**Interviewing tips**

* **Interview questions**
  + Interview questions are intentionally vague. This is to encourage questions.
  + Ask clarifying questions
  + Questions reveal how you approach problems.
  + Write down notes about the question. This is so you don't forget details and only partially answer, or give the wrong answer.
  + Interviews should be more like a conversation with a lot of back and forth, thoroughly explore scenarios together and avoid jumping too fast to a solution.
  + The interviewer can only make an evaluation on your suitability for the job based on the things you *say*.
  + **Interviewers test depth of knowledge**
    - There will be questions about technical details on topics to the point where it'll be hard to answer. This is okay, try your best to work through it and say what you're thinking.
    - Interviewers often aren't looking for specific answers, they just want to see how deeply you know a topic.
    - Approach the question from a low level and even ask your interviewer if you need to add more details before moving on.
  + **Interviewers test breadth of knowledge**
    - There will be questions related to the role you're applying for and some that aren't. This is to explore breadth of knowledge.
    - Try your best to explore the scenarios and ask questions. It's very important to say your thinking aloud, you might be on the right track.
* **Show comprehension**
  + Try to always ask clarifying questions even if you think you already know the answer. You might learn some nuance that even improves your idea.
  + Always repeat the question back to the interviewer to both check your understanding and give yourself thinking time.
  + *"Okay, I'll repeat back the question so I can check my understanding…"*
  + *"Just to clarify…"*
  + *"I just want to check I heard correctly…"*
* **State your assumptions**
  + Your interviewer will provide feedback if your assumptions are unreasonable.
  + *"I am going to assume that the organisation is collecting x,y,z logs from hosts and storing these for at least 90 days…"*
  + *"Can I make the assumption that…?"*
  + *"Let's say that we can get x,y,z information…"*
* **When asked a question you're not sure of the answer to right away, try these phrases:**
  + *"I don't know but if I had to invent it, it would be like this…"*
  + *"I don't know that exactly but I know something about a similar subject / sub component…"*
  + *"This is what's popping into my mind right now is…"*
  + *"The only thing that is coming to mind is…"*
  + *"I know a lot about [similar thing], I could talk about that instead? Would that be okay?"*
* **Say what you are thinking**
  + The interviewer can only make an evaluation on your suitability for the job based on the things you *say*.
  + If you don't say your thought process aloud, then the interviewer doesn't know what you know.
  + You may well be on the right track with an answer. You'll be kicking yourself afterwards if you later realise you were but didn't say anything (I missed out on an internship because of this!)
  + Write pseudo code for your coding solution so you don't have to hold everything in your head.
  + *"Right now I am thinking about…"*
  + *"I am thinking about different approaches, for example…"*
  + *"I keep coming back to [subject/idea/thing] but I think that's not the right direction. I am thinking about…"*
  + *"I'm interested in this idea that…"*
* **Reduce cognitive load**
  + Take notes on the question and assumptions during the interview.
  + If the infrastructure is complicated, draw up what you think it looks like.
  + Write pseudocode.
  + Write tests and expected output for code you write, test your code against it.
* **Prepare**
  + Make a checklist that reminds you of what to do for each question, something like:
    - Listen to interview question
    - Take notes on the question
    - Repeat the question
    - Ask clarifying questions
    - State any assumptions
  + Prepare questions that you want to ask your interviewers at the end of the interview so you don't need to think of them on the spot on the day. Since an interview is also for you to know more about the workplace, I asked questions about the worst parts of the job.
  + Bring some small snacks in a box or container that isn't noisy and distracting. A little bit of sugar throughout the interviews can help your problem solving abilities.
  + Stay hydrated - and take a toilet break between every interview if you need to (it's good to take a quiet moment).
* **Do practice interviews**
  + Do them until they feel more comfortable and you can easily talk through problems.
  + Ask your friends/peers to give you really hard questions that you definitely don't know how to answer.
  + Practice being in the very uncomfortable position where you have no idea about the topic you've been asked. Work through it from first principles.
  + Practice speaking aloud everything you know about a topic, even details you think might be irrelevant.
  + Doooo theeeeemmm yes they can be annoying to organise but it is *worth it*.

