

Network Security 5



matt blaze ✓
@mattblaze

Following

Cryptocurrency somehow combines everything we love about religious fanatics with everything we love about Ponzi schemes.

7:08 AM - 23 Oct 2017

68 Retweets 128 Likes



4

68

128



Tweet your reply



matt blaze ✓ @mattblaze · 18m

Replying to @mattblaze

There's a lot to dislike about the world we're in, but at least Ayn Rand didn't have bitcoin to write about.

Hack of the Day:

International House of Pancakes

- IHOP uses Accellion's File Transfer Appliance
 - Effectively a customized web/data server for transferring sensitive data with access controls
 - So any data on the appliance was innately sensitive!
 - Oh, and technically discontinued months ago and end-of-life'd, but 🙄
 - Which had 4 vulnerabilities discovered in December and January:
 - 1x SQL injection (translation: no prepared SQL)
 - 2x OS Command Injection (translation: using **system** instead of **execve**)
 - 1x SSRF
- Two separate criminal gangs massively exploited these systems
 - And are now threatening IHOP with releasing data...
 - And others are following on and using that data for further attacks...

April Fools!!!!

- IHOP is one of the few big companies not affected by this data breach...
- But the breach is very very real and going to be very very expensive
- Because, lets correct the slide...

Hack of the Day:

~~International House of Pancakes~~ UC Berkeley

- ~~IHOP~~ UC Berkeley uses Accellion's File Transfer Appliance
 - Effectively a customized web/data server for transferring sensitive data with access controls
 - So any data on the appliance was innately sensitive!
 - Oh, and technically discontinued months ago and end-of-life'd, but 🙄
 - Which had 4 vulnerabilities discovered in December and January:
 - 1x SQL injection (translation: no prepared SQL)
 - 2x OS Command Injection (translation: using **system** instead of **execve**)
 - 1x SSRF
- Two separate criminal gangs massively exploited these systems
 - And are now threatening UC Berkeley with releasing data...
 - And others are following on and using that data for further attacks...

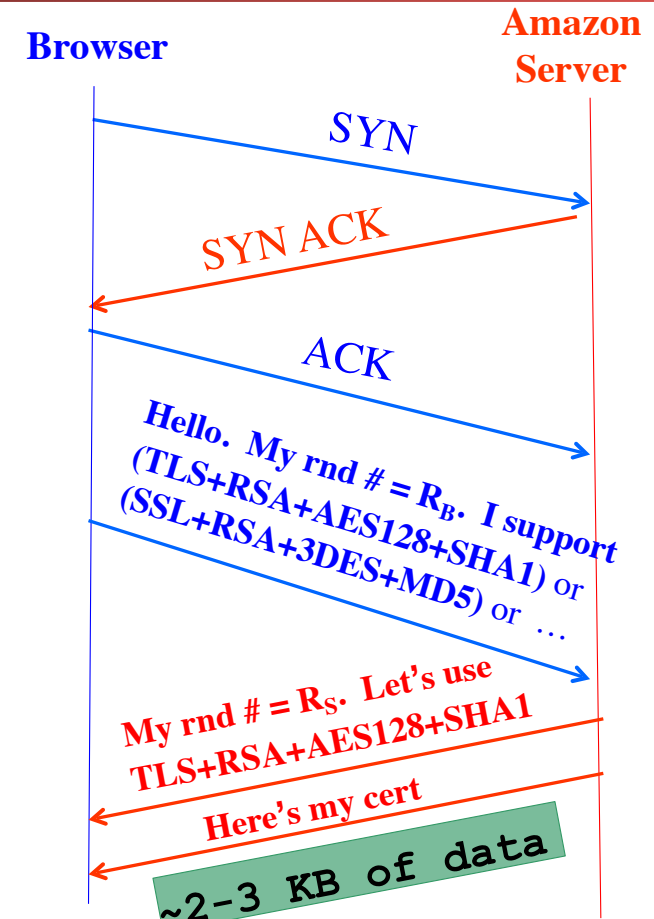
And Thus The Scary Emails...

- #1: the data breach emails
 - UC should know what data was copied...
 - But it is a Metric Crapton of PII
 - And there are California Data Breach notification laws that will make this very expensive
- #2: the follow-on phishing emails
 - Using the data obtained to try to trick users into providing the necessary data to file a fraudulent tax return


Reminder:

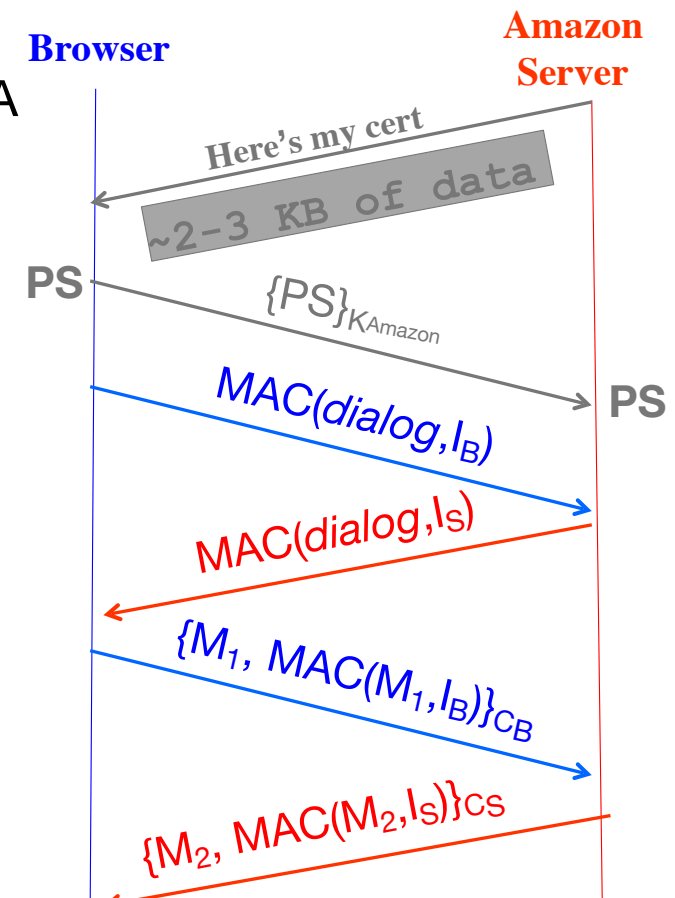
HTTPS Connection (SSL / TLS)

- Browser (client) connects via TCP to Amazon's HTTPS server
- Client picks 256-bit random number R_B , sends over list of crypto protocols it supports
- Server picks 256-bit random number R_S , selects protocols to use for this session
- Server sends over its certificate
 - (all of this is in the clear)
- Client now **validates** cert



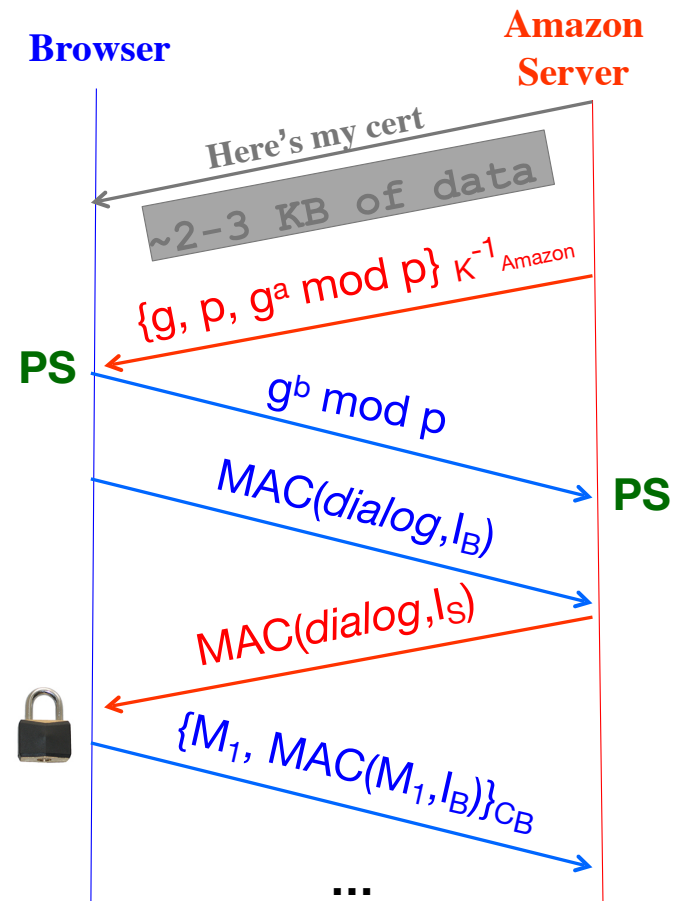
HTTPS Connection (SSL / TLS), cont.

- For RSA, browser constructs “Premaster Secret” PS
- Browser sends PS encrypted using Amazon’s public RSA key K_{Amazon}
- Using PS, R_B , and R_S , browser & server derive symm. cipher keys (C_B , C_S) & MAC integrity keys (I_B , I_S)
 - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displays 
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
 - Sequence #'s thwart replay attacks



Alternative: Ephemeral Key Exchange via Diffie-Hellman

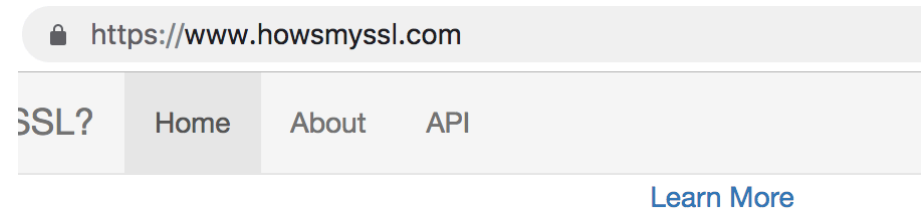
- For Diffie-Hellman (DHE), server generates random a , sends public parameters and $g^a \bmod p$
 - Signed with server's private key
- Browser verifies signature
- Browser generates random b , computes $PS = g^{ab} \bmod p$, sends $g^b \bmod p$ to server
- Server also computes $PS = g^{ab} \bmod p$
- Remainder is as before: from PS , R_B , and R_S , browser & server derive symm. cipher keys (C_B , C_S) and MAC integrity keys (I_B , I_S), etc...



