Cyber Security & Testing

Introduction

Security Assessment = proc of determining how effectively entity being assessed meets specific security objs * usually form of vuln assessment which scans infra/ID vulns * when people say they want pentest they mean this
Methods: Examination (study assessment object, e.g host/sys/network, to obtain evidence) * Interview * Test (exercise assessment obj under specified conditions 2 compare actual/expected behaviours) which can focus on target identification and analysis (network disc/port and service id/ vuln scanning) or vulnerability validation (password cracking, pentesting, etc)

Security Audit = Verification tht secure procedures are correctly carried out (checked against requirement) * so might be secure 4 what was built, but is what was built sufficient? etc

Security Testing

White box = detailed target info * insider/informed attacker perspective
Grey box = some info on target * most common type bc good accuracy, reduced cost, timeliness of feedback * get targets, area of testing, OPS sys, IDS/IPS in place etc.
Black Box = no info on target * use open source intelligence gathering * always validate targets as authed
Double blind = Testing defensive teams response * usually black box test * incl. Post-mortem and verification of what was/should have bee detected
Red teaming = anything goes * more time and cost * techqs like social engineering, phs tests, data extraction etc

Ethical questions: Sensitive data (own systems holding need to be secure) * crash a server * uncover illegal activites (duty to the law)

Management Methodologies & Legal Aspects

UK Certs/Standards: CREST (non-profit, ert 2 bcm trusted provider of proff services) * NSCS (check scheme of approved pentesters for HMG) * PCI (Payment card industry standard mut be complied w by financial services)

Repeatable, doce'd, assess methodologies
Adv: Consistency and structure 2 testing * clear on goals & wht ur gonna do (4 boundaries) * assign roles & responsibilities * Minimise risks (over-running, sensitive info disclosure, not gathering enough info, of affecting systems) proof u did what u said u would.
Disadv: fix steps before engaging → subverting understanding

Methodologies

Standard Stages

- Target Scoping = what testing/info given/ what limits/how long what buis objectives/ who will know * avoid scope creep, legal truble, false expectations * Project Charter (defines scope of project incl statement of work w bus case of project, description, key contraints, risks, roles & responsibilities) * ID Stakeholders (those who will be affected, positively or negatively)
- Information Gathering = talking or not to 2 the org, online sources, etc
- Target Discovery = produce a probable network topology (w live hosts/ID routes/ perimiter network mapping e.g router, firewall/ OS mapping)
- Enumerating Targets = expand on target discovery & find services running on them
- Vulnerability Mapping = ID vuln services * scan 4 known vulns → ID false positives and false negatives after correlating w other info → enumerate discovered vulns → estimate impact → ID attack paths & scenarios
- Social Engineering : check if allowed (can have ethical impact eg. corrosion of trust) * within reason (Scope and national laws)
- Target Exploitation = get access * find/develop/test concept code/tools

- Priv Escalation: passwords (cracking/in traffic) / cookies 4 sessions etc * perform prev stages again from new starting point
- Maintaining access: covert channels/rootkits/backdoors * usually used in double blind tests & red teaming
- Documenting and Reporting : you are paid for this stage really

Popular Methodologies:
NIST SP 800-155 :-Guide to Info Sec testing and assessment
Methods: Testing, Examination, Interviewing
Sections: 2 Overview (policies/roles/responsibilities/methodologies/techniques) * 3 Desc of teschniques for examination (doc review/log review/network sniffing/file integrity checking) * 4 Techniques for Iding & Analysing targets (network discovery/vuln scanning) * 5 Techs for validating existence of vulns (password cracking, pentesting) * 6 Apprch & Process for planning security assessment * 7 Factors key to execution (coordination, assessment, analysis, data handling) * 8 Apprch for after (doc finding, remediation activities)

PCI Pentesting guidance
For compliance with PCI-DSS * focus on pre-engagement/reporting * you'll likely breach contract if u dont follow
Areas: Pentester name/org/contact * Executive Summary (summarise: testing done, results, steps 4 remediation)* Scope (scope documented, how, definition of attack perspective, type of testing, contraints) * Methodology (is it stated? Is it best practice?) * Narrative (other things that came up) * Discovery * Results (incl. What further testing/remediation)

