ISAE

Institut supérieur de l'aéronautique et de l'espace

Embedded Systems Master

2017-2018

(Rev.2 – Work in progress...)

Embedded Systems and Computer Security

Rodolphe Ortalo

Ainsi que la vertu, le crime a ses degrés
Et jamais on n'a vu la timide innocence
Passer subitement à l'extrême licence.

Phêdre (1677), IV, 2, Jean Racine

Document description
Title : Embedded systems and computer security
Created : 2004-05-10 22:34, Rodolphe Ortalo
Modified : 2018-05-03 19:43,
Revision n° : 1530
Statistics : 82 pages, 318456 characters, 2 tables, 7 figures.
Topic : computer security
Keywords : computer science, security, embedded systems

Table of content

Index des tableaux

Index des illustrations

1 Introduction

Security is about malicious faults. That is to say intelligent adversaries trying actively to take advantage of your computer system, usually called attackers. Though computer security is not exactly game theory, because neither attackers nor defenders follow perfect behaviours, it shares with this domain much of the difficulty. On the one hand, it is even harder than a game because attackers may disguise or hide themselves easily on the Internet and do not want to respect any game rules. On the other hand, it is not desperate as the defender has extensive access to the internals of his or her own computer system and can setup all the affordable protections he sees fit. There is no absolute loss or total victory as in real games ; neither next game possibility. So, like chess, practical security is not only about black and white senseless pieces.

1.1 Securitease

As a field, computer security needs little advertisement. For as long as the author remembers, there has always been a lot of hype with security issues and few classical text books[1].

Unfortunately, this is not doing any good to the field, quite the contrary. It is a nuisance. This popularity attracts those who do not like security issues and complexity, but simply tolerates the topic intricacies because it pays them well or offers them some career progression that they would not have access to otherwise. But such people would probably do many things for money – except try to learn skills and solve actual issues – and they do little good to the field. Sometimes, they simply fuel the hype with their own little invention or calumny for a chance to expand their own interests. The first thing to do in security is to spot these actors and flee them for two reasons. First, because they are part of the attackers, the worst ones, the internal ones. And second, for one's own moral equilibrium before being tempted to imitate them to grab the easy fruits for the kids while compromising their education[2].

This popularity is partly due to the fact that security touches everyone, as everyone can be a victim of malicious foes. But then, everyone also thinks such an actual direct concern entitles him or her to advise and regulate on security rules and techniques. Unfortunately, being a potential victim does not mean being competent or able to defend oneself adequately[3] ; even if you are a person with power. Most computer security professionals are frequently being lectured about how they should manage security. The fact that they do not agree *technically* on the specific recommendations they are given is infrequently taken into account. We are all potential victims after all so we all have the right to speak, no? Well, no actually. Real victims have a right to complain (and even of being assisted to do that). Potential ones mostly have a right to listen and comment after the fact (or a frequently ignored duty to testify). So please, first sit down and read. We'll talk again as soon as you have completed reading the bibliography – which in the first year of this course construction, should still be rather easy.

These two paragraphs are an illustration of what some call the *security circus*. That circus is real. I would even say that at the beginning of this 21st century, it encompasses the majority of computer security activity. I am not so competent to comment similarly on general security but it looks like too, and for example Ross Anderson in a classical textbook, goes as far as to say that *the rampant growth of the security-industrial complex since 9/11, and the blatant fearmongering of many governments, are a scar on the world and on our profession. We have a duty to help it heal, and we can do that in many different ways* ([Anderson2008], p.891).

Hopefully, the comedy will fade out as an historical relic when the domain will mature and computers will start to exhibit security properties for their end users (again). But for the moment, the circus performs everywhere on our devices and takes a heavy tribute on the available means.

You had better be warned if you do not want to spill money and energy in an entertaining but otherwise useless spectacle, in your own organisation.

Anyway, taking interest professionally in computer security specifically involves first wondering about general security management issues. The author is not familiar with so many diverse fields or organizations to say that these aspects are really specific to security but from his narrow point of view they do seem to be. So let us browse them, my way.

---

1 A tentative list : [Bishop2003], [Anderson2008], [Gar2011], [Pfleeger2015].

2 Food vs. education is an interesting compromise question for a security professional when you think about it. Of course, the present text may be biased. Note the author is not speaking about how to eat correctly before attending the exam. [Univ48], art. 25, art. 26, [Hugo49], [Hugo50], [Condorcet92].

3 All children know that. Children listen very carefully about security rules. Unfortunately, they grow up too fast.

1.1.1 General security management rules

Organizational security is fundamental for building and later on using secure systems. Once secure systems are built, they may help us maintain organizational security. But for the moment, we will mostly have to stick to manual enforcement of these organization rules to reach a first grade of reasonable security management.

1.1.1.1 Skills

The first aspect linked to people is selecting the appropriate skills for addressing the computer security field. Even that is problematic.

a - High level academic skills

Computer security knowledge involves classical computer science skills. Most of the time, *security protection* aspects emerges at the most advanced levels of computer science specialities: compiler design, specific hardware processing design, formal analysis of programs. Software analysis or *attack techniques* appeal to much more basic computer science skills: assembly language programming, overall memory layout, scripting languages intricacies and erroneous data management. As one usually wants to protect systems more than attack them, the technical skills needed are classical high level computer science skills like those found in any good computer engineering academic curriculum. If you only want to attack systems[4], you may lower your recruitment level a little, but you still need to focus on computer science only.

Additional skills to deploy may vary from mathematical and theoretical grounds with cryptography and formal verification issues to less scientific skills when it comes to understanding and managing correctly the administrative and organizational aspects of security.

Therefore, we should expect the typical profile of a debutant security professional to be a high level computer science engineer with some domain specialization in some computer security specific thing.

So totally banal, albeit A-level scientific education. You would expect the same from another computer science engineer claiming a specialization in one field or another, no? So, there really is nothing special about the skills needed for computer security: they are those of computer science[5].

Admittedly, computer science is still a rather new field: probably only 30 years old. So sometimes, you may be tempted, especially in those speciality fields where academic diplomas did not exists 10 years ago, to believe a candidate who shamelessly claims that autodidacts rule the world of computer security. Well, the truth is that computer science is also *already* 30 years old. True autodidacts are now extremely rare and probably *also* have an engineering diploma[6].

After initial diploma and academic studies, professional experience is evaluated normally ; with attention to the fact that computer science is key to the domain. Organizational security rules and risk management are simply possible useful addons. Project management or people management has few things to do with computer security. Legal management similarly (though it may have a link with general security).

Finally, at the end of this section which probably says nothing, except that you should recruit a computer science professional, let's finish by underlining that, in order to address *computer security or security*, one needs **computer security or security skills**. It should not be worth repeating the obvious, but it is.

Especially in the embedded systems world, whether the computer in question is embedded in a car, a plane, a spaceship or a train is irrelevant. You need security skills first, not avionics, railways signalling, car manufacturing, or whatever domain-specific standard knowledge you may wish. That's nonsense to expect non-security expertise to be more important to solve security issues than security knowledge. And I'd add that if domain specific experts of any of those other fields were able to solve computer security issues they would have done it themselves already. And they would also have solved the security issues of desktop systems too. That's probably again a part of the security circus that, even in the technical domain, all engineers from the other domains claim to be able to solve security problems and that their managers believe them and endlessly support their half-baked solutions. But that's just false, that's pretentiousness and addiction to the popularity of the 'security' buzzword. Hard computer security issues, especially for distributed

4 But why would you do that? You are not allowed to attack systems.

5 And as distinct of project management as any other. Nothing in computing is special for project management (or even acquisition). But that's another debate.

6 First hand. And nearly nobody ever asked me to prove my Apple][c hard earned skills ; which shows too that autodidacts skills probably do not age well. But the 65SC816 hardware bugs were real and instructive. The community too.

systems, necessitate specific computing experience ; and these issues are starting to become a majority in networked embedded systems which are distributed systems in the first place from the security point view. And these distributed systems usually come without any distributed security mechanisms nowadays[7] ; which is somehow revealing of the average embedded systems engineers awareness level with respect to general security theory.

More empirically, computer security engineers that would claim to solve, for example, aeronautical systems issues would certainly be laughed at. What do you expect from aeronautical systems engineers trying to solve computer security issues?[8]

b - Critical behavioural qualities

Outside of computer science and computer security knowledge, which are primarily the topics of this document; some non technical skills are specifically relevant to the security field. One could even say that these requirements are not skills, they are behavioural qualities pertinent to this field.

Because security is about trust, computer security is about trusting the computer systems that we use. And embedded systems security is about trusting critical embedded computer systems which failure could lead to human loss or catastrophic consequences ; in presence of malicious attackers. So the first skills to expect from someone entering the domain are those you would expect in the other fields where trust is a first and foremost requirement. Like for a policeman, or an accountant, a high level of honesty and transparency are *required* skills to work in computer security. Security is about trustworthy people at the highest possible level and in difficult circumstances.

This should not be theoretical. One can be forgiven for presenting oneself under good light at a recruitment interview, a management review or for the corporate picture. But those playing with numbers to present better statistics, those who never want to announce bad news because it could be detrimental to their careers, those who never want to hear bad news because it would oblige them to make choices ; all those who repeatedly play the security circus in fact, should simply be gotten out the field as early as possible. A special mention will be given to those who always defer responsibility to someone else. They are frequently interested in the security field, where the culprit is known from the beginning ; we call it the “attacker”. So final responsibility is never on them. But those who fail to assume responsibility cannot help. Trust implies liability[9].

This rough presentation may sound rude. It is. But the situations implied by security management are rude. Even trusted people failure[10] should be taken into account, so we need to gather as much good will as possible in the first place or it leads no where. Again, this will sound familiar to law enforcement or financial accounting people where these behavioural skills are also determinant.

Given these remarks, do not expect a smooth character but be strict on honesty and transparency, first.

c - The hacking no-skills and certified not-a-diploma

Technical skills are obscured in this domain by the self-proclaimed genius hackers discovering magical computer attacks as teenagers at home ; later becoming pal-acclaimed cybersecurity researchers from the latest industrial company in need of a white-hat alibi to hide their lack of basic software engineering knowledge ; said “researchers” discovering always similar cybersecurity vulnerabilities on their laptops during international flights back from the latest buzzword conference.

Rightly, finding attacks may necessitate computer science skills. But it may not. Some attacks are stupidly easy. Some are astoundingly clever. Some appeal to research level statistical algorithms and signal analysis against the latest cryptography. Others only require college level macro programming on desktop software. Both will look arcane, but nothing decisive will stand out for an outsider eye. You will have to resort to deeper evaluation techniques to have an idea of the technical level of your candidate.

As a recruiter, you may just have been ill advised temporarily when only looking at offensive trends. You just engaged your company on a bad and aggressive profile for a few decades (hopefully the candidate will never go to a management position). But think about the people recruited only against this profile type who are stacked in their endless research of software vulnerabilities. Some try to escape the trap the honourable way reinventing classic decades old testing tools before migrating to software engineering functions. Others simply keep on bookkeeping endless

7 The juxtaposition of several point to point security mechanisms does not make a distributed security system, even if there are many of them, especially if there many of them, then the protocols start short-falling.

8 The aeronautical domain is taken as an example here both for personal reasons and given the expected audience of the lecture. Next targets are starting to appear.

9 Childish finger-pointing must be left to the elementary school.

10 For example under threat.

inventories of bad software while whining constantly for wage increases[11]. And organizations relying only on these people are doomed to setup huge security circuses internally and face increasing difficulty addressing actual computer security problems.

The competent security professionals who focus on protection and do no try to attack software feel a little lonely at the moment. At least, they have their moral for them.

Others pieces of the security circus that obscure the vision are the paying security "certifications" popularized by many groups of professionals. These certifications are the collective variant of 'self-proclaimed experts'. Contrarily to scientific learned societies, which are backed by academic institutions nomination rules and legitimacy, these groups are mostly self defining and valorizing their products like many companies. This does not mean that a certification is effortlessly obtained. But one has to realize that it does not mean much in terms of selectivity among people profiles. It usually primarily means that one has thoroughly read one specific (big and technical) book and successfully answered a lot of (automated) questions about it. And paid the fee to access the book and the questions list.

It does not mean that this specific book is a good one. It may be a lower than average book. It usually is a pretty consensual one, so again nothing decisive. But worse of all is the fact that such certifications are *competing*. Security certification (and possibly certifications in general) encourage people to read and trust one book only – the one upon which designers base their certification tests and their global knowledge value. (Possibly split in several steps of increasingly expensive fees.)

What would you think of a teacher who would recommend his students to read one book only (certainly his one) and not try to search the literature to balance several books and compare several authors opinions? This is what I came to think about professional certifications[12]. At best the time taken to obtain them can be better spent on other things ; at worst they will lock you under a specific mindset which may be totally outdated sooner or later.

And, you can be sure that a wise network manager with high level engineering diplomas will very fast learn how to address network security issues accurately and intelligently. If he is trusted and well advised, he will solve problems faster than a network-security-certified robot will be able to list all its possibly-useless out-of-context and soon-to-be-outdated components-off-the-shelf. I have seen it all the time[13]. So certified professionals are soon outperformed by other computer science engineers, fuelling (a).

1.1.1.2 Money

The issue of money, and more generally material means, in the field of security and computer security is also pretty difficult to address.

a - Under threat or in full confidence

Security is the kind of expense all of us would happily discard entirely. Seriously. We all would love to live in a secure world where, whatever valuable, each and every thing would be secure, and everyone, regardless of origins, would be nice to everyone. It would be great, and indeed extremely economical because full security granted by society would cost us nothing[14].

Security is also the kind of expense we could multiply by ten as soon as tomorrow morning because we suddenly realized we were living in a dangerous world. Dangerous means that we can be or already have been the target of criminals that may destroy, deface or steal something valuable to us just for the fun of it. Maybe they have already done it so it is too late or maybe it is just fear raised by some neighbour mishap ; but the sensation is still so present that you start to add cameras everywhere, to install firewalls everywhere, to hire a lot of self-proclaimed security experts that will confirm your nascent feelings and finally all of the executives committee fall into costly paranoia[15].

11 Probably due to the fear of being fired if the true content of their activity is uncovered. Fear can make people succeed at doing incredible things.

12 And do not distort my reasoning by requesting all possible certifications! All their courses look the same. The real weakness of my argument is section 2.

13 So in some sense, a parrot makes a better security consultant than a robot. Do not ask me what parrots can do to robots.

14 I am already hearing would be activists adding "except our liberty" and warn against surveillance-state dictatorship, but wait for the next footnote.

15 Do you really think a privately-managed self-inflicted dictatorship is any better than state dictatorship? The truth is that reasonable security delegation to public powers under transparent law enforcement is not negotiable ; and that serious matters necessitates serious thinking, not thoughtless reaction.

None of the attitudes is reasonable. But both are frequent in the security field. That's a real problem for security professionals. This money pays wages but I have seldom seen it managed reasonably ; or more exactly, managed at all. The most disturbing question you can ask to an organization nowadays with respect to computer security is : " *How much is the computer security budget?* " and check its content if you are given a figure to verify the most common items are within (including estimates of your own future pay). If you are satisfied by the answer in a specific company, give me a call and sign immediately[16].

b - Not infinite

When not under the innocent restraint of best world idealists but nourished by media coverage or political postures, security budgets can be comfortable and made available in surges of attention. Unfortunately, this fuels the opportunistic behaviour of fast commercials which are ready to give the first attractive technical idea to take the money. Similarly, employees from the information technology department may be interested in focusing all the available security money on one big budget (for ease of management issues primarily) which tends to concentrate naturally on isolated significant projects. Such a combination of demand and supply favours fast selection of candidates and big monolithic single-does-all solutions (whether technological or organisational). Unfortunately, this is probably exactly the opposite of what a technically difficult, fully transversal and permanent problem field requires : focussed, numerous, well chosen and well coordinated solutions.

So, when money is available for security, which has been the case in several places in the last decade, it is not necessarily spent adequately on the most interesting security options. However, for many years, the general consensus, including among security professionals, has always been to consider that, even if these investments are not optimal, they are useful for computer security in general (and they pay the bills).

But this is not the case. Admittedly, it pays some of my bills. But available means and available money is not infinite for computer security. Not at all. Such budgets allocation is in competition with strong opponents, like directly profitable activities in commercial companies, production-oriented investments in any other organization and even offensive weapons acquisition in the "no security compromise" military domain. On the contrary, to enhance its scope, the security budget can only count on things like paranoia (an unreliable and generally questionable ally) or risk analysis results (that is to say the work of resources usage optimization and minimisation specialists) for spending support. So, outside of paranoid surges[17] and war conditions, the budget is objectively far from infinite. And that is a normal situation.

So security is a field where paying attention to the spending decision is important and the accurate selection of working and optimal solutions with respect to the protection needs and the security objectives is necessary. The cost of a solution should neither be a deterrent or an invitation. The adequacy of a solution is a necessity regardless of its cost, even a high one. (Furthermore, by experience, an expensive solution is likely to be an inadequate one similarly to a technically overpowered inadequate one.) On the other side, in security more than anywhere, there is no free lunch. If a solution is inexpensive : either your security objectives are erroneous or someone else is paying for them (and you may be thankful but still prudent).

A limited budget means also encouraging an attitude pretty difficult to enforce in practice. Implementing a bad solution will prevent you from switching to the correct one later on. So it is probably better to *take* a risk than to spend money on a solution that will not work correctly. If you did not find something satisfying but you select a solution anyway just to do something, in the end, you will still take the risk and you will have spilled money which may have been spent more adequately on a better option (including something entirely unrelated to security). Even legal concerns may not be so much covered by inadequate spending.

This is not at all the reasoning of a production environment. It has links with insurance coverage or financial risk management. It usually defeats return on investment logic and is not familiar with many managers outside of top level executives (who do not usually master the effectiveness details until it is too late to reverse their lack of wisdom). This is an extremely unpopular statement at the moment too, because most of the industry has invested big money in big (network filtering firewall) projects, in big (white-hat hacker or software update administrator) teams or services and the recurring costs of these things now dry out any new additional security projects. At the same time, the associated managers now feel so much responsible of the situation and of the associated teams and costs, that reconsidering the strategy would mean putting them into question, which is obviously counter-intuitive for them. Current executives, mostly under influence by the cybersecurity industry, are then totally deaf to logical reasoning.

16 The attentive reader may have noticed the ordering of events has been intentionally perturbed as a security exercise for him.

17 That may even compromise budgets for later years.

However, spilling money without being able to justify all of it extensively is a recipe for disaster in the long term. As soon as the industry consensus over suboptimal common usages will fade out in favour of strong security mechanisms, the players who did not evolve will simply have to disappear.

Furthermore, with respect to security *per se*, this is simply a lack of professionalism. Security is about discovering proofs about an attacker behaviour or a legitimate user demand. Transparency and accountability are basic demands to organizations in charge of security. Of course, these requirements should apply first to all their security expenses.

c - Transparency / accountability

However, in many cases, security managers have a slight bias to resort to confidentiality statements when asked about their budgets… But not only when asked about their budget, also when asked about proofs of their attacks statistics for example, they warn about not being able to show information due to integrity risks or whatever (while obviously, they should pursue legal action in public if they had real proofs of intentional damage attempts).

Unfortunately, the author has now enough experience to conclude that the lack of pragmatic, understandable and verifiable elements about security statements means these statements are void. There is no specific trust to have in someone claiming security failures, alleged attackers or found vulnerabilities if he cannot prove them to you. Similarly, there is no point in trusting security mechanisms that their promoters do not want to explain or that they prevent you to examine for whatever confidentiality reason (or even because you do not have authority for looking, see next section).

People are usually proud of good security systems. They show off, they show you how it works, how strong it is. Sometimes even carelessly. They rarely hide them and certainly not among their peers so in the worst case they can redirect you to someone else who will tell you they trust a system because some explanation was given.

It is the lack of security that makes managers appeal to false confidentiality reasons or missing certification standards.

But, here, transparency is not a philosophical or political attitude. It is a requirement for adequate operation. Security professionals obviously cannot share credentials of systems under their control, but there is little reason for them to refuse to explain the mechanisms they set up especially to those relying on the system. Whatever the system and even at the highest levels of security[18] if nothing can be known about the security of a system, maybe it would be wise for said users to simply reconsider their trust entirely. All precedents have demonstrated spectacular failures : “secret mechanisms” are really for kids assets.

