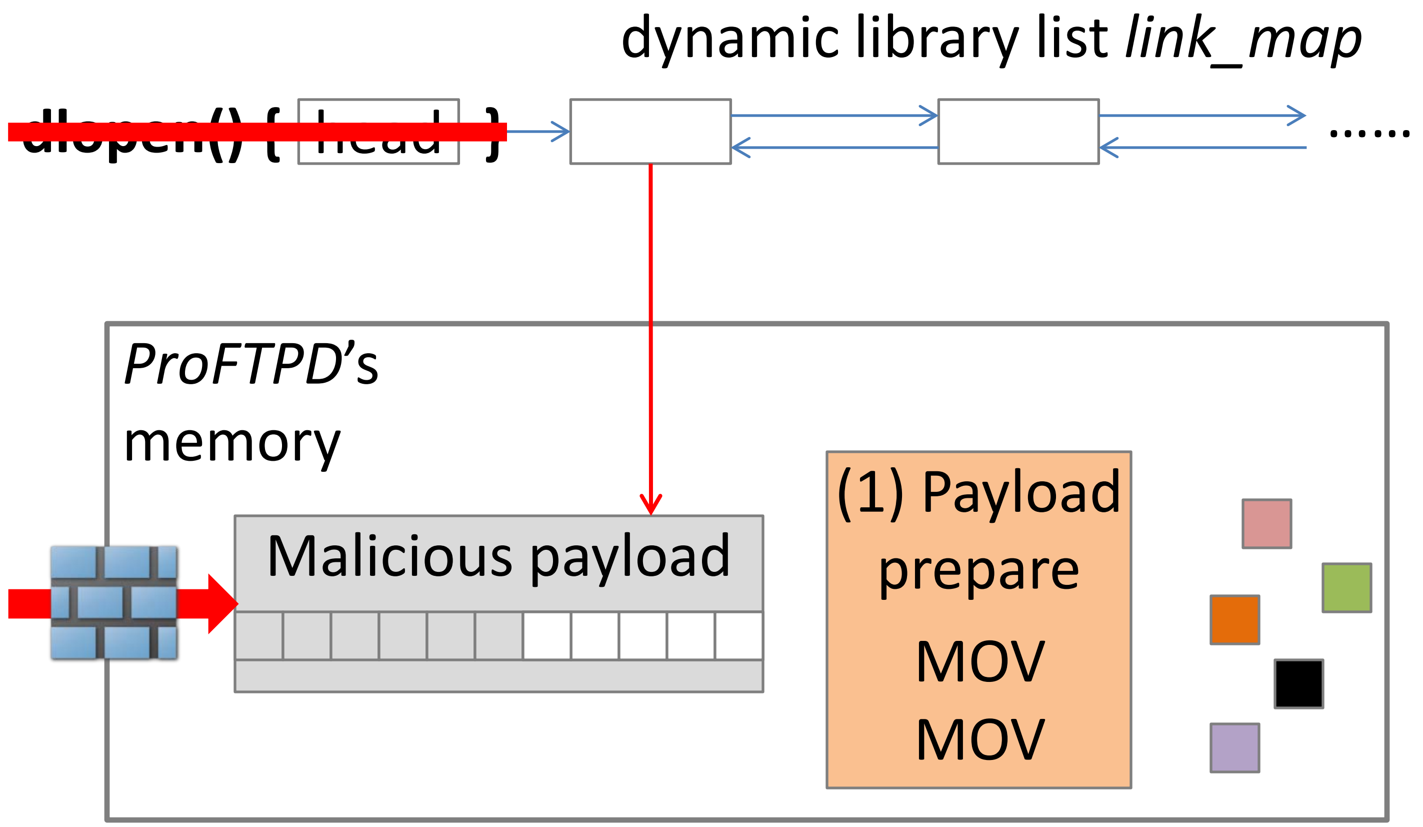
Case Study: Simulating A Network Bot

- DOP attack addresses the problems
  - send malicious payload   invalid input
  - corrupt link list & call *dlopen*   no call to *dlopen*



