ABOUT EXPLOITS WRITING

Gerardo Richarte | gera@coresecurity.com

Outline

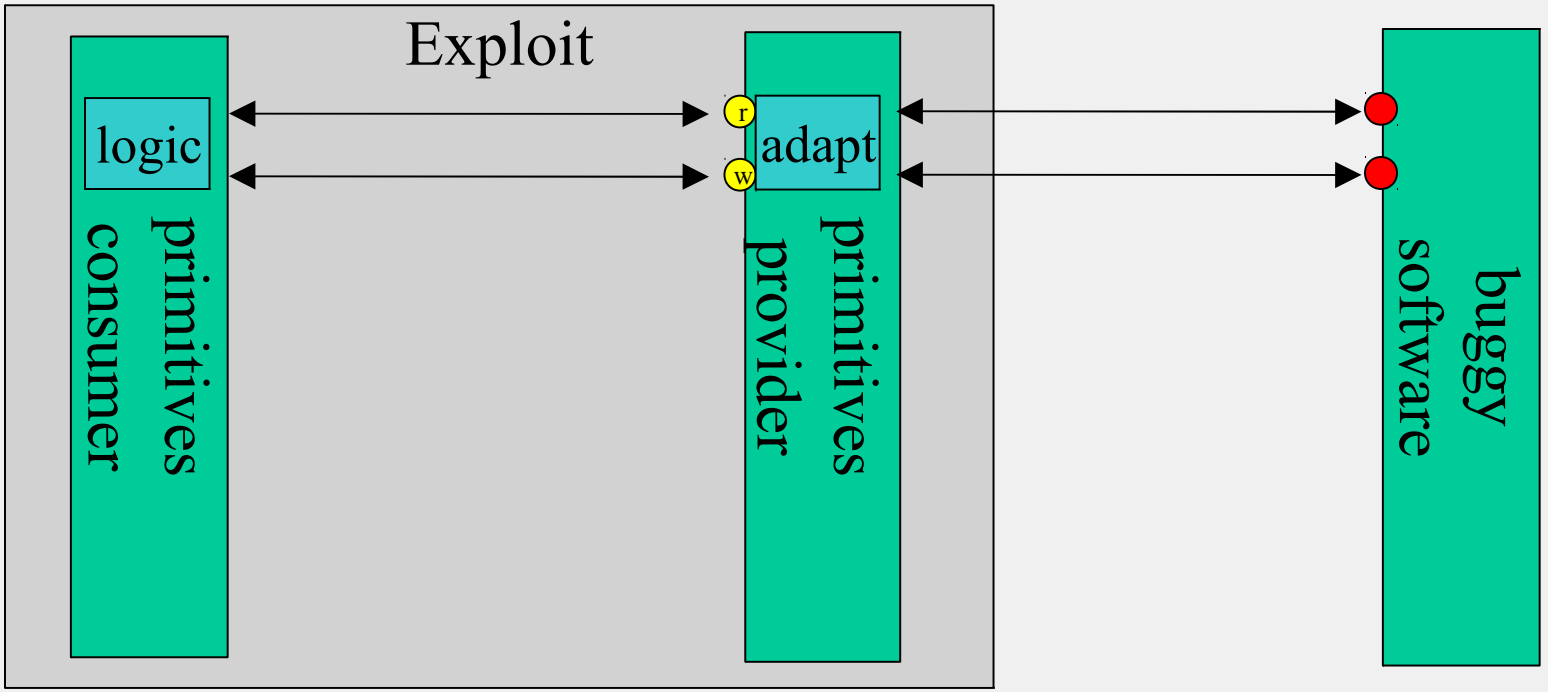
- Basics
- Turning bugs into primitives
- Turning primitives into exploits
- Conclusions

