# Computer Security

10r. Network Security – continued
DNS, VPNs, TLS

Paul Krzyzanowski • David Domingo • Ananya Jana
Rutgers University
Spring 2019



# Domain Name System (DNS) Vulnerabilities



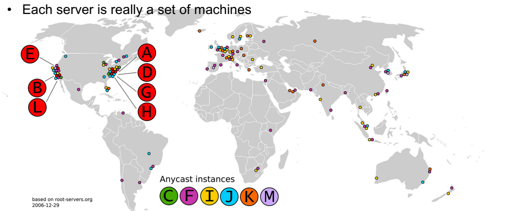
## Domain Name System

- Hierarchical service to map domain names to IP addresses
- How do you find the DNS Server for **rutgers.edu**?
  - That's what the domain registry keeps track of
  - When you register a domain
    - You supply the addresses of at least two **DNS servers** that can answer queries for your zone
    - You give this info to the **domain registrar** (e.g., Namecheap, GoDaddy) who updates the database at the **domain registry** (e.g., Verisign for .com, .net, .edu, .gov, … domains)
      - **Domain registrar:** Sells domain names to the public
      - **Domain regsirty:** Maintains the top-level domain database
- *So how do you find the right DNS server?*
  - Start at the root



## Root name servers

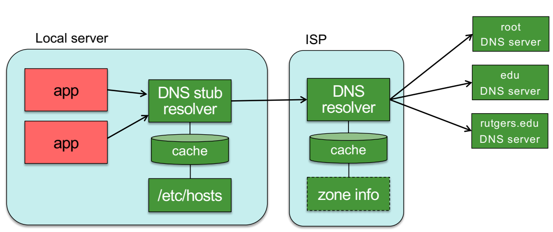
- The **root name servers** provide lists of authoritative name servers for top-level domains
- 13 root name servers
  - `A.ROOT-SERVERS.NET`, `B.ROOT-SERVERS.NET`, …
  - Each has redundancy (via *anycast* routing or load balancing)
    - Each server is really a set of machines




Download the latest list at http://www.internic.net/domain/named.root



## DNS Resolvers in action




Local stub resolver:
- check local cache
- check local hosts file
- send request to external resolver

External resolver:
- Running at ISP, Cloudflare, Google Public DNS, OpenDNS, etc.

E.g., on Linux: resolver is configured via the `/etc/resolv.conf` file



## **Pharming** attack

- Redirect traffic to an attacker's site by modifying how the DNS resolver gets its information
- Forms of attack
  1. Use malware or social engineering to modify a computer's *hosts* file
     This file maps *names→IP addresses* and avoids DNS queries
  2. Attack the router & modify its DNS server setting
     Direct traffic to the attacker's DNS server, which will give the wrong IP address for certain domain names

## DNS Vulnerabilities

- **Programs (and users) trust the host-address mapping**
  - This is the basis for some security policies
    - Browser same-origin policy, URL address bar
- But DNS responses can be faked
  - If an attacker gives a DNS response first, the host will use that
  - Malicious responses can direct messages to different hosts
  - A receiver cannot detect a forged response
- DNS resolvers cache their results (with an expiration)
  - If it gets a forged response, the forged results will be passed on to any systems that query it
  - **Cache-poisoning attack**



## **DNS spoofing** attack

Redirect traffic to an attacker via DNS **cache poisoning**

- An attacker sends the wrong DNS response
  - The DNS resolver requesting it will cache it and provide that to anyone else who asks in the near future
- How does we prevent spoofed responses?
  - Each DNS query contains a 16-bit Query ID (**QID**) – only 65,536 to guess
    - Response from the DNS server must have a matching QID
  - DNS uses UDP and this was created to make it easy for a system to match responses with requests
- An attacker will have to guess the QID number
  - But numbers were sequential and not hard to guess
  - Fix by using random Query IDs



