GDB

run (r)

break (b) <func | *addr | line>

step (s), next (n)

stepi (si), nexti (ni) next

continue (c) (to next breakpoint)

<enter> prev command

print (p) [/f] <var | $register>
-print specified value (format f)

list (l) [line] - show src code around current line or line

layout split - split GDB into source, assembly, commands sections

disassemble (dias) [func] - show assembly for current context, or func

x [/nx [b|w] addr : print n bytes (b) or 4-byte words (w) of memory as hex (x)

x86 : little endian

e.g. 0x12345678

0x100	0x101	0x102	0x103
78	56	34	12

last byte of binary representation of multi-byte data type is stored first.

---

x/x : x/x : reveal first two words of stack memory

x/s : next word of stack memory as address

FORMAT STRING VULNERABILITY - can overwrite saved RA

negative val → unsigned int : huge int. wraparound : less than y.

strlcpy (buffer, str, bytelimit)

Defensive Programming - each module takes responsibility for checking validity.

Canary - random value just before the return address in each stack frame.

attacker can read RA still - buffer overrun in inputs (args) that copies arg value at negative indicies into some buffer returned to attacker

Args
RA
canary
local vars

Non-executable stack space
- overwritten RA points to code on stack, but code (injected) cannot be executed
- doesn't protect against all buffer overflows. overwrite vars - passwords, etc.

W^X (write or execute)
- each piece of memory is writeable or executable
  - attacker cannot inject/execute code.
- attack: Return oriented programming → points to standard library
  counter: Address Space Layout Randomization
  - randomize where library code and other text segments are in memory
  → attacker doesn't know address to return to.

Security is a negative property - absence

Testing:
- random inputs (fuzz)
- mutation - change certain statements in src code.
- spec driven

Trusted Computing Base - portion of system that must operate correctly for security goals to be assured.
- unbypassable
- tamper-resistent
- verifiable

reference monitor - TCB for access control

Time of Check to Time of Use
TOCTTOU
- concurrent code execution

Modularity / Isolation
- privilege seperation.

- Security is economics
  → not worth spending more than system's worth
  → secure weakest links
- Least Privilege
  - give program minimal privileges it needs to do job
  - can do less harm
- Fail Safe Defaults
  - default-deny policies
- Seperation of responsibility
  - split up privilege, no complete power.
- Defense in Depth
  - multiple redundant protections
- Psychological Acceptability / Human Factors.
- Ensure Complete Mediation
  - check every access to every object
- Know your Threat Model
- Detect if you can't prevent
- NO Security Thru Obscurity
- Design security in from start
- Conservative Design
  - evaluate systems according to worst security failure possible
- Kerckhoffs Principle
  - systems should remain secure even when attacker knows all internal details of system.
  - KEY only secret.
- Break Own Systems

---

access in one place

Security Props:
confidentiality : people that access should have access to data
integrity : only people w/ access can edit data
availability - server up to answer queries.

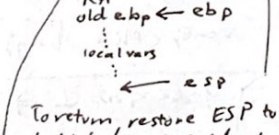
Authorization - who should be able to perform what

Authentication - verifying who's requesting action

Audit - log of actions

Accountability - holding people legally responsible for what they do.

Two Factor Authentication
- Something you know
- something you have } 2.
- Something you are
1) password, account deets
2) token/phone
3) biometrics.



To return, restore ESP to value held in EBP (old ebp), pop old EBP, return

RA = ebp + 4.

1st param = ebp + 8

local var 1 = ebp - 4.

---

canaries defeated by info leakage, or random guess

Format string : learn contents of stack frame, other parts of memory, write to other addresses.

Privilege seperation reduces size of TCB.

---

junk_rip
junk_sfp ← func1_sfp
func1_rip
func1_sfp ← ebp, esp
func1_var

junk_rip
junk_sfp ← func1_sfp
func1_rip
func1_sfp ← func2_sfp
func1_var
func2_arg
func2_rip
func2_sfp ← ebp, esp
func2_var

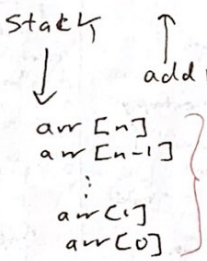
Return... from func2
junk_rip
junk_sfp ← func1_sfp
func1_rip
func1_sfp ← func2_sfp, ebp
func1_var ← esp
func2_arg
func2_rip
func2_sfp
func2_var

---

off-path
- blind hijacking (established conn)
- blind spoofing (new conn) only need ACK easier than blind hijacking

---

x86



Spoof

Port #s and seq/ack nums

only port #s

TCP vs UDP
- TCP is connection-oriented, reliable, bystream
  - rate-controlling, responds to changes in network
- UDP is datagram-oriented, unreliable
  - lightweight
  - best-effort

seq nums is important bc dest will ignore anything w/ incorrect seq num

---

- JS can access cookie based on SOP
- browser attaches cookie based on cookie policy
- browser deletes cookies after expires

HttpOnly : cookie cannot be accessed by JavaScript, only sent to browser

Cookie Scope:

SET
Server can set cookie to domain: any domain-suffix of URL-hostname except TLD
path can be set to anything

SENT
Goal: server only sees cookies in its scope
Browser sends all cookies in URL scope
- cookie-domain is domain-suffix of URL-domain
- cookie-path is prefix of URL-path
AND - secure

ISPs not obligated to verify IP source address (1)

One Time Pad

Key: shared random key K

E: $C = M \oplus K$

D: $M = C \oplus K$

$x \oplus 0 = x$

$x \oplus x = 0$

$x \oplus y = y \oplus x$

$(x \oplus y) \oplus z = x \oplus (y \oplus z)$

$x \oplus y \oplus x = y$

Shared key cannot be reused:

$C = M \oplus K$, $C' = M' \oplus K$

$C \oplus C' = M \oplus M'$ IND-KPA

Block Cipher - deterministic

- symmetric: shared random key
- k bit random key: K

$E: \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$

- $E_K: \{0,1\}^n \to \{0,1\}^n$ "random"

E_K: permutation (bijective)

Inverse $= D_K$

$D_K(E_K(M)) = M$

e.g. Advanced Encryption Standard

block length: 128 bits

K: 128 bits

Advantage: can reuse.