Penetration Testing Execution Standards (PTES)
Stages: pre-engagement interacation * intelligence gathering * threat modelling * vuln analysis * exploitation * post-exploitation * reporting
Threat-modelling: optimise assessment using context of org beyond intelligence gathering * build by feeding intel gathering back into pre-engagement phase * NOTE: some orgs think they know everything about their org and will skip this

Open Web App Sec Project
Focus on dev life cycle & detailed steps for testing * reporting guidelines are brief
Phases: Info gathering * config & deployment management testing * Identity management testing * Authentication testing (how get on system) * Authorization testing (can we manipulate system) * session management testing * Input validation testing * Error handling testing * Weak crypto testing * business logic testing * client side testing * mobile web service testing * cloud service testing * HTTP DOS attack testing

The Web Application Hackers Methodologies
Technical approch 2 test web apps * doesn't follow waterfall (seperates app logic, access handling, input handling, app tessting into discrete tests) * let app guide testing * covers unexpected attacks/indirect successes (e.g info leakage)

Open source Security Testing Methodology
Created the 'rav' to quantify security * attempts to put number 2 deviation from perfect security
Goals: full coverage * compliance w law in approach * consistent/repeatable resulsts * contains only facts derived from tests themselves
Issues: TF is perfect security * Takes in no context * human interaction = dangerous

Information Gathering

First stage of actual test * Allows PM 2 define/refine resources w understanding of team/ skills/effort * Important even if white box * Goal to create crucial fingerprint of org * attackers perspective on them * their security posture (tech, capability, exposure, security team) * shud b systematic & accurate * treat info collected as restricted (don't lower technical sophistication required 4 attack)

Approaches

Passive info Gathering = no interaction w target network (even website visits)
Semi-passive info gathering = appear like normal network traffic and behaviour
Active info gathering = connection 2 target network and systems & behave in unexpected ways

DNS

DNS Enumeration = query registrars about organisations DNS servers * passive * WHOIS leaves no trace on target networks * some restrict queries * info may be out of date * GDPR restricts available info

DNS Interrogation = query orgs DNS servers * Active * reverse lookups can be denied (or addresses configured not 2 respond to lookup) * target name server can be misconfigured 2 give info about internal network not just internet facing

Passive Subdomain Enumeration

Search engines * malware databases (search on domain) * dedicated subdomain DB * TLS certificate database * subfinder subdomain discovery tools (which uses many methods but brute forcing as well if necessary)

Target Discovery

*Obtain probably network topology * mostly thru communicating w target*

Things of interest in network: DMZ network behind firewall resolving outgoing requests * Internal DNS server resolve internal connection & house hidden subdomain

Network devices: Hubs (layer 1, repeat signal, no security, analyse using network tap) * Bridges and Switches (Layer 2, keep tables w MAC addresses and transport 2 correct part) * Router (Layer 3)

Link Layer

Handles: Ethernet (MTU = 1500b, 2 MAC addresses), WiFi (MTU = 2346b, up to 4 MAC addresses, handles fragmentation, bad cytpo protocols include WEP/WPA/WPA2) * Virtual LAN (switches group subnets of ports into isolated virtual broadcast domains via tags, hosts don't know what VLAN they're in, configuration complex and prone to errors, can be defeated by VLAN Hopping)

Address Resolution Protocol = translate IP address 2 network (Ethernet) address * host/gateway broadcast ethrnt frame w ARP req 'do u have this IP?'and answered w hardware address, response uni cast 2 requester * host/gwy maintains ARP cache 2 avoid sending continously (can poison w own MAC for DOS or MITM) * InARP (in reverse address lookup) provide MAC address 2 IP addr conversion * Passive host discovery (need to be on LAN) incl. listen 2 ARP request/responses/ send out own ARP request (e.g netdiscover tool)

Network Layer

Routing based on Routing Table info (IP address in packet header): tables use IP routing protocols based on trust (Border Gateway Protocol, Open Shortest Path First, Routing Info Protocol) * Routers implement security policy (they can drop unauth packets)

Border Gateway Protocol = for interdomain routing (share your route table – these are the addresses I can reach) * when router updates propogate table 2 1 hop (they can then choose 2 update and propogate etc) * No protection against modification (subject 2 all TCP/IP attacks like IP spoofing, session stealing) so any party can inject false BCP info (use rules like whitelist) * promiscuous propagation can be used for network fingerprinting

