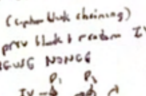
Matthew Tran
CS161
MT1

- Security - enforcing desired property even w/ attacker
  - ex. data confidentiality, privacy, integrity, availability etc (authentication)
  - Avoid attacks bc attacker does it anyway
    - Intimate Partner Threat (IPT)
  - users are biggest problem, but don't blame them!
  - cost benefit analysis both security & exploit
  - Prevention - not only thing
  - Detection and Response
    - False positive - cost money
    - False negative - Failure
  - TINSTAAFL (no free lunch)
    - defense in depth - layers
      - parallel - either => ↓ false neg ↑ false pos
      - serial - both => ↑ false neg ↓ false pos
  - no password reuse! password managers!
    - 2 factor, security key
  - economics
    - standards based
    - more people use it => cheaper as gets safe
  - least privilege
  - Trusted Computing Base (TCB)
    - what security relies upon? ↑ give privilege
    - correct
    - complete - no bypass KISS
    - secure - no tamper
  - Privilege Separation
    - OS process - isolation, permissions
    - ex. Chrome sandbox each page
  - Reference monitor
    - single point of access
    - unbypassable "complete mediation"
    - tamper proof } same for TCB
    - verifiable
    - ex race condition big problem
      - TOCTTOU - time to check to time of use
      - use after free
  - separation of responsibility
    - multiple parties have privileges, need to work together to use
  - NO security thru obscurity
    - Kerckhoffs's Principle - assume attacker knows all except key
    - Shannon's Maxim - enemy knows system
  - Trusted path - how to know correct person / server?
  - Fail safe - default deny or allow?
  - Common Assumptions
    - interact w/ system anywhere
    - know general info
    - can get system copy
    - automation 2^{30} is small, doable
    - complex coordination
    - large resources, but not like ISP
    - can get privileges
    - infrastructure well protected
    - network not hardened better, except physical
    - people hack for fun
    - dev gets caught
  - Patch - fix vulnerability
  - 0-day - only attacker knows, rare
- Buffer Overflows
  - Internet of Shit - weak attack surface, low cost
  - we use x86-32
    - EAX-EDX - gp registers
    - EBP - top of frame
    - ESP - grows down
    - call func
      - args on stack
      - CALL => RA onto stack (RIP)
      - save old EP on stack (SFP)
      - restore EBP, pop ESP & restore
      - RET - jump to RA & ESP

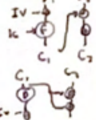
      (ret, rip, sfp)
  - printf dangerous
    - %n - writes num chars so far into arg
    - %s - print string
  - vtable ptr - tells where to jump to
  - heap overflow - can affect ↑
  - use-after-free - write new data in new block overwrite vtable
  - NOOP sled - in case address a little off

- Buffer Overflow Defense
  - C bc speed, but unsafe
    - better alternatives: Java, Python, JS, Go, Rust
    - but legacy!
  - 32b v. 64b
    - 2^{32} bits, 2^{12} pages
    - 2^{42} bits max usable, addr 22, 2^{28} pages
  - Non Executable Memory (W^X)
    - 0 overhead! just mark pages
    - counter: call existing func, like execv
    - counter: ROP chain, use fragments of code
      - ROP compiler exists
  - Stack Canary (RIP, SFP, canary)
    - random value below SFP
    - usually compiler flag!
    - small overhead, required
    - counter - find canary
      - write around (printf?)
      - attack heap
    - 1 byte always 0 bc null stop string copy
      - brute force! - counter
        - counter-counter - timeout!
  - Pointer Authentication (ARM)
    - use upper 22b of 64b for auth
    - PAC - calc code
    - AUT - check and remove
    - XAUT - strip PAC w/o checking
    - 0 overhead if supported! well 1 instruction
    - 5 secrets - 128b each
      - DA/DB, IA/IB, GA
    - counter - trick into calculating ret
      - reuse
    - basically cheap stack canary
    - PACIA, AUTIA, if backward compat
    - PACIA jump/trigger
    - protect vtable ptr too!
      - also vtable in read only
  - Address Layout Space Randomization (ASLR)
    - lib, stack, heap, etc
    - 0 overhead!
    - counter: leak info, 1 ptr enough to know lib loc
      - need R/W area, ROP chain
      - randomize every func? w/ IP, lots of overhead
  - JS engine [illegible]
    - compiles hot code, attack overwrite code w/ [illegible]
    - explain currently compilation
- Crypto
  - Goals
    - confidentiality - no reading private data
    - integrity - no altering
    - authentication - prove who created
  - Eve knows - any combo
    - ciphertext only - probs english
    - known plaintext - but might reuse
    - chosen plaintext - spoof
    - chosen ciphertext - fool Bob
  - IND-CPA (only chosen plaintext)
    - Eve give Oracle 2 text, can't have > 50% chance guess which one encrypted
    - basically can't tell 2 apart based on encrypted
  - One time Pad - XOR
    - key as long as message M, diff key
    - secure! but impractical...
  - Traffic Analysis - know who talks to who & when
  - Sidechannels - something which crypto gives info
  - Block Cipher - key K, E(M,K), D(C,K), known ideal
  - AES
    - 128 bit block size, 128, 192, 256 bit key
    - low CPU use, not proven secure
  - AES-ECB (electronic code book)
    - split M into blocks, encrypt each independently
    - not IND-CPA! bad!
  - AES-CBC (cipher block chaining)
    - idea: use prev block & random IV nonce
    - NEVER REUSE NONCE
    - encrypt



