ELEC 377 – Operating Systems

Week 10 – Class 3

Last Class

- Security

Today

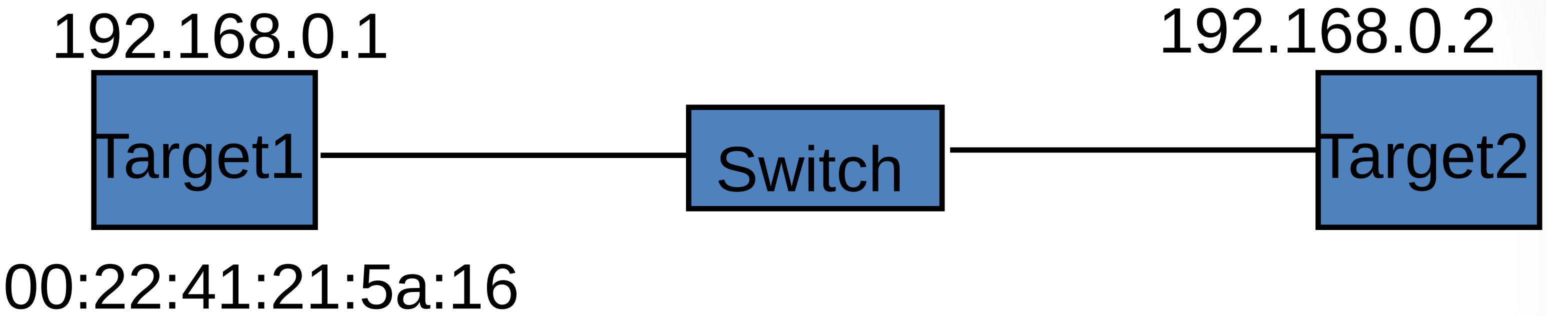
- Security (cont'd)

Network Security

- Eavesdropping
  - ◊ WAR driving
  - ◊ WEP Vulnerability
  - ◊ Switches only route to specific ethernet addresses
    - ARP poisoning