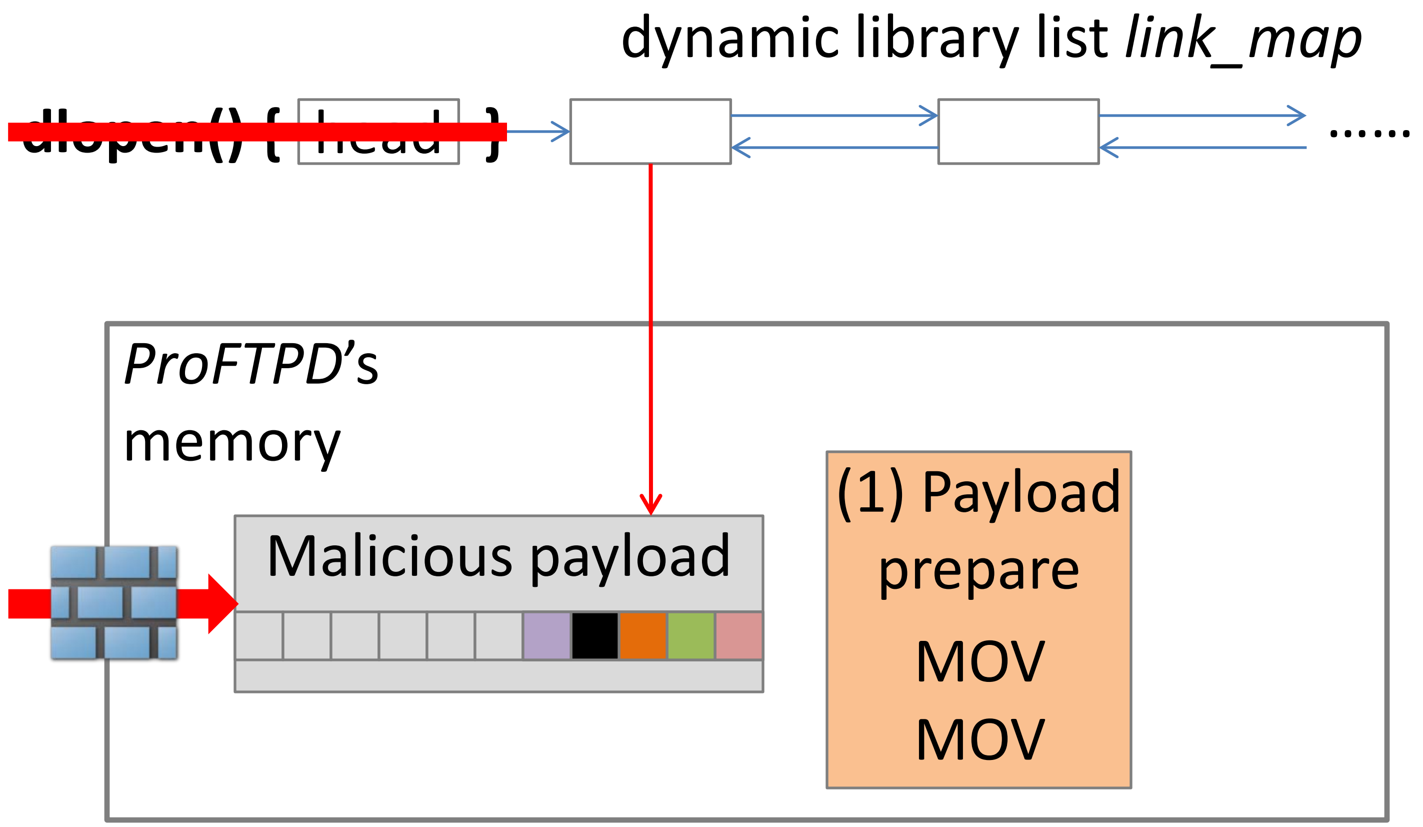
Case Study: Simulating A Network Bot

- DOP attack addresses the problems
  - construct payload in memory — invalid input
  - corrupt link list & call *dlopen* — no call to *dlopen*



