Internet Security

Security threats: Leakage, Tampering, Resource stealing, Vandalism
} Information Domain
} Operation Domain

Attacking methods: Eavesdropping, Masquerading, Message tampering, Replaying, Exploiting + Combinations
Social Engineering

Passwords: long, mixed chars, no security questions, password policy, traps

OSI Layers: Physical, Data, Network, Transport
↳ Fragmentation — Ethernet — IP, ARP(→Frag) — TCP, UDP, ICMP

Layer 2/3 Attacks:
- Ping of Death: include max IP datagram size ↳ with fragmentation
- IP fragment overwrite: overwrite Layer 4 properties (+ tiny fragment) ↳ Port (firewall)
- Network sniffing: MAC flooding, MAC duplicating ↳ countermeasures
- ARP poisoning: MITM, DoS, (gateway), single host
- ICMP echo attack: packet amplification (SMURF) ↳ amplification attacks
- IP spoofing

Layer 4:
- UDP Spoofing / Hijacking
- UDP Storm
- UDP Portscan: zero length packet

UDP: connectionless unreliable
TCP: connection-oriented, three-way-handshake ↳ SYN, SYN+ACK, ACK

- TCP Scanning: open connection ↳ SYN Scan: not logged ↳ FIN Scan: ignored if open port ↳ OS fingerprinting
- TCP Spoofing / Hijacking: hard
- TCP DoS ↳ SYN Flooding ↳ Process Table Attack

Race conditions: parallel execution of tasks, result of tasks depend on timing of events → indeterministic behavior

→ Sequence of operations (A,B): not atomic, can be interrupted
↳ window of vulnerability: (ta, tb)

→ TOCTOU: time of check, time of use: assume something is still valid on time of use
→ problem, if time of attack is between check and use
→ check permission ↔ open file
↳ abstraction if using file-names: symbolic links

↳ exercise race, non-x race, temporary files: guess name, link to own file

→ Beating the odds:
- attacker can try 1 million times
- attacker can slow down programm:
  - slow filename lookups: deeply nested directories
    ↳ chain of symbolic links (file system race)
  - computational complexity attacks: worst case performance
    ↳ overflow caches in hashtables (worst case)

→ Detection & Prevention:
- operate on file descriptors
- do not check access by yourself
- drop privileges
- locking (O_EXCL → file open)
- static code analysis
- model checking

→ Examples:
- passwd, sendmail
- Signals
- Race against the cops (Android)
- Web: Facebook, Starbucks

Web Application Security : mixture of different protocols, formats, languages
→ big technology stack + attack surface

HTTP: request / response line, header section, (entity body)

OWASP Vulnerabilities:
- Injection
- Cross Site Scripting
- Broken Auth. + Session Management
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Failure to restrict URL Access
- Missing Function Level Access Control
- Cross Site Request Forgery
- Insufficient Transport Layer Protection
- Using components with known Vulnerabilities
- Unvalidated Redirects + Forwards
- Buffer Overflows
- Improper Error Handling
- Denial of Service
- Unvalidated Input

Unvalidated Input:
- many different ways of encoding information
- check via validation rarely bypassed
  + taint analysis
  + whitelist validation + validation / sanitizing

SQL Injections:
- craft malicious query with input parameters
- gather information through errors

  – Blind SQL Injection → add knowledge
  – Second Order SQL Injection → store malicious text / wait for another use which is maybe not validated

  + prepared statements, stored procedures

Parameter / Command Injection:
- input parameters for shell command
  + filter special-meaning chars

Session management:
- session data stored on server, associated with ID
  ↳ HTTP is stateless

  Session ID Transmission: GET parameter, POST parameter, Cookies
  ↳ Cookies: non-persistent, secure, IP address validated
  encrypting cookie values → replaying

  Session Attacks: Interception, Prediction, Brute Force, Fixation
  + SSL + random + hashing

→ use existing solutions
→ timeouts
→ logout link to invalidate

Javascript: sand-boxed, same-origin-policy

Cross Site Scripting: make server render malicious script code
→ rewrite content, phishing, cookie stealing
→ stored vs. reflected

Techniques: wrong unicode source, to read information
beginners filter by unicode equivalents, bit breaks
wrong layer to read data

+ Content Security Policy
+ Static code analysis
+ http only cookies

Improper Error handling: fail-open security checks
display critical data in errors (database, stack)
+ error handling policy

Insecure Configuration Management: improper file / directory permissions
default accounts / settings
unnecessary backups / sample scripts
misconfigured SSL

Insecure Storage: failure to encrypt critical data
poor choice of algorithm
own algorithms
(passwords, credit cards, keys)

Denial of Service Attacks: consume resources, at such a rate that no other use can enjoy the service
→ Distributed variants possible DDoS

