

## Network Security (TCP/IP and DNS)

Tadayoshi Kohno

Some slides based on Dan Boneh's and Vitaly Shmatikov's

## Programming Project #2

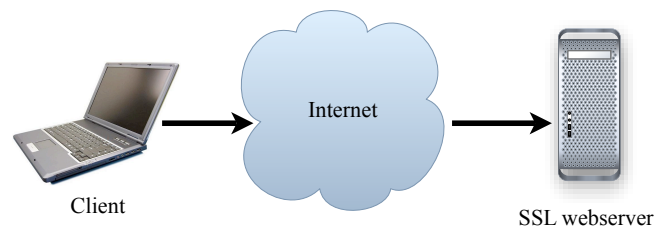
- ◆ Out today, Tuesday, May 8
- ◆ Due Thursday, May 24, 11:59pm
  - Submit via Catalyst system
- ◆ Teams of up to three people
  - New teams OK (old teams also OK)
- ◆ Basic idea: Implement a "Man-in-the-Middle" attack against SSL
- ◆ Recall Security and Privacy Code of Ethics form
- ◆ Based on Dan Boneh's CS255 project (Stanford)
  - Slides: [http://crypto.stanford.edu/~dabo/cs255/proj2\\_pres.pdf](http://crypto.stanford.edu/~dabo/cs255/proj2_pres.pdf)

## Overview

- ◆ MITM attack against SSL
  - Not at network layer (not re-writing packets, etc)
  - At SSL Proxy Layer, in Java
    - Networking
    - SSL
    - Certificates
- ◆ Password-based authentication for MITM server
  - Hashed, salted passwords
  - Password file encrypted with an authenticated encryption scheme.

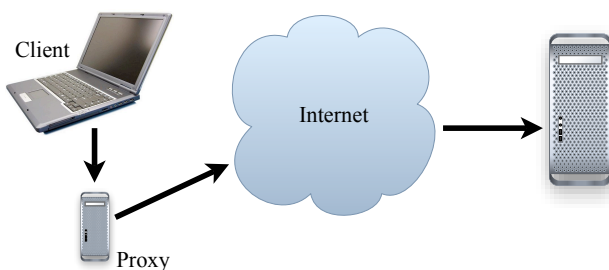
## Overview

- ◆ Normal SSL
  - SSL encrypted data routed like normal TCP/IP over Internet



## Proxy Server

- ◆ Browser connects to proxy
- ◆ Proxy connects to web server and forwards between the two



## "Man in the Middle"

- ◆ Instead of forwarding encrypted data between the two hosts, the proxy will set up two different SSL connections
  - Proxy <--> Remote Server
    - Normal SSL client connection to remote site
  - Proxy <--> Browser
    - SSL server connection to the browser, using its own certificate, with some data cloned from the remote hosts' certificate
    - If browser accepts this fake certificate, the proxy has access to the data in the clear!

## What we provided

- ◆ Basic Proxy Server setup
  - Parses CONNECT request and sets up a connection between client and remote server
- ◆ Basic Admin Server/Client
  - Server listens for connections on plain socket and parses out username/password/command that client sends

## Basic Admin Server/Client

- ◆ Goal: Experience in adding security features to an application
  - Secure connection between admin client and proxy server using SSL
  - Password based authentication for client
    - Secure storage of password file (authenticated encryption)
    - Passwords stored, hashed, using public and private salt
  - Extra credit: Challenge / Response authentication
    - In addition to password authentication, not instead of.

## Proxy Server

- ◆ Already listens for browser CONNECT requests and sets up the needed SSL connections
- ◆ You should
  - Understand the connections being made
  - Obtain the remote server certificate from the remote SSL connection
  - Copy the relevant fields and sign a forged certificate using your CA cert (from your keystore); use IAIK
  - Modify the code creating the client SSL connection to use the newly forged certificate

## Signing Certificate

- ◆ Build a self-signed certificate for the proxy server (the proxy server's "CA" certificate)
  - `keytool -genkey -keyalg RSA`
  - Store this in a JKS keystore for use by your proxy server
  - Use it for signing your programmatically generated certs
  - Your proxy pretends to be the CA
- ◆ Submit a keystore with your project

## Generating Certs "On the Fly"

- ◆ Not easy to generate certificates programmatically using standard Java libraries
- ◆ Instead, use the IAIK-JCE library
  - `iaik.x509.X509Certificate` (class)