Outline

Basics

Basics

- Argot
- Invalid assumptions
- Security bugs
- Execution flow



Turning bugs into primitives

Turning bugs into primitives

writing

Stack based buffer overflow:

```
int main(int argv, char **argc) {
    char buf[80];

    strcpy(buf,argc[1]);
}
```

buf	80 bytes
frame pointer	4 bytes
return address	4 bytes
main's args	4 bytes

multiple stack frame overwrite primitive

jump primitive

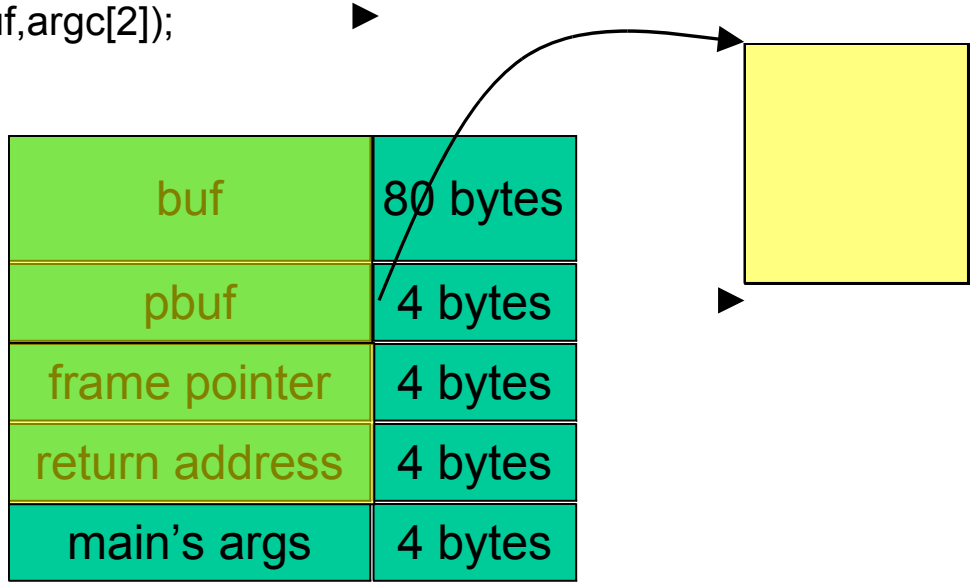
write-anything-somewhere primitive

Turning bugs into primitives

writing

Stack based complex buffer overflow:

```
int main(int argv, char **argc) {
    char *pbuf=malloc(strlen(argc[2])+1);
    char buf[80];

    strcpy(buf,argc[1]);
    strcpy(pbuf,argc[2]);
    exit(0);
}
```



write-anything-anywhere primitive

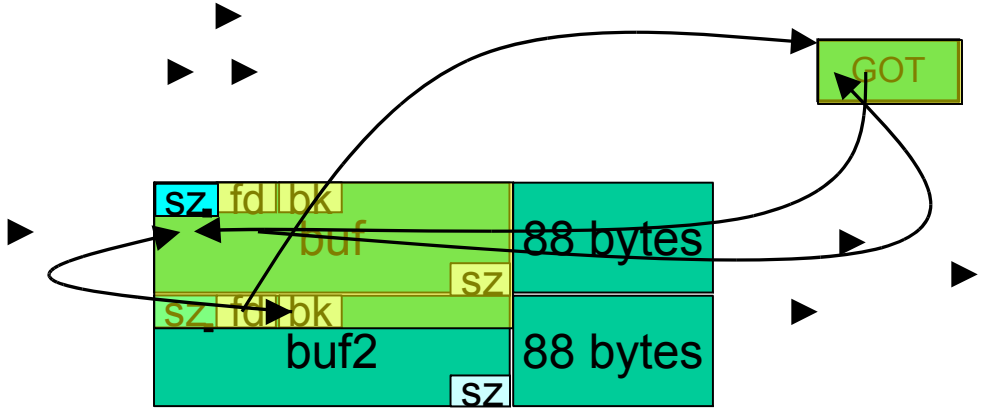
multiple stack frame overwrite primitive

Turning bugs into primitives

writing

Heap based buffer overflow (free bug):

```
int main(int argv, char **argc) {
    char buf=malloc(84), buf2=malloc(84);

    strcpy(buf,argc[1]);
    free(buf2);
}
```