Network Security

- ARP Poisoning



192.168.0.1

Target1

Switch

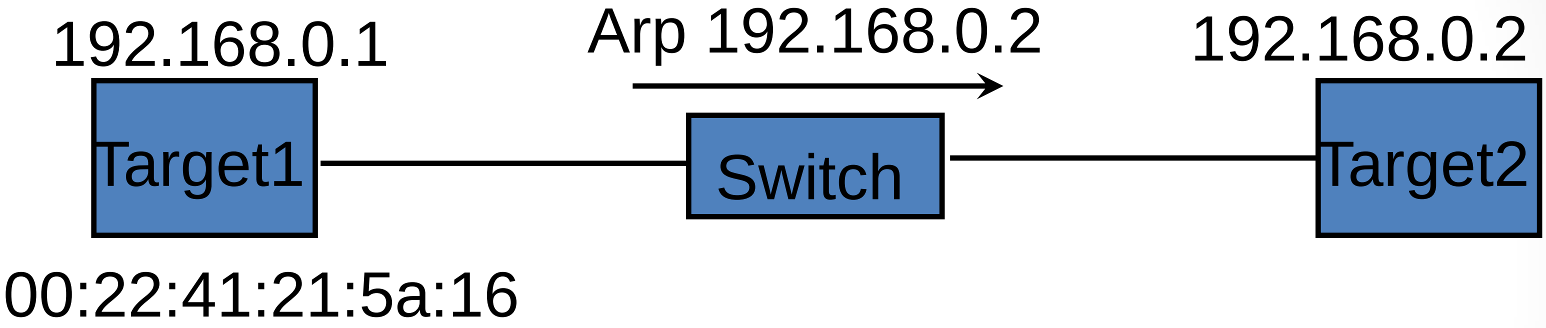
192.168.0.2

Target2

00:22:41:21:5a:16

Network Security

- ARP Poisoning



192.168.0.1

Arp 192.168.0.2

192.168.0.2

Target1

Switch

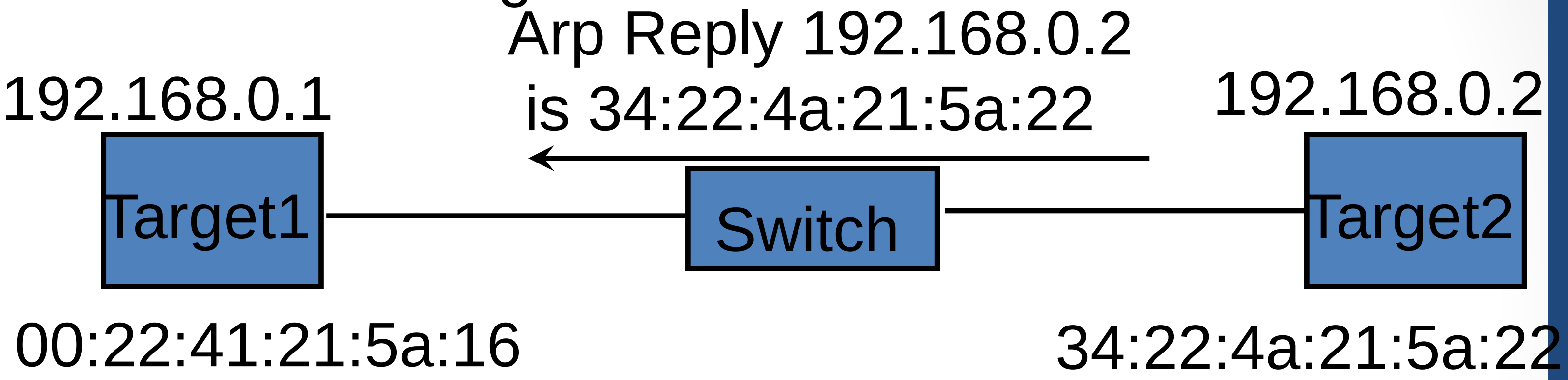
Target2

00:22:41:21:5a:16

Note: Arp is a broadcast packet

Network Security

- ARP Poisoning



Arp Reply 192.168.0.2
is 34:22:4a:21:5a:22

192.168.0.1

192.168.0.2

Target1

Switch

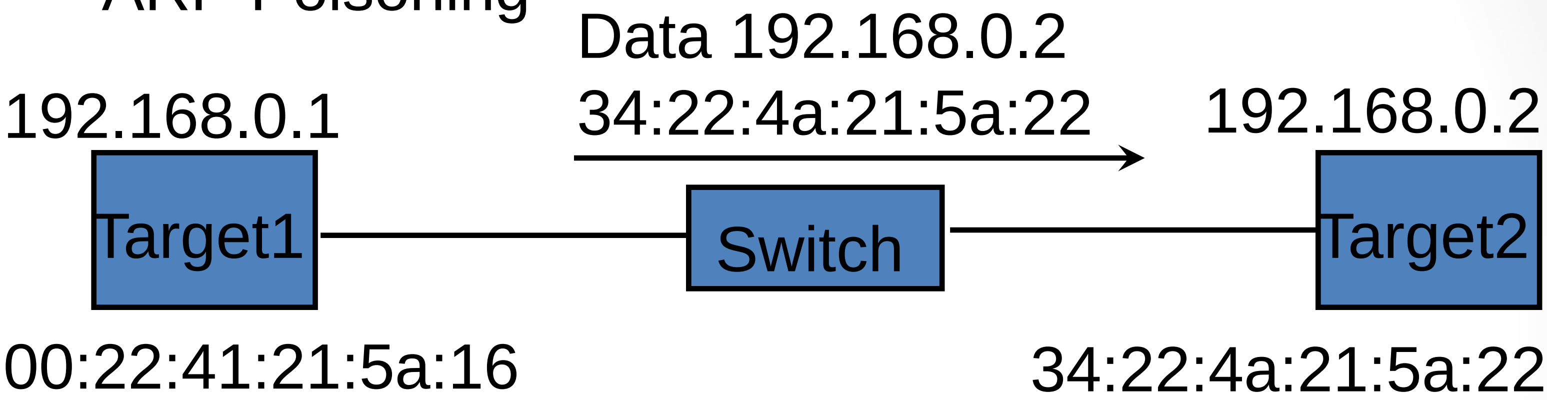
Target2

00:22:41:21:5a:16

34:22:4a:21:5a:22

Network Security

- ARP Poisoning



Data 192.168.0.2

34:22:4a:21:5a:22

192.168.0.1

192.168.0.2

Target1

Switch

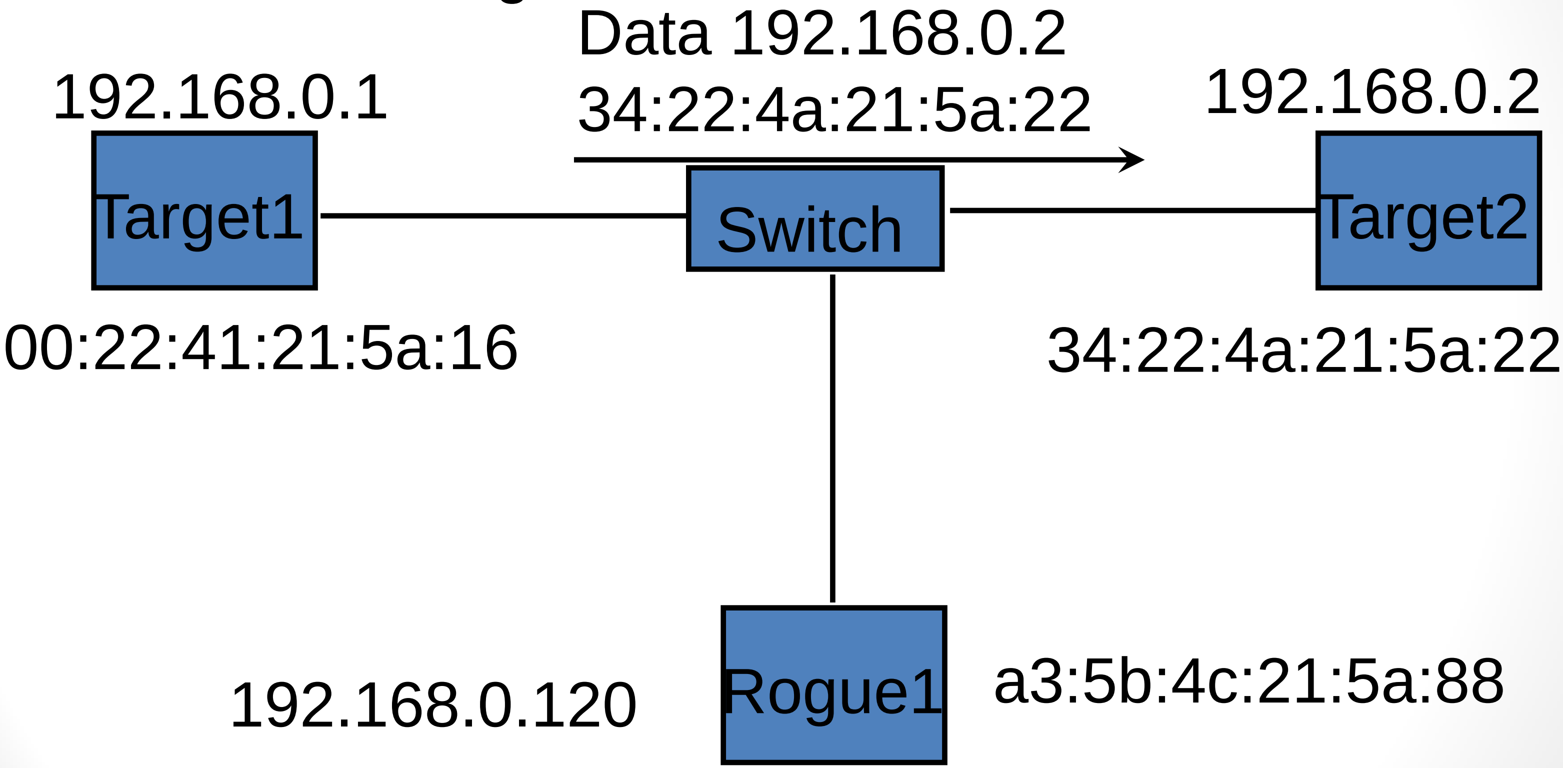
Target2

00:22:41:21:5a:16

34:22:4a:21:5a:22

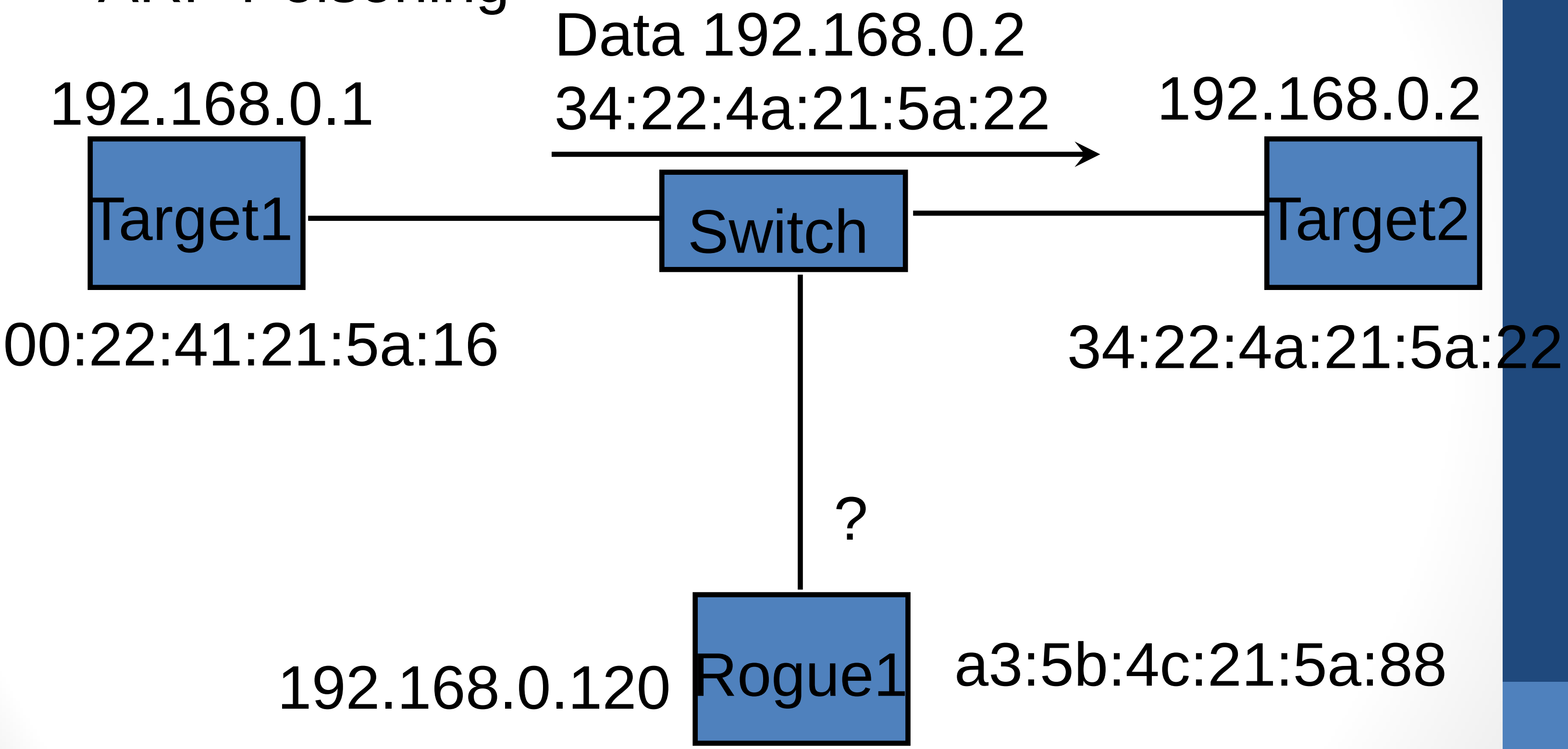
Network Security

- ARP Poisoning



Network Security

- ARP Poisoning

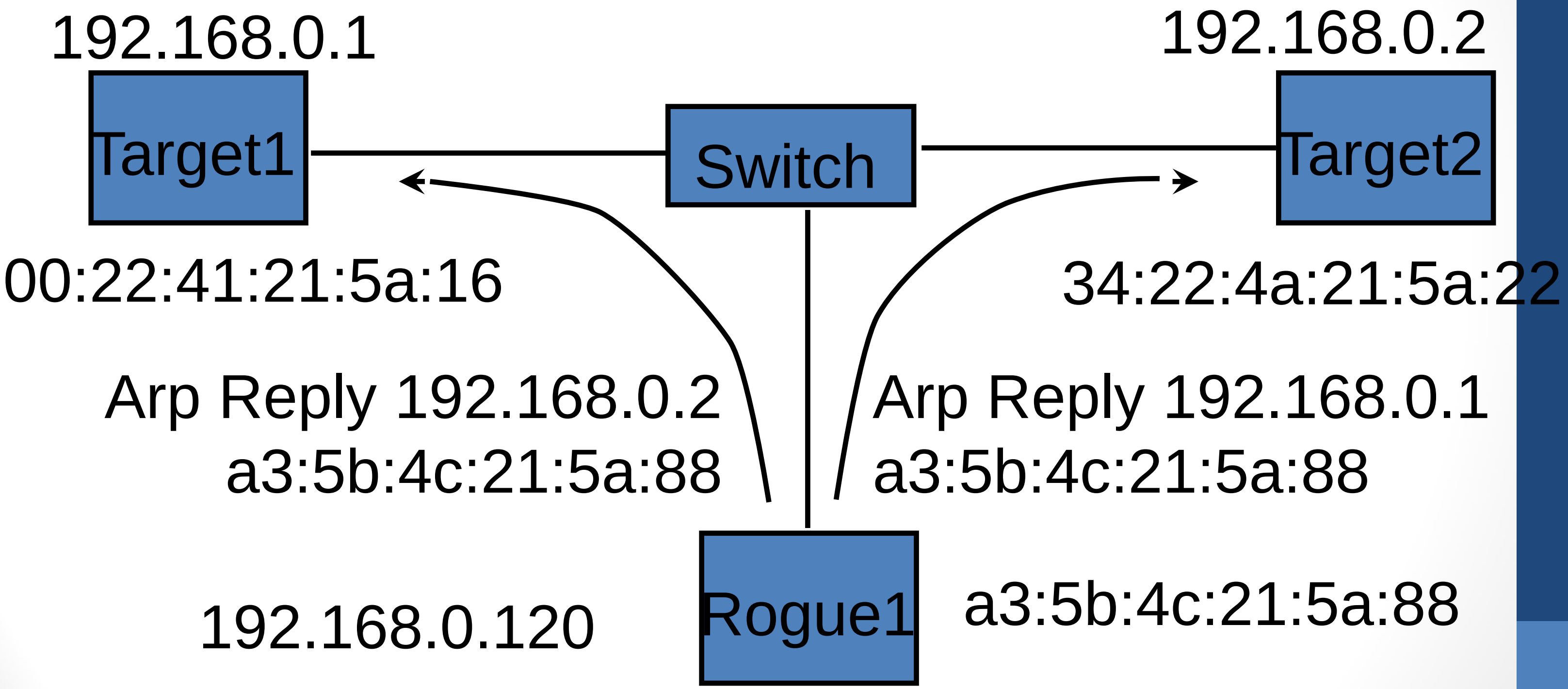


Data 192.168.0.2

34:22:4a:21:5a:22

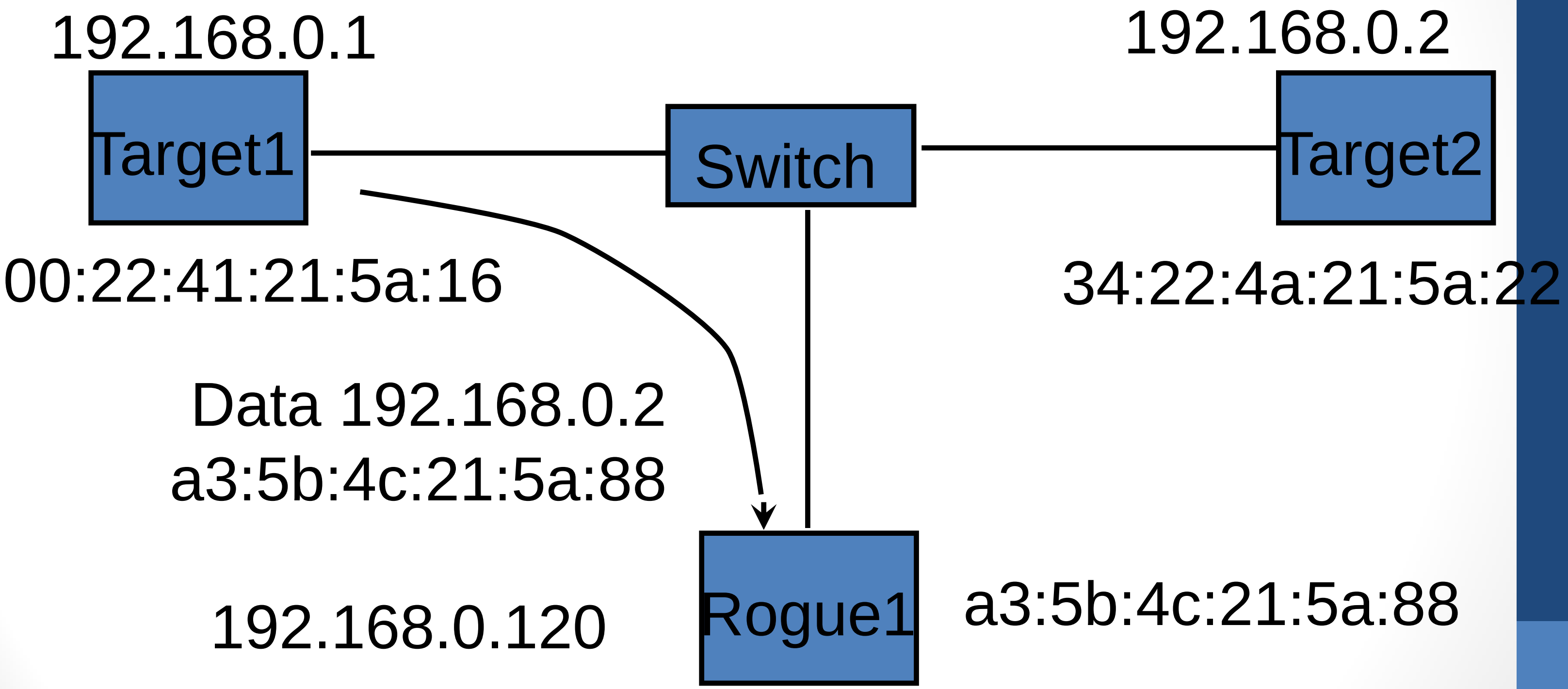
Network Security

- ARP Poisoning



Network Security

- ARP Poisoning



192.168.0.1

Target1

00:22:41:21:5a:16

Switch

192.168.0.2

Target2

34:22:4a:21:5a:22

Data 192.168.0.2
a3:5b:4c:21:5a:88

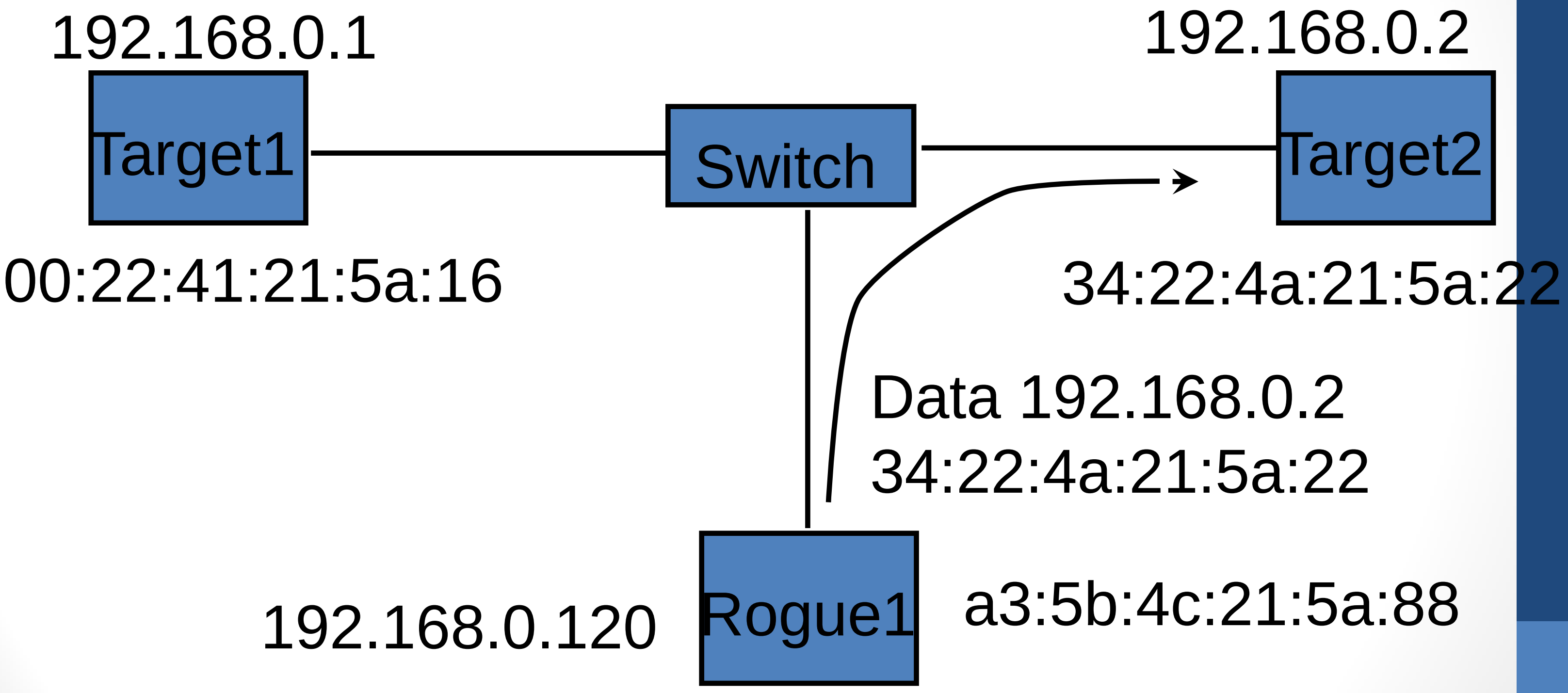
192.168.0.120

Rogue1

a3:5b:4c:21:5a:88

Network Security

- ARP Poisoning



Arp Poisoning

- Protections
  ◊ Don’t use replies you did not ask for.
  ◊ If MACs change unexpectedly, log changes, so a record available.

Network Security

- Eavesdropping
  - ◊ WAR driving
  - ◊ WEP Vulnerability