Complementing transparency there is an accountability requirement for most of security managers actions. And by accountability, we do *not* mean exhaustive microsecond precise logging of petabytes of useless traces. We mean the responsibility of actions should be clear and available for everything. If Alice has decided to revoke Bob’s access rights to her personal agenda, this is Alice and Bob problem and they should sort it out themselves whether Alice simply misclicked or has decided to break engagement. In any case, Bob had better accept the situation and find himself another tennis partner for friday ; and Alice cannot expect any serious security staff to hide her accountability in the access rights modification either.

This accountability must extend to security staff actions, especially when it involves special access rights like those allowing to bypass normal security rules or perform investigations (either a posteriori or through anomaly detection software)[19].

Of course, it also extends to security expenses which all should be justifiable and associated pretty precisely to specific operations. And this may apply not only to computer security administrators but also to several information system managers as well with current operating systems techniques[20].

Making finally the link between budget usage and accountability, we usually fall under the next section because it frequently reveals in an organization that many people claim doing security expenses, independently of security management.

---

18 Nobody sane ever wants to know the codes to launch those famous nuclear missiles, except the one who carelessly ran for elections and became commander-in-chief against all odds. I would shamelessly advise him or her to ask army officers to share publicly something about the chain of command security ; and if nobody wants to say anything publicly, to ask for a new command system. And repeat until something is said. (“**#%!-ing professor!*” does not count.)

19 Law enforcement officials would *love* you if you provided them the application and an automated way for justifying and tracing all their actions while they perform an investigation in a computer system. And yes, I am shamelessly trying to bribe everyone to the cause (in the name of universal progress).

20 Tell us how much all these security updates deployments cost exactly! And no, the browser version upgrade is not necessarily security related.

1.1.1.3 Authority

That's the first thing to note with respect to authority of security management. The security budget should be managed by security managers ; and the computer security one by computer security managers. Modern advanced analytical accounting may offer some highly sophisticated ways of counting money spent on security and some organizations may want additional control (and diverse data) on the spending decisions ; but the truth is that centralization of security expenses is simpler. It should be an initial step to simply know how much is spent on the topic. And those most competent to define the budget usage are those who have the skills for this evaluation (see 1.1.1.1). Spreading expenses decisions or evaluation is just budget mismanagement. The latter is not only frequent, it is a sign of the lack of maturity of organizations with respect to security handling[21]. But like giving advice on security, everyone loves to spend security money, possibly even more than giving advices.

We do not necessarily mean that computer security management, and its associated means, must be independent from the information technology department (or the security management from overall logistics for example). The IT officer may very well want to assume directly computer security responsibility, and isolate a computer security budget inside his or her IT expenses. However, it means that frontiers must be well defined and aligned with finances and responsibilities.

Because authority is key to security management. In the information system, it is even the heart of security to provide the basic blocks usable to distribute and manage authority areas over the organization computer system. At the organization level, authority of security personnel is usually the most worrying concern. Because skills, money and transparency are so rarely aligned with the needs of the activity, the authority of security managers frequently simply fades away behind the one of their stakeholders. IT managers, developers, job processes holders, legal departments, executives all appeal to security management for enforcement of their own view of how security must operate. Most of them do not intend to share budget or personnel over a new or invading activity that they had rather isolate or absorb (if possible). Furthermore, given its scope and its natural association with spectacular failures, hype or false alarms, security is a perfect new excuse for most of usual organization perturbations.

Needless to say, the authority of security management suffers seriously from all of these internal competitors, sometimes even when they are claiming to help.

The only key issue is to define clearly authority. Obviously, a narrow and restricted perimeter for security management will be associated with a small budget and an overall boring activity of basic security micromanagement that no-one else in the organization wants to do. A wide authority perimeter going from design to operation in a top level industry may give an engaging challenge and a tremendous budget for security for computer security people recognized in the whole company.

Note that this scale is orthogonal to success. The smaller perimeter may be perfectly managed by few skilled or dedicated people. The wide perimeter may be a wide failure due to skills mismanagement of numerous people, unaccounted money expenses and good security marketing hiding the whole thing for years. Either way will lead to small security advances for the world in the end (albeit much more cheaply in the first case).

The problem with these small advances is that, combined with the exponential expansion of computer systems in all areas of activity, they may lead overall to a significant degradation of computer security as a whole. Some indicators of this situation presented in the next section, heavily supplemented by some decade old similar observation made in [Spaff03], are in fact at the origin of the comments made up to now.

1.1.2 CVE and statistics

The most popular public database referencing software vulnerabilites is the CVE database (CVE stands for Common Vulnerability Exposure), managed by MITRE (cve.mitre.org). As time flows, this data was established by MITRE as the most valuable reference in this domain with the additional important quality of being independent from a specific software manufacturer.

Figure 1 presents the evolution of the total number of vulnerabilities recorded in the CVE database, since the end of the nineties.

21 And the first thing certified-but-non-section 1.1.1.1-compliant auditors forget to check in security maturity level evaluations.

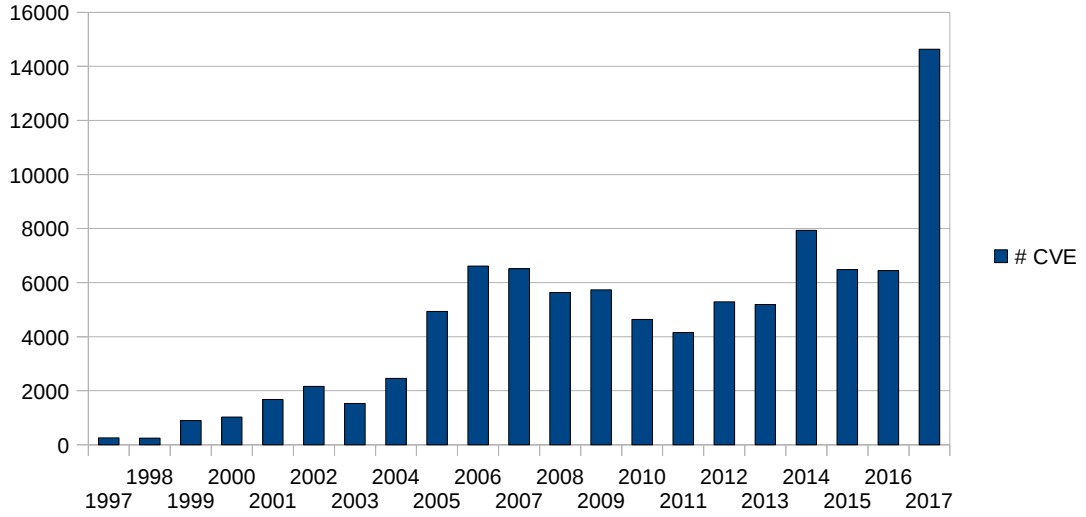


Figure 1: Evolution of the total number of vulnerabilities listed in CVE

From the analysis of this figure we see the current trend of the number of known vulnerabilities in common software since the end of the nineties. We are obviously breaking record after record of total number of knows vulnerabilities. Your interpretation may vary of course. You may feel safer thinking that all these vulnerabilities are corrected now and that we correct more and more of them. You may question the figures because such raw numbers do not take into account the severity[22] of problems or, obviously, the value of the assets held by the vulnerable computers. Or you may feel like the author pestering on overall security degradation and wondering if all those software companies and their developers are aware of the record numbers of security vulnerabilities they are producing.

Something clearer in the end is how attackers operate to attack computer systems. Most of the time they simply use known vulnerabilities: there are dozens of them newly made available everyday. Why bother searching them? Only the most skilled and dedicated attackers try to exploit original ones, most probably in governmental agencies. Note also, the least scrupulous of these attackers could simply try to install hidden vulnerabilities into popular software. Which may or may not be counted in the above figures if they are well hidden [Thomson84].

The above number taken from the CVE database is by definition just a lower bound.

And the most important thing in the database is the information on the vulnerability existence and the corresponding software, not the overall count. In practice, one may evaluate one's own computer system situation based on this information. The author feels much obliged to MITRE for maintaining such an invaluable source of information against slings and arrows for many years.

1.1.3 The embedding of computer security into things

Most worrying is the fact that these unsecure systems may now become critical embedded systems too.

The easiest way to fill this section with fancy convincing elements is to use pictures.

Nowadays, you just have to browse the network with your smartphone to find pictures of critical embedded unsecure[23] systems. Just take a picture of the smartphone itself in the first place to notice your first security issue: that is to say, try to borrow your neighbour's phone to take the photo and then try to share it correctly. Then go back to your browsing session just to quickly pace through the videos of the latest armed land drones initiatives from armies, astonishingly[24] already a decade old. Have a look at that connected home electricity smart meter device linked to all French homes. Tentatively compare all these futuristics cars that grown up rich kids can remote into the garage (from the former smartphone) but that will drive themselves (soon™) and avoid all the real kids playing around. Try to keep count of all the rotors of these drones flying around houses, except the White one due to the latest security patch. Try to reboot these screens popping up everywhere to entertain you when you are being driven, carried or flown for more than an

22 Check http://www.cvedetails.com/ for that. Unfortunately, not much more reassuring than the raw count...

23 If only because you cannot find anything convincing about their security.

24 You and me: naive.

hour: maybe arguing with the charming crew on the true danger of these devices is the real entertainment (unless the device calls security personnel immediately). But please, do not touch or even wirelessly perturb these medicine insulin pumps that connect to one of your relatives when, on (hopefully rare) occasion you walk through the local hospital[25]: manufacturer pictures are enough to understand that remote access will be absolutely critical for someone in medical systems.

But using pictures is not the most pertinent way. The problem with computing devices is not only that you see them pop up everywhere and constantly fail on bad security practices. It is that you do not see all of these embedded systems hidden in bigger systems. It is also that nothing is said about their security, or so little that you cannot even identify those that may have tried to put some effort into protecting them, or their users, or their owners, or their data, or some other data, or well, protect something computer-related in the neighbourhood. Another concern is that the security standards which are supposed to be used to design the security functions in these specific domains may have yet to be written. That the certification authorities are innocently waiting for those standards to be written; while the manufacturers intelligently wait for the certification authorities to start to write them themselves while mimicking all the efforts they put up internally on their own business protection as initial start-up activity on the standardisation topic.

In front of such a virgin land, the best way to raise readers awareness to the problem of computer security with current and future embedded systems may be to appeal to their imagination. Go back to memories of any science-fiction movie of your choice and remember that killer robot, this assassin drone, these god-like omniscient governmental police services, these planes, cars, and trains all crashing at will, misguided by vulnerable autopilots, these self-replicating bugs eating planets, or the initial sin on *USSC Discovery One*.

Through imagination, you may realize that embedded systems and computer security is one of those invisible issues that constitutes a strong technological barrier between the promises of computer systems and actual technological advances. But that would still be imaginary. So let's have a look and try to throw our own stone to break that barrier.

25 At least, if you walk, things are not so bad.

2 Fast paced computer security walkthrough

In this chapter, we will walk at a very fast pace through the general security mechanisms and protection techniques applicable to computer systems. We will not do a detailed analysis of these various areas as it is usually required to resort to dedicated literature to address some techniques in detail. On the contrary, we will browse through them in order to give the reader a global idea of their applicability to computing in general.

We may also specifically focus on some of the numerous misuses of techniques commonly found in this field where practitioners frequently buy or sell subtle and complex variants of digital snake oil[26].

2.1 Security properties

In the classical terminology of computer systems dependability established by [avizienis2004], security is defined as a combination of three basic properties: confidentiality, integrity and availability.

Confidentiality is the property of information not to be revealed to non-authorized users. First, of course, it means the information system should be able to prevent users from reading confidential data unless they are authorized. But less intuitively, confidentiality also means that it may be necessary to prevent authorized users from communicating confidential data to non-authorized users. This involves controlling more of the information flows potentially existing inside the system. In practice, ensuring the confidentiality of a piece of information may involve controlling the copying of a file containing it for example.

Integrity is the property of information to be accurate. It aims to prevent an inadequate alteration of data (either a modification or mere destruction), either because it is performed by an unauthorized user without any "write" access to information or because it involves an authorized user trying to forge illicit data while preserving its innocuous appearance. In practice, one can be sure that a forged financial ledger will look like a valid one and, for example, will certainly exhibit equality between income and expense totals even if it contains fictitious transactions.

Availability is the property of information to be accessible when it is needed by legitimate users. So the system should probably offer reservation mechanisms to allow access to authorized users for reading and writing when they request it. It should also prevent any user from monopolizing resources in order to prevent others accesses (so also pretty powerful resource management). In practice, especially considering current asynchronous and time-unconstrained technologies, availability in front of a malicious and powerful attacker is frequently considered to be very difficult to achieve[27]. Many companies simply resort to allocating enormous amount of resources in order to overwhelm most common attackers means ; an approach which obviously cannot work with respect to attackers controlling key elements of the information system infrastructure or simply those matching them in term of raw capacity[28].

When speaking of security, we mention the confidentiality, integrity and availability of information. But what is the information we are dealing with in fact ? Information can be taken in the sense of usual data, like the one most computers manipulate and store in files. But data is not only in storage or at the computation stage. Data is also typed (at the keyboard), generated (by sensors), displayed (on screen) or transmitted (on a wired network or in the air). And the security of data at all these steps must also be considered in order to protect information – not only when it is stored on a magnetic disk[29].

But there is also of lot of hidden information associated to other data and accessed by the computing processes that is pretty important to the system security. This information is commonly regrouped under the denomination of "meta-data". File access rights are typical examples of such meta-data which importance to security management is immediate. But other things like identities, names, pathnames, time of computation, usually associated to some information or processes in a computer system are frequently as important to handle correctly for maintaining its security than regular data operation.

26 Whether the snake or the oil is digitized first is left as an immediate research diversification topic to the aspiring practitioner.

27 That means many experts just expect you to forget about availability. We strongly suggest to grab the occasion to forget about them too.

28 i.e.: governments, network operators, etc.

29 "*Of course*", I hear many say. But who knows the bandwidth of a VGA cable and its actual radiative falloff distance ? Note old standards still matter until all the interesting conference rooms discard them.

Many other properties are found in the literature. Being faithful to his own teachers opinion[30], the author will stick to the vision that most of those other properties can be reduced to a combination of either confidentiality, integrity or availability applied to specific instances of data or meta-data. For example, anonymity is the confidentiality of user identity, non-repudiation the availability of the sender identity combined with integrity guarantees of the data itself, etc.

2.2 Attacks categories

Most people interested in computer security are frequently attracted initially by the perspective of learning things about attacks and attackers. Fortunately, this hope is frequently deceived. Fortunately because it is not the aim of a computer security course to train new attackers. All those trying to embrace the career of cyberwarriors currently may call it unfortunate, but the author does not agree with them or more precisely with their ill-advised self-proclaimed commanders usually simply trying to advance their own career agenda without caring about the consequences.

2.2.1 The unknown

Knowing attacks, working on attacks, in computer security like in cryptography, should exclusively aim at improving the existing protection systems. It may involve finding flaws in current systems, even hypothetical ones, in order to propose ways of eliminating them entirely or to evaluate residual risks inherent to real systems. Sometimes you also want to double-check alleged flaws are real (especially when reported by an innocent third-party). But implementing security attacks like regular software testing programs is the sure recipe for missing the security target.

So, when studying attacks, we will first do some pretty abstract work about classification and overall modelling hypothesis. And first of all, we will assume that we do not and will not know much about actual attackers.

In practice, the innocent computer scientist exploring the computer security domain is in a much worse position than the beginner player adventuring the realm of chess masters world championship: in chess, at least, you can name your opponent and he agrees to follow the rules when moving his pieces. Attackers evidently do not obey this logic and first of all, will not reveal anything of themselves if they can avoid to do it. They will even try to disguise as much as possible as existing innocent users. Henceforth, all those statistics about attackers you find in the newspapers will usually reveal more of the intent of the statistician than of the (usually empty) covered class of attackers[31].

2.2.2 The assumed

Faced with so much uncertainty, we rational human beings of course react humanly: we classify and regroup under convenient etiquettes what we cannot evaluate certainly[32].

This is actually pretty legitimate because managing security, like classical risk management, is about managing uncertainty and necessitates a lot of assumptions. A significant part of the work is to make sure these assumptions are not entirely out of scope.

Up to our own knowledge, a pretty good attempt at defining interesting classification axis for computer system attackers was proposed in the ITSEM ([ITSEM93], §3.3.29-32, §6.C.28-34).

It started by trying to provide a rating of the strength of security mechanisms among a simple scale of three levels: basic, medium and high. The emphasis of the rating is on the amount of effort required to exploit a vulnerability (not discover it or later reading about it). The evaluator rating of the strength of the mechanism is based on several aspects. Let's study the actual text of this reasonable[33] standard :

> « **Estimating the Strength of Mechanisms**
>
> 6.C.28 According to the ITSEC (Paragraphs 3.6-3.8) the meaning of strength of mechanisms ratings is as follows:

30 As well as finding the argument both convenient and somehow elegant.

31 When you think about it, such statistics are in fact extremely useful currently, to understand some of the practitioners of the security circus.

32 The younger or the most foolish which only realizes that those things may actually do hide in shadows also sometimes take the road of a paranoia. It proves extremely hard to cure. The usual “last kiss in bed” protection measure stops working after around eight years old victims.

33 In the sense that it has reached the age of reason. All readers trying to speak of an “old” standard will be kindly requested to decline their own birth date for comparison. As with many aspects of computer science up to now, reasonable solutions are frequently discarded in favour of brand new unproven but fashionable tools.

a) For the minimum strength of a critical mechanism to be rated *basic* it shall be evident that it provides protection against random accidental subversion, although it may be capable of being defeated by knowledgeable attackers.

b) For the minimum strength of a critical mechanism to be rated *medium* it shall be evident that it provides protection against attackers with limited opportunities or resources.

c) For the minimum strength of a critical mechanism to be rated *high* it shall be evident that it could only be defeated by attackers possessing a high level of expertise, opportunity and resources, successful attack being judged to be beyond normal practicability.

6.C.29 These definitions are informal, intended to be meaningful to users of a TOE. This subsection gives guidance on more objective means of measurement.

6.C.30 Since strength of mechanisms concerns expertise, opportunity and resources, it is necessary to expand on the meaning of these terms:

a) *Expertise* concerns the knowledge required for persons to be able to attack a TOE. A *layman* is someone with no particular expertise; a *proficient* person is someone familiar with the internal workings of the TOE, and an *expert* is someone familiar with the underlying principles and algorithms involved in the TOE.

b) *Resources* concern the resources an attacker must expend to successfully attack the TOE. Evaluators are usually concerned with two types of resources: *time* and *equipment*. Time is the time taken by an attacker to perform an attack, not including study time. Equipment includes computers, electronic devices, hardware tools, and computer software. For the purposes of this discussion,

- *In minutes* means an attack can succeed in under ten minutes; *in days* means an attack can succeed in less than a month, and *in months* means a successful attack requires at least a month.
- *Unaided* means no special equipment is required to effect an attack; *domestic equipment* is equipment which is readily available within the operational environment of the TOE, or is a part of the TOE itself, or can be purchased by the public; *special equipment* is special-purpose equipment for carrying out an attack.

c) *Opportunity* covers factors which would generally be considered outside an attacker's control, such as whether another person's assistance is required (collusion), the likelihood of some specific combination of circumstances arising (chance), and the likelihood and consequences of an attacker being caught (detection). These factors are difficult to rate in the general case. The case of collusion is covered here, but other factors may have to be considered. The following forms of *collusion* are discussed: *alone* if no collusion is required; *with a user* if collusion is required between an attacker and an untrusted user of the TOE for an attack to succeed; and *with an administrator* if collusion with a highly trusted user of the TOE is required. This definition of collusion presumes that the attacker is not an authorised user of the TOE. » ([ITSEM93], §6.C.28-30)

Such assumptions about the attacker categories one system faces, or more precisely is supposed to resist to, are adequate for evaluating the protection of a system. They justify studying specific attack techniques in more detail in order to clarify the rating of these various aspects.

Detailed implementation of a vulnerability exploit up to in-the-field execution capabilities evidently goes much further than this rating necessity. Sometimes it may be useful to fight scepticism, but even there, experience shows that it does perform pretty poorly or is only effective on niche issues. The actual reasons for wanting to explore systematic practical implementation of attacks finally seem pretty unreasonable.