mirrored 4 bytes write-anything-anywhere primitive

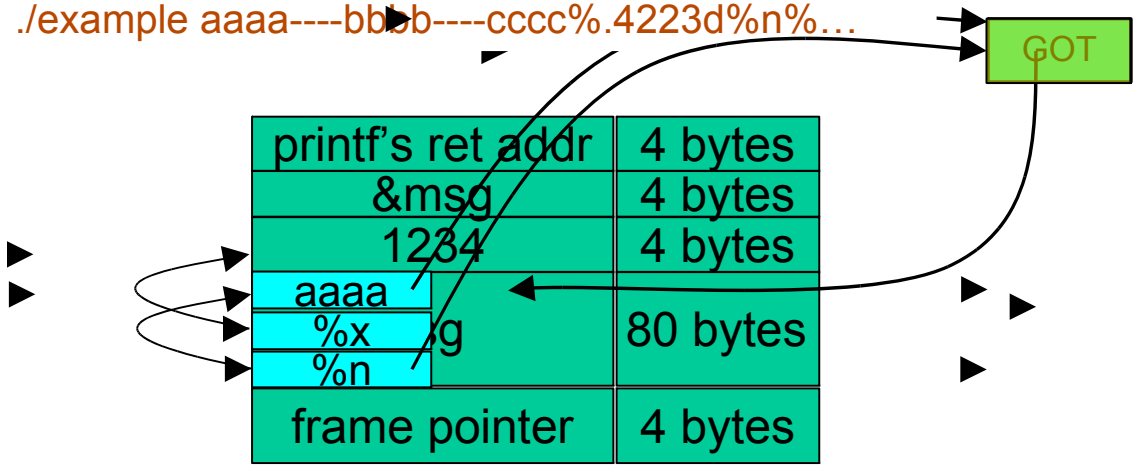
Turning bugs into primitives

writing

Format string bug:

```
int main(int argv, char **argc) {
    char msg[80];
    strcpy(msg, argc[1]);
    printf(msg,1234);
}
```

./example aaaa----bbbb----cccc%.4223d%n%…



write-anything-anywhere primitive

Turning bugs into primitives

writing

Other writing bugs:

mirrored 4 bytes write-anything-anywhere:

- Double free()
- Corrupted heap + malloc()
- Double fclose()

single stack frame overwrite:

- Negative length on bcopy() / memcopy()
- Stack based buffer overflow in PA-RISC

other:

- Array overflows

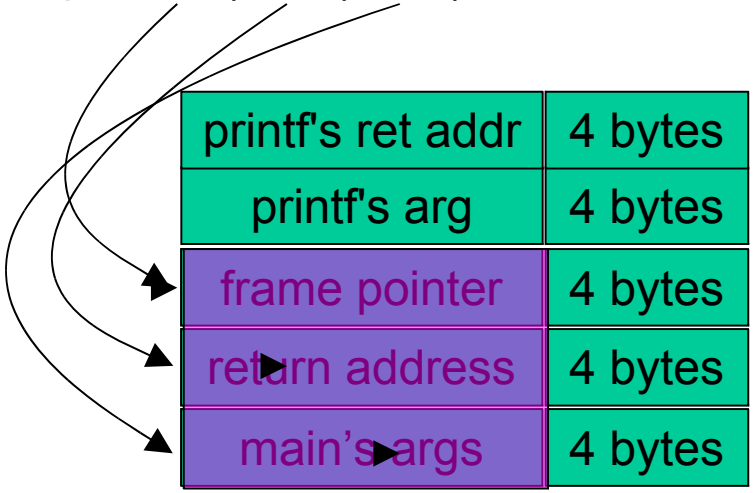
Turning bugs into primitives

reading

format string:

```
int main(int argv, char **argc) {
    printf(argc[1]);
}
```

./example %08x|%08x|%08x|...