Electronic Code Book

M: broken into n bit blocks

$M_1, \ldots, M_n$

$C_i = E_K(M_i)$

$C = C_1 \cdot C_2 \cdot \ldots \cdot C_n$

(concatenation)

DETERMINISTIC, no entropy

CBC (Cipher Block Chaining)

$C_0 = IV$

$C_i = E_K(C_{i-1} \oplus M_i)$ ← message, can tamper

$C = IV \cdot C_1 \cdot C_2 \ldots \cdot C_n$

Output Feedback

$Z_0 = IV$, $Z_i = E_K(Z_{i-1})$

OTP: $C_i = Z_i \oplus M_i$

$C = IV \cdot C_1 \cdot C_2 \cdots C_e$

- easy to tamper w/ C.

Counter Mode

$Z_i = E_K(IV + i)$

$C_i = Z_i \oplus M_i$ ← message block

$C = IV \cdot C_1 \cdots C_e$

Decrypt

$M_i = C_i \oplus E_K(IV + i)$

- parallelizable

CBC, CTR are KPA/CPA

reuse same nonce:

CBC: leak info about initial plaintext blocks up to first difference between two messages.

CTR: leak info about various blocks.

Pseudorandom Generator

$PRG(K, IV) = E_K(IV \| 1),$

$E_K(IV \| 2),$

$\ldots$

$E_K(IV \| ceil(M/N))$

E_K - n bits

produces M random bits.

PRG

- key length: K - 2^K possibilities
- produces any number of random bits

OWF

- given x, compute f(x) in poly time (input size)

Modular Exponentiation

- compute $a^b \bmod p$

$a^1, a^2, a^4, a^8, \ldots a^{2^{\log b}}$, mod p

- modular multiplication

↳ $O(\log^2 p)$

↳ $O(\log^2 p \log b)$

$g \xleftarrow{\$} [1, p-1]$ p, 2048 bits

$x \xleftarrow{\$} [1, p-1]$

$y \leftarrow g^x \bmod p$

$Pr(Adv(g, p, y) = x^*$ st $g^{x^*} \bmod p = y) =$ negl.

Diffie Hellman Key Exchange

public: $p \sim 2048$ bits, prime, $1 < g < p-1$

Alice: $a \xleftarrow{\$} [1, p-2]$ secret

Bob: $b \xleftarrow{\$} [1, p-2]$ secret

$A = g^a \bmod p$ → A → $B = g^b \bmod p$

public key ← B ← public

$B^a = g^{ab} \bmod p$ $\quad A^b = g^{ab} \bmod p$

$\| \quad \|$

K_{AB} ← shared symmetric key → K_{AB}

MITM

A → E ← B

E ← → E

$E^a = g^{ae} \bmod p$ $\quad E^b = g^{eb} \bmod p$

K_{AE} $\quad$ K_{EB}

$A^e = g^{ae}$ = K_{AE} $\quad$ $B^e = g^{be}$ = K_{EB}

check $K_{AE} = K_{EB}$.

If not, MITM.

El Gamal

$p \sim 2048$ bits, $2 < g < p-2$ } public

$b \xleftarrow{\$} [0, p-2]$ private

$B = g^b \bmod p$

$r \xleftarrow{\$} [0, p-2]$

$E_B(m) = (g^r \bmod p, m \cdot B^r \bmod p)$ = (R, S)

$D_b(R, S) = R^{-b} \times S = g^{r \cdot -b} \cdot m \cdot B^r$

$= g^{-br} \cdot m \cdot g^{br}$

$= m \bmod p$.

- r changes every time
- OTP: modular multiplication.

Hashing

$H: \{0,1\}^* \to \{0,1\}^l$

fixed-length

e.g. SHA 256 → 256 bits of output.

- deterministic
- output looks random

1) One Way

$x \Rightarrow h(x)$ easy

given y, $y = h(x)$, find x: hard.

2) No one can find x and x' st $H(x) = H(x')$, $x \neq x'$

Collision Resistant

MACs: K - secret key, M - message

$F(K, M) = T$ (tag)

Alice: K $\quad$ Bob: K

$F(K, M) = T$

(M, T) ⟶ $F(K, M) \stackrel{?}{=} T$

- deterministic, i.e. $F(K, M) \neq F(K, M')$

⇒ not IND-CPA

Given $T = F(K, M)$, cannot find T', $T' = F(K, M')$

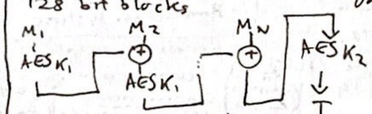
- existentially unforgeable.

