Network Security

Attacks Against the TCP/IP Stack

Christopher Kruegel
Spring 2020
UC Santa Barbara
(1) The TCP/IP model
(2) Link-layer security
(3) Internet-layer security
(4) Transport-layer security
(5) Wireless security
The TCP/IP Model
THE ARPA NETWORK

DEC 1969

4 NODES
TCP/IP Model

Figure 1: TCP/IP model
Link-Layer Security
struct EthernetFrame {
    mac_destination: [u8; 6],
    mac_source: [u8; 6],
    vlan_tag: Option<u32>,
    type_or_length: u16,
    payload: Vec<u8>,
    crc: u32,
}

- Ubiquitous link-layer medium for wired networks
- Frames labeled with source and destination physical (MAC) addresses
**Ethernet (802.3) [1]**

```rust
g struct EthernetFrame {
    mac_destination: [u8; 6],
    mac_source: [u8; 6],
    vlan_tag: Option<u32>,
    type_or_length: u16,
    payload: Vec<u8>,
    crc: u32,
}
```

- Ubiquitous link-layer medium for wired networks
- Frames labeled with source and destination physical (MAC) addresses

How are IP addresses translated to physical addresses?
Address Resolution Protocol (ARP) [2]

- Protocol to establish IP ↔ physical address mapping
- Mappings are cached and commonly preemptively established (*gratuitous ARP*)
ARP Cache

$ ip neigh
10.254.1.1  dev wlp58s0  lladdr 8c:ae:4c:ff:c6:06  REACHABLE
10.254.1.2  dev wlp58s0  lladdr 2c:30:33:9e:33:22  STALE
10.254.1.150 dev wlp58s0  lladdr b4:f6:1c:53:8d:b9  STALE
10.254.1.120 dev wlp58s0  lladdr 50:f5:da:57:19:62  STALE
10.254.1.118 dev wlp58s0  lladdr 00:1e:06:34:96:cd  STALE
10.254.1.176 dev wlp58s0  lladdr d8:50:e6:87:d6:ca  STALE
10.254.1.189 dev wlp58s0  lladdr 18:b4:30:57:5d:c6  STALE
ARP Cache

$ ip neigh
10.254.1.1   dev wlp58s0  lladdr 8c:ae:4c:ff:c6:06  REACHABLE
10.254.1.2   dev wlp58s0  lladdr 2c:30:33:9e:33:22  STALE
10.254.1.150  dev wlp58s0  lladdr b4:f6:1c:53:8d:b9  STALE
10.254.1.120  dev wlp58s0  lladdr 50:f5:da:57:19:62  STALE
10.254.1.118  dev wlp58s0  lladdr 00:1e:06:34:96:cd  STALE
10.254.1.176  dev wlp58s0  lladdr d8:50:e6:87:d6:ca  STALE
10.254.1.189  dev wlp58s0  lladdr 18:b4:30:57:5d:c6  STALE

How trustworthy are ARP messages?
ARP Cache Poisoning

Figure 3: ARP cache poisoning

- Hosts can inject arbitrary ARP messages!
  - Can spoof messages from existing hosts
  - Can redirect traffic from legitimate host to attacker
Switched Ethernet learns physical address ↔ port mappings to partition LANs and avoid broadcast behavior
- Mappings contained in a content-addressable memory (CAM)
- But, ARP cache poisoning requires victim addresses!
- MAC flooding can overflow the CAM
Broadcast vs. Switched Ethernet

- Switched Ethernet learns physical address $\leftrightarrow$ port mappings to partition LANs and avoid broadcast behavior
  - Mappings contained in a content-addressable memory (CAM)
- But, ARP cache poisoning requires victim addresses!
- MAC flooding can overflow the CAM

How should a switch respond to this condition?
Virtual Local Area Networks (VLAN, 802.1Q)

```rust
struct EthernetFrame {
    // ...
    vlan_tag: Option<u32>,
    // ...
}
```

- Mechanism for partitioning physical networks into separate broadcast domains
- If enforced at network switches, operators can isolate untrusted traffic → compartmentalization
- Contains damage of an endpoint compromise
Internet-Layer Security
Internet Protocol [3]