**Interviewers are potential friends and they want to help you get the job, they are on your side. Let them help you, ask them questions, recite everything you know on a topic and *say your thought process out loud*.**

**Networking**

* OSI Model
  + Application; layer 7 (and basically layers 5 & 6) (includes API, HTTP, etc).
  + Transport; layer 4 (TCP/UDP).
  + Network; layer 3 (Routing).
  + Datalink; layer 2 (Error checking and frame synchronisation).
  + Physical; layer 1 (Bits over fibre).
* Firewalls
  + Rules to prevent incoming and outgoing connections.
* NAT
  + Useful to understand IPv4 vs IPv6.
* DNS
  + (53)
  + Requests to DNS are usually UDP, unless the server gives a redirect notice asking for a TCP connection. Look up in cache happens first. DNS exfiltration. Using raw IP addresses means no DNS logs, but there are HTTP logs. DNS sinkholes.
  + In a reverse DNS lookup, PTR might contain- 2.152.80.208.in-addr.arpa, which will map to 208.80.152.2. DNS lookups start at the end of the string and work backwards, which is why the IP address is backwards in PTR.
* DNS exfiltration
  + Sending data as subdomains.
  + 26856485f6476a567567c6576e678.badguy.com
  + Doesn’t show up in http logs.
* DNS configs
  + Start of Authority (SOA).
  + IP addresses (A and AAAA).
  + SMTP mail exchangers (MX).
  + Name servers (NS).
  + Pointers for reverse DNS lookups (PTR).
  + Domain name aliases (CNAME).
* ARP
  + Pair MAC address with IP Address for IP connections.
* DHCP
  + UDP (67 - Server, 68 - Client)
  + Dynamic address allocation (allocated by router).
  + DHCPDISCOVER -> DHCPOFFER -> DHCPREQUEST -> DHCPACK
* Multiplex
  + Timeshare, statistical share, just useful to know it exists.
* Traceroute
  + Usually uses UDP, but might also use ICMP Echo Request or TCP SYN. TTL, or hop-limit.
  + Initial hop-limit is 128 for windows and 64 for \*nix. Destination returns ICMP Echo Reply.
* Nmap
  + Network scanning tool.
* Intercepts (MiTM)
  + Understand PKI (public key infrastructure in relation to this).
* VPN
  + Hide traffic from ISP but expose traffic to VPN provider.
* Tor
  + Traffic is obvious on a network.
  + How do organised crime investigators find people on tor networks.
* Proxy
  + Why 7 proxies won’t help you.
* BGP
  + Border Gateway Protocol.
  + Holds the internet together.
* Network traffic tools
  + Wireshark
  + Tcpdump
  + Burp suite
* HTTP/S
  + (80, 443)
* SSL/TLS
  + (443)
  + Super important to learn this, includes learning about handshakes, encryption, signing, certificate authorities, trust systems. A good [primer](https://english.ncsc.nl/publications/publications/2019/juni/01/it-security-guidelines-for-transport-layer-security-tls) on all these concepts and algorithms is made available by the Dutch cybersecurity center.
  + Various attacks against older versions of SSL/TLS (with catchy names) on [Wikipedia](https://en.wikipedia.org/wiki/Transport_Layer_Security#Attacks_against_TLS/SSL).
* TCP/UDP
  + Web traffic, chat, voip, traceroute.
  + TCP will throttle back if packets are lost but UDP doesn't.
  + Streaming can slow network TCP connections sharing the same network.
* ICMP
  + Ping and traceroute.
* Mail
  + SMTP (25, 587, 465)
  + IMAP (143, 993)
  + POP3 (110, 995)
* SSH
  + (22)
  + Handshake uses asymmetric encryption to exchange symmetric key.
* Telnet
  + (23, 992)
  + Allows remote communication with hosts.
* ARP
  + Who is 0.0.0.0? Tell 0.0.0.1.
  + Linking IP address to MAC, Looks at cache first.
* DHCP
  + (67, 68) (546, 547)
  + Dynamic (leases IP address, not persistent).
  + Automatic (leases IP address and remembers MAC and IP pairing in a table).
  + Manual (static IP set by administrator).
* IRC
  + Understand use by hackers (botnets).
* FTP/SFTP
  + (21, 22)
* RPC
  + Predefined set of tasks that remote clients can execute.
  + Used inside orgs.
* Service ports
  + 0 - 1023: Reserved for common services - sudo required.
  + 1024 - 49151: Registered ports used for IANA-registered services.
  + 49152 - 65535: Dynamic ports that can be used for anything.
* HTTP Header
  + | Verb | Path | HTTP version |
  + Domain
  + Accept
  + Accept-language
  + Accept-charset
  + Accept-encoding(compression type)
  + Connection- close or keep-alive
  + Referrer
  + Return address
  + Expected Size?
* HTTP Response Header
  + HTTP version
  + Status Codes:
    - 1xx: Informational Response
    - 2xx: Successful
    - 3xx: Redirection
    - 4xx: Client Error
    - 5xx: Server Error
  + Type of data in response
  + Type of encoding
  + Language
  + Charset
* UDP Header
  + Source port
  + Destination port
  + Length
  + Checksum
* Broadcast domains and collision domains.
* Root stores
* CAM table overflow

