Malicious Software

Introduction

Malware – computer soft tht violate sec policy

Taxonomy – malware classification 2 communicate in precise lang: functional (distinguishing features ; goals) * behavioural (actions performed ; how accomplish goals) * authorship (author/tools ; attribution)

type > family > sample (instance)

Trojan horse: overt & covert purpose * gemini (android TH)

Root kit : TH that hides itself
Early: install backdoors * change system programs tht report on sys setting (list contents of director, list network conns etc) * Linux Rootkit IV (gave priv shell if specific password)
Counter: run non-standard programs for that info (access directory directly etc) 2 bypass system program using sys calls and info from kernel
Later: altered parts of kernel to filter returned info * Knark Rootkit (hijack sys calls tht examine file system and net connections)

Virus: inserts itself into other files and mayb other action [payload] * requires spreading of host file to propogate

Two phases

- Insertion phase (needs to eb present, but might have condition where doesnt execute)
- Execution phase

Virus types: File infectors (direct infectors or memory resident infectors) * Boot sector viruses * Multi-partite viruses

FI-DI-Overwriting – overwrite section of program file * may break infected program * if bigger than infected, none of original file left

FI-DI-Companion – call virus file first then pass control 2 intended file * virus make new file w same name as intended but diff extension (or hiding original if COM ext already taken) * user ignorant * DOS calls in order: COM,EXE, then BAT if intended ext not given

FI-DI-Parasitical – edit code so intermingled with original w/o breaking * prepending (insert at beginning of exe, shift other code down) * appending (at end, insert JMP at beginning) * fragmenting (virus intermingled)

FI-MRI – dont infect files directly but wait in mem until host prog executed then infect it * can b done w DOS's TSR (terminate but stay resident) system call.

Boot sector viruses – hijack 1st instruct of boot sector to point to virus and pass control back to boot-sector code after exe * boot sector only 512 bytes, so usually uses other sectors of disk to hide its code * target master boot record (MBR) or Partition Boot Sector (PBS)

Multi-partite viruses – infect both boot sector and files * not in particular order

Infecting file types
Executable programs – exe when infected program run * viruses → normal code.
Device drivers – exe in kernel mode
Archives – insert trojan horse in 2 ZIP file * social engineering
Dynamically Linked Library – exe when prog loads/runs infected DLL
Macro viruses – seq of instructs interpreted not directly executed * execute on any sys with interpreted * limited/enabled by features of macro lang * can infect exe or data files * (Melissa virus – infect word and sent using user's address book) (Concept – infect NORMAL.DOT, templ for new word doc, payload does nothing)
Script viruses – like macro but higher level programming technique constructs and interpreters, not tired to applications. * vb Is both macro and script

Early viruses: Elk coner (infected apple II, spread by floppy) * Brain virus (IBM, spread by floppy) * Ghostball (infected MS-DOS, 1st multipartite virus)

Worm: copies itself from one prog to another * non-infecting * uses network connections
Three phases

1. Target selection
2. Propagation
3. Execution (maybe)

Examples; morris worm (three strategies to open remote shell * install bootstrap code: weak passwords, buffer overload in finger daemon, vuln in send mail * bootstrap code transferred full code)

Downloader – download malicious content via net connection

Dropper – install malware which it may contain [single stage] or it may download [two stage]

Backdoor – to bypass auth * **RAT** incl allowing ctrl of compromised host

Rabbit viruses – absorbs all of some class of resource

Logic bomb – violates sec polics when some external event occurs * (march 2013 logic bomb wipes hard drives and MBR of > 3 banks & > 2 media companies simult.)

