Lecture 6: Software Security, Society and Economics

> *Note: There is no hard truth to many of the topics, and for example, software companies, hackers, and governments could have wildly differing incentives and opinions. You are encouraged to make up your own opinion. The lecturer's opinions are of his own, and do not necessarily reflect the opinion of his employer – although I'd be happy if they did.*

> *Note: This session concentrates on Finland and the European Union – because we're in Finland. We purposefully do not cover legislation elsewhere in the world, neither do we cover national differences within the EU. EU legislation through directives allows for considerable national variance.*

Part 1: Network effects, externalities, disclosure and vulnerability markets

- Security Debt
  - An economic view to software development often talks about ”technical debt” or ”quality debt”.
  - This means that there are various quality assurance tasks (like testing, or engineering tools required for efficient testing) that can be left undone. This often happens in a hurry, or if quality assurance is not prioritised.
  - The idea is that this can be viewed as a ”debt”: You ”borrow” from the future by moving activities into the future.
  - The concept also has a notion of ”interest” of the debt: The developers may later have to spend more time fixing bugs than avoiding them would have cost earlier.
  - ”Security debt” has been proposed as a subcategory of ”quality debt”.

> Discuss*: What would security debt consist of? What does its interest look like? Can quality (or security) debt be a calculated risk management strategy? Why or why not? (For example: We take a bit of quality debt to be able to meet a shipping deadline, e.g., a Christmas market, that has much larger expected payback than the debt.)*

- Incentives of secure development
  - Incentives driving towards insecurity: ”features first”
    - Functional features are often prioritised higher by software development, because software security is mostly a non-detectable property until someone actually starts attacking. Unless customer has explicit software security requirements, it is far easier for a salesperson to sell a tangible feature than a quality aspect.
    - Time-to-market may be a critical factor, especially when security testing is not integrated into development and maximally automated. ”Extra” quality assurance overhead related to software security can be seen as additional latency.
    - Management often talks about *velocity* which is the amount of value produced in a unit of time. Unless security activities are recognised as work that needs to be doing, they seem to be reducing the apparent velocity.
    - Convenience of use is a very strong driver. This holds true both with usability to end-users and to developers that develop on top of a platform. End-user convenience is often equalled to easy usability, and this may sometimes clash with certain security protocols (key management for encrypted email could be an example). Developer convenience is exhibited by tendency of developers to turn security checks off so that their tests, or even the released app, works; granting extra privileges to an app in order to “get something done” is very typical.
    - For a mass-market product, customers might assume that they would not be affected by a potential security or privacy issue. Especially for privacy issues, customers still think about the first-order effects (”so what if someone can see my holiday pictures”). However, we do not yet even understand the second- and nth-order effects of widely available private information, which could allow far more invasive privacy problems.
      - The 2018 debacle of Cambridge Analytica using Facebook to allegedly flip the results of the United States election, and China using social media for grading their citizens, are examples of the nth-order effects which probably would have been bad science fiction in 2008.
      - As long as these, arguably not well defined threats are not understood, there is also not a very strong motivation for the customers to demand these aspects. Currently it remains an interesting question whether this will actually cause changes in user behaviour.
    - More and more, users are in fact not the customers, but they are part of the product. Most social media sites that are ”free” to users package the users' data and metadata into the actual product, which is sold to the actual customer. Because the user has not paid for the service, they have very little in the way of being able to claim that they would be customers, or would have obtained any sort of product. This undermines the user's capability to demand security or privacy.
  - Disincentives for software vendors to react to software security issues:
    - Lock-in and network effects: Customers cannot often easily change their software vendor. They may be held hostage to a common or popular operating system, or need to use compatible software, or cannot open files with other applications due to DRM, or use a service because all their friends use it. Even if they would be let down by security issues, they may have no real recourse to switch.
    - Software development is iterative by nature, and customers have been conditioned to expect that all software has bugs, and they will get updates (fixing bugs, but also introducing new features) repeatedly. Software companies assume that they can use this patch cycle to fix security issues. Security issues' effects are, at least on the surface, mitigated partially by a quick patch release cycle. However, this only applies if the issues are found and reported to the vendor. If the issue is not found by, or reported to the vendor, the customers do not benefit from the fix.
    - A normal customer cannot tell from the product whether it is secure or insecure. A customer sees the user interface, and feels the user experience. The customer may obtain a distinct value from a software product. In many physical products, shoddy construction and bad mechanics are visible from the feel, and they spill over to the user experience. However, in software, you can actually have a very polished package that is completely broken in security sense. This asymmetry of available information gives the software vendor an incentive to play it against the customer.
    - Fixing security vulnerabilities costs money, and usually vendors are expected (note: often not required) to issue patches and fixes. Keeping security vulnerabilities secret is incentivised in various ways:
      - Vendors could hope that nobody (else) will find a non-public vulnerability, and just let it exist. If they learn about the vulnerability confidentially, they can always claim they did not know about it earlier, to avoid looking bad.