**Web application**

* Same origin policy
  + Only accept requests from the same origin domain.
* CORS
  + Cross-Origin Resource Sharing. Can specify allowed origins in HTTP headers. Sends a preflight request with options set asking if the server approves, and if the server approves, then the actual request is sent (eg. should client send auth cookies).
* HSTS
  + Policies, eg what websites use HTTPS.
* Cert transparency
  + Can verify certificates against public logs
* HTTP Public Key Pinning
  + (HPKP)
  + Deprecated by Google Chrome
* Cookies
  + httponly - cannot be accessed by javascript.
* CSRF
  + Cross-Site Request Forgery.
  + Cookies.
* XSS
  + Reflected XSS.
  + Persistent XSS.
  + DOM based /client-side XSS.
  + <img scr=””> will often load content from other websites, making a cross-origin HTTP request.
* SQLi
  + (Wo)man in the browser (flash / java applets) (malware).
  + Validation / sanitisation of webforms.
* POST
  + Form data.
* GET
  + Queries.
  + Visible from URL.
* Directory traversal
  + Find directories on the server you’re not meant to be able to see.
  + There are tools that do this.
* APIs
  + Think about what information they return.
  + And what can be sent.
* Beefhook
  + Get info about Chrome extensions.
* User agents
  + Is this a legitimate browser? Or a botnet?
* Browser extension take-overs
  + Miners, cred stealers, adware.
* Local file inclusion
* Remote file inclusion (not as common these days)
* SSRF
  + Server Side Request Forgery.
* Web vuln scanners.
* SQLmap.
* Malicious redirects.

**Infrastructure (Prod / Cloud) Virtualisation**

* Hypervisors.
* Hyperjacking.
* Containers.
* Escaping and privilege escalation techniques.
* Site isolation.
* Network connections from VMs / containers.
* Side-channel attacks.
* Beyondcorp
  + Trusting the host but not the network.

**OS implementation and systems**

* Privilege escalation techniques, and prevention.
* Buffer Overflows.
* Directory traversal (prevention).
* Remote Code Execution / getting shells.
* Local databases
  + Some messaging apps use sqlite for storing messages.
  + Useful for digital forensics, especially on phones.
* Windows
  + Windows registry and group policy.
  + Windows SMB.
  + Samba (with SMB).
  + Buffer Overflows.
  + ROP.
* \*nix
  + SELinux.
  + Kernel, userspace, permissions.
  + MAC vs DAC.
  + /proc
  + /tmp - code can be saved here and executed.
  + /shadow
  + LDAP - Lightweight Directory Browsing Protocol. Lets users have one password for many services. This is similar to Active Directory in windows.
* MacOS
  + Gotofail error (SSL).
  + MacSweeper.
  + Research Mac vulnerabilities.

**Mitigations**

* Patching
* Data Execution Prevention
* Address space layout randomisation
  + To make it harder for buffer overruns to execute privileged instructions at known addresses in memory.
* Principle of least privilege
  + Eg running Internet Explorer with the Administrator SID disabled in the process token. Reduces the ability of buffer overrun exploits to run as elevated user.
* Code signing
  + Requiring kernel mode code to be digitally signed.
* Compiler security features
  + Use of compilers that trap buffer overruns.
* Encryption
  + Of software and/or firmware components.
* Mandatory Access Controls
  + (MACs)
  + Operating systems with Mandatory Access Controls - eg. SELinux.
* "Insecure by exception"
  + When to allow people to do certain things for their job, and how to improve everything else. Don't try to "fix" security, just improve it by 99%.
* Do not blame the user
  + Security is about protecting people, we should build technology that people can trust, not constantly blame users.