On the contrary, sketching such implementation can be fruitful in order to explore limitations of protection techniques and improve them.

2.3 Elements of cryptography

Cryptology is an essential tool for computer security. However, it is the author duty to first warn the reader that our text is *not* a cryptology textbook. Cryptology is a mathematical domain. It does not allow improvisation or approximation. The author is far from mastering enough of the field to do more than speak of cryptology. And also make note that it seems to be both a new and hard mathematical domain. It is also a mysterious and fascinating topic leading to all sorts of erroneous or abusive statements.

Then, the objective of this section is primarily to offer the reader the overall knowledge of cryptologic tools needed to use them correctly and prevent some of the most dangerous misuses found in the field. Readers wanting to know more about cryptology are invited to refer to more competent authors work, such as [Oppliger2011], [Handbook1996], [Schneier93].

The first mistake in this field is the confusion between cryptology, cryptography and cryptanalysis.

Cryptology is the combination of two domains :

- *cryptography*[34] which aims at producing hidden messages, not understandable by third parties ;
- and *cryptanalysis*, which aims at discovering these hidden message, decrypt them.

Other vocabulary clarifications may be useful. Avoid confusion between cryptography and *steganography* ; the latter addressing covert information. Sympathic ink or watermarks are example of steganographic techniques aiming at *hiding* information (usually among other unrelated data).

Encryption is the process of converting ordinary information, called *plaintext* (or *cleartext*) into unintelligible text, called *ciphertext*. *Decryption* is the reverse, moving from the unintelligible ciphertext to the original plaintext. The pair of specific cryptographic algorithms performing encryption and decryption is called a *cipher*. Most of the time, operation of these algorithms involves a specific secret information set called the *key(s)* – which may involve several elements.

As a domain of mathematics, defining new correct cryptographic algorithms is difficult. Many of the proposed algorithms have been broken after a while. Important fundamental advances in this field, both on the cryptographic and the cryptanalytic side, are pretty recent compared to the usual time scale of mathematical results. For example, public knowledge of public key systems is from the seventies, differential cryptanalysis from the nineties[35]. These recent results are sometimes not very well demonstrated and may exhibit some theoretical weaknesses or inaccuracies.

Implementing a cipher using a computer program is difficult too. The internal parameters, the timing of program execution, the padding of empty blocks all can lead to decisive information leaks that compromise the whole algorithm security. And finally the environment of the algorithm can have an influence too on the overall security as initialisation may involve big random number generation, secure storage, deletion of temporary data, user interaction, etc.

All these elements factor into making cryptographic engineering a difficult task.

However, with all these warnings made, it has to be said to that the practical progresses in publicly available cryptography have been tremendous. Nowadays, with a very common computation device, it is totally possible to operate on more data than one would probably ever need in a lifetime some cryptographic protection algorithm that the most highly ranked military officials of the most powerful nations would have only dreamed of half a century ago. Actually, it seems that this is now a part of the problem, because those officials descendants are starting to get mad at the fact that they were not the exclusive benefactors of these advances and fearing that the regular public would use them for evil purposes[36].

2.3.1 Overall view of an encryption algorithm

34 Impertinently, Wikipedia says “some use the terms cryptography and cryptology interchangeably in English”. The author notes that Wikipedia is then undeniable on the issue ; and himself vehemently denies all signs of jealousy in this footnote.

35 All these results are even younger than the author ! You could very well fall on the inventors at one of the social events where these arcane magics are celebrated ; or (less probable) even hire them if you are wise and rich.

36 While thinking about it, hopefully, this is not exactly like in the statistician case (see note 31). Hopefully.

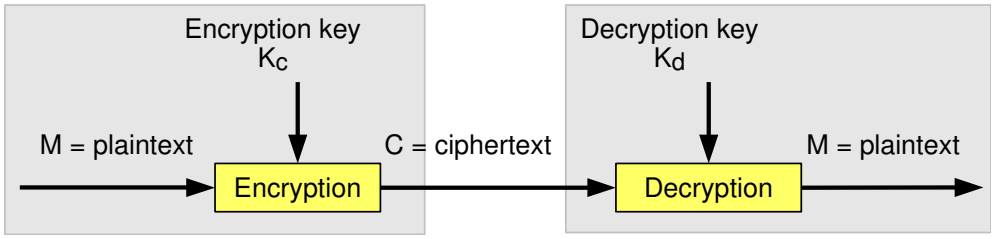


Figure 2: Overall operation of an encryption algorithm

We will use the following notation to denote encryption and decryption, using the parameters presented in the figure above:

- encryption : $C = \{M\}_{Kc}$
- decryption : $M = [C]_{Kd}$

A good cipher must offer several properties in order to ensure the confidentiality of message M data :

- it should be impossible to find M from C without knowing Kd ;
- it should be impossible to find Kd, even knowing C and M (*known cleartext* attack) ;
- and it should be impossible to find Kd, even knowing C while *choosing* M (*chosen cleartext* attack).

In all three cases above, one should make the hypothesis that all the details of the algorithm itself are known to those who could try to break these properties, but not the keys of course. The statement “impossible” in these properties is to be taken both in the usual sense and in the algorithmic sense. It must as improbable for a competent and well equipped attacker to violate one of these properties as it is for a layman to guess the right solution at random.

Such a high level view illustrates the main improvement brought by encryption algorithms. Encryption algorithms do not really solve any information protection problem, but they allow something that was previously impossible, they allow to *move* this problem and transform it into the problem of encryption keys protection which we hope will be easier to solve than protecting all the original information in the first place.

2.3.2 Symmetric ciphers

When the encryption key Kc and the decryption key Kd are identical, the algorithm is a symmetric *cipher*, which single key is usually denoted K (Kc = Kd = K).

All publicly known cryptographic algorithms were symmetric ciphers until 1976 (and the publication of the first asymetric cipher). The most recent and common examples of symmetric ciphers are the two standard encryption algorithm : DES et AES.

DES (*Data Encryption Standard*) was officially defined in 1976. It is an encryption algorithm using 64 bits data blocks and 56 bits key (with 8 parity bits of protection). DES design spanned several years : from a public base proposed by an IBM team, the algorithm was improved[37] several times by teams from the NSA before being submitted back to the (very suspicious) scrutiny of the original IBM team. The algorithm design is clearly oriented towards a hardware implementation (as shown by the embedding of parity bits the key itself). A generic improvement of DES is Triple DES or 3DES which offers twice[38] the key length at 112 bits[39].

AES (*Advanced Encryption Standard*) succeeded to the previous standard and was defined officially in 2000. It is an encryption algorithm using 128 bits data blocks and offering 3 possible key lengths of 128, 192 or 256 bits. AES was chosen after a public call of proposals and selection process, similar to the one leading to DES. Many more proposals were submitted to the selection committee which operated in a transparent process, but cryptanalysis efforts against the AES candidates may have been more fragmented than for DES initially. But these efforts were subsequently focused on

37 During the two decades following publication of DES, this word would have been written between quotes to underline the suspicion raised by NSA modifications and the fear that they had introduced a back door in the algorithm. Today, it seems that the NSA modification indeed improved the resistance of the original IBM proposal towards a general attack technique (differential cryptanalysis) then unknown from cryptographers working publicly. The suspicion towards NSA enhancements is then probably unfounded. All doubts on the absolutely exceptional capabilities of NSA in the field of cryptology at that time are also cleared. This note is also a nice practical exercise in viewpoint time management.

38 3DES involves 3 iterations of DES, hence its name. It is therefore approximatively three times slower than a DES, but one can reuse the hardware implementation to mask the impact. However, the key length is double as the middle iteration keying is a permutation of the first half of the key. Even today, 2^112 is still pretty reasonable ; though nostalgia plays a role too probably.

39 As well as an incredibly efficient screening case for evicting merchants of encryption devices powered by snake oil.

the selected algorithm and, after more than 15 decades of heavy cryptanalysis, AES security is still uncontested with respect to its initial specification[40]. Of note is the fact that AES is the first and only publicly accessible cipher approved by the NSA for top secret information[41] protection when used with an officially approved module. AES is available in many different encryption packages (including hardware implementations on common CPU). Note the name given to the candidate algorithm finally selected to become AES was *Rijndael*[42], a contraction of the name of its two inventors and one can still frequently see AES nicknamed with it.

The main advantage of symmetric ciphers is due to their encryption speed. 1 Gbit/s in hardware and 100 Mbit/s in software are pretty realistic figures with modern implementations[43] and symmetric ciphers speed can generally reach top networking speeds. Another advantage is the relative short key length they require for a given level of security. A 80 bits key length is still pretty acceptable to resist brute force attacks today for some time. Most algorithms allow for a key length between 128 and 256 bits now, hence with a considerable margin. Therefore, the key of symmetric ciphers is rather short and easy to store or manipulate[44].

The primary drawback of symmetric cipher is due to their symmetrical nature and the necessity to share the key between who encrypts messages and who deciphers them in a communication. Sender and receiver must trust each other for securing the cleartext appropriately and also trust each other to protect the key correctly. This mutual trust constraint is further enhanced by possibly out-of-band secure key distribution or renewal issues.

2.3.2.1 Special cases

Experience let us think of two specific points worth an additional paragraph or two.

First, there is apparently a need to clarify again and again the security level of a specific type of symmetric cipher: the exclusive-or with a constant key value K. This scrambling operation does *not* offer any security and will *not* resist more than a few minutes to a serious cryptanalyst. Unfortunately, those who propose this kind of cipher are not usually able to perform that cryptanalysis[45] ([Will1], V.i.). And it remains surprisingly difficult to convince all those who understand the operation of this nice wonderful scrambling that it has nothing to do with serious cryptography. The simplest technique may be to point out that it is a modern implementation of a classical polyalphabetical substitution, the "Vigenere cipher", which first usage is attributed to the french diplomat *Blaise de Vigenere* (1523-1596). Reliance on 16th century technology sometimes raises the appropriate concern.

To the credit of those insisting in such belief, one has to confess that the same binary operation can be useful in another context. As a matter of fact, if you are looking for a very good and proven encryption algorithm for the highest level of security, you have to know that the perfect and unbreakable cipher has been known for a long time. And it is implemented using an exclusive-or. However, the key must be a perfectly random bitstream as long as the cleartext and must never be reused (in practice an infinite key length condition). According to Shanon information theory, this is a perfect cipher. As the key must be as long as the entire set of all the data transmitted eventually, as it must be truly random and of course distributed without errors both to the sender and the receiver prior to actual transmission ; this is not really a convenient cipher. But it can be worth knowing in specific applications[46]. It has been proven unbreakable. It is usually referred as the *one-time pad* (OTP).

2.3.3 Public key cryptography

With public key encryption algorithms, the encryption key and the decryption key are different and do not play the same role anymore, Kc ≠ Kd, and :

- Kd must be kept secret, as only Kd owner can decrypt a ciphertext ;

40 At least, up to our knowledge.

41 When using 192 or 256 bits keys.

42 A prononcer « *rhine-delle* » ce qui est révélateur de l'origine wallone du chiffre. Certains mauvais esprits frontaliers du lieu de naissance de l'AES en déduiront donc qu'il s'agit d'un chiffre belge et qu'il n'y a aucune raison de traduire cette note...

43 And this was written ten years ago… But performance improvements have slowed down since the surge of software memory boulimia.

44 For example, it's realistic to store it on paper. *Hand written* paper if you follow me.

45 If they are able to do it, usually they are going to propose you next a variant of a classical escrow scheme or some handmade special algorithm, unless you ran away very far first.

46 Such as protecting the confidentiality of the discussions between two deep pockets paranoid high level executives, or between two nations leaders, or between one nation leader and some of the navy strategic force commanders, in case of heavy disagreement between the formers.

- however Kc is public, which means that everyone can encrypt a message.

The most well known public key encryption algorithm is RSA. This algorithm relies on the difficulty of find the factors of big numbers with small numbers of prime factors (typically two). In this case, the public key corresponds to this big number *N*, product of two (big) prime factors *p* and *q* which themselves are the components of the private key. The encryption algorithm transformation operate on the message using *N* to build the ciphertext. The inverse operation necessitates the knowledge of *N* secret prime factors to perform the decryption in a reasonable time. In practice, keys are built by choosing the prime factors first[47], hence the private key, before computing their product to build the corresponding public key. If one does not know the private key, one assumes that decryption of a message built with RSA is equivalent to *N* factorization problem ; which is infeasible in the general case for a computer as soon as *N* is big.

The primary advantage of public key ciphers is, of course, that no trust is needed between the sender and receiver of a message as they do not share any key. The management of encryption key is facilitated by their public nature : they can be sent directly by peers or gathered in key directories. This easiness is also sometimes misleading as security vulnerabilities of public directories are probably more subtle than those of private or symmetric keys. On the contrary, the private key must never be sent as its disclosure usually cancels the security of all the system, possibly including past messages. Public key algorithms apparition was a revolution as they opened new domains of application : as a mean for distributing symmetric keys, as a support for electronic signatures, for electronic certificates, etc. The main difficulty linked to public keys directories or public key distribution in general is linked to the need to protect the integrity of these keys, or more precisely the integrity of the link between one key and the identity of its holder. Cryptographic principles protect the link between the public and the private key, but it is the job of the operational protocol to ensure that the user identity associated to one pair of keys is not altered. (The risk being that an attacker replaces a public key belonging to its target with another one which private key part he holds before he further eavesdrops on future messages sent to the original peer by other users of the public key directories.)

Therefore the operation of public key ciphers is frequently coupled with integrity mechanisms surrounding the public key. There are currently two main approaches for protecting them.

First, the private key can be included in a certificate including the desired administrative information and the signature of a trusted third party. A user can the verify himself the integrity of a certificate he got from the peer with which he wants to communicate, provided he already held the third party public key. The latter is usually embedded in another certificate. This gives birth to certificate authority hierarchies, with self-proclaimed[48] certification authorities at the root. Afterwards, all the integrity guarantees associated to some user level certificate rely on the actual validation actions of these authorities (e.g. in person signature generation or alternatively remote identity card checking). This is the approach underlying the X.509 standard[49].

A public key can also be simply signed by a set of other actors, without distinguishing any specific actor *ex ante*. Our objective being to guarantee that a specific property string (like the name, an email, a pseudo, etc.) corresponds to a specific public key, it is possible to obtain this guarantee progressively by a chain of interlocutors signature until one who was able to perform direct verification of the claimed property (in person[50] for example). This is the approach adopted in PGP and OpenPGP afterwards.

The precautions for public key management and day to day usage of public key encryption seems to be counter-intuitive. It is not uncommon to be unable to recover the original version of a message one has just sent encrypted for example, unless you ask the receiver to send it back to you after decryption (plus re-encryption with your own public key unless the information itself is revealed on the network). It is also pretty difficult for a layman to understand the impact, the objective and the *modus operandi* of key signing[51]. Usually it is difficult to reach a correct and informed operation of the whole organization using cryptographic tools. This situation leaves the organization pretty vulnerable

47 Somehow randomly by the way, and this note is absolutely not a joke.

48 They signed themselves their own public key. Everyone can do that. Most commercial or administrative bodies would like everyone to think that only them should do that but necessity knows no law.

49 The most common certificate standard, which initially targeted phone directories, with major telecommunication operators as natural self-proclaimed authorities, user level certificates containing name and (wired) phone number and some convenient intermediate delegation opportunities to partner telephone companies. And yes, it is a decades old standard for reliable phone numbers publication that is supposed to protect Internet commerce and probably also most major industry software. Cross check with figure 8 for further insight into overall computer security status in this age.

50 As an improvement to the arcane but short-lived public key secure hash hexadecimal representation verification ceremonial of the 20th century between isolated computer geek pairs, the 21st century proposes so-called “crypto-parties” with even more geeks and also the public and even sometimes communication between the two groups.

51 Though the very existence and expansion of those crypto-parties contradicts this statement nowadays.

to social engineering attacks or mere user errors. Finally, some aspects of public keys management are not really implemented in the existing protocols or tools: things like revocation, renewal or controlled generation for example.

Outside of these operational problems, asymmetric ciphers also exhibit other more generic drawbacks.

First, these algorithms are relatively slow. You can reach speeds of a Mbits/s ; which means in practice one or two orders of magnitude slower than symmetric algorithms. In practice, these algorithms are frequently applied in combination with a symmetric algorithm to improve the overall performance. (For example, it is possible to use the public key algorithm to protect a random key sent together with the message itself encrypted using a symmetric algorithm using that random[52] key.)

Second, the length of the keys typically provided by these algorithms is pretty important, especially in comparison with those used with symmetrical algorithms. Public keys and private keys of 1024 bits up to 4096 bits are common with classical algorithms. Given that the private key must be very well protected, such a size can be a problem[53].

Frequently, public key lifetimes are chosen to span several years. As we noted the difficulty to protect public keys directories integrity and to perform key revocation efficiently, we consider such time frame to be a drawback, though it may prove necessary for usefulness in the context of signature or certificate publication.

Finally, given current common algorithms, it is not really possible to share a private key between several users. Additional protocols are needed to cover actual users needs which frequently involve signature or access rights delegation, information sharing and interim or multiple signing rights.

Of course, asymmetric algorithms are among the most interesting discoveries of modern cryptography, especially for civilian usage ; but their useful application necessitates insertion in a full system using other components, symmetric algorithms in particular[54].

2.3.4 Cryptographic hash functions

A cryptographic hash function is (probably) a collision-free one way function given current terminological knowledge of the author[55]. It is anyway a function H which allows to generate, from message M, a *fingerprint* or *hash* H(M) such as :

- the fingerprint H(M) has a fixed width *n* (e.g. 128 bits) whatever the size of M ;
- the probability that 2 different messages M and M' have the same fingerprint H(M) =H(M') is ~$1/2^n$;
- knowing M, it is easy to compute H(M) ;
- knowing M, it is impossible[56] to find M'≠M such that H(M')=H(M).

Typical examples of hash functions are MD5, SHA-1, SHA-256 or DES in CBC mode[57]. Typical examples of cryptographic hash functions are SHA3 or AES using a Miyaguchi-Preneel construction[58].

The first usage of such functions was associated to data integrity, when sending a file on the network or with respect to eventual alterations of a filesystem. In either case, even if an attacker is able to change the data, it could be possible to detect the modification using an offline fingerprints database computed beforehand. The most well known tool in this area is named *tripwire* and gave its name to the class of tools[59] as well as a commercial company[60].

Another common use case is electronic signature in practice by applying an asymmetric encryption algorithm to a fingerprint of the signed file instead of the full lengthier file directly.

---

52 Truly random. As in not pseudo-random generated. Really.

53 This size problem impact has changed with technology evolution. For example, the memory capacity increase of smartcard memory probably means asymmetric algorithms key size does not matter as much today in their case ; or that it matters in another context, like for the *Internet of Things*.

54 Or hidden components, like the (slow but exact) transition hidden at this precise place.

55 A few minutes ago, the section was titled “secure hash functions”, which is probably still not so bad.

56 In the computational sense of course, that is to say, there are no known polynomial time algorithm that can find a result really faster than simply trying randomly.

57 Note how we omitted a word to trick the inattentive student. Those are interesting, but not necessarily recommended nowadays.

58 The whole idea will lead the interested student to [Handbook1996] (chapter 9, figure 9.3) or to the Whirlpool hash function and its noteworthy birth place, the NESSIE European project.

59 Among which Samhain is a popular one in the GNU toolkit.

60 www.tripwire.com

2.3.4.1 Cryptanalysis : evil activity or fruitful effort?

The hash functions domain allows us to look in more detail at the interest and potential impact of the arcane side of cryptology : cryptanalysis. Contrary to popular belief, cryptanalysis is not an activity especially associated to blameworthy organizations or specialized army units. It is an integral part of the day to day activity of cryptology research. The design of an adequate encryption algorithm necessarily involves trying to break it via various means to evaluate its resistance and at the next step sharing the work with other cryptologists in order to try to break or improve each other work.

However, this mathematical activity is very obscure to outsiders, even for engineers familiar with ciphers implementation. The impact of some advances in cryptanalysis can be underestimated or, conversely, successful attacks on simplified variants of an algorithm without consequences on the full version feeds useless paranoia. This is just the normal back and forth cycle of this domain and it simply necessitates gathering knowledge without misconceptions.

