給菜鳥的一封信

by

Triton Ho 

前言

- 這是工作十五年後，給大學剛畢業的自己的一封信
- 這封信的讀者們，其中一定有人比我更聰明，走得更遠。
  這份簡報是網絡上的一個小小建議，而不是金科玉律。希望這簡報不會妨礙你們的成長。

大綱

- 淺談「安全性」
- 淺談技術債
- 淺談 test case
- Refactoring
- Design Review / Code Review
- 淺談「演算法」重要性

淺談「安全性」

- 現實上的資安事故
- 別自行發明「安全」的架構
- defense in depth

普通人覺得的資安事故.....

- 某 IQ 200 的天才 cracker ，以他私藏的系統漏洞入侵系統.....
- 然後系統管理高手使用價值數百萬 NT 的軟體，跟 cracker 鬥智.....

94.87% 的資安事故

- Database 的 5432 / 3306 port 是公開的
  然後其密碼是 abcd1234
  - 我作夢夢見的，絕對跟政府專案無關！
- 在 2020 年使用 ubuntu 16.04LTS ，而且從沒跑過更新
  - 拉遠一句： ubuntu 20.04LTS 是史上最偉大的版本！
- 工讀生錯手打翻泡麵，然後主管對外宣稱「被黑客攻擊」
- 存密碼時不知 bcrypt / scrypt / argon2 ，而使用 murmur

所謂的「入侵」.....

- （以下請心中默念 100 次）
  完美沒 bug 的系統是沒法被入侵的
  完美沒 bug 的系統是沒法被入侵的
  完美沒 bug 的系統是沒法被入侵的
  .....
- 傳統的 buffer overflow 手段，也是 application bug 一種
  - 更別說著名的 SQL injection 了
- Password = “abcd1234” 則是 configuration 上的 bug

作為一個開發者

- 寫出清爽容易理解的 source code ，減少有 bug 的可能性，就是你的系統安全的最大貢獻

別自行發明「安全」的架構

- 像是 sha256 被廣泛認可前，你知道多少數學專家再三地試圖找出其數學漏洞嗎？
- 你肯定你真的有足夠的料子嗎？
- 你一個人的力量，比得上全球資安專家？
  - 數不清的「高手」，自己用 sha256 + salt 來做 password digestion
  - 然後因為沒加上 iteration ，被人以 GPU 暴力破解

請站在巨人的肩膀上

- aka：有事要死一起死
- 你想一下：
  - 現在你待的沒什麼錢的小公司
  - 跟一堆銀行都是用上偉大的 ubuntu 20.04 LTS（這份簡報寫在 2020 年）
  - 而且，你有做好定期系統更新
- 現在 cracker 在 ubuntu 20.04 找到了一個全新的 system bug 並且能以此入侵，你覺得他會.....
  - 把自己頭像掛到貴公司的首頁，然後在 ptt 高調「我幹的」

    還是
  - 入侵銀行資料庫，在不觸發 auditing system 下把錢偷轉到自己戶口

defense in depth

- 名句：有人的地方就有 bug ，有 bug 的地方就需要工程師
  - 即是說：除非系統是貓寫的，否則沒法保證 100% 沒 bug
- 智障想法：
  - 反正我的資料庫沒公開 port 3306 / 5432 ，那我的資料庫很安全啊，所以我用 plaintext 存用戶帳密也沒差
- 正常人類想法：
  - 雖然我的資料庫是安全的，但是難保 app. server 被入侵， cracker 能看光光資料庫
    所以用戶帳密我們只存 bcrypt digest ，讓 cracker 得物沒所用

淺談技術債

- 所謂技術債.....
- 藝術家 vs 工程師
- 「債務」成因

一般人眼中的技術債

- 前人留下來的爛攤子
- 讓系統越來越難改動
- Source code 越來越臭，越來越難以理解
- 最終在某個「系統小修改」引入災難級的 bug ，害公司倒掉
- 這些都不是錯的，但是.....