## iaik.x509.X509Certificate

- ◆ To convert from a java certificate:
  - `new X509Certificate(javaCert.getEncoded());`
- ◆ Signing
  - `cert.sign(AlgorithmID.sha256withRSAEncryption, issuerPk);`
- ◆ See `iaik.asn1.structures.Name`
  - For extracting info (e.g., common name) from the certificate's distinguished name (`cert.getSubjectDN()`)

## Managing Certs and SSL Sockets

- ◆ Use the KeyStore class for
  - Loading certificates from file (e.g., your CA certificate)
  - Storing programmatically generated certificates
- ◆ Use SSLContext class for setting up certificates to be used with SSLServerSocket
  - Create a certificate
  - Load into new KeyStore
  - Init a KeyManagerFactory with new KeyStore
  - Init SSLContext with new KeyManagerFactory and provided "TrustEveryone" TrustManager
- ◆ Use SSLContext for creating SSLSocketFactories
- ◆ See MITMSSLSocketFactory.java

## Admin Server

- ◆ Already listens for client connections and parses the data sent using plain sockets
- ◆ You should
  - Modify code to use SSL sockets (see the proxy server code for examples)
  - Implement authentication for the transmitted username and password
  - Implement required admin commands
    - Shutdown
    - Stats

## Password file

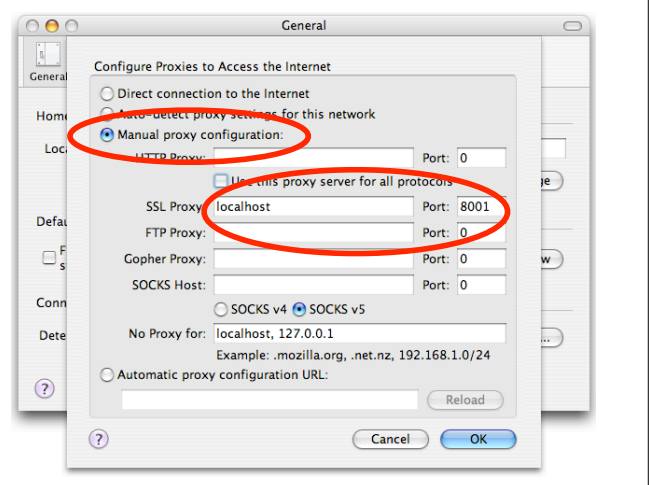
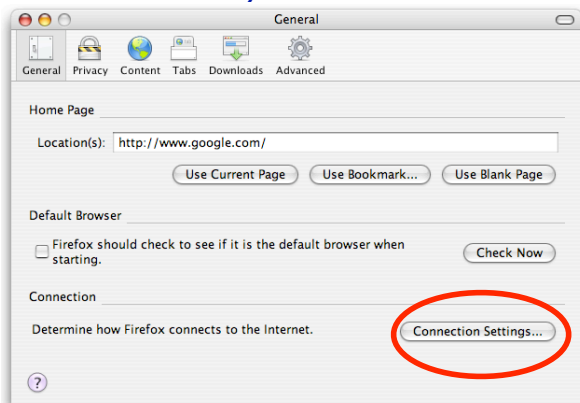
- ◆ Need to store a file containing usernames, salts, and hashed passwords
  - Both public and secret salts (aka pepper)
- ◆ Should be stored encrypted with an authenticated encryption scheme
  - I recommend Encrypt-then-MAC
  - Maybe AES in CTR mode to Encrypt, and HMAC-SHA1 to MAC
  - But be careful about security!!

Username	Salt	Hashed password
Alice	S	H(Pwd  S  P)
Bob	...	...

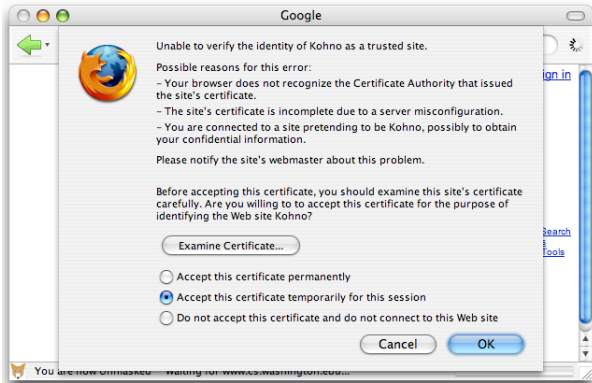
## Password File Utility

- ◆ You should add a utility for creating these password files
- ◆ Simple method:
  - Make a class to take a file and a list of usernames and passwords, and convert it to a password file.