Spyware – TH records info (immediately sent or stored for later sending) * invisible to user and sys * (Pegasus – apple iphones)

Botnet – botmaster control collection of bots via C&C server

Ransomware – inhibits use of some resource until ransom is paid

Wiper – wipe drives/data * (Shamoon: dropper, wiper, reporter)

Cryptominer – use comp resources for crypto mining * (coinhive: monroe cryptocurrency miner, via web servers, uses up processing power of visiting browsers)

Grayware – annoying/undesirable but less serious or troublesome than malware. Altho can still affect performace of computer and may intro sig security risk * Adware * Bloatware

Adware: TH gathers info 4 marketing and display ad * may be benign if user coneents * transmission of data might have covert purpose

Levels of (mobile) adware :

- Low severity behaviour: display no transmission
- Medium severity: transmit low risk info
- High severity (madware): transmit high risk info * ads aggressive

Browser Security

Not web security

Potentially dangerous features

Active X: software applets tht can be re-used by web browser

Risk: controls mayb malicious

Defence: check origin/sig * browser dedicated facility 4 managing * dont install if unnecessary

Java applets: active content for websites * convenient to run * exe in sandbox

Risk: poor implementation → bypass restrictions

Defence: ensure doesnt set CLASSPATH (path of classes that run with relaxed security) * dont enable if unnecessary

Plugins: like active x but cant be used outside browser * can handle files like DOC or PDF

Risk: programming flaws like buffer overflows * design flaws like cross-domain violations * can be intentionally malicious

Defence: check origin/sig * dont install if unnecessary

Cookies: small file on local drive w info about web sites visited and rel preferences

Risk: may have sens info * normally not encrypted * some are persistent and dont finish when session closed

Defence: dont enable if unnecessary * cancel cookies frequently

Javascript: make websites more interactive * standards restrict actions like accessing local files

Risk: attacks initlialising field and auto submissions

Defence: Check id of website using JS * dont enable if unnecessary

VBScript: like JS but only usable on IE

Helpful features

Content Security Policy (CSP): only execute scripts from pages tht the server admin says is valid * help protect against vulns like XSS

HTTP Strict Transporting Security (HSTS): browser only access over HTTPS pages tht server admin sets to carry HSTS header * thwart SSL stripping attacks

X-Frame-Options: prevent reframing of pages that server admin sets 2 carry approp X-Frame-Options heading * protect against clickjacking (when css overlays a frame)

Sources of auth: US-CERT (outdated, obvious) * NIST (removed browser sec chapter, outdated, obvious) * OWASP (brow sec proj set to inactive) * RFCs (valid for ref, not frequently updated)

Browser implementations:

Vendors: may ignore guidelines * we cant see closed src browser * analysing open src browser is difficult
Incorrect execution of https: e.g heartbleed (buffer over-read issue)
Incorrect display of padlock (opera mini split encryption over 2 legs, not end-to-end but still padlock displayed)
Insecure storage of user passwords: browsers store in clear, or encrypted with key available in another file * user can set master password or use a 3-rd party software vault

Web certificates:
If priv key of CA lost: Other keys not compromised * Trust compromised (those certified by potentially untrusted authority)

Root cert: Attacker can't decieve browser unless can install fake root cert on bic browser that certs their fake page * write protection important *

Trust: User may choose to temp or permenantly trust a claimed but uncertified identity

Types of invalid certificate:

- Unknown or untrusted certificate issuer
- Expired certificate
- Revoked cer
- Mismatched cert and id

* not all make webstie malicious *

Check status: check revocation status via seperate protocol which cert specifies (if cert doesn't specify, status = unavailable) → revocation status may fail (status = unknown)

Malware Functionalities

Infection vector

Method of propagation or infection

Phishing – impersonate legit entity to get infor with auth

Homograph attacks – unicode domains problematic bc unicode chars hard to distinguish * some protection mechanisms bypassed when every char replaced with char from single foreign lang * web browsers use punycode convert non-ASCII codes into ASCII codes for display (some may put xn-- prefix first but some dont)

Spearphishing – targeted phishing

Web vulnerabilities: compromised websites (mal scripts injected ; credentials compromised) * code hosted on malicious site (URL redirection) * drive-by-download (unintentional download; take adv of sec flaws due to lack of updates) * watering holes
Malvertising (compromise legit advertiser/ad agency; register to legit ad platform and spread mal code ; pretend to rep legit brand)
Drive by download – exploiting APIs for browser plugins to enable downloading and exe of arbitrary files * exploiting vulns in the web browser or plugins
Cross-site Script (XSS) – mal script injected into trusted web site to be exe by different end user * browser has no reason not 2 trust (allows access to sens info)
Cross site request forgery (CSRF) – exploit end user trust in browser * make broswer perform unwanted action on trusted site to which user is curr authenticated

