CS161 Final Cheat Sheet

Denial-of-Service (DOS):

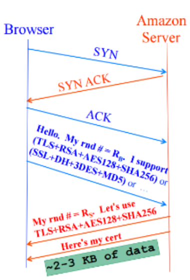
- We had Confidentiality, Integrity, and Authentication, but now we also have **Availability**: we can access our data/use communication
- This is an attack on availability, with a super large attack surface.
- Two basic attacks on availability:
  - Deny via a **program flaw** ("*NULL") – input crashes server
  - Deny via **resource exhaustion** ("while true")
- **Amplification** is when the attacker leverages system's own structure to pump up the load they induce on a resource (ex. DNS lookups)
- In TCP SYN flooding, the attacker targets memory – server creates some state for TCP SYN packet which requires a lot of work
  - Defenses: get tons of memory or identify bad actors
  - Idealized: don't keep state, but TCP protocol isn't helpful Practice defense: SYN Cookies: encode critical state entirely within SYN-ACK's sequence number. Create state on ACK
  - Spread service across lots of different physical servers
- Application layer DOS: overwhelm a service's processing capacity

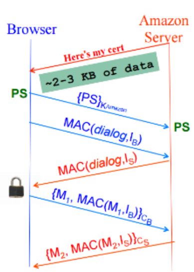
Securing Internet Communications:

- **Channel security** = securing a means of communication
- **Object security** = securing data values
- End to end = communication protections achieved throughout

TLS:

- **Confidentiality:YES**,**Integrity:YES**,**Authentication:YES**,**Availability:NO**
- Provides **CHANNEL SECURITY** for communication over TCP
- After 3 way handshake (RSA):
  1. Client sends: R_B (rand #), supported ciphersuite
  2. Server sends: R_S (rand #), selected ciphersuite & cert
  3. Client validates cert. Constructs 368 bit Premaster Secret



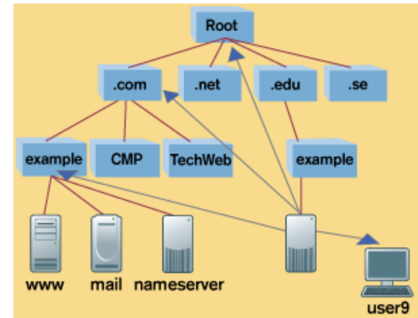


  4. Client sends PS encrypted with server's RSA key
  5. Client and server construct cipher keys C_B, C_S and MAC integrity keys I_B, I_S, one pair to use in each direction
  6. Client and server exchange MACs computed over dialog

- Using Diffie-Hellman, in step 3 and 4, the PS is generated through Diffie-Hellman: server sends g, p, g^a mod p signed and client sends g^b
- Relies on you trusting the CA (that issued the certificate)
- **SSLSTRIP** attack is MITM in which client goes to https, but attacker forces http and tunnels data from http <-> self <-> https

Securing DNS Lookups (DNSSEC):

- **Confidentiality:NO**,**Integrity:YES**,**Authentication:YES**,**Availability:NO**
- Provides **OBJECT SECURITY** for DNS results
- We could run DNS over TLS, but performance and caching issues
- Use DNSSEC. Computes hash of next key and signature of record
- No signature over the NS or A information. Only sign the keys
- Only at top level domains, is there a signature over the message
- Having separate key signing keys vs. zone signing keys allows a zone to change its zone signing key without needing parent to resign
- Issues: replies are big; 69 bytes query -> 3,419 byte reply. Not everyone is using it; how should we deal with unsigned responses. Negative results is a headache (homework 5).



Root signs the .com, .net, .edu, and other TLD (top-level domain) records. The .com domain then signs all of its delegated zones. Example signs records for hosts under it. When user9 tries to resolve *www.example.com*, its nameserver walks down the DNS tree as normal, but also validates responses at each step. If no problems are detected, nameserver sends a response to user9. Optionally, user9 could request all DNSSec records and do its own validation.