Cipher Suite Negotiation

Computer Science 161

- Chrome's cipher-suite information
 - Client sends to the server
 - Server then chooses which one it wants
- Chrome does a trick tho:
 - Concern that servers might not respond properly if they don't know all the cipher suites...
 - So Chrome adds a garbage one:
GREASE_IS_THE_WORD_{RAND}
to break all the web servers that don't follow the standard



Given Cipher Suites

The cipher suites your client said it supports, in the order it sent them, are:

- TLS_GREASE_IS_THE_WORD_9A
- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

Certificates

- Cert = signed statement about someone's public key
 - Note that a cert does not say anything about the identity of who gives you the cert
 - It simply states a given public key K_{Bob} belongs to Bob ...
 - ... and backs up this statement with a digital signature made using a different public/private key pair, say from Verisign (a "Certificate Authority")
- Bob then can prove his identity to you by you sending him something encrypted with K_{Bob} ...
 - ... which he then demonstrates he can read
- ... or by signing something he demonstrably uses
- Works provided you trust that you have a valid copy of Verisign's public key ...
 - ... and you trust Verisign to use prudence when she signs other people's keys

Validating Amazon's Identity

- Browser compares domain name in cert w/ URL
 - Note: this provides an **end-to-end** property (as opposed to say a cert associated with an IP address)
- Browser accesses separate cert belonging to issuer
 - These are hardwired into the browser – **and trusted!**
 - There could be a chain of these ...
- Browser applies issuer's public key to verify signature **S**, obtaining the hash of what the issuer signed
 - Compares with its own SHA-1 hash of Amazon's cert
- Assuming hashes match, now have high confidence it's indeed Amazon's public key ...
 - assuming signatory is trustworthy, didn't lose private key, wasn't tricked into signing someone else's certificate, and that Amazon didn't lose their key either...

End-to-End \Rightarrow Powerful Protections

- Attacker runs a sniffer to capture our WiFi session?
 - But: encrypted communication is unreadable
 - No problem!
- DNS cache poisoning?
 - Client goes to wrong server
 - But: detects impersonation
 - No problem!
- Attacker hijacks our connection, injects new traffic
 - But: data receiver rejects it due to failed integrity check since all communication has a mac on it
 - No problem!
- Only thing a ***full man-in-the-middle*** attacker can do is inject RSTs, inject invalid packets, or drop packets: limited to a ***denial of service***

Validating Amazon's Identity, cont.

- Browser retrieves cert belonging to the issuer
 - These are hardwired into the browser – and trusted!
- But what if the browser can't find a cert for the issuer?



This Connection is Untrusted

You have asked Firefox to connect securely to **www.mikestoolbox.org**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Get me out of here!

▼ Technical Details

www.mikestoolbox.org uses an

The certificate is not trusted by

(Error code: sec_error_untruste

► I Understand the Risks



Validating Amazon's Identity, cont.

- Browser retrieves cert belonging to the issuer
 - These are hardwired into the browser – and trusted!
- What if browser can't find a cert for the issuer?
- If it can't find the cert, then warns the user that site has not been verified
 - Can still proceed, just without authentication
- Q: Which end-to-end security properties do we lose if we incorrectly trust that the site is whom we think?
- A: All of them!
 - Goodbye confidentiality, integrity, authentication
 - Active attacker can read everything, modify, impersonate

SSL / TLS Limitations

- Properly used, SSL / TLS provides powerful end-to-end protections
- So why not use it for everything??
- Issues:
 - Cost of public-key crypto (fairly minor)
 - Takes non-trivial CPU processing (but today a minor issue)
 - Note: symmetric key crypto on modern hardware is effectively free
 - Hassle of buying/maintaining certs (fairly minor)
 - LetsEncrypt makes this almost automatic
 - Integrating with other sites that don't use HTTPS
 - Namely, you can't: Non-HTTPS content won't load!
 - Latency: extra round trips \Rightarrow 1st page slower to load

SSL / TLS Limitations, cont.

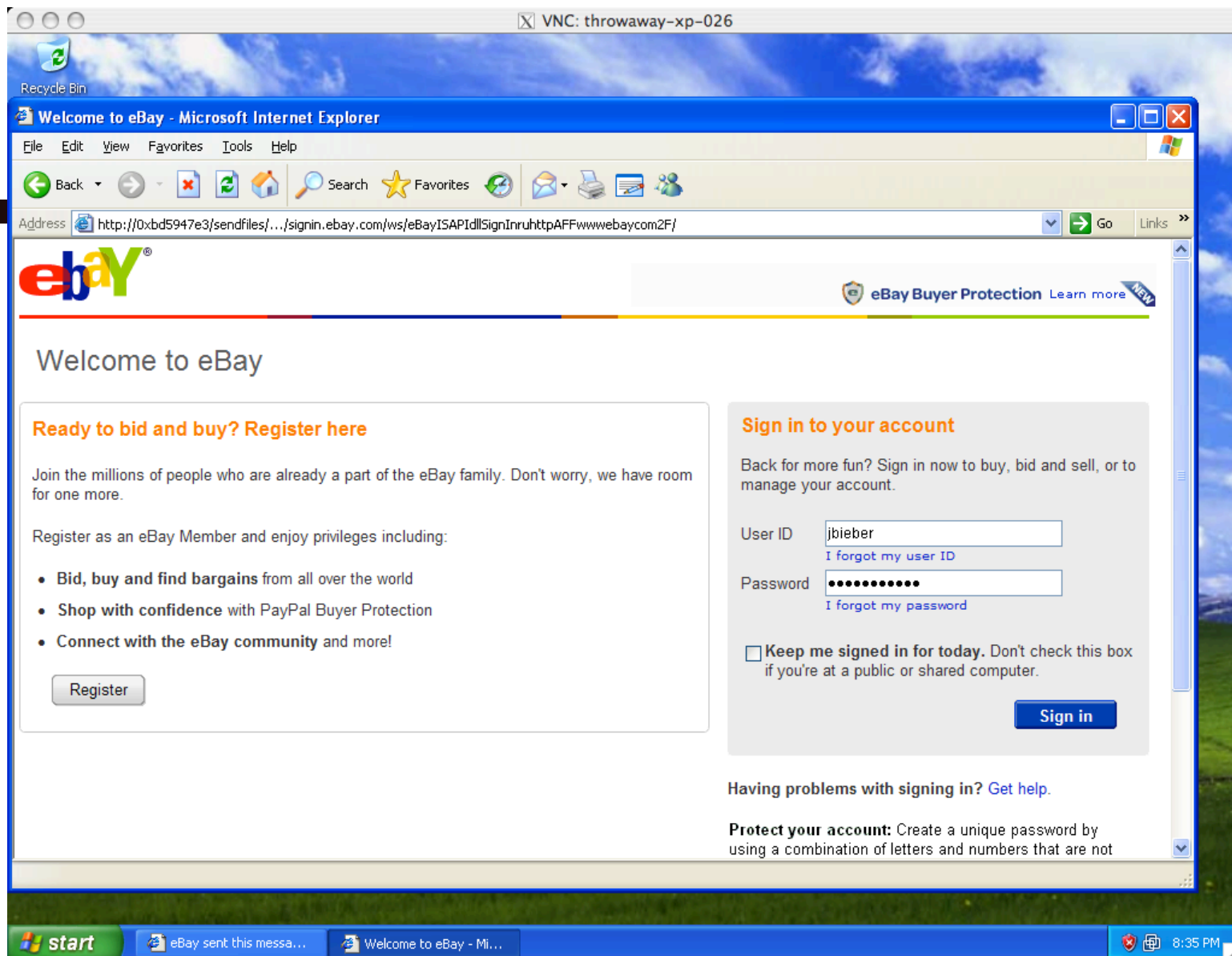
- Problems that SSL / TLS does not take care of ?
- Censorship:
 - The censor sees the certificate in the clear, so knows who the client is talking to
 - Optional Server Name Identification (SNI) is also sent in the clear
 - The censor can then inject RSTs or block the communication
 - TLS 1.3 supports encrypting the certificate & SNI (ESNI), but....
 - Censors are just blocking all ESNI connections.
- SQL injection/XSS/CSRF/server-side coding/logic flaws
- Vulnerabilities introduced by server inconsistencies

