**Cryptography Failures**

*Josip Medved*

24 participants

Style: Story-telling

Bernardo Provenzano (mafia boss)

Rajib Karim

Old “software” dies hard 😊

Obfusication is sometime ok

Cipher: Vigenere

* First historical appearance of proper key
* Letters and sum them together
* Described by Giovan Battista Bellaso in 1553
* Broke by Charles Babbage in 1846
* If you missed a letter, it will give you rubbish
* Was not in common use

Cipher: Enigma

* Rotor stream cipher
* Period of 16,900 (25\*25\*26) – letters start repeating after 16k
* Existed before war in 1920s
* Developed by Arthur Scherbius in 1920s
* Math behind enigma was cracked by Marian Rejewski in 1932
* Lesson: Output must be random. Letter never encrypted to itself
* Lesson: Correlations are killer
* It is a 1 to 1 translation
* Lesson: Attack on partial rounds can always be expanded

Lorenz

* Rotor stream cipher
* First encryption method using (relative) primes
* Nicknamed Tunny by British
* Based on work by Gilbert Vernam at AT&T Bell Labs in 1917. Invention of XOR table.
* Used by Hitler
* Used for army communications from mid-1942
* Failure: 4000-character message sent in August 1941 from Athens to Vienna
* Receiving operator asked for repeat
* Sending operator repeated message with the same key but slight abbreviations
* Both plain texts were extracted together with their key
* Side note: German words (a lot of words) could only mean 1 thing
* Lesson: in symmetrical key encryption, never ever encrypt different plaintext with the same key
* Lesson: Don’t leak encryption state information (sending the settings of the keys)
* Men went to war, Women did the decryption
* Lesson: 41-character repetition doomed Lorenz
* Failure: Germans were chatty, they didn’t have a decryption team or even transcribers

MD-5

* History
* 1993: pseudo-collision
* Attacks
* 2012: Flame malware forged Windows update certificate
* Lessons: allow algorithms to change over time, quite a few applications had 16 bytes for hash hardcoded
* Replace algorithms early
* Symantec phased MD-5 in 2009
* Microsoft phased MD-5 in 2015
* Still often used in anti-virus industry

SHA-1

* Designed as part of US government Capstone project
* Authored by NSA
* Original implementation (now called SHA-0) was slightly corrected
* Published in 1995
* 160-bit in size
* 2005: reduced version attack (53 rounds)
* 2015: first full collision (aka SHAppening)
* 2017: first public collision (aka SHAttered)
* 2017: SSL certificates not accepted by major browsers, after first public collision
* Lessons: allow algorithms to change over time, quite a few applications had 20 bytes for hash hardcoded
* Git uses SHA-1. Hardcoded.

How attacks look these days?

Keyloggers

* Keyllama 4MB USB Value Keylogger – can be bought in Amazon
* National keyboard layout support
* Software keyloggers are easier
* In 1999” used encryption, used dial-up, FBI broke in his offices twice
* Larry Ropp, 2004, KEYKatacher, did not violate federal wire-tapp law
* Josh Glazebrook, 2007, handwritten notes to anonymous email, FBI installed CIPAV on MySpace profile. Got his IP in MySpace
* Advice: watch what you click, cover camera

Network Interception

* Used for pentesting
* You get it for free on public wireless
* In terms: “we may monitor or log your traffic”
* GOGO INFLIGHT, 2016, man-in-the-middle attack, issues fake SSL certificates
* STARBUCKS, 2017, man-in-the-middle attack, someone setup rogue hotspots, did Monero mining
* VPNFILTER, 2018, Russian botnet, industrial hardware control, seized by FBI
* Advice: Always use VPN

Backdoors

* Accidental – Debian’s OpenSSL issue in 2008
* Intentional – elliptic curve RNG
* Hardware-based
* DEBIAN OPENSSL, broken in 2006, fixed in 2008, bug limited number of keys to 32,768, also presented in Ubuntu
* DUAL\_EC\_DRBG, weakened encryption
* SIMON AND SPECK, lightweight block ciphers, optimized for IoT
* Advice: Trust no one, physical security is important

FUTURE

* IoT encryption algorithms, primary goal to have it run on weak hardware (goal: decent encryption)
* Quantum computers, making RSA obsolete
* Politics

**Winning Strategies for the Tech Job Interview**

*Vivekanand Kirubanandan – Career Coach at SDE Skill*

Question: First Tech Job to Next Tech Job

Technical interview Job Process

We will take it apart

Speaker

* Founder of SDE Skills
* Software Development Manager at Amazon
* Active interviewer with over 500 technical and behavioral interviews

Interview Process

* Resume Review
* Recruiter Screen
* Phone Screen
* Onsite
* Profile Matching
* Negotiation

How is internship interview different?