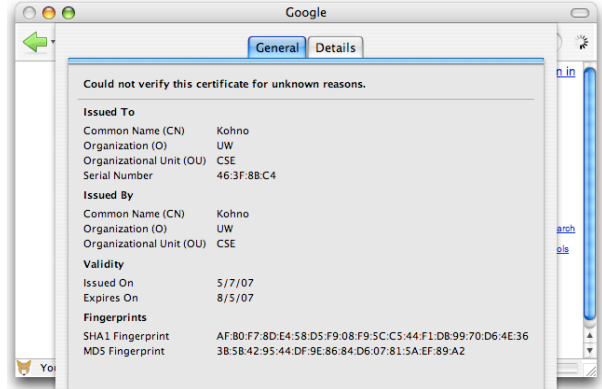
## Configuring Firefox (under OS X, similar for Linux)



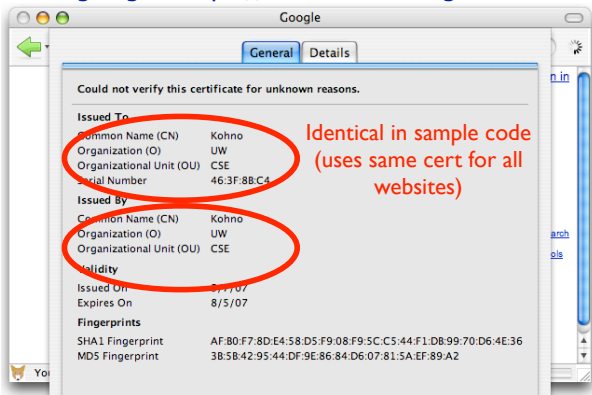
### When going to <https://www.cs.washington.edu>



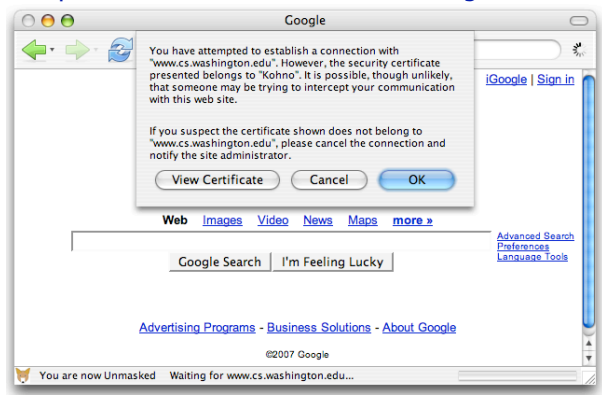
### When going to <https://www.cs.washington.edu>



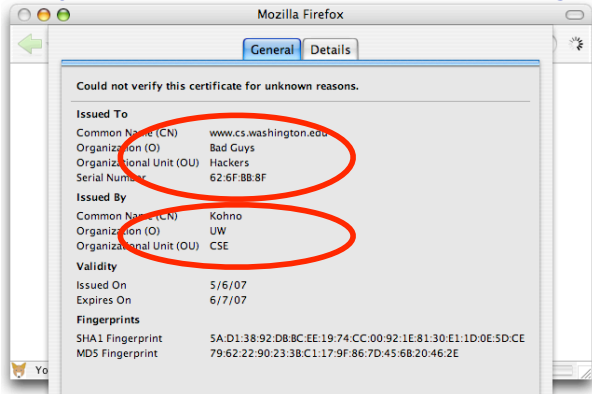
### When going to <https://www.cs.washington.edu>



### Sample code causes this second warning



### Your job - new certificates, avoid second warning

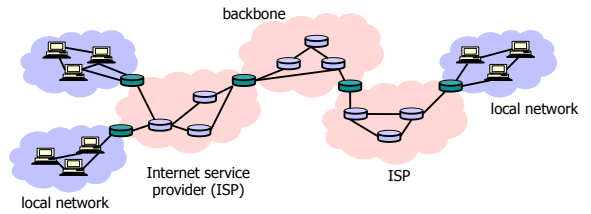


### Possible Problems

- ◆ You should be able to start up the proxy and connect to it "out of the box"
  - After you create your keystore with "keytool"
- ◆ If you are having problems
  - Is someone else trying to use your machine and that port? (Default 8001.)
    - Try a different port on the command line
  - Firewall problems
    - Try to telnet to the needed ports (8001/8002/...)
  - Try running your browser on the same machine, and setting the proxy as "localhost"
- ◆ Course mailing list: Great place to share knowledge