**Cryptography, authentication, identity**

* Encryption vs Encoding vs Hashing vs Obfuscation vs Signing
  + Be able to explain the differences between these things.
  + [Various attack models](https://en.wikipedia.org/wiki/Attack_model) (e.g. chosen-plaintext attack).
* Encryption standards + implementations
  + [RSA](https://en.wikipedia.org/wiki/RSA_(cryptosystem)) (asymmetrical).
  + [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) (symmetrical).
  + [ECC](https://en.wikipedia.org/wiki/EdDSA) (namely ed25519) (asymmetric).
  + [Chacha/Salsa](https://en.wikipedia.org/wiki/Salsa20#ChaCha_variant) (symmetric).
* Asymmetric vs symmetric
  + Asymmetric is slow, but good for establishing a trusted connection.
  + Symmetric has a shared key and is faster. Protocols often use asymmetric to transfer symmetric key.
  + Perfect forward secrecy - eg Signal uses this.
* Cyphers
  + Block vs stream [ciphers](https://en.wikipedia.org/wiki/Cipher).
  + [Block cipher modes of operation](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation).
  + [AES-GCM](https://en.wikipedia.org/wiki/Galois/Counter_Mode).
* Trusted Platform Module
  + (TPM)
  + Trusted storage for certs and auth data locally on device/host.
* Integrity and authenticity primitives
  + [Hashing functions](https://en.wikipedia.org/wiki/Cryptographic_hash_function) e.g. MD5, Sha-1, BLAKE. Used for identifiers, very useful for fingerprinting malware samples.
  + [Message Authentication Codes (MACs)](https://en.wikipedia.org/wiki/Message_authentication_code).
  + [Keyed-hash MAC (HMAC)](https://en.wikipedia.org/wiki/HMAC).
* Entropy
  + PRNG (pseudo random number generators).
  + Entropy buffer draining.
  + Methods of filling entropy buffer.
* Certificates
  + What info do certs contain, how are they signed?
  + Look at DigiNotar.
* O-auth
  + Bearer tokens, this can be stolen and used, just like cookies.
* Auth Cookies
  + Client side.
* Sessions
  + Server side.
* Auth systems
  + SAMLv2o.
  + OpenID.
* Biometrics
  + Can't rotate unlike passwords.
* Password management
  + Rotating passwords (and why this is bad).
  + Different password lockers.
* U2F / FIDO
  + Eg. Yubikeys.
  + Helps prevent successful phishing of credentials.
* Compare and contrast multi-factor auth methods

**Malware & Reversing**

* Interesting malware
  + Conficker.
  + Morris worm.
  + Zeus malware.
  + Stuxnet.
  + Wannacry.
  + CookieMiner.
  + Sunburst.
* Malware features
  + Various methods of getting remote code execution.
  + Domain-flux.
  + Fast-Flux.
  + Covert C2 channels.
  + Evasion techniques (e.g. anti-sandbox).
  + Process hollowing.
  + Mutexes.
  + Multi-vector and polymorphic attacks.
  + RAT (remote access trojan) features.
* Decompiling/ reversing
  + Obfuscation of code, unique strings (you can use for identifying code).
  + IdaPro, Ghidra.
* Static / dynamic analysis
  + Describe the differences.
  + Virus total.
  + Reverse.it.
  + Hybrid Analysis.

**Exploits**

* Three ways to attack - Social, Physical, Network
  + **Social**
    - Ask the person for access, phishing.
    - Cognitive biases - look at how these are exploited.
    - Spear phishing.
    - Water holing.
    - Baiting (dropping CDs or USB drivers and hoping people use them).
    - Tailgating.
  + **Physical**
    - Get hard drive access, will it be encrypted?
    - Boot from linux.
    - Brute force password.
    - Keyloggers.
    - Frequency jamming (bluetooth/wifi).
    - Covert listening devices.
    - Hidden cameras.
    - Disk encryption.
    - Trusted Platform Module.
    - Spying via unintentional radio or electrical signals, sounds, and vibrations (TEMPEST - NSA).
  + **Network**
    - Nmap.
    - Find CVEs for any services running.
    - Interception attacks.
    - Getting unsecured info over the network.
* Exploit Kits and drive-by download attacks
* Remote Control
  + Remote code execution (RCE) and privilege.
  + Bind shell (opens port and waits for attacker).
  + Reverse shell (connects to port on attackers C2 server).
* Spoofing
  + Email spoofing.
  + IP address spoofing.
  + MAC spoofing.
  + Biometric spoofing.
  + ARP spoofing.
* Tools
  + Metasploit.
  + ExploitDB.
  + Shodan - Google but for devices/servers connected to the internet.
  + Google the version number of anything to look for exploits.
  + Hak5 tools.

**Attack structure**

Practice describing security concepts in the context of an attack. These categories are a rough guide on attack structure for a targeted attack. Non-targeted attacks tend to be a bit more "all-in-one".