The hash functions domain during the first decade of the 21st century is a pretty good illustration of this state of fact. At the end of the nineties, with the first widespread deployment of some cryptographic tools accompanying the deployment of Internet, most implementations reused the pre-existing hash functions readily available : MD5 and SHA-1. Both soon were present everywhere without anyone questioning them specifically, though a few old masters noted from their offices that they had been created pretty fast a few years ago to justify a visiting research grant between universities. In the boom of the Internet, self-proclaimed security engineers with 2 months experience in cryptography implementation but soon-to-become MBA accredited businessmen took appropriate action to ignore entirely these (soft) academic warnings and wire these free (as in free beer) algorithms MD5 and SHA-1 in every networking protocol they could find.

Starting 2004, theoretical advances in cryptanalysis, coming from the far-east, raised doubts on the collision resistance of MD5. The next year, cryptographers improved the attack, retracted their trust in MD5 (with demonstration of actual collisions for meaningful documents) and started to raise doubts on SHA-1. Previously mentioned engineers and businessmen alike started to register to scientific conferences on cryptography for a few years in order to get free advice on the attitude to adopt but probably failed to get the spiritual illumination they were looking for. The number of attendees came back to normal after a few years. The computer industry really does not want to learn these dangerous things which can kill a business with a bunch of algorithmic improvements on a few mathematical functions[61].

Fortunately, the attacks on MD5 and SHA-1 were probably not successful enough to compromise the implementations based on them a few years before. However, they were potent enough to require a stop in their usage and the search for an alternative. This alternative did not really exist at that time so a competition was started by the usual standardisation organization in this area (NIST). The interim could be assured by a variant of the less problematic algorithm with a much longer fingerprint size: SHA256.

NIST started the competition in 2007/2008, in order to select an algorithm that would become SHA3 and the next standard in the domain ; but wisely did it at a normal and calm pace so that the whole competition provides many possible alternatives and a better final standard choice.

Five finalists were selected among a dozen of initial candidate algorithms, among which some of them were coming from research projects anterior to the whole affair[62]. Among these finalists, it is the one initially named *Keccak* that was finally selected after international review as the new (American) standard cryptographic hash function.

2.3.4.2 SHA-3 & co.

SHA-3 (ex-*Keccak*) is a pretty fast hash function. It specifically allows for even faster hardware implementations (the main motivation behind its selection apparently among the other close finalists).

Keccak now SHA-3 has been studied for several years, only, but its adoption has been extremely fast (like for AES).

Therefore, nowadays, many people probably put all their faith in the RSA+AES+SHA-3 cryptographic triplet. In case you did not learn anything about irrevocable algorithm selection and one size fits all security devices, please start again at 2.3.4.1.

61 Though bitcoin iterated hash systems and multiple magnitudes money multiplication properties raised renewed interest some years later, but still little appropriation of actual knowledge about failure potential.

62 Like the Whirlpool proposal, which further demonstrates that for researchers MD5 and SHA1 never were the only choices in 2000 ; neither any other combination of algorithms in the writer current time frame.

2.3.5 Signing

Electronic signature, or authentication, is a security function which makes heavy usage of cryptographic algorithms. Without going further into this topic, we present two methods for generating a message signature and their respective use cases.

We denote Ks the signing key and Kv the verification key.

First, one can consider symmetric signature using a symmetric encryption algorithm and, in this case, Ks = Kv. For example, the last block of a DES encrypted cryptogram using CBC mode is a signature. Both the signing and verifying party must trust each other as the second, knowing the key, can generate a valid signature too for any input. This type of electronic signature is thus useless in front a judge (a third party) in case of later disagreement between signing and verifying parties ; though it is useful to prevent foreign alteration.

Asymmetric electronic signature schemes correspond to Ks ≠ Kv. In this case, a signing protocol may consist in taking the fingerprint of a message using a cryptographic hash function, then signing this message using a public key. Thus, we have Ks = Kd and Kv = Kc[63]. In this case, the signature can be verified by third parties (if they hold the public key).

This type of signature mechanism can be used to protect public key directories : each directory entry is signed by a certification authority. Certification authorities keys can be further organized in a directory hierarchy. This is the approach usually found in public key infrastructure systems (PKI) like X.509.

One last important point for electronic signature is linked to use cases. It is pretty important for the signing party to check entirely the signed document[64]. But this is not so straightforward with electronic documents. Tricks available for a malicious third party when confronted with (complex format based) electronic documents are more numerous and probably more efficient than the usual “fine small print” sometimes found in paper contracts. In practice, one still needs to be pretty prudent with electronic signature schemes when they are used with complex file formats or exotic software[65].

2.3.6 Other topics

We glanced at the most common topics of cryptology. The reader should not think that this field is limited to encryption algorithms or hash functions. Other algorithms are studied in this field, for example :

- steganographic algorithms which aim at hiding information into other data ;
- watermarking, which aims at incorporating non-removable (and possibly invisible) marks in data ;
- secure random number generation, with good properties against attackers predictions ;
- prime number generation ;
- escrow systems ;
- voting systems ;
- secure timestamping ;
- secure destruction (or erasure) of data ;
- and secure communication protocols, which is a whole field *per se* with key exchange or initialization protocols, mutual agreement, secure consensus, zero-knowledge proofs, etc.

2.4 Introduction to mandatory security policies

In the ITSEC, the “*system security policy specifies the set of laws, rules and practices that regulate how sensitive information and other resources are managed, protected and distributed within a specific system*.” ([ITSEC91], 2.9)

We consider that a security policy must define :

- security objectives, that is to say confidentiality, integrity or availability properties expected from the computer system ;

63 Note the signing step corresponds to a decryption operation.

64 Because you never know what can be hidden in the small characters at the bottom of the page... Relax. We are just speculating.

65 Faced with the typical word processor document with modification marks for example, how would it be possible to guarantee that the signing party actually checked and signed all modifications for example ? Contrarily, simple text signing inside a mail client software sounds easier to achieve.

- and security rules that allow to change the system security state, and which are imposed on the system in order to reach these properties.

A security policy is sane (consistent) if, starting from a secure state where such properties are satisfied, it is not possible, without violating some security rule, to reach an insecure state where such properties would not be satisfied. Security objectives and security rules are related to the security needs identified in the system. Security objectives describe the expected properties and define what a secure state means inside the system. The specification of these objectives usually necessitates notions like permission, interdiction or obligation and how they apply to the system. Security rules describe more precisely how basic security mechanisms are used inside the system. If specific security attributes are introduced, these rules define how they are to be manipulated. The set of security rules is a specification of how it is possible to manipulate the security state inside the system. Some rules may be introduced for the specific purpose of whole policy validation.

A security policy can be developed in three main dimensions : *physical*, *administrative* and *logical*.

A physical security policy defines everything related to the physical situation of the protected system. More specifically, it defines its critical elements and the protection measures targetting prevention of theft, aggressions, hazards like fire, etc. Given its target, a physical security policy primarily describe system elements from a physical point of view and define protection objectives. If such objectives are not reached, a physical intervention is usually necessary (like armour reinforcement, adding a locking system, etc.).

An administrative security policy is a set of procedures that define everything security-related inside an organization. Functions distribution in the organigram, task management and functions sharing are part of that, along with a precise definition of the associated powers. Some of the security objectives that may be found in these policies aim at preventing abusive delegations or guaranteeing a certain level of separation of power for certain activities.

The logical security policy deals more specifically with the information system. It describes logical access control and define general security access rules. The logical security policy is further refined in various instances associated to differing steps in the information system. A user accessing the system controlled under the policy first must identify himself or herself and then prove that he or she is actually the user he or she claims. These two steps are associated to the *identification* and *authentication* policy. Once both steps are completed, the *authorization* policy defines the operations one user is allowed to do inside the computer system.

2.4.1 Security models

Most formal works targeting computer security modelling have been associated to authorization policies. In order to define security objectives, these authorization policies introduce dedicated modelling elements. Most of the time, they start with a high level division of the system between its active entities, called the subjects, and its passive elements, called the objects. Additional security attributes may also be introduced in the model (security levels for example in multilevel policies). Sometimes, specific modelling methods are introduced to represent the system (like in control-flow policies for example).

Most security models found in the literature are associated to specific security policies : for example, a lattice is usually associated to multilevel policies attributes.

Using a security model guarantees to the user that the way security is represented in the system description is not ambiguous and possibly can be proved conforming to the security objectives defined in the overall security policy. The model choice is motivated by operational reasons: the need to reflect as simply as possible the mechanisms available in the system. Finally the expected security properties should be verified, at least unambiguously represented, in the chosen model.

We think that, furthermore, it could be interesting to be able to represent in the security model what can occur when a violation of the security objectives is observed. Usually, classical models do not take into account this approach and clearly favour the expected security properties verification.

2.4.2 Mandatory and discretionay access control policies

Authorization policies, are classified in two main categories : *discretionary policies* and *mandatory policies*. Such a distinction is pretty influential in practice. In both cases, we partition the system entities into two categories : active entities called subjects (users, processes, etc.) which manipulate[66] information and passive entities or objects (documents, files, etc.) which hold information.

66 Observe or alter.

In a discretionary policy, each object is associated to a specific subject, its *owner*, which can manipulate access rights at his or her discretion. The owner of some information can thus freely define and transfer access rights to himself or another user. The Unix filesystem access rights is a classical example of such a discretionary access control policy. If we suppose that user u_1 owner of file f_1 trusts user u_2 but not user u_3 ; u_1 gives a read access to u_2 over file f_1 but not to u_3. However, in this case, u_2 can make a copy of the data embedded in file f_1 into another file f_2 which he owns directly. Then, he can freely give u_3 a read access right over this copy. This is an information flow that contradicts the initial security objective formulated by u_1, but it is impossible to control it within the framework of a discretionary access control policy. Similarly, a discretionary access control policy cannot prevent situations associated to Trojan horse software. A Trojan horse program (or Trojan) is a program that, while performing an innocuous or legitimate function, also performs on behalf of the user executing it another covert function contrary to the security policy of the system. For example, a program that mimics the normal operation of a login system can fool a user into communicating his or her actual login password to a third party while trying to perform a normal session initialisation[67].

In order to solve such problems, mandatory policies impose, in addition to discretionary rules, new mandatory security rules that aim at ensuring such general security properties. For example, new security attributes (informally associated to security levels) may be associated to data containers and propagated with each manipulation or creation of information. Only those users specifically associated to a given security level would then be allowed to manipulate or access the information in these containers. Such mandatory rules enforce global system properties (for confidentiality or integrity). They may come as an addition to conventional discretionary security rules (which offer a convenient method for manipulating access rights inside one level). Therefore, a user will only be allowed to perform an action if both mandatory rules and discretionary access rights allow it.

Classical examples of mandatory policies are the DoD multilevel confidentiality policy formalized by Bell - La Padula [BLP75], the Biba integrity policy which follows the same guidelines for integrity assurance or the Clark&Wilson [Clark&Wilson87] policy which targets some commercial systems. Some other examples will also be found in the forthcoming sections.

2.4.3 Discretionary access control policy modelling

In this section, we present the most common models found in the literature and associated to discretionary policies. These model are general enough to represent mandatory policies, but those are usually associated with other specific models more suitable for reasoning about them and their mandatory rules, presented later in this document.

2.4.3.1 Models based on the access control matrix

The notion of an access control matrix was first introduced by Lampson in 1971 [Lampson71]. In his model, the access control matrix (or more precisely, the array) is dedicated to the representation of access rights. These models are structured around a state machine where each state is a triplet (S, O, M), with S a set of subjects, O a set of objects (with $S \subset O$) and M an access control matrix. Matrix M has a line for every subject s, a column for every object o and $M(s, o)$ is the set of access rights that subject s holds on object o. The access rights are taken inside a fixed finite set A, defined in the security policy, and corresponds to all the operation a subject may perform over an object. The access control matrix is not fixed, it evolves with system transitions, with the creation of new subjects, new objects or the operations performed by users. All the actions that modify M change the system security state.

$$S=\{s_1,\dots,s_n\}\quad O=\{o_1,\dots,o_m\}\quad A=\{a_1,\dots,a_p\}$$
$$n\le m\quad S\subset O$$
$$M_{i\in[\![1\dots n]\!],\, j\in[\![1\dots m]\!]}=M(s_i,o_j)=\{\alpha_1,\dots,\alpha_r\}\,/\,\alpha_{k\in[\![1\dots r]\!]}\in A$$
$$s_i \text{ is authorized to } \alpha_{k\in[\![1\dots r]\!]} \text{ over } o_j$$

Most systems based on this modelling add rows or columns to the access control matrix each time a new process is created to act on behalf of one user or each time a new file is created[68]. Those new lines or columns are initialized with default values as specified in the user configuration files. Later on, a user can change the access rights associated to files he created (especially in a discretionary access control policy) but he does not directly operate on M. As a matter of fact, these access rights modification operations must be legitimate and they may also be submitted to additional

67 The fake login software then probably bails out as if the user had made an error to perfect the illusion.

68 Si it is not really a matrix with fixed dimensions in the strict mathematical sense ; it is nearer from a 2-dimensions dynamic array like those found in programming languages.

control rules (like those impose by a mandatory access control policy). Hence, the user performs these operations using system utilities that only make them if they are conformant to the system authorization scheme.

The access control matrix model was a basis for a lot of subsequent work.

a - The HRU model

Harrizon, Ruzzo et Ullman used Lampson access control matrix model in order to study the feasibility of the problem of the verification of security properties represented using this model. To conduct their, they considered a specific security model, the HRU model [HRU76], similar to Lampson but where only a subset of the matrix *M* modification commands are considered, of the following form, where $a^{(i)} \in A$:

command $\alpha(x_1, x_2, \ldots, x_k)$

if $a' \in M(s', o') \wedge a'' \in M(s'', o'') \wedge \ldots \wedge a^{(m)} \in M(s^{(m)}, o^{(m)})$

then $op_1; op_2; \ldots; op_n$

end

Table 1: HRU command format

x_i is a parameter of command α and each op_i is an elementary operation among the following ones (where semantic conforms to denomination) :

enter *a* into *M*(*s*,*o*)	delete *a* from *M*(*s*,*o*)
create subject *s*	delete subject *s*
create object *o*	delete object *o*

Table 2: HRU elementary operations

Given an initial configuration Q_0, an access right *a*, we say Q_0 is secure with respect to *a* if there is no sequence of commands that, executed starting from state Q_0, can bring access right *a* into a cell of the matrix where it is not already. Demonstration of this property established the *protection problem*. Harrizon, Ruzzo and Ullman first demonstrated two founding theorems with respect to the protection problem complexity :

- the protection problem is undecidable in the general case ;
- the protection problem is decidable for mono-operation systems, systems where all commands contain only one elementary operation.

With additional constraints on the command allowed in the system, several other decidability demonstrations have been proposed. However, since their seminal presentation in the context of HRU, these two first properties have clearly identified some basic problems with computer security. On one hand, a model like HRU without restrictions can represent a wide array of security policies, but then there is no general mean to verify such policies properties. On the other hand, even if he may be possible to manipulate it for demonstration, the mono-operation HRU model is too simple to represent practical usable security policies. For example, in a mono-operation system, one cannot represent security policies where subjects that create objects are given specific access rights, as there is no elementary operation that can simultaneously create an object and associate access rights to it. Furthermore, decidable does not mean verifiable (especially within reasonable time).

b - The Take-Grant model

Various variations inspired by HRU were proposed later on in order to address the representation of a security model expressive enough to represent complex authorization policies, but nevertheless easy to manipulate mathematically.

The Take-Grant model, introduced in 1976 is a first variant of HRU [Jones76], built by restricting available commands. Commands should be taken from four main categories :

- commands of *create* type which allow to create an object with an initial access right from a subject on this object ;
- commands of *remove* type which allow to retract an access right from one subject over an object ;
- commands of type *grant* which allow any subject holding an access right over an object as well as the special right *g* over another subject to grant that access right to that latter subject ;

- commands of type *take* which allow any subject holding a special access right *t* over a subject to take any access right this subject holds over objects.

These four categories lead to define four new commands for every basic access right defined in the authorization policy. The special access rights *t* and *g*, and the associated *take* and *grant* rules, are related to these additional rules imposed on the authorization scheme in order to control the system security state evolution (ie. the access control matrix modifications). These new rules also guarantee that the Take-Grant model offers a protection problem decision algorithm with linear complexity [TG77]. However, some of the assumptions underlying this model are also pretty unrealistic: most of the achievable properties are associated to a worst case hypothesis where all users collaborate to defeat the system security objectives. Several refinements of the properties achievable in the Take-Grant models have henceforth been proposed in order to retract this worst case hypothesis in favour of one where one user [Snyder81] or a subset of several users [Dacier93] only try to defeat the system security objectives.

The Take-Grant model also offers a convenient graph representation where subjects and objects are represented by graph nodes and access rights are represented by oriented links in the graph.

c - TAM

More similar to HRU, the SPM model (*Schematic Protection Model*) from [Sandhu88], which also incorporates access right types, offers a decidable subset more extended that Take-Grant. This model is also the basis of the TAM model (*Typed Access Matrix*) . TAM is defined by introducing strong typing inside the HRU model.

Like HRU, TAM is undecidable in the general case. However, if the number of parameters allowed in a command definition is limited to three, while preventing cyclic object creation, the resulting model is decidable in polynomial time, while still being expressive enough to represent a significant set of security policies.

2.4.3.2 Role based access controle models

A role based access control model does not directly associate privileges (in the sense of a set of access rights) to users in the system. Privileges are associated to intermediate abstract entities, called roles. Different users can be associated to various roles and the two relations (user, role) and (role, privilege) lead to the definition of the specific permissions granted to a specific user. Such roles can further be organized in a hierarchy of roles, which allow for progressive and structured refinement of the permissions granted to each role.

2.4.4 Multilevel policies

Multilevel authorization policies rely on partitions of the system subjects and objects. Each *level* is associated to a partition. These security levels are usually totally or partially ordered. Security objectives can be associated to confidentiality or integrity of objects, and they are expressed using these levels. The authorization model security rules which drive the mandatory access controls defined in the security policy also rely on these levels.

2.4.4.1 The DoD policy

The DoD mandatory access control policy, formalized by Bell and LaPadula [BLP75], is a multilevel authorization policy targeted at confidentiality properties. While defining the security policy, [BLP75] also introduces a lattice-based security model which offers a formal definition of the security objectives and the authorization scheme of this policy, and the opportunity to demonstrate soundness.

Models based on a lattice model rely on the association of different security levels to subjects and objects in the system. We denote *h(s)* the security level of subject *s* and *c(o)* the security level of object *o*. Each security level *n* represents military or governmental security designations given to people or documents. These levels $n=(cl, C)$ are built with two components: one is *classification* or *clearance cl*, taken in a totally ordered set (for example : UNCLASSIFIED, CONFIDENTIAL, SECRET and TOP-SECRET) and the other a *compartment C* defined as a *set* of *categories* (taken among, for example, « Nuclear », « NATO », « Crypto », etc.)

The classification *cl* given to an object or a piece of data represents the risk associated to the divulgation of the information it contains. In addition, this information is associated to a compartment *C* which identifies all the domains where such information is relevant. The clearance of a user also incorporates a classification corresponding to the trust he is given and a compartment incorporating the categories for which this trust is granted.

The security levels create a lattice partially ordered by the weak ordering relation $\preccurlyeq$:

$$n \preccurlyeq n' \text{ if and only if } cl \leq cl' \text{ and } C \subseteq C'$$

The security objectives properties expected from this policy are the following :

- prevent any information flow from an object of a specific classification level to another object with an inferior classification level ;
- and prevent any subject of a given security clearance to get information coming from an object whose classification level dominates his or her clearance.

The associated authorization schema directly emanates from these objectives. With respect to confidentialy, we partition the operations a subject can perform on an object between *read* and *write* operation, and we introduce the following two rules :

- a subject can read from an object only if the clearance level of this subject dominates the classification level of the object (« simple rule ») ;
- a subject can simultaneously access object o for reading and object o' for writing only if the classification level of o' dominates the classification level of o (« ★-rule »).