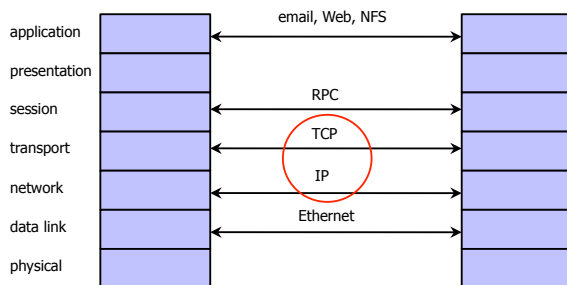
## Attacks on TCP/IP and DNS

## Internet Infrastructure

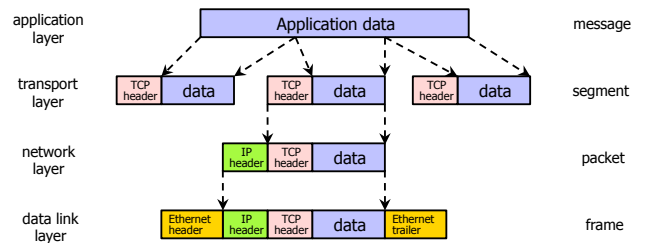


- ◆ TCP/IP for packet routing and connections
- ◆ Border Gateway Protocol (BGP) for route discovery
- ◆ Domain Name System (DNS) for IP address discovery

## OSI Protocol Stack

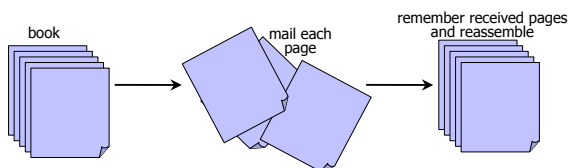


## Data Formats



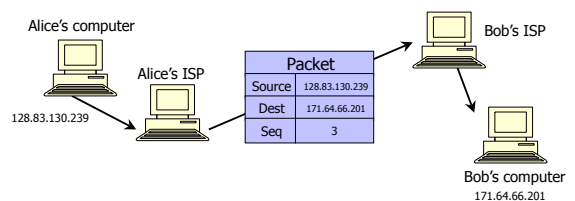
## TCP (Transmission Control Protocol)

- ◆ Sender: break data into packets
  - Sequence number is attached to every packet
- ◆ Receiver: reassemble packets in correct order
  - Acknowledge receipt; lost packets are re-sent
- ◆ Connection state maintained on both sides



## IP (Internet Protocol)

- ◆ Connectionless
  - Unreliable, "best-effort" protocol
- ◆ Uses numeric addresses for routing
  - Typically several hops in the route



## IP Routing

- ◆ Routing of IP packets is based on IP addresses
- ◆ Routers use a forwarding table
  - Entry = destination, next hop, network interface, metric
  - For each packet, a table look-up is performed to determine how to route it
- ◆ Routing information exchange allows update of old routes and creation of new ones
  - RIP (Routing Information Protocol)
  - OSPF (Open Shortest Path First Protocol)
  - BGP (Border Gateway Protocol)

## Routing Attacks

- ◆ Source routing
  - Source of the packet specifies a particular route
    - For example, because the automatic route is dead
  - Attacker can spoof source IP address and use source routing to direct response through a compromised host
  - Solution: reject packets with source routing!
    - More heavy-duty: allow source route only via trusted gateways
- ◆ Routing Information Protocol (RIP)
  - Use bogus routing updates to intercept traffic
    - RIP implicitly assumes that routers are trusted
  - “Black hole” attacks and many others

## BGP Misconfiguration

- ◆ Domain advertises good routes to addresses it does not know how to reach
  - Result: packets go into a network “black hole”
- ◆ April 25, 1997: “The day the Internet died”
  - AS7007 (Florida Internet Exchange) de-aggregated the BGP route table and re-advertised all prefixes as if it originated paths to them
  - In effect, AS7007 was advertising that it has the best route to every host on the Internet
  - Huge network instability as incorrect routing data propagated and routers crashed under traffic

## ICMP (Control Message Protocol)

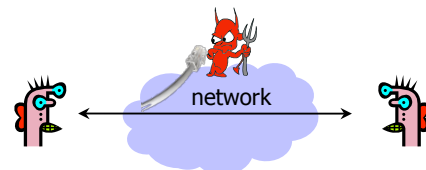
- ◆ Provides feedback about network operation
  - “Out-of-band” messages carried in IP packets
  - Error reporting, congestion control, reachability, etc.
- ◆ Example messages:
  - Destination unreachable
  - Time exceeded
  - Parameter problem
  - Redirect to better gateway
  - Reachability test (echo / echo reply)
  - Message transit delay (timestamp request / reply)