* Reconnaissance
  + OSINT, Google dorking, Shodan.
* Resource development
  + Get infrafrusture (via compromise or otherwise).
  + Build malware.
  + Compromise accounts.
* Initial access
  + Phishing.
  + Hardware placements.
  + Supply chain compromise.
  + Exploit public-facing apps.
* Execution
  + Shells & interpreters (powershell, python, javascript, etc.).
  + Scheduled tasks, Windows Management Instrumentation (WMI).
* Persistence
  + Additional accounts/creds.
  + Start-up/log-on/boot scripts, modify launch agents, DLL side-loading, Webshells.
  + Scheduled tasks.
* Privilege escalation
  + Sudo, token/key theft, IAM/group policy modification.
  + Many persistence exploits are PrivEsc methods too.
* Defense evasion
  + Disable detection software & logging.
  + Revert VM/Cloud instances.
  + Process hollowing/injection, bootkits.
* Credential access
  + Brute force, access password managers, keylogging.
  + etc/passwd & etc/shadow.
  + Windows DCSync, Kerberos Gold & Silver tickets.
  + Clear-text creds in files/pastebin, etc.
* Discovery
  + Network scanning.
  + Find accounts by listing policies.
  + Find remote systems, software and system info, VM/sandbox.
* Lateral movement
  + SSH/RDP/SMB.
  + Compromise shared content, internal spear phishing.
  + Pass the hash/ticket, tokens, cookies.
* Collection
  + Database dumps.
  + Audio/video/screen capture, keylogging.
  + Internal documentation, network shared drives, internal traffic interception.
* Exfiltration
  + Removable media/USB, Bluetooth exfil.
  + C2 channels, DNS exfil, web services like code repos & Cloud backup storage.
  + Scheduled transfers.
* Command and control
  + Web service (dead drop resolvers, one-way/bi-directional traffic), encrypted channels.
  + Removable media.
  + Steganography, encoded commands.
* Impact
  + Deleted accounts or data, encrypt data (like ransomware).
  + Defacement.
  + Denial of service, shutdown/reboot systems.

See [Mitre ATT&CK](https://attack.mitre.org/) for more info.

**Detection**

* IDS
  + Intrusion Detection System (signature based (eg. snort) or behaviour based).
  + Snort/Suricata/YARA rule writing
  + Host-based Intrusion Detection System (eg. OSSEC)
* SIEM
  + Security Information and Event Management.
* IOC
  + Indicator of compromise (often shared amongst orgs/groups).
  + Specific details (e.g. IP addresses, hashes, domains)
* Things that create signals
  + Honeypots, snort.
* Things that triage signals
  + SIEM, eg splunk.
* Things that will alert a human
  + Automatic triage of collated logs, machine learning.
  + Notifications and analyst fatigue.
  + Systems that make it easy to decide if alert is actual hacks or not.
* Signatures
  + Host-based signatures
    - Eg changes to the registry, files created or modified.
    - Strings in found in malware samples appearing in binaries installed on hosts (/Antivirus).
  + Network signatures
    - Eg checking DNS records for attempts to contact C2 (command and control) servers.
* Anomaly / Behaviour based detection
  + IDS learns model of “normal” behaviour, then can detect things that deviate too far from normal - eg unusual urls being accessed, user specific- login times / usual work hours, normal files accessed.
  + Can also look for things that a hacker might specifically do (eg, HISTFILE commands, accessing /proc).
  + If someone is inside the network- If action could be suspicious, increase log verbosity for that user.
* Firewall rules
  + Brute force (trying to log in with a lot of failures).
  + Detecting port scanning (could look for TCP SYN packets with no following SYN ACK/ half connections).
  + Antivirus software notifications.
  + Large amounts of upload traffic.
* Honey pots
  + Canary tokens.
  + Dummy internal service / web server, can check traffic, see what attacker tries.
* Things to know about attackers
  + Slow attacks are harder to detect.
  + Attacker can spoof packets that look like other types of attacks, deliberately create a lot of noise.
  + Attacker can spoof IP address sending packets, but can check TTL of packets and TTL of reverse lookup to find spoofed addresses.
  + Correlating IPs with physical location (is difficult and inaccurate often).
* Logs to look at
  + DNS queries to suspicious domains.
  + HTTP headers could contain wonky information.
  + Metadata of files (eg. author of file) (more forensics?).
  + Traffic volume.
  + Traffic patterns.
  + Execution logs.
* Detection related tools
  + Splunk.
  + Arcsight.
  + Qradar.
  + Darktrace.
  + Tcpdump.
  + Wireshark.
  + Zeek.
* A curated list of [awesome threat detection](https://github.com/0x4D31/awesome-threat-detection) resources