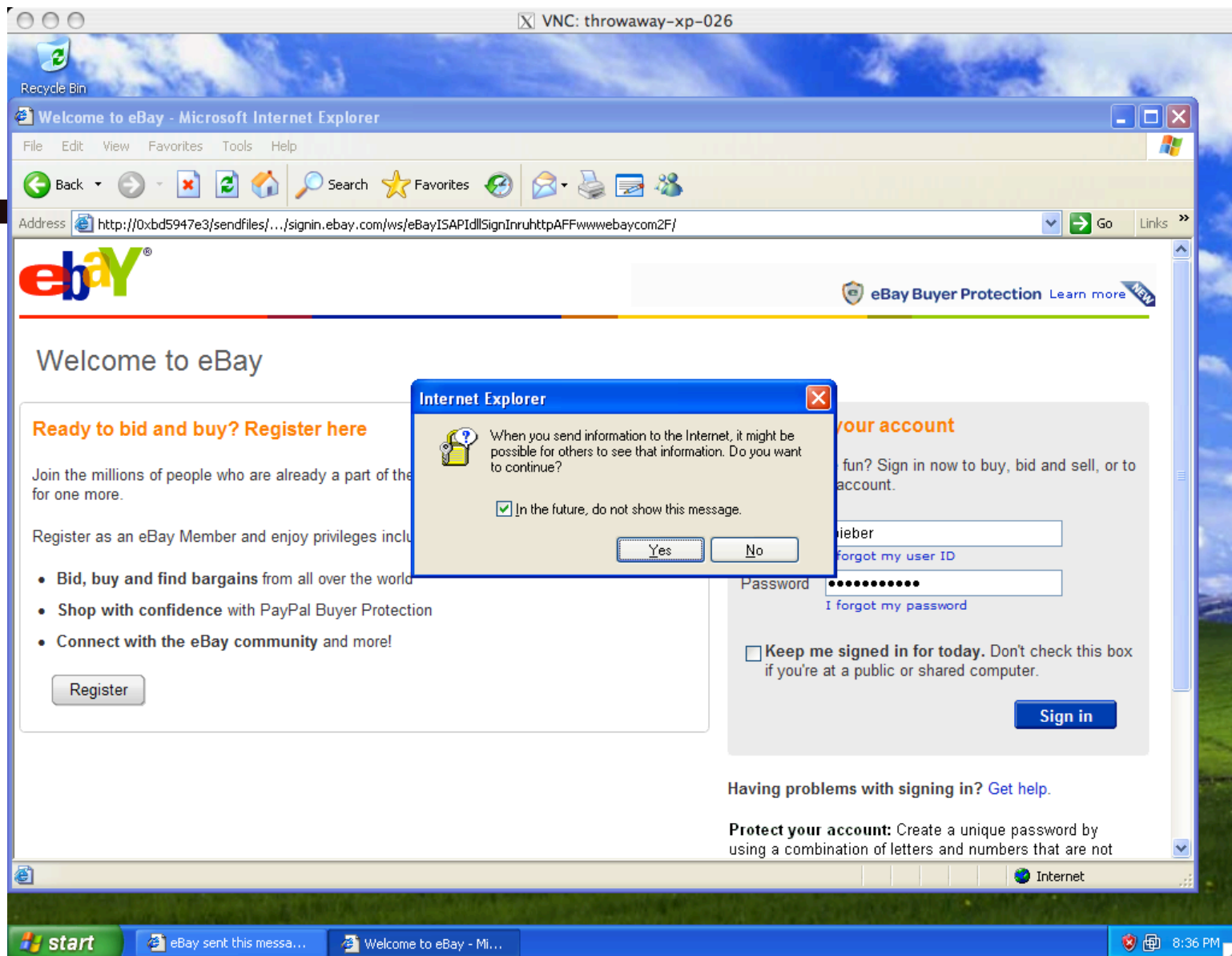
SSL/TLS Problem: Revocation

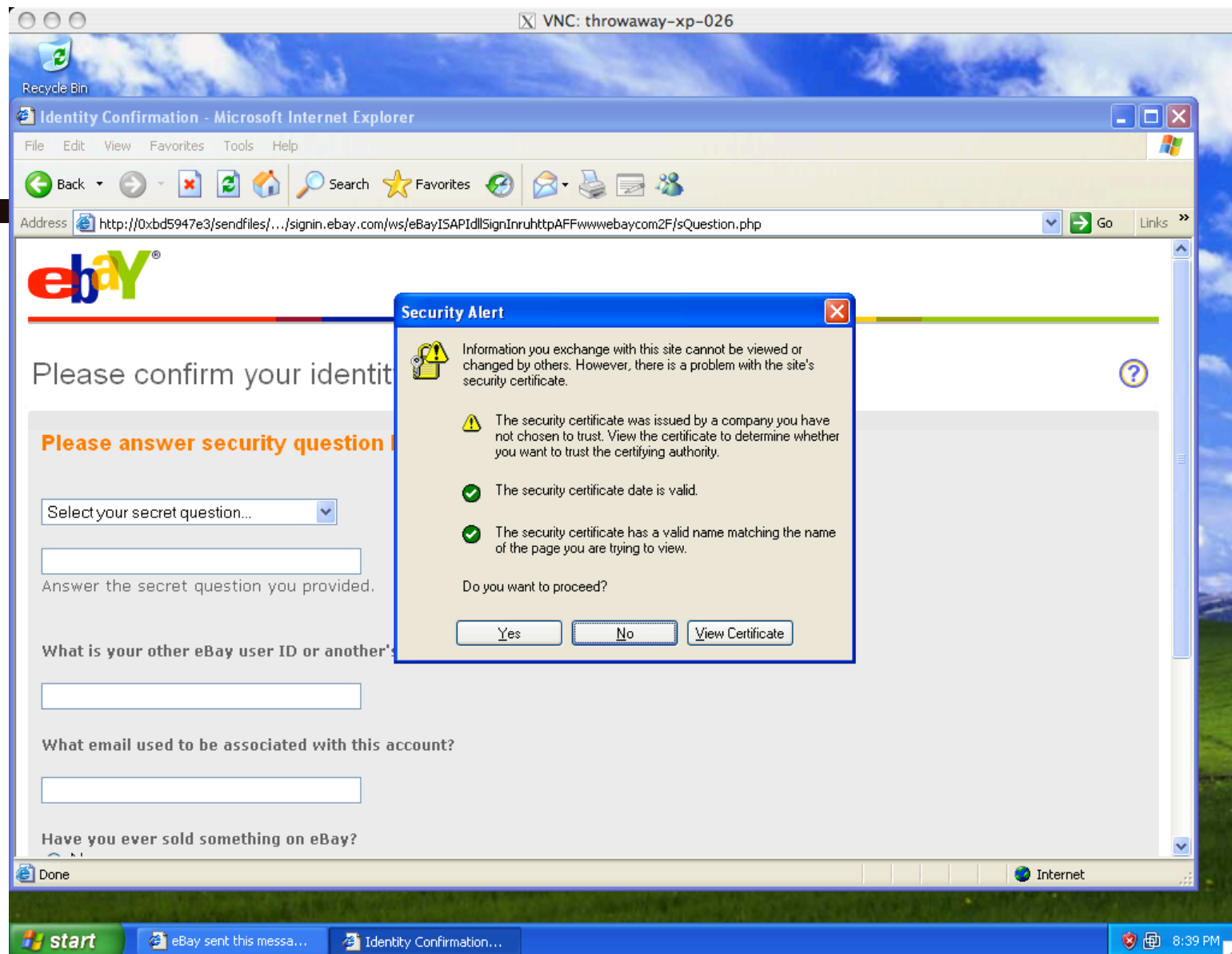
- A site screws up and an attacker steals the private key associated with a certificate, what now?
 - Certificates have a timestamp and are only good for a specified time
 - But this time is measured in years?!?
- Two mitigations:
 - Certificate revocation lists
 - Your browser occasionally calls back to get a list of "no longer accepted" certificates
 - OSCP
 - Online Certificate Status Protocol:
https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol