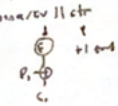
send IV, C_i's
sequential λ

    - decrypt



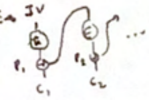
P_i needs C_i, C_{i-1}
in parallel!

- Padding to block size
  - usually just add how many bytes left
- AES-CTR
  - parallel encrypt / decrypt!
  - encrypt nonce/IV || ctr

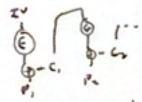


+1 each block

  - length same - swap C_1, P_1
    - note: only uses encryption!
  - stream cipher, no padding
  - NEVER USE if pwned
    - CBC reuse fine know first block
    - CTR fails MISERABLY
      - JUST LIKE ONE TIME PAD
      - NEVER REUSE IV
- AES-CFB (cipher feedback)
  - Enc



  - Decrypt



  - no pad bc XOR w/ plaintext
  - like CBC, little worse if reuse IV
- Integrity?
  - CTR can replace any part
- Hash Functions
  - variable input, fixed output
  - efficient, pseudorandom
  - one-way - can't find input given output
    - "preimage resistance"
  - 2nd ✓ => no x' given x H(x)=H(x')
  - collision resistance => can't find x,y ⇒ H(x)=H(y)
  - integrity if can send securely "fingerprint"
  - common
    - MD5 - broken
    - SHA1 - broken
    - SHA-2 ✓ ← vulnerable to length extension bc if len(x), H(x) ⇒ can H(x||m)
    - SHA-3 ✓
- Message Authentication Code (MAC)
  - symmetric key
  - to send {M,T}, T = MAC(K,M) like hash, but key
  - req
    - no $T^* = MAC(M^*, K)$
    - no $M^* \neq M$ ⇒ $MAC(M^*,K) = T$
  - NEVER REUSE KEY
    - MAC → encrypt $T = MAC(M,K_1)$ then $C = E(M||T, K_2)$
    - same time $C = E(M,K_1)$, $T = MAC(M,K_2)$, send C||T
  - ALWAYS encrypt then MAC
    - mistakes happen
  - HMAC - hash + mac, best!
    - if len(K) > block: key = H(key)
    - pad to blocksize w/ 0x00
    - o_key_pad = 0x5c... ⊕ k
    - i_key_pad = 0x36... ⊕ k
    - return H(o_key || H(i_key || m))
    - i_key and o_key need diff
    - 2nd hash prevent appending data
    - if mess low, probs safe BUT DON'T
  - AEAD - integrity + confidentiality } no reuse IV!
  - GCM (Galois counter mode)
- TRNG
  - mix entropy sources
  - seed a deterministic PRNG
- PRNG
  - rollback resistant - nice to have
    - next if can find state, back ones not
- HMAC-DRBG best!
  - need in lib
  - start k=v=0
  - update
    - k = hmac(k, v || 0x00 || input)
    - v = hmac(k, v)
    - k = hmac(k, v || 0x01 || input)
    - v = hmac(k, v)
  - generate

```
tmp = ""
while len(tmp) < N
    state.v = hmac(state.k, state.v)
    tmp = tmp || state.v
state.k = hmac(state.k, state.v || 0x00)
state.v = hmac(state.k, state.v)
return tmp[0:N]
```