- A vulnerability that is not public can be fixed according to planned maintenance schedules. Out-of-schedule updates are usually always more expensive as they have ripple effects on resourcing of planned work. (Agile work management and Continuous Delivery do, however, greatly reduce this effect.) It also has to be remembered that not all systems can be realistically fixed at all.
- Some, especially institutional, customers, may want to get their fixes according to a pre-determined cadence (e.g., Microsoft's Patch Tuesday on the second Tuesday of every month). Keeping vulnerabilities secret allows hitting this type of customer schedule.

> Discuss*: Can you name any software products that you really could not switch even if they had security issues? Would you need to continue using them?*

- Security as an externality
  - Concept: Lack of software security as a negative externality
  - Externality: A side-effect of an activity that affects someone else than the actors (and usually is not absorbed by the actors). These can be positive or negative.
  - Negative externality: A negative side-effect, usually a cost. For example, airborne pollution can be a negative externality of a factory; or mistreating workers can be a negative externality of producing a cheap electronics gadget.
  - Internalising an externality means that the actor that causes the side-effect covers the costs. An example: Carbon emissions from an airplane would be reclaimed from the atmosphere, and the this process would be paid by the traveller; or that the consumer would pay a price for an electronic gadget that allows workers to be treated well.
  - It is usually well accepted that states (governments) have a right to impose internalisation of externalities through regulation, unless this happens through industry self-regulation (examples: environment, car safety, etc.) Existence of negative externalities therefore has a tendency of inviting outside regulation.

> Discuss*: What do you think about describing software security as an externality? What externalities are there that relate to lack of software security? What would be an internalising activity for these?*

