HTTP header injection
- Untrusted data included in response headers
- Can include CRLF and new headers, or premature end to headers
- AKA “response splitting”

Content sniffing
- Browsers determine file type from headers, extension, and content-based guessing
  - Latter two for ~1% server errors
- Many sites host “untrusted” images and media
- Inconsistencies in guessing lead to a kind of XSS
  - E.g., “chimera” PNG-HTML document

Cross-site request forgery
- Certain web form on bank.com used to wire money
- Link or script on evil.com loads it with certain parameters
  - Linking is exception to same-origin
- If I’m logged in, money sent automatically
- Confused deputy, cookies are ambient authority

CSRF prevention
- Give site’s forms random-nonce tokens
  - E.g., in POST hidden fields
  - Not in a cookie, that’s the whole point
- Reject requests without proper token
  - Or, ask user to re-authenticate
- XSS can be used to steal CSRF tokens

Open redirects
- Common for one page to redirect clients to another
- Target should be validated
  - With authentication check if appropriate
- Open redirect: target supplied in parameter with no checks
  - Doesn’t directly hurt the hosting site
  - But reputation risk, say if used in phishing
  - We teach users to trust by site

Misconfiguration problems
- Default accounts
- Unneeded features
- Framework behaviors
  - Don’t automatically create variables from query fields
Openness tradeoffs
- Error reporting
  - Few benign users want to see a stack backtrace
- Directory listings
  - Hallmark of the old days
- Readable source code of scripts
  - Doesn't have your DB password in it, does it?

Using vulnerable components
- Large web apps can use a lot of third-party code
- Convenient for attackers too
  - OWASP: two popular vulnerable components downloaded 22m times
- Hiding doesn't work if it's popular
- Stay up to date on security announcements

Clickjacking
- Fool users about what they’re clicking on
  - Circumvent security confirmations
  - Fabricate ad interest
- Example techniques:
  - Frame embedding
  - Transparency
  - Spool cursor
  - Temporal “bait and switch”

Crawling and scraping
- A lot of web content is free-of-charge, but proprietary
  - Yours in a certain context, if you view ads, etc.
- Sites don’t want it downloaded automatically (web crawling)
- Or parsed and user for another purpose (screen scraping)
- High-rate or honest access detectable

Outline
- More web risks
- Confidentiality and privacy
- Announcements intermission
- More crypto protocols
- More causes of crypto failure
- Firewalls and NAT boxes
- Intrusion detection systems

Site perspective
- Protect confidentiality of authenticators
  - Passwords, session cookies, CSRF tokens
- Duty to protect some customer info
  - Personally identifying info ("identity theft")
  - Credit-card info (Payment Card Industry Data Security Standards)
  - Health care (HIPAA), education (FERPA)
  - Whatever customers reasonably expect

You need to use SSL
- Finally coming around to view that more sites need to support HTTPS
  - Special thanks to WiFi, NSA
- If you take credit cards (of course)
- If you ask users to log in
  - Must be protecting something, right?
  - Also important for users of Tor et al.

Server-side encryption
- Also consider encrypting data “at rest”
  - (Or, avoid storing it at all)
- Provides defense in depth
  - Reduce damage after another attack
- May be hard to truly separate keys
  - OWASP example: public key for website → backend credit card info
**Adjusting client behavior**

- HTTPS and password fields are basic hints
- Consider disabling autocomplete
  - Usability tradeoff, save users from themselves
  - Finally standardized in HTML5
- Consider disabling caching
  - Performance tradeoff
  - Better not to have this on user's disk
  - Or proxy? You need SSL

**User vs. site perspective**

- User privacy goals can be opposed to site goals
- Such as in tracking for advertisements
- Browser makers can find themselves in the middle
  - Of course, differ in institutional pressures

**Third party content / web bugs**

- Much tracking involves sites other than the one in the URL bar
  - For fun, check where your cookies are coming from
- Various levels of cooperation
- Web bugs are typically 1x1 images used only for tracking

**Cookies arms race**

- Privacy-sensitive users like to block and/or delete cookies
- Sites have various reasons to retain identification
- Various workarounds:
  - Similar features in Flash and HTML5
  - Various channels related to the cache
  - Evercookie: store in n places, regenerate if subset are deleted

**Browser fingerprinting**

- Combine various server or JS-visible attributes passively
  - User agent string (10 bits)
  - Window/screen size (4.83 bits)
  - Available fonts (13.9 bits)
  - Plugin versions (15.4 bits)