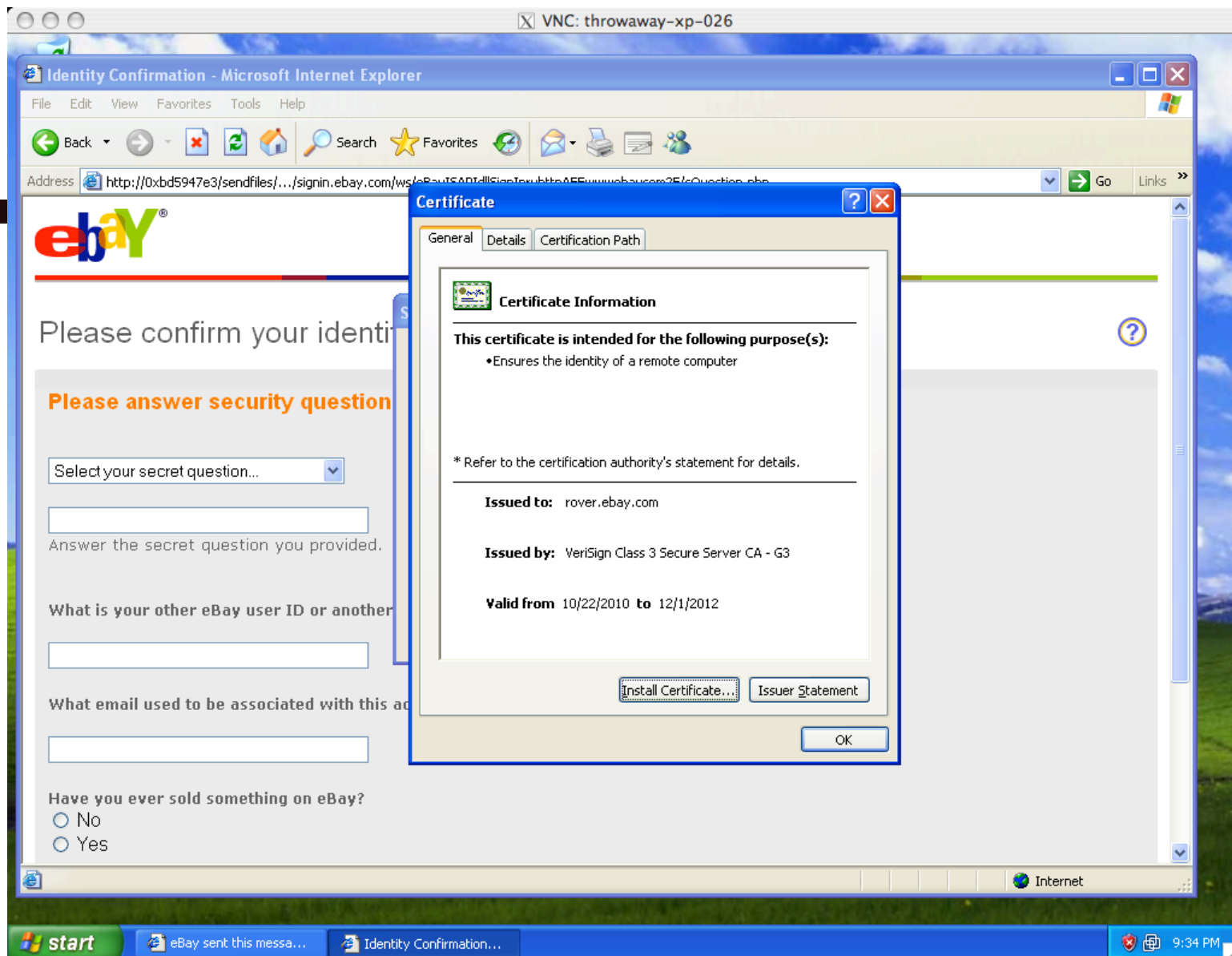
TLS/SSL Trust Issues

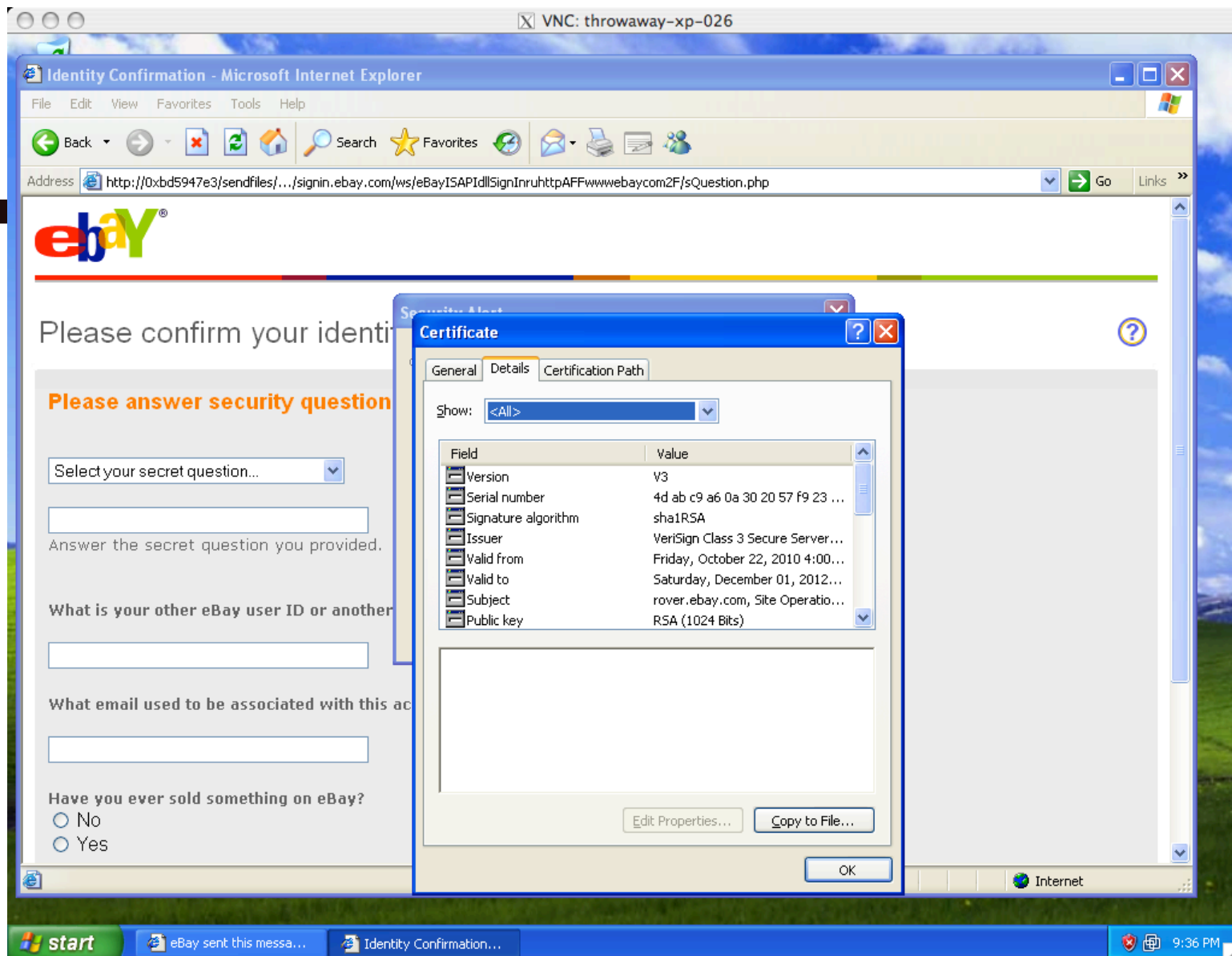
- User has to make correct trust decisions ...