stack reading primitive

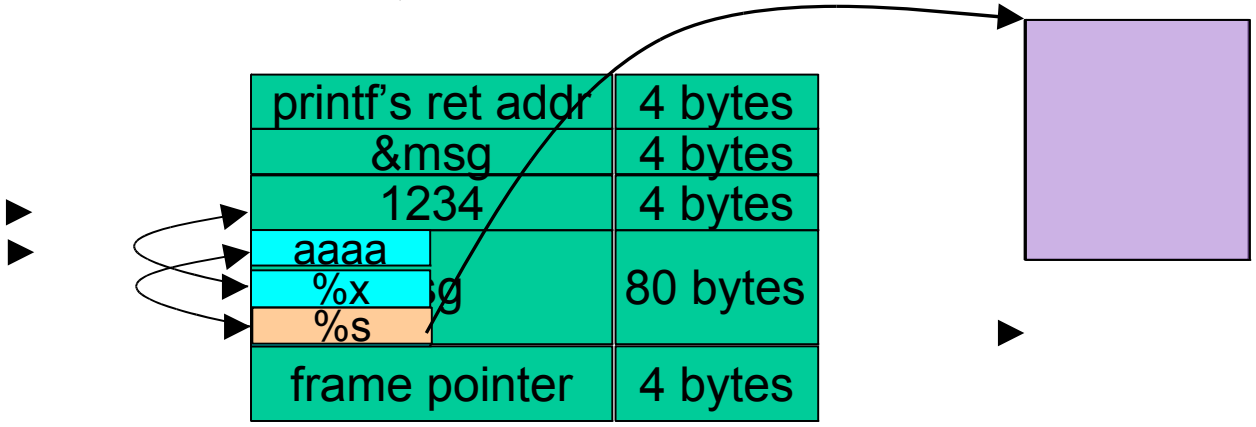
Turning bugs into primitives

reading

Format string bug:

```
int main(int argv, char **argc) {
    char msg[80];
    strcpy(msg, argc[1]);
    printf(msg,1234);
}
```

./example aaaa%x%s



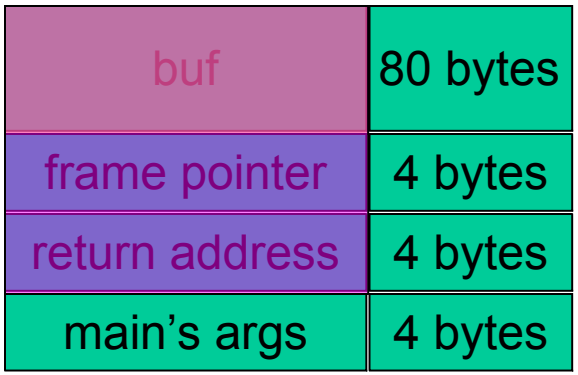
read-anywhere primitive

Turning bugs into primitives

reading

Unterminated string:

```
int main(int argv, char **argc) {
    char buf[80];

    strcnpy(buf,argc[1],sizeof buf);
    printf("%s", buf);
}
```



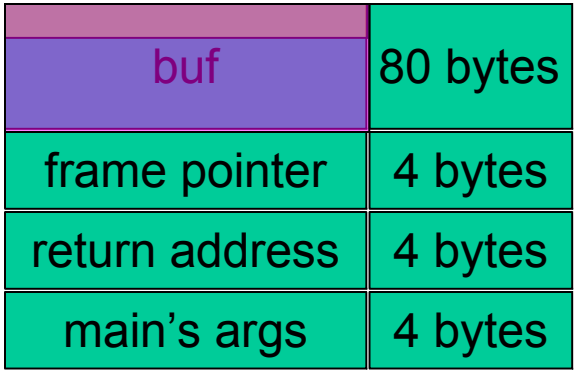
stack or heap reading primitive

Turning bugs into primitives

reading

Fixed size write():

```
int main(int argv, char **argc) {
    char buf[80];

    strlcpy(buf, argc[1], sizeof buf);
    write(1, buf, sizeof buf);
}
```



stack or heap reading primitive

Turning bugs into primitives

other

Other primitives:

restart the target application (crash)
consume in-process memory
pre-forking vs. post-forking daemons

consume system memory
consume processor usage
consume network bandwith
consume file system / file access
signal dispatching (divert execution flow)

Questions ?

Turning bugs into primitives

transforming

transforming primitives:

write --> jump
write --> read
write --> crash

jump --> crash
jump --> read
jump --> processor usage / any other?

signal --> crash
signal --> write
write + signal --> jump

read --> crash

crash --> signal (SIGSEGV) --> etc...

