# ECE 435 – Network Engineering
# Lecture 4

Vince Weaver

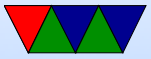
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

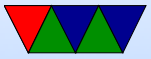
13 September 2018

# Announcements

- HW#1 was due.
- HW#2 will be posted. Write a mini webserver.
- Cybersecurity club Linux Security Lab on Saturday

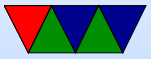


# WWW wrapup



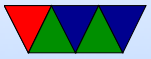
# What if Server Overloaded?

- Slashdot effect (modern: HackerNews?)
- caching/proxy – squid
- Content Delivery Network – akami
- Server farms



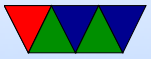
# Security

- SSL – Secure Socket Layer
- Replaced by TLS (Transport Layer Security)
- Port 443 for https
- Public key encryption.



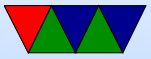
# Setting Up a Web-server

- Apache
- Easy to do, more difficult to secure

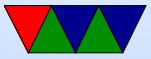


# Web Seach

- Web-bots index the web. robots.txt file
- Altavista, Hotbot, Excite, Inktomi, etc.
- Curated search like Yahoo (people organize links rather than automatically search)
- Google (1996 some machine in Stanford, 1997-1998)
- MSN search 1999, rebranded Microsoft Bing 2009

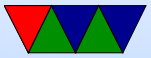


# Remote Connections



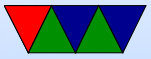
# telnet/rlogin/rsh/ssh

- telnet – login to remote system (tcp port 23) everything (including passwords) sent in plain text
- rsh/rlogin – remote shell, remote login. (tcp port 514) Didn't even need password, could configure to let you run commands on remote machine. Security based if you had same username on both machines, assumption was getting root on a UNIX machine and connected to Ethernet was expensive/difficult



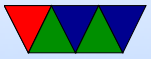
# SSH secure shell

- tcp port 22
- can login, run commands, tunnel tcp/ip, tunnel X11, file transfer (scp, sftp)
- Large number of RFCs
- Version 1: 1995, originally freeware but became private
- Version 2: 2005, openBSD based on last free version
- For security reasons there's a push to drop Version 1
- uses public-key cryptography
- transport layer: arranges initial key exchange, server



authentication, key re-exchange

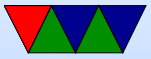
- user authentication layer: can have password, or can set up keys to allow passwordless, DSA or RSA key pairs
- connection layer: set up channels
- lots of encryption types supported, old ones being obsoleted as found wanting
- Various ssh servers/clients. openssh. dropbear
- Diffie-Helman key exchange?
  - This is a public/private key thing
  - Based on discrete logarithms?
  - Wikipedia has a weird colored paint analogy



# ssh security

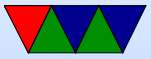
Brute forcing passwords is a major issue.

- Fail2ban
- Nonstandard port
- Port knocking
- Call asterisk for one-time pin?
- No-password (key only)
- Two-factor authentication (LCD keyfob)



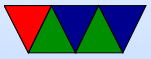
# Alternatives to SSH?

- mosh

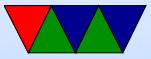


# Encryption

- Most crypto papers involve Alice and Bob (maybe Eve)
- **Plaintext** is transformed by some sort of function parameterized by a "key" into **Ciphertext**.
  This is then transmitted. The other side then decrypts it.
- What can be kept secret? Security by obscurity? Kerckhoff's principle: "All algorithms must be public; only the keys are secret."
- Combination lock analogy. Longer the key, the harder it

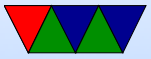


is to brute-force



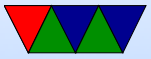
# Encryption Types

- easy: rot13
  Substitution cipher. Weakness: English text easy to predict ('e' most common letter)
  What about double-rot13?
- transposition cipher, keep letters same, re-arrange order
- hard: one-time-pad
  unbreakable. Downside, must keep it, must have enough bits, cannot reuse, transporting.



