# ECE 435 – Network Engineering
# Lecture 6

Vince Weaver

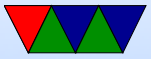
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

3 February 2022

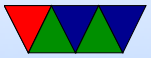
# Announcements

- HW#3 will be posted. Encryption.
- HW#2 was extended to Friday

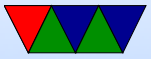


# HTML/3 in news

- https://hacks.mozilla.org/2022/02/retrospective-and-technical-details-on-the-recent-firefox-outag
- Firefox stopped responding worldwide because of a bug in their HTTP/3 stack made their telemetry break a few weeks ago
- The fact that they let the telemetry break the browser is a whole other concerning tale
- But it turns out recent firefox has HTTP/3 set to automatic, and will use it if found, and google has been rolling out HTTP/3

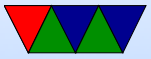


- Part of the bug is http headers are supposed to be case-insensitive, and HTTP/2, HTTP/3 suggests they should be all lowercase, which can break your parser if you don't expect it



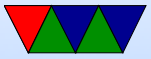
# HW#2 Notes

- There are a lot of issues people are having with C. I didn't explain things as well as I could because HW#1 had gone relatively well
- Don't use `strlen()` to find the size of something that's not explicitly a string. It stops at the first 0 (NUL), and possibly runs off the end if no 0 is found. For tracking `malloc()`d areas it's often best to have a separate integer where you track the current size.
- While the "right" thing to do with code like this is to

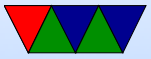


`malloc()` and auto-grow the input buffer in the face of unknown header size, in practice this is tricky to do in C. I'd prefer you just start with a really large buffer size (32k or more?) and error out if the input is too big, rather than wasting a lot of time chasing malloc/calloc/free pointer errors

- When you use a char pointer to point into a string (as when using `strstr()` or `strtok()` remember what you have is a pointer, not a copy of the string you're pointing to. So if the buffer gets freed or re-used your pointer may suddenly point to something different.

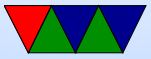


- You can use `sprintf()` or similar to generate http headers. Just be careful with the size of the buffer (maybe use `snprintf()` instead. Also try not to be too fancy with functions in arguments, as C is allowed to evaluate function arguments in arbitrary order.
- The biggest cause of problem is Content-length: header not being right. This value is the size of the data you are sending (after the header), it does not include header size.
- Instead of reading the file from disk into a buffer, then sending it all at once with the header, it makes more



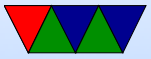
sense to write the headers to the socket, then have a loop to read/write the file out in chunks separately. This avoids needing to allocate large amounts of memory just to read in then write out.

- Also don't use sprintf() to print the file contents. It might work for html, but won't work for binary data (like images) that might contain a 0.
- If you are using `malloc()` and friends, a useful free tool for finding bugs is Valgrind.
- You can use the `wget -s` command on Linux to see what headers are being sent by your server

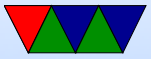


# Asymmetric / Public Key Encryption

- Asymmetric/Public Key
- Key is weakest link of symmetric encryption, as both sides have it and if anyone leaks it, all is lost
- Have a public key that anyone can use to encrypt a message. Can only be (easily) decrypted by a secret, private key
- Hard to solve math problems. Integer factorization, discrete logarithm, elliptic curves
- Often only used to encrypt small amounts of data,

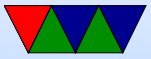


i.e. used to encrypt a symmetric key used for longer transactions

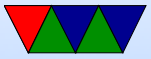


# RSA

- Rivest/Shamir/Adleman at MIT
- Choose two large primes p and q (1024+ bits)
- Compute: n=p*q, z=(p-1)*(q-1)
- Choose number relatively prime to z: d
  (no common factors)
- Find e such that e*d mod z=1
- Divide plaintext into blocks $0 \leq P < n$, blocks of k bits where k largest $2^k < n$
- To encrypt, compute $C = P^e \; mod \; n$

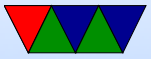


- To decrypt, compute $P = C^d \ mod \ n$
- public key is e,n. private key is d,n
- Hard to break as you need to factor n (hard)
- How do you find p and q? Random number, then apply various tests to determine if prime



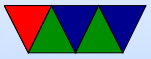
# RSA Example

- Example from Tanenbaum Figure 8-17:
  Pick two large primes: p=3, q=11
  n=p*q=33, z=(p-1)*(q-1)=20
  d=7 (no common factors with 20)
  $7 * e \; mod \; 20 = 1$ so e=3
  To encrypt say "13", $13^3 = 2197, mod 33 = 19$
  To decrypt say "19", $19^7 = 893871739 mod 33 = 13$



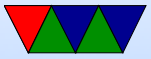
# Other Algorithms

- Prime Number Factoring
- Elliptic Curve Cryptography (ECC)
  Smaller keysize



# Uses of Public Key Crypto

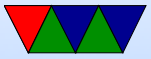
- public key encryption, public key used to encrypt message only holder of private key can decrypt
- digital signature: message signed with private key and anyone with access to public key can verify the original sender