## Security Issues in TCP/IP

- ◆ Network packets pass by untrusted hosts
  - Eavesdropping (packet sniffing)
- ◆ IP addresses are public
  - Smurf attacks
- ◆ TCP connection requires state
  - SYN flooding
- ◆ TCP state is easy to guess
  - TCP spoofing and connection hijacking

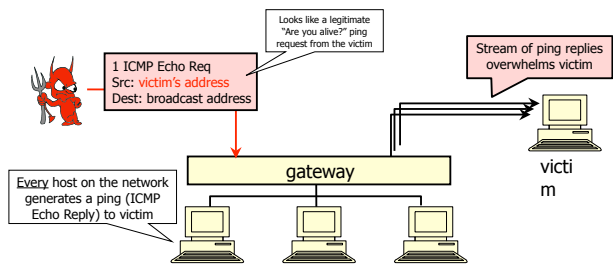
## Packet Sniffing

- ◆ Many applications send data unencrypted
  - ftp, telnet send passwords in the clear
- ◆ Network interface card (NIC) in “promiscuous mode” reads all passing data



Solution: encryption (e.g., IPSec), improved routing

## Smurf Attack



Solution: reject external packets to broadcast addresses

## "Ping of Death"

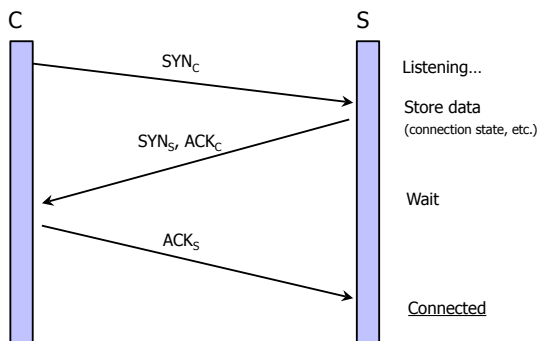
- ◆ If an old Windows machine received an ICMP packet with a payload longer than 64K, machine would crash or reboot

- Programming error in older versions of Windows
- Packets of this length are illegal, so programmers of Windows code did not account for them

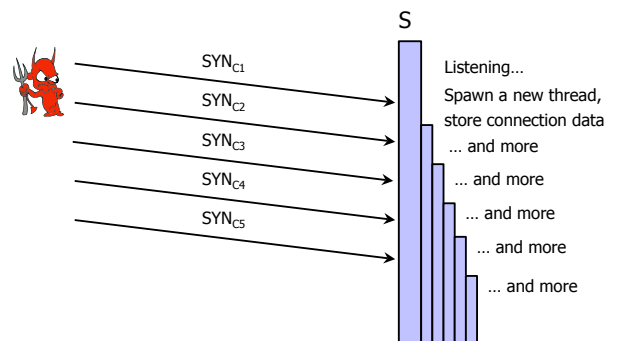
- ◆ Recall "security theme" of this course - every line of code might be the target of an adversary

Solution: patch OS, filter out ICMP packets

## TCP Handshake



## SYN Flooding Attack



## SYN Flooding Explained

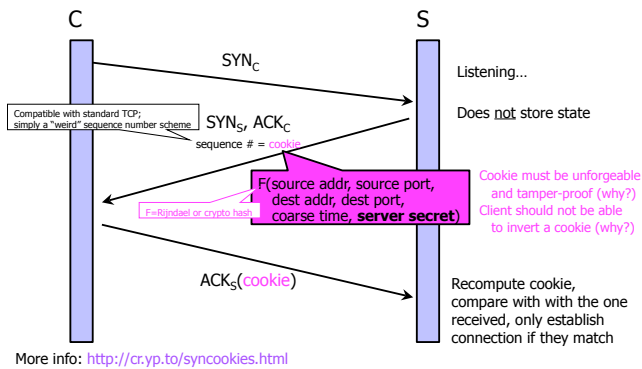
- ◆ Attacker sends many connection requests with spoofed source addresses
- ◆ Victim allocates resources for each request
  - Connection state maintained until timeout
  - Fixed bound on half-open connections
- ◆ Once resources exhausted, requests from legitimate clients are denied
- ◆ This is a classic **denial of service (DoS)** attack
  - Common pattern: it costs nothing to TCP initiator to send a connection request, but TCP responder must allocate state for each request (**asymmetry!**)