ICMP = Error messaging * can leak data (malformed ICMP returning when app alive, host live but port unreachable, etc)

Ping = host reachable (ICMP echo request) * absence of confirmation =/= confirmation of absence (some net perimeter devices block ping reqs) * IDS may detect ping sweep

Nmap

IPv6: address space big, brute forcing bad * use heuristics (if manually assigned might follow pattern: sequential, words in hex, IPv4 addresses in IPv6 space, number w service port)

Traceroute = use ICMP and TTL to obtain info about route between systems * sends UDP datagram w likely closed port num w increasing TTL (receiving back 'time exceeded' or 'port unreachable' in map network) * can lie (multiprotocol label switching routing technique fools it) * ICMP and UDP can be blocked (try setting UDP to DNS port – 53 -, try tcptraceroute)

Ping w record route enabled = provide round trip info of intermediate hops * support spotty

Transport Layer

Well know ports = 0-1023
Registered ports = 1024 – 49151
Dynamic or Private Ports = 49152 – 65535

Types of connection

- TCP SYN Ping:
  - If closed port: SYN → RST
  - If open port: SYN → SYN/ACK → RST
- TCP ACK ping
  - If host up: ACK → RST
- UDP Ping
  - If port closed, host up : UDP packet → ICMP port unreachable
  - If port open: UDP packet most likely dropped
  -

OS Fingerprinting

OS implement TCP/IP stack differently in recognisable ways

Examples

- FIN probe = TCP FIN received on open port → most OS wont respond, windows vista returns a FIN/ACK
- TCP ISN sampling = patterns in initial seq number can depend on OS
- "Dont Fragment Bit" = some OS set DF bit in IP header by default
- TCP initial window size = can be unique to OS
- TCP options = can be unique to OS

Can be done passively by listening to net traffic w/o connecting 2 target host * listen for TTL Value, window size Don't fragment bit * less accurate than active

Enumerating Target

Goal: *verify existence of target systems (follow up on target discover) * obtain list of comm channels (ports) whc accept connections*

Port states: Open * Closed (reachable, no service present) * Filtered (dont know if open or closed) * Unfiltered (returned by nmap, reachable, dk if open or closed)

Detection: Network IDS (avoid thru fragmentation) * Host Intrusion Detection System * NOTE: just bc detected by device doesn't mean someone will notice

Prevention: perimeter devices (e.g firewall) * Note: shouldn't rely on being able to stop scanning behaviour

Firewalls

Device/software implements org network access control policy

Network Layer – Basic Packet Filtering
Firewall: filters based on origin/dest addr * very efficient and scales well * limited functionality * config time consuming and error prone (specially in large network w segmentation etc) * most 1LOD (reduce what futher perimter devices have 2 deal with).

Network/Transport layer – Stateful Packet Filtering: based on origin/dest addr & connection status * computationally intensive checking performed only at time of connection setup * requires memory for each connection (TCP connection: SYN ACK flags det. State, connectionless UDP/ICMP: state of flow might be trackable thru matching header fields from in/out bound datagrams)

UDP Hole Punching = get around firewall traffic rules * A open session for UDP from port on B by sending first packet → A tells C → C tells B → B opens same way with A → session now open * used by skype

Transport Layer – Circuit Level Gateway/Proxy: Replicate out-going connections on behalf of internal client → allowing traffic 2 be re-written/ scrubbed * normalise traffic (denying some attack vectors and limit ops for remote fingerprinting) * rules out end-to-end encryption

Application Layer – Application Layer Firewalls: inspect actual app data & filter * cost and performance an issue * new modules reqd 2 support each app layer protocol * e.g big firewall of china

Network Security Devices Issues
Devices usually provide remote access 4 device managemnet tht can be exploited (e.g telent, web-based SSH, SNMP, TFTP)

Avoiding firewalls: Use Record Route 2 return record of successful path and Loose Source 2 set a route so u can avoid firewalls (or 2 determine exposed routes)

Scanning
TCP Syn Scan: RST after SYN/ACK * more efficient * less likely to be logged * requires raw socket priv (so admin access)