* Bypasses a lot of process, can only negotiate start date, etc.

Resume Review

* Facts
* 6-60 seconds – if you are lucky
* Screened by Recruiters / Hiring Managers
* Z Scan
* Optimized
* One page resume / Max 2 sides
* Be brutal, leave out unnecessary items

Resume are entry ticket

Take a lot of time and effort screening a lot of applications

Z Scan, scan the resume whatever is catchy. Top-left, top-right, bottom-left, bottom-right

Heatmaps for eye-tracking in looking at resume

The decision is simple, Red pile, green pile. Are you worth my time in talking 15 minutes?

Application Tracking System – parse the resume to put red pile and green pile. But this software is junk.

Latext Resume – Beautiful CSS, position things, organize content. For human eye is great, for software eyes not good.

Make sure it is your resume is in .doc, .rtf., .txt. easy to parse. Tables are terrible.

Personal preference of speaker: PDF

If junior dev, go word document format

Optimize: thinks yourself as the interviewer

* Shrink your page to one-side, if you can

Recruiter Screen

* Facts
* Often via email / phone
* Recruiters may be differently motivated
* Optimize
* Be nice and polite
* Be persistent in follow-ups – people won’t feel bad on follow ups

Pipeline process

* Each has an objective
* Advance people who has higher change of success rate for the next process

Asked Google Recruiter

* Why Amazon managers fail on your interview?
* This gives more insights

Amazon emphasis on leadership

Facebook is a lot more bottom-up, leadership supports the bottom

Pick a company is large. What is the ratio of offered job to accepted?

Top-rank companies 80% - acceptance rate

Small dev shop 30% - acceptance rate

Estimation: In order to get 1 successful hire, you need around 150-200 applications

Recruiter looks for roots too.

Tips: By-pass Resume Review and Recruiter Screen with referral

Message in Linkedin but..

* Check for jobs, ask for advice or referral for that job

Phone Screen

* Facts
* Trying to identify if you are worth the time
* Often about 1 hour
* Optimize
* Prepare, do your Homework
* Listen first, then Talk
* Take notes – take notes on what did you say. They will cross-reference you. Companies have interview notes stored.

People are over-indexed on technology

* I’m iOS I only want IOS dev job

Other people are flexible

* Tool / programming language agnostic

Figure out the CEO, manager, background

In big tech, people follow good managers when they leave.

Go Glassdoor for start-ups

Phone interview is the first technical screen

* Coding interviews
* Analyze data
* Looks for your background

Onsite

* Facts
* Assessing you to ensure you are right for the job
* 3-6 interviews, different formats
* Optimize
* Prepare, learn about the team, interviewers
* Be structured, reduce interviewer’s work
* Know the assessment criteria

Cascading interview

* People know the previous interview (from recruiter to hiring manager)

Objective interview

* No comparing of notes for interviewers
* No same question

Onsite

* Evidence A – Good, Evidence B – Bad, doesn’t mean you are immediately rejected
* Interviewers have a harder time in this process

STAR format of interview

Profile Matching

* Facts
* They like you, but its not over yet
* Finding Team Fit
* Optimize
* Ask questions to help decide between options
* Do not reject / cast away any teams
* Have (or create) a preference order)

Good questions: why is this position open?

After the answer reply with, this might let me not push through

Negotiation

* Facts
* There is almost always room for negotiation
* It is still not a done deal
* Optimize
* Know the market / demand
* Be polite
* Don’t close doors that you may regret

Good thing to say: I’m happy with the offer, give me 2 weeks to think about it. But if you give me 1.5x I’ll accept it right away

Good thing to say: I’m waiting for my bonus in October, give me x amount of money, then I’ll leave now and join you.