AES-EMAC

$K = 256$ bits $= (K_1, K_2)$

2 AES keys

MAC(K, M) - split M into 128 bit blocks



$S_i = AES_{K_1}(S_{i-1} \oplus M_i)$ (128 bits)

w/ only 1 key, forgeable.

e.g. M_1, M_2

$MAC(K, M_1) = S_1$

$MAC(K, M_1') = S_1'$

$MAC(K, [M_1, M_2]) = S_2$

$\Rightarrow MAC(K, [M_1, S_1 \oplus M_2 \oplus S_1']) = S_2$.

successful forge ⇒ MAC insecure.

$MAC(K, E_K(M))$ ☆

Digital Signature

Keygen() → SK, PK

Sign(SK, M) → sig

Verify(PK, M, sig) → 1 or 0

non-repudiation - cannot deny that it was signed but doesn't mean she created

Alice — M, sig(SK, M) → Bob

Verify(PKA, M, sig)

signature - existentially unforgeable

$\gcd(x, n) = 1 \Rightarrow x^{\phi(n)} = 1 \bmod n$

IV's

1) cannot repeat across messages

2) unpredictable

soln: random IV.

	Symmetric	Asymmetric
Confident	Symmetric key Encryption AES-CBC	Public key Encryption El Gamal
Integrity, Auth	MACs	Digital Signatures

RSA

- no attacker can forge a signature

Keygen() - pick random primes, p, q (2048 bits)

$p, q \equiv 2 \bmod 3$

public key $n = pq$

$\phi(n) = (p-1)(q-1)$

$x^{\phi(n)} \equiv 1 \bmod n$

- compute d st $3d \equiv 1 \bmod \phi(n)$
- PK = n
- SK = d
- Sign(SK, M) = sig $= H(M)^d \bmod n$, H is crypto hash.

Verify(PK, M, sig):

$sig^3 \bmod n \stackrel{?}{=} H(M)$

$(H(M)^d)^3 \stackrel{?}{=} H(M) \bmod n$

$(H(M))^{3d} \equiv H(M) \bmod n$

$H(M)^{\phi(n)+1} = H(M)$

$1 \cdot H(M) = H(M) \bmod n$.

Salt - added to make a common password uncommon.

Hash is used bc one way. (S, M) pair is easy to compute.

$S^3 = M \bmod n$

↳ easy to forge.

Can easily find (picking S) $S^3 = M \bmod n$

but cannot find M st $S^3 = H(M) \bmod n$ ← verification.

PW
- store hash of PW

Dictionary Attack: try all pw against hash
- amortized pw hashing
→ compute hash once for each PW
50% of PW, 2^{20}

SOLN: salt
- randomize hashes w/ salt

server stores (salt, h(pw, salt))
- each PW is guaranteed to be different
→ 1 brute force attack per hash

slow down attacker doing dict attack using slow hash
- $H(x) = h(h(h...(h(x))))$

Key Management - Certificates

PKI: infrastructure for managing PK and certificates

Trusted Dir Service - contains all PKs, susceptible to MITM.

CA signs all PK,
- msg includes username, PK

w/out expiry → susceptible to replay attack if PK has private key compromised

Expiry ⇒ prevents replay attack across expiration periods

For all time:
- can use nonce; requires direct communication w/ server
- TD must be online
- not scalable - must know all PK
- central point of attack and failure.

Hierarchy PKI
- parent signs PK of child
- CA at root
→ certificate chain
- verified by browser
- scalable: no one entity issues all cert.
- SK compromise leads to impersonation of only children under compromised

---

TLS/SSL

end to end: communication protecting from OG client to intended server

Secure Sockets Layer / Transport Layer Security
- secure for any app that uses TCP
- ensures confidentiality/I/A for server

RSA Encryption
- encrypt = verify BUT use special padding instead of hash
- decrypt = sign

B —R_B→ S : 256 bits random num (supported crypto algs)
Supported ciphers
prevent replay attacks

←R_S, algs— : 256 bits random num

← : certificate. Browser validates cert

$\overrightarrow{(PS)_{PK_S}}$ RSA: browser constructs premaster secret, sends it to S using public RSA key of S

PS, R_B, R_S → C_B, C_S : sym cipher keys
I_B, I_S : MAC integrity
2 for better randomness

$\overrightarrow{MAC(dialog, I_B)}$
$\overleftarrow{MAC(dialog, I_S)}$
} to ensure same keys
Keys: 2 to prevent reflection attacks

Prevent MITM for PS: Diffie Hellman
- server sends $\{g, p, g^a \bmod p\}_{SK_S}$ (a = random; signed w/ secret key of server (integrity))
- browser verifies
- browser generates random b, sends $g^b \bmod p$
replay-only client may hits
$PS = g^{ab} \bmod p$

DH has fwd secrecy - PS never sent over network
Different symmetric keys prevent replay
certificate also sent over.

HTTPS: first TCP, then TLS

Cert: browser compares domain name in cert w/ URL
→ end to end
- compare sent over cert and own cert (hashed)
- incorrectly trust ⇒ lose all props.
- cannot find cert: lose auth

TLS secures network, not site
↳ SQL injection / XSS
↳ latency: extra round trips
- TLS relies on trustworthiness of certificates
- Browser has separate cert belonging to issuer - different from cert sent over network.