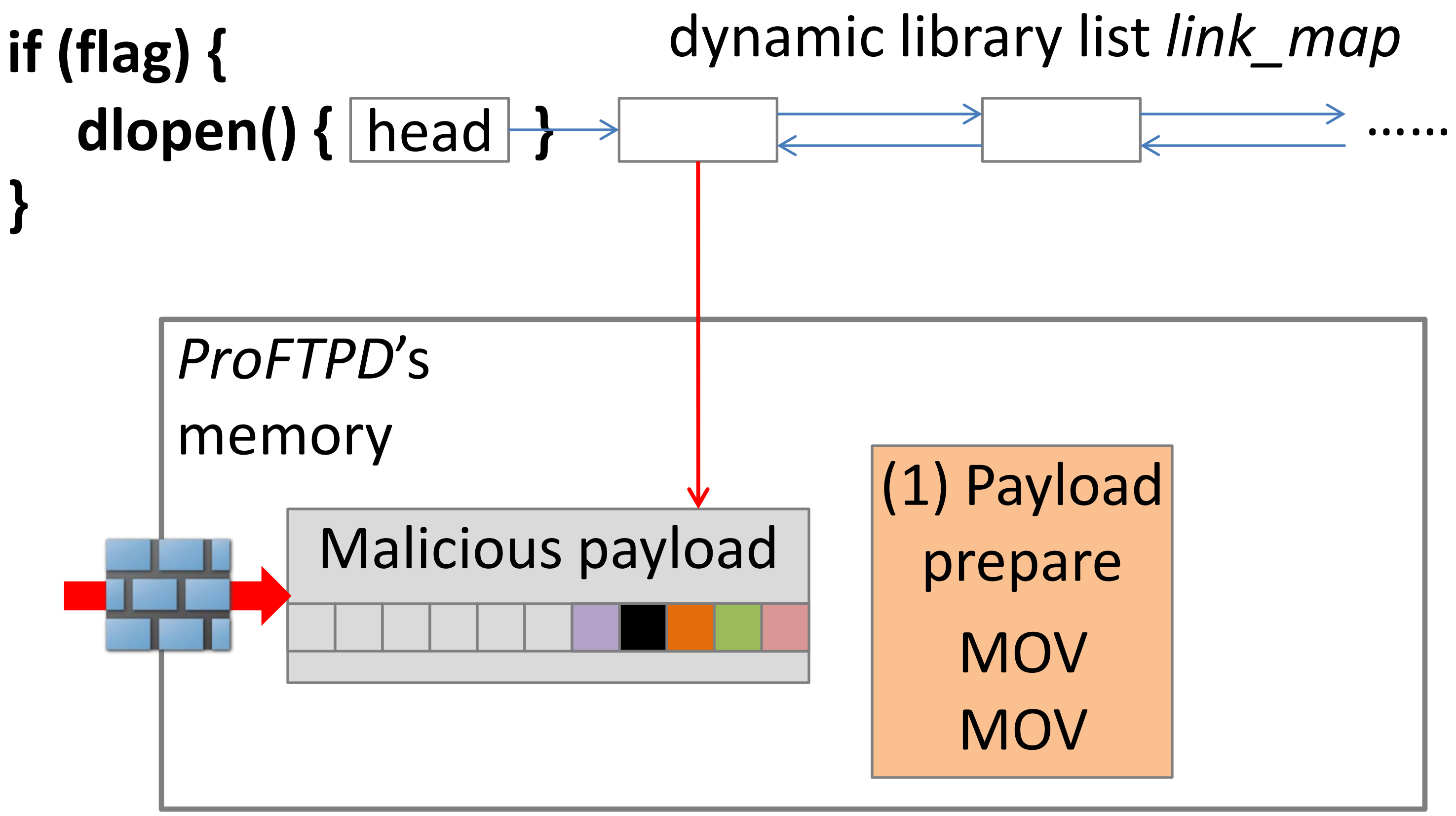
Case Study: Simulating A Network Bot

- DOP attack addresses the problems
  - construct payload in memory — invalid input
  - corrupt link list & call *dlopen* — no call to *dlopen*