In the Bell-LaPadula model, the system is represented by a finite state machine, where states are defined by a matrix $M \subset (S \times O \rightarrow A)$ which associates each subject $s \in S$ and each object $o \in O$ to the access rights $a \in A$ that this subject holds on this object (with $A = \{\text{read}, \text{write}\}$). Each subject and each object is associated to security levels $h(s)$ et $c(o)$, respectively. Two properties, associated to the two security rules presented previously ensure that a given system state is secure :

- the simple property : $\forall s \in S, \forall o \in O,\ \text{read} \in M(s,o) \Rightarrow c(o) \leqslant h(s)$
- the ★-property : $\forall s \in S, \forall (o,o') \in O^2, \text{read} \in M(s,o) \wedge \text{write} \in M(s,o') \Rightarrow c(o) \leqslant c(o')$

This security policy raises several negative concerns :

- On the one hand, the security level of information degrades constantly due to overclassification. In practice, the authorization scheme rules impose that any information security level only increases, slowly bringing the system two a state where only few people are cleared high enough to access these informations.
- On the other hand, this model does not represent all the possible system information flows nor does it represent the covert channels that may exist in the system.

One can also note that the lattice structure can be used for modelling other security properties outside of Bell-LaPadula (most notably with respect to integrity protection).

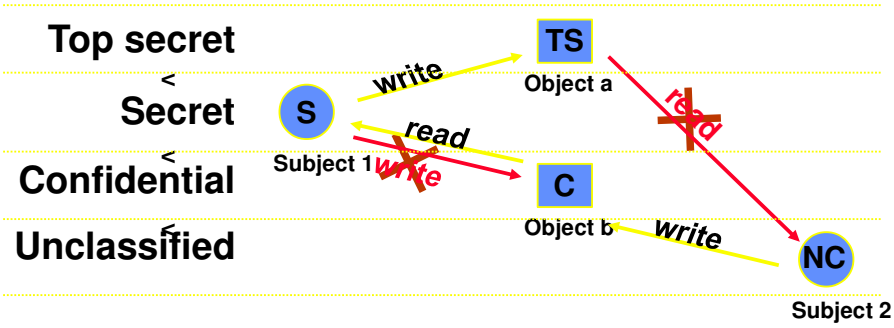


Figure 3: Bell-LaPadula security policy operation example

In the figure 3, a simplified example of Bell-LaPadula policy operation is given, using a classical set of classification and clearance levels without any compartment (i.e. with security levels fully ordered).

2.4.4.2 Biba integrity policy

The security policy introduced by Biba in 1977 is the policy dual from Bell-LaPadula where the objective is to ensure the integrity of the system objects [Biba75]. Each level associated with a subject or object is therefore an integrity level. The security objectives of this policy are therefore :

- to prevent any information flow between an object at a certain integrity level towards an object with a superior integrity level ;
- and to prevent any subject placed at a specific integrity level to change any information belonging to an object with a superior integrity level.

The authorization schema comes from these properties, considering integrity-oriented security labels and the fact that system operations can be grouped into three classes : modification or observation of an object by a subject and invocation of a subject by another subject.

- A subject can only modify an object if the integrity label of the subject dominates the integrity label of the object.
- A subject can only observe an object if the integrity label of the object dominates the integrity label of the subject.
- A subject *s* can invoke another subject *s'* only if the integrity label of *s'* dominates the integrity label of *s*.

A drawback of this policy, once again dual from those of Bell-Lapadula, lies in the constant integrity level degradation of any piece of information while the system processes it.

2.4.5 Information flow control policy

The Bell-LaPadula model represents only those flows of information that go through knowns objects (documents, files) and does not offer much mean for identification and control of covert channels. Such a model does not consider information flows that may reach a subject without going through specific system objects. Control flow models refers to a wider view of the system. They do not only consider read and write operations over objects, but also potential information flows between subjects. These models try to describe all the communication channels available in the system, either explicit or covert.

This original approach for the representation of information flows within a system relies on the identification of the causal dependencies that may exist between the various objects existing in the system at various points in time [Bieber92]. We consider that an object is observable by a user reading a specific system output if this object is causally related to this output. In this model, a system is represented by a set of points (o,t). A point represents the state of object o at time t. Some of these points are inputs, others are outputs of the system, while the remaining points represent the internal state of the system. This set of points may evolve in time and this evolution is the consequence of the elementary system transitions. Such a transition can, at time t, associate a new value v to object o at this point. This instant and this new value therefore depends on a subset of the preceding points.

Such a functional dependency of one point with respect to preceding points is called a causal dependency. The causal dependency of (o,t) with respect to (o',t') with $t'<t$ is denoted $(o',t') \rightarrow (o,t)$.

The transitive closure of relation $\rightarrow$ (denoted $\rightarrow^*$) at point (o,t) defines the causal cone at this point : $cone(o,t)=\{(o',t')/(o',t') \rightarrow^* (o,t)\}$.

Reciprocally, we define the dependency cone as the set of points which depend on (o,t) : $dep(o,t)=\{(o',t')/(o,t) \rightarrow^* (o',t')\}$.

Causal dependencies represent the system information flow organization. If subject *s* has knowledge over some of the internal behaviour of the system, he can learn about these causal dependencies. In this case, observing a specific output x_o, he may infer information belonging to $cone(x_o)$. Reciprocally, by interfering with one input x_i of the system, *s* may be able to alter all points belonging to $dep(x_i)$[69].

The expected security properties that may be described in this model can be related to confidentiality or integrity in the system. Specifically, if subject *s* can observe the set O_s of system output x_o, we denote Obs_s the set of all points that *s* can observe in the system :

$$Obs_s = \bigcup_{x_o \in O_s} cone(x_o)$$

Similarly, if subject *s* can alter the set A_s of system inputs x_i, we note Alt_s the set of system points that *s* can influence :

$$Alt_s = \bigcup_{x_i \in A_s} dep(x_i)$$

69 Obviously, both statements are worst-case hypothesis. However, in security engineering, one should only believe in worst-case hypothesis ; when hypothesis are used at all. Speculative thinking is not recommended, but optimism is simply forbidden. After all, non worst-case hypothesis are recipes for disaster ; if only from the commercial point of view because any competing company marketing department will come out with much more appealing ideas for compromises than your own engineering department. Marketing department imagination benefits rather than suffers from physical laws constraints (or so they say). But I am digressing. Especially in this case, you would have to expect that the inference capabilities of your opponent are more limited than your own if you want to further restrict $cone(x_o)$ or $dep(x_i)$. In other words, you expect your opponent to be dumber than you are. This is not wise at all, hence this long footnote in favour of the worst case hypothesis for your enlightenment.

If R_s is the set of points that subject s is allowed to observe according to the security policy of the system, we can say that the system is secure (with respect to confidentiality) if subject s can only observe those objects that he is allowed to observe, that is to say if : $Obs_s \subseteq R_s$. If W_s is the set of points that subject s is allowed to modify according to the security policy of the system, then we can say similarly that the system is secure (with respect to integrity) if subject s can only alter those objects that he is allowed to modify, that is to say if : $Alt_s \subseteq W_s$.

If some security levels are associated to subjects and objects, the relation $Obs_s \subseteq R_s$ relative to confidentiality can be obtained via enforcing two rules in the system, analogous to those defined in the Bell-LaPadula policy :

- a subject is allowed to observe an object only if the object classification is dominated by the subject clearance ;
- and, if an object o' has a causal dependency over object o, then the classification of o' must dominate the classification of o.

This model is particularly remarkable as it introduces a new approach for information flows formalization inside a system. The main interest of this formalization is its simplicity: causal dependencies allow to describe very briefly and strictly information flows. However, implementations of this model are rather rare and targetted at specific domains.

2.4.6 Interface security models

Rather than specifying specific mechanisms for enforcing security, interface models specify restrictions on a system's input/output relation that are sufficient for ruling out nonsecure implementations [McLean94]. This class of security models deals more naturally with the dynamic nature of systems, especially networks, and relies on pretty general – albeit rather abstract – modelling formalisms. The system model is made of all different subjects (or users) in the system as well as the set of all execution *traces* associated to these users. A trace is the history of all inputs made by this user, that is to say the ordered sequence of system states occurring after each user input (or transition or command). Some specific commands allow to isolate the system outputs for one user and one usually primarily focus on the system properties associated to those outputs.

The main advantage of these formal models is to allow for a better understanding of significant security properties formalization issues. Very interesting security properties can be modelled and compared using these generic system formalization models.

The main properties identified in the literature, associated to the main security objectives relevant to each of these security policies are the following :

- *non-interference*, defined as that a group of users, using a specific set of commands, cannot interfere with another group of users, if whatever is done by the first groupe with their commands has no effect whatsoever on what the second group can observe on outputs. Given the system model of its formal definition, this property only applies to deterministic systems.
- *non-deducibility*, corresponding to the fact that whatever the ouput observed by a low classification level user, this ouput is compatible with any acceptable input form a high classification level user.
- *generalized non-interference*, which complements non-deducibility to patch a problem associated to the fact that non-deducibility does not guarantee that a low level user is prohibited to access high level information, provided that they are mixed with random data[70].
- *restriction*, or *non-inference*, which further restricts generalized non interference to allow for composability of the property with respect to several non-deterministic subsystems.
- *non-influence*, with further complements non-interference protecting the visibility of events by *non-leakage* protecting the secrecy of a system state.

We present a few of these in more detail in 2.4.6.2.

Interface models rely on a system representation which is a finite state automata with observable outputs. Such a system is built by :

- a set S of subjects or users ;
- a set Σ of system states ;
- a set Γ of commands or operations that can be executed in the system ;
- a set Out which elements are the user visible outputs ;

70 In a non-deterministic system, non-deducibility does not make any difference between random noise added to information and an actual cryptogram, intelligible with the corresponding key.

as well as :

- a function $out : \Sigma \times S \rightarrow \text{Out}$ which represents what a given user can observe when the machine is in a specific state, called the *output function* ;
- a function $do : \Sigma \times S \times \Gamma \rightarrow \Sigma$ which represents how commands alter states, called the *transition function* ;
- and a constant $\sigma_0 \in \Sigma$, which is the initial machine state.

If *w* is an input sequence or *trace* in this system, that is to say a sequence of commands Γ started by users $w \in traces$ with $traces = (S \times \Gamma)^*$, we denote [*w*] the state reached by the state machine after execution by all users of all commands listed in *w*, starting from initial state σ_0. We denote $\langle\,\rangle$ the empty trace (no command), and, *in extenso*, $\nu \cdot \gamma_1(u_1) \cdot \gamma_2(u_2) \cdot \ldots \cdot \gamma_n(u_n)$ the trace *w* built by prefix trace ν (possibly empty) followed by the sequence of commands $(\gamma_i)_{1 \le i \le n}$ of Γ performed by users $(u_i)_{1 \le i \le n}$.

This state machine can be extended in order to account for multilevel security properties, such as those embodied in the Bell-LaPadula model, therefore building up a system with security labels. In this case, it is sufficient to consider a state space made of an access control matrix and access rights modification commands as those defined in 2.4.4.1. It is also convenient to isolate the commands of Γ that allow to perform inputs or outputs towards a user and to consider traces built with a sequence of inputs (commands) followed by a final output operation. The set of output operations is denoted Γ_{out}. As a matter of fact, it is on these specific output operations that a given security policy will focus. Each time such a classical distinction between commands will be done in this text, the command names, such as read(*u*), highin(*u*), lowout(*u*), lowin(*u*), will indicate their category without ambiguity.

2.4.6.1 Deterministic systems: Non-interference

Let *h* be a function providing the clearance level of users, so that *h*(*u*) is the security level (clearance) of *u* (cf 2.4.4.1). Let *purge* be a function from $S \times traces^*$ in *S* so that :

$$purge(u, \langle\,\rangle) = \langle\,\rangle$$

$$purge(u, hist \cdot command(u')) = \begin{cases} purge(u, hist) \cdot command(u') & \text{if } h(u) \geqslant h(u') \\ purge(u, hist) & \text{if } h(u) < h(u') \end{cases}$$

A system satisfies the *non-interference* property if and only if :

$$\forall u \in S, \forall w \in traces^*, \forall c \in \Gamma_{out} \quad out(u, w \cdot c(u)) = out(u, purge(u, w) \cdot c(u))$$

This is the original presentation from the initial article by Goguen and Meseguer in 1982 [Goguen82].

An alternative formulation can be given using a partition of memory between high and low parts. If *M* is a memory configuration, with M_L and M_H the low and high level parts respectively. Let $=_L$ be the function that compares the low parts of memory, i.e. $M =_L M'$ iff $M_L = M'_L$. Let $(P, M) \rightarrow^* M'$ be the execution of program *P* starting with memory configuration *M* that ends with memory configuration *M'*. The non-interference is also defined for program *P* as :

$$\forall M_1, M_2 : \; M_1 =_L M_2 \; \wedge \; (P, M_1) \rightarrow^* M'_1 \; \wedge \; (P, M_2) \rightarrow^* M'_2 \; \Rightarrow \; M'_1 =_L M'_2$$

It is not always easy to compare precisely the Bell-LaPadula model and models based on non-interference. However, one can note that, in general, the Bell-LaPadula model offers weaker properties than non-interference in the sense that the latter prevents the occurrence of some covert channels that would be available with the standard implementation of primitive operations from the Bell-LaPadula model. On the other hand, non-interference allows the implementation of operations that would not be permetted by Bell-LaPadula, like the possibility for a low level (in a confidentiality policy) user to copy directly a high level file into another high level file (provided he does not access the file himself). In both cases, however, the non-interference property seems to correspond better to the intuitive notion of security (confidentiality) than the Bell-LaPadula model[71].

However, this initial model suffers from several limitations. On the one hand, non-interference is a very strong property and can be seen as too strong: for example, it leads to prohibits the usage of encrypted communication channels between high level users (even perfect ones in the Shannon sense) if low level users can have access to the cryptogram. On the other hand, the model only applies to deterministic systems. Despite these limitations and the implementation difficulties, non-interference still is probably at the state of the art in terms of security guarantees for deterministic system model security definition. Its extension towards non-deterministic systems, composability, or most recently internal state protection, lead to multiple later work.

71 You wouldn't have thought that all these maths definitions could be more intuitive than military ink stamps, don't you ?

2.4.6.2 Non-deterministic systems : Non-deducibility, Generalized non-interference, Restriction

In order to give a non-deterministic version of non-interference, one has first to present how it is possible to describe a non-deterministic system. With the former modelling, one can consider that an execution trace is an acceptable (possible) behaviour of the system. In this case, a non-deterministic system is described by a set of acceptable system behaviours. In order to define later properties, one also has to make a distinction between two interaction levels with the system (in the sense of Bell-LaPadula) that corresponds to a high and low confidentiality level.

Non-deducibility, proposed by Sutherland in 1986 [Sutherland86], corresponds to the fact that, for any pair of acceptable traces *T* et *T'*, there must exist an acceptable trace *T''* that gathers : the low level commands of *T* (in their original order), the high level input commands of *T'* (in their original order), and all other commands. This property corresponds to the fact that everything a low clearance user observes is compatible with any input from a high level user.

Even if non-deducibility is a more general property than non-interference for a specific system ; given that it does not imply that the system be deterministic, it is not equivalent to non-interference for deterministic systems with more than two users. In this case, non-deducibility is weaker than non-interference.

This analysis and subsequent problems identified by McCullough in 1987 [McCullough87] led to the introduction of an alternative version, called generalized non-interference. A system exhibits the generalized non-interference property is and only if, given an acceptable trace *T* for the system and an altered trace *T'* built by inserting or removing a high level input from *T*, there exits an acceptable trace *T''* built by inserting or removing a high level output of *T'* just after the alteration of *T* leading to *T'*. (Any acceptable trace with a high level input is equivalent to another acceptable trace with some high level output instead, located at the same place in system history.)

Non-deducibility and generalized non-interference though still both suffer from a major drawback : none of these properties is preserved by composition of two systems. The property of restriction was introduced as a way to solve this problem.

A system exhibits the *restriction* property [McCullough90] if and only if, given an acceptable trace T in the system, and an altered trace T' built by inserting or removing a high level input from T, there exists an acceptable trace T'' built by inserting or removing a high level output from T' just after the alteration of T leading to T', and after each low level inputs which follow the alteration of T. (Any acceptable trace with a high level input is equivalent to another acceptable trace with some high level outputs instead, located at the same place in system history, or after any other later low level output sequence.[72])

72 So, chatting is an effective way of enforcing the security of your discourse, even if you repeat high level data, provided that you hide the right item…? Many natives of southern Europe seriously implement the idea.

3 Embedded systems and security

3.1 Specificities (or not)

As a first step towards finally focussing the course on its original title, we should be considering first defining the target of our security-oriented focus: embedded systems. Over the first years of exploring the field, characterizing embedded systems proved to be more difficult than initially thought. Fortunately, bringing up the topic of computer security in these contexts is less problematic.

3.1.1 Definition attempts

As a matter of fact, it seems those building, buying or using embedded systems show much difficulty defining them and alleged specificities with respect to general purpose computing devices – especially nowadays with such devices ranging from kids toys or game consoles to enterprise data servers. The typical simple question "*What really makes your devices so specific ?*" frequently generates some anxious silences (especially among COTS fanatics).

A characterisation attempt was nevertheless attempted, if only to fill the void raised by the question. The easiest differentiation is the clarification of three main cases where people speak of embedded systems generically while they are meaning that the system is either:

- a real-time system, that is to say a system with timing deadlines to respect – deadlines in the sense that the service offered by the system is useless if it is not delivered before the deadline ;
- a critical system, that is to say a system which failure may have catastrophic consequences (in fact a system with safety requirements according to [avizienis2004] terminology) ;
- a computer system embedded in a vehicle.

These three dimensions are relevant to a given system characterization, and they are orthogonal, in the sense that a critical system may also be a real-time system embedded in a vehicle. So we should have a clear understanding of these attributes when delimiting our systems. But alone they may not be sufficient to characterize the kind of system we usually have in mind[73] which may exhibit some more exotic features.

An important dimension of embedded systems seem to be linked with their energy autonomy. Sometimes they are designated as autonomous systems, not really to highlight their autonomy but more to underline their dependence on a limited reserve of energy, hence their limited autonomy. Hence, a requirement for embedded systems seem to be that they perform correctly (and possibly regularly) when distant from permanent power plugs. They are in need of electric batteries as a consequence and such limited energy reserve has impact on the overall device operation (if only to save energy to extend their operation time).

Another aspect of such systems is the relative lack of interaction with any (human) user. Compared with general purpose computer systems, it is usually thought that such systems operate with little physical user intervention. This may be due to distance as in the (extreme) case of deep space probes or the most common case of the inaccessible places of a vehicle (plane, train, car) while moving ; but it is not necessarily a physical distance. The system may also be hidden from the surrounding users and, even if it provides a service to them, they do not or can not interact physically with it outside of the provided service functions. This is for example the case with hidden CPU (like a smartphone baseband, its smartcard SIM), with RF or WiFi access points and more generally network infrastructure equipment (thought in these cases, human intervention is still possible, only infrequent and costly), RAID cards, home automation, security devices (when the owner is not the access rights holder), etc.

These aspects also show us that these systems are usually integrated systems, in the sense that they usually combine specific software with a more or less specialised hardware platform. Thus, these embedded systems are not only software and they may be interesting to tinker with from the computer hacker[74] point of view (probably including the author). From the security point of view, adding computer hardware insecurity to software insecurity is an interesting case too, pretty promising from the enterprising engineer perspective, though more of the nightmare variety at the moment from the IT security point of view.

73 Think of a general purpose computer used for encryption of interactive voice communication on a navy command ship for example.

74 Note the author is also old enough to make a clear distinction between hacker (good guy) and cracker (bad guy).

Finally, now most of these embedded systems are communicating devices, which form a distributed system. They are not point to point actuating devices anymore, they have more or less sophisticated networking interfaces, more and more regular network interfaces off the shelf. They do not communicate directly, physically, with a user, but they communicate with other computers and collectively operate their function.

More provocatively, some less usual characteristics of those embedded systems can be listed. Many of these embedded systems are sometimes lost in the organization or the enterprise, nobody remembers where they are exactly. They are the kind of system that their owners really do not want to be stolen, in the sense that their loss will incur more than the mere economical cost of their replacement : they hold sensitive or valuable data or service that the user does not want to be stolen[75]. Such devices may be usefully repurposed, which shows, afterwards, that they were not so specific and probably held much in common with their original off the shelf (unsecure) counterpart.