- Stream Cipher
  - E(M,K) = PRNG(K, IV) ⊕ M
  - D(C,K) = PRNG(K, IV) ⊕ C
  - ctr → $2^{n/2}$ where n is seed length
  - CTR is like this
- UUID - normally random id
- Public Key Encryption
  - private, public
  - can sign things
  - much slower than symmetric
- Diffie-Hellman (DH) E for ephemeral
  - pick large p prime and 1 < g < p-1
  - Alice pick random secret a
    - Bob pick b
  - A send $A = g^a \mod p$
    - $B = g^b \mod p$ ($A^b \mod p$)
    - $K = B^a \mod p = g^{ba} \mod p$ = same (shared secret key)
    - $K = g^{ab} \mod p$ only receivers
  - vulnerable to MITM!
- RSA
  - 2 large primes p, q ⇒ n = p·q, φ(n) = (p-1)(q-1)
  - pick e $2 < e < \phi(n)$ ↪ relatively prime, can be small
    - & $d = e^{-1} \mod \phi(n)$
  - n, e public
    - $C = m^e \mod n$ encr
    - $m = c^d \mod n$ decr
  - not IND-CPA! need padding
  - RSA-OAEP (optimal asymmetric encryption padding)
    - like block cipher, also pads, also random
  - Timing attacks? can guess bits
  - RSA sign $S = (H(m))^d \mod n$
    - verify $V = S^e \mod n = H(m)$
- El-Gamal
  - like DH, pick p and g
  - Alice pick X as private key
    - pub $h = g^x \mod p$ = public key
  - Bob random y, $c_1 = g^y \mod p$
    - encr $s = h^y \mod p$
    - $c_2 = m' \times s \mod p$
    - send c_1, c_2
  - decr $s = c_1^x \mod p$
    - $m' = c_2 \cdot s^{-1} \mod p$
    - then map to m
  - can $m' \Rightarrow 2m'$ easy
    - why? $\frac{c_2}{c_2'} = \frac{m'}{m''}$
- DSA sig
  - L, N, hash func H, L = key length, N ≤ len(H)
  - N-bit prime q, L-bit p, (p-1) multiple of q
    - $g = h^{(p-1)/q} \mod p$ for h ∈ 1 < h < p-1
    - private key $y = g^x \mod p$, x private key
  - [illegible]

- Trusted Authorities
  - Digital Cert
  - hierarchy
  - Certificate Authority (CA) form PKI
  - Revocation
    - expiration date
    - announce revoke (CRL)
- Trust but verify
- Password Hashing
  - store H(P)
    - con: brute force tables
  - salt! H(P||salt) and salt
    - vuln to offline attack
  - slower hash!
  - PBKDF2 - lots "rounds" hashing from pass, also key
  - secure enclave!
- Blockchain
  - block has hash of previous block
  - Merkle Tree
  - public ledger
  - users, work to add to it
    - "proof of work"
    - gather unconfirmed transactions, add itself & hash of prev block
    - hash until start w/ enough 0s
      - 2^n hash for n 0s
  - inefficient, not easy transaction
  - power waste
  - trust? largest miners
  - irreversible! an issue
  - not really decentralized
- Signal
  - E2E messages
  - what if Bob offline? forward secrecy?
  - mutual authentication & deniability
  - keys
    - A: IK_A, EK_A - just for Bob
    - B: IK_B
      - SPK_B - medium lived, Sign(IK_B, SPK_B)
      - OPK_B - one time, can reuse? prevent replay
    - $DH1 = DH(IK_A, SPK_B)$
    - $DH2 = DH(EK_A, IK_B)$
    - $DH3 = DH(EK_A, SPK_B)$
    - $DH4 = DH(EK_A, OPK_B)$
    - $SK \leftarrow HKDF(DH1 \| DH2 \| DH3 \| DH4)$
      - hash key deriv
  - send AEAD $E(SK, M, IK_A \| IK_B)$
    - send IK_A, EK_A, OPK used?
  - trust server IK_A, IK_B
    - verify in person
    - plus voice channel
  - ratchet one way fn for keys