Networking

LAN - no router
WAN - has router
} uses packets

Protocol - agreement on how to communicate
incl syntax - structure of comm
semantics - what comm means

Dumb Network - interior nodes "routers" have no knowledge of ongoing connection thru them

Packet: Header (delivery)
Payload

Layering: Internet
- each layer relies on services provided by next layer down

Application	7	TLS
Transport	4	TCP/UDP
Inter Network	3	IP
Link	2	
Physical	1	packets

Lower layers headers precede higher layers' in packets.

Physical - encode bits to send them over a single physical link

Link - framing/transmission of bits into individual messages sent across 1 subnetwork
- may involve multiple physical links
- supports broadcast transmission

IP - bridges multiple subnets to provide end to end internet connectivity between nodes
→ global addressing

Transport - end to end communication between processes
e.g. TCP, UDP

Application - communication of whatever you wish
use convenient transport
e.g. Skype, HTTP

3, 4, 7 only implemented at hosts

IP Packet:
16 bit total length of entire packet (bytes)
8 bit protocol - how to interpret start of payload - header of transport protocol
32 bit source/dest IP addr
"best effort" packet delivery

Transport:
TCP - connection oriented
reliable, in-order, byte-stream delivery

Packet: src, dest ports
IP addr + TCP port uniquely identify connection

Sequence # - starting seq # of data carried in packet

ack - gives seq # just beyond highest seq. received in order
N bytestream bytes at seq S,
ack = S + N

Flags: ACK; setup SYN; close connection FIN, RST

TCP handshake
A —SYN→ B (ISN)
A ←SYN+ACK— B (ISN)
A —ACK→ B

e.g.
A → B SeqNum = x
A ← B SeqNum = y, Ack = x + 1
A → B Ack = y + 1

UDP Packet
src port dest port
length checksum

HTTP Request
GET: download data
POST: upload data

Hostname - full abc, used by humans

IP - numerical address used by routers
fixed length
hierarchical, related to host location

Layers 1 + 2 Threats

Eavesdropping - Free
- each system's NIC can capture comm on subnet IF using broadcast technology

Confidentiality - no one can read

I - no one can manipulate data / processing / communication

Availability - we can access when we want

Disruption (I) - jam packets (integrity)

Spoofing - create a message w/ forged src addr

on-path: easy. can see headers
off-path (blind): guess headers

IP Layer Threats
- set arbitrary src addr - spoofing
- arb dest addr - scanning
→ search for hosts
- send like crazy (flooding)

Dynamic Host Configuration Protocol

client (get IP) —discover→ DHCP server (broadcast)
← offer — IP addr, DNS server, "gateway router", lease time
request →
← ack

Attack: fake DNS server
↳ redirect any of host's lookups to machine of attacker's choice

FAKE gateway router
↳ intercept all of hosts off-subnet traffic
↳ modify contents

TCP Threat: Data Injection
know port/seq #s, attacker can inject data into any TCP connection
= "connection hijacking" - on estab. conn.
- attacker who can observe TCP connection can manipulate
- predict ISN chosen by server, can "blind spoof" a conn to server

⇒ no security against on-path attacker, more against off-path

---

DNS lookups via resolver
- caching used to minimize lookups

DNS: path-critical
hostnames → IP addr

dig - dump resource records
DNS name, time-to-live, family, type, associated val

DNS message: identification - 16 bit # for query/reply
Question, Answer, Authority
↳ name server responsible for answer
Additional - info client is likely to look up

To fix cache poisoning:
only accept Additional records if they're for domain we're looking up
e.g. look up eecs.mit.edu, only accept *.mit.edu

Risk 1: malicious DNS server
- fixed cache poisoning

Risk 2: on path attacker
based - can see query and 16 bit identifier

Risk 3: off-path attacker
Blind spoofing
- force us to visit some website
- guess identification, race server

But once reply from legit server comes, victim innoculated

Hierarchy of Domain Names 161

empty
↙ ↘
.com .edu
google.com
↙
www.google.com
↙
www.mail.google.com

TLD: Top level domains
Generic: .com, .edu
Country-code: .us, .it

Cybersquatting: buying a domain in advance of it becoming desirable and selling it to agency who needs it.

Resolver queries distributed hierarchy of DNS servers (name servers)

e.g. client ↔ resolver (1, 8); resolver ↔ root DNS (2, 3); resolver ↔ .edu (4, 5); resolver ↔ mit.edu (6, 7)
"IP for eecs.mit.edu?"
"dk, ask .edu"
"dk, ask mit.edu"

authenticity / integrity for DNS results

DNSSEC: standardized DNS security extensions currently being deployed
- ensures integrity of DNS lookup

do DNS lookups over TLS
↳ flawed

make DNS results like certificates (expire)

DNS lookups susceptible to enumeration attacks if consequent domains are signed to indicate absence

DNSSEC - sign all DNS records to verify answer to DNS query
resolver gets signed statement regarding keys used by next level

DNSSEC includes next server's public key and a signature over DS record
↳ tell which ID (child)

parent gives signature on child's key
⇒ parent verifies child's key

Issue 1: partial deployment
- only some sign

Issue 2: Negative results
- either dynamic key signing (stay online) or enumerate

Issue 3: Replies are big - amplification

TLS: channel security over TCP
- CIA
- client/server agree on crypto, session keys
- security dependent on trust in CA

DNSSEC: object security for DNS results
- just integrity/authentication
- no client/server dialog
- cache friendly
- dependent on trust in root name server's key, all other signing keys

---

Firewalls
- access control policy
- distinguish between inbound/outbound connections.