– Standard for “internetworking” ⇝ “Internet”
– Standardized addresses and routing
  – v4: 4 bytes, v6: 16 bytes
  – e.g., 127.0.0.1 ≡ 0x7f000001
– Packet abstraction
  – Datagrams can be dropped or reordered
  – Payloads can be corrupted
**IPv4 Packet Format**

```
struct IPv4Datagram {
    version: u4, header_length: u4, differentiated_services: u6,
    explicit_congestion: u2, total_length: u16, identifier: u16,
    reserved: bool, do_not_fragment: bool, more_fragments: bool,
    fragment_offset: u13, time_to_live: u8, protocol: u8,
    header_checksum: u16, source_address: u32, destination_address: u32,
    options: Vec<u8>,
}
```
IP Addresses

- IP addresses are used as access control principals
  - Host-based authentication
  - Security middlebox whitelists, blacklists
- But, are IP addresses an appropriate authentication factor?
IP Addresses

- IP addresses are used as access control principals
  - Host-based authentication
    - Security middlebox whitelists, blacklists
- But, are IP addresses an appropriate authentication factor?
  - Absolutely not!
    - Addresses can be spoofed
    - Addresses can be reassigned
    - (Under what constraints?)
IP Flooding

– IP flooding is a canonical denial-of-service (DoS) attack

– One version
  – Many attackers send ICMP PING messages to a victim
  – Victim’s uplink bandwidth is overwhelmed

– Another (stealthier) version
  – Many attackers send ICMP PING messages to other machines with the victim as the source IP address
  – Victim’s uplink bandwidth is overwhelmed without revealing the attackers’ identities
DoS Amplification

Figure 4: Smurf attack
IP Spoofing Defenses

- Detection
  - Backscatter via network telescopes
  - Traceback using unused IP packet header fields
- Filtering
  - Egress checks (e.g., validation against local network allocations)
  - Ingress checks (e.g., validation against routing tables)
  - Reverse path forwarding
Unicast Reverse Path Forwarding (uRPF) [4]

Idea: Validate source addresses against the routing table, and drop “unlikely” packets

**Strict**  Source must match the best route

**Feasible**  Source must match “reasonable” routes

**Loose**  Source can match any route
# uRPF Example

## Table 1: Example routing table

<table>
<thead>
<tr>
<th>Network</th>
<th>Next Hop</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.61.0.0/16</td>
<td>192.168.27.11</td>
<td>1000</td>
</tr>
<tr>
<td>10.61.72.0/24</td>
<td>192.168.31.2</td>
<td>1000</td>
</tr>
<tr>
<td>10.61.72.0/26</td>
<td>192.168.136.5</td>
<td>100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Strict uRPF, source 10.61.72.129, last hop 192.168.27.11? Feasible uRPF? Loose uRPF?
IP Fragmentation

- IP datagrams must tolerate varying segment maximum transmission units (MTUs)
- One strategy: datagrams are fragmented
  - Total length field set to fragment length
  - “More fragment” bit set unless last fragment
  - Fragment offset set to original offset / 8
  - Checksum recomputed
- Subsequent router or receiving endpoint reassembles fragments into the original packet
IP Fragment Reassembly

-Endpoints must buffer fragments for reassembly

(2^{16} - 1) - 20 = 65515 < 65528 = (2^{13} - 1) \cdot 8
IP Fragment Reassembly

- Endpoints must buffer fragments for reassembly
- Maximum size of an IP packet? Of a reassembled packet?

\[(2^{16} − 1) − 20 = 65515 < 65528 = (2^{13} − 1) \cdot 8\]
IP Fragment Reassembly

- Endpoints must buffer fragments for reassembly
- Maximum size of an IP packet? Of a reassembled packet?

\[
(2^{16} - 1) - 20 = 65515 < 65528 = (2^{13} - 1) \cdot 8
\]
Transport-Layer Security
Transmission Control Protocol (TCP) [5]

- TCP provides a stream abstraction on top of the IP layer
  - Bytes as opposed to datagrams
  - TCP transparently segments byte stream into packets
- Provides some reliability
  - Data integrity
  - In-order presentation of byte stream
  - Retransmission of dropped or corrupted packets
  - Flow and congestion control
struct TCPPacket {
    source_port: u16, destination_port: u16,
    sequence_number: u32, acknowledgment: u32,
    data_offset: u4, reserved: u3, ns: bool,
    cwr: bool, ece: bool, urg: bool, ack: bool,
    psh: bool, rst: bool, syn: bool, fin: bool,
    window_size: u16, checksum: u16, urg_ptr: u16,
    options: Vec<u8>,
}
TCP Connections