Tor
- onion routing
- good for local, not global!
- easy to spot
- looks same as all
- trust exit node ...
- hidden service, hash of public key
  - anonymous rendezvous point
  - 3 jumps

Matthew Tran
CS161
Final

- Command Injection
  - basically whenever insert user data into command string
  - input sanitization - remove bad bits, but hard to do right
  - input become "control information", not just "data"
  - key issue - command to call stuff too powerful, interprets string
    - ex. C system vs execve + bypass shell
  - fix - only offer safe version!
- SQL injection
  - user → website → database query
  - tables w/ rows of data w/ columns
  - relevant SQL
    - SELECT * FROM * WHERE *
      - also *, ', ", <>, AND, OR
    - INSERT INTO * VALUES (...)
    - DROP * - concatenate ||
  - ideas
    - close strings w/ '
    - add comment --
    - ; to separate commands
    - usually '<user input>'
  - Fix
    - sanitization - hard
    - prepared statement - good, but needs library, not native SQL
- Data Storage Methods
  - manual - define byte order, safe, but PITA
  - Java mistakes, Python pickle - simple, arbitrary, dangerous, can deserialize into anything
  - Google Protocol Buffers - efficient, safe, external compiler
  - XML/JSON - human read, less compact (compression)
    - use JSON when in doubt
- HTML - [illegible]
- HTTP - how to fetch docs
- URL - where is it
- URI
  - scheme - ftp, http, etc [illegible]
  - // - remote location
    <user>@<host>:<port>
  - path - starts w/ /
  - query - ? or %3F, usually key/value
  - fragment - #, not sent to remote
  - all printable ASCII
    - else %XX most
- HTTP
  - GET - no side effect, maybe
  - POST - possible side effect, attach data
  - HEAD
- JS
  - Java code in browser
  - uses JIT, convert to machine code, very dangerous bc W&X memory
  - use after free
- DOM (Document Object Model)
  - cross platform model for representing HTML
  - JS can modify anything in DOM, access cookies?
- Frames
  - embed page within page, isolated, only specify size and location
- Security goals
  - integrity - no tamper
  - confidentiality - no learn secrets
  - privacy - can't spy on activities
  - availability - yes access, no stop
- Web Risks
  - mod local files - JS sandboxed, browser privileges
  - tamper other sites - Same origin policy
  - unauthorized access - server side security
  - Same Origin Policy
    - origin: domain, protocol, port
    - browser enforced isolation
    - fetch anything, no details / properties, ex image blank
    - scripts run as origin, not other
  - Cross Origin Resource Sharing (CORS)
    - other origin can allow sharing
  - can change origin up, like [illegible] → a.com, not down
- Cookies
  - set by server or JS
  - usually for user authentication - allowed if suffix fully match, ex a.b.com can set for b.com
  - key value pair
  - hostname/path binding
    - sent if suffix fully match
  - can't know if 2 copies of cookies which one longer to send one/subdomain
  - can send over https only if needed
- Spectre - speculative execution to read memory
  - exploit browser running in same process
  - train branch predictor, access invalid, can leak it, timing
  - fix: all origin in unique process
- Cross-Site Scripting (XSS) ← bypass same origin of the web
  - stored XSS - script on victim server, sends it
  - reflected XSS - script from victim
  - DOM, cookies [illegible] - ex. user input put into HTML
- Cross-Site Request Forgery (CSRF/XSRF)
  - use JS to send data to other origin
    - ex Google Analytics
  - bad site can make user who visit their site send request w/ our cookie
    - basically bad URL
  - hiding attacks
    - frames! invisible
    - no click needed!?
    - still look normal site?
  - OWASP (Open Web App Security Project)
    - never insert untrusted data except allowed
    - escape before insert if allow untrusted attribute
      - escape &, <, >, ", ', /
      - use known sanitizer
- Content Security Policy
  - allowlist for where to load things
  - in HTTP header