TCP Connect scan: Establish full connection * more likely 2 be blocked/logged * theoretically slower * uses TCP socket provided by OS (no admin)

UDP scan: less likely 2 respond to random reqs (if know protocol, u can solicit specific responses) * if port closed => ICMP dest unreachable error * No response → port open/filtered? * ICMP rate limited so slow

Timing Issues:
Rate limit for ICMP packets: Per host (# ICMP packets sent 2 specific host) * per second (# ICMP packets sent from this host in second) * burst (# per second but average timing so can tbe abused by front loading, takes between time stamps)

Avoiding: Strictest is per host so scan w outerloop = port, innerloop = host * to avoid per second, spoof from addresses in our block and intercept on way back (relies on IP address not resolving to 1 host, might not b true in ipv6)

Timing options available on NMAP

TCP Filtering Detection Scanning:
- TCP Ack Scan = determines if filtered by sending out of order TCP segment * RST sent if unfiltered * No response or ICMP = Error response if filtered * use to map out firewall rule set

TCP Filtering Avoidance Scanning:
- Abuse TCP RFC: any TCP packet not containing SYN/RST/ACK flags will return RST = closed, or nothing = open * Windows actually always returns RST (might be able to fingerprint using) * Types of scan: FIN scan, NULL scan, XMAS tree scan

- Fragmentation: frag across fields do don't trigger pattern matching (overlap on reassembly using different offsets) * requires 'default allow, deny if match' policy 2 work

Stealth Scans
No packets sent to target w scanner IP – instead use side channel exploits

TCP Stealth Scan:
1. Probe zombies IP ID
2. Make SYN packet, say is from Zombie, send 2 target. Depending on port state, Zombie IP ID may be incremented by target
3. Probe IP ID and compare to initial

UDP Stealth Scanning:
1. Forge UDP from Zombie 2 target
2. Send 49 UDP to closed port on Zombie thru 49 diff source IPs (prevent ICMP per host rate kicking in)
3. If target port was open, UDP response sent to Zombie
4. If UDP req received by combie, ICMP unreeachable returned
5. Probe a closed port on zombie. If no response then ICMP limit reached and UDP must have been sent (port open!)

Note: need to be able to spoof IP * No issue on LAN * maybe issue on internet (ISPs don't usually allow spoof, data centres are the same, routers may refuse 2 route packets)

Tools: Nmap (standard) * ZMAP (also gives JSON of responses not just existence of response) * MASSCAN (fast for large scane scanning, only does SYN, requires special network driver for peak performance) * NetBios on Windows will return all services host offers (on port 137 UDP, can be queried by nbtscan)

Security and Vuln Identification

Banner Grabbing: Telnet netcat * nmap -sV –script=banner * zarab (makes abnormal requests and sees how responded to, results in JSON)
Passive scanning: limited info * use sites like Shodon, Censys

Vulnerability Mapping

Sec Vuln = flaw or weakness in sys design (in is spec), implementation (bugs in code or integrated systems), or operation and management (bad config or deployment of sys in specific env) that cud be exploited to violate the system's sec policy

Query known vuln databases * DHS National Vuln database etc * Question: robustness, credibility, immediacy (up to date?)

CVE identifiers = unique common identifiers for known vuln

vuln taxonomies = attempt to catagorise vulns * used by some security assessment tools * some create unhlepful perceptions (CVSS gives scores to vulns that imply priorities that aren't realistic)

Manual Steps: Banner grabbing → Database searching
Automated tools: Commercial (Nessus, Core Impact) * Open Source (OpenVM – forked off nessus) * Might b blocked or give false positives
Note: So do combo of both

Target Exploitation

Vulnerability Verification = verify vulns identified (not enough 2 say they could exist)
Exploit = software/techng 2 take advantae of a vuln 2 violate a security policy * normally online * rarely have time 2 develop one * all-included vuln exploit tools exist (e.g metasploit) but make sure these are themselves secure

Forms Of Attack: Passwords

- Default Credentials = use factory-set default creds * tools: Zctgrace/changeme (for FTP/HTTP/Monogdb/MSQL…)
- Brute force auth = guess * tools: brutus, THC Hydra, enum, nmap * Good targets: telnet/FTP/SMB/Web apps/SNMP * simple * easy 2 detect and protect against (lockout, disabling services, auditing & logging) * may be feasible due 2 misconfig (or unmonitored by default services)