- TCP is connection-oriented
  - Clients connect to a server
  - Servers bind to a port, listen for connections, and accepts them
- TCP tracks both client- and server-side states
TCP State Diagram [6]
TCP Flags

**SYN**  Synchronize sequence numbers
**ACK**  Sequence number acknowledgment
**PSH**  Flush any buffered data
**FIN**  End of data from sender
**RST**  Reset the stream

- TCP flag bits dictate transitions through TCP state diagram
TCP Connection Establishment

Figure 6: TCP three-way handshake
**Data Transmission**

Figure 7: TCP data transmission

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Established connection</td>
<td></td>
</tr>
<tr>
<td>&lt;PSH</td>
<td>ACK, P_src, P_dst, n_i, S_i, S_j&gt;</td>
</tr>
<tr>
<td>&lt;ACK, P_dst, P_src, 0, 0, S_i + n_i&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;PSH</td>
<td>ACK, P_src, P_dst, n_j, S_i + n_i, S_j&gt;</td>
</tr>
<tr>
<td>&lt;ACK, P_dst, P_src, 0, 0, S_i + n_i + n_j&gt;</td>
<td></td>
</tr>
</tbody>
</table>
TCP Graceful Shutdown

**Figure 8: TCP graceful shutdown**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Established connection</td>
<td></td>
</tr>
<tr>
<td><code>&lt;FIN, P_src, P_dst, 0, S_i, S_j&gt;</code></td>
<td></td>
</tr>
<tr>
<td>`&lt;FIN</td>
<td>ACK, P_dst, P_src, 0, S_j, S_i + 1&gt;`</td>
</tr>
<tr>
<td><code>&lt;ACK, P_src, P_dst, 0, _, S_j + 1&gt;</code></td>
<td></td>
</tr>
<tr>
<td>Full-duplex shutdown</td>
<td></td>
</tr>
</tbody>
</table>
// Initialize a server address descriptor
struct sockaddr_in server_addr;
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr = htonl(0x7f000001);
server_addr.sin_port = htons(22);

// Create a TCP socket
int sk = socket(AF_INET, SOCK_STREAM, 0);

// Connect to the server
connect(sk, (struct sockaddr*) &server_addr, sizeof(server_addr));
// Initialize a server address descriptor

```c
struct sockaddr_in server_addr;
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr = INADDR_ANY;
server_addr.sin_port = htons(22);
```

// Create and bind a TCP server socket

```c
int server_sk = socket(AF_INET, SOCK_STREAM, 0);
bind(server_sk, (const struct sockaddr*) &server_addr, sizeof(server_addr));
listen(server_sk, 1);
```

// Accept a client connection

```c
struct sockaddr_in client_addr;
memset(&client_addr, 0, sizeof(client_addr));
socklen_t client_addr_len = sizeof(client_addr);
int client_sk = accept(server_sk, (struct sockaddr*) &client_addr, &client_addr_len);
```
Raw Sockets

- Raw sockets allow user programs to bypass layers of the networking stack
- Distinction between “raw” and “cooked” packets
- Raw socket creation requires high privilege (Why?)
TCP SYN Flooding

- Attackers send many SYN packets to a victim
  - Server creates transmission control blocks (TCB) to track connection states
  - Creates memory pressure and long connection queues
  - No analogous client-side requirement $\Rightarrow$ asymmetric advantage

- Defending against SYN floods not straightforward
  - Any defense must retain backwards-compatibility
**SYN Cookies [7]**

*Idea: Avoid the need to keep server-side state until a connection is fully opened*

1. Server encodes a “cookie” into its initial sequence number (ISN) \( n \)
2. Client response ACKs \( n + 1 \)
3. Server verifies that the cookie is correct
4. Connection state created from the decoded cookie
Let \( t \) be the current timestamp >> 6 bits, \( m \) be a 3-bit encoding of the initial MSS, and \( S_k \) be a keyed cryptographic 24-bit hash function. Then, given a connection tuple \( \langle a_{src}, p_{src}, a_{dst}, p_{dst} \rangle \), a SYN cookie \( c \) is

\[
c = (t \mod 32) \ || \ m \ || \ S_k(a_{src}, p_{src}, a_{dst}, p_{dst}, t).
\]
TCP Connection Resets

- Attackers can send RST packets to victims to destroy existing connections
  - Addresses and ports must be correctly spoofed
  - Sequence number must be within the destination’s receive window
- Simple and effective attack for an on-path network adversary
Figure 9: TCP connection reset attack example
What if an attacker is off-path; can a reset attack still be mounted?
TCP Connection Resets

– What if an attacker is off-path; can a reset attack still be mounted?