(Data from panopticlick.eff.org, far from exhaustive)

**History stealing**

- History of what sites you’ve visited is not supposed to be JS-visible
- But, many side-channel attacks have been possible
  - Query link color
  - CSS style with external image for visited links
  - Slow-rendering timing channel
  - Harvesting bitmaps
  - User perception (e.g. fake CAPTCHA)

**Browser and extension choices**

- More aggressive privacy behavior lives in extensions
  - Disabling most JavaScript (NoScript)
  - HTTPS Everywhere (whitelist)
  - Tor Browser Bundle
- Default behavior is much more controversial
  - Concern not to kill advertising support as an economic model

**Outline**

- More web risks
- Confidentiality and privacy
- Announcements intermission
- More crypto protocols
- More causes of crypto failure
- Firewalls and NAT boxes
- Intrusion detection systems
Upcoming events

- Individual progress reports due tonight
- Exercise set 4 out, due next Wednesday
- Project meetings next week
- HA2 due a week from Friday

Outline

- More web risks
- Confidentiality and privacy
- Announcements intermission
- More crypto protocols
- More causes of crypto failure
- Firewalls and NAT boxes
- Intrusion detection systems

Abstract protocols

- Outline of what information is communicated in messages
  - Omit most details of encoding, naming, sizes, choice of ciphers, etc.
- Describes honest operation
  - But must be secure against adversarial participants
- Seemingly simple, but many subtle problems

Protocol notation

A → B : N_B, (T_B, B, N_B)_{K_B}
A → B: message sent from Alice intended for Bob
B (after :): Bob's name
f g K: encryption with key K

Needham-Schroeder

Mutual authentication via nonce exchange, assuming public keys (core):

A → B : \{N_A, A\}_{E_B}
B → A : \{N_A, N_B\}_{E_A}
A → B : \{N_B\}_{E_B}

Needham-Schroeder MITM

A → C : \{N_A, A\}_{E_C}
C → B : \{N_A, A\}_{E_B}
B → C : \{N_A, N_B\}_{E_A}
C → A : \{N_A, N_B\}_{E_A}
A → C : \{N_B\}_{E_C}
C → B : \{N_B\}_{E_B}

ATTACK AGAINST DENNING-SACCO

A → S : A, B
S → A : C_A, C_B
A → B : C_A, C_B, (Sign_A(K_{AB}))_{K_S}
B → S : B, C
S → B : C_B, C_C
B → C : C_A, C_C, (Sign_A(K_{AB}))_{K_C}

By re-encrypting the signed key, Bob can pretend to be Alice to Charlie

Certificates, Denning-Sacco

- A certificate signed by a trusted third-party $S$ binds an identity to a public key
  - $C_A = \text{Sign}_S[A, K_A]$
- Suppose we want to use $S$ in establishing a session
  \[
  A \rightarrow S: A, B \\
  \text{key } K_{AB}: S \rightarrow A: C_A, C_B \\
  A \rightarrow B: C_A, C_B, (\text{Sign}_A(K_{AB}))_{K_S}
  \]
Envelopes analogy

- Encrypt then sign, or vice-versa?
- On paper, we usually sign inside an envelope, not outside. Two reasons:
  - Attacker gets letter, puts in his own envelope (c.f. attack against X.509)
  - Signer claims “didn’t know what was in the envelope” (failure of non-repudiation)

Design robustness principles

- Use timestamps or nonces for freshness
- Be explicit about the context
- Don’t trust the secrecy of others’ secrets
- Whenever you sign or decrypt, beware of being an oracle
- Distinguish runs of a protocol

Implementation principles

- Ensure unique message types and parsing
- Design for ciphers and key sizes to change
- Limit information in outbound error messages
- Be careful with out-of-order messages

Outline

- More web risks
- Confidentiality and privacy
- Announcements intermission
- More crypto protocols
- More causes of crypto failure
- Firewalls and NAT boxes
- Intrusion detection systems

Random numbers and entropy

- Cryptographic RNGs use cipher-like techniques to provide indistinguishability
- But rely on truly random seeding to stop brute force
  - Extreme case: no entropy → always same “randomness”
- Modern best practice: seed pool with 256 bits of entropy
  - Suitable for security levels up to $2^{256}$

Netscape RNG failure

- Early versions of Netscape SSL (1994-1995) seeded with:
  - Time of day
  - Process ID
  - Parent process ID
- Best case entropy only 64 bits
  - Not out of step with using 40-bit encryption
- But worse because many bits guessable