Exploit kit – bundle collection of exploits

Fileless malware – use microsoft utilities w/o relying on compiled executables (powershell scripting ; windows management information)

Malware functionalities

Downloader – downloads another malware component

Dropper – embeds additional components within self & extracts them to disk when exe

Keylogger – intercept and log keystrokes (hook procedure to monitor: notify → log)

Replication – infect removable media and use one of: 1) autorun 2) trick user into clicking file 3) exploit vuln to run)

Persistance – adding entry to run registry keys (run at startup) * create scheduled task utility * add malicious binary in startup folders * modify winlogin reg entries (modify default user's shell) * abuse image file execution option (used to launch the mal prog) * DLL search order hijacking * COM hijacking (hijack com references so mal code executes instead of legit soft) * create service (background task launch automatically)

Code injection

Code injection = inject code into process and exe in context of that process (legit process, bypass security products, without exploiting vulnerabilities 2 interact with OS)

Remote DLL injection = force to load malicious DLL (get process id → copy DLL pathname into its mem → call loadlibrary on specified DLL → dealocate mem and close handle)

Remote executables/shellcode injection = inject into target proc directly w/o needing malware 2 be loaded on disk (get process id → allocate mem and copy malicious code into it → exe threat with pointer 2 address of injected code)

Hollow Process Injection = load legit process to act as container for hostile code →at launch deallocate legit code and replace with mal code * bypass sec products and remain undetected

Process Doppelganging – temporarily mod trusted file in mem without committing changes to disk,

Hooking Techniques – replace entries in import address table to block calls/ monitor intercept and modify input params passed to api/ filter output params returned from API * IAT hooking (inject DLL → locate IAT by parsing exe imag in mem → id entry of function to hook → replace address)

Inline hooking = API functions modified to redirect the malicious code * 1st instruction rewritten to jump statement usually * *size of instruction to overwrite/instruction writing with matters*

Android [and IOS] Malware

OS Security features:

- Market level – app review * app signing
- System level – access control * sandboxing * permissions * full disk encryption

App-review - from distributed via official markets * many apps downloaded not thru official store.

android: auto analysis(fast not thorough) – 8 million downloads of potetent harmful apps thru google store

apple: auto and manual (thorough not fast)

Sandboxing – apps execute under Minimum Privilege Policy (only access own directory, OS Mediates access to all other resources)

Android: each app exe as diff user * SELinux sandbox

iOS: app runs as user 'mobile' * can't access other app data * sens API calls allowed thru user granted permissions or entitlements

Permissions – many request more than they need due 2 confusing permission names ; 2 prepare 4 future updates ; devs copy & paste

Android: permissions declared at install time * dangerous ones requested on run time * include usage of SMS and phone

iOS: no SMS no phone * granted on-run time * apps have to be prepared for denied permissions * permissions modifiable by user at any points in time

Fighting malware

- Static – Reverse Engineering tools (Androguard , APKtool, IDAPro, Radarez) * knowledge/libraries (virustotal)
- Dynamic
- Hybrid

Attacking Android

Exploits – difficult to detect and mitigate * expensive 2 dev

Abuse permission system – easy 2 detect (sigs easy to blacklist) * easy 2 develop (malware generators ; repackage old malware)

Application collusion – two or more apps work together 4 malicious action couldn't do on their own.

Communication channels:

- Overt [exploiting OS API calls and info leaks] (intents, content providers, external storage, shared preferences)
- Covert [exploit APIs or features offered by OS

4 comm between processes] (audio settings, settings broadcasts, file locks, process and socket enumeration, resource usage)

Identifying collusion Difficulties: analysis of one app is resource intensive and app combinations create exponential set of possibilities.