Software Security

Vulns usually from: assumptions made (about env and operation) * programming errors

Buffer Overflow

Store data beyond boundaries of allocated buffer * Overwrite return address 2 execute code not normally executed or shell code u also overflowed the buffer with
Mitigation: Safe libraries * Stack protection (Canaries) * non-executable mem locations (NX hardware based) * Address space layout randomisation (ASLR software based)
Defeating mitigations: Canaries (if predictable; I attacker can read/write arbitrary memory then read and overwrite w what read) * ASLR (detec mem layout by reading addresses e.g format string attack) * NX (use Return Orientated Programming to string together code already in sys libraries)

Integer Overflow

Pass integer limit it wraps around (mod 2^32) and use 2 bypass checks * hard to detect possibility (usually present when comparing how large a buffer is 4 malloc etc) * more likely 2 cause malfunction thn exploitable attack

Mixing Data and Controls
If u use in-band signalling, the signal can be used 2 adjust expected behaviour (e.g point 2 point protocol)

Format String Attack
Dump from stack memory * printf('\x10\x01\x48\x08', %x%x%x%x%s) - %x increases internal stack pointer to top of stack (where our provided addr is) now 5S reads data from given memory location * num of %x needed varies

Software Testing
White box = Source code review * manual or automated checks
Grey box = spectrum of source code access and app interfaces * reverse code engineering (disassembler, decompiler, debugger)
Black Box = use of software system interface * Functional Testing (test against req spec) * Robustness testing (ill-formed test cases 2 verify reliability and security)

Fuzzing
Performed on software interfaces * trigger faults in app (sending unexpected data, setting unexpected app config/environment) * black/grey/white box * look for: valid responses (incl. Error mssg)/ Anamolous responses (fatal flaws, corrupted msgs etc)/ crash & unexpected failure

Types
- Random fuzzing = generate random test data
- Mutational (Dumb) fuzzing = randomly (or using heuristics) modify requests 2 create anomalous valid data * simple 2 write * hard 2 work w challenge and response protocols * e.g Radams, zxuf
- Generational Fuzzing = generate input based on templates and mutate 2 be off w awareness of exact data boundaries * works on complicated protocols * time expensive (writing templates) * e.g codenomicon, peach, sully
- Feedback Fuzzing = obsrv prog behaviour & adjust input according to strategy * if input triggers new code path, continue down that path (more complete code coverage) * no need 2 write protocol specs (fire & forget) * recomplilation often needed 2 work well * NOT BLACK BOX * doesn't like randomness (will hang on CRCs/hashes) * e.g AFL, libFuzz
- Symbolic Execution (Whitebox fuzzing) = mark input w symbols associate w possible execution paths and enumerate * in theory automated * experimental

Uses of Fuzzing: mostly in software testing during/after dev, and pentesting (app levl and network protocol w understanding of protocol specs) * Can be easy 2 detect (particularly if prog keeps crashing) * e.g BED Bruteforce Exploit Detector (designed 2 fuzz plain text protocols against buffer overflow/ format string bugs / integer overflows / DoS conditions etc, supports FTP, SMTP, POP3, HTTP, IRC, IAP4, LPD, SOCKS4, SOCKS5), e.g JbroFuzz

Exploit Project Manager Issues
Scope creep (messing schedule) * compromises are normally asset misconfig which is harder to find/takes more time (soft devs can find same exploits u can) * stake holder management (expectations e.g uncovering vs demonstating, communicating channels 4 discovery of severe vulns, issues regarding outages and complications)

Web App Security Testing

Why attacks effective: Accessible interface * complex apps error prone * great availability of data * high impact (downtime, data leakage, reputation) * misconfig is rampant

HTTP = stateless (use cookies; URL-rewriting; hidden HTML variables; client keeping state) * Encoding (URI; unicode; HTML; Base64 ; Hex)

Handling User Access

Authentication: password/cert/smartcard/token * login/registration/account recovery * pass change
Attack: Leakage of user info * Guessing password * Bypassing auth

Session Management: bind actions w authed sessions
Attack: Guess session identifier * poor handling of tokens