Debian/OpenSSL RNG failure (1)

- OpenSSL has pretty good scheme using /dev/urandom
- Also mixed in some uninitialized variable values
  - “Extra variation can’t hurt”
- From modern perspective, this was the original sin
  - Remember undefined behavior discussion?
- But had no immediate ill effects

Debian/OpenSSL RNG failure (2)

- Debian maintainer commented out some lines to fix a Valgrind warning
  - “Potential use of uninitialized value”
- Accidentally disabled most entropy (all but 16 bits)
- Brief mailing list discussion didn’t lead to understanding
- Broken library used for ~2 years before discovery
Detected RSA/DSA collisions

- 2012: around 1% of the SSL keys on the public net are breakable
  - Some sites share complete keypairs
  - RSA keys with one prime in common (detected by large-scale GCD)
- One likely culprit: insufficient entropy in key generation
  - Embedded devices, Linux /dev/urandom vs. /dev/random
- DSA signature algorithm also very vulnerable

Side-channel attacks

- Timing analysis:
  - Number of 1 bits in modular exponentiation
  - Unpadding, MAC checking, error handling
  - Probe cache state of AES table entries
- Power analysis
  - Especially useful against smartcards
- Fault injection
- Data non-erasure
  - Hard disks, "cold boot" on RAM

WEP “privacy”

- First WiFi encryption standard: Wired Equivalent Privacy (WEP)
- F&S: designed by a committee that contained no cryptographers
- Problem 1: note “privacy”: what about integrity?
  - Nope: stream cipher + CRC = easy bit flipping

WEP shared key

- Single key known by all parties on network
- Easy to compromise
- Hard to change
- Also often disabled by default
- Example: a previous employer

WEP key size and IV size

- Original sizes: 40-bit shared key (export restrictions) plus 24-bit IV = 64-bit RC4 key
  - Both too small
- 128-bit upgrade kept 24-bit IV
  - Vague about how to choose IVs
  - Least bad: sequential, collision takes hours
  - Worse: random or everyone starts at zero

WEP RC4 related key attacks

- Only true crypto weakness
- RC4 “key schedule” vulnerable when:
  - RC4 keys very similar (e.g., same key, similar IV)
  - First stream bytes used
- Not a practical problem for other RC4 users like SSL
  - Key from a hash, skip first output bytes

New problem with WPA (CCS’17)

- Session key set up in a 4-message handshake
- Key reinstallation attack: replay #3
  - Causes most implementations to reset nonce and replay counter
  - In turn allowing many other attacks
  - One especially bad case: reset key to 0
- Protocol state machine behavior poorly described in spec
  - Outside the scope of previous security proofs

Trustworthiness of primitives

- Classic worry: DES S-boxes
- Obviously in trouble if cipher chosen by your adversary
- In a public spec, most worrying are unexplained elements
- Best practice: choose constants from well-known math, like digits of π
Dual_EC_DRBG (1)
- Pseudorandom generator in NIST standard, based on elliptic curve
- Looks like provable (slow enough) but strangely no proof
- Specification includes long unexplained constants
- Academic researchers find:
  - Some EC parts look good
  - But outputs are statistically distinguishable

Dual_EC_DRBG (2)
- Found 2007: special choice of constants allows prediction attacks
- Big red flag for paranoid academics
- Significant adoption in products sold to US govt.
- FIPS-140 standards
  - Semi-plausible rationale from RSA (EMC)
- NSA scenario basically confirmed by Snowden leaks
- NIST and RSA immediately recommend withdrawal

Outline
- More web risks
- Confidentiality and privacy
- Announcements intermission
- More crypto protocols
- More causes of crypto failure
- Firewalls and NAT boxes
- Intrusion detection systems

Internet addition: middleboxes
- Original design: middle of net is only routers
  - End-to-end principle
- Modern reality: more functionality in the network
- Security is one major driver

Security/connectivity tradeoff
- A lot of security risk comes from a network connection
  - Attacker could be anywhere in the world
- Reducing connectivity makes security easier
- Connectivity demand comes from end users

What a firewall is
- Basically, a router that chooses not to forward some traffic
  - Based on an a-priori policy
- More complex architectures have multiple layers
  - DMZ: area between outer and inner layers, for outward-facing services