  你認為最垃圾的 source code 是什麼？

現實世界

- 現實中，>94.87% 生存少於十年的公司，其倒閉原因都是銀行債務，而不是技術債
- 現實世界：
  - 沒法賺錢的公司是可恥的
  - 上不了 production，沒法替公司省錢 / 賺錢的 source code 更是垃圾
- 你的競爭對手比你早把產品推出市場，那麼你的產品就必須加倍地優秀才讓客戶回頭
- 你的系統必須有帶來價值（賺錢 / 省錢），才能說服公司進一步投資把不好的地方改善

藝術家 vs 工程師

- 藝術家
  - 我要做一件完美的藝術品
  - 最終是否能做出來沒有太大所謂
  - 不能接受不完美的成果
- 工程師
  - 我要在一堆資源（錢 / 時間 / 技術力）的限制下，做出能解決問題的方案
  - 盡可能以已知的工程手段來實行，來盡可能減低專案失敗的風險
  - 只要滿足法規和商業合約，並且客戶肯付錢，那就是好的方案。今次執行時學習到的知識，可以用在改善下一個專案上

身為工程師，請記住.....

- 把專案早日推出市場，才能知道用戶真正想要什麼
  - 市場調查 / 用戶研究不是萬能的。
    Microsoft Excel 推出市場前，有那個用戶想到其改革了辦公室行政？
- 很多專案，事前內部想得很完美，推出後才發現用戶不肯付帳
  - 你越花時間把程式寫得完美，就越把上市日期延後
  - 用戶不肯付帳的軟體，寫得再完美也是毫無價值的

繼續補充

- 我沒有鼓勵大家「寫爛爛的 source code 也沒差」
- 我是跟大家說：你的 code quality ，要視專案性質和其商業價值而決定
  - 建貓窩和建核電站的抗地震要求也不同吧
- 你賺到了錢，公司可以活下去，你才有資源去追求你心中的完美質量
  - FAANG 他們的系統不是一天建成的，而是隨時間演化改進的

「債務」成因

- 技術債從來不是源自衝死線而作出 code quality 上妥協
- 最常見的原因：
  - 衝完死線後，老闆有~~不經大腦~~的新點子讓你去衝，不給你還債
  - 老闆沒正確理解系統現況，輕看工程師們的意見
  - 團體內有「忙啊忙啊」整天在混的工程師，而老闆放任質量不管
- 所有的「技術債」都只是表徵，root cause 永遠是公司制度和管理上的問題

淺談 test case

- 真實專案中的 test case
- 設計 / 架構上的 bug
- 某公司 (作夢中) 災情
- code coverage
- Test case 小心得

真實專案中的 test case

- Test case 就像戰艦的護甲
  - 每多一個 test case 你的系統當然更少有 bug 的可能性
  - 但是，戰艦沒可能無限地增加護甲（太重會直接沉進水底），太多 test case 也會讓你的客戶付不起開發費
- 現實世界：你客戶不會清楚告訴你他們要戰艦還是空母，在需求 / 設計未明朗化前太早建的 test case，很可能最終要全丟！
  - 你們之中，誰沒試過寫到一半改需求 / 改設計的，可以用石頭丟我
- 適量的 test case 重要，但是別迷信 test case 萬能
  - Test case 只能防止執行面（ coding ）上的 bug
  - 防不了設計 / 架構上的 bug
  - 在需要支援 multithread 的 package（例子： caching library ），test case 是高度困難的
- Clean code 也是防止 bug 的重要手段，但需要時間來修練的

設計 / 架構上的 bug

- 例子，我想寫一個 vim 自動換行的功能：
  - 每行最多 80 個 char
  - 如果超過字數時，自動以 word 的單位，把最後一個 word 換到新行
  - word 的定義： [\w]+
- 這需求是有 bug 的
  - 如果有一個 word ，其長度超過 80char 呢？
- 問題是：你只能測到你想像到的 input

某公司（作夢中）災情

- 系統全滅 / 變慢：
  - App bug ：這麼久只試過一次
  - Algorithm design flaw ： ~20%
  - 硬體（ aws ）問題： ~70%
  - 人為出錯： ~10%
- Test case 可以讓你看到 Application Logic 是否寫得如設計一樣
  但是，全系統性問題，很多都不是 test case 能保護的

code coverage

- 某公司的 HR 說：嘛，你們是工程師，在公開活動時替公司多說好話來幫公司招才嘛
  - 工程師們：招才是 HR 的 KPI 不是我們的 KPI……
- 盲目地談 code coverage 的壞處：你會迫工程師為了 code coverage 而寫 test case ，而不是為了保護系統而寫
- 另外：困難的 / 錯了要賠很多錢的 package ，理應比輕鬆的容易寫更多 test case 去保護
- 空談劃一性的 code coverage 指標，就會讓資源錯配到輕鬆的 package ，讓真正需求的 package 不夠保護
  - 能做事的工程師的時間，永遠是公司的稀缺資源.....
- 老闆付你錢是看你寫完了多少功能，而不是看你寫了多少行
  同理， test case 應該看你有多少 edge case 防止了多少個雷，而不是看你 code coverage