Finally, these embedded devices are frequently manufactured in numbers, sometimes huge numbers. And all these millions of devices share the same characteristics, which raise challenges, especially for the now dominant approach to security improvement in the industry which primarily relies on software updates.

However, all these characterizations do not really make any of those system fundamentally different from what they are in the first place – as opposed to the sensors, actuators or mechanical systems they control – they are *computer systems*. Therefore, for the moment and until we get a proper definition, we will consider that we are dealing with the security of *specialized computers*.

Whether embedded systems is then only used in the industry as a buzzword to indicate that the computer is not a regular general purpose computer (at least unless repurposed first)[76] is left as an open question for the reader.

3.1.2 Security aspects

A general current trend on those specialized computers is the evolution from independent isolated or very centralized systems to regular distributed systems with networking software stacks. This raises the usual distributed security problems with authentication and authorization, not to mention consensus. At the moment, these are usually addressed primitively using point to point protocols and iteration, and those approaches logically do not scale well or impose a centralized architecture again.

But apart from this trend, especially outlined in the *Internet of Objects* (IoT) phenomenon, these specialized computers also exhibit multiple security requirements that are not so common in the field of general purpose computers, especially when combined. Indeed, when dealing with embedded systems, we observe such security requirements as :

- Supplier protection (like in the cellular networking content where the networking operator want to protect its network infrastructure) or protection of the content owned by the supplier of the system (especially multimedia content).
- The embedded system environment protection, whether it is the vehicle itself and its passengers, or the vehicle resources (e.g. for a satellite, where the payload maybe more valuable than the satellite itself).
- The protection of the embedded system owner, who usually legitimately[77] thinks his requirement as first priority.
- The protection of the embedded system itself, even against direct physical threats of an attacker possibly disguising as a legitimate owner (or threatening him or her) – this is especially the case for security-specific embedded systems like smart cards or cryptographic chipsets (dedicated board).
- And finally, the protection of a whole embedded information system made up of several networked specialized computers.

Therefore, such requirements are not limited to file access rights management or network TCP connexion authorization like most IT managers would love to restrict them. They are not even centered on the end user of the device. Furthermore, we are currently seeing an evolution of the security requirements exposed by such systems from some security functionalities to add to the systems to the need of security management at the design and architectural level. This scope extension is further motivated by some of the challenges faced by these specialized computers with respect to security.

75 And that, consequently, an attacker may probably find interesting to steal...

76 And that some engineers want to set up their own niche on the job market associated...

77 But possibly erroneously.

3.1.3 Challenges

The motivations for evolution first include a widening attack surface due to the increasing complexity of embedded systems. These now offer numerous hardware components, complex software and multiple auxiliary channels. These new items are all potential targets for an attacker and with the increase in complexity seldom comes a comparable increase in the validation or protection effort.

Such systems usually have limited computing resources, which makes the most straightforward protection mechanisms, especially those heavily computation-based like cryptography, much less straightforward to use than expected. Embedded systems have limited resources in general and these limitations impact the realistic security mechanisms, especially those some engineers would simply like to import as-is from desktop computers[78]. Energy is most commonly identified first as a limiting factor, but storage space too can be very problematic and not only in permanent storage. For example, public key signatures necessitate a few kilobytes of data for storing the signature, in many cases this is several times bigger than the entire data message a sensor-oriented embedded system would like to send on its network.

At the moment, standards and industry-standard components are evolving pretty fast with the technology of embedded systems which raises additional challenges for the security mechanisms that they could incorporate. This is probably not specific to security issues as these specialized computers are evolving very fast anyway.

Finally, as we saw previously, different security functionalities are expected by the various "users" of these systems, where by user we do not only mean the end user, but also designers, manufacturers, suppliers, operators, governments, etc. The security properties expected by so different users would necessitate very flexible features that are not available in simple security systems. For servicing these complex requirements, practical security mechanisms technically available on these systems are not so varied.

Embedded software is getting more and more complex and make frequent use of efficient programming languages (like C or C++) which are not specifically secure. In some other areas of software engineering for embedded systems some of the proposed programming languages (like Java) have been designed for extension, but dynamic updates with code execution is really a can of worms when considering security. Full security validation of virtual machines with sandboxing is, of course, a topic *à la mode* ; but we fear this field may stay open a long time[79]. And networking with latest embedded systems is using common networking technologies, like WiFi, bluetooth and the Internet.

The combination of increased complexity, extensibility and networking is, in our opinion raising challenges for security management. There is nothing really specific to embedded systems here in fact, but highlighting that the security of those specialized computers is not going to be any easier than for other computers seems to be an underestimated reality check.

3.2 Physical attacks

As embedded systems fully includes the physical part of the computer, physical attacks are worth mentioning in a study of specific security issues. Furthermore, a focus on security-oriented embedded systems[80] allows us to highlight pretty interesting physical attacks targetting the cryptographic processors of smart cards which, in our opinion, provide very instructive examples of complex but deadly vulnerabilities that computer systems may exhibit.

But let's start first by listing conventional classical physical attacks[81] on computing devices that may allow to uncover their most heavily hidden secrets. We may first list direct hardware attacks, like:

- micro-probing, which involves attaching small wires to internals of semi-conductors to have access to their internal state ;
- substrate deconstruction, which involves pealing progressively the layers of a semi-conductor in order to reveal its internal structure, and possibly also its internal state, depending on the environment (at very low temperature levels, the charged state of a logical gate may stay visible for some time) ;

78 A situation which is, in fact, totally normal. Why would off-the-shelf security software be suitable to these so-domain-specific embedded systems?

79 It has not been solved in the last twenty years after all...

80 So specialized computers specialized in security. Double specialization! Security expert computers…

81 We did not write "common" attacks. USB keyloggers or shoulder-sniffing drones for stealing users password are intentionally left aside. Let's try to read the 8192-bits private key of that paranoid government contractor executive from the protected memory of his high-end outrageously expensive smartphone in fair competition. Of course, at the moment, pirates or cyberwarriors alike do not even need to invest in such costly attacks given the usual vulnerability level of most computers, their total despise for rules of engagement and their infinite budget for the aforementioned goodies.

- or, more prosaically but probably more familiar to embedded systems designers, access to debugging interfaces (like industry-standard JTAG , or other industry-specific interface), which may involve soldering but may also simply imply accessing to internal connectors.

These physical attacks allow to access the internal memory of a device and consequently most of its internal data, which is for example extremely interesting for a cryptographic chipset. It may also allow to perturb its normal operation (in order to take advantage of the perturbation).

But these direct hardware attacks have several drawbacks:

- They are usually seen as costly with respect to other attacks, because they necessitate time, specialized equipment and skills, possibly several similar target devices.
- They are sometimes destructive.
- They are usually not sufficient alone and are precursor attacks for other attacks (such as a network intrusion).

Another class of attacks targeting cryptographic processors (which are at the heart of many security kernels) is associated to the exploitation of *auxiliary channels*. First examples of auxiliary channel exploitation were using timing differences in computation loops of a cryptographic algorithm due to speculative execution features of the underlying CPU (the initial Pentium). This idea was further extended to other ciphers with ideas to search for other auxiliary channels of information to improve cryptanalysis [Kocher96], [KSWH98].

The most powerful variants are based on power consumption analysis. Two kind of attacks have been proposed: SPA for simple power analysis, and DPA for differential power analysis. Both techniques take opportunity of the existence of auxiliary channels to find correlation between measurements of the attacked system and secret keys contained in it. Some of these attacks proved to be very efficient (especially the differential variants) and counter-measures are often said to be very costly : they must be implemented very rigorously, they are counter-intuitive and patented with a lot of secrecy. By the way, this means in practice that this is all we can say about these counter-measures. To be fully honest, the reader should also refer to 1.1.1.2 c, though there is always hope[82].

Both attacks are based on a simple observation, which is that the CMOS cell for a basic logical gate has a different power consumption profile when going from 0 to 1 or when going from 1 to 0, i.e. charging or discharging. So a typical power analysis profile (SPA) involves monitoring closely the power consumption of electronic devices, and try to correlate these power variations with the computing device internal state. In initial experiments, monitoring the power consumption of a smart card allowed to identify easily the time intervals corresponding to the execution of the two multiplication intensive parts of a typical RSA implementation. Not only did it allow to identify that the chipset was executing the RSA algorithm, but closer examination allowed to see some of the values of the internal state, which should correspond to the prime factors of the private key. Needless to say that the leak of multiple binary parts of these factors is a fatal weakness to the secrecy of this key, which security relies on the difficulty of big numbers factorization.

Basic defences to raw simple power analysis certainly involve adding noise to the signal or modifying the implementation to introduce dummy instructions or de-synchronization into the cryptographic chipset operation. However, noise can be eliminating by averaging or by DPA. DPA implies computing the differences between two power measurements curves, themselves averaging several successive measurements. Both operation will remove much of the added noise and amplify the information available for the cryptanalysis. Consecutive attack steps are chosen to correspond to small modifications of inputs so the information leaked shows the correlation between the modifications made and the internal secret values. Several variants and improvements have been proposed over power analysis, with the remarkable exploitation of noise as another usable auxiliary channel [GST2014] later renewed via old school EMA remastering [GPPTY2016] ; but overall, the global weakness is summarized in one of the seminal articles on this type of physical attack : *Cryptosystem designers frequently assume that secrets will be manipulated in closed, reliable computing environments. Unfortunately, actual computers and microchips leak information about the operations they process. This paper* ([DPA99]) *examines specific methods for analysing power consumption measurements to find secret keys from tamper resistant devices.*

Cryptosystem designers very probably learned from the design mistake they made two decades ago. Whether secure systems designers also learned something probably still needs to be confirmed ; but secure computing needs to take into account hardware as well as software protection and this is especially prominent when dealing with those specialized computers appearing in embedded systems.

82 There must be someone honest in this industry… No? And who invents such clever an attack cannot be fully bad. Add to that the fact that the author may simply be ignorant of how to protect his chipsets from DPA.

Trusted computing is an area of the computing industry which has tried several times to bring attack resistance to general purpose computing. Somewhat similar from what you would expect from a smart-card, but not as a separate component and with the design objective to be much more integrated into the overall computer hardware platform[83], this industry collaboration provided a chipset which is interesting to study as a physical component for protection.

3.3 TPM

This chipset is usually designated by the TPM acronym, for *Trusted Platform Module*. TPM, as defined by its designers which formed the Trusted Computing Group (http://www.trustedcomputinggroup.org/) aimed to be an *open, vendor neutral, industry standards for hardware-enabled trusted computing and security*. In 2008, its promoters counted all the big hardware vendor names of the industry.

A successor to less successful older initiatives, TPM has seen a pretty wide industry acceptance, though primarily offered on professional product lines and now enters its second decade of existence. The first version of the specification is TPM 1.2 and is now complemented by an up to date though not backward compatible upgrade which is TPM 2.0. The main 1.2 vs 2.0 difference is that while TPM 1.2 has SHA1 and RSA2048, TPM 2 is designed to have many possible algorithms. They called it *algorithm agility*. There is no special requirement for any implementation of TPM 2 to support any specific algorithm, so you actually have to query a given chipset to see what it supports. According to some user[84] *The bedrock for TPM2 in the West seems to be RSA1024-2048, ECC and AES for crypto and SHA1 and SHA256 for hashes.*

Accordingly, while TPM 1.2 had root keys stored inside, TPM 2.0 has seeds of these and a key derivation function.

From the point of view of a teaching book, TPM is also pretty interesting due to the availability of extended documentation surrounding its specification and potential use cases. Some of these use cases are now a little outdated (e.g. those related to mobile phone operator lock-in) but this literature allows to illustrate not only the basic security blocks implemented in the device, but also the way it may be used to provide some security functions. The evolution of the specification towards a TPM 2.0 version has confirmed its establishment as a hardware solution, though this success also lead to a multiplication of application documents that does not necessarily help clarifying the subject.

The generic reference architecture adopted by the TCG is a very simple computer architecture, with a CPU surrounded by memory, a display and a communication controller giving access to them as well as other devices (fixed or removable), a boot ROM and the TPM itself. The objective is to cover a wide spectrum of computing devices, including those corresponding to embedded systems (except maybe from the energy point of view).

83 And probably also the industrial objective to be just different from a smart-card, because well, it has to.

84 https://blog.hansenpartnership.com/tpm2-and-linux/

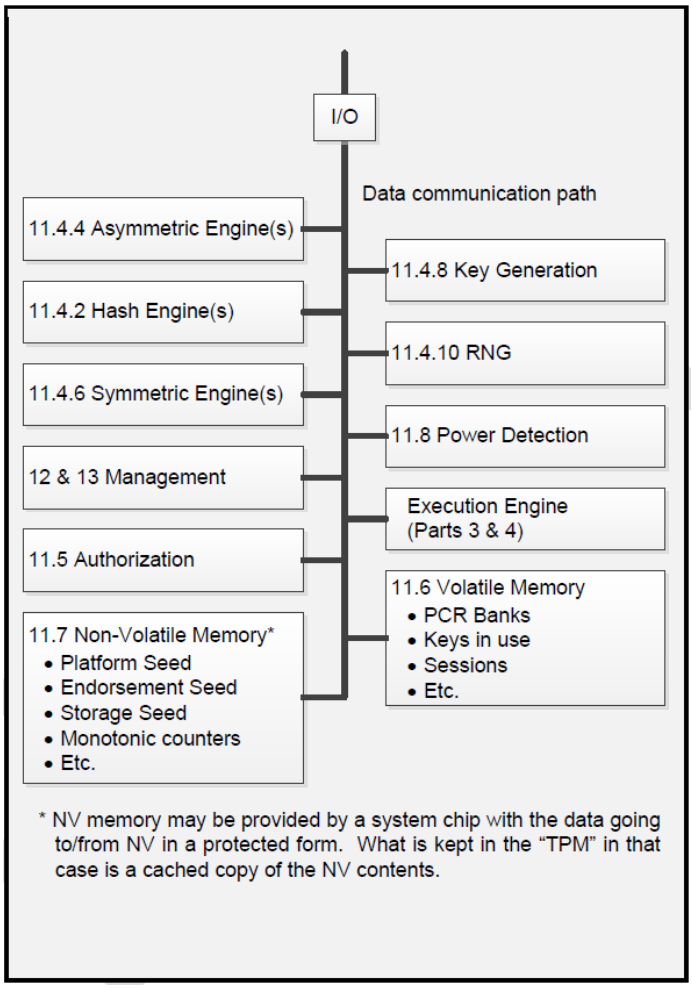


Figure 4: TPM 2.0 Architectural overview

The fundamental trusted platform features offered by TPM chipsets are associated to the following designations (sometimes specific to this hardware community, but otherwise understandable) :

- protected capabilities:
  - either shielded locations (protected memory, specifically designed registers, etc.)
  - and hardware security-oriented features: key management, random number generators, sealing, etc.
- attestation functionalities:
  - which offer various functions to certify some results, from the point of view of the TPM, to the platform or with respect to the rest of the platform[85] or simply authentication of the platform for a remote communication
- and integrity functions, called measurement, logging and reporting with associated registers and metrics definitions (nicknamed PCR, for platform configuration registers) for providing proofs about the state of the platform, and attesting measurements made.

Typical usage of this kind of device (in its older version) is demonstrated by its use with Linux to secure the boot process. It is noteworthy to remark that several years were needed to reach a prototype implementation with Linux, most certainly due to the fact that this is low level programming and that software security engineers were probably busy elsewhere. Anyway, such an example implementation working is interesting to follow in order to have an idea of the security offered as well as the intricacies of starting a full Unix kernel while still providing strong guarantees on the security of the boot process (with a physical hardware element)[86] :

85 This means the TPM may have autonomous access to, e.g. the RAM of the platform, in order to perform its own verifications.

86 Note also we focus on TPM 1.2, no known workflow for TPM 2 are really clear, though support was added to linux in

- First, one need to enable the TPM in the device BIOS and load adequate drivers.
- Then, install the adequate software (tpm-tools and TrouSers are useful keywords in this context).
- Take ownership of your device's TPM. Note this is a one way operation that will, once and for all, activate the security features and give you control over them ; which also means that if you loose control, nobody may be able to help you regain it. Nobody means nobody here. You are on your own for real. Do *not* forget the password.
- Afterwards you need to install and setup a TPM-aware bootloader, most probably TrustedGRUB and most probably after having compiled it yourself[87]. Setting up this alternative bootloader will allow you to examine in more detail the operation of the TPM:
  - First restart successfully, of course
  - And then have a look at the various registers holding "attestation values" (i.e. some form of signature, most frequently in the simplest form of a secure hash function result, e.g. 2.3.4.2). These registers (PCRs in TPM 1.2) are used to check all the parts of the system participating to the boot operation :
    - The BIOS and ROM code itself,
    - the master boot record and stage1[88] boot information,
    - then the bootloader code verification (stage 2, usually in 2 parts)
    - then the command line arguments and the optional ones entered via the boot shell prompt (checked and signed probably after verification of the boot loader)
    - then several registers containing the verification of all the files configured to be checked by the TPM, including the cricital files associated to the operating system startup (for example kernel, initrd, modules under most Linux distribution).
    - At this step, several registers of the TPM may already be dedicated to the boot procedure security, others should still be available for further usage by security services of the operating system.
- To use some TPM features, one could then try to add some new files to the boot checking procedure and verify that alteration of these files will generate detectable warnings at startup time.
- Of course, do not alter the critical boot files or risk not to be able to boot. Note that, contrarily to many embedded devices, un-bricking procedures may not be available or may require that you recalculate signatures of existing files (hence activate the TPM).
- Up to this point, a Unix kernel has been started with decent security controls (more on the potential vulnerabilities later) but it remains to be used. Obvious use cases would involve key management functions for asymmetric cryptography, filesystem encryption, user key ring management, etc. See the appropriate project, if it is still alive. Maybe start your own, preferably in the embedded systems domain, though a TPM may not be the ideal solution in these cases.

A final issue is associated with TPM and TCG related technology. It is more of a political debate than a technological one, though the risk analysis aspect is somewhat interesting in these. Many people nicknamed the trusted computing terminology as *treacherous computing* in order to outline the fact that these industry technologies also find their roots in (mostly USA) governmental-funded projects and that the design choices carefully protect the ability from state authorities to find some path into the security of the system. Neither these accusations nor their rebuttals are usually backed by verifiable information, so we are still left with little facts to analyse in this matter from an objective[89] point of view, though we can always simply read the specifications (the old version), which states that *The TPM has the EK generated before the end customer receives the platform. (…) 1. The EK MUST be a 2048bit RSA key (…) c. The PRIVKEY SHALL exist only in a TPMshielded location* ([TPM2007] rev. 103, section 5 lines 1004-1040). Through this specification, we see the initial core key of the TPM 1.2 is based on a reasonably secure asymmetric encryption

kernel 4.0. Check: https://blog.hansenpartnership.com/tpm2-and-linux/ for the latest information on T2.

87 If you do not know how to compile TrustedGRUB and you did not take the previous point warning seriously, please, stop the experiment entirely. (Though the author suspects you are not reading footnotes either.)

88 The stage 1 of the bootloader is the initial small program that loads the bootloader… Stage 2 is the bootloader itself, which loads the kernel.

89 A factual baseline to which we will try to stick to like an adhesive dressing ([Hergé56], pp.45-49) in an educational document like the present one.

algorithm (e.g. according to NIST [NIST80057]), stored in the protected location, but generated before the chipset is sent to its end user hence under control of the manufacturer.

4 Software development and security

Once upon a time, a world writeable memory access device vulnerability was reported on a commonly available smartphone (CVE-2012-6422). Somehow remarkable was the fact that this security problem was very probably deliberately introduced in the phone software (the actual reason is still not know to the author, most probably for "convenience") and affected millions of users (most probably some are still affected and most still do not care at all). So, from the point of view of this text, the most noteworthy aspect of this security event is a remark of another security engineer, summing it up all nicely with respect to the most common attitude towards software development and security in our age:

"*My experience from most places: nobody cares, nobody reviews. If a problem is discovered later, we will fix it later – why worry now and delay the release? What "/dev/mem"?? Enough with this mumbo-jumbo we have a release to make and management bonuses to earn.*

In fact people who do care and worry about esoteric things like "security", or "good design" or "code quality" are universally viewed as trouble-makers or ivory tower idiots both by management and most of the engineers. It is an uphill battle even to do what used to be the baseline 10-15 years ago."[90]

All warnings given to the reader on the somehow slightly desperate perspectives of this section, let's climb the ivory stairs and throw a few arrows on the wealthy software merchants wandering nearby. All those leaving the trouble-makers team, even for pragmatic reasons, before (and *including*) section 4.4 will be thrown additional stones.