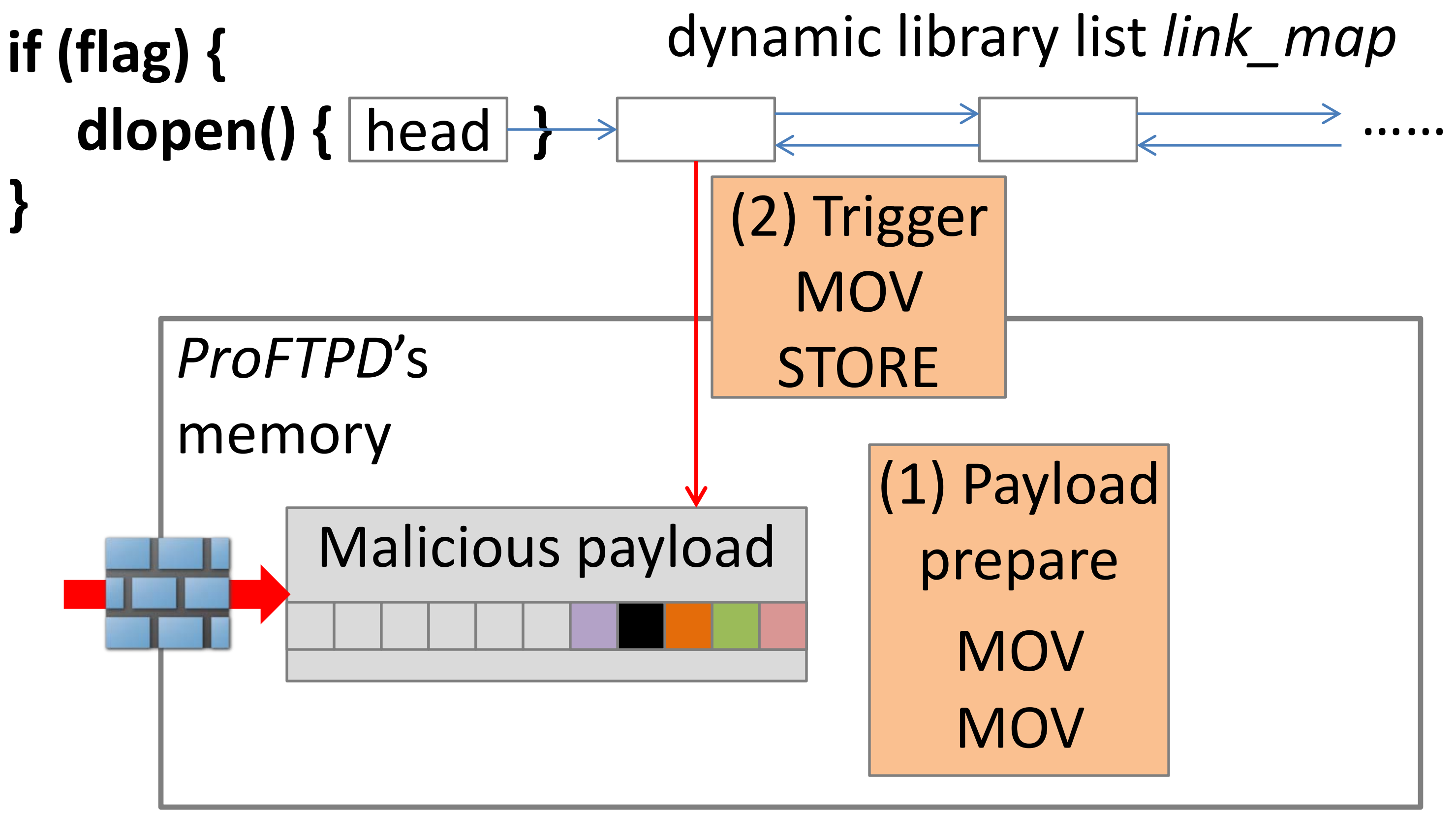
Case Study: Simulating A Network Bot

- DOP attack addresses the problems
  - construct payload in memory — invalid input
  - force call to *dlopen* — no call to *dlopen*