  - browser enforced
- Clickjacking
  - transparent iframe somewhere else
  - fix
    - confirmation action
    - framebusting - don't allow framed
    - HTTP X-Frame-Options - whitelist frameable sites
- Phishing
  - links can look like another site,
  - chinese chars look like /.?=
  - Two-Factor - second comms channel
    - transient pairing - immediate login
    - [illegible] - HMAC(key, time)
  - CAPTCHA
    - make bad guys do work for us AI train
    - reverse Turing test - easy human, hard comp
    - issue - too hard, adversarial web crawlers
      - outsourcing attacks
    - accessibility - DSP
    - granularity - not all bots bad
    - ambiguity - modern [illegible]
- Network Security
  - protocol - agreement on how to talk
  - layering - OSI model
    - 8 - people/politics
    - 7 - application
    - 4 - transport
    - 3 - network — each layer adds header, can use first
    - 2 - link
    - 1 - physical
  - packets - [illegible], unreliable
  - diff per hop { - physical - raw bits; - link - send group of bits, message
  - network - IP, global addr,
  - transport - TCP, UDP
  - crypto - "layer 4.5"
  - IPv4
    - length, TTL, protocol, src, dst
  - Ethernet - link layer
    - preamble
    - src, dst MAC, VLAN, type
    - MAC - 3 byte manufacturer, 3 byte ID
    - should ignore non-match, but promiscuous mode
  - Subnet /8, /16, /24
    - # bits in network (upper bits)
    - same dedicated for [illegible] private
  - Link layer threats
    - eavesdropping / sniffing
      - broadcast bc like WiFi easy to sniff
      - tcpdump, Wireshark, [illegible]
      - can mirror a port
      - [illegible]
  - Attacker Types
    - off path - no sniffing
    - on path - see, all packets, no block
    - in path - MITM, also block, replace, [illegible]
  - ARP - match IP → MAC
    - needed if same subnet
    - if not, just use gateway
    - broadcast protocol
    - easy to spoof, just respond faster
      - take place as gateway
  - Packet injection
    - broadcast is dangerous
    - race condition attacks
    - always use powerful WEEE
    - fix
      - will see both packets
      - late not used
  - WPA2 PSK — PTK: PMK, Anonce, Snonce, MAC addrs
    - 4 way handshake → Anonce, Snonce
      - offline attack possible MIC (HMAC)
      - rogue AP attack
    - client ↔ AP: Anonce, Snonce+MIC, GTK+MIC, Ack → PTK, share key
  - WPA enterprise
    - actually secure
    - authentication server,
- KRACK
  - only send packet number instead of full IV
  - usually CBC-MAC, encrypt w/ CTR
  - GCM - higher performance
    - reuse IV, forge communication!
- WPA handshake replay attack
  - 3rd packet - reinstall key, IV
  - IV reuse, decrypt in CCM, forge in GCM
- Dragonfly
  - force online attack only
  - DH w/ EC
  - used in WPA3 w/ EC
  - 2 extra handshakes
- DHCP - like ARP, get an IP addr
  - IP, DNS, gateway, location
  - easy spoof
  - fix
    - Ethernet switch
      - not like hubs
      - keep track of MAC address
      - broadcast if no cache entry
    - idea - constrain broadcast domain
      - filter request, reply
    - VLAN
      - ~~[illegible]~~
      - diff trust domains
      - switch enforces VLANs on ports
      - trunk - multiple VLANs
- Autonomous System (IP routing)
  - IP packet → gateway → AS → dst
  - BGP protocol - how AS know where to go
  - 25% of packets spoofing
    - bad - ex. reflected DoS
- UDP - datagram
  - any order, fragmentable, but [illegible]
- DNS - resolve
  - site → IP addr, caching helps a lot
  - [illegible] trust
  - UDP based
  - Protocol - important bits
    - ID # 16 bit
    - question, answer ← glue record
    - authority, additional
    - RRs have TTL for caching
  - cache poisoning
    - send diff IP for diff domain
    - easy fix, only cache same domain
  - type ← RR: name, type, TTL, record
    - A: IPv4
    - AAAA: IPv6
    - CNAME: alias
    - MX: mailserver
    - NS: authoritative server for DNS
    - SOA: operator
  - blind spoofing
    - guess ID w/ another request
  - Kaminsky attack