4.1 Security requirements engineering

Security requirements definition is not an exact science as it heavily involves capturing users concerns and evaluation of assets values. Furthermore, it is not so frequently done. So we mostly try to list general guidelines and some common pitfalls to avoid in order to obtain such requirements in the best possible way.

The first thing to note is that requirements elicitation should be done as early as possible with respect to a project or an application. Most specifically, it has to be completed at the application design phase. Some even say it should be started earlier, because security issues should theoretically already have been studied at the project feasibility stage. However, completion of security definition at the application design phase is a more realistic objective and most sensible because that's the only way for security to stay cost effective. Adding security to a design in production is either a technical nightmare (i.e. it is not done) or a money sink (which does not survive very long in cost-conscious environment, though some cases have shown extreme resilience to those added costs in recent years). In most other cases, it is our feeling that including appropriate security features can stay cost effective. With respect to this statement, we leave aside the most stringent security requirements imaginable in academic studies or security-critical systems ; which still cannot benefit from the incremental value of decades of public effort like in other areas of computer science. But if we focus on specific pragmatic concerns and system functions, achievable security can be at a pretty high technical level using affordable technology. What probably cannot be solved is the impossible (and so common) industry demand of adding security afterwards. If you can afford to add security later, then you can certainly live without it entirely and you probably already decided to do that, so just find and read another book and most probably hire a marketing assistant to counter a competitor FUD communication[91]. If you study the security of your application, do it before you start coding.

When security requirements are considered, definition documents have a strong tendency at the moment to contain general lists of security features, like passwords constraints, firewalls installation requirements, antivirus software budgets, etc. Unfortunately, like for any functional feature, we have first to outline that implementation mechanisms are *not* requirements. Such formulation is simply erroneous and these mechanisms are mentioned as an implicit intent to satisfy unstated requirements that necessitate further explanation : the need for authenticated access (with a certain quality level), the need to protect some specific data, the need to communicate with integrity guarantees with some third party, etc. This pitfall is probably due to a strong lack of practice in the domain of security requirements definition, but off-the-shelf security features or devices should be banned from requirements documents to make place for better analysis of actual security needs of the industry, company, users, etc. (and those incapable of eliciting these requirements should be replaced by those who have the knowledge and who do not only want to sell something specific).

90 https://lwn.net/Articles/529496/

91 In the worst case, stop lying to yourself about your security requirements, get yourself a conscience and retire urgently your dangerous systems before someone gets hurt. Nothing technical or even process oriented here.

Security requirements are also frequently defined in an isolated section where they exist by themselves (frequently copied from a generic set, which exacerbates the previous trend). Though this is not necessarily bad in itself, attention should be given to the fact this does not reveal a lack of analysis and adaptation to the target. Security requirements are requirements and like all requirements should be extracted from an elicitation process that allow user inputs and specialization to the actual target. It is there that the legitimacy of the security constraints finds its true roots and that users can accept security rules (because yes, that's perfectly possible that users agree with security rules and cease seeing them negatively – it should be the norm in fact).

In requirements engineering, a lot of attention is given to what the system should do. That's logical. Security necessitates a change of point of view that requirement engineering should pay attention to. Security requirements are associated to what the system or the application should *not* do (in any case) [MMEBA2008]. The perspective change is necessary, but uncommon and uncomfortable for many users, so attention should be paid to how these requirements are going to be captured. Original techniques may be needed. (No one said such techniques should be depressing.)

Another view on the classical project lifecycle (needs, specs, devel, testing, validation, operation ; modulo the appropriate variant) allows to illustrate the specificities of security with respect to projects phases.

We already underlined the first issue: security should be taken into account pretty early. Security policy enforcement should be considered as early as in the opportunity study phase. Security needs are part of the project definition and security properties specifications should be included in the project specification among all the other "non functional" requirements (safety for example). Once specified, a security-specific activity is much less prominent in the project life. Developers can really implement security functions like any other functions (and they usually adopt themselves some of the programming rules we will see later on, even if they are not imposed on them). Of course, bad development will lead to bad software, whatever the field.

Security validation or security-specific configuration[92] (for example, of the environment) is a work-intensive phase appearing at the near end of the project validation, mostly seen as the end of the project from the software developer point of view. Before putting the system in exploitation, security-oriented concerns are usually submitted to heavy scrutiny, both from the point of view of security officers, who frequently (and legitimately) see new code as a new source of numerous software vulnerabilities and careless users, and from the point of view of project managers who frequently (and questionably[93]) see security officers as surrealistic technocrats or paranoid naysayers. Such close examination may or may not help improve the security of the software, but up to the author knowledge, very rarely prevents it from entering production[94], though security requirements may see a fast update beforehand[95].

Things are not always as dark as these lines suggest[96]. Monitoring and management functions may improve due to careful specification of security features. Henceforth, exploitation may proceed peacefully and give a lot of input to the operational security teams that closely monitor the security improvement of the system[97]. In fact, contrary to what most project managers consider with respect to software development, a significant part of the security work related to the system takes place during its operational life. The fact that development too could extend past initial production steps is out of the scope of this document. But with respect to security, most of the actual work may occur during operation: users and access rights configuration, intrusion alerts management when available, data classification and confidentiality management, availability are all aspects which will necessitate additional work during the exploitation of the system. This work is not negligible with respect to the security investment done on the entire project. It is mostly this part that makes us warn about the need to study security requirements very early at the design phase, because early investments will pay off hugely in savings with respect to exploitation and management of security during production. This is surely not specific to security, but it really has a tremendous impact in this field. Good security mechanisms design may nearly eliminate all exploitation maintenance, while poor security design of an otherwise widely successful software may generates huge hidden costs to its users for security maintenance (see 4.1.1).

92 Frequently seen designated as security *hardening*.

93 Obviously, the author is totally objective on the matter, as any reader checking his bio could attest. (Do not forget to invite me and recommend me on the professional social network of the day.)

94 Only rarer is the case where a vulnerability (or even a system failure) will lead someone to halt a system preventively.

95 We really mean that end users and project managers alike usually really consider removing all the failed security tests from their own requirements document in order to shortcut security validation and start operation at their own risk. We could humbly consider this a failure in security requirements elicitation process, but that would not fit 1.1.1.1 b any more.

96 Sometimes it's worse. I swear!

97 Stop dreaming!

A last point illustrates the wide difference of point of view that emerges sometimes between security and application oriented developers. The last steps of a project lifecycle are of importance to security functions, and disposal is certainly a step in the security procedures, if only to achieve decent destruction of data. On the other hand, abandoning systems is still a frequent way of halting a project for most IT managers. Needless to say that abandoned systems data is of most interest to attackers, especially those targetting confidential information. Gathering interest over data removal or system destruction inside an IT department is frequently very difficult. It is surprisingly difficult to gather adequate service for timely data destruction, mostly because it is difficult to motivate people enough about the subject. It is conversely surprisingly easy to define a satisfying data destruction procedure. We used to go into more detail about that, but we found a very entertaining reference on the subject to which we refer the reader. Let's just say that you can trust good old physical procedures [Haigney2017] though you can always try something more complex if you feel inclined ([Gutmann96] is a nice initial reference to start from).

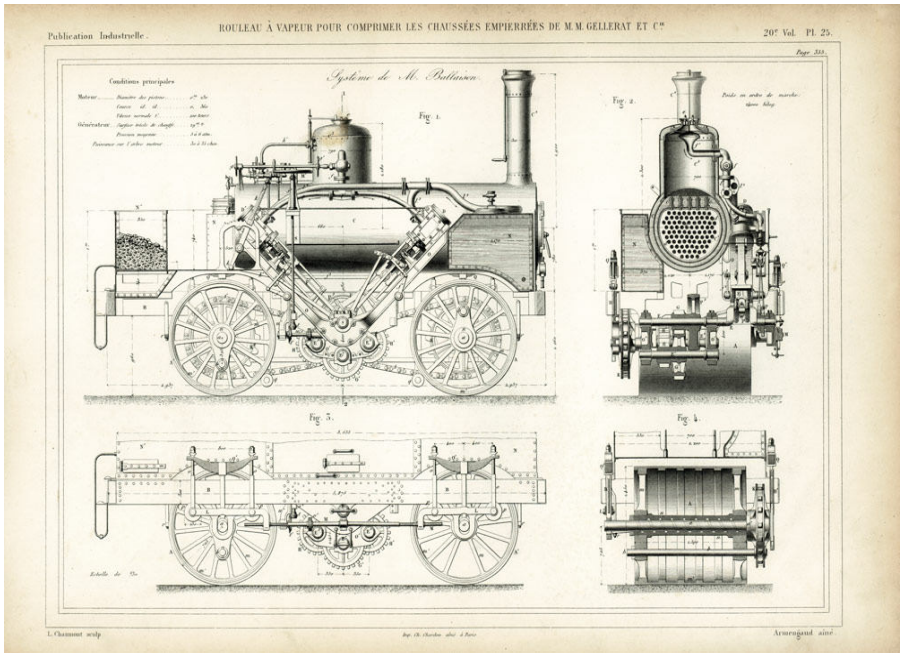


Figure 5: Innovative, open source, general purpose, good old and ground breaking digital data destruction device

4.1.1 Note on security updates

As an illustration of the consequences of not trying to integrate security requirements early in the design of software, let us have a look at what happens when some successful piece of software is given late attention with respect to security. In such cases, we may need to invent some original way of managing software vulnerabilities (instead of resorting to boring classical textbook).

For example, we may propose to wait until the vulnerabilities are identified in the wild or even until they are actively exploited by an attacker. The target will helpfully provide the manufacturer some information (and possibly some motivation) to help find the vulnerability. Armed with the information gathered by the first victims sacrifices, the manufacturer would quickly provide a patch that corrects the problem – *if possible* without introducing a new one (because providing fast fixes of, e.g. complex core operating system kernel functions, is always a little tricky). Then we would provide a worldwide technique for distributing efficiently over highly privileged channels these binary fixes to all computers (with intermediate server proxies we could easily cover the whole planet[98]). Of course, the manufacturer software managers and most security-oriented managers would still whine loudly because system administrators do not install patches fast enough – though smiling secretly in private as these undoubtedly customer-induced delays would probably discharge the aforementioned managers of most legal responsibility over the vulnerabilities.

98 And even create jobs for all the customer-paid system administrators of these additional servers.

Astonishingly to us, this funny technique for managing software security vulnerabilities is very popular. It is called software updates, all serious editors do that and provide heavy software for supporting it, while most companies invested and created jobs to manage the receiving side of the infrastructure, and users apparently really feel more secure after their work has been interrupted by security updates – even if those have the bad habit to install forever in a loop from time to time (but it improved).

Well, possibly some day someone will remember the days where people expected software to be secure initially and all the time, *without updates*. We used to call that secure software. The drawbacks of the approaches relying solely on (known and exploited) vulnerabilities patching had long been identified and even anticipated by those trying to work on developing such software.

In the meantime, we will keep on reporting on the funny situations induced by application of this systematic updates approach to securing system in fields where it is entirely inadequate, such as those of many embedded systems. A blatant example of the shortcoming of this approach have been given e.g. by the first drones firmware security update proposed for a model of a publicly available flying drone. This precise patch integrated a no-fly zone in the drone firmware. It was provided by the manufacturer at the request of an angry citizen who found one of those drones lying on the lawn. The motivation of the manufacturer celerity to producing the patch can be better understood by analysing the location of the no-fly zone : a 15.5 miles radius circular area around some house located in the middle of Washington D.C. (the big white building). As French citizens, we grumble a lot on the reasons why our own smaller but older property located in the middle of Paris was not given appropriate consideration. We could even consider that many other critical industrial or military buildings could be worth considering[99]. But you know how French people are… always arguing about something ; when they are given for free real blatant examples of why security updates cannot work.[100]

4.1.2 Risk analysis

Risk analysis results are a very interesting input to software developers, system architects and project managers that care about security.

To nuance immediately however, the frequent trend of people doing risk analysis to extend their study on all links to security issues and embrace all the spectrum of everything is somehow of disturbance. Risk analysers can be a little annoying also to implement focused technical mechanisms and pretty annoying to get the adequate career recognition for developing good security software. The last point may be due to the fact that funding and money is the key interest of risk analysis, a fact that does not necessarily attract people with the right skills for letting you invest in your own ideas at the detriment of theirs.

However, a risk management step is always a good idea in the first works associated to addressing security. A risk analysis (contrarily to what most consultants sell) is frequently very configurable in scope and length, down and including one minute thoughts about the things you care most[101]. Therefore, let us address briefly but efficiently how to approach the analysis, its benefits and its shortcomings.

A risk analysis (whatever method) usually involves the five following steps:

1. In the first step, we focus on assets identification and evaluation of their value. This evaluation is usually simply made in terms of money value. We are not necessarily material security officers, but money is a convenient way of rapidly (albeit unreliably) comparing things and nearly everything can be roughly evaluated in monetary terms even if it involves morally questionable shortcuts[102]. This material evaluation is indeed very useful for comparing concerns of differing magnitude and identifying objectively critical concerns over personal individual judgement. The objective is not to put a cost on everything, quite the contrary, but to give an *honest list of valuable items*. This list may be pretty difficult to build, especially in the context of information systems where value does not necessarily relate to topologically delimited things, but may also be information or some computation results. Furthermore, this (high) value is not at all necessarily an economic value, it may have many qualitative dimensions (e.g. family pictures, artwork, intellectual creation). So, the endeavour of this step is really to build a list of assets and give estimates of their value in easily understandable ways.

99 Not to speak about my own home privacy !

100 See, e.g.: https://www.wired.com/2015/02/white-house-drone/

101 In your software. Extending the analysis to ordering everything a man should care about in his lifetime risks getting too long. (I would not dare order a thing about the other genre.)

102 Such as adequate compensation for loss of human life, work cost for rebuilding data, legal costs for impunity attempts, etc.

2. The second step is to sort this list according to the security priority given to them. This step shows that value is clearly not the only criteria (or the second step would simply be done automatically). Here, we will define the things we really want to protect and the choices made : reputation versus money, business versus customers, cargo versus boat, passengers versus vehicle or whatever strange dilemma hazards and imagination may throw at you. Fortunately for the security officers who do not feel at ease with tragical roles, many of those decisions are pretty reasonable and do not necessitate engaging one's soul. At least in the first years.
3. Once the most important assets are identified, actual analysis of the system operation can allow to identify some vulnerabilities, some threats and derive potential damages made to the system by these threats.
4. More information can be entered in the analysis process by considering threats priority. Most of the time, this step is, for the author, the one most questionable in risk analysis methods. By definition in security, threats are intelligent attackers and intelligent attackers do not explain their attack plan beforehand, quite the contrary. Therefore, threat determination is an oracle problem.
5. Finally, all these analysis steps allow for the optimization of counter-measures selection in order to protect the most valuable assets from the most dangerous threats and vulnerabilities and start a virtuous improvement circle.

These steps, both in theory or practice, should help the reader understand that risk analysis is inherently a qualitative approach, in the sense that it is based on human/expert opinion. This is not necessarily a drawback of the approach[103].

These methods are easily applicable to many contexts: organizations, systems, products. Similarly, the length and detail of analysis can be tuned. This is not necessarily an advantage (because you may never know when to stop your analysis).

Several methodologies have been proposed to bring some rigour to conducting a risk analysis. Some of these approaches offer tremendous help for practical application and are currently or have been well recognized among security professionals. In these methods we find names like :

- MARION, MEHARI, EBIOS, etc.
- HAZOP, FMEA, ISO31000, etc.

The ISO 27000 standards family is the latest and most popular incarnation of this work. Of course, out of popularity, specific selection of a precise standard is frequently the source of endless debates. For example, the above list has been cleverly ordered from a totally factual point of view: French methods versus the rest of the world. Clearly a decisive factor sometimes.

Outside of nationality, risk analysis techniques offer several advantages from the point of view of the author :

- First of all, they allow identification of the assets to protect in the security target, as well as a possible estimation of their values. (Realistically, the asset values are not so frequently given.)
- When given, these monetary values offer a rare opportunity to budget realistically for protection[104], if only simply by investing a fraction of this value.
- Risk analysis reports can be given in plain language and are quite easier to understand than assembly language exploits or cryptographic hash functions. Hence your target audience starts to understand computer security issues and is frequently willing to help – at least until budget capture specialists clear the field for you. End users understanding also frequently help better allocation of risk management strategies.
- Risk analysis clearly identifies the various risk management alternatives. Frequently, engineers mostly focus on risk reduction, but other options are available :
  - risk transfer : via insurance, state[105], etc.
  - risk acceptance : because in some cases, there is the feeling that the only realistic alternative is to live with a risk (usually minor, possibly temporary), but these decisions are better taken collectively at the appropriate management level.
  - risk reduction : through work, additional work and then more work, etc.

103 For example, you can easily spot bad risk analysis consultants when they claim that they rely on beyond dispute scientific approaches, without even having to know their (possibly confidential) secret techniques.

104 Note to executives : also an easy way to spot those who do not like much those *realistic* budgets.

105 For example : one does not usually manage the risk of military invasion oneself.

- risk avoidance : frequently neglected, risk avoidance may be interesting at the technical level, especially for information systems – just choosing another algorithm, another implementation technique, etc. may allow to avoid a given risk altogether – but frequently unpopular[106].

- Both the readability and the value integration of a risk analysis presentation offer the opportunity to clarify management priorities with respect to security to orient the overall security policy of the organization.

In front of these advantages, risk analysis exhibits also several drawbacks, which its proponents frequently forget to outline :

- The first and big problem is due to the fact that most analysis methodologies involve analysis of the threats and attackers faced by the security target. While general ideas may be given, a good inventory of potential threats and attackers starts to look like an oracle problem : if only we knew them all, why not simply eliminate them ?
- The other pretty common problem with risk analysis is that this is usually the technique which is frequently used to demonstrate that all risks are already managed. A qualitative technique is also of course manipulable, if only by selecting the right experts. Executives remember this and rarely want to show an unfavourable picture of their own risk management, be it justified or not.
- In practice, recent risk management methodologies have started more and more to rely exclusively on best practices and "standard" risks lists (see 1.1.1.1 c auditors certificates too). But these lists, whatever their quality at the moment of production, do not help to target real assets in actual organizations and especially to convince users to confess their probable value. In the worst case, these lists fuels paranoia by frightening executives or employees with extremely unlikely scenarios or similarly orient them over ready-made useless tools.
- Experience frequently confirms that actual research of management priorities does not end well. Management rarely wants to decide. Most of the time, many managers have reached their position by avoiding conflict and promoting consensus and compromise. Arbitration of theoretical tragic choices between two equally important divisions of their own company with respect to fictional computer attacks does not really fit the picture ; neither officially publicizing the security policy rules that put customers data protection at a (much) lower priority than business objectives and bonuses attribution. So decisions may not exactly take the expected form[107].
- Finally good risk analysis does not always end well morally speaking. For example, product lifetime optimization is a typical application of well done and well tuned risk management, for the benefit of the manufacturer to the detriment of the customer. Risk analysis being inherently viewpoint-based, its optimization may not be in favour of the needy (in the Robin Hood sense).

So overall, even if risk analysis is a nearly mandatory first step of many computer security engineering processes, it should not be considered alone.

4.2 Static verification and (secure) software development tools

The need for secure development has fueled research and development of secure development tools.

4.2.1 Source code analysis tools

For careful developers, understanding some of the secure programming techniques and origins of security bugs mostly immediately renew the envy for automation that frequently fuel their interest for machines. Indeed, in many cases, improvement of software security looks like a perfect playground for automated help of software developers :

- security bugs are especially dangerous (due to the failure impact deliberately added by the attackers) ;
- they are commonly associated to intentional programming faults which impact the programmer did not predict ;
- they usually involve intricate and complex pieces of software.

106 Just try the obvious suggestions : use another operating system, use another middleware, use another networking technology, use another CPU, use another programming language, use the compiler (yourself). See if people really want to avoid risks...