Turning bugs into primitives

transforming write--►read ►

binary search

```
class Exploit:
    def _write(self, addr, data):
        ...

    def isWritable(self, addr):
        self._write(self, addr, '1234')
        return self.targetCrashed()

    def get_textTop(self):
        addr      = 0x8048000
        delta     = 0x0080000
        while delta:
            addr += delta
            if self.isWritable(addr):
                addr -= delta
            delta >>= 1
        return addr+4
```

Turning bugs into primitives

transforming jump-->read

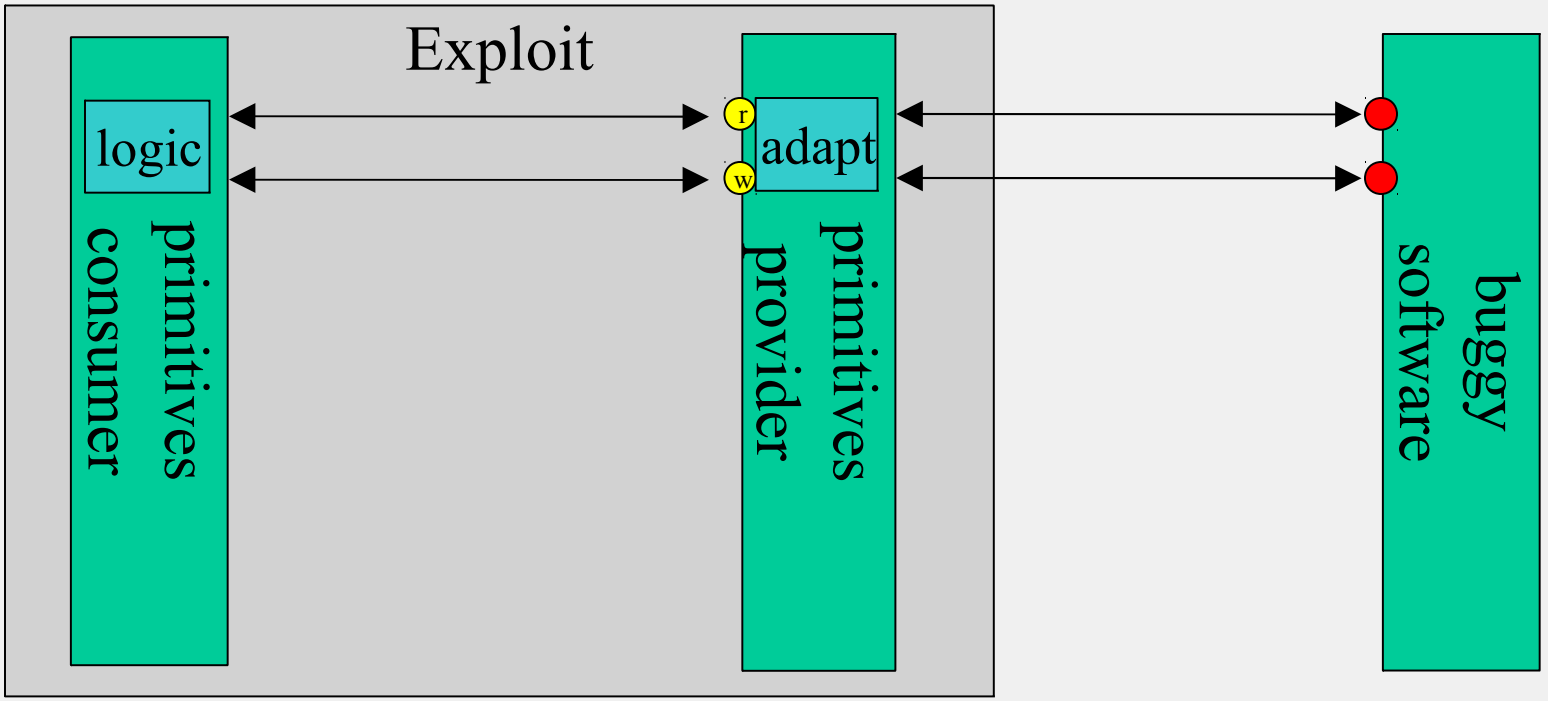
```
{
    ...
    printf("Magic Number:
    %d\n",mn);
    ...
}
```

```
...
8048804 mov  -4(%ebp),%eax
8048808 push %eax
804880a push 0x8044430
804880f call printf
...
```

```
class Exploit:
    def _jump(self, addr, data):
        ...

    def _read_codeAddr(self):
        self._jump(0x8048808)
        ...
```

Questions ?

Ideas ?