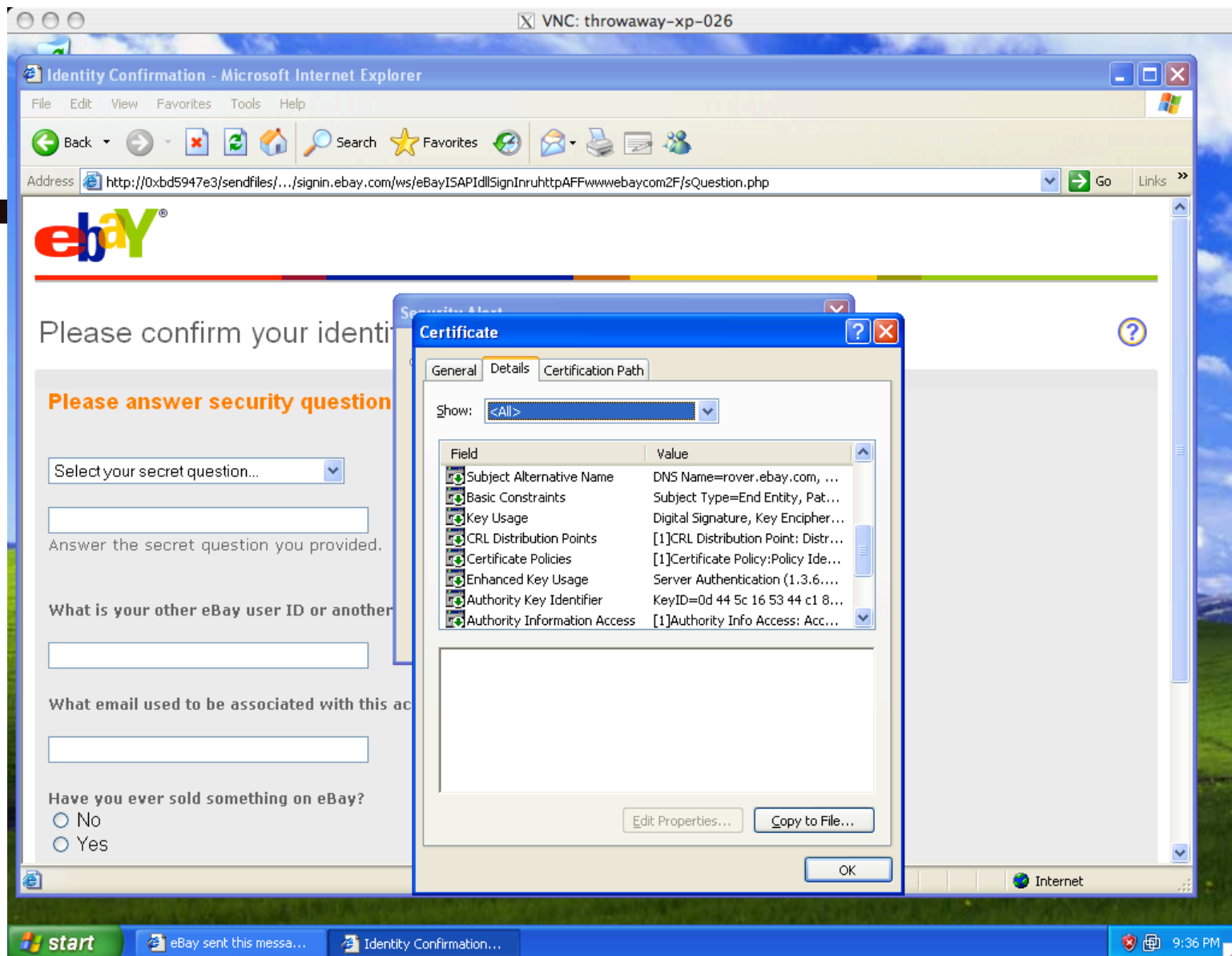


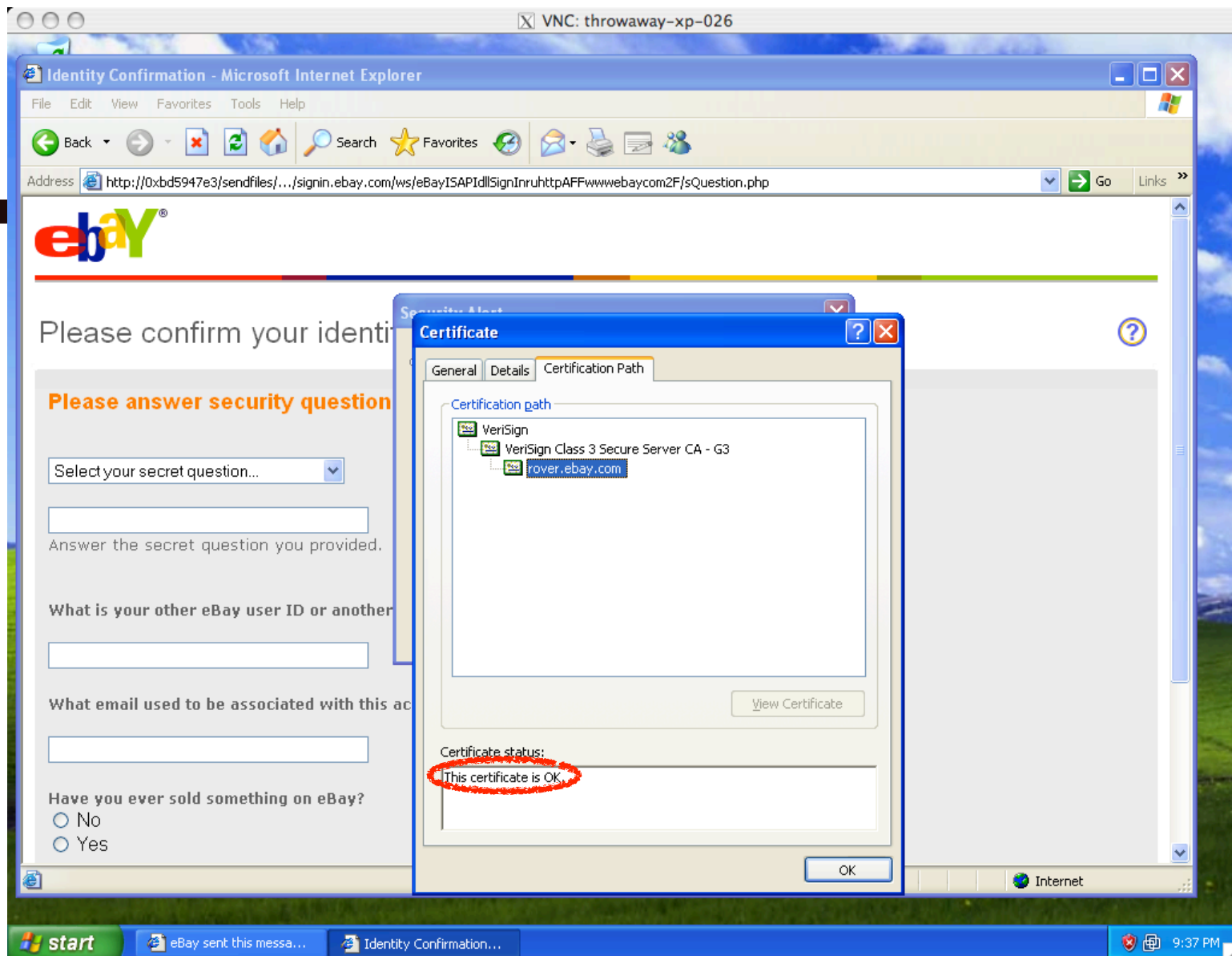




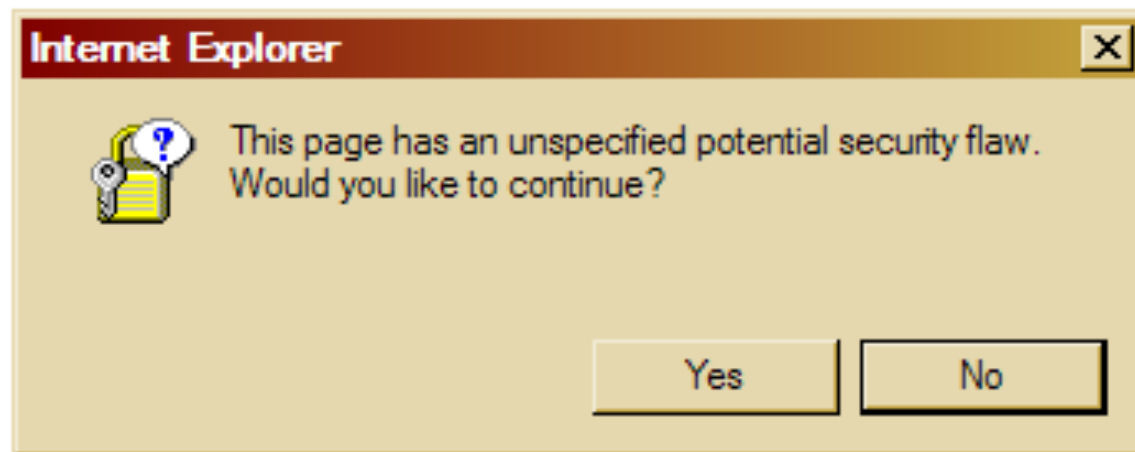








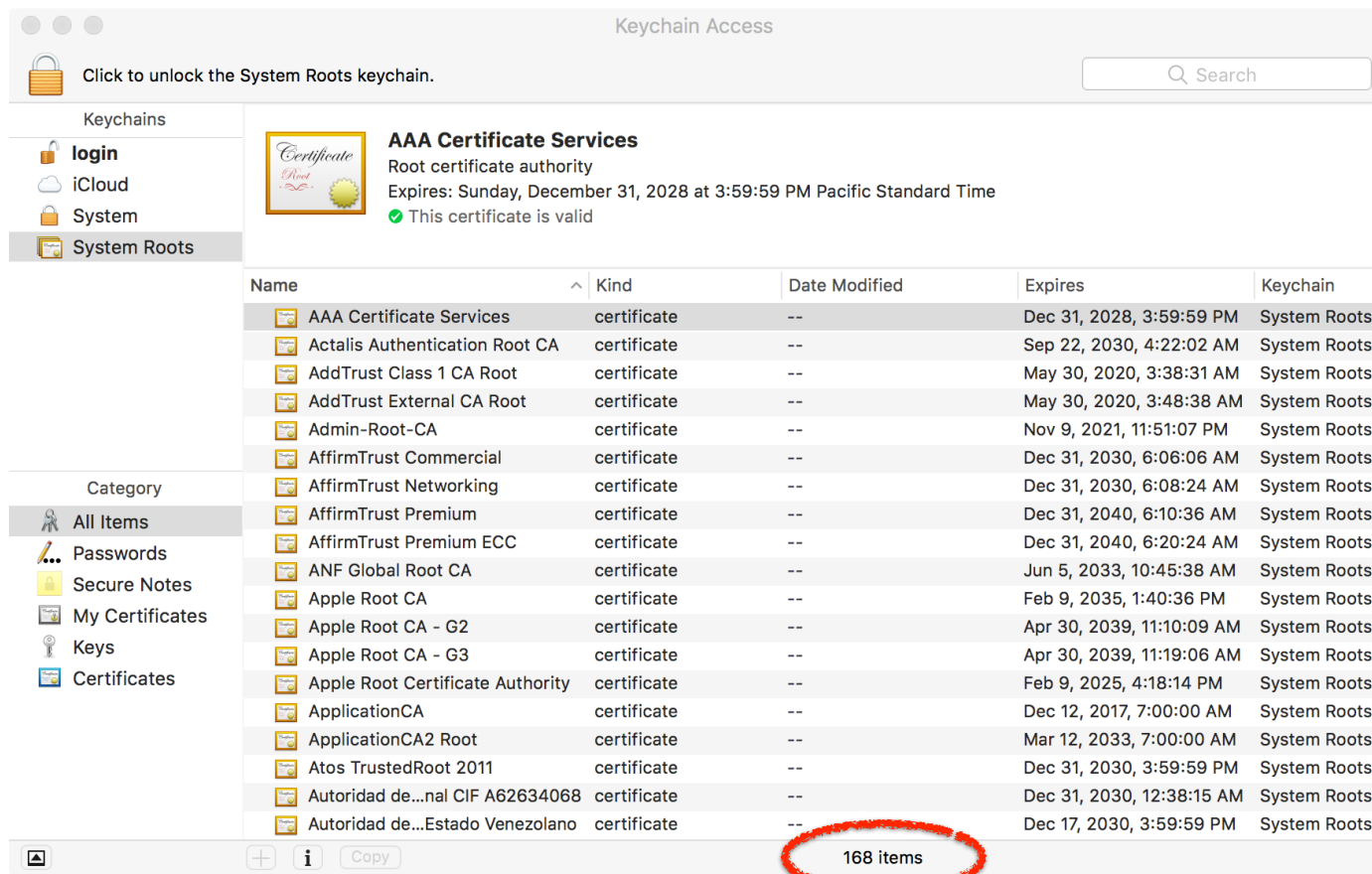
The equivalent as seen by most Internet users:



(note: an actual Windows error message!)

TLS/SSL Trust Issues, cont.

- “*Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.*”
 - Matt Blaze, circa 2001
- So how many CAs do we have to worry about, anyway?



TLS/SSL Trust Issues

- “*Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.*”
 - Matt Blaze, circa 2001
- So how many CAs do we have to worry about, anyway?
- Of course, it's not just their greed that matters ...

News

Solo Iranian hacker takes credit for Comodo certificate attack

Security researchers split on whether 'ComodoHacker' is the real deal

By Gregg Keizer

March 27, 2011 08:39 PM ET

 [Comments \(5\)](#)

 [Recommended \(37\)](#)

 [Like](#)

84

Computerworld - A solo Iranian hacker on Saturday claimed responsibility for stealing multiple SSL certificates belonging to some of the Web's biggest sites, including Google, Microsoft, Skype and Yahoo.

Early reaction from security experts was mixed, with some believing the hacker's claim, while others were dubious.

Fraudulent Google certificate points to Internet attack

Weaver

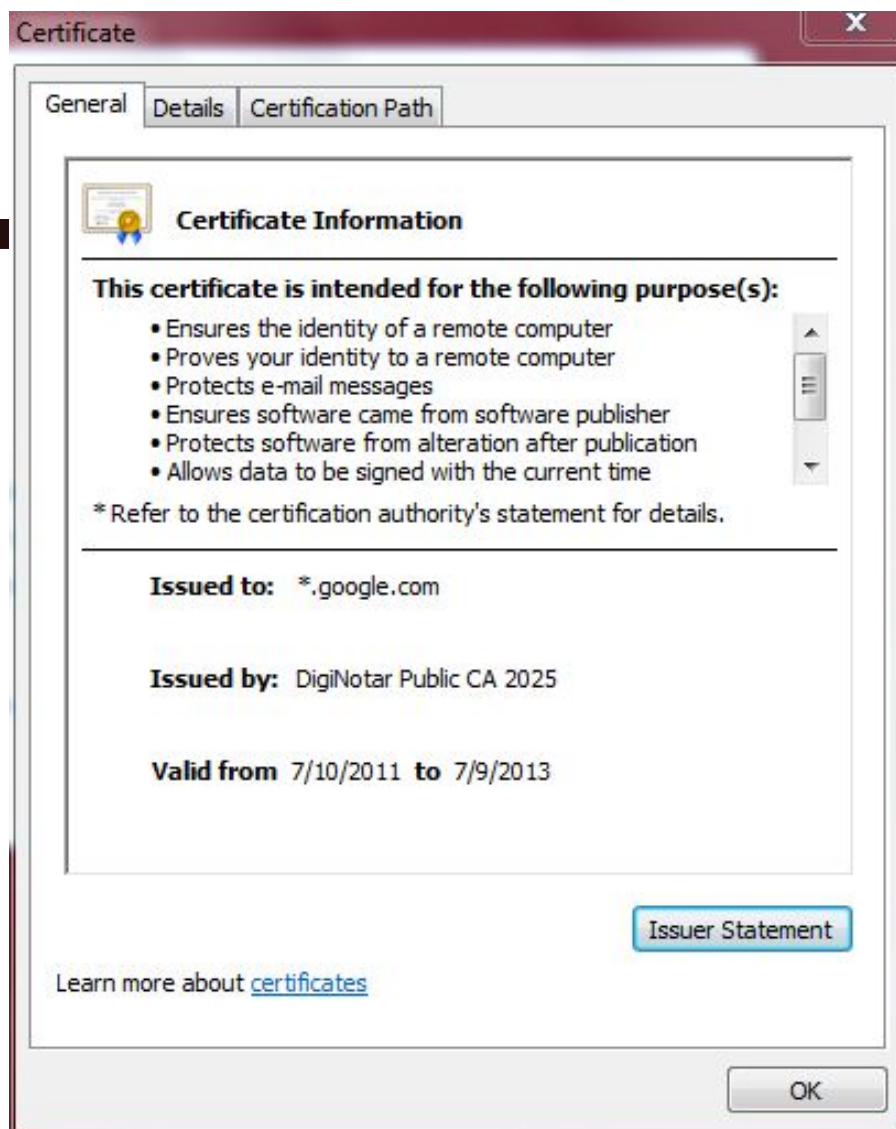
Is Iran behind a fraudulent Google.com digital certificate? The situation is similar to one that happened in March in which spoofed certificates were traced back to Iran.



by [Elinor Mills](#) | August 29, 2011 1:22 PM PDT

A Dutch company appears to have issued a digital certificate for Google.com to someone other than Google, who may be using it to try to re-direct traffic of users based in Iran.