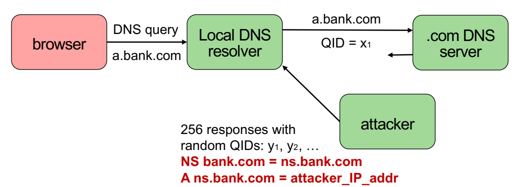
## DNS spoofing via Cache Poisoning

- What happens?
  - Malicious JavaScript on a web page cuases the client to try to look up a.bank.com, b.bank.com, etc.
  - At the same time, the attacker is sending a stream of DNS “responses” hoping that one will have a matching QID
- If the attacker is successful, one of the responses matches up?
  - But we expect the victim to go to **bank.com**, not **f.bank.com**
  - However….
    The DNS response can also define a new DNS server for bank.com!
  - This overwrites any saved DNS info for bank.com that may be cached



## DNS spoofing via Cache Poisoning

JavaScript on a website may launch a DNS attacker




If there is some $j$ such that $x_1 = y_j$ then the response will be cached
All future DNS queries for anything at bank.com will go to attacker_IP_addr
If it doesn’t work … try again with **b.bank.com**, **c.bank.com**, etc.



## Defenses against DNS cache poisoning

- Query IDs used to be predictable
  - Easy to guess
  - Have a web page make a DNS query to a domain under the attacker’s control & look at the QID
  - The attacker can then guess the next one
- Randomize source port # – *where DNS queries originate*
  - Attack will take several hours instead of a few minutes
  - Will have to send responses to a range of ports
  - But this is tricky in real environments that use NAT (network address translation) and may limit the exposed UDP ports
- Issue double DNS queries
  - Attacker will have to guess the Query ID twice (32 bits)



## Defenses against DNS cache poisoning

- Use TCP instead of UDP
  - It’s much harder to inject a response into a TCP stream
  - But
    - Much higher latency
    - Much more overhead at the DNS resolver
- The better long-term solution: DNSSEC
  - Secure extension to DNS that provide authenticated requests & responses
  - Responses contain a digital signature
  - But
    - Adoption has been very slow
    - DNSSEC response size is much bigger than a DNS response, which makes it more powerful for DoS attacks



## DNS Rebinding



## DNS Rebinding

Attack that allows attackers to run a script to attack other systems on the victim's private network

- What is the **same-origin policy**?
  - The core web application security model
  - Client web browser scripts can access data from other web pages ***only*** if they have the same **origin**
  - Origin = same { protocol, host name, port number }
- The policy relies on **comparing domain names**
- If we can change the underlying address:
  - We can send messages to an attacker's system while the software thinks it's still going to the same domain
  - This can let us access private machines in the user's local area network
  - Example: access local web services, cameras, thermostats, printers, …



## DNS Rebinding

- **Attacker**
  - Registers a domain (attacker.com)
  - Sets up a DNS server
  - DNS server responds with very short TTL values – response won't be cached
- **Client (browser)**
  - Script on page causes access to a malicious domain
  - Attacker's DNS server responds with IP address of a server hosting malicious client-side code
  - Malicious client-side code makes additional references to the domain
    - Permitted under **same-origin policy**
      - A browser permits scripts in one page to access data in another only if both pages have the same origin & protocol
    - The script causes the browser to issue a new DNS request
    - Attacker replies with a new IP address (e.g., a target somewhere outside the domain)
    - The script can continue to access content at the same domain
      - But it really isn't in the domain!



## Defending against DNS rebinding

- Force **minimum TTL values**
  - This may affect some legitimate dynamic DNS services
- **DNS pinning**: refuse to switch the IP address for a domain name
  - This is similar to forcing minimum TTL values
- Have the local DNS resolver make sure DNS responses don't contain private IP addresses
- Server-side defense within the local area network
  - Reject HTTP requests with unrecognized Host headers
  - Authenticate users



## Network Layer Conversation Isolation: Virtual Private Networks (VPNs)



## Fundamental Layer 2 & 3 Problems

- IP relies on store-and-forward networking
  - Network data passes through untrusted hosts
  - Routes may be altered to pass data through malicious hosts
- Packets can be sniffed (and new forged packets injected)
- Ethernet, IP, TCP & UDP
  - All designed with no authentication or integrity mechanisms
  - No source authentication on IP packets
  - TCP session state can be examined or guessed … … and TCP sessions can be hijacked
- ARP, DHCP, DNS protocols
  - Can be spoofed to redirect traffic to malicious hosts
- Internet route advertisement protocols are not secure
  - Can redirect traffic to malicious routers/hosts