    - DNS caches, steal priority [illegible]
    - no attack more diff sub
      - a.goo.com, b.com
    - only need one success
    - fix
      - don't blindly accept [illegible]
      - only use the fix - DNS lookup randomization
      - validate w/ another fetch
- TCP
  - bidirectional byte stream, in order
  - 5-tuple (src ip, port, dst ip, port, TCP or UDP)
  - not same response { - sequence #, doesn't start at 0
  - ack - # done recvd, SYN, → SYN
    - next expected byte
  - Flags
    - SYN - start connection
    - ACK
    - FIN - no send, can still receive
    - RST - just stop
  - 3 way handshake - SYN, SYN-ACK, ACK (x, y, x+1, y+1)
  - RST only accepted if correct seq #
  - hijacking
    - inject packets, later ones ignored bc processed
  - RST injection - close the it
  - Blind hijacking - guess port, sequence #
  - Blind spoofing - guess init sequence #
    - [illegible] - attacker doesn't see packets, packet ISN
  - SYN Flood DoS attack
    - lots of mem per TCP
    - spoof src IP
  - fix
    - wait until ACK
    - pseudorandom ISN and check
    - HMAC?
- TLS (or DTLS for UDP) — limits attacks to DoS
  - SSL - predecessor
  - HTTPS
    - TCP connection first
    - plaintext { - client sends 256-bit random R_B + supported ciphers; - server picks cipher, sends 256-bit random R_S + cert
    - use RSA, client send secret PS w/ public key server's
    - also can DH, server signs randoms → use PS, R_B, R_S, get symmetric cipher keys C_B, C_S + server MAC integrity I_B, I_S — one pair per direction
    - both send MAC over entire dialog so far
  - Seq # == no replay
  - R_B, R_S == no handshake replay
  - depends on PRNG to be all bad, restart
  - TLS 1.3 - ECDHE/DH only bc RSA compromised is very bad
    - if that, not to be MITM
- PKI
  - require trust CA
  - revocation of cert
    - timestamp - but wait years...
    - cert revocation list
    - OSCP - online status
  - browsers hardwired w/ trusted CA
- DOS
  - hard to block completely, esp w/ spoofing
  - amplification - ex. use DNS amp
  - botnets
  - [illegible] connections w/ hash table
  - all firewall
    - default deny better
- NAT - network address translation
  - router has one to many, [illegible] table of stateless
- DNSSEC
  - recursive resolver big issue bc often bad, in path
  - idea: DNS is PKI and has certs
  - group records of same name/type into RRSETS
  - new type OPT — signal DNSSEC and [illegible]
  - 2 new flags DO, CD ← don't check DNSSEC
  - I can DNSSEC — add the EDNS0
    - CLASS field to give UDP size
    - use TTL to add 16 bit flags
  - certs → new type RRSIG
    - signature over RRSET, DS
  - DNSKEY record type
    - key info,
    - public key — Key Signing Key (KSK) - sign DNSKEY RRSET
    - Zone Signing Key (ZSK) - sign everything else
    - client hardwired w/ root's KSK
    - also RRSIG signed w/ KSK
  - DS record hash of KSK
    - key fingerprints — delegated signing, parent signs records for child keys
    - RRSIG signed w/ ZSK
  - ex. for www.isc.org
    - ask root for www.isc.org and DNSKEY for .
    - ask org for " and " for org
    - ask isc.org for " and " for isc.org
  - NSEC record
    - no exist — name doesn't exist - NXDOMAIN
    - NOERROR - name exists, but no record of type
    - has next domain name
      - easy enumeration
      - NSEC3 - hash instead so can't
- Network Intrusion Detection (NIDS)
  - approaches
    - scan request like /etc/passwd
      - [illegible], false positives
      - [illegible]
      - HTTPS? can't see
    - HIDS - host based, scan app to back end programs
      - work for HTTPS!
    - log analysis - scan logs each night
    - system call monitoring
  - FP, FN, packet, cost, comparison — [illegible]
  - match on signature
  - anomaly-based
  - specification based
  - behavioral - like what expected from program
  - attack yourself
  - honeypots
- Malware
  - virus, worm, trojan
  - [illegible] [illegible]
  - { - signature detection; - encryption; - polymorphic code; - metamorphic code
  - rootkit - [illegible]
  - hidden BIOS [illegible]