Packet Filter
- inspects each packet for certain filtering rules to determine whether to pass or block
- no logging

Stateful Packet Filter
- keeps track of all connection
- which connections are allowed/denied
- uses TCP to reconstruct order based on seq num

Application Level Firewall
- TCP connection from client to firewall, firewall to server
- can examine traffic in detail, log
- processing intensive

Advantages:
- central control
- easy to deploy (transparent to endusers)
- addresses important problem

Dis:
- functionality loss - less connectivity
- assumes insiders are trusted
- firewalls establish security perimeter

Secure external access to inside machines
- create secure channel to tunnel traffic from outside host/network to inside. VPN 3

VPN - User authenticate, packets forwarded to dest.
Firewall allows traffic to VPN server
Tunneling: mechanism for encapsulating one protocol in another.

Network Intrusion Detection (NIDS)
- passively monitor network traffic for signs of attack
- has table of active connections, maintains state for each
  ↳ has it seen partial match of attack/scanning?

Evasion Attacks:
- reflect incomplete analysis
- imperfect observability
  - NIDS can't see exactly what arrives at dest

⇒ look at network traffic
- no need to touch/trust end systems
- cover many systems w/ single monitor
- centralized management

Host Based Intrusion Detection
- instrument the server
- scan arguments sent to (program input) back-end programs
  Pro - no problems w/ HTTPS complexities
  - works for encrypted HTTPS
  Cons - have to add code to each web server
  - have to consider UNIX filename semantics
  - have to consider other sensitive files

Log Analysis
- each night, scripts to analyze log files generated by web servers
  - scan args sent to backend programs
Pros: cheap; servers have loggy facilities
Issues: - must consider filename tricks
  - can't block attacks, prevent from happening
  - detection delayed and attacks succeed
  - malware may alter logs

System Call Monitoring
- monitor system call activity of backend processes
Pros: no issues w/ HTTP complexities, filename tricks
Cons: - maybe other processes make legit accesses to sensitive files
Detection Accuracy:
False positive - alert when no problem
False negative - failing to alert about a problem
I = instance of intrusive behavior
A = alert
FP = P(A|¬I)
FN = P(¬A|I)

ARP (Address Resolution Protocol)
- every device that connects to a network has a network interface which has an identifier
- MAC addr = identifier for device, 48 bit identifier
- can be changed by software
- data frames at link layer are sent to MAC addr not IP addr
- at link layer, packet needs to be forwarded to MAC addr
- sender has dest IP, needs MAC
ARP spoofing - no authentication
- all machines on network cache ARP reply
MITM - Eve sends ARP reply to Alice to associate Bob's IP w/ Eve's MAC
  Eve sends ARP reply to Bob to associate Alice's IP w/ Eve's MAC
  - also DoS
Soln: only trusted users have access to local network
  - multiple occurrences of same MAC addr on LAN
  - static ARP tables

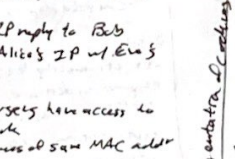
DoS on IDS
- send so many attacks that match also to IDS
- OR fake new connections

typical HIDS
- combo of system call monitoring, program inputs, system logs

Types of Detection:
- Signature based
  → matches structure of known attack
  → conceptually simple, easy to share
  → blind to new attacks
  → looks at low-level syntax, not powerful
Vulnerability Signatures
- match on known problems
- can detect variants of known attacks
- more concise than per-attack signatures
- can't detect novel attacks
- signatures can be hard to write (need to see HOW attack works)
Anomaly Based Detection
- attacks look peculiar
- develop a model of normal behavior
- potential detection of wide range of attacks, including new ones
- can fail to detect known attacks, and new attacks
- need good data to train on
Specification-Based Detection
- specify what's allowed
- can detect novel attacks
- low false positives
- BUT lots of labor to derive specs
Behavioral Detection
- look for evidence of compromise
- can detect new attacks
- low false positives
- cheap
- cannot prevent problem, can only detect
- BRITTLE! attackers can avoid
Evasion Terms
- arise from uncertainty or incompleteness to detectors not after behavior it can't directly observe
1) impose "canonical" form: normalize
2) analyze all possible interpretations rather than assume one
3) fix basic observation problem
  - monitor directly at end systems
HIDS: URL/web blocking
  protocol scanning of network traffic
  payload scanning
  cloud giveaway regarding reputation
  - sandbox execution
  - file scanning (malware install)
  - memory scanning (malware never appears on disk)
  - runtime analysis
NIDS
- deployment inside network as well as at border
- full protocol analysis
- signature analysis

NIDS	vs	HIDS
- cover a lot of systems		- direct access to semantics of activity
- no need to touch end systems		- better positioned to block attacks
- does not steal production resources		- protect against network threats
- harder for attacker to subvert		- visibility into encrypted activity
		- performance scales much more readily

Web - platform for deploying applications and sharing info, robustly + securely
URL: global identifier of network-retrievable resources.
http://safebank.com:81/account?id=10#top
protocol | hostname | port | path | query | fragment
JS - manipulate web pages; high level, untyped, interpreted w/ support for objects



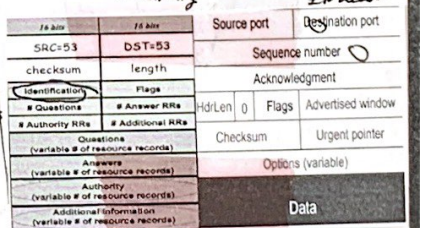
Document Object Model - cross platform model for representing and interacting w/ objects in HTML
JS can access cookies