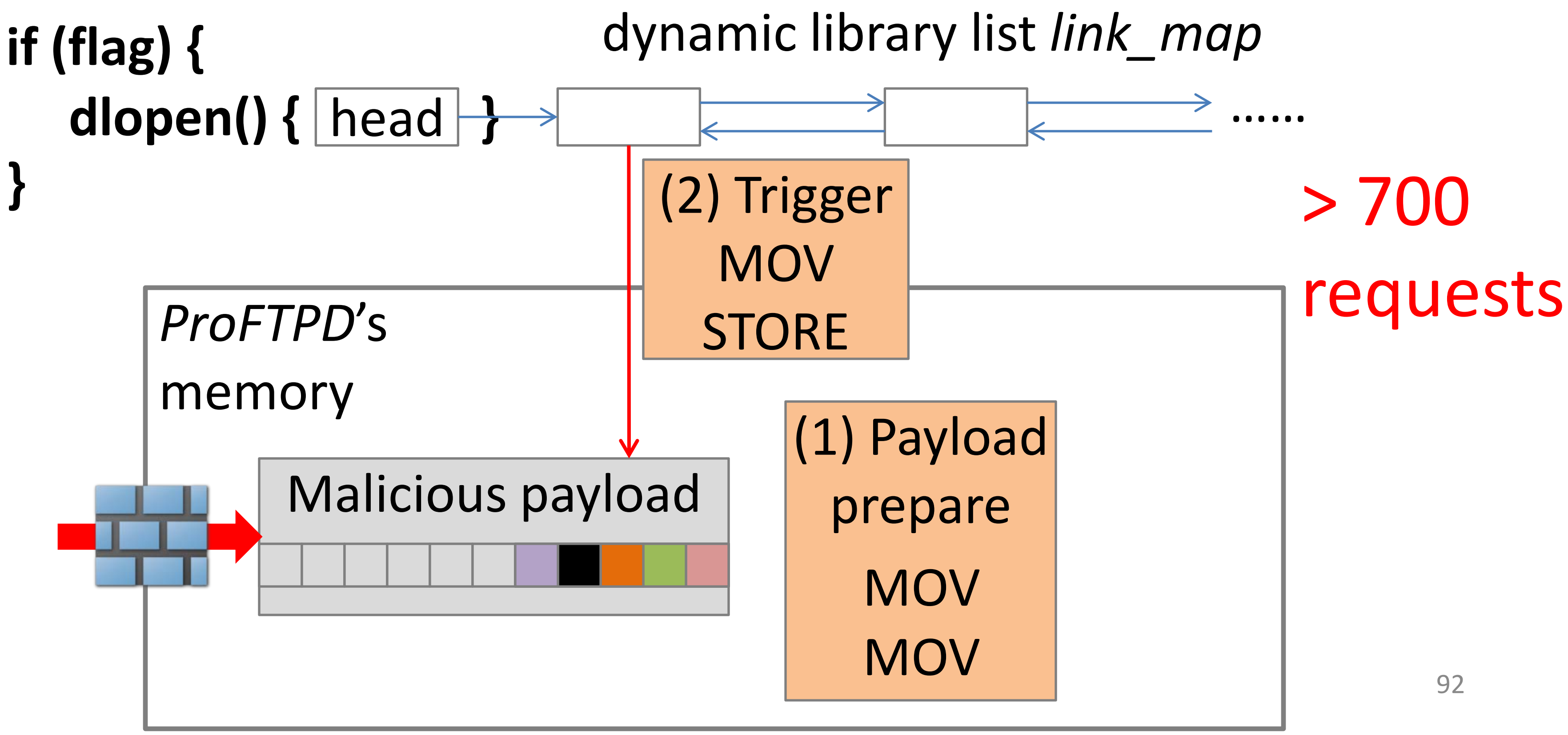
Case Study: Simulating A Network Bot

- DOP attack addresses the problems
  - construct payload in memory — invalid input
  - force call to *dlopen* — no call to *dlopen*