Access Control: App enforce access control for resources
Attack: complex reqs => unexpected paths * errors in configurations

Handling User Input

Attack vectors: leak info * crash app * provide unauthenticated and unathorised access 2 data & other resources
Input vectors: HTML forms * URI * hidden form data * cookies * HTTP headers
Approaches 2 Handling: Blacklist (limited due 2 encoding and new attacks) * Whitelist (may be excessive) * Sanitisation (remove, encode, escape pattens) * safe data handling (ensure processing of data is secure; parametrised queries for database access, no direct passing of user input 2 OS command interpreter) * Boundary Validation (every boundary untrusted, check and sanitise etc) * Multilevel Validation (sanitise recursively) * Semantic Check (correct circumstances e.g not referral)* Canonicalisation (convert and decode data to a canonical form e.g using pre-determined character set)

Handling Attackers: Error handling & msging * maintain logs and audit * Alert admin * account lock out

Managing the App: Manage user accounts/roles * Monitor and Audit Functions * Diagnostic & Config facilities * Security policy
Note: Management Interface may be target for attack

Web App Pentesting

Web App Mapping = mapping/identification of functionality

- Enumerate content and functionality: web scraping * discover hidden content (brute force, inference, public info) * functionality paths * hidden params
- Analyse App: core functionality (logging, errors, links) * tech and sec mechsnisms (auth, session state, forms, scripts, cookies, file extensions, dir names, session tokens) * location of data processing (url strings, forms, cookies, HTTP headers)

Vulnerability Identification = vuln scanners

- OWASP [open web app security project] top 10: Injection (untrusted data sent to interpreter as part of command/query) * broken authentication * sensitive data exposure * XML External entities (eval external entity references from poor XML processor congig) * Broken access control * security misconfig (error message dislosing, open cloud storage) * Cross-site scripting * Insecure deserialisation * using components w known vuln * Insufficient logging/monitoring (missing, bad integration with IRT, average 200 dys to detect data breach)
- Web app vuln scanners e.g niklo2, burp suite etc

Verification = Exploit of Vulns

Web App Attacks

Cross Site Request Forgery [XSRE]
Inject HTML that, when victim reads, causes their browser 2 submit request to a vuln app * Token Misuse
Counter: Session tokens transmitted via hidden HTML fields * sensitive actions multi stepped

Cross Site Scripting [XSS]

- XSS = Inject script to web page when read by victim will cause their browser 2 run script and perform some action
- Reflected XSS = send URL to victim w script in URL (2 get cookie info by sending to attacker owned site w cookie in URL)
- Stored XSS = app stored attacker script in DB and loads for victim
- Dom-based XSS = Mod dom tree in victim browser (so HTTP respone by server doesn't contain the attacker payload)

Detecting vulns: automated tools (better 4 reflected than stored) * POC strings as parameters and see if response modified
Prevention: Cookies marked HttpOnly so cant be access by scripts (prevent session hijacking) * Input validation * Output validation (canonicalisation, sanitsation)

Code Injection
Use interpreted languages 2 provide instructions rather than data as input e.g OS Command Injection * Shellshock = using vuln in how env variables are processed
SQL Injection vuln detection: Cause error messages thru malformed input * look for alternative interfaces 2 bypass encoding * Blind SQL = no results displayed (so craft input for yes/no queries and see if permitted) * automate using tools
SQL Injection prevention: Parametrise querires (so DB distinguish between code and data) * Escape user-supplied input * Strict validation

Cryptographic Attacks
Bliechenbacher Attack agains PKCS#1 v1.5
Leverage padding scheme: where random string of bytes has porb 2^-16 to 2^-18 2 follow padding format and error messages allow attacker 2 use leaked info to reveal a message m
Padding scheme: *c = (0|2|r|o|m)^e mod N*
Attacker send for many s: *(0|2|r|0|m) . s mod N*

Length Extension Attack
Mac for auth'd requests: *H(secret|m)*
H = merkle demigard construction

Attacker send: *H(secret|m|blahblahblah)* from known *h = H(secret|m)*

Mitigation: use HMAC construction or SHA-3

Physical Security
If disk unencrypted, boot into own OS and read from disk (or modify) e.g password hashes, config files, etc