Identifying collusion approach: lightweight analysis 4 single app (look for access, send and receive signature ASR)→ method to combine analysis (model in prolog and define collusion rules) → resource intensive analysis on apps w collusion potential

Mo plus SDK – software development kit w functionality that can be abused 2 install backdoors on devices tht have installed the developed app * developed app installed → launch unauth web server tht can accept requests from any source (attackers cn read sensitive info, install apps, add contacts)

Botnets

Bot master controls network of compromised boots via C&C server

Uses: scanning * DDOS attack * SPAM campaign * click fraud * info theft

Command and Control

IRC - command published in IRC channel: push style (bot waits) * if C&C server isolated, control lost (tho could use mult C&C for redundancy)
HTTP: pull based (bot queries) * difficult to block at network level (with firewalls) and at DNS level (with domain blacklisting)
p2p – distributed C&C and/or commander address published in P2P network * protocol based on overnet (search 4 keys [IP & port of proxy] in p2p network to locate proxies → connect to proxy and wait 4 commands → proxy forwards commands from master to worker) * master servers on bullet-proof hosts * workers w best resources elected 2 proxies.

Locating C&C servers
Hard coded IP: (easy to thwart)
Bullet-proof hosting: users can do virtually anything * unlikely to coop with law enforcement * high rate of turnover * can be blacklisted
Dynamic DNS: Dynamic domain name service links domain name to dynamically changing IP address * easy registration * limited to second-level domains only * can be taken down by providers * can require additional software on the bot

Fast-Flux: numerous IP addresses associated with single domain name (round-robin style DNS with short TTL)
Single-Flux: IP addresses are addresses of comp machines w role of 'flux agents' * agents redirect reqs & data 2 another back end server [fast flux mothership] * agents protect mothership from discovery * agents easily replaces (resiliency)
Double-Flux: glue record also changed constantly * comp machines are authoritative DNS and their IP address also being fluxed
Fast-Flux service network: problematic agents replaced * id of code components of the infrastructure is well protected * botnets use multiple domains (not enough to shut one down)

Domain flux – bots periodically gen new C&C names [using seeds like system time] * botmaster registers one of these domains and 'responds properly' so recognised as C&C server * defenders would have to register all domains 1st to take it down * diversion by using domain generation algorithm to make lage numbr of potential names but only a few are motherships * deterministic psuedo-random generator
Pros 4 attacker: evades blacklisting (2 many names) * domain reputation systems are useless (domain names quickly discarded)
Cons 4 attacker: produces lot of noise so easy to detect DGA capable malware * if defender can reverse DGA component they could register future domains and point it to sinkholes

Example: Torpig
Injects itself as DLL * steals sens data using HTML phishing injected thru http * uses encr HTTP as C&C protocol * uses domain flux (same DGA ; 3 fixed domains if all else fails ; until successfully connects: generated weekly → daily → fixed)

Rootkits

hide presence of malware from sys admin
want 2 hide: files * network connections * registry keys * processes * services

Types of rootkit:

- Ring 3 – user
- Ring 0 – kernel
- Ring – 1 – hypervisor
- Ring -2 – BIOS/SMM [system manager mode]
- Ring -2 – chipset

Hooking

Ring 3 – user

Mod Import Address Table: (lookup address 4 DLL point to malware instead)

Ring 0 – kernel

Interrupt Descriptor Table
Change wat routine called when interrupt triggered (run with higher priv) * In windows KiSystemService [0x2e] supposed to be called, point to malware instead * will not b called if syscall is used instead of interrupt * cant filter return data to help hide rootkit * easy 2 detect (IDT[0x2e] != KiSystemService…]

SSDT Hooking
Mod System Service Dispatch Table (addresses to sys calls) * can filter return data * easy 2 detect (are entries pointing into kernel space?)