– Yes, by predicting sequence numbers!
  – One can assume that all other information is known or trivially guessable
  – Requires exploitable determinism in ISN selection (Why?)
TCP Connection Hijacking

- Attacks can hijack existing TCP connections
  - Requires sending forged packets to one host participating in the connection
  - If one spoofed packet is accepted by the receiver, the connection can be desynchronized from the other victim host
- More difficult than resetting connections
  - Victim hosts can easily reset the connection during the attack
TCP Connection Hijacking

Figure 10: TCP connection hijacking
Port Scanning

- Reconnaissance is often a prerequisite for launching attacks
- Port scanning is a technique for remotely gathering information on target networks
  - Topology, access control policy, network service availability and versions
  - Relies on combination of direct and side-channel leakage
- nmap is the *de facto* standard scanner
TCP SYN Scans

IP 10.254.1.119.32996 > 10.254.1.118.22: Flags [S], seq 1186093762, win 29200
IP 10.254.1.119.55054 > 10.254.1.118.23: Flags [S], seq 1770108639, win 29200
IP 10.254.1.118.22 > 10.254.1.119.32996: Flags [S.], seq 245736174, ack 1186093763, win 28960
IP 10.254.1.119.32996 > 10.254.1.118.22: Flags [..], ack 1, win 229
IP 10.254.1.119.32996 > 10.254.1.118.22: Flags [R.], seq 1, ack 1, win 229
IP 10.254.1.119.55098 > 10.254.1.118.23: Flags [S], seq 2435541012, win 29200
IP 10.254.1.119.33072 > 10.254.1.118.22: Flags [S], seq 2969806151, win 29200
IP 10.254.1.118.22 > 10.254.1.119.33072: Flags [S.], seq 4159765487, ack 2969806152, win 28960
IP 10.254.1.119.33072 > 10.254.1.118.22: Flags [..], ack 1, win 229
IP 10.254.1.119.33072 > 10.254.1.118.22: Flags [R.], seq 1, ack 1, win 229
TCP ACK Scans

IP 10.254.1.119.48479 > 10.254.1.118.22: Flags [..], ack 2601494423, win 1024
IP 10.254.1.118.22 > 10.254.1.119.48479: Flags [R], seq 2601494423, win 0
IP 10.254.1.119.48479 > 10.254.1.118.23: Flags [..], ack 2601494423, win 1024
IP 10.254.1.119.48480 > 10.254.1.118.23: Flags [..], ack 2601428886, win 1024
IP 10.254.1.118.22 > 10.254.1.119.48490: Flags [R], seq 2584716951, win 0
IP 10.254.1.118.22 > 10.254.1.119.48491: Flags [R], seq 2567939479, win 0
UDP Scans

IP 10.254.1.119.54491 > 10.254.1.1.54: UDP, length 0
IP 10.254.1.119.54491 > 10.254.1.1.53: 0 stat [0q] (12)
IP 10.254.1.119.54492 > 10.254.1.1.53: 0 stat [0q] (12)
IP 10.254.1.119.54492 > 10.254.1.1.54: UDP, length 0
IP 10.254.1.119.54493 > 10.254.1.1.54: UDP, length 0
IP 10.254.1.119.54493 > 10.254.1.1.53: 0 stat [0q] (12)
IP 10.254.1.119.54494 > 10.254.1.1.53: 0 stat [0q] (12)
IP 10.254.1.119.54494 > 10.254.1.1.54: UDP, length 0
IP 10.254.1.119.54495 > 10.254.1.1.53: 0 stat [0q] (12)
- Differences in TCP/IP implementations can be used to fingerprint remote operating systems [8, Ch. 8]
  - Due to specification ambiguity and implementation flaws
- Most services can also be trivially fingerprinted
  - Banners are very helpful
  - Services are also prone to information leakage
Remote Host Scanning

$ nmap -A 10.254.1.118
Nmap scan report for 10.254.1.118
Host is up (0.018s latency).