## Solution: Use private networks

Connect multiple geographically-separated private subnetworks together

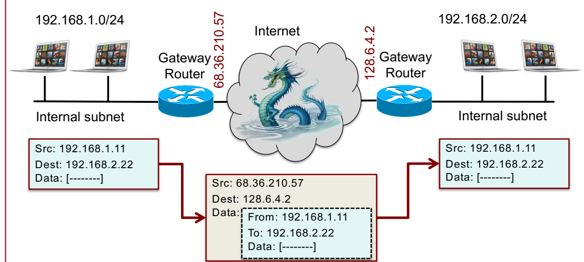

But this is expensive … and not feasible in many cases (e.g., cloud servers)



## What's a tunnel?

Tunnel = Packet encapsulation

Treat an entire IP datagram as payload on the public network






## Tunnel mode vs. transport mode

- **Tunnel mode**
  - Communication between gateways: *network-to-network*
  - Or *host-to-network*
  - Entire datagram is encapsulated
- **Transport mode**
  - Communication between hosts
  - IP header is not modified



## Virtual Private Networks

Take the concept of tunneling

… and safeguard the encapsulated data

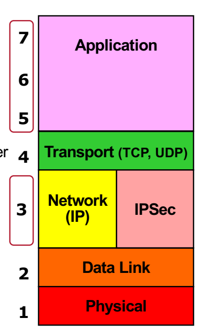
- **Add a MAC**
  - Ensure that outsiders don't modify the data
- **Encrypt the contents**
  - Ensure that outsiders can't read the data



## IPsec

Internet Protocol Security

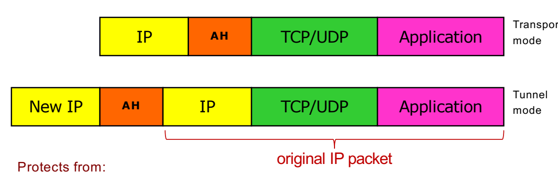
- End-to-end solution at the IP layer
- Two protocols:
  - **IP Authentication Header** Protocol (AH)
    - Authentication & integrity of payload and header
    - ***Provides integrity***
  - **Encapsulating Security Payload** (ESP)
    - AH + Confidentiality of payload
    - ***Adds content encryption***






## IPsec Authentication Header (AH)

**Guarantees integrity & authenticity of IP packets**
- MAC for the contents of the entire IP packet
- Over unchangeable IP datagram fields (e.g., not TTL or fragmentation fields)



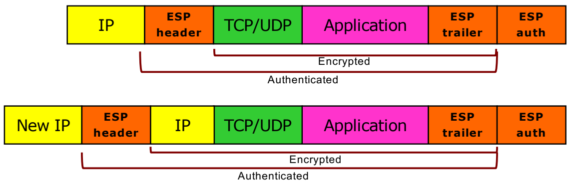

Protects from:
- Tampering
- Forging addresses
- Replay attacks (signed sequence number in AH)

Layered directly on top of IP (protocol 51) - not UDP or TCP

## IPsec Encapsulating Security Payload (ESP)

**Encrypts entire payload**

– Plus authentication of payload + IP header (everything AH does) (may be optionally disabled – but you don't want to)




Directly on top of IP (protocol 51) - not UDP or TCP



## IPSec algorithms

- **Authentication**
  – Certificates, or pre-shared key authentication
- **Key exchange**
  – Diffie-Hellman to exchange keying material for key generation
  – Key lifetimes determine when new keys are regenerated
- **Confidentiality**
  – 3DES-CBC
  – AES-CBC
- **Integrity protection & authenticity**
  – HMAC-SHA1
  – HMAC-SHA2



## Transport Layer Conversation Isolation: Transport Layer Security (TLS)



## Transport Layer Security

- Goal: provide a *transport layer* security protocol
- After setup, applications feel like they are using TCP sockets