Frames
- enable embedding of page within a page
- frames are isolated
JS is sandboxed - site cannot touch computer
Same Origin Policy:
- each site in browser is isolated from all others
Origin: protocol + hostname + port
STRONG MATCHING
SOP: one origin cannot access resources of different origin
- origin is derived from URL it was loaded from
  ↳ images, JS all in embedding page
- iFrame - origin of URL from which iFrame is served
Cross Origin Communication
- allowed thru postMessage API
- receiving origin decides whether to accept

Compromised web servers
- steal data, change data, gateway to exploit attacks on clients, impersonation
SQL Injection
- browser sends malicious input to server
- bad input checking
XSS
- inserts client-side script into pages viewed by other users, script runs in users' browsers
CSRF - cross site request forgery
- bad website sends request to good website, using credentials of innocent victim

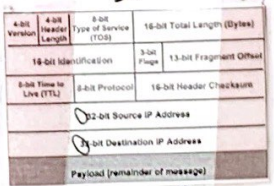
code injection
- attacker provides bad input
→ execute code on server
modern web services:
browser → web server → database server
database - structural collection of data
- store server and user info
- runs as separate process to which server connects
- web server sends query/commands
SQL: Insert into <table> values <vals>
Delete from <table>;
; is key
SQL login:
' or 1=1 --
↑ rest of line ignored
SQL injection prevention:
sanitize user input:
enforce that value/string does not have commands of any sort
- disallow special chars
- escape input string
  \'
- escape SQL parser by only backslash in front of special chars, e.g. ', \
SQL Parser
- sees ' - string is starting or ending
- sees \', considers char as part of str and continues to '
- avoid building SQL command based on raw user input
Parameterized/prepared SQL
- ensures args are properly escaped
- use framework/tools that check user input

XSS
- attacker injects malicious script into webpage viewed by victim user
  - script runs in user's browser w/ access to page's data
- subverts SOP
- attacker tricks server to send malicious script to ~~server~~ users
  ↳ same origin (malicious, .com)
Stored XSS
- attacker leaves JS lying around on e.g. forum, saved on web service for victim to load server
Reflected XSS
- attacker gets user to click on URL w/ script in it, web service reflects it back
Stored - can only happen when server doesn't check user input
  - use images - request for image that is actually a script
Reflected: - web service includes parts of URLs it receives in web page output it generates
  - attacker runs script in browser w/ same access as provided to ~~the~~ server's regular scripts
  - attacker needs user to click on specially-crafted URL
  - server fails to ensure that output it generates doesn't contain embedded scripts other than its own
Prevent XSS
- input validation: inputs are of expected form
- output escaping: escaped embedding
Content-Security Policy
- web server supplies whitelist of scripts that are allowed to appear on page
  - specify domains browser should allow for executable scripts
HTTP is stateless - apps do not store state in client browsers
  - maintain no info from request to request
Cookies - way of maintaining state
Cookie Policy
access: visit site → user's browser sends relevant cookies
  JS
  ↳ which cookies will be sent by browser to which sites
When browser connects to same server later, cookie is included, header containing name/value. server uses to connect related requests.
domain: any domain-suffix of hostname, except TLD
path:
secure: over TLS
expires: | HttpOnly
cookie tells server you are you
Cookie not set when domain more specific than origin. BUT cookies can be read/set by subdomain
host = "login.site.com"
set cookie for: login.site.com, .site.com
browser sends all cookies in URL scope
path can be set to anything



DNS
TCP
src port randomization defends against off-path attackers for DNS.

IP



(4)

Session
- sequence of requests/responses from one browser to a/1 site
- management: authorize user once, subsequent requests tied to user

Past: HTTP auth
- 1 username, pwd for user group

Now: session token
- browser cookie
  - sends cookie w/ every request, even when it shouldn't (CSRF)
- embed in URL link
  - token leaks via HTTP referer header, users might share URLs
- hidden form field
  - short sessions only

Better soln: combination, e.g. browser cookie w/ CSRF protection using form secret tokens

CSRF (cross-site request forgery)
HTML forms provide data that get sent w/ POST request to server
- session assigns session token to user after log in puts in cookie
- server keeps table of username to session token

1) establish session w/ victim server
2) visit attack server
3) receive malicious page from attack server
4) send forged request w/ cookie to victim server

Defenses
CSRF token
- include secret token in web page
- request include secret
- server checks that token embedded in web page is expected one
- need hard to guess token
- server stores state that binds user's CSRF token to user's session ID
- embeds token on every form
- server validates supplied CSRF token is associated w/ user's session ID
- but server needs to maintain large state table to validate tokens

Referer Validation
- when browser issues HTTP request, includes referer header that indicates which URL initiated request
- used to distinguish between same site / cross site request
- strict policy disallows (secure, less usable)
- lenient policy allows (less secure, more usable)

Privacy Issues
- referer contains sensitive info
- reveals contents of search query that lead to website visit

Blocking: sites
- network stopping by organization, " " " " local machine
- stripped by browser for HTTPS → HTTP
- user preference

Authentication
- verifying who some claims to be is who they are
- web server/client should authenticate

How:
- Something you know: pswd, pin
- Something you have: smartphone, ATM card
- Something you are: fingerprint, iris scan

Two factor (2FA)
- uses two of above

After authenticating:
- session established
  ⇒ session ID stored in cookie
- web server has list of active sessions