Test case 小心得

- 能 compile-time 就浮出來的問題就不要讓他在 runtime 才發現
- 能 unit test 保護的就別用 integration test
- 能用 framework / automation 解決的就別手動來
  - 像是 dependency injection 時，自動檢查所有物件是否能串起來
- 能別用 generic type 就別用
  - 像 java.lang.Object

Refactoring

- 在說出「我想改用ＸＸＸ」前
- 怎看待網上「技術文」
- Refactoring 小建議
- 爬山小技巧
- 如果你的 Refactoring 建議被拒絕
- punctuated equilibrium

在說出「我想改用ＸＸＸ」前

- 例子：現在系統用 pg 沒有 scalibility 耶，我想換成 noSQL ！
- 請問一下自己：
  - 萬一改動時不順利，你肯負責任到底，而不是一走了之嗎？
  - 你是否有足夠功績 / 成就，來說服別人？
  - 你了解之前會用 YYY 而不是 XXX 的原因嗎？
  - 你是否真的了解 XXX ？別告訴我你只看過 XXX 的宣傳文和一兩篇網絡文就叫「研究」

怎看待網上「技術文」

- 先問你一下，如果有人向你介紹一種藥物：
  - 可以治肝癌和 AIDS，也能治感冒
  - 三歲到八十都能用，絕無副作用
- 正常人會回答：「幹！你當我智障好騙嗎？」
- 我會回答：你知道現實不可能有這麼好用的藥物，為何你會覺得有某一語言 / 某一資料庫可以應用在任何場景？
- 做人別太「甜」

Refactoring 小建議

- 有人之處就有 bug ，有人近期改過的 source code 就會引入新的 bug
- 日久失修的 legacy code ，就像長期沒人清理的屎坑
  - 大家都習慣蓋起來不想面對
  - 雖然會臭，但是還是勉強能忍
  - 打開來清理時，其臭味才是最難以忍受的
  - 屎坑的深度和廣度，很多時候都遠超大家想像
  - 未清理完成前，你會全身髒一直臭

爬山小技巧

- 錯誤示範：
  - 一個從沒爬山經驗的人說要去爬玉山
  - 然後裝備隨便看 google 買一買就起行
- 正確示範：
  - 我想爬玉山！
  - 我把爬玉山計劃成 A B C D E .....的 checkpoint
  - 在每一個 checkpoint，我都看看我是否夠體力再爬下去，不然就打個卡等下次再來
  - 這些 checkpoint 都是可以拿來說嘴的，即使不能一次攻頂，也不會讓人覺得一事無成
  - 裝備買了後找個簡單路線先試用，以防買了某國黑心貨
- 現在上老闆對專案的耐心不會超過三個月
  三個月內沒法到達 checkpoint 的專案，一定會被砍掉的

如果你的 Refactoring 建議被拒絕

- 請試一下站在老闆來看：
  - 之前的新人也是這麼說，然後做了一半就留下爛攤子和一堆 bug 給我.....
  - 現在系統不是很完美，但是還是能賺錢，為何不去做新的系統賺更多
- 很多時，你需要忍待等「危機」來到，你才能動手改革的
  - 像是爆雷了引發全面性當機
  - 像是有 bug 要公司賠大錢
  - 像是 server cost 太高 / 程式太爛改不動

Punctuated Equilibrium

- 一般人以為的物種演化：
  - 今年成長了一點點，明年成長了一點點
  - 不停累積下去的
- 真正演化：
  - 如果環境沒改變，物種會長期地停滯
  - 環境改變時，物種就會急遽改變（適應不了就是死嘛）
  - 適應後，又是長期的停滯
- 如果６５００萬年前隕石沒撞上地球，今天恐龍還在統治地球
- 越是大型的改革，你越需要游說多方持份者，越是需要等待時機來臨

Code Review / Design Review

- Code Review 目的
- 淺談 comments
- comment 常見用途
- 淺談 code quality
- 著眼大局
- 做好的 Reviewer