  - ◊ Switches only route to specific ethernet addresses
    - - ARP poisoning
    - - MAC Flooding
  - ◊ unencrypted protocols
    - - ftp, telnet
  - ◊ encrypted protocols
    - - sftp, scp, ssh

Network Security

- Other Network Attacks...
  - smurf attack
    - ping response....
  - oversize ICMP packet
    - ICMP packet that is too big....
  - Xmas Tree Packets
    - turn on all of the flags
      - ACK, SYN, etc..

Network Security

- pharming
  - ◊ reverse proxy for a online bank/Paypal
  - ◊ compromise a DNS server/Or DHCP server
    - - new attack, DNS poisoning
  - ◊ point bank/Paypal at your reverse proxy
  - ◊ pass transactions through to the bank
    - - but record information for later use.
    - - security images???
  - ◊ compromise router
    - - backbone routers
    - - cosumer grade routers
  - - DLINK advertising...

Authentication

- Passwords
  - ◊ main login
  - ◊ access to resources (databases, Unix groups)
- Vulnerable
  - ◊ guessing - most user chosen passwords are easy to remember, short, easy to guess
    -WPA interface
  - ◊ shoulder surfing (ATM hack)
  - ◊ packet sniffing (conferences)
  - ◊ masquerade
  - ◊ account sharing
- System generated?
  - ◊ too hard to remember?

Passwords

- Must store to verify?
  - ◊ If passwords are stored on OS must be secure
  - ◊ encrypted passwords (websites??/London)
  - ◊ one way encryption
    - - how to check?
    - - safe???
  - ◊ brute force attack (Dictionary Attack)
  - ◊ public file?

    /etc/secure

Passwords

- One Time Passwords
  - ◊ challenge response
    - – hardware key