## Preventing Denial of Service

- ◆ DoS is caused by asymmetric state allocation
  - If responder opens a state for each connection attempt, attacker can initiate thousands of connections from bogus or forged IP addresses
- ◆ **Cookies** ensure that the responder is stateless until initiator produced at least 2 messages
  - Responder's state (IP addresses and ports of the connection) is stored in a cookie and sent to initiator
  - After initiator responds, cookie is regenerated and compared with the cookie returned by the initiator

## SYN Cookies

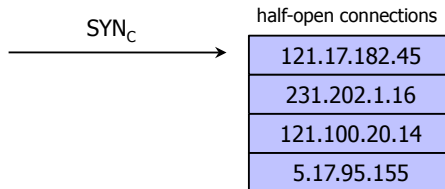
[Bernstein and Schenk]



## Anti-Spoofing Cookies: Basic Pattern

- ◆ Client sends request (message #1) to server
- ◆ Typical protocol:
  - Server sets up connection, responds with message #2
  - Client may complete session or not (potential DoS)
- ◆ Cookie version:
  - Server responds with hashed connection data instead of message #2
  - Client confirms by returning hashed data
    - If source IP address is bogus, attacker can't confirm
  - Need an extra step to send postponed message #2, except in TCP (SYN-ACK already there)

## Another Defense: Random Deletion

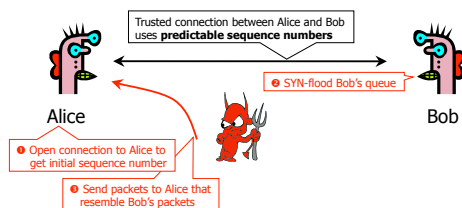


- ◆ If SYN queue is full, delete random entry
  - Legitimate connections have a chance to complete
  - Fake addresses will be eventually deleted
- ◆ Easy to implement

## TCP Connection Spoofing

- ◆ Each TCP connection has an associated state
  - Sequence number, port number
- ◆ TCP state is easy to guess
  - Port numbers are standard, sequence numbers are often predictable
  - Can inject packets into existing connections
- ◆ If attacker knows initial sequence number and amount of traffic, can guess likely current number
  - Send a flood of packets with likely sequence numbers

## "Blind" IP Spoofing Attack



- ◆ Can't receive packets sent to Bob, but maybe can penetrate Alice's computer if Alice uses **IP address-based authentication**
  - For example, rlogin and many other remote access programs uses address-based authentication

## DoS by Connection Reset

- ◆ If attacker can guess current sequence number for an existing connection, can send Reset packet to close it
  - With 32-bit sequence numbers, probability of guessing correctly is  $1/2^{32}$  (not practical)
  - Most systems accept large windows of sequence numbers  $\Rightarrow$  much higher probability of success
    - Need large windows to handle massive packet losses



## User Datagram Protocol (UDP)

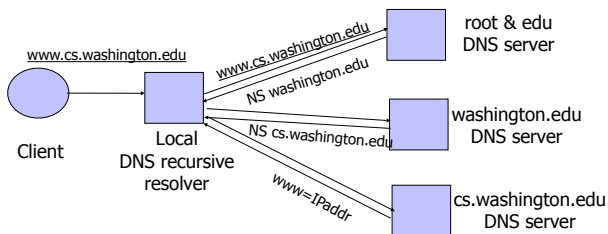
- ◆ UDP is a connectionless protocol
  - Simply send datagram to application process at the specified port of the IP address
  - Source port number provides return address
  - Applications: media streaming, broadcast
- ◆ No acknowledgement, no flow control, no message continuation
- ◆ Denial of service by **UDP data flood**

## Countermeasures

- ◆ Above transport layer: Kerberos
  - Provides authentication, protects against spoofing
  - Does not protect against connection hijacking
- ◆ Above network layer: SSL/TLS and SSH
  - Protects against connection hijacking and injected data
  - Does not protect against DoS by spoofed packets
- ◆ Network (IP) layer: IPSec
  - Protects against hijacking, injection, DoS using connection resets, IP address spoofing

## DNS: Domain Name Service

DNS maps symbolic names to numeric IP addresses  
(for example, [www.cs.washington.edu](http://www.cs.washington.edu) ↔ 128.208.3.88)



## DNS Caching

- ◆ DNS responses are cached
  - Quick response for repeated translations
  - Other queries may reuse some parts of lookup
    - NS records for domains
- ◆ DNS negative queries are cached
  - Don't have to repeat past mistakes
    - For example, misspellings
- ◆ Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record