PORT    STATE     SERVICE       VERSION
22/tcp   open     ssh           OpenSSH 7.6 (protocol 2.0)
            | ssh-hostkey:
[...]
MAC Address: 00:1E:06:34:96:CD (Wibrain)
OS details: Linux 3.10 - 4.8, Linux 3.2 - 4.8
Network Distance: 1 hop
Firewalls

- Firewalls inspect network traffic and filter or modify it according to some predicates (a ruleset) [9]

**Packet filters**  Operate over individual packets

**Stateful filters**  Operate over connection abstraction

**Application-level**  Operate over application features
Intrusion Detection (IDS)

- General term for detecting attacks against systems, networks, and software
- Many ways to characterize an IDS

**Domain** Network, host, application events

**Models** Misuse/signature/negative vs. anomaly/positive

**State** Stateless vs. stateful
Detection Domains

- Detection is pattern matching over an event stream
  - Network packets, connection streams
  - System calls, control-flow transfers
  - Application events (e.g., HTTP messages, user logins)
- Detection over IDS alerts → correlation
Misuse Detection

alert tcp any any -> 192.168.0.0/24 80 \n  (content: "|90 90 90 90|"; msg: "NOP sled");

– Misuse detection searches for direct evidence of attacks
– Models of malice encoded as *signatures*
– Capable of precise matching (low false positives)
– Prone to evasion and ineffective for 0-days
Anomaly Detection

benign any any -> 192.168.0.0/24 80 \
  (content: ascii; msg: ”Non-ASCII request”;
  - Anomaly detection identifies previously unseen behavior
  - Assumes unknown behavior is indicative of an attack
  - Models can be specified or built using machine learning
  - Can detect 0-day attacks
  - Prone to false positives
Training data is synthetic, or is attack-free real behavior
Training data is synthetic, or is attack-free real behavior

– What if the synthetic data is not representative of real behavior? [10]
– What if the real data does contain attacks? How can you be sure it doesn’t? [11]
“Normal behavior” is constant
"Normal behavior" is constant

– What if normal behavior changes over time?
– Concept drift [12]
– If models can be updated, what if the attacker can influence the retraining procedure?
Attacks are distinguishable from normal behavior
Attacks are distinguishable from normal behavior

- What if an attacker can create “normal-looking” attacks?
- Mimicry attacks [13]
def check_malware():
    if not D(check_malware):
        execute_attack()

- Intrusion detection has been reduced to the halting problem [14]
- In practice, false positives are the limiting factor for an IDS
  - Base rate of attacks is low in most environments
  - Even miniscule false positives rates are magnified
### Table 2: Confusion matrix for a binary classifier

<table>
<thead>
<tr>
<th></th>
<th>Event occurred</th>
<th>Event did not occur</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event detected</td>
<td>True positive</td>
<td>False positive</td>
</tr>
<tr>
<td>Event not detected</td>
<td>False negative</td>
<td>True negative</td>
</tr>
</tbody>
</table>
Base Rate Fallacy

Let $A, I$ be two boolean random variables where $I$ is whether an event represents an intrusion and $A$ is whether an alert is raised. Then,

$P(A|I)$ Probability of a true positive

$P(A|\neg I)$ Probability of a false positive

Given this, we can calculate $P(I|A)$...
Base Rate Fallacy

\[ P(I) = 2 \cdot 10^{-5} \]

\[ P(A|I) = \frac{P(A) \cdot P(I|A)}{P(I)} \]

\[ P(I|A) = \frac{P(I) \cdot P(A|I)}{P(I) \cdot P(A|I) + P(\neg I) \cdot P(A|\neg I)} \]

\[ = \frac{2 \cdot 10^{-5} \cdot P(A|I)}{2 \cdot 10^{-5} \cdot P(A|I) + 0.99998 \cdot P(A|\neg I)} \]
Base Rate Fallacy

\[ P(I) = 2 \cdot 10^{-5} \]

\[ P(A|I) = \frac{P(A) \cdot P(I|A)}{P(I)} \]

\[ P(I|A) = \frac{P(I) \cdot P(A|I)}{P(I) \cdot P(A|I) + P(\neg I) \cdot P(A|\neg I)} \]

\[ = \frac{2 \cdot 10^{-5} \cdot P(A|I)}{2 \cdot 10^{-5} \cdot P(A|I) + 0.99998 \cdot P(A|\neg I)} \]

This term dominates \( P(I|A) \)
Consider the case of a medical test that has 99% accuracy ($P(A|I) = 0.99$, $P(\neg A|\neg I) = 0.99$). During consultation, the doctor tells you that you tested positive, but only 1 in 10,000 have the condition ($P(I) = 0.0001$).