ColdBoot attacks
Decryption key only in memory when running and info in RAM not lost when power cut (lasts longer when transistors cold: deep freeze → remove RAM → plug into another machines → dump contents
Finding key in dump:

- Data in ram has low entropy, key have high (Easy to spot)
- DES: derive its round keys in repeated patterns w trivial error correction * algorithms online * quick easy * memory unintensive
- AES: making educated guesses rather than brute forcing

I/O Ports
Autorun particularly dangerous * Can exploit using LNK vuln, PDF preview, Image preview

- BadUSB: misrepresent USB as MCI controller, imitating keyboard, to enter commands stealthily * can only analyse thru sandboxes
- Firewall & Thunderbolt serieal ports allow devices to bypass OS and access memory directly * performance enhancing * blatantly bad idea

Privilege Escalation
Note: Might not b required 4 project bc cost to org/tester

Password recovery
Cracking with real time:
Brute-force (A^n hash ops for A = alphabet, n = length of hash) * Dictionary Attacks * Hybrid
Time = A^n , Memory = 1

Pre-computation:
Time = 1, memory = A^n
Reduce Storage using Rainbow Tables:
Want: increase length of chains t to save space, but that increases chance of merging (and u may not cover all keys which is hard to detect)
Actually: Generate more tables using different R functions 4 high prob every key in at least 1 table * Opt number of tables = length of chain = number of chains = number of keys^(1/3)

Choosing Password Hash Functions:

- Make A and n large
- Make H() slow: iteration
- Use password salts: w per-user diversification * not harder 2 brute force * harder 2 precompute
- Make password function memory hard (w large internal state)

Maintaining Access

Save from: lost connection 2 host * cost of exploit 2 high 2 repeat * no longer possible to exploit (patched, firewall rule change, etc)

Covering Tracks: Manipulate log files (can be flagged) * Disable logging and auditing (log files might alert in real time so need 2 intercept/ discard before written, easier if written 2 remote) * Hide files (innocuous names/extensions)

Backdoor: incl reverse backdoor (connect out instead of listen) * e.g netcat (TCP/UDP, detectable bc all data cleartext, can use SSH/socat etc for encrypting, if reverse backdoor set shell as reconnecting script)

Rootkits
User-level (Ring 3): less challenging to write * easier 2 detect * modify processes via code injection (DLL injection) 2 filter results of functions used to detect * modify file or registry keys
Kernel-level (Ring 0): harder 2 detect * challenging 2 write * modify syscall tables or syscall/kernel function can intercept syscalls (syscall hijacking)
Hardware: overwrite BIOS firmware or replace firmware of hard disk 4 persistence

Detection: integrity checking (behaviour analysis) * signature/anomaly based

Note: Rarely use low-level rootkits, you could really fuck up.

Covert Channels
e.g TCP/IP headers * DNS lookups (returned IP A.B.C.D translated 2 commands in look up table)

NOTE: Mostly you'll just use backdoors * Leave nothing behind! * Ask what's in scope

Documenting And Reporting
Close Project = release of report w final risk assessment, detailing all vulns and exploits, remediation options.
Close procurement = releasing all other docs that the pentesters found/were given

Report overview:
Mixed audience (high-level and low level tech explanation)
Structure: Title * exec sum * report body * conclusions * references * appendix
Report content

- Description of larger network: high-level * how it works (devices, topology, etc)
- Out-of-scope issues: what could have been investigated but wasn't (bc limitatio of resources, potential damage, out-of-scope)
- Findings: Vuln ID'd * Vuln verification (enough detail so can be repeated) * shouldn't contain customer sensitive info
- Remediation Options: not usually purpose of report * high-level
- References Section: vuln databases/protocol spec etc
- Appendix: for detailed exploites

Risk Eval
Pentesting part of larger risk management: they need likelihood of occurance (motivations, exploit difficulty, nature of vuln) and impact * customer chooses treatment (assumption, avoidance, limitation, transference)

Data handling During Pentest

Docs incl. Protocol specs, reports, emails, attack record

Should you retain?

Pro: Use to reconstruct work * more accurate replies to later queries

Cons: risk of losing client data * cost protecting sensitive data

Either way need 2 communicate clear policy: cover access, archiving method, location, destruction, protection (encryptuon, secure location etc)