## DNS Vulnerabilities

- ◆ DNS host-address mappings are not authenticated
- ◆ DNS implementations have vulnerabilities
  - Reverse query buffer overrun in old releases of BIND
    - Gain root access, abort DNS service...
  - MS DNS for NT 4.0 crashes on chargen stream
    - telnet ntbox 19 | telnet ntbox 53
- ◆ Denial of service is a risk
  - Oct '02: ICMP flood took out 9 root servers for 1 hour

## Reverse DNS Spoofing

- ◆ Trusted access is often based on host names
  - E.g., permit all hosts in `.rhosts` to run remote shell
- ◆ Network requests such as rsh or rlogin arrive from numeric source addresses
  - System performs reverse DNS lookup to determine requester's host name and checks if it's in `.rhosts`
- ◆ If attacker can spoof the answer to reverse DNS query, he can fool target machine into thinking that request comes from an authorized host
  - No authentication for DNS responses and typically no double-checking (numeric → symbolic → numeric)

## Other DNS Risks

- ◆ DNS cache poisoning
  - False IP with a high time-to-live will stay in the cache of the DNS server for a long time
  - Basis of pharming
- ◆ Spoofed ICANN registration and domain hijacking
  - Authentication of domain transfers based on email address
  - Aug '04: teenager hijacks eBay's German site
  - Jan '05: hijacking of panix.com (oldest ISP in NYC)
    - "The ownership of panix.com was moved to a company in Australia, the actual DNS records were moved to a company in the United Kingdom, and Panix.com's mail has been redirected to yet another company in Canada."
- ◆ Misconfiguration and human error

## JavaScript/DNS Intranet attack (I)

## JavaScript/DNS Intranet attack (I)

- ◆ Consider a Web server intra.good.net

## JavaScript/DNS Intranet attack (I)

- ◆ Consider a Web server intra.good.net
  - IP: 10.0.0.7, inaccessible outside good.net network

## JavaScript/DNS Intranet attack (I)

- ◆ Consider a Web server intra.good.net
  - IP: 10.0.0.7, inaccessible outside good.net network
  - Hosts sensitive CGI applications

## JavaScript/DNS Intranet attack (I)

- ◆ Consider a Web server intra.good.net
  - IP: 10.0.0.7, inaccessible outside good.net network
  - Hosts sensitive CGI applications
- ◆ Attacker at evil.org gets good.net user to browse www.evil.org

## JavaScript/DNS Intranet attack (I)

- ◆ Consider a Web server intra.good.net
  - IP: 10.0.0.7, inaccessible outside good.net network
  - Hosts sensitive CGI applications
- ◆ Attacker at evil.org gets good.net user to browse www.evil.org
- ◆ Places Javascript on www.evil.org that accesses sensitive application on intra.good.net

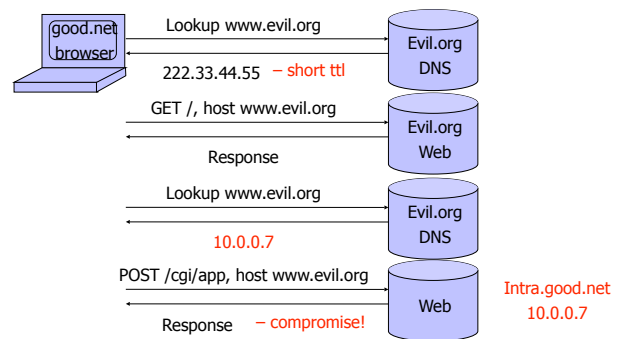
## JavaScript/DNS Intranet attack (I)

- ◆ Consider a Web server intra.good.net
  - IP: 10.0.0.7, inaccessible outside good.net network
  - Hosts sensitive CGI applications
- ◆ Attacker at evil.org gets good.net user to browse www.evil.org
- ◆ Places Javascript on www.evil.org that accesses sensitive application on intra.good.net
  - This doesn't work because Javascript is subject to "same-origin" policy

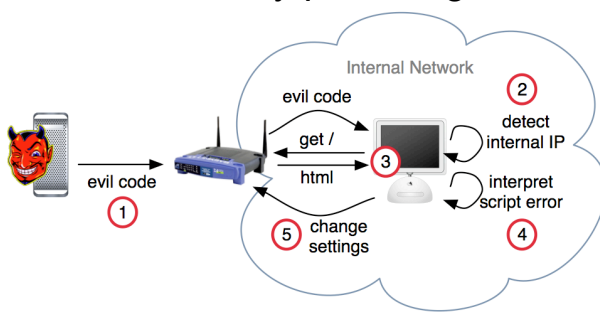
## JavaScript/DNS Intranet attack (I)