  - ◊ one time pad
    - – list of random numbers
    - – early on-line banking

- Biometrics
  - ◊ Fingerprints, retina, iris
  - ◊ replay attacks?
  - ◊ major disadvantage

Passwords

- Biometrics
  - ◊ Fingerprints, retina, iris
  - ◊ accuracy

    - false positives (identifies me as you)

    - false negatives (denies you)
  - ◊ anonymity (my yahoo account is anoymous)
  - ◊ multiple accounts

    high security/low security

    - limited number of biometric keys

Passwords

- Biometrics
  - ◊ false sense of security

    - thermal sensors

    - repudiation
  - ◊ replay attacks?
  - ◊ fake fingers

    - silicone fingers

  Tsutomu Matsumoto of Yokohama National University

  - Gelatin fingers (same electrical characteristics as flesh)

  - can be made from finger prints left on any object

About Accuracy

- accuracy - what does it mean?
- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 40 percent positive detection (finds 40%) (400 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that someone identified as a terrorist is a terrorist?

About Accuracy

- accuracy - what does it mean?
- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 40 percent positive detection (finds 40%) (400 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that someone identified as a terrorist is a terrorist?

400/30,000 = 1.32 %

About Accuracy

- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 70% positive detection (700 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that someone identified as a terrorist is a terrorist?