- Vulnerability disclosure
  - Concept: ”Full” vs. ”coordinated” disclosure
  - Full disclosure: A security vulnerability is published in detail to everyone. The software vendor learns of this problem at the same time as potential attackers.
    - A disclosed vulnerability that has no mitigation is often referred to as a ”zero-day”.
      - ”Coordinated” disclosure is also often referred to as ”responsible” disclosure, but that may not be an appropriate word. The thinking goes: If a software vendor has been irresponsible enough to have a security bug, then they have no moral high ground to tell what would be “responsible” disclosure.
      - Coordinated disclosure is usually thought of consisting a selection of the following aspects:
    - Vulnerability reported to software vendor, and potentially some key relying parties, before anyone else;
    - Reporter giving some time for the vendor to fix the vulnerability before coming public.
    - Often responsible disclosure is coordinated through a third party, for example, a CERT (“Computer Emergency Response Team”) organisation (e.g., NCSC-FI, National Cyber Security Centre in Finland). (See the CVE numbering system from session 1.)
    - These third parties are a kind of trusted mediator. From the perspective of an individual security researcher, it is often safer because a large company usually does not want to bully or pressurise a governmental agency.
- *Discuss*: What are the good and bad sides of full/responsible disclosure? There are arguably good and bad sides for the vendor, the customers, and the attackers, but what could they be?
- Bug bounties
  - Concept: Bug bounties (Vulnerability Reward Programs or VRPs), vulnerability markets
    - Some companies provide ”bug bounties”, that is, they offer rewards for responsible disclosure of security vulnerabilities.
    - The bug bounty market has taken off really strongly in the past couple of years. There are platforms for managing bounties, such as HackerOne and Bugcrowd, which have brought bounties within the reach of smaller companies, too.
    - Bug bounties seem to be rather cost-effective ways of getting vulnerability data. A fairly recent study estimated that a bug bounty program can find bugs at the rate of around USD 1000 to USD 3000 per bug in web browsers.
      - Pricing (Microsoft 2014): white market USD 500-20000; grey market >20000; black market even higher
      - Because the bounty market is hot right now, I am unfortunately not sure if these values are current at the time you are reading this.
    - Bug bounty programs also show that external security research finds bugs that software security programs generally don't. The same study indicated that around ¼ of found issues came from a bug bounty program.
    - Vulnerabilities are also brokered. For example, a third party can buy a vulnerability from a security researcher in order to issue a mitigation in their defensive product, and then coordinate the disclosure towards the affected vendor later. (Microsoft 2014 data suggests a vendor bounty program can increase the ratio of direct-vendor-reported vs. brokered bugs, which is usually better for vendor)
    - Vulnerabilities can also be bought and sold on a free market. Interested parties typically include intelligence agencies, military contractors and criminals. Of course, not everybody wants to sell to those parties. Offering even a small bounty to white/gray hats may tilt the balance from full to coordinated disclosure.
    - Bug bounties are used to bridge the ”demand gap” for bugs. If nobody else wants to pay for your bugs (yet), you might want to do so. You may want to do this during, e.g., alpha/beta releases, when you can buy bugs cheaper than after real release. (ref. Microsoft 2014)
- Export controls
  - Exploits are regulated as dual-use goods, and fall under export controls
  - ”Dual-use” goods are goods with a dual civilian/military use, like certain types of high-end electronics, crypto, and so on.
  - Since the end of 2013, trade in exploits has been regulated under the Wassenaar Arrangement. The Wassenaar Arrangement is a successor to CoCom, a Cold War era mechanism for restricting the arms trade; it covers
    - “Software” specially designed or modified for the generation, operation or delivery of, or communication with, “intrusion software”
    - “Technology” for the “development” of “intrusion software”
    - Enforced by an EU regulation in December 2014

> Discuss: *Do you think that selling vulnerabilities is ethical? If not, why? Isn't it a natural market economy solution to negative externalities?*

> Discuss: *Do you think governments should stock up with vulnerabilities that can be used in attack / retaliation? If you were the supreme commander, when would you authorise exploiting vulnerabilities, and against whom?*