Run Time patching
Manip memory image of module rather than binary file on disk (which would leave traces) to pass control 2 malware * harder to detect (no common hooking point)

Direct Kernel Object Manipulation
in memory alter kernel structure * hide malware by altering linked list containing active threads and processes so it points around malware * doesnt stop malware running bc scheduler doesnt rely on list

Bootkits
Infect MBR or Volume Boot Record * rootkit remains active even after system reboot

Ring -1 – hypervisor

A Virtual Machine Based Rootkit installs itself beneath existing OS and then hoists that OS into VM * sys calls now pass thru the rootkit

Ring -2 - BIOS/SMM level

UFFI/SMM attack allows for installation of rootkit on firmware * SMM rootkit installed by redirecting to our SMM handler from I/O Advanced Programmable Interrupt Control * avoid detection by returning control to the local APIC (Advances Programable Interrupt Controller)

Ring -3 – chipset rootkits

most vPro chipsets = over-priviliged and store sofware (e.g Intel Active Management Tech) * allows backdooring a system * survives OS reinstall * some AMT code executes regardless of whether enabled in BIOS

Malware Analysis & Evasion

Old-school: samples RE → signatures * heuristics (code execution starts in last section ; incor header size ; suspic header size ; patched tbl of imported funcs) → signatures * AV has database of em
Now: rep * string sigs * suspic behaviour (~ dynamic)

Static Analysis

Disassembly: take blob → sep data code → trans machine code 2 mnmemonic instrucs * depends on quality of dissasemble alg
Issues: code & data in same space * variable length instruc * indirect control transfer [arg specifies address] * info might dissapear after compilation [var names; type info; macro & comments] * diff to detect functions and function params

Dissably algs

Linear sweep algorithm
Algorithm: Locate instruc (starts and ends) * everything in code section assumed as machine instructions * usually start from first byte and linear
Pro: complete coverage of program code sections * when error in disassembled, eventually ends up re-synching
Con: no control flow understanding * compiler mixes code and data

Recursive Traversal Algorithm
Focus on control flow
Instruction classification : Sequential flow (pass exe onto next instruct that follows ; e.g add, move, push, pop…) * conditional branching (in static both disassembled, in dynamic just one executed) * unconditional branching (jmp) * function call * return
Pro: distringuish code from data

Con: Indirect code invocations (can use control flow graphs)

Code flow graphs: directed graph of all paths program might traverse 2 perform sophisticated prog analysis s.a
optimisation: (constant propogation ; deadcode elimination thru reachability anal ; backwards slicing ; chopping)
slicing: [backward slicing/chopping] 2 see what operates on sensitive database

Call graphs (callee-caller relationship)

Encrypted viruses: anti-detection technique * encrypt virus payload and decrypt at run time * BUT AV can just focus sig on the decryption module

Oligomorphic viruses: like encrypted v but decryptor swapped out in new generations * BUT decryptor comes from finite set so it is still possible 2 use sigs

Polymorphic viruses: like encrypted except uses variable encryption keys, dynamically adjusts decryptor layout, and randomly insert junk instruct to produce million of variations of the decryptor

Metamorphic viruses: doesn't need to encrypt the payload * apply polymorphism to payload so new gens look diff but semantically equiv (1) analyse own code and split into blocks → 2) mutate blocks separately)

Code obfuscation: obfuscation (preserve semantics) * junk insertion (nonsense partial instructs not reached at runtime) * branch functions (replace call w indirect jmp) * overlapping functions (any byte following first could be re-used as start of anohter)* packer (encryption tech that doesn't require particular utilities on victimes machine) * emulation technology (convert malware binaries to byte code only can be interpreted by included VM ; byte code mutated each sample ; difficult to reliably reverse)

Fighting packing
Algorithmic unpacking: implement in AV routine semantically equiv to 1 in malware
Algorithmic agnostic unpacking: use dynam analysis and emulate until termination of unpacking

Dynamic based analysis
monitor code as executed to reason about execution and precise sec analysis

Approaches: debugging * add semantic preserving code * taint-tracing (taint input and follow progress)

Watch interaction with env on linux: lsof (file detail) * netstat (contents of network strucs) * tcpdump, wireshark * itrace (intercept calls made to/from dynamic library)

Watching interaction with OS: sys call tracer (calls and recieves sigs intercepted)

Debugging: breakpoints * watchpoints (stop when val of var or expr changes) * catchpoints (stop when event occurs) * monitor single process * single-step * analyse cpu env
Linux: gdb
windows: ollyDbg

Sandbox
goals: visibility (c as much of exe as poss) * resistance to detection * scalability (effective efficient, one sample shouldnt effect the next)