Detecting Attacks:

- This is the alternative to an air-tight system (cost is a factor)
- NIDS look at network traffic: scan HTTP requests for malicious activity
  - Good: Bolt on security, cheap and centralized management
  - Bad: Doesn't work with HTTPS and hard to get all activity
- HIDS scan arguments for backend programs instead
  - Good: Works for HTTPS and no tricky strings finding
  - Bad: Have to add code to each web server + Unix semantics
- Logging Attacks and analyzing logs
  - Good: Cheap and no problems with %-escapes or HTTPS
  - Bad: Delayed detection, modified logs, filename tricks
- Monitor system call to backend process (look for access to password)
  - Good: No issues with filename tricks, no false negatives
  - Bad: Lots of data, lots of false positives for legit access
- Measure with precision=P(I|A) or recall=P(A|I) (A=alert, I=Intrusion)

- Bayes rule: P(A|B)=(P(B|A)*P(A))/P(B), Recall = 1 – P(FN)

Styles of Detecting Attacks:

- **Signature based** look for activity that matches a known attack
  - Good: Conceptually simple, easy to share signatures
  - Bad: Blind to novel attacks and misses variants, many FP
- **Vulnerability Signatures** try to match on known problems (> x byes)
  - Good: detects variants of known attacks and concise
  - Bad: Can't detect new vulnerabilities, hard to write/express
- **Anomaly-Based** try to look for peculiar behavior (attacks are weird)
  - Good: Wide range of novel attacks are detected
  - Bad: Doesn't detect non-peculiar attacks and high FP rate
- **Specifications Based** says to specify what's allowed
  - Good: Detect novel attacks, low FP rate
  - Bad: expensive to derive and update specifications
- **Behavioral** look for evidence of compromise (unset HISTFILE)
  - Good: 0 FP rate, cheap, detects range of novel attacks
  - Bad: Brittle, easy to evade, post facto detection
- **HoneyPots** deploy a sacrificial system that has no operational use
  - Good: Identifies/tracks, studies and diverts intruders
  - Bad: A lot of work to build a convincing environment

Malware:

- How it runs: vulnerable service/client, social engineering, local access
- **Virus**: code propagates across systems by executing (stored code)
- **Worm**: code self-propagates executing immediately (running code)
- **Rootkits** are kernel patches to hide its presence
- **Zero day** are exploits that are previously unknown

Viruses:

- Looks for opportunities to infect additional systems.
- Signature based detection looks for bytes corresponding to injected virus code. AV companies market on number of signatures
- **Polymorphic code**: Viruses can encrypted themselves to look different each propagation, new key but same decryptor. Detect by targeting decryptor or see execution patterns
- **Metamorphic code**: Generate semantically difference versions each propagation. Code rewriter can renumber registers, change conditionals, reorder operations, replace algorithms with another, padding, etc. AVs analyze virus to find behavioral signatures

Worms:

- Spreads much quicker because they parallelize propagation
  - Ex. XSS worms run when viewed (Squigler)
- Growth formula: $i(t) = e^{Bt}/(1+e^{Bt})$ where i(t) = proportion of infected hosts at time t and B = contact rate.
- Growth of worm spread levels out once carrying capacity is reached

Botnets:

- Collection of machines (**bots**) under control of a **botmaster**
- To fight bots/botnets, we can take down the master server or prevent the initial bot infection (this is hard).
- Botmaster counter measures include moving master server around or to buy of the ISP (bullet proof hosting).
- We can size the domain name, but they can buy bullet-proof domains
- Botmaster could also pay to get their badness installed on users

Special Topics:

- Attackers can monitor or affect physical interactions leaking useful side channel information, aptly called **side channel attacks**
  - Overused pin pad keys, exploiting page fault timings, checking power usage, cache hits, data remanence, much more
- 3 classes of attacks on password authentication: online guessing, server compromise (offline) and client password compromise
- Secure password storage req: salt + security hashed password
- 2 factor authentication to prevent password compromise
- 2 approaches to searching encrypted data:
- use **specialized cryptography** to compute directly on encrypted data
  - **Property preserving encryption**: ex. order preserving, if x > y then E(x) > E(y) **Performance: fast**, **Security: leaks information**, **Functionality: limited use**
  - **Partially homomorphic encryption**: allows multiplication or addition over ciphertexts, **Performance: efficient**, **Security: AES level confidentiality**, **Functionality: limited use**
  - **Fully homomorphic**: Enables arbitrary functions ex. F(E(x), E(y)) = E(F(x,y)). **Performance: prohibitively slow**, **Security: AES level good**, **Functionality: Allows arbitrary computations**
- use **specialized hardware** to use shielded computation (Intel SGX)
- Memory and other components can't snoop (enclaves, minimal TCB) and attestation property