Inbound and outbound control
- Most obvious firewall use: prevent attacks from the outside
- Often also some control of insiders
  - Block malware-infected hosts
  - Employees wasting time on Facebook
  - Selling sensitive info to competitors
  - Nation-state Internet management
- May want to log or rate-limit, not block

Default: deny
- Usual whitelist approach: first, block everything
- Then allow certain traffic
- Basic: filter packets based on headers
- More sophisticated: proxy traffic at a higher level
IPv4 address scarcity
- Design limit of $2^{32}$ hosts
- Actually less for many reasons
- Addresses becoming gradually more scarce over a many-year scale
- Some high-profile exhaustions in 2011
- IPv6 adoption still quite low, occasional signs of progress

Network address translation (NAT)
- Middlebox that rewrites addresses in packets
- Main use: allow inside network to use non-unique IP addresses
  - RFC 1918: 10.*, 192.168.*, etc.
  - While sharing one outside IP address
- Inside hosts not addressable from outside
  - De-facto firewall

Packet filtering rules
- Match based on:
  - Source IP address
  - Source port
  - Destination IP address
  - Destination port
  - Packet flags: TCP vs. UDP, TCP ACK, etc.
- Action, e.g. allow or block
- Obviously limited in specificity

Client and server ports
- TCP servers listen on well-known port numbers
  - Often < 1024, e.g. 22 for SSH or 80 for HTTP
- Clients use a kernel-assigned random high port
- Plain packet filter would need to allow all high-port incoming traffic

Stateful filtering
- In general: firewall rules depend on previously-seen traffic
- Key instance: allow replies to an outbound connection
- See: port 23746 to port 80
- Allow incoming port 23746
  - To same inside host
  - Needed to make a NAT practical

Circuit-level proxying
- Firewall forwards TCP connections for inside client
- Standard protocol: SOCKS
  - Supported by most web browsers
  - Wrapper approaches for non-aware apps
- Not much more powerful than packet-level filtering

Application-level proxying
- Knows about higher-level semantics
- Long history for, e.g., email, now HTTP most important
- More knowledge allows better filtering decisions
  - But, more effort to set up
- Newer: “transparent proxy”
  - Pretty much a man-in-the-middle

Tunneling
- Any data can be transmitted on any channel, if both sides agree
- E.g., encapsulate IP packets over SSH connection
  - Compare covert channels, steganography
- Powerful way to subvert firewall
  - Some legitimate uses
Tunneling example: HA2

Outline
More web risks
Confidentiality and privacy
Announcements intermission
More crypto protocols
More causes of crypto failure
Firewalls and NAT boxes
Intrusion detection systems

Basic idea: detect attacks
- The worst attacks are the ones you don’t even know about
- Best case: stop before damage occurs
  - Marketed as “prevention”
- Still good: prompt response
- Challenge: what is an attack?

Network and host-based IDSes
- Network IDS: watch packets similar to firewall
  - But don’t know what’s bad until you see it
  - More often implemented offline
- Host-based IDS: look for compromised process or user from within machine

Signature matching
- Signature is a pattern that matches known bad behavior
- Typically human-curated to ensure specificity
- See also: anti-virus scanners

Anomaly detection
- Learn pattern of normal behavior
- “Not normal” is a sign of a potential attack
- Has possibility of finding novel attacks
- Performance depends on normal behavior too

Recall: FPs and FNs
- False positive: detector goes off without real attack
- False negative: attack happens without detection
- Any detector design is a tradeoff between these (ROC curve)

Signature and anomaly weaknesses
- Signatures
  - Won’t exist for novel attacks
  - Often easy to attack around
- Anomaly detection
  - Hard to avoid false positives
  - Adversary can train over time
Base rate problems

- If the true incidence is small (low base rate), most positives will be false
  - Example: screening test for rare disease
  - Easy for false positives to overwhelm admins
  - E.g., 100 attacks out of 10 million packets, 0.01% FP rate
  - How many false alarms?

Adversarial challenges

- FP/FN statistics based on a fixed set of attacks
- But attackers won’t keep using techniques that are detected
- Instead, will look for:
  - Existing attacks that are not detected
  - Minimal changes to attacks
  - Truly novel attacks

Wagner and Soto mimicry attack

- Host-based IDS based on sequence of syscalls
- Compute $A \cap M$, where:
  - $A$ models allowed sequences
  - $M$ models sequences achieving attacker’s goals
- Further techniques required:
  - Many syscalls made into NOPs
  - Replacement subsequences with similar effect

Next time

- Malware and network denial of service