Part 2: Software Security Regulation

- Software security is currently not very widely regulated directly. The following types of laws have (had) aspects in Finland that may spill over to the software security side, at least if widely interpreted:
    - Data protection (privacy) laws, most specifically the General Data Protection Regulation, but also including healthcare-specific laws, workplace specific laws
        - Must implement the necessary technical controls (but the definition of "necessary" is left for interpretation)
        - Typically, must guarantee confidentiality, integrity and availability
    - Product liability laws
        - Especially those products which have safety aspects
    - Criminal law covering intrusion into information systems
    - Security evaluation of various governmental systems
    - Secrecy of information in government and laws concerning archival
        - Requires the necessary information security controls that are based on threat analysis
        - Archives must guard against unauthorised access
    - Information security requirements for information exchanged with other countries
    - Copyright law (DRM circumvention aspects; definition of "effective" DRM)
    - Laws on strong authentication and digital signatures
    - Finnish Constitution
        - Confidentiality of telecommunications is constitutionally protected
- Of course, in most of these cases, there isn't any reference to software security per se, or even to information security. Whether or not a privacy or security requirement is interpreted so that it would actually trigger software security activities is just that - interpretation. Many of these laws use language that specifically thinks security as an operational aspect.
- Regulation and standards are usually fulfilling one or both of the following needs:
    - Internalising an externality (that is, mandating something that would not be done otherwise). This is mainly the role of government regulation, but may also be a factor in vendor management.
    - Having a ready-made set of requirements that can be referred to, so that everyone doesn't have to reinvent the wheel. The risk here is that these lists will become "gold standards" and meeting the requirements is what gets optimised, not security.
    - Some standards aim at keeping parties out of litigation and in a contractual world of private arbitration. These are usually security standards of closed ecosystems where the standard is a prerequisite of a membership.
    - Provide common vocabulary and concepts.
- Regulation through legislation usually does not make specific technical requirements.
    - An exception was, for example, Germany's digital signature legislation.
    - Usually, legislation just requires something that is "good enough", and interpretation is left for courts.
    - Often legal help (a lawyer) is required. However, lawyers that can effectively interpret law into engineering requirements are not common.
- Product liability legislation may affect software security if software is a part of a physical device. It is driven by safety considerations.
    - This liability requires the product to cause damage (death, bodily injury, damage to an item or property) to consumer; and the product to be defective (in "reasonable use").
        - Autonomous vehicles are very interesting in this sense, and the German liability regulations for autonomous cars puts software security pretty much in the driving seat (sorry)
    - Interesting examples that can may be covered are therefore embedded systems: software in vehicles, battery charging software, software that controls actuators in consumer products, software that has a role in how much energy the device emits (radio transmitters, flashes, volume levels).
- Some specific areas of software (again, mostly embedded software) fall under safety requirements. Often these are sectoral safety requirements, which will in turn require software security engineering.
    - Robotics, vehicles
    - Healthcare technology
        - Notably the FDA in the US has released guidance documents (2014) on the security of medical devices, when the guidance was earlier rather scarce. In the US, there are also large healthcare players (Veterans' Affairs, as an example) who have their own software security guidance.
    - Software security activities not specifically included, mainly list technical controls and often emphasis on operational.
- On the financial sector, PCI-DSS (Payment Card Industry Data Security Standard) has had a large impact on application security awareness.
    - In order to process credit card data (of the large card brands), PCI-DSS is a contractual requirement.
    - On the surface, mainly aims at protecting credit card information. May have positive spill-over effects on other software security aspects.
    - It is likely that the main reason is, however, to keep everyone who processes card data in a single contractually enforced system with less chance of olitigation outside the card issuers' control sphere.
    - Specifically for payment application development, PA-DSS (Payment Application Data Security Standard) is applicable. It has quite a few of specific security feature requirements (like, on using encryption), and also has a requirement of secure and documented software development in general.
    - Compliance is measured by specifically qualified companies / people who are typically security consulting companies and consultants, called QSAs (PCI-DSS) or PA-QSAs (PA-DSS). At the time of writing in Spring 2018, there are 386 QSA companies and 62 PA-QSA companies (Payment Application Qualified Security Assessor companies). In practice, getting a QSA status has traditionally been currently a ticket to guaranteed full employment (if you like that sort of a job and don't smell very bad).
- The GDPR specifies that organisations could certify against a code of practice, allowing them to show compliance (and possibly get off the hook a bit easier).
    - At the time of writing, none of the codes of practice have yet a formal status. However:
        - For cloud infrastructure providers, the EU Cloud CoC and CISPE (not to be confused with CISSP) are competing
        - For software development, the European Privacy Seal (EuroPriSe) is a strong candidate that has been already in existence during the directive-based legislation

- The most software security centric regulatory instrument in Finland appears to be VAHTI 1/2013, the application security guideline for government (”Sovelluskehityksen tietoturvaohje” in Finnish).
    - Specifies 118 requirements for application development
    - In theory, ought to become the de facto requirements document for any public sector procurement.
- Another fairly important guideline in Finland is the National Security Audit Criteria (Kansallinen turvallisuusauditointikriteeristö, KATAKRI)
    - Applies, for example, when private companies get access to documents having national security significance.
    - Covers not only IT security but also physical security.
    - For example, has an audit question ”How has the security of executable code been ensured”. On the level III & II (”elevated” and ”high” security levels) secure software engineering principles are required from suppliers.
    - Example requirements include threat modelling, robustness testing, and static analysis.
    - Other requirements set out a fairly large number of specific controls on, e.g., use of cryptography.
- On the EU level, a directive 2013/40/EU ”on attacks against information systems” that is in force and is being implemented nationally
    - Makes ”intentional production, sale, procurement for use, import, distribution or otherwise making available” intrusion tools illegal