Yesterday, someone reported on a Google support site that when attempting to log in to Gmail the browser issued a warning for the digital certificate used as proof that the site is legitimate, according to [this thread](#) on a Google support forum site.



This appears to be a fully valid cert using normal browser validation rules.

Only detected by Chrome due to its introduction of cert “pinning” – requiring that certs for certain domains must be signed by specific CAs rather than any generally trusted CA

October 31, 2012, 10:49AM

Final Report on DigiNotar Hack Shows Total Compromise of CA Servers

The attacker who penetrated the Dutch CA DigiNotar last year had complete control of all eight of the company's certificate-issuing servers during the operation and he may also have issued some rogue certificates that have not yet been identified. The final report from a

Evidence Suggests DigiNotar, Who Issued Fraudulent Google Certificate, Was Hacked *Years Ago*

from the *diginot* dept

The big news in the security world, obviously, is the fact that a **fraudulent Google certificate made its way out into the wild**, apparently targeting internet users in Iran. The Dutch company DigiNotar has put out a statement saying that **it discovered a breach** back on July 19th during a security audit, and that fraudulent certificates were generated for "several dozen" websites. The only one known to have gotten out into the wild is the Google one.

The DigiNotar Fallout

- The result was the “CA Death Sentence”:
 - Web browsers removed it from the trusted root certificate store
- This happened again with “WoSign”
 - A Chinese CA
- WoSign would allow an interesting attack
 - If I controlled `nweaver.github.com...`
 - WoSign would allow me to create a certificate for `*.github.com!?!?`
 - And a bunch of other shady shenanigans

TLS/SSL Trust Issues

- “Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.”
 - Matt Blaze, circa 2001
- So how many CAs do we have to worry about, anyway?
- Of course, it’s not just their greed that matters ...
- ... and it’s not just their diligence & security that matters ...
 - *“A decade ago, I observed that commercial certificate authorities protect you from anyone from whom they are unwilling to take money. That turns out to be wrong; they don't even do that much.”* - Matt Blaze, circa 2010

So the Modern Solution: Invoke Ronald Reagan, “Trust, but *Verify*”

- Static Certificate Pinning:
The chrome browser has a list of certificates or certificate authorities that it trusts for given sites
 - Now creating a fake certificate requires attacking a *particular* CA
- Transparency mechanisms:
 - Public logs provided by certificate authorities
 - As a hash chain: We are actually serious so we don't call it a “blockchain”
 - Coupled with the server able to say “ONLY accept certificates from me that are from a CA implementing transparency”
 - Browser extensions (EFF's TLS observatory)
 - Backbone monitors (ICSI's TLS notary)

SSL/TLS Problem: Revocation

- A site screws up and an attacker steals the private key associated with a certificate, what now?
 - Certificates have a timestamp and are only good for a specified time
 - But this time is measured in years?!?
- Two mitigations:
 - Certificate revocation lists
 - Your browser occasionally calls back to get a list of "no longer accepted" certificates
 - OSCP
 - Online Certificate Status Protocol:
https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol

And Making It Cheap: LetsEncrypt...

- Coupled to the depreciation of unencrypted HTTP...
 - Need to be able to have HTTPS be just about the same complexity...
- Idea: Make it easy to "prove" you own a web site:
 - Can you write an arbitrary cookie at an arbitrary location?
- Build ***automated*** infrastructure to do this
 - Script to create a private key
 - Generate a certificate signing request
 - PKI authority says "here's a file, put it on the server"
 - Script puts it on the server
 - PKI now returns certificate...
 - Signed with a limited duration so you ***must*** automate this process

And Now A Song: 50 Whys to Stop A Server...

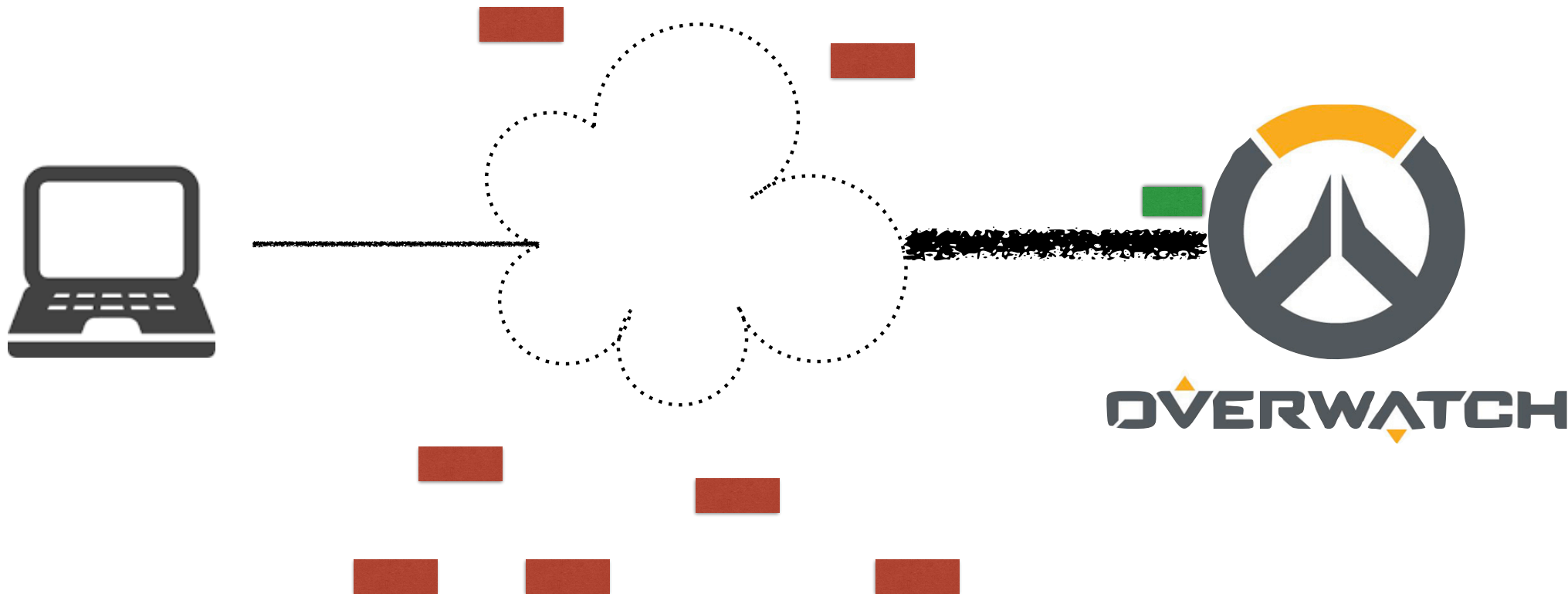
- You are a bad guy...
 - And you want to stop some server from being **available**
- Why? You name it...
 - Because its hard for someone to frag you in an online game if you "boot" him from the network
 - Because people will pay up to stop the attack
 - Because it conveys a political message
 - Get paid for by others



The Easy DoS on a System: Resource Consumption...

- Bad Dude has an account on your computer...
 - And wants to disrupt your work on Project 2...
- He runs this simple program:
 - while(1):
 - Write random junk to random files
 - (uses disk space, thrashes the disk)
 - Allocate a bunch of RAM and write to it
 - (uses memory)
 - fork()
 - (creates more processes to run)
- Only defense is some form of quota or limits:
The system itself ***must*** enforce some isolation

The Network DOS



DoS & Networks

- How could you DoS a target's Internet access?
 - Send a zillion packets at them
 - Internet lacks **isolation** between traffic of different users!
- What resources does attacker need to pull this off?
 - At least as much sending capacity (**bandwidth**) as the bottleneck link of the target's Internet connection
 - Attacker sends maximum-sized packets
 - Or: overwhelm the rate at which the bottleneck router can process packets
 - Attacker sends minimum-sized packets!
 - (in order to maximize the packet arrival rate)

Defending Against Network DoS

- Suppose an attacker has access to a beefy system with high-speed Internet access (a “big pipe”).
- They pump out packets towards the target at a very high rate.
- What might the target do to defend against the onslaught?
 - Install a network filter to discard any packets that arrive with attacker’s IP address as their source
 - E.g., `drop * 66.31.33.7:* -> *:*`
 - Or it can leverage any other pattern in the flooding traffic that’s not in benign traffic
 - Note, the filter needs to be **before** the bottleneck!
 - Attacker’s IP address = means of identifying misbehaving user

Filtering Sounds Pretty Easy ...