Code Review 目的

- 作為 yellow duck
  - 人總有睡不醒的時候.....
- 讓一個 module 有更多人為其負責
  - 在 code review 時，讓知識傳給更多人
  - 避免團隊人員出現 single point of failure
- 讓 source code 滿足團隊的標準
- 看清 implementation 是否有依照最初談好的 design 來做

淺談 comment

- 有人說：「好的 source code 應該簡潔易明，讓人一看就懂，所以不需要 comment 」
  - 我的回應：一堆網絡小男生，女孩的手也沒摸過就來當情聖給建議.....
- 現實世界，除非你要 debug 某一 package ，否則你都只會看文件看 header file 的
  - 大家不會看完 mysql / postgresql 的 source code 才來使用的
- 漂亮的 source code 可以告訴你現在正在做什麼，但是不會告訴你為何這麼做！

comment 常見用途

- ~~咒罵老闆和 PM~~
- 說明這 package 的用法和限制
  - 例子： optimisticLock 會警告不能高 collision
- 說明這 package 背後所用的演算法和其 big-O
- 說明這 package 的 engineering decisions
- 說明這 package 的負責人
  - 出事時要找的人 / 要修改時找誰來談

淺談 code quality

- Code quality 是高度主觀的
- 「好的 source code 應該讓笨蛋也看得懂」這句是毒藥
  - https://en.wikipedia.org/wiki/CXCR4
  - CXCR-4 is an alpha-chemokine receptor specific for stromal-derived-factor-1 (SDF-1 also called CXCL12), a molecule endowed with potent chemotactic activity for lymphocytes. CXCR4 is one of several chemokine co-receptors that HIV can use to infect CD4+ T cells. HIV isolates that use CXCR4 are traditionally known as T-cell tropic isolates.
  - 這一段我都看不明白，這代表我在生物學知識太爛，而不是作者寫得爛

Design Review

- 軍事上：戰略失敗是不能用戰術來補回的
  - Heart of Iron 是好物， xcom2 也是（謎之聲：喂）
  - 「積小勝為大勝」是戰略上採用遊擊，不是叫你用戰術勝利去補回戰略失敗
- 在 Design review 看重的：
  - 演算法是否用對， engineering decision 是否正確？
  - 有沒有滿足 single responsibility principle ？
  - 團隊有沒有能力去實行這設計？
- 記住： Design review 比 Code Review 重要
  - package 內的 local variables 改不好只影響要 debug 這 package 的人
  - API 設計不良，是影響到所有 package 使用者

做好的 Reviewer

- 請多抱持同理心，別去要求自己做不到的標準
- Code quality 不是一成不變的，請視專案性質和時間來彈性改變
  - 再三地說：不能替公司賺錢 / 省錢的專案，才是專垃圾的專案
- 優先看架構和 class design
- 越是 local scoping 的東西，其重要性越低
- 別要求人家用一個 Pull Request 來把所有 tech debt 都改好

淺談「演算法」重要性

- 回應「演算法在職場上沒用」
- 抓卡比獸的故事
- 我在職場上用過的演算法 / 知識

回應「演算法在職場上沒用」

- 便當店賣炸雞的，也會說告訴你「英語沒什麼用」
- 名句：「學海無涯，~~回頭~~唯勤是岸」
- 書到用時方恨少

抓卡比獸的故事

- 有１０個小精靈訓練員，預知到晚上會有卡比獸在某公園出現
- ５個訓練員帶了多個後備電池做好了充足準備，另外５個訓練員則只帶了手機
- 卡比獸在 11:30pm 才出現
  - 帶了多個後備電池的訓練員抓到卡比獸後高興地離開了
  - 只帶手機的訓練員，他們早便玩光了電池
  - 當他們衝到捷運站充電，然後再跑回去公園時，卡比獸早便不在了
- 卡比獸不會等人的（我也只抓到了一隻）
- 同理，職場上的機會是充滿偶然性。
  沒人等你到時才慢慢花時間補回知識，你做不來，機會就是別人的。

我在職場上用過的演算法 / 知識

- Trie
  - 用來解大量文章和 keyword 的 matching
- BloomFiltering
  - 用來解特殊的用戶流量限制
- TokenBucket
  - 用來系統內部限流保護
- Shortest Path Algorithm
- Monte Carlo algorithm
  - 用來解 O(1) 的 priority queue

最後一句

- 請關注台灣石虎和眾多野生動植物的保育
- 希望.....
  人類衝出太陽系時，會帶著石虎一起同行
- 也希望.....
  明天會更好

完