Flooding: exploited zombies controlled by masters
→ hide masters behind stepping stones

vulnerability attacks: vulnerability cause service to crash or stuck in loop

→ hard to tell difference between
↳ anomaly detection / signature detection

Internet Applications

Remote Access: telnet, rlogin: insecure
ssh: replacement, since version 2 secure

DNS: map domain names to IP adresses
↳ distributed database
↳ name space hierarchically divided: each domain managed by a name server
↳ mostly UDP

→ 13 root name servers
→ server types: primary
secondary
caching-only
forwarding

→ if server cannot answer request, forward up in hierarchy
→ reverse lookup: ip-adress.in-addr.arpa
→ queries can be recursive / iterative

Simple DNS Spoofing: authentification based on DNS names with reverse lookup
DNS server which is responsible for own IP responds
fake wanted name

→ countermeasure: normal lookup for returned name

DNS cache poisoning: DNS replies sent by UDP
→ spoof UDP reply

Spoofed DNS reply: match query, UDP port, DNS nonce field
→ attack caching server only

→ Denial of Service, MITM, redirect traffic
→ also used by providers

FTP: control and data stream, active and passive mode
active: client tells server to connect to used port
passive: server opens port and sent it to client

→ configuration: anonymous write to home dir
→ passive connection trick
→ FTP bounce: port scan, scan behind firewall, send data to arbitrary port

SMTP: no authentification → open distributed system
Open Mail Relay: does not verify message sender

authentification: IP adress, SMTP AUTH, POP before SMTP
encryption: SMTPS, like HTTPS, STARTTLS

Adress Spoofing: spoof IP adress, forge source adress
→ countermeasures: SPF (uses TXT records of DNS)

Spam: Gathering Email Adresses: brute force
harvesting
malware

Delivering: open mail relays
web forms
zombie nets

Countermeasures: spam filtering
blacklist
spam list

NTP: time synching protocoll
→ all time based authentification
→ monlist: sends last 600 ntp clients
↳ reflection attack, amplification attack

Testing: validation: does design meet problem requirements
verification: does implementation meet design requirement
→ classification: whitebox / blackbox
static: theoretically, no code is executed
dynamic: feed program with input, observe behaviour
⇒ automatic testing, regression testing, Software fault injection
⇒ testing at all development phases

Requirements Analysis Phase:
explicitly state security requirements: how the system needs to attack scenarios

Design Phase:
formal methods: mathematically describe and verify.
formalize state transitions and describe unsafe states
model checker ensures, that no unsafe state is reached
attack graphs: finite state model, security property,
set of attacker of state model that violate security property

Implementation Level

Static security testing: manual audits
syntax checker: basic check
annotation based system
model checking: model realized as state machine
certain states are insecure
analyze control and data flow
meta compilation: programmer adds compiler extensions
extensions check the code

Dynamic security testing: check system calls
check library function calls (eg valgrind)
Profiling: sequence of funct. calls, arguments
Fuzz testing

Fuzz testing: randomly program with (semi-) random input
monitor for crashes, deadlocks, ..
→ protocoll parsing
→ minimal protocoll specification possible

Pre-Rollout: remove debug code, information, test accounts
reset security settings

Penetration testing: analysis for design weaknesses, vulnerabilities
results delivered in a report
- external penetration testing (from outside)
- internal security assessment (from inside)
- application security assessment

Limitations: timeslots, client defines scope, rarely for reports

Buffer Overflows: input instructions into memory
exploit programm to change flow of execution

- modern languages provide automatic buffer size checks (memory management)
  ↳ C/C++ do not

- stack is composed of frames, upon function call: new frame pushed on stack
  frame contains: local variables, previous frame address, return address, parameters

→ place shell code in unchecked buffer
→ overwrite return adress to address of shell code

Shell Code: usually used to start shell with root rights
use system call to start shell
↳ passing arguments in registers and interrupt
↳ gather string addresses with jmp and call in shell code

Code Pointer: usually return address is used
overwrite with value which points to shell code
↳ NOP slid in shell code → value just needs to hit slid

Small Buffers: store shell code in environmental variable

Defenses: not executable stack, stronger version: write XOR execute (DEP)
address space randomization
stack canaries

Avoiding: use safe function equivalents, language extensions

- Non Executable Stack: code can be injected in heap
return-into-libc possible
→ Return Oriented Programming: use existing code fragments
ROP call stack to execute these fragments

- Adress Space Layout Randomization: avoid hitting shellcode by
randomize addresses
different layout on each execution
on 32 bit system: brute forcing

compile time def: high level language
safe coding techniques, safe libraries
canaries

run time def: data execution prevention
aslr
guard pages

Cryptography

Shannon's law: Crypto is hypothesis, not penetrated
→ Cryptography vs. Security Engineering

Cryptographic goals: confidentiality, integrity, authentication, non-repudiation

Kerckhoff's principle: system should be secure, even if everything is public except key
Schneier's law: everyone can invent security systems that he can't think how to break

→ Unconditional Security (Perfect) : secure against any adversary
Computational Security (Conditional) : secure against computationally bounded advers.
Provable Security : proof that breaking is as hard as solving known problem