Really happens on every request - contains cookie

Session hijacking - attacker steals session ID (tells server you are you), e.g. using packet sniffer
- impersonates user

To protect:
- Send encrypted session ID over HTTPS: use secure flag
- cookie should expire more often to be secure, but is less usable
- httponly flag to prevent scripts from getting cookie
- when user logs out, server must remove user's session ID from active sessions,
- don't reuse same session ID
- server authenticates using digital certificates

Phishing
- attacker creates fake website that appears similar to real one
- trick user to visit site
- user inserts credentials and sensitive data which gets sent to attacker
- web page then directs to real site
- user should check URL they are visiting
- check address bar ★
- URL obfuscation
- choose similarly looking URL w/ typo
- Homograph
- unicode characters from international alphabets may be used in URLs
- URL seems correct but is not

Spear Phishing
- targeted phishing that includes details that make it seem legit

Sophisticated Phishing
- context-aware: 10% users
- includes info related to recent action
- social: 70% users
- send spoofed email appearing to be from one of victim's friends

Phishing Works!
- user mental model vs reality
  - browser security too hard to understand
- easy paths insecure
- but many risks

Authenticating Server
- users should:
  - check address bar
  - guard against spam
  - don't click on links/attachments from unknown
- browsers receive blacklists of phishing sites
- mail servers eliminate phishing email

Clickjacking
- User's mouse click is used in a way not intended by user

Frames
- outer page can set frame width, height
- only framed site can draw in own rectangle

Modularity!
- brings together code from different sources

Clickjacking using frames
- evil site frames good site
- covers good site by putting dialogue boxes or other elements on top of parts of framed site to create a different effect
- inner site looks different to user

Visual Integrity - target
- hiding target
- partial overlays

Script compromises context integrity of another application's UI when user acts on UI

1) visual integrity
   - cursorjacking

Site key
- sites used secret image to identify site to user
- only good site should know secret image
- users should check for correct image
- aim to protect against phishing

not effective bc users ignore images and don't remember specific image

Clickjacking Substrate Site Keys
- phishing sites frame login page like get correct image
- overlay input box from outer frame at same location as password box for inner frame

Defenses
- User confirmation
  - pop dialogue box w/ info on action it's about to make
  - degrade UX
- UI randomization
  - embed dialogs at random locations so it's hard to overlay
  - difficult & unreliable
- frame busting
  - site includes code on page that prevents other pages from framing it
  - uses conditional, counter action
  - easily defeated
  - abuse XSS filter
    - figure out framebusting code
    - put framebust code in script tags
    - light framebust code is disabled

Ensure visual integrity of pointer
- remove cursor customization
- freeze screen outside of target display area when real pointer enters target
- lightbox effect around target on ptr entry

Temporal!
- as you click on button for insensitive action, button for sensitive action is overlayed and you click it by mistake

Defense
- UI delay: after visual change on target/ptr, invalidate clicks for X ms
- pointer reentry - after visual changes, invalidate clicks until ptr re enters target

X-Frames Options
- web server attaches HTTP header to response
- DENY: browser will not render page if framed
- SAMEORIGIN: browser will only render if top frame is same origin as page giving directive

Sites you visit learn!
- URLs you like
- IP address
- browser's capability
- OS
- language
- which URL took you there
- cookies

Private Browsing -
- privacy from families
- deletes history, passwords, cookies
- maintains for as long as private browsing window is open

Third Party Cookies
- include third party cookies on site
- business relationship w 3rd party
- can track you
- 3rd party sees all activity that involves their sites

Google Analytics
- any site can register to instrument their site for analytics
- site adds small JS snippet that loads GA
- Google gets info associated w/ visit to site

Any scenario where browsers execute programs that manage persistent state can support tracking by cookies
- every site that embeds FB like button lets FB track you
- cookies form core of how internet advertising works today
  - without, have to pay

BUT privacy concerns

Better Privacy!
- force of law
  ↳ adhere to policies.
  ↳ 58 1386 (CCA) forces sites to disclose security breach
  ↳ GDPR
  - allows ppl to better control personal data
    - consent, reason to store info
    - user has right to know stored info
    - user can req info to be deleted

Better Privacy
- tech:
  - browser additions
  - browser extensions
  - Tor

Browser tracking protection!
- choose blocking list
  - basic: won't break
  - strict: blocks all trackers, may break functionality

Do not track flag says you don't want to be tracked

Malware
- malicious code pretends to be good software or attaches itself as good software
- some need host programs (trojan horses, logic bombs (harmful instruction if condition satisfied), backdoors)
- 'independent': worms, automated viruses
- modern: trojan, rootkit, worm functionality

Virus - infects other programs to propagate
- inserted into host

Worm - propagates automatically by copying self to target systems
- standalone program
- cannot trust code you did not create yourself

Propagation Methods!
- insert copy into every executable
- insert copy into boot sectors of disks
- infect common OS routines

Encrypted Virus - constant decrypter content followed by encrypted virus body

Polymorphic Viruses - each body creates new random encryption of same virus body

Detection!
Anti-Virus Scanners
- signatures
- heuristics for recognizing code associated w/ viruses
- integrity checking

Generic Decryption and Emulation
↳ run in sandbox
↳ check signature at runtime

Metamorphic Viruses
- mutates virus body

Obfuscation and Anti-Debugging
- prevent code analysis and signature based detection, full reverse engineering
- code obfuscation
  - packed binaries, hard to analyze code structures
  - different code in each copy of virus
- detect debuggers and VMs, terminate execution