About Accuracy

- 300 Million People in the USA
- Assume 1000 terrorists (1 per 300,000 = .00033%)
- Assume 70% positive detection (700 terrorists)
- Assume 0.01% misidentification (30,000 people)

So What is the chance that somone identified as a terrorist is a terrorist?
700/30,000 = 2.3%

Program Threats

- Trojan Horse
  - ◊ game program that sends the contents your mail box to another server
  - ◊ utility that wipes out your accounting program (DOS)

- Masquerade
  - ◊ special type of trojan horse
  - ◊ pretends to be a valid service
  - ◊ login masquerade
  - ◊ web site masquerade (spelling error/email)

Program Threats

- Trap Door/Back Door
  - ◊ Intentional hole left by programmer
  - ◊ Hard coded account numbers or Ids
  - ◊ War Games (Matthew Broderick)

Buffer Overflow (Globals)

- Variants

◊ function pointers in the heap within range of a global buffer (simple overwrite)

```
 char buffer[1024];
 struct proc_dir{
    int (*read_proc)(char *page, char**start...)
} theProcDir;
```

◊ theProcDir is after buffer in memory, overwrite read_proc variable, next time called, calls our code

Buffer Overflow (Globals)

- Variants

◊ vtable pointers (C++)

```
class A {                                    class B: public A {
   virtual int foo(){....};                  virtual int foo(){....};
   int bar(){.....};                                             int
  bar(){.....};
}                                                                 }
```

- call bar is known at compile time (called directly)
- foo is based on type of instance in variable
- called through a global table of functions

Buffer Overflow in the Heap

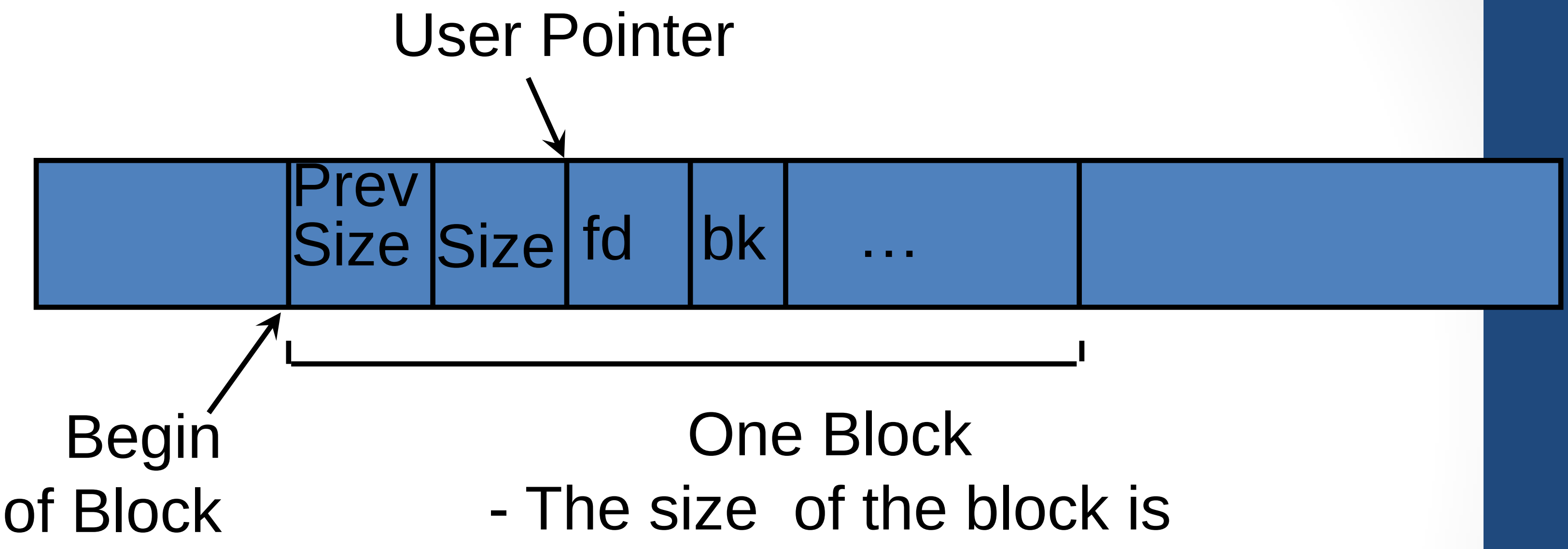
- What if the buffer is in the heap (after pointers)?
  - unused memory is kept in bins based on size of block
  - each bin is represented by a double linked list

```
#define INTERNAL_SIZE_T size_t

struct malloc_chunk {
    INTERNAL_SIZE_T prev_size;
    INTERNAL_SIZE_T size;
    struct malloc_chunk * fd;
    struct malloc_chunk * bk;
};
```

This section based on “Smashing the Heap for Fun and Profit”, Michel "MaXX" Kaempf, http://doc.bughunter.net/buffer-overflow/heap-corruption.html