Turning primitives into exploits

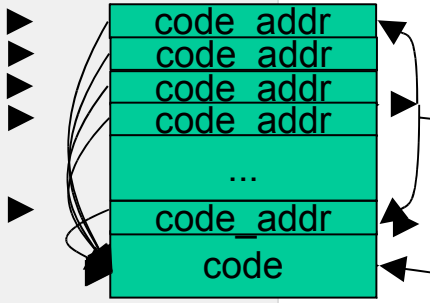
Turning primitives into exploits

write+jump

```
class Exploit:
    def _write(self, addr, data):
        ...

    def maxToWrite(self):
        return 1000

    def _jump(self, addr):
        ...

    def tryAttack(self):
        if len(code) > self.maxToWrite():
            raise “shellcode is too big”

        self._write(0x08084488, code)
        self._jump(0x08084488)
```

Turning primitives into exploits

write

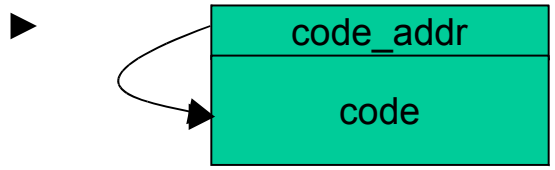



```
class Exploit:
    def _write(self, addr, data):
        ...

    def maxToWrite(self):
        return 1000

    def tryAttack(self, addr):
        addr = self.get_textTop()
        while not self.done():
            padd = self.maxToWrite() - len(code)
            code_addr = addr + padd
            buf = string(code_addr)*(padd/4)
            buf = buf + code
            self._write(addr, buf)

            addr += padd
```

Turning primitives into exploits

read + write

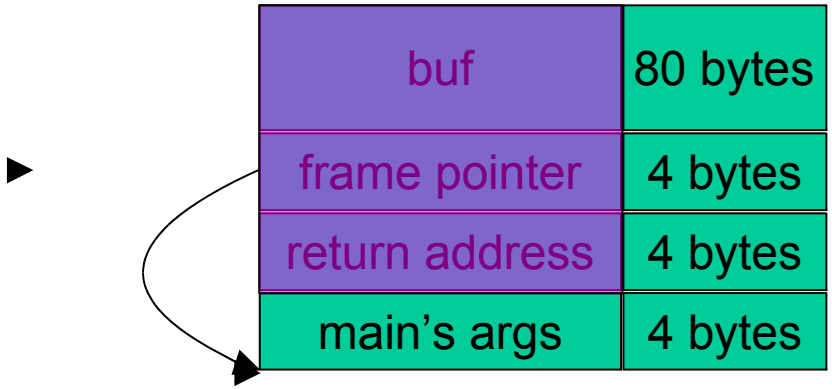
```
class Exploit:
    def _write(self, addr, data):
        ...
    def _read(self, addr, len):
        ...
    def tryAttack(self):
        if self._read(0x8048000,3) != 'ELF':
            raise "ELF not found"

        addr = self.find_GOT("exit")

        self._write(addr, string(addr+4)+code)
```



Turning primitives into exploits

stack-read

```
class Exploit:
    def _stackRead(self):
        ...
    def tryAttack(self):
        stack = self._stackRead()
        fp = stack.longAt(BUFSIZE)
        code_addr = fp – BUFSIZE – 3*4
        self._writeSomewhere(code)
        self._jump(code_addr)
        ...
```



Questions ?

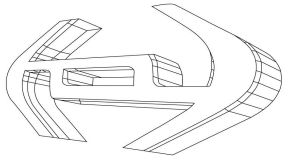
Ideas ?

Extra

- Higher level language for exploits
- Modeling the vulnerable program
- Natural evolution of security bugs
- Data inference

Thank You!

Gerardo Richarte | gera@coresecurity.com



CORE SECURITY TECHNOLOGIES · Offices Worldwide

Headquarters
44 Wall Street | 12th Floor
New York, NY 10005 | USA
Ph: (212) 461-2345
Fax: (212) 461-2346
info.usa@corest.com

Florida 141 | 2º cuerpo | 7º piso
(C1005AAC) Buenos Aires
Tel/Fax: (54 11) 4878-CORE (2673)
info.argentina@corest.com

Rua do Rócio 288 | 7º andar
Vila Olímpia | São Paulo | SP
CEP 04552-000 | Brazil
Tel: (55 11) 3054-2535
Fax: (55 11) 3054-2534
info.brazil@corest.com

www.corest.com