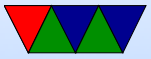
# Secret Key Algorithm

- Key is secret
- How do you get it to the other person?
- How many keys do you need (ideally one per connection)



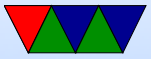
# Symmetric Key Algorithms

- Use same key for encryption and decryption
- Block ciphers, take block of data and encrypt it to same size block (why in blocks?)
- P-box (permutation), S-box (substitution)
- shift/permute/xor
- *very* important that the key is picked randomly.
- DES – Data Encryption Standard
  From 1976. 64 bit key (56-bits used). NSA had say on key size. 19 stages based on Key. widely used until



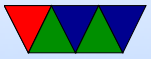
broken. Competition to break various sizes.

- 3DES (running DES three times) [encrypt/decrypt/encrypt with only two keys? Why? 112 bits seen as enough, also if set keys to same then it's same as single-DES (back compat)]
- AES – Advanced Encryption Standard – replaces DES
  NIST had a contest to find new standard
  Rijndael won. Intel chips have AES instructions
  Galois Field Theory



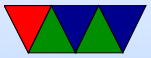
# Assymetric / Public Key Encryption

- Assymetric/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,

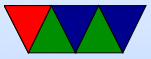


i.e. used to encrypt a symmetric key used for longer transactions

- 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 $C = P^e \; mod \; n$



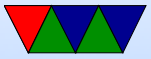
- To decrypt, compute $P = C^d \; mod \; n$
- public key is e,n. private key is d,n
- Hard to break as you need to factor n (hard)
- How do you find p and q? Random number, then apply various tests to determine if prime
- Example from Tanenbaum Figure 8-17:
  Pick two large primes: p=3, q=11
  n=p*q=33, 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 Types
  - Prime Number Factoring
  - Elliptic Curve Cryptography (ECC)

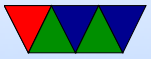
    Smaller keysize
- Common uses: public key encryption, public key used to encrypt message only holder of private key can decrypt digital signature: message signed with private key and anyone with access to public key can verify the original sender



# Cryptographic Hash Functions

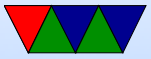
- Maps a document of arbitrary size to a fixed size
- Easy to calculate, hard to reverse. Only real feasible way to reverse is brute-force search
- Should not be able to find two different messages with same hash
- Small changes in document should lead to very different hashes
- Two items with same hash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

  Are collisions useful? If you can map documents of

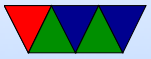


same filetype, or if somehow same document with lots of garbage on end

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

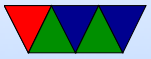


- Uses: passwords (/etc/shadow), (mostly) uniquely iding a file (git), verifying file contents (download, error checking), bitcoin?
- Problem: how do you verify the public key belongs to the person who they say it is? (on website? what if someone intercepts and replaces, mitm style)



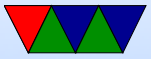
# Proof of Concept —— GTFO

- Has fun generating collisions

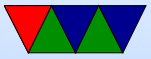


# Other tools that use encryption

- 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

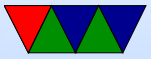


- Can also PGP sign a message. Not encrypted, but signed with your key to verify it was in fact sent by you. Takes hash of the input, then encrypts the hash with key. Also, downloads from servers (like debian)

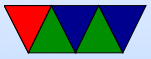


# Other Encryption Concerns

- 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.
- Block chain ciphers

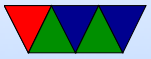


- Stream Ciphers



# Encryption Problems

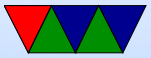
- Keys leaked (DVD/game console issues)
- poor random numbers used (Debian problem)
- differential cryptanalysis (start with similar plaintexts and see what patterns occur in output) [DES IBM/NSA story]
- Power/Timing analysis – note power usage or timing/cache/cycles when encryption going on, can leak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



# Certificate Authorities

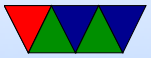
Problem: how do you verify the public key belongs to the person who they say it is? (on website? what if someone intercepts and replaces, mitm style)

- Certificate authorities
- Signed data block from official organization
- Hashed?
- Can be revoked
- Digital Signature Algorithm



# 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. Diffie-Hellman key exchange – random number plus unique session key prevents problems if server private key leaked