Heap Data Structure

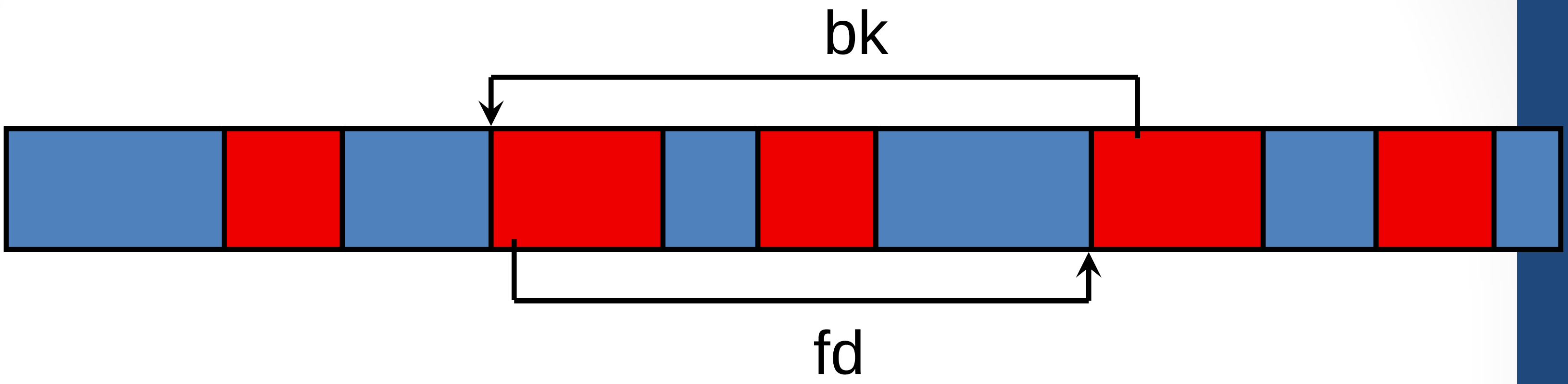


- The size of the block is given by the Size field

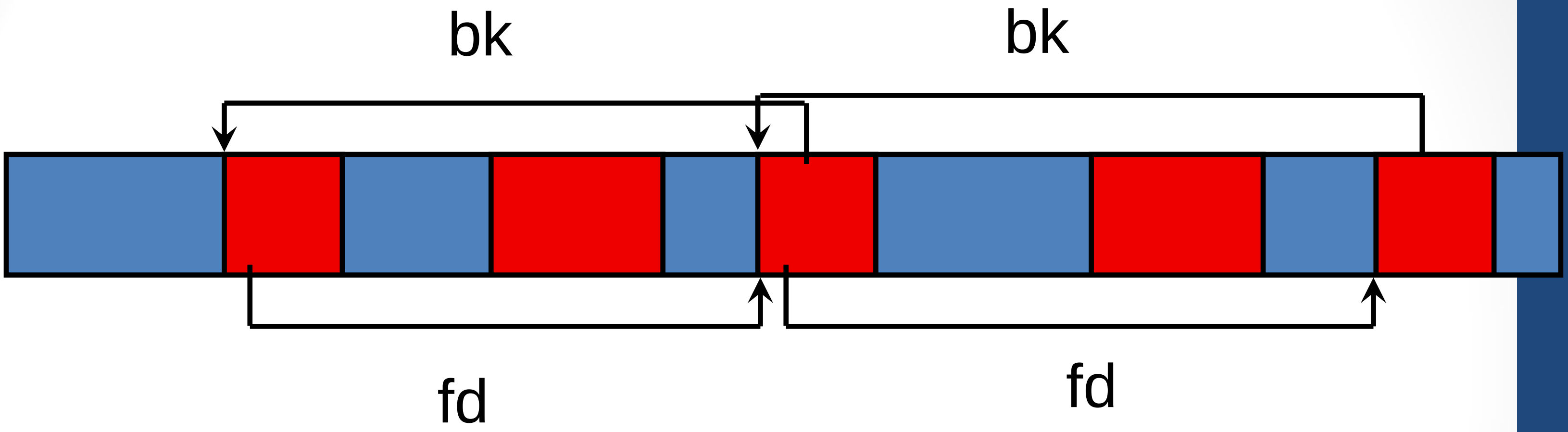
fd (forward) and bk (backward) are only used when the block is unallocated

Prev Size and Size are always used

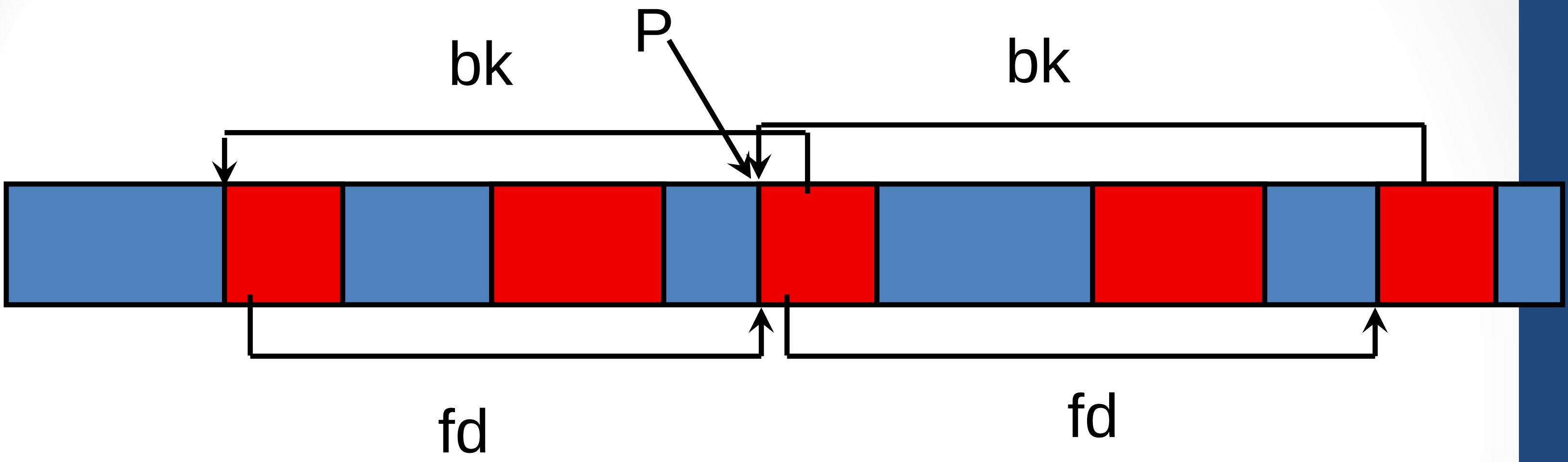
Linking Blocks



Linking Blocks

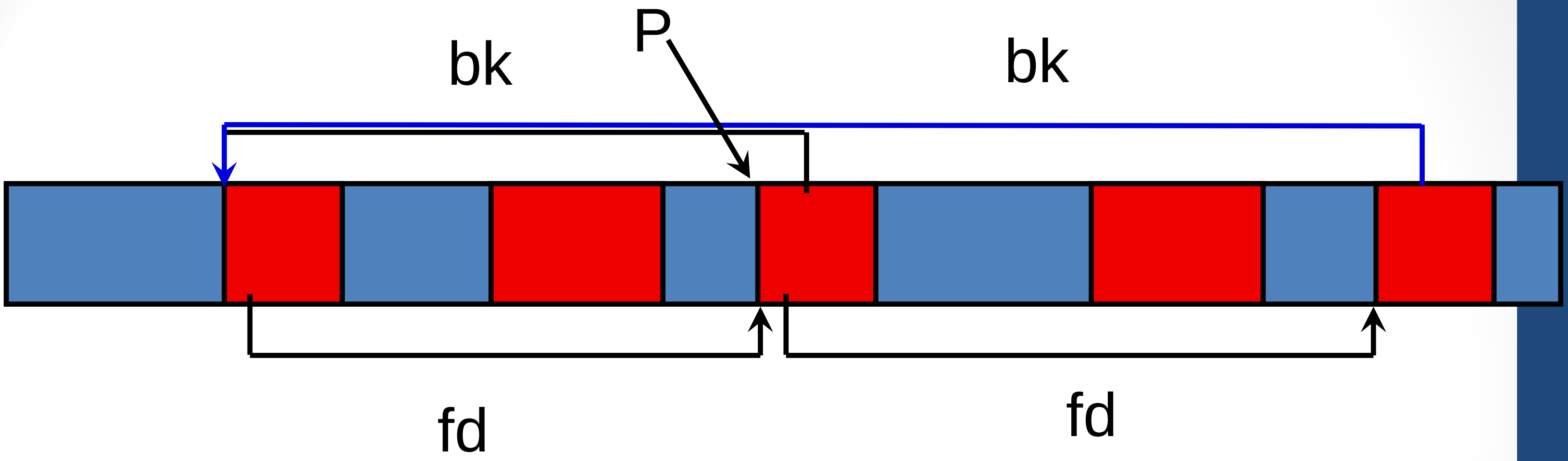


Unlinking



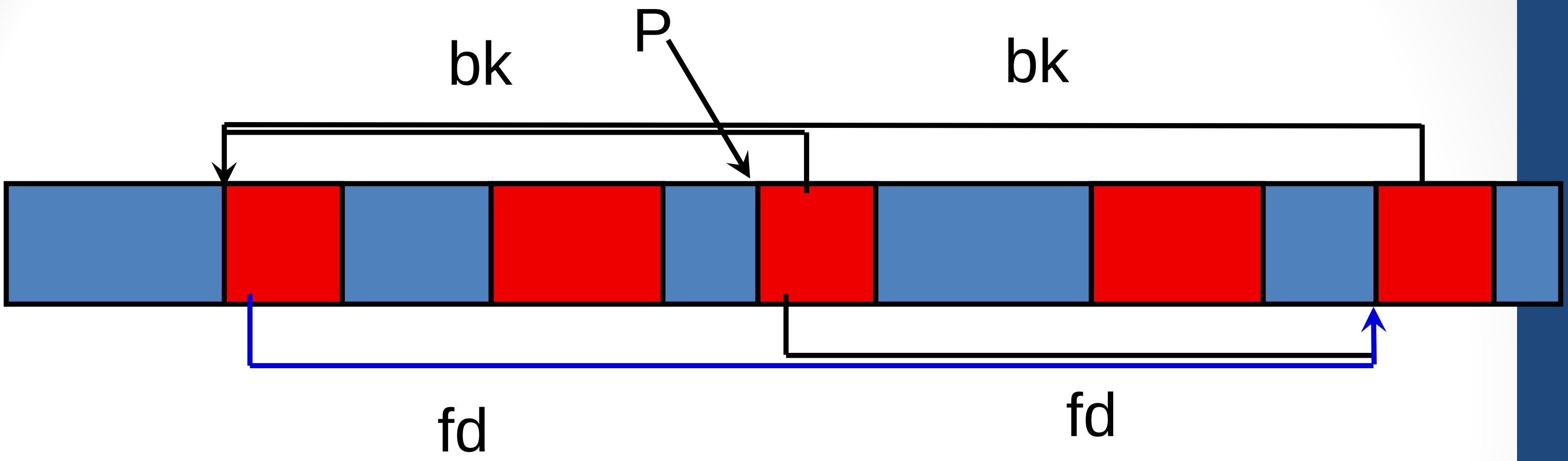
```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                \
  FD = P->fd;               \
  FD->bk = BK;               \
  BK->fd = FD;              \
}
```