Cryptoanalysis: frequency analysis
linear cryptoanalysis: correlation between key and cipher
related-key cryptoanalysis correlation between key change on cipher
differential cryptanalysis correlation between part of cipher input output

- models of attacker: ciphertext only
known plaintext
chosen plaintext
chosen ciphertext

Symmetric key encryption: block ciphers, stream ciphers
↳ substitution + transposition (permutation): Feistel network

Block cipher modes:
Electronic Code Block: split message into blocks, encrypt each block separately
→ does not hide data patterns, same key, message → same result
↳ block replay

Cipher block chaining: encryption depends on previous block, initialization with XOR each plaintext with cipher of last block
→ CBC bit flipping attack: byte in next block altered
→ CBC padding oracle attack: returns for padding in last block / different error message for corrupt padding reconstruct intermediate value

↳ CBC MAC

Counter mode: generate encryption block with key, nonce, counter
XOR encryption block with plaintext block

Asymmetric cryptography: Trapdoor one-way functions

Diffie Hellmann key exchange: generate key without sending it over wire
based on discrete logarithm
→ MITM attack

Elliptic Curve Cryptography: discrete logarithms over algebraic structure
of elliptic curves over prime fields
→ shorter key for equivalent security
→ ECDH, ECDSA

RSA: based on integer factorization

Hash functions: maps message of arbitrary size to hash of fixed size (one-way)

cryptographic hash function: easy to compute for every message,
hard to generate message that has specific hash
hard to change message without changing hash
hard to find messages with same hash

Language Security

Postel's Law: Be conservative in what you send,
be liberal in what you accept

Input Path: Path of actions until input is displayed
↳ Synchronization, Layered Data, Compressed Data...

→ Polyglots, Error Correction, Packet in Packet

Polyglot: Source / input that is valid in different protocolls
→ binary polyglot: pdf, zip, jpeg

Protocols are languages: Input → Parse, Validate, Use
↳ more powerful than you think
↳ input recognition is undecidable if protocol too powerful
→ context sensitive languages are undecidable
→ Protocols are everywhere!

Postel's Law brings massive security implications:
→ Ambiguity is insecurity

Transmission Security: Packet in Packet: interference at right place
Browser in Packet: wrong client must not be connected

Compiler optimizations: functional equivalence != security equivalence
- dead storage elimination
- inline functions

Mobile Networks

Architectures: 1G, 2G - GSM, 3G - UMTS, 4G - LTE

Identifiers: International Mobile Equipment Identifier: phone
International Mobile Subscriber Identifier: sim card
Temporary Mobile Subscriber Identifier: session

Attacks: TMSI deanonymization
Network interconnection
SIM cloning
Decryption: decode session key
IMSI catcher (Stingray)

IMSI Catcher: tracking users, eavesdropping calls/text/data,
interrupt connection, attack phone

catcher acts like signal station to mobile phone
catcher acts like mobile phone to other signal station

catcher send changed location area identity
↳ mobile phone starts location update procedure
↳ catcher sends identity request (IMSI/TMSI + IMEI)

catcher downgrades connection security mode
↳ data is not encrypted anymore
↳ traffic MITM possible

→ possible because mobile network must not authenticate itself
↳ only mobile phone needs authentification

Catch an IMSI catcher:

Artifacts: Frequency, Cell ID, Location Update / Register, UMTS handling (downgrade)

- Frequency: frequency plans
- Cell ID: Cell database, Correlation with GPS coordinates
↳ usually Cell IDs are stable
- UMTS handling: Cell database: where is UMTS available?
- Traffic forwarding: Call Test

→ Mobile IMSI catcher catchers: Standard android API
→ Stationary IMSI catcher catchers: network of measuring stations

Embedded Security

Embedded System: not general purpose, but specialized application
often no OS, or highly specialized OS
interfaces to communicate with outside
various architectures
often used for critical tasks: airplanes, payment systems

-> in comparison to PCs relatively closed
↳ security by obscurity
- Proprietary implementation
- undocumented interfaces
- undocumented protocols

High-Level Security Analysis: analyze communication protocol
replay attacks
fuzzy testing
-> does it crash?

Low-Level Security Analysis: analyze implementation
provides in-depth insights
time-consuming!

Goals: console access, restricted features, alternative firmwares
extract secrets

Techniques: Firmware Extraction: downloadable updates, logic sniffing
↳ Analysis: static, dynamic: emulation, real execution
↳ hard with black box devices

Debugging: debugging interfaces for testing, production
↳ JTAG Test Access Port, debug server
Sniffing HW signals
Signal Injection

Countermeasures: remove debugging interfaces
SoC design
Tamper detection

Side Channel Attack: listen for relevant information on power usage, timing
-> timing, power usage, electromagnetic emanation, heat
↳ password check terminates on wrong char
-> timing

Fault Injection Attack: Clock Glitching
Voltage Glitching
-> alter device operation to skip password check

Microchip Reverse Engineering: Optically read out content
-> Preparation for other attacks