- ... but DoS filters can be easily evaded:
 - Make traffic appear as though it's from many hosts
 - Spoof the source address so it can't be used to filter
 - Just pick a random 32-bit number of each packet sent
 - How does a defender filter this?
 - They don't!
 - Best they can hope for is that operators around the world implement anti-spoofing mechanisms (today about 75% do)
 - Use many hosts to send traffic rather than just one
 - Distributed Denial-of-Service = DDoS (“dee-doss”)
 - Requires defender to install complex filters
 - How many hosts is “enough” for the attacker?
 - Today they are very cheap to acquire ... :-)

It's Not A “Level Playing Field”

- When defending resources from exhaustion, need to beware of asymmetries, where attackers can consume victim resources with little comparable effort
 - Makes DoS easier to launch
 - Defense costs much more than attack
- Particularly dangerous form of asymmetry: amplification
 - Attacker leverages system's own structure to pump up the load they induce on a resource

Amplification

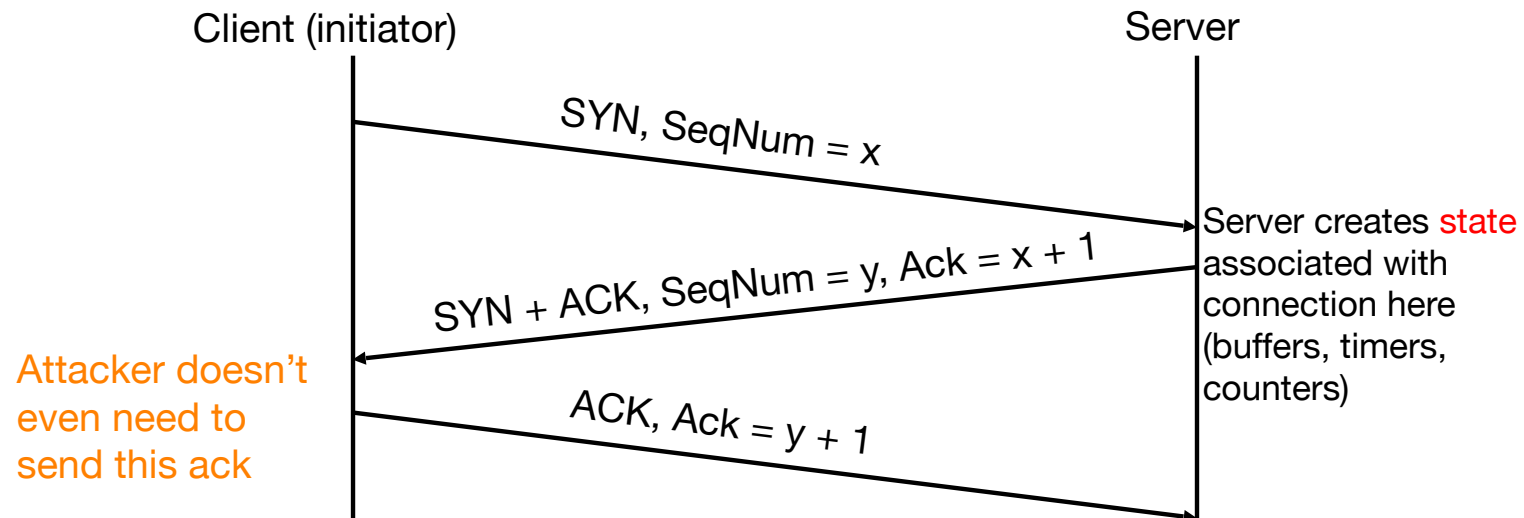
- Example of amplification: DNS lookups
 - Reply is generally much bigger than request
 - Since it includes a copy of the reply, plus answers etc.
 - Attacker spoofs DNS request to a patsy DNS server, seemingly from the target
 - Small attacker packet yields large flooding packet
 - Doesn't increase # of packets, but total volume
- Note #1: these examples involve blind spoofing
 - So for network-layer flooding, generally only works for UDP-based protocols (can't establish a TCP connection)
 - But any single-packet UDP protocol where the response is bigger can be used for amplification!
- Note #2: victim doesn't see spoofed source addresses
 - Addresses are those of actual intermediary systems

Botnets

- If an attacker can control a *lot* of systems
 - They gain a huge amount of bandwidth
 - Modern DOS attacks approach 1 Terabit-per-second with direct connections
 - And it becomes very hard to filter them out
 - How do you specify 1M machines you want to ignore?
- You control these "bots" in a "botnet"
 - So you can issue commands that cause all these systems to do what you want
- This is what took down dyn DNS (and with it Twitter, Reddit, etc...) two years ago: A botnet composed primarily of compromised cameras and DVRs:
 - The Miraj botnet

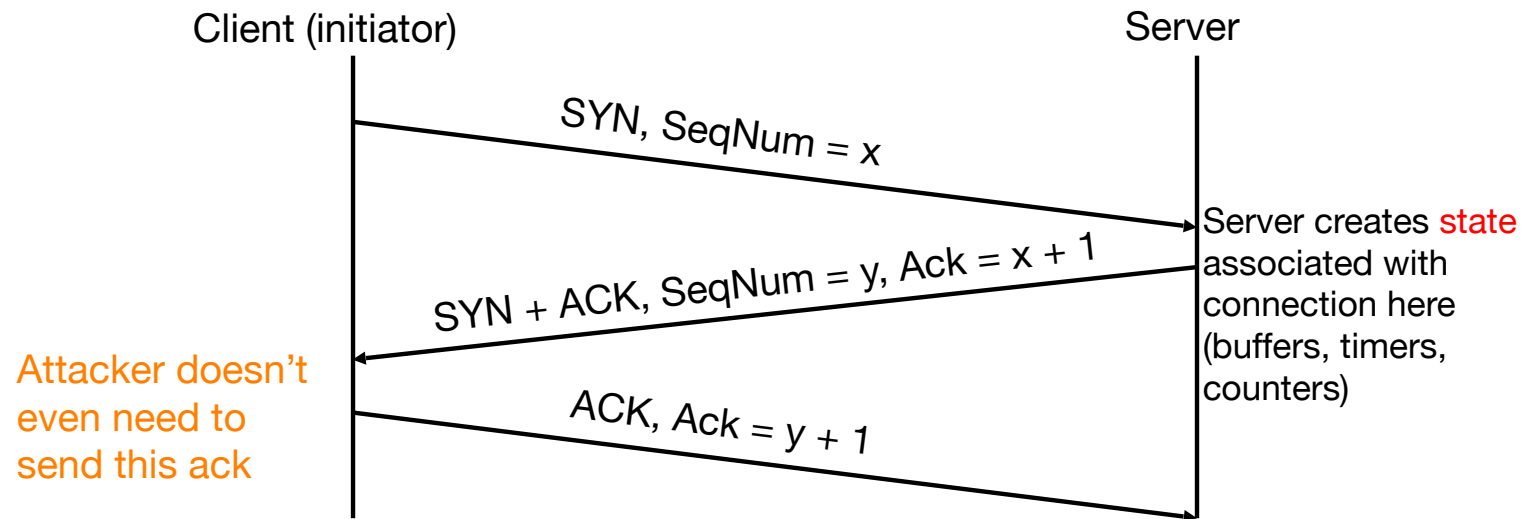
Transport-Level Denial-of-Service

- Recall TCP's 3-way connection establishment handshake
 - Goal: agree on initial sequence numbers



Transport-Level Denial-of-Service

- Recall TCP's 3-way connection establishment handshake
 - Goal: agree on initial sequence numbers
- So a single SYN from an attacker suffices to force the server to spend some memory



TCP SYN Flooding

- Attacker targets memory rather than network capacity
- Every (unique) SYN that the attacker sends burdens the target
- What should target do when it has no more memory for a new connection?
- No good answer!
 - Refuse new connection?
 - Legit new users can't access service
 - Evict old connections to make room?
 - Legit old users get kicked off

TCP SYN Flooding Defenses

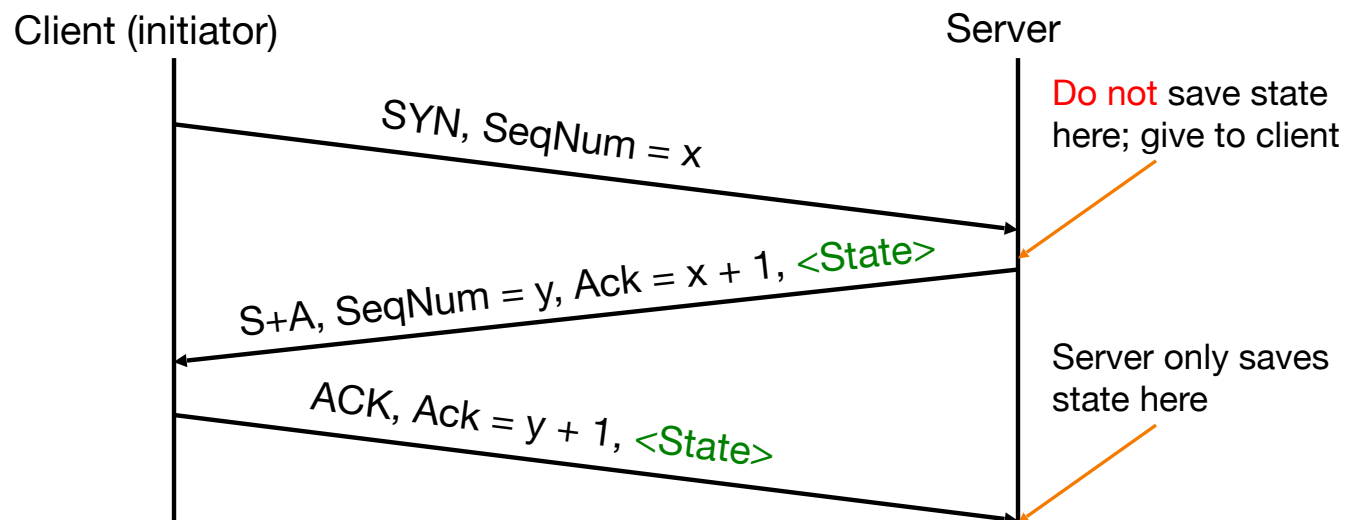
- How can the target defend itself?
- Approach #1: make sure they have tons of memory!
 - How much is enough?
 - Depends on resources attacker can bring to bear (threat model), which might be hard to know
- Back of the envelope:
 - If we need to hold 10kB for 1 minute: to exhaust 1GB, an attacker needs...
 - 100k packets/minute, or a bit more than 1,000 packets per second

TCP SYN Flooding Defenses

- Approach #2: identify bad actors & refuse their connections
 - Hard because only way to identify them is based on IP address
 - We can't for example require them to send a password because doing so requires we have an established connection!
 - For a public Internet service, who knows which addresses customers might come from?
 - Plus: attacker can spoof addresses since they don't need to complete TCP 3-way handshake
- Approach #3: don't keep state! ("SYN cookies"; only works for spoofed SYN flooding)

SYN Flooding Defense: Idealized

- Server: when SYN arrives, rather than keeping state locally, send it to the client ...
- Client needs to return the state in order to establish connection



SYN Flooding Defense: Idealized

- Server: when SYN arrives, rather than keeping state locally, send it to the client.
- Client needs to establish connection

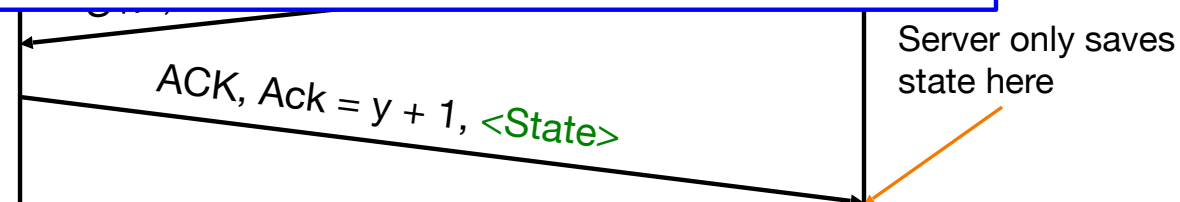
Problem: the world isn't so ideal!