- ◆ Consider a Web server intra.good.net
  - IP: 10.0.0.7, inaccessible outside good.net network
  - Hosts sensitive CGI applications
- ◆ Attacker at evil.org gets good.net user to browse www.evil.org
- ◆ Places Javascript on www.evil.org that accesses sensitive application on intra.good.net
  - This doesn't work because Javascript is subject to "same-origin" policy
  - ... but the attacker controls evil.org DNS

## JavaScript/DNS Intranet attack (II)

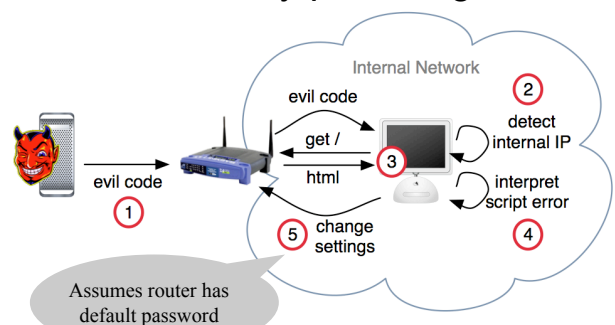


## Drive-by pharming

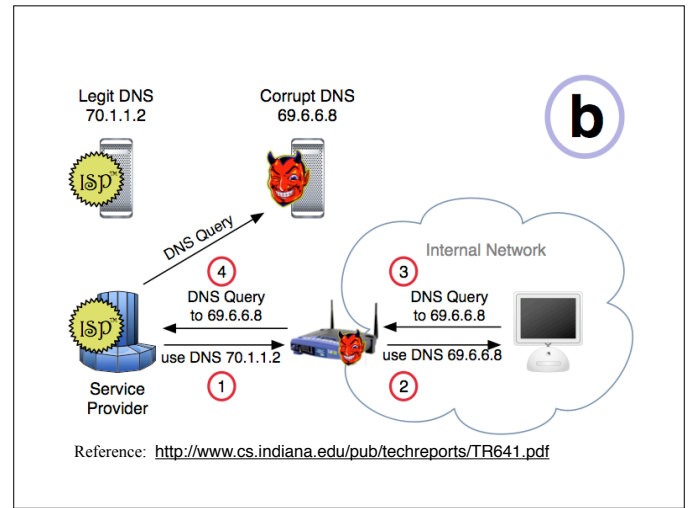
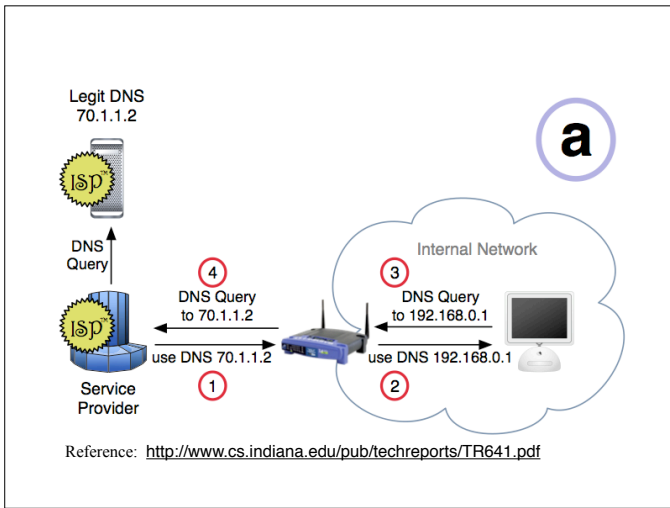


Reference: <http://www.cs.indiana.edu/pub/techreports/TR641.pdf>

## Drive-by pharming



Reference: <http://www.cs.indiana.edu/pub/techreports/TR641.pdf>



## DNSSEC

- ◆ Goals: authentication and integrity of DNS requests and responses
- ◆ PK-DNSSEC (public key)
  - DNS server signs its data (can be done in advance)
- ◆ SK-DNSSEC (symmetric key)
  - Encryption and MAC:  $E_k(m, \text{MAC}(m))$
  - Each message contains a nonce to avoid replay
  - Each DNS node shares a symmetric key with its parent
  - Zone root server has a public key (hybrid approach)