SSL: Secure Socket Layer

- Created with HTTP in mind
  – Web sessions should be secure
  – Mutual authentication is usually not needed
    - Client needs to identify the server but the server won't know all clients
    - Rely on password authentication after the secure channel is set up

## TLS vs. SSL – versions

SSL evolved to TLS (Transport Layer Security)

SSL 3.0 was the last version of SSL
… and is considered insecure

We now use TLS (but is often still called SSL)

– TLS 1.0 = SSL 3.1, TLS 1.1 = SSL 3.2, TLS 1.2 = SSL 3.3
– Latest version = TLS 1.3 = SSL 3.4

- Retired versions
  – TLS 1.0/SSL 3 are not considered strong anymore and their use is not recommended
  – As of 2019, Google Chrome deprecates support for TLS 1.1

## TLS Protocol

Goal:

**Provide authentication (usually one-way), privacy, & data integrity between two applications**

- Principles
  – **Data encryption**
    - Use symmetric cryptography to encrypt data
    - **Key exchange**: keys generated uniquely at the start of each session
  – **Data integrity**
    - Include a MAC with transmitted data to ensure message integrity
  – **Authentication**
    - Use public key cryptography & X.509 certificates for authentication
    - Optional – can authenticate 0, 1, or both parties
  – **Interoperability & evolution**
    - Support many different key exchange, encryption, integrity, & authentication protocols – negotiate what to use at the start of a session



## TLS Protocol & Ciphers

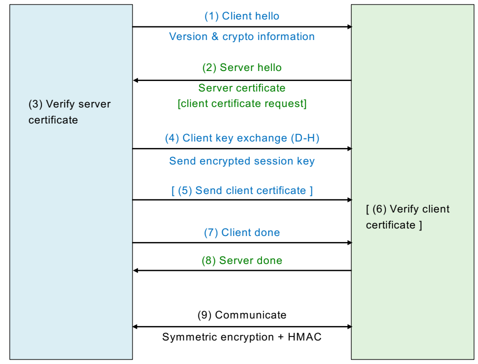
Two sub-protocols

1. Authenticate & establish keys
2. Communicate
   - HMAC used for message authentication

- Authentication
  – Public keys (X.509 certificates and – usually – RSA cryptography)
- Key exchange options
  – Ephemeral Diffie-Hellman keys (generated for each session)
  – Pre-shared key
- Data encryption options
  – AES GCM, AES CBC, ARIA (GCM/CBC), ChaCha20-Poly1305, …
- Data integrity options
  – HMAC-MD5, HMAC-SHA1, HMAC-SHA256/384, …



## TLS Protocol






## Benefits of TLS

- Benefits
  – Protects integrity of communications
  – Protects the privacy of communications
  – Validates the authenticity of the server (if you trust the CA)



## Some attacks on TLS

- Man-in-the-middle: BEAST attack in TLS 1.0
  – Attacker was able to see Initialization Vector (IV) for CBC and deduce plaintext (because of known HTML headers & cookies)
  – An IV doesn't have to be secret – but it turned out this wasn't a good idea
  – Attacker was able to send chosen plaintext & get it encrypted with a known IV
  – Fixed by using fresh IVs for each new 16K block
- Man-in-the-middle: crypto renegotiation
  – Attacker can renegotiate the handshake protocol during the session to disable encryption
  – Proposed fix: have client & server verify info about previous handshakes
- THC-SSL-DoS attack
  – Attacker initiates a TLS handshake & requests a renegotiation of the encryption key – repeat over & over, using up server resources



## Other problems with TLS

- Client authentication Problem
  – Client authentication is almost never used
    - Generating keys & obtaining certificates is not an easy process for users
    - Any site can request the certificate
      – User will be unaware their anonymity is lost
    - Moving private keys around can be difficult
      – What about public computers?
  – We usually rely on other authentication mechanisms
    - Usually user name and password
    - But no danger of eavesdropping since the session is encrypted
    - May use one-time passwords or two-factor authentication if worried about eavesdroppers at physical premises



The end

