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

7:08 AM - 23 Oct 2017

matt blaze @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.
After 10 failed logins, Giuliani had Apple Store wipe his iPhone: Report

The 2017 incident occurred shortly after Trump named Rudy cybersecurity advisor.

TIMOTHY B. LEE - 10/31/2019, 12:20 PM

On January 12, 2017, President-elect Donald Trump named Rudy Giuliani to be his cybersecurity advisor. A month later, on February 7, Giuliani walked into a San Francisco Apple Store with a problem: his iPhone had gotten locked down after 10 unsuccessful passcode attempts, NBC reports.
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

Browser

Amazon Server

SYN

SYN ACK

ACK

Hello. My rnd # = $R_B$. I support (TLS+RSA+AES128+SHA1) or (SSL+RSA+3DES+MD5) or ...

My rnd # = $R_S$. Let's use TLS+RSA+AES128+SHA1

Here's my cert

~2-3 KB of data
HTTPS Connection (SSL / TLS), cont.

• For RSA, browser constructs “Premaster Secret” PS
• Browser sends PS encrypted using Amazon’s public RSA key KAmazon
• 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 \( g^a \mod p \)
  - Signed with server’s private key
- Browser verifies signature
- Browser generates random b, computes \( PS = g^{ab} \mod p \), sends \( g^b \mod p \) to server
- Server also computes \( PS = g^{ab} \mod p \)
- 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...

Here’s my cert

\(~2–3\) KB of data

\[ \{g, p, g^a \mod p\} k^{-1}_{Amazon} \]
Cipher Suite Negotiation

- Chrome's cipher-suite information
  - Client sends to the server
  - Server then chooses which one it wants
    - It should pick the common mode that both prefer
- Then its the bulk encryption
- Then key exchanges with encryption mode
  - Description is key exchange, signature (if necessary), and then bulk cipher & hash
Why $R_b$ and $R_s$?

- Both $R_b$ and $R_s$ act to affect the keys... Why?
  - Keys = $F(R_b || R_s || PS)$
- Needed to prevent a **replay attack**
  - Attacker captures the handshake from either the client or server and replays it...
- If the other side choses a different $R$ the next time...
  - The replay attack fails.
- But you don't need to check for reuse by the other side..
  - Just make sure you don't reuse it on your side!
And Sabotaged pRNGs...

- Let us assume the server is using DHE...
  - If an attacker can know $a$, they have all the information needed to decrypt the traffic:
    - Since $PS = g^{ab}$, and can see $g^b$.
- TLS spews a lot of "random" numbers publicly as well
  - Nonces in the crypto, $R_s$, etc...
- If the server uses a bad pRNG which is both sabotaged and doesn't have *rollback resistance*...
  - Dual_EC DRBG where you know the secret used to create the generator...
  - ANSI X9.31: An AES based one with a secret key...
- Attacker sees the handshake, sees subsequent PRNG calls, works *backwards* to get the secret
  - Attack of the week: DUHK
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
“sslstrip”
(Amazon FINALLY fixed this recently)

Regular web surfing: http: URL

So no integrity - a MITM attacker can alter pages returned by server ...

And when we click here ...

... attacker has changed the corresponding link so that it’s ordinary http rather than https!

We never get a chance to use TLS’s protections! :-(

Amazon.com: Online Shopping for...

Strathwood Patio Furniture up to 40% Off
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 ⇒ 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 insecure server certificate.

The certificate is not trusted because:

(Error code: sec_error_untrusted)

I Understand the Risk:
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 ⇒ 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

• 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
TLS/SSL Trust Issues

- User has to make correct trust decisions …
Welcome to eBay

Ready to bid and buy? Register here

Join the millions of people who are already a part of the eBay community.

Register as an eBay Member and enjoy privileges including:
- Bid, buy and find bargains from all over the world
- Shop with confidence with PayPal Buyer Protection
- Connect with the eBay community and more!

Register

Internet Explorer

When you send information to the Internet, it might be possible for others to see that information. Do you want to continue?