Visualisation – a level of abstraction * resource virtu or platform (emulator/simulator)

Emulator – duplicate functions of sys A using sys B that behaves like A
Simulator – provide realistic imitation of an abstract model or system [maths/physics]

Sandbox: sec mech for sep running programs (provide controlled resources, based on virtualisation [emulation])
pro: run untrusted * no risk to host machine* automate analysis * process high vol of mal * get actual executed code
Cons: mb expensive * some code might not be triggered * might be detected

Sandbox evasion
sleep (anti-sleep: cuckoo skips steep steps exe in first seconds) * reverse turing test (anti: cuckoo emulates human interaction) * examine resources * delete trampolines (inline hooks used by cuckoo 2 monitor api calls are easily detected) * red pill (program capable of detecting if exe in emulator)

Ransomware

- low grade – scareware
- middle grade – browser or screen locker
- most dangerous – encrypting

Stages of infection:
Break in: phishing * exploit kits * self propagation * exploit server vulns * malvertising
Installation & persistence: copy itself * edit registry 2 auto start @ boot
Key generation: locally (e.g cerber) * from C&C server (chimera)
Encryption: delete original files & shadow copies & encryption keys.
Levels of skill:

1. Encryption key derives so predictable
2. Key recovered from files or memory
3. Key recoverable from C&C
4. Decryption key recoverable in analysis
5. Encryption model flawless

Paying ransomware: scaremongering
Recovering data: easier from locker-ransomware than crypto * get sent decryption key * reboot b4 ransomware termianted * law enforcement access servers * AV makes repo of recovered key * attacker left priv key in clear * attacker designed own custom encryption alg that can be reversed

Protection
remove admin rights * ant-ransomware tools (Ransom free, Emisoft, Ransomwhere)

Machine Learning and Malware

Collection of data: private company data (emails ; net traffic) * public datasets (virusTotal & androZoo 4 both goodware and malware)

Ideal datasets: rep of real world * statistically significant size (>10000) * realistic ratio goodware 2 malware (100-to-1) * reliable labels

Features
Static features: from metadata (manifest ; ELF/PE) * from code (control flow graphs) * over-approximation of app behaviour * costly 4 finding structural rep of code and data flows * fast for extracting API call w/o relationship
Dynamic features: (sys call seq ; URLs called) * under-approximation of app behaviour * costly bc need to stimulate the applicatio

Good feature props: highlight commonalities and differences * represented by numbers and matrices (count occurrences of feature, or presence as boolean)

Model selection
Dataset split into: training, validation, & testing

No Free Lunch Theorem – best alg depends on task

Questions 2 ask: Simplest model that can solve problem best? * do we have enough data 4 this model?

Fitting:
fitting error as Mean Square Error (sum error of square distance between line and training data) half it.

Linear seperation * maximum margin * underfitting (meaningless) * overfitting (effected by noise – specif 2 our training dataset)

Evaluation

Precision = $\frac{TP}{TP+FP}$

Recall = $\frac{TP}{TP+FN}$

F1 score (harmonic mean of precision and recall)
$= 2*\frac{precision.recall}{precision+recall}$

Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$ - misleading when datasets imbalanced (dumb classifier will have 99% accuracy with 99% goodware)

ROC curve – graphical plot (plot TPR y axis (true positive/total positive) vs FPR (false positive/total negative))
AUC curve – area under ROC curve * higher better * random classifier has AUC = 0.5

k-fold cross evaltuation etc

Robustness
Robustness against time: malware evolves and mutates

Robustness against adversaries: evasion of detection * poisoning of training database ('badword' obfuscation/ good word insertion)
2 combat: reactively (detect attacks; frequent retraining; decision verification) * proactive defense (security-by design defenses: secure/robust learning,

attack detection ; security-by-obscurity: information hiding, randomisation, detection of probing attacks)

Concept drift: after model trained, new malware family may arise
2 combat: periodic re-training ; evaluate classifier with respect to time (but distribution of change not even so may calc misleading 'time decay')