Case Study: Simulating A Network Bot

- DOP attack addresses the problems
  - construct payload in memory — invalid input
  - force call to *dlopen* — no call to *dlopen*



Case Study: Altering Memory Permissions

- Defenses based on memory permissions
  - DEP: non-writable code
  - CFI: non-writable jump tags

Case Study: Altering Memory Permissions

- Defenses based on memory permissions
  - DEP: non-writable code
  - CFI: non-writable jump tags
- *dlopen(): relocation*
  - change any page permission to writable
  - update page content
  - change the permission back

Case Study: Altering Memory Permissions

- Defenses based on memory permissions
  - DEP: non-writable code
  - CFI: non-writable jump tags
- *dlopen(): relocation*
  - change any page permission to writable
  - update page content
  - change the permission back
- DOP attacks
  - *dlopen*(code_addr, shellcode)

Case Study: Altering Memory Permissions

- Defenses based on memory permissions
  - DEP: non-writable code
  - CFI: non-writable jump tags
- *dlopen(): relocation*
  - change any page permission to writable
  - update page content
  - change the permission back
- DOP attacks
  - *dlopen*(code_addr, shellcode)
- Code injection is back!

Related Work

Techniques	Turing Complete?	Preserve CFI?	Independent of specific data / funcs?
Non-control Data Attacks (Chen *et al.* 2005)		✓	
COOP (Schuster *et al.* 2015)	✓		✓
FlowStitch (Hu *et al.* 2015)		✓	
Printf-Oriented Programming (Carlini *et al.* 2015)	✓	✓	
Control Jujustu (Evans *et al.* 2015)		✓	
Data-Oriented Programming	✓	✓	✓

Potential Defenses

- Memory Safety
  - e.g., Cyclone (Jim *et al.* 2002), CCured (Necula *et al.* 2002) , SoftBounds+CETS (Nagarakatte *et al.* 2009, 2010)
  - high performance overhead (> 100%)
- Data-flow Integrity
  - e.g, DFI (Castro *et al.* 2006) , kernel DFI (Song *et al.* 2016)
- Fined-grained randomization in data space
  - e.g., DSR (Bhatkar *et al.* 2008)
- Hardware & software fault isolation
  - e.g., HDFI (Song *et al.* 2016) , MPX

Potential Defenses

- Memory Safety
  - e.g., Cyclone (Jim *et al.* 2002), CCured (Necula *et al.* 2002) , SoftBounds+CETS (Nagarakatte *et al.* 2009, 2010)
  - high performance overhead (> 100%)
- Data-flow Integrity
  - e.g, DFI (Castro *et al.* 2006) , kernel DFI (Song *et al.* 2016)
- Fined-grained randomization in data space
  - e.g., DSR (Bhatkar *et al.* 2008)
- Hardware & software fault isolation
  - e.g., HDFI (Song *et al.* 2016) , MPX

No practical defenses yet !

Conclusion

- Non-control data attacks can be Turing-complete

Conclusion

- Non-control data attacks can be Turing-complete
- Data-Oriented Programming (DOP)
  - build expressive non-control data attacks
  - independent of specific data / functions

Conclusion

- Non-control data attacks can be Turing-complete
- Data-Oriented Programming (DOP)
  - build expressive non-control data attacks
  - independent of specific data / functions
- In real-world programs, DOP can build attacks
  - bypass ASLR w/o address leakage
  - simulate a network bot
  - enable code injection

Thanks!

Hong Hu

huhong@comp.nus.edu.sg

http://www.comp.nus.edu.sg/~huhong

Non-control data attacks are available

http://huhong-nus.github.io/advanced-DOP/