What is the probability that you have the condition?
Base Rate Fallacy Example

\[
P(I|A) = \frac{P(I) \cdot P(A|I)}{P(I) \cdot P(A|I) + P(\neg I) \cdot P(A|\neg I)}
\]

\[
= \frac{0.0001 \cdot 0.99}{0.0001 \cdot 0.99 + (1 - 0.0001) \cdot 0.01}
\]

\[
= 0.0098
\]

\[
\approx 1\%
\]
False Positives

- A higher false positive rate has several negative effects aside from increasing inaccuracy
  - Desensitizes users (*user fatigue*)
  - Decreases trust
  - Can have a high cost [15]

It is more important to minimize false positives rather than false negatives
Emergency Alert
BALLISTIC MISSILE THREAT INBOUND TO HAWAII. SEEK IMMEDIATE SHELTER. THIS IS NOT A DRILL.
Comparing Detection Systems

- Receiver operating characteristic (ROC) plots used to compare IDSs
- Plots true positives vs. false positives while varying threshold
- Indicates the expected TPR/FPR trade-off
IDS Evasion [16]

- Overwhelm the IDS
- Overwhelm the operators
- Desynchronization from network, system, and service states
IDS Evasion [16]

- Overwhelm the IDS
- Overwhelm the operators
- Desynchronization from network, system, and service states

How does the IDS know the actual state of monitored hosts and services?
IDS Evasion

Figure 12: Overlapping TCP segments

- What bytestream is received by the victim?
- Depends on packet ordering, victim OS!
Wireless Networking
Wireless Networking [17]

- Link-layer attacks are new again in a wireless environment (Why?)
Wireless Networking [17]

- Link-layer attacks are new again in a wireless environment (Why?)
- Wireless networks reintroduce a shared broadcast medium!
- ARP spoofing is a major problem
  - Possible to MITM many sensitive protocols

Idea: Introduce cryptographic controls
Wired Equivalent Privacy (WEP)

- First standard for wireless network security
  - $\text{RC4}_{K\parallel V} \oplus P = C$ (one-time pad)
  - 40/104-bit key, 24-bit IV
- Quickly broken: Fluhrer-Mantin-Shamir (FMS) attack [18]
  - Exploits reuse of weak IVs
  - Made more efficient with DEAUTH packets and ARP re-injection
Wired Equivalent Privacy (WEP)

- First standard for wireless network security
  - \( \text{RC}4_{K\|V} \oplus P = C \) (one-time pad)
  - 40/104-bit key, 24-bit IV
- Quickly broken: Fluhrer-Mantin-Shamir (FMS) attack [18]
  - Exploits reuse of weak IVs
  - Made more efficient with DEAUTH packets and ARP re-injection

What sorts of threats does WEP et al. defend against?
Wi-Fi Protected Access (WPA)

- Standard introduced in 2003 to replace WEP
- Adopted the Temporal Key Integrity Protocol (TKIP)
  - Per-packet 128-bit keys to prevent WEP-like vulnerabilities
- Introduced cryptographically-strong integrity checks instead of weak CRCs
- WPA2 added CCMP, an AES-based encryption mode
- WPA3 adds additional cryptographic protocols
  e.g., AES256-GCM-HMAC-SHA384
WPA Modes of Operation

– WPA Personal
  – 256-bit pre-shared key derived using PBKDF2

– WPA Enterprise
  – Builds on 802.1X authentication protocol
  – Requires additional infrastructure
    e.g., RADIUS server, PKI

– Wi-Fi Protected Setup (WPS)
802.1X Authentication

- Standard for port-based network access control (PNAC)
- Three participants: supplicant, authenticator, and authentication server
  - Supplicant port starts in *uncontrolled state*
  - Supplicant exchanges EAPOL frames with the authenticator
  - Authenticator proxies messages to authentication server
  - If successful, supplicant port transitions to *controlled state*
802.1X Authentication

Figure 13: 802.1X authentication protocol
Access Point Spoofing [19]

- How do you verify that you’re connecting to the network you think you are?
- How are SSIDs allocated?
- How are SSIDs bound to an organization?
DEAUTH Attacks

- 802.11 standard [17] supports a deauthentication frame, a “sanctioned technique to inform a rogue station that they have been disconnected from the network”
- Anybody can send DEAUTH frames!
- Useful for forcing clients to connect to a rogue access point
- Also useful for gathering packets useful for WPA password cracking
Conclusions
Conclusions

In this module, we covered:

1. The TCP/IP model
2. Link-layer security
3. Internet-layer security
4. Transport-layer security
5. Wireless security
References