Challenges: High cost of false negatives * hard to find public datasets * high cost of labelling * explainability is hard * imbalanced datasets (most events are benign) so perfom metrics may be misleading & positively biased towards larger class

SVM

Find optimal hyperplane that linearly seperates data (input: training sample w same number of features with label ; output: set of weights for each feature thats linear comb predicts value of y)

Soft-margin (intro slack variables 2 allow 4 some errors) * non-linear seperation requires mapping feature spaces (kernel trick) * multi-class SVM (one-vs-rest etc)

Supervised

Random forest 4 decision trees (no need to prune)

Case study DREBIN

static analysis of android app (takes static binary features) → embedding in vector space → learning based detection (uses linear SVM to seperate) → explanation (using feature weights w) of why labeled malware

Unsupervised

Clustering: high intra-cluster similarity, low inter-cluster similarity
Non-parametric clustering involves steps:
1) Define measure of (dis)similarity between observations
2) Define criterion function for clustering
3) define algorithm to minimize (or max) the criterion function

Measure of similarity:
function d(x,y) with property:

- d(x,y) > = 0
- d(x,y) = 0 <=> x = y
- d(x,y) = d(y,x)
- d(x,y) <= d(x,z) + d(z,y)

Commonly used metrics:
Minkowski distance LK norm =

$$\|x - y\|_k = (\sum_{i=1}^{n} |x_i - y_i|^k)^{1/k}$$

Euclidean distance L2 norm =

$$\|x - y\|_E = (\sum_{i=1}^{n} |x_i - y_i|^2)^{1/2}$$

Criterion function:
Most used by minimum variance clustering method sum-of-squared errors

$$J_{SSE} = \sum_{j=1}^{K} \sum_{x \in \omega_j} |x - \mu_j|^2 \quad where \quad \mu_j = \frac{1}{|\omega_j|} \sum_{x \in \omega_j} x$$

where dataset is represented by cluster centers

$$\mu = \{\mu_1, \cdots, \mu_K\},$$

Cluster validity = Highly subjective (not like with label) * measure & criterion function have major impact

Algorithm to minimise:
iterative approach (sub-optimal but computationally tractable):

1. Find reasonable initial partition
2. Move observation from cluster to another to reduce criterion function

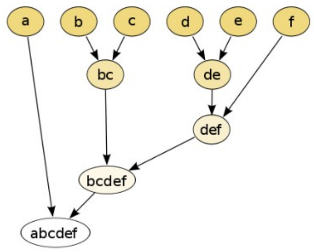
Two groups of iterative methods:

1. Flat clustering
   - Produce disjoint clusters
   - Include K-means
2. Hierachical clustering

Hierachical:
Nested clusters arranged in tree-structure:

1. Agglomerative (bottom-up)
   1. n observations and a measure that keeps all in own clustering
   2. Identify least dissimilar
      1. fuse these 2 (dissimilarity indicated by height at which fusion should be placed
      2. repeat
2. Divisive (top-down)

Dendrograms: to visualise hierachical clustering calculation



can also use sets (not as quantitative)

Cutting the tree:
Different partitions at each level, so extract one set of disjoint clusters by cutting the dendrogram (cutting criterion mb defined using threshold)

Types of linkage:
Complete: maximal inter-cluster dissimilarity * compute pairwise dissimilarities between obs in cluster A and B and record largest
Average: average inter-cluster dissimilarity * calc pairwise dissimilarities between A and B, record average

Finding nearest pair of clusters:
Single-linkage: minimal inter-cluster dissimilarity * compute all pairwise dissimilarity between observations in cluster A and B, record smallest * can result in extended trailing clusters which funse one-at-a-time * not balanced
Centroid linkage: dissimilarity between mean vector of cluster A and B * record smallest* can cause inversion: similarity increases as clusters merged so they fuse higher than the clusters are currently (difficult to visualise and interpret)

Case study BotMiner
Monitors botnet
C-plane clustering: clusters according to communication patterns (performs basic filtering, white listing, multi-step clustering)
A-plane clustering: clusters according to activity type/ activity features
Cross-plane clustering: cross-check clusters in 2 planes to find intersections * score s(h) computed for each host