Propagation via Website
- Drive-By Downloads
  - malicious executable pushed to browser w/ JS, pop-up windows
  - automatically installs malicious software
- many bad sites exist only for a short time

Trust in Web Advertising
- not transitive

Social Engineering
- trick user into installing malicious binary

Rootkit
- gives attacker access to machine while hiding presence
- hides using hidden dir
- install hacked scans for system programs
- can't detect attackers processes, files, network connections by running std UNIX commands

Find rootkit
- need outside disk space bc of sniffer logs (invisible)
- manual confirmation
  - reinstall clean os and see what's running
- automatic detection: HIDS

Signature Based Defenses Don't Help
- worms are brand-new, scanners cannot detect new signatures
- detection of "zero-day" attacks
- signature not extracted until hours or days later

Botnet
- network of autonomous programs controlled by a remote attacker and acting on instructions from the attacker
- machine owners not aware they've been compromised
- used as platform for various attacks: DDoS, spam, click fraud, launching pad for new exploits

Life Cycle:
- exploit vulnerability to execute bot program on victim's machine
- bot is downloaded and installed
- disable firewall/anti virus
- locates IRC server, connects, joins channel
  - makes DNS server lookup for IP address of IRC server
  - joins channel of attacker

Detect:
- bots controlled via IRC, DNS
- visible in network
- look for host performing scans and for IRC channels w/ high percentage of hosts, hosts who ask many DNS queries but receive few queries about themselves
- bots can evade detection using encryption, P2P

propagation

Scan rate - rate at which IP addr of potential target is generated

Man in the browser attack
- victim runs compromised browser, browser modifies user requests

BTC proof of work
- Bob provides random challenge r
- find x s.t. H(r,x) has 33 leading 0's
- on average takes 2^{33} hash computations
- verify: check that H(r,x) starts w 33 leads

hash chain
- on day i, need to add item D_i
- have hash h_{i-1} from days 1...i-1
- $h_i = h(h_{i-1}, D_i)$
- any switch or truncation in chain is evident bc collision resistance

Bitcoin
- cryptocurrency: digital currency whose rules are enforced by cryptography and not by trusted party
- avoid trust in institutions
- built on blockchain
- each user has PK, SK
- user signs transactions (transfer) using SK
- transaction on blockchain
- all signatures, transactions sorted in order of creation; including previous transaction where money came from
- each transaction contains hash of previous transaction
- block includes transaction description, signature

block i:

$TX_i = CPK_x \rightarrow PK_y$; list of transactions for x'; $h(block_{i-1})$

$sign_{SK_x}(TX_i)$

- can verify that blocks 1...i are not compromised using h(block i)
- only miners allowed to add blocks to blockchain
- miners must solve proof of work → can include random #s in block so hash changes until proof of work is done
- new transaction is broadcast to everyone
- solve proof of work → block includes all transactions miner checked were valid
- consensus - longest correct chain wins - everyone checks all blocks and all transactions
- assumes most miners are honest
- reward given to miners
- need ≥ 51% of computing power to fork chain
- fork is harder unless s.t. can mine new entries faster than combined everyone else
- have to wait a few transactions to guarantee that your transaction is permanent
- each transaction includes a fee which goes to miner
- proof of work is adaptable
  - too long: less zeros
  - too short: more zeros

Mining Pools
- need too much computing power for regular person to mine
- can contribute cycles to mining pool: group of many machines w/ good success of mining on average
- more predictable income

BTC is not anonymous
- mitigation: use >1 PK
- OR ZCASH

Blockchain is powerful
idea of storing info in immutable way

Problem: cannot erase info

Discrete Log
hard to find a,
$A = g^a \bmod p$

censors can block encryption

side channel attack - computer system implementation

defend against replay attack: timestamps, counters, nonces

- to hijack TCP connection, need to eavesdrop to get sequence #s, inject packets w/ forged IP src
- on path attacker can see TCP sequence #s

code injection - buffer overflow, XSS

VPN, TLS, WPA make it hard to observe traffic

DNS
- compromise DNS server and use fake signing key - can do everything

1988 Morris/Redux
- measure internet
- uncontrolled spawning
- remote execution using rsh and cracked pwds

Code Red I
- buffer overflow in Microsoft's Internet Info server
- finds targets by random scans of IP addr 1-20
- 21-end: attack

Code Red II
- scan prefers nearby addr
- payload, installs root backdoor for unrestricted remote access

Nimda
- 9/18/01: Multi-modal worm using several vectors
- bulk emails itself
- copies itself across open network shares
- adds exploit code to web pages on compromised sites
- scans for CR2 backdoors

Slammer
- UDP worm exploiting buffer overflow MSSQL
- entire code in 1 UDP packet
- scan rate: 55 mil/second
- saturated carrying capacity of internet in 10 min

Botnet
Storm:
- spreads via exciting email subject w/ malicious attachment

Zero Access
- no CRC
- used for click fraud
- trojan downloads ads and clicks them

StuxNet
- computer worm, trojan horse, virus
- targeted industrial systems
- 1st to exploit multiple zero day vulnerabilities
- 1st to use stolen signing keys and valid certificates of two companies
  ↳ Iran Nuclear Program

Modular multiplication can be compromised over ElGamal
↳ vulnerable to chosen ciphertext attack

- clickjacking won't allow user to steal cookies
- DNS amplification

Tapping
- UCSB study to study botnet
- bots generated domain names to find command + control server (C&C)