**Digital Forensics**

* Evidence volatility (network vs memory vs disk)
* Network forensics
  + DNS logs / passive DNS
  + Netflow
  + Sampling rate
* Disk forensics
  + Disk imaging
  + Filesystems (NTFS / ext2/3/4 / AFPS)
  + Logs (Windows event logs, Unix system logs, application logs)
  + Data recovery (carving)
  + Tools
  + plaso / log2timeline
  + FTK imager
  + encase
* Memory forensics
  + Memory acquisition (footprint, smear, hiberfiles)
  + Virtual vs physical memory
  + Life of an executable
  + Memory structures
  + Kernel space vs user space
  + Tools
  + Volatility
  + Google Rapid Response (GRR) / Rekall
  + WinDbg
* Mobile forensics
  + Jailbreaking devices, implications
  + Differences between mobile and computer forensics
  + Android vs. iPhone
* Anti forensics
  + How does malware try to hide?
  + Timestomping
* Chain of custody
  + Handover notes

**Incident Management**

* Privacy incidents vs information security incidents
* Know when to talk to legal, users, managers, directors.
* Run a scenario from A to Z, how would you ...
* Good practices for running incidents
  + How to delegate.
  + Who does what role.
  + How is communication managed + methods of communication.
  + When to stop an attack.
  + Understand risk of alerting attacker.
  + Ways an attacker may clean up / hide their attack.
  + When / how to inform upper management (manage expectations).
  + Metrics to assign Priorities (e.g. what needs to happen until you increase the prio for a case)
  + Use playbooks if available
* Important things to know and understand
  + Type of alerts, how these are triggered.
  + Finding the root cause.
  + Understand stages of an attack (e.g. cyber-killchain)
  + Symptom vs Cause.
  + First principles vs in depth systems knowledge (why both are good).
  + Building timeline of events.
  + Understand why you should assume good intent, and how to work with people rather than against them.
  + Prevent future incidents with the same root cause
  + Response models
    - SANS' PICERL (Preparation, Identification, Containement, Eradication, Recovery, Lessons learned)
    - Google's IMAG (Incident Management At Google)

**Coding & algorithms**

* The basics
  + Conditions (if, else).
  + Loops (for loops, while loops).
  + Dictionaries.
  + Slices/lists/arrays.
  + String/array operations (split, contaings, length, regular expressions).
  + Pseudo code (concisely describing your approach to a problem).
* Data structures
  + Dictionaries / hash tables (array of linked lists, or sometimes a BST).
  + Arrays.
  + Stacks.
  + SQL/tables.
  + Bigtables.
* Sorting
  + Quicksort, merge sort.
* Searching
  + Binary vs linear.
* Big O
  + For space and time.
* Regular expressions
  + O(n), but O(n!) when matching.
  + It's useful to be familiar with basic regex syntax, too.
* Recursion
  + And why it is rarely used.
* Python
  + List comprehensions and generators [ x for x in range() ].
  + Iterators and generators.
  + Slicing [start:stop:step].
  + Regular expressions.
  + Types (dynamic types), data structures.
  + Pros and cons of Python vs C, Java, etc.
  + Understand common functions very well, be comfortable in the language.

**Security themed coding challenges**

These security engineering challenges focus on text parsing and manipulation, basic data structures, and simple logic flows. Give the challenges a go, no need to finish them to completion because all practice helps.

* Cyphers / encryption algorithms
  + Implement a cypher which converts text to emoji or something.
  + Be able to implement basic cyphers.
* Parse arbitrary logs
  + Collect logs (of any kind) and write a parser which pulls out specific details (domains, executable names, timestamps etc.)
* Web scrapers
  + Write a script to scrape information from a website.
* Port scanners
  + Write a port scanner or detect port scanning.
* Botnets
  + How would you build ssh botnet?
* Password bruteforcer
  + Generate credentials and store successful logins.
* Scrape metadata from PDFs
  + Write a mini forensics tool to collect identifying information from PDF metadata.
* Recover deleted items
  + Most software will keep deleted items for ~30 days for recovery. Find out where these are stored.
  + Write a script to pull these items from local databases.
* Malware signatures
  + A program that looks for malware signatures in binaries and code samples.
  + Look at Yara rules for examples.