Unlinking



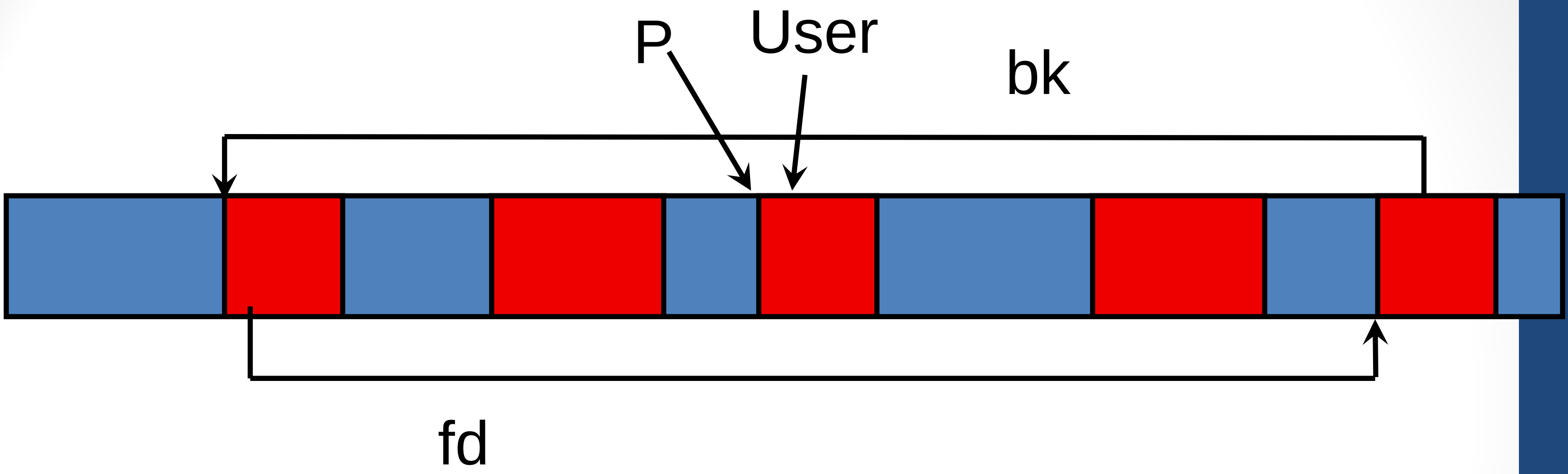
```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                 \
  FD = P->fd;                 \
  FD->bk = BK;                \
  BK->fd = FD;                \
}
```

Unlinking



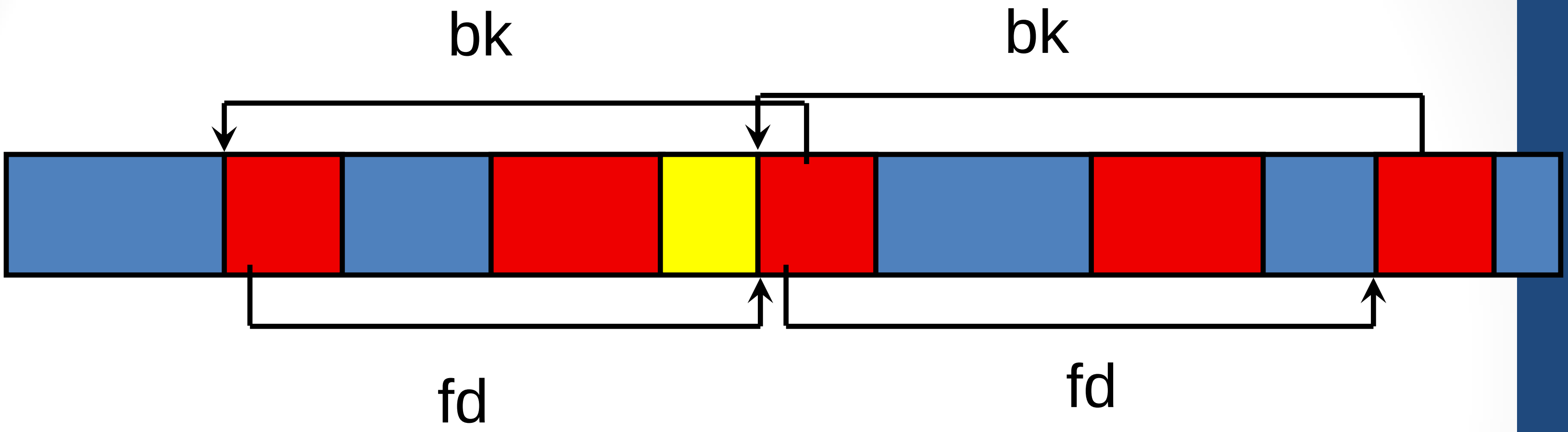
```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                \
  FD = P->fd;                \
  FD->bk = BK;               \
  BK->fd = FD;               \
}
```

Unlinking

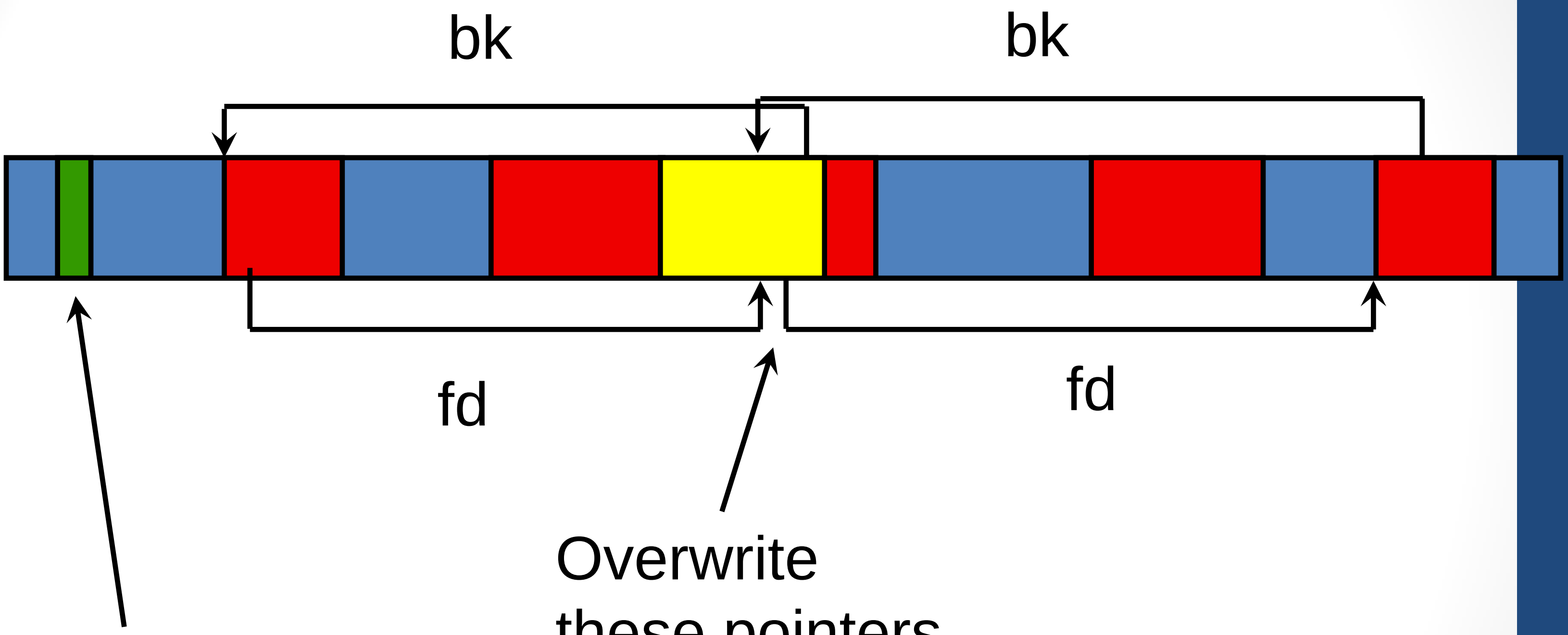


```
#define unlink( P, BK, FD ) { \
  BK = P->bk;                    \
  FD = P->fd;                    \
  FD->bk = BK;                    \
  BK->fd = FD;                    \
}
```