# Cryptographic Hash Functions

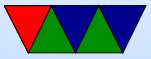
- Maps a document of arbitrary size to a fixed size
- Easy to calculate, hard to reverse. Only real feasible way to reverse is brute-force search
- Should not be able to find two different messages with same hash
- Small changes in document should lead to very different hashes
- Two items with same hash are a collision

  Are collisions useful? If you can map documents of



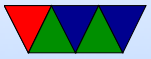
same filetype, or if somehow same document with lots of garbage on end

- Break file up into chunks, do a series of operations to "compress" it, often shift, xor, or, add, and, not



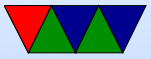
# Cryptographic Hash Algorithms

- md5 `md5sum`
  128-bit md5 hashes, create checksum, uniquely ID file
  Well, not really unique. It's been broken, can find (with great difficulty) collisions
- SHA-1
  Developed by NSA
  Used by git
  Broken since 2017
- SHA-2, SHA-3



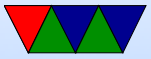
# Cryptographic Hash Uses

- passwords (/etc/shadow)
- (mostly) uniquely identifying a file (git),
- verifying file contents (download, error checking),
- bitcoin?



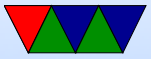
# Proof of Concept —— GTFO

- Has fun generating collisions



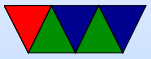
# Other Encryption Concerns

- Redundancy, some way to validate plaintext is valid. Example: if encrypting a binary blob where each byte indicates something ( 12 34 means order 34 cows or something), random garbage might decode to valid message
- Freshness – replay attacks. What if you record old message (Bank deposits $100 to account) and replay. Will have valid encryption.

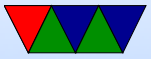


# Encryption Problems

- Keys leaked (DVD/game console issues)
- poor random numbers used (Debian problem)
- differential cryptanalysis (start with similar plaintexts and see what patterns occur in output) [DES IBM/NSA story]
- Power/Timing analysis – note power usage or timing/cache/cycles when encryption going on, can leak info on key or algorithm
  Bane of perf

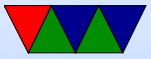


- Quantum computers



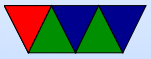
# Trusting Trust

- When setting up an encrypted connection, how do you verify who is on the other side?
- How can you protect from man-in-the-middle attacks (MitM) where someone intercepts them downloading your public key, replaces with their own, then sits in the middle decrypting/re-encrypting in a transparent way?
- Some companies/countries will actually do this quite openly



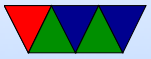
# Key Signing Parties

- One way is to have get-togethers were friends sign each others keys
- If enough people do this, you can create a "chain of trust" where you can track someone's identity to someone you trust
- Linux kernel sorta tries this for git development
- Trouble for new people, or remote people, or people who don't travel much, or don't have many friends



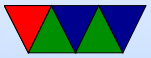
# Certificate Authorities

- Certificate authority – an official organization that verifies identities
- Will sign a "certificate" saying who you say you are
- Operating Systems/Web-browsers will ship with a list of officially trusted Certificate Authorities
- Can hover over the lock symbol in URL bar to verify who signed for a website
- Hashed?
- Can be revoked



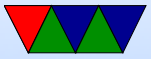
# SSL/TLS

- Secure Socket Layer / Transport Layer Security
- Handshake protocol followed by key exchange
- Browser says hello, which hashes/algorithms it supports
- Server picks one and sends back
- Server then sends a certificate (signed by authority) saying who it is, and what its public key is
- Client verifies certificate (via the CA public key it has stored)
- client generates a random number, encrypts with servers



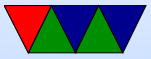
public key, sends to server, used as symmetric key

- What could go wrong, what if someone gets a hold of server private key? could decrypt logged data.
- Could try Diffie-Hellman key exchange – random number plus unique session key prevents problems if server private key leaked



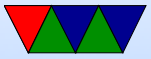
# Diffie-Hellman (used by ssh)

- Both sides agree on large prime number
- Both sides agree on algorithm (AES?)
- Each side picks independently picks another secret prime number.
  This is not the authentication private key.
- The secret prime, AES, and shared prime are used to make a public key derived from the private key.
- The generated public key is shared
- The other side uses their own private key, the other side



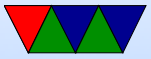
public key, and shared prime to figure out the shared secret key.

- This secret key is then used for symmetric encryption.
- Example on p812



# Other tools that use encryption

- How do you encrypt an e-mail, or a hard-drive, etc
- PGP – pretty good privacy
  OpenPGP RFC 4880
  Encrypt message with symmetric key, send along the key encrypted via asymmetric
  was illegal for a while (more than 40 bit encryption an exportable munition)
  people got RSA algorithm in perl tattoos
- GPG – free software replacement for PGP



- Can also PGP sign a message. Not encrypted, but signed with your key to verify it was in fact sent by you. Takes hash of the input, then encrypts the hash with key. Also, downloads from servers (like debian)