Bigger Picture

* Be structured in your approach
* There is a time and place for everything
* Incremental efforts pay off a lot – a big burst of work is harder
* Always Be Closing / You are your best advocate. Don’t sell yourself short

**Networking with Passion and Purpose**

*Kenlyn Terai @kkterai*

Notes: speaker is not good, generic idea

Web Development instructor at Flatiron School

Keys to authentic communication

Ikigai (a reason for being)

**How to be a Software Architecture**

*Vidya Vrat Agarwal -* T-Mobile vidya\_mcl@yahoo.com

Notes: soft skills, advice-type

Trait for a software architecture

* How to speak

Evolutionary software

No company has an ideal architecture

SAFE

* They have an architectural runaway

Core quality a Software Architect

* Have technical background
* Must have been through complete SDLC of various types of applications
* (Win, Web, Mobile etc..)
* Must understand concepts of technology
* Hands on coding is a must

Communication & Collaboration

Mentoring

Presentation

Drive results

Dive Deep

Big Picture

Ownership & Accountability

Backbone & High standards

Influential

WPF Form in an energy firm

You don’t need to know both Azure and AWS. Only 1.

Orchestrator in Cloud technologies

* Spins your VMs

Architects do not fight or argue like Azure is better than AWS

There is a governing body for programming language

Airport Control Tower, only English for asking permission to land

ECMA – European Computers Manufacturers Association

Learning how to setup CI/CD pipeline

Why do I need it?

Focus on core fundamentals on Software Architecture

So many varieties of “architects” in job description

Don’t be an ivory tower architect, ivory tower – can talk but don’t understand technology

Today, even architects are coding, not coding like in a software dev but can understand code

**Communication and Collaboration**

Architects are called in so many meetings, if you don’t talk, what’s the value in you?

Who wants power? But don’t want to be a manager?

Answer: Software Architect

Waterfall may be due to business decisions

Have full hands-on view of SDLC

* How many environments?
* On-prem? On-cloud? Hybrid?
* Source code repo?

The speaker designed a solution for a Chinese resto

* Didn’t want on-Cloud

Collaborate

* Know them personally

You can’t document everything

T-Mobile

* Uses confluence
* Uses sequence diagram

Document – keep it simple, easy to follow

Certifications

* Make sure you renew it
* Caveats: may not bring working knowledge
* But will bring credibility and influence
* Keep certifications relevant to your job profile
* TOGAF – open group architecture framework
* What do you need certifications?

You don’t need to know everything

Linkedin description of the speaker

* People

Copy job description of people in the Linkedin

* Understand it
* May get you to the job interview

Process brings value

TOGAF

* Architecture repository

Architecture is all about Patterns.

* That’s not right
* Many times, pattern is already there.

Do Architects know everything?

* No
* The speaker came from the server side

What is the Shape of an Architect?

* Letter “T”
* Means they have the breadth across and depth.
* You know high availability works, doesn’t mean I know how to do load-balancer. But knows the value of having it.

Dev’s are “I”, they don’t care everything around

Know JavaScript and TypeScript, it is the key today.

Know the code, know the UI architectural pattern

NFR (Non-functional Requirements)

* UI
* Login
* Security
* Scalability
* High availability – keep spinning VMs, get proxies
* Etc

Ask the customer, what type of workload do you expect?

**How does a day look like?**

Meetings

Roadmap / discussions

Impact analysis

Design discussions

Production issues

Performance improvements

Code quality / Best practices

Dashboards

Process improvements

Cross-team-collaboration

Hands-on-Code

Code Reviews

Mentoring/coaching

Innovating

Leadership discussions

Start Small, Think Big

* Get the context of the problem
* Ask clarifying questions/understand requirements
* Cut the ambiguity
* Think of customer journey
* Think long-term
* Design a reactive system
* Own the design and handoff to TA/BA/Dev etc.
* Validate

Customer (internal/external) obsession came from Amazon

* Who are you building the software for?

Circuit breaker pattern

* Makes sure you have a reactive system
* Avoids ripple effect of API failure

Resiliency - Cache

* Try it again? Or show an error?

Reliability

* Right answer all the time?
* Put a lot of unit testing

“Fake it until you make it” – Steven Tyler

* Be positive
* Build a personality
* Talk like an architect
* Do a lot of whiteboarding

CQRS – Command Query Responsibility Segregation