Part 3: Software Security Standardisation

- ISO 27001:2013 has new language on software security more specifically.
    - The previous version of ISO 27001 (2005) largely treated software security as an input validation issue.
    - The new version has fairly strong language that requires actual software security activities:
        - A.14.2.1 Secure development policy
        - A.14.2.5 Secure system engineering principles
        - A.14.2.6 Secure development environment
        - A.14.2.8 System security testing
        - A.15.1.1 Information security policy for supplier relationships
    - It is likely that once contracts that have references to ISO 27001 / 27002 get updated, software security development practices get quite a bit more pull.
- Software security itself is being standardised in ISO under ISO/IEC 27034.
    - Whether or not 27034 becomes as popular as 27001/27002 (more general information security standard) is yet to be seen, but especially in circles who like standards, this is something to keep an eye on.
    - ISO/IEC 27034-1 defines the following terms:
        - Organizational Normative Framework (ONF): All aspects affecting the software security of an organization, including their ASC library (below)
        - Application Normative Framework (ANF): All aspects affecting the software security of an application [project], a subset of ONF
        - ONF Committee: The folks in the organisation who decide and shepherd the ONF. Loosely aligned with the ”software security group” in an enterprise.
        - Application Security Management Process: The process consisting of requirements definition, risk assessment, selecting the ANF subset for an application from the ONF, actual implementation related issues, use of the application, and auditing.
        - Application Security Control (ASC): A software security activity. Collected into an ASC library.
        - The standard includes the Microsoft SDL mapping to ISO 27034 terms in an appendix.
- Privacy impact assessments (but not the EU GDPR Data Protection Impact Assessments) are standardised by ISO/IEC 29134.

Reading list

This is a list of useful documents that will enhance your understanding of the course material. The reading list is session-by-session. “Primary” material means something you would be expected to read if you are serious about the course, and may help you to do the weekly exercise; “Additional” material you may want to read if you would like to deepen your understanding on a specific area.

> *Note: There are further pointers in the week 6 weekly exercise background material.*

Primary material

- Have a look at NCSC-FI (ex-CERT-FI) vulnerability coordination policy. This explains how they run the coordinated vulnerability process.
- A 2013 study on Mozilla Foundation and Google bug bounty programs, Finifter et al.: An Empirical Study of Vulnerability Rewards Programs. This is where some of the quoted numbers in the lecture notes came from.

Additional material

- The best book I know on the economics of information goods is Carl Shapiro and Hal R. Varian: Information Rules. If you want to read more about the effects and valuation of information assets, this is a very useful kickstart into that side of economics. It might even help you if you’re planning to start a company.
- Jari Råman: Regulating Secure Software Development, 2006. A doctoral thesis looking at the law and economics side of software insecurity. It happens to be available from the University of Helsinki library, and if you are interested in product liability side and economics of software security, it makes interesting reading.
- Christopher Millard: Cloud Computing Law has a good section III: Protection of personal data in the clouds. If you are interested in the problem field of cloud services and privacy, this is a fairly recent (2013) take on the subject. Note, however, that once the EU data protection regulation is passed, parts of this book can get old pretty fast.
- Graham Greenleaf: Asian Data Privacy Laws: Trade and Human Rights Perspectives. Most privacy-related writing is about the EU and US legislation, and there is very little material available on China (and other large Asian economies). The section 7 of this book covers Chinese legislation in more detail that I’ve seen anywhere else.

Endnotes

This is lecture support material for the course on Software Security held at Aalto University, Spring 2018. It is not intended as standalone study material.

Created by Antti Vähä-Sipilä <avs@iki.fi>, @anttivs. Big thanks to Sini Ruohomaa and Prof. N. Asokan.