The Vulnerable Buffer



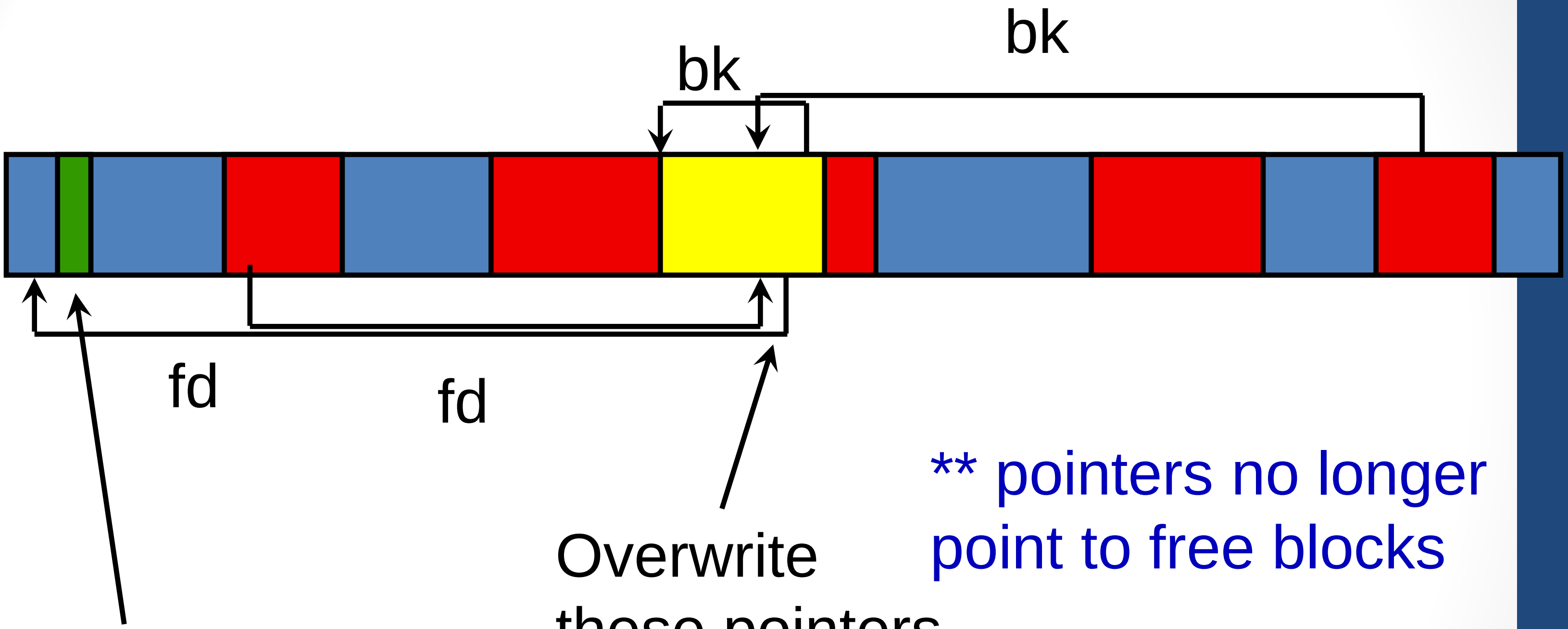
The Vulnerable Buffer



Overwrite
these pointers

A function
pointer used
by program (e.g. malloc_hook)

The New Pointers



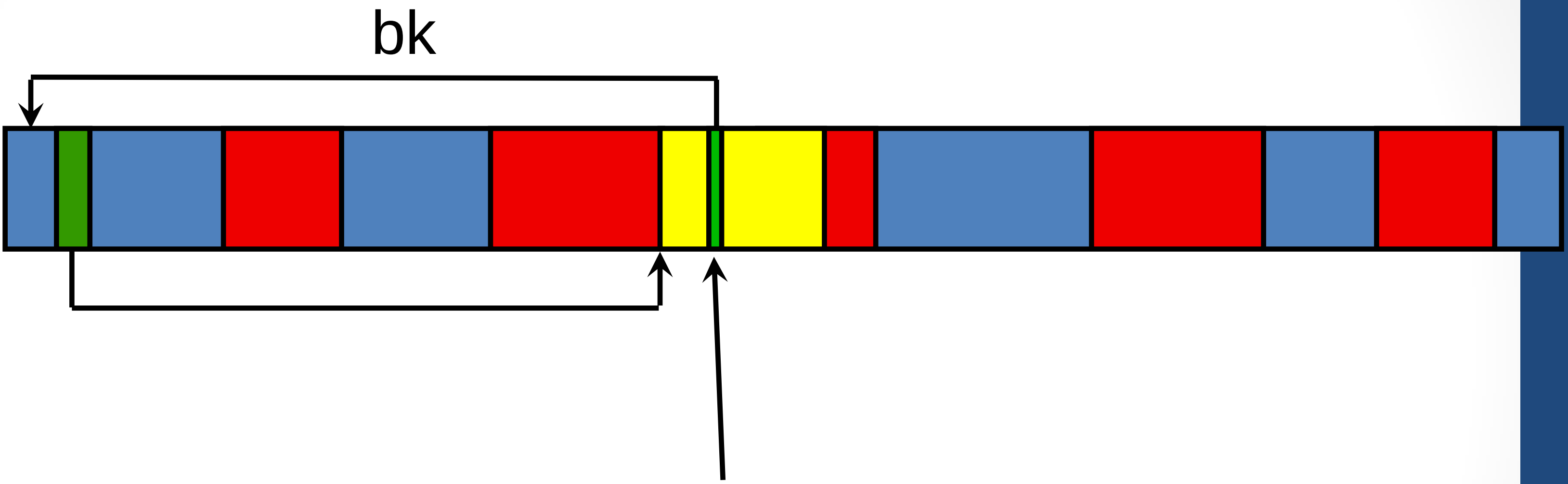
Overwrite these pointers

** pointers no longer point to free blocks

A function pointer used by program (e.g. _malloc_hook) -- located fd+8

** wait for a malloc call.......

After Unlinking...



- 4 bytes at offset 8 get overwritten
- shell code has to jmp around..

**Next time the function pointer is used...
Our code gets executed!!