- In the future, do not show this message

Yes No

Password ************

I forgot my password

Keep me signed in for today. Don’t check this box if you’re at a public or shared computer.

Sign in

Having problems with signing in? Get help.

Protect your account: Create a unique password by using a combination of letters and numbers that are not
Please confirm your identity.

Please answer security question.

Select your secret question:

Answer the secret question you provided.

What is your other eBay user ID or another?

What email used to be associated with this account?

Have you ever sold something on eBay?

Do you want to proceed?

Yes  No  View Certificate
Please confirm your identification.

Please answer security questions:

Select your secret question...

Answer the secret question you provided:

What is your other eBay user ID or another email address used to associates this account?

Have you ever sold anything on eBay?

No

Yes
Please confirm your identity.

Please answer security question:

Select your secret question:

Answer the secret question you provided:

What is your eBay user ID or another identifier?

What email used to be associated with this account?

Have you ever sold something on eBay?

Yes

No

Certificate Details:

Field | Value
--- | ---
Subject Alternative Name | DNS Name=ever.ebay.com...
Basic Constraints | Subjek Topo=End Entity, Full...
Key Usage | Digital Signature, Key Encipher...
CRL Distribution Points | [email] CRL Distribution Point, Dist...
Enhanced Key Usage | Server Authentication (1.3.6...
Authority Key Identifier | KeyID=00 44 5f 36 83 44 c1 8...
Authority Information Access | Authority Info Access: Authority...

Copy to File...
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?
Click to unlock the System Roots keychain.

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Date Modified</th>
<th>Expires</th>
<th>Keychain</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA Certificate Services</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2028, 3:59:59 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Actalis Authentication Root CA</td>
<td>certificate</td>
<td>--</td>
<td>Sep 22, 2030, 4:22:02 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Class 1 CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020, 3:38:31 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust External CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020, 3:48:38 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Admin-Root-CA</td>
<td>certificate</td>
<td>--</td>
<td>Nov 9, 2021, 11:51:07 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 6:06:06 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 6:08:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2040, 6:10:36 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2040, 6:20:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ANF Global Root CA</td>
<td>certificate</td>
<td>--</td>
<td>Jun 5, 2033, 10:45:38 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>--</td>
<td>Feb 9, 2035, 1:40:36 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA - G2</td>
<td>certificate</td>
<td>--</td>
<td>Apr 30, 2039, 11:09:09 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA - G3</td>
<td>certificate</td>
<td>--</td>
<td>Apr 30, 2039, 11:19:06 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>--</td>
<td>Feb 9, 2026, 4:18:14 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ApplicationCA</td>
<td>certificate</td>
<td>--</td>
<td>Dec 12, 2017, 7:00:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ApplicationCA2 Root</td>
<td>certificate</td>
<td>--</td>
<td>Mar 12, 2033, 7:00:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Atos TrustedRoot 2011</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 3:59:59 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Autoridad de...nai CIF A62634068</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 12:38:15 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Autoridad de...Estado Venezolano</td>
<td>certificate</td>
<td>--</td>
<td>Dec 17, 2030, 3:59:59 PM</td>
<td>System Roots</td>
</tr>
</tbody>
</table>
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 …
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

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

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.
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)
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
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 it’s 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
  - 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

Client (initiator) — Server

SYN, SeqNum = x

SYN + ACK, SeqNum = y, Ack = x + 1

ACK, Ack = y + 1

Server creates state associated with connection here (buffers, timers, counters)

Attacker doesn’t even need to send this ack
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

\[\text{Client (initiator)}\]

\[\text{SYN, SeqNum} = x\]

\[\text{SYN + ACK, SeqNum} = y, \text{Ack} = x + 1\]

\[\text{ACK, Ack} = y + 1\]

\[\text{Server}\]

\[\text{Server creates state associated with connection here (buffers, timers, counters)}\]

Attacker doesn’t even need to send this ack
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 return the state in order to establish a 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?

ACK, Ack = y + 1, <State>

Server only saves state here
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/server 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