TCP doesn't include an easy way to add a new <State> field like this.

Is there any way to get the same functionality without having to change TCP clients?

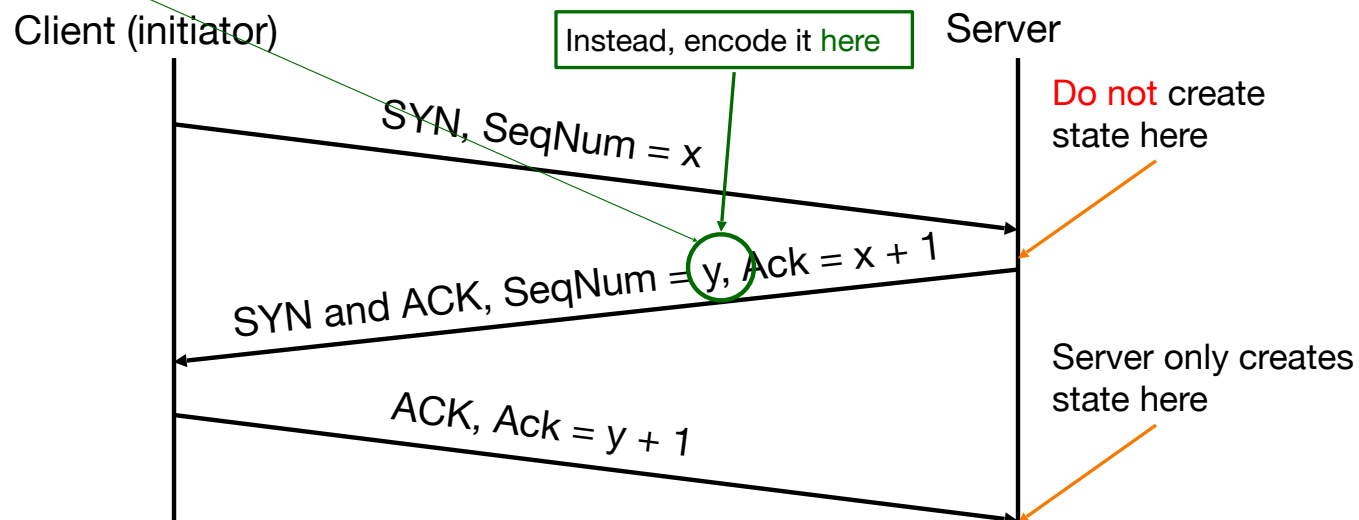
Client

save state
give to client



Practical Defense: SYN Cookies

- Server: when SYN arrives, encode connection state entirely within SYN-ACK's sequence # y
- y = encoding of necessary state, using server secret
- When ACK of SYN-ACK arrives, server only creates state if value of y from it agrees w/ secret



SYN Cookies: Discussion

- Illustrates general strategy: rather than holding state, encode it so that it is returned when needed
- For SYN cookies, attacker must complete 3-way handshake in order to burden server
 - Can't use spoofed source addresses
- Note #1: strategy requires that you have enough bits to encode all the state
 - (This is just barely the case for SYN cookies)
 - You can think of a SYN cookie as a truncated MAC of the sender IP/port/sequence:
And really, HMAC is the easiest way to do this!
- Note #2: if it's expensive to generate or check the cookie, then it's not a win

And Once Again, HMAC to the rescue...

- HMAC is a great way to force others to store state...
 - Create cookie:
 $\text{HMAC}(k, \text{data}) \rightarrow$ 🍪
 - Check cookie:
 $\text{HMAC}(k, \text{data}) \stackrel{?}{=}$ 🍪
- Allow you to force others to store all the data you want that you can then verify later
 - All you need to do is make sure that they know they need to send all the data back to you with the cookie...
 - And you need the cookie to be big **enough**

Application-Layer DoS

- Rather than exhausting network or memory resources, attacker can overwhelm a service's processing capacity
- There are many ways to do so, often at little expense to attacker compared to target (asymmetry)

Uncategorized

The Ethereum network is currently undergoing a DoS attack

Posted by [Jeffrey Wilcke](#) on  [September 22nd, 2016](#).

URGENT ALL MINERS: The network is under attack. The attack is a computational DDoS, ie. miners and nodes need to spend a very long time processing some blocks. This is due to the EXTCODESIZE opcode, which has a fairly low gasprice but which requires nodes to read state information from disk; the attack transactions are calling this opcode roughly 50,000 times per block. The consequence of this is that the network is greatly slowing down, but there is NO consensus failure

Algorithmic complexity attacks

- Attacker can try to trigger worst-case complexity of algorithms / data structures
- Example: You have a hash table.
Expected time: $O(1)$. Worst-case: $O(n)$.
- Attacker picks inputs that cause hash collisions.
Time per lookup: $O(n)$.
Total time to do n operations: $O(n^2)$.
- Solution? Use algorithms with good worst-case running time.
 - E.g., using b bits of HMAC ensures that $P[h_k(x)=h_k(y)] = .5^b$, so hash collisions will be rare.
 - If the attacker doesn't know the key that is

Application-Layer DoS

- Defenses against such attacks?
- Approach #1: Only let legit users issue expensive requests
 - Relies on being able to identify/authenticate them
 - Note: that this itself might be expensive!
- Approach #2: Force legit users to “burn” cash
 - This is what a captcha really is!
- Approach #3: massive over-provisioning (\$\$\$)
 - Or pay for someone else who massively over provisions for everyone:
A content delivery network

DoS Defense in General Terms

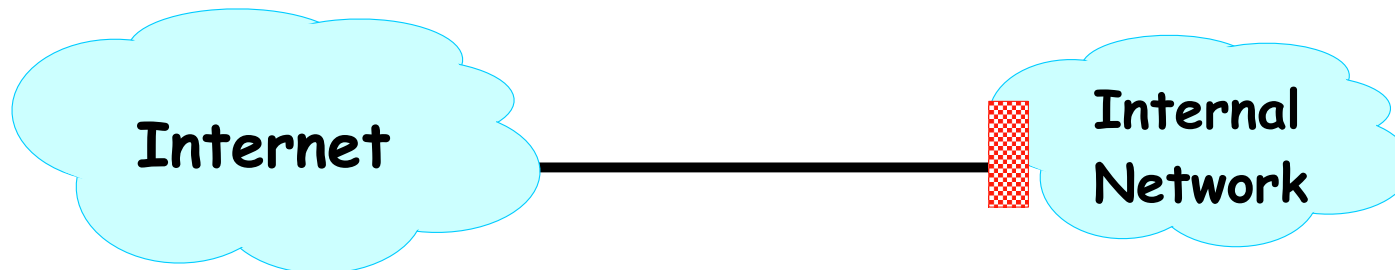
- Defending against program flaws requires:
 - Careful design and coding/testing/review
 - Consideration of behavior of defense mechanisms
 - E.g. buffer overflow detector that when triggered halts execution to prevent code injection ⇒ denial-of-service
- Defending resources from exhaustion can be really hard.
Requires:
 - Isolation and scheduling mechanisms
 - Keep adversary's consumption from affecting others
 - Reliable identification of different users
 - Or just a ton of \$\$\$\$

Controlling Networks ... On The Cheap

- Motivation: How do you harden a set of systems against external attack?
 - Key Observation:
 - The more network services your machines run, the greater the risk
 - Due to larger attack surface
- One approach: on each system, turn off unnecessary network services
 - But you have to know all the services that are running
 - And sometimes some trusted remote users still require access
- Plus key question of scaling
 - What happens when you have to secure 100s/1000s of systems?
 - Which may have different OSs, hardware & users ...
 - Which may in fact not all even be identified ...

Taming Management Complexity

- Possibly more scalable defense: Reduce risk by blocking in the network outsiders from having unwanted access your network services
 - Interpose a firewall the traffic to/from the outside must traverse
 - Chokepoint can cover thousands of hosts
 - Where in everyday experience do we see such chokepoints?



Selecting a Security Policy

- Firewall enforces an (access control) policy:
 - Who is allowed to talk to whom, accessing what service?
- Distinguish between inbound & outbound connections
 - Inbound: attempts by external users to connect to services on internal machines
 - Outbound: internal users to external services
 - Why? Because fits with a common threat model. There are thousands of internal users (and we've vetted them). There are billions of outsiders.
- Conceptually simple access control policy:
 - Permit inside users to connect to any service
 - External users restricted:
 - Permit connections to services meant to be externally visible
 - Deny connections to services not meant for external access

How To Treat Traffic Not Mentioned in Policy?

- Default Allow: start off permitting external access to services
 - Shut them off as problems recognized
- Default Deny: start off permitting just a few known, well-secured services
 - Add more when users complain (and mgt. approves) ✓
- Pros & Cons?
 - Flexibility vs. conservative design
 - Flaws in Default Deny get noticed more quickly / less painfully

In general, use Default Deny

A Dumb Policy: Deny All Inbound connections...

- The simplest packet filters are ***stateless***
 - They examine only individual packets to make a decision
- But even the simplest policy can be hard to implement
 - Deny All Inbound is the default policy on your home connection
- Allow:
 - Any outbound packet
 - Any inbound packet that is a reply... OOPS
- We can fake it for TCP with some ugly hacks
 - Allow all outbound TCP
 - Allow all inbound TCP that does not have both the SYN flag set and the ACK flag not set
 - May still allow an attacker to play some interesting games
- We can't even fake this for UDP!