ECE 435 – Network Engineering
Lecture 21

Vince Weaver

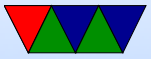
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

21 November 2016

Announcements

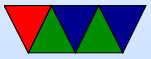
- HW#9 was posted Friday but Mainestreet was down
- Project status report
- Remember no class Wednesday, Thanksgiving



HW#8 Review

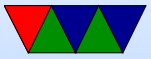
1.
```
0x0022:  bda5 ____________________ Source port (48549)
0x0024:  0050 ____________________ Destination port (80)
0x0026:  cdc4 6a49 _____________ Sequence Number
0x002a:  3c7b 6ca5 _____________ Acknowledgement Number
0x002e:  80 ______________________ 1000 header length = 8*4=32
0x002f:  18 ______________________ 11000 ACK+PSH
0x0030:  00e5 ____________________ Window Size = 229
0x0032:  79f4 ____________________ Checksum = 0x79f4
0x0034:  0000 ____________________ Urgent = ?
0x0036:  01       _Option: NOP (padding)
0x0037:  01        _Option: NOP (padding)
0x0038:  080a      _Option: Timestamp, 10 bytes
0x003a:  0104 3e58 _Timstamp TSval
0x003e:  34a8 7bc3 _Timestamp TSecr Echo Reply
TCP_PART
0x0042:  4745 540a _____________ GET\n
```

2. server=http, *total* frame size= $4/70 = 5.7\%$



3. packet summaries. Can see timestamps set. why? advanced flow control?
   a) 3-way handshake [s] is syn
   b) hi sent, acked, sends HI back, acked.
   Why not piggy back ack on reply? took too long? als P set?
   c) closing down. F (FIN) set not really a handshake

4. Whois. maine.edu setup: December 1988
   actually asked for umaine.edu which was 1997. registrar is EDUCAUSE



5. dig

   130.111.218.23

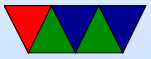
   typo in example, weaver has no IPv6. maine.edu

   2610:48:100:821::15

   SOA not same as AAAA

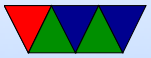
   nameo.unet.maine.edu / namep.unet.maine.edu

   ALT1.ASPMX.L.GOOGLE.COM



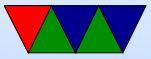
Web Seach

- 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



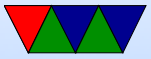
telnet/rlogin/rsh/ssh

- 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



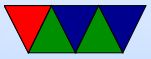
SSH secure shell

- tcp port 22
- can login, run commands, tunnel tcp/ip, tunnel X11, file transfer (scp, sftp)
- Large number of RFCs
- Version 1 released 1995, originally freeware but became private
- Version 2, openBSD people based on last free version (2005)
- For security reasons there's a push to drop Version 1

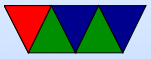


support

- 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? Based on discrete

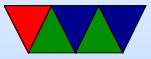


logarithms?



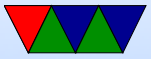
encryption

- Plaintext is transformed by some sort of function parameterized by a "key" into cyphertext. 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
- easy: rot13



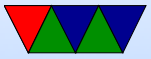
Substitution cipher. Weakness: English text easy to predict ('e' most common letter)

- transposition cipher, keep letters same, re-arrange order
- hard: one-time-pad
  unbreakable. Downside, must keep it, must have enough bits, cannot reuse, transporting.

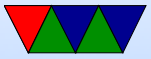


Symmetric Key Algorithms

- Use same key for encryption and decryption
- Block ciphers, take block of data and encrypt it to same size block
- P-box, S-box
- shift/permute/xor
- *very* important that the key is picked randomly.
- DES – Data Encryption Standard
  From 1976. 64 bit key (56-bits used) widely used until broken. Competition to break various sizes.

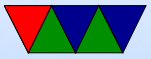


- 3DES (running DES three times) [encrypt/decrypt/encrypt with only two keys? Why? 112 bits seen as enough, also if set keys to same then it's same as single-DES (back compat)]
- AES – Advanced Encryption Standard – replaces DES
  NIST had a contest to find new standard
  Rijndael won. Intel chips have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



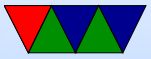
Public Key Encryption

- Assymetric/Public Key
- Encryption key 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,



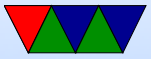
i.e. used to encrypt a symmetric key used for longer transactions

- RSA – Rivest/Shamir/Adleman at MIT
  - Choose two large primes p and q (1024+ bits)
  - n=p*q, z=(p-1)*(q-1)
  - Choose number relatively prime to z: d
  - Find e stuch that e*d=1 mod z
  - Divide plaintest into blocks 0 le P lt n, blocks of k bits where k largest $2^k < n$
  - To encrypt, compute $C = P^e mod n$
  - To decrypt, compute $C = C^d mod n$



- 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 8-17:
  p=3, q=11, n=33, z=20. d=7 (no common factors with 20)
  e is $7e = 1 mod 20$ so e=3
  To encrypt say "13", $13^3 = 2197, mod 33 = 19$
  To decrypt say "19", $19^7 = 893871739 mod 33 = 13$

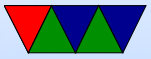
- Other Types



- Prime Number Factoring
- Elliptic Curve Cryptography (ECC)

  Smaller keysize

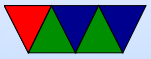
- Common uses: public key encryption, public key used to encrypt message only holder of private key can decrypt digital signature: message signed with private key and anyone with access to public key can verify the original sender



Cryptographic Hash Functions

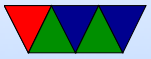
- Maps a document of arbitrary size to a fixed size
- Easy to calculate, hard to reverse. Only real feasible way to reverse is brute-force search
- Should not be able to find two different messages with same hash
- Small changes in document should lead to very different hashes
- Two items with same hash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

  Are collisions useful? If you can map documents of

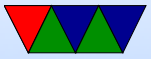


same filetype, or if somehow same document with lots of garbage on end

- 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

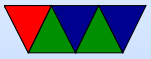


- Uses: passwords (/etc/shadow), (mostly) uniquely iding a file (git), verifying file contents (download, error checking), bitcoin?
- Problem: how do you verify the public key belongs to the person who they say it is? (on website? what if someone intercepts and replaces, mitm style)



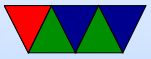
Certificate Authorities

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

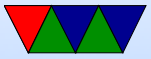


Tools that use encryption

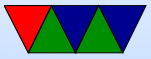
- PGP – pretty good privacy
  Encrypt message with symmetric key, send along the key encrypted via assymetric
  was illegal for a while (more than 40 bit encryption an exportable munition)
  people got RSA algorithm in perl tattoos
- GPG – free software replacement for PGP
- SSH
- SSL/TLS:



Browser says 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
client generates a random number, encrypts with public key, sends to server, used as symettric 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

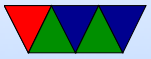


key leaked



ssh security

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



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