107 Like in : "*You have to protect both ; with the same budget. (Yes, the restrictions decided last week included.)*"

Source code analysis tools are obviously very helpful to try to address them. On the other hand, secure software programming rules have a strong trend to lead to boring development rules so many programmers do not really like them ; automating for enforcing them can be a nice option to have them comply more happily. Whether programmer satisfaction is really worth any effort compared to other approaches – for example immediate replacement by new software development methodologies – given the observation of figure 1 is left as a political debate to the reader.

In any case, among all the source code analysis machinery developed through computer science, we can try to qualify the variables capabilities of security analysers according to several features sets (which, by the way, may not all necessitate access to the source code but may sometimes be performed over the compiled software) :

- First, the simple direct study of calls to potentially insecure library functions (or less frequently[108], the invocation of unsecure programs).
- Bounds checking analysers and those detecting simple (scalar) type confusion are those which will allow to address the most common sources of vulnerabilities in classical low level (and efficient) programming languages (C and the like), typically buffer overflows.
- More advanced analysers will address complex type confusions as well as pointer arithmetic and more generally pointers-related software bugs that may lead to memory corruption issues or exploitable vulnerabilities. The analysis involves more complex calculation for static analysis of program behaviour or sometimes dynamic checking.
- Memory management errors are not only prone to security exploitation but also a common target for dedicated debugging tools which frequently favour dynamic analysis or program instrumentation for detecting these problems at test time. These are pretty useful and are a good example of the fact that software analysers useful for security are not necessarily security-specific tools.
- More advanced program analysis techniques available in complex modern analysers or compilers may allow for detection of vulnerabilities that involve sequences of operations, via control flow analysis of the program (especially with respect to known problematic sequences).
- Dedicated analysis techniques, like data flow analysis or pointer aliasing analysis are frequently helpful in order to improve control flow analysis results (especially in order to reduce false alarms). But more generally, static software analysers, and compilers (which frequently incorporate a specific analysis technique once it has shown its usefulness), offers strong opportunities to improve the detection of potential software vulnerabilities.

Outside of vulnerabilities removal, software development tools may also help in implementing mitigation techniques that will help further protect the system software. They also frequently appear in compilers with a link with analysis tools. But the latter are more related to vulnerability prevention (or removal in the development phase), so we make a distinction. And the actual mechanisms involved are also frequently pretty different and we will look at them later, in association with coding techniques.

Several classes of tools evaluation can be envisaged, especially from a commercialization point of view:

- source code analysis tools
  - simple searching tool (grep-like)
  - lexical analysis
  - abstract syntax tree (AST) construction and analysis (parsing)
  - advanced works
    - global / local analysis
    - type calculus, logical reasoning, range calculus
    - false alarms reduction techniques
    - IDE integration, specification-based verifications (or testing [MFCLWS2009])
    - etc.
- penetration testing tools
  - port scanners : nmap, etc.
  - vulnerability scanners : nessus, etc.

108 Because the persistence of unsecure programs usage on secure systems is far from easy to envisage wisely except from a very stupid point of view.

- application scanners, or web application assessment software.

Some lists (for source code analysers, other tools may operate on binary or bytecode):

- OWASP's: https://www.owasp.org/index.php/Source_Code_Analysis_Tools
- NIST's: https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html
- OWASP's (for dynamic analysis or vulnerability testing): https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools

4.2.2 Code integrity

Successful usage of checking tools may improve a lot the baseline security of software. Additional measures can also of course be applied, but among tooling preventing the occurrence of security vulnerability introduction in software, code integrity checks are also pretty important. The above tools address intentional but non-malicious development faults (most commonly bugs) but with respect to security we must also take into account malicious faults, that could be introduced deliberately inside the source code (after analysers examination of course).

In order to protect the software against such malicious alteration[109], code integrity protection procedures and tools should be used. The usage of cryptographic signatures of source code coupled with integration with a source code version management software is among the available approaches. At the moment, classical solutions in the open source domain rely on PGP and using it with the git version management software and its most common instantiation on the GitHub service. This allows you to sign your commit tags using a cryptographic asymmetric secret keys and verify source code pulled from the service. Additional signing may be needed in order not only to certify the contents of the commit, but also its intent and position in the versioning[110] tree (on git pushes).

The guide at: https://github.com/lfit/itpol/blob/master/protecting-code-integrity.md proposes a practical example of using these tools to improve the integrity guarantees offered during the development process.

Though this guide is specifically suited for open source software development, the features are certainly applicable to any development, including the personal private key protection aspects for developers (here on a smart cart).

4.3 Security Evaluation Criteria

4.3.1 Security standards as criteria

The first sets of security standards was established in the military domain by the United States Department of Defense, soon followed by other countries in the formulation of security standards for computer systems and a unification effort finally concluded at ISO in the 2000s :

- TCSEC – Trusted Computer System Evaluation Criteria – DoD 1985 (Orange book) and TNI – Trusted Network Interpretation of the TCSEC (Red book) – and later documents grouped in the *Rainbow series*.
- ITSEC – Information Technology Security Evaluation Criteria (EEC[111] 1991)
- JCSEC (Japan), CTCPEC (Canada), etc.
- CC – Common Criteria also known as ISO15408 (ISO standard since ~2000)

Astonishingly for us young apprentices, the oldest *Orange book* from the pentagon still offers valuable teaching interest due to its adherence to a qualitative separation of systems into different levels of security with a meaning from the *functionality* point of view. 7 levels were available in the Orange book:

109 At least, at the source code level [Thomson84].

110 In order to avoid an attack using a previous faulty commit submission, initially correctly signed but discarded from the software and that could be reused by an attacker to perturb a later version.

111 *Id est* EU nowadays.

Figure 6: The rainbow series documents

- Minimal protection (level D) which also meant failure to succeed demonstrating a better level on all verification criteria[112].
- Discretionary protection as in
  - discretionary access control policy (level C1)
  - added with logging for auditing most operating system operations (level C2[113])
- Mandatory protection with
  - object labels for simple mandatory (multilevel) access control policies (level B1)
  - structured protection (level B2)
  - security domains (level B3)
- and verified protection (level A) with formal verification of the security kernel.

Each level is cumulative and includes the functionalities of the former. In addition to the classification of the system with respect to these levels that imposed the availability of (more and more advanced) security functions, assurance criteria were demanded in several dimensions to guarantee the effectiveness of these functions. In particular, specific assurance elements were needed on the following topics for gaining the evaluation status[114] :

- with respect to the available security policy in the system :
  - on discretionary access control implementation ;
  - on object reuse control (ie. logical data destruction) ;
  - (security) labels operation ;
  - on mandatory access control implementation (if available).
- in relation with imputability :
  - on identification and authentication of users ;
  - on the availability of a trusted path (to prevent trojan horses) ;
  - on audit.
- with respect to assurance of security in the operational life :
  - system security architecture rationale ;
  - system integrity guarantees ;

112 As in “Try again, boys.”
113 Hence the “C2 audit” logging functions of several operating systems nowadays.
114 An additional *assurance level* qualifies the *quality* of the evidence brought to reach the level.

- covert channel analysis ;
- installation management ;
- and secure recovery (in case of security failure).

- in relationship to the life cycle of the system (development primarily) :
  - security tests ;
  - specification and verification process ;
  - configuration management (for development) ;
  - secure distribution (of the software to the customer).
- and concerning the security documentation available :
  - in the user guide ;
  - in the installation manuel which had to provide a secure installation procedure ;
  - in the tests documentation ;
  - and documentation for the security management.

These various elements were the *criteria* upon which the evaluation of the claimed level was performed and a given assurance of the achievement granted (or not). *Security criteria* hence was the overall name of the qualification process. It stayed in the terminology of computer security.

Note that, whatever the terminology, this evaluation remains an ordinal and qualitative one. Though the expert advice is very well bordered by exhaustive guidelines on the evaluation process and state of the basic mechanisms requirements for various class, his or her expertise is still key to the final evaluation decision. (Later on in the nineties, people might have spoken of certification rather than evaluation.)

After the TCSEC, the European initiative on the ITSEC brought an interesting modification. Instead of wiring the description of the target of evaluation in the criteria documents[115] and thus somehow limiting the scope of the standard, the ITSEC asked the candidate to define this target of evaluation in terms of functionalities and assurance criteria to provide (within borders, somehow rather similar to the TCSEC for common computer systems). With respect to the functionalities claimed in the target of evaluation (and adequate evidence), assurance correctness criteria allowed to classify the final security level of a (successfully) evaluated system between 6 levels, from E1 to E6.

Some effectiveness assurance criteria allowed to further evaluate the security of the system (given its security functionality level) in the following dimension:

- security of system construction :
  - suitability of functionality ;
  - binding of functionalities ;
  - strength of mechanisms ;
  - construction vulnerability assessment ;
- security of system operation :
  - ease of use ;
  - operational vulnerability assessment.

4.3.2 Common criteria / ISO 15408

Finally the *common criteria (CC)*, which later became the ISO 15408 standard, further enlarged this vision by not only allowing the submitter to propose a given target of evaluation, but by offering the opportunity to define protection profiles as a kind of models of a common target of evaluation. Initially mostly presented to allow reuse between versions of the same product, these profiles quickly became an opportunity in the industry to define protection profiles for classes of products in order to allow for easier evaluation of product security. Though the approach may have succeeded in a sense due to the augmentation of the number of evaluated products available for security purchase (though the augmentation of the proportion of security products evaluated among all computer products may still have to be confirmed), it also raised the concern of an industrial consensus over some protection profiles functionality sets which may not be very difficult to claim but which offer limited interest to users.

115 Yes, we are oversimplifying.

For example, a later widely available Linux distribution (successful) evaluation raised the following comment among kernel developers, which illustrates very well our concern : « *For the most part, the protection profiles define away nearly all of the interesting threats that most systems face today.* » in Fedora and CAPP, lwn.net, 10 dec. 2008. Earlier, a very successful operating system managed to obtain an EAL4 evaluation and the associated favourable press for a desktop system without any networking device (which obviously facilitated the evaluation but considerably reduced the practical relevance of the target of evaluation).

Since 2012, the CCRA management committee agreed on a harmonization on the application of the CC moving to a more protection-profile (PP) oriented way of using the criteria. The idea was to facilitate the development of protection profiles based on work between government agencies, product vendors and evaluation laboratories. These protection profiles were intended to be used for procurement purposes in several nations. The move to more standardization of the protection profiles should have supported the goal of reasonable, comparable, reproducible and cost-effective evaluation results. The common protection profiles would also somehow offer more guarantees to purchasers that a meaningful security target was embedded in these protection profiles (to defeat rogue evaluations). Evaluation was to be done against these protection profiles, if not mutual recognition of security target evaluations would be limited to a low level.

Outside of these concerns, the Common Criteria standardized several elements of terminology and their acronyms associated to security evaluations :

- Target of Evaluation (TOE): which is the system subject of the evaluation.
- Protection Profile (PP): a document, which identifies the security requirements for a class of security devices relevant to a group of users (and generally written with this user community).
- Security Target (ST): a document that identifies the security properties of the TOE, possibly using one or more protection profiles.
- Security Functional Requirements (SFR): a document which specify all the individual security function provided by the subject of evaluation. The CC presents a standard catalogue of such functions and identifies the dependencies of such functions where they exist. It if is not using standard elements (whether functions or full profiles), the SFRs document must describe the security functions in detail.
- Security Assurance Requirements (SAR): documents and measures taken in the development process and used in the evaluation process to give a level of trust to the product security features. Specific requirements for particular targets or types of products are documented in the ST or PP.
- Evaluation Assurance Level (EAL) : the EAL is the numerical rating describing the result of the evaluation of a given SAR. CC lists seven levels, ranging from EAL1 to EAL7. Higher EAL do not necessarily imply better security. They mean that the security assurance of the TOE has been much more extensively verified. Better security depends not only on the EAL level achieved but also on the available security functions in the TOE and their adequacy to the security needs. Of course, in practice, many people forget to read the specification document and focus only on the final number[116].

The implication of the use of PP(s) and ST documents for the evaluation of products was an opportunity of generic models definition and possibly cost reduction for evaluation but also opportunity for an abuse of “narrow” ST definitions for marketing-oriented security ratings. Governmental bodies reacted by validating some specific PP(s), see for example: https://www.niap-ccevs.org/Profile/PP.cfm or https://www.ssi.gouv.fr/entreprise/produits-certifies/cc/profils-de-protection/ . These PPs provide much more detailed lists of security functions expected from trusted systems but their variety make them relatively difficult to describe in detail in this document. We encourage the reader to browse them but, first and foremost, to remember that precise examination of the security functions and security requirements available and associated to a computer system is always required before trusting it – whatever the rating marks printed on the case.

4.3.3 Note on DO-178C

In the context of this course lectures, we also need to go into more details into the standards of the aeronautics field with respect to software engineering. That is to say the RTCA DO-178C.

116 So, there is a caveat in the human resources screening rules for security procurement people : you have to fire first those who do not know what the ST document is, but you may keep those who do not know to count to seven if your security assurance level target is not so high (note the latter criteria could also ensure prices remain affordable, though your mileage may vary).

Other standards numbers have been allocated to cybersecurity-specific issues, but as with many things in cybersecurity, little is known about the actual content and things to do with these standards, outside of their numbers which are DO-326A, DO-355 and DO-356[117].

With respect to DO-178C, software is a part of the design process (ARP 4754A), DO-254 addressing the hardware development life-cycle and DO-178C the software development life-cycle. The given safety assessment (ARP4761) of the intended aircraft function addressed by the system determines a software (criticality) level among the 5 levels labelled from A to E (from the most critical to the non critical one) with the associated denomination : A (catastrophic), B (hazardous), C (major), D (minor), E (no safety effect).

Each part of software life cycle process is a section of DO-178C:

- System aspects relating to software development (section 2)
- Software life cycle (section 3)
- Software life cycle processes
  - Software planning process (section 4)
  - Software development processes (section 5)
    - Software requirements process
    - Software design process
    - Software coding process
    - Integration process
  - Integral processes
    - Software verification process (section 6)
    - Software configuration management process (section 7)
    - Software quality assurance process (section 8)
    - Certification liaison process (section 9)
- Overview of certification process (section 10)
- Software life cycle data (section 11)
- Additional considerations (section 12)

Traceability is full for (high level) software developed under DO-178C, starting with system requirements down to source code (and even object code) via high level and low level requirements.

There are several supplements to DO-178C (and its ground-based companion DO-278A[118])

- DO-330 – Tool qualification
- DO-331 – Model based development
- DO-332 – Object orientated technology
- DO-333 – Formal methods
- DO-248C – Supporting information for DO-178C and DO-278A

4.3.4 Alternatives

- SQUALE (AC097 of FP4-ACTS): http://cordis.europa.eu/project/rcn/30538_en.html

4.4 Coding

117 And, to be fully complete and honest, about a few things they do *not* address: aircraft or ground physical security, airport, airline or air traffic and communication or navigation. Whether it is useful for an airworthiness standard not to address these issues is left to the reader for appreciation. Please refer directly to the standard description for the actual document scope summary (the author still has not managed to figure how to understand it).

118 DO-278A is *GUIDELINES FOR COMMUNICATION, NAVIGATION, SURVEILLANCE, AND AIR TRAFFIC MANAGEMENT (CNS/ATM) SYSTEMS SOFTWARE INTEGRITY ASSURANCE*, obviously accompanying the former.

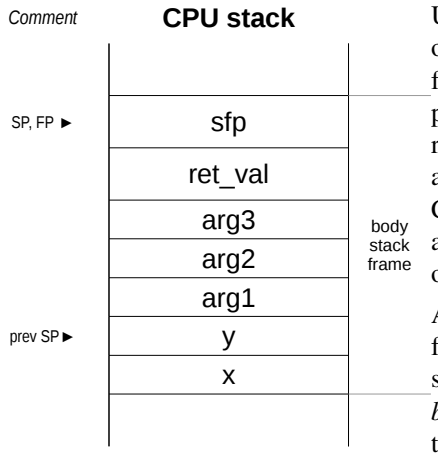
This section is the most important of the text[119], hence hidden somewhere in the middle. Here we are going to talk about programming real software and trying to avoid the most common security mistakes. Regular practice and wise control of this aspect of programming will give you a decisive and definitive advantage over most other programmers. Your software will have the same kind of superiority over all other unsecure software. The author suspects that running such software will give the feeling of operating a 20th century tank in the middle of a middle-age battle: why bother with plate or mail armour, crossbows, bows and arrows if you can easily drive something like a M4 Sherman? Upgrading to the cyberspace equivalent of the latest AMX Leclerc or M1 Abrams is left as an exercise to the (hopefully well funded) reader[120]. Needless to say, all these investments will also extend your (virtual) life accordingly[121].

4.4.1 Frequent or knowledgeable attack classes

Most of the time currently, acquiring a security job (which novice students are always prioritizing) involves primarily knowing attacks, even ultimately finding a new one[122]. Indeed, some inspiration is to be found initially in the examination of the most frequent sources of development faults that elementary attacks exploit. They show that software developers typically ignore some of the ways their software can operate, which obviously pave the way to abuse. They also show that such abuse (naturally) implies using the software or the computer in non conventional ways, more or less complex, most frequently pretty low level (hence attractive to hardware and microprocessor designers and generally very boring to graph colouring algorithms designers), but certainly not as complex as ICBM design[123].

4.4.1.1 Understanding buffer overflows

On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind in [Levy96] is the main and first publicly available description of this classical source of security issues. Buffer overflows, or more precisely stack-based buffer overflows, intentionally provoke such fault-inducing conditions to redirect the CPU execution to an instructions sequence unintended by the original software developer or the legitimate computer user.



Understanding a little more precisely what happens during such a buffer overflow involves describing what happens when a function is called (in C) from the main program body on a common stack-based CPU. Arguments passed to the function are generally stored on the stack first. Then general registers (e.g. status bits or the current stack pointer) are saved on the stack and the CPU return address is saved on the stack for (later) function return. Control is then transferred to the called function program code. Many details are dependent on the precise type of hardware architecture used but the overall idea is the same.

Afterwards, we should look also into more details into the way the called function operates. Again in C, most local variables of a function are also stored on the stack, including local fixed size arrays, e.g. small temporary *buffers*. Now, if the called function can be induced into manipulating in this temporary area some input data bigger than the expected size of this buffer, of course the stack components will get corrupted. However, the idea of the manipulation is to cleverly corrupt the original return address saved on the stack so that the end of the function does not send the CPU execution into a random area of memory but to a carefully chosen address under control of the malicious user. (In the initial variants of

119 And this is the motivation harangue of the part. Time to get bloody.

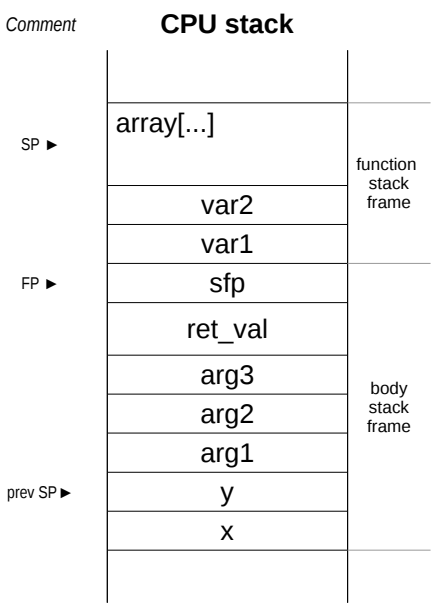
120 Appropriate verifications of solvency resolved, the author is of course available for (expensive) consulting or even (exorbitant) management positions on the topic. Results guaranteed (for the moment). Seriously.

121 We do not say anything about hair loss or fertility because we would not like to sound over optimistic, but still some promising internal results have leaked.

122 Just to get rid of the issue as fast as possible: in order to find a "new" attack and forever impress recruiters, just look at some existing old attack class and find a recent spot where developers made the same kind of mistake *once again*. You can also note how the author rewards students who *patiently* read hundreds of footnotes. But do *not* do the same to find a vulnerability, use all possible shortcuts like an attacker, lazy learner or successful career builder would. And remember you can only shine in middle-age cyber-battles with such a basic weapon.

123 Small trap directed at cybersecurity analysts blindly using keywords to grep wiretapped Internet traffic. (NB: Shame on you by the way: quit and go get a real job !)

these tricks, this zone was even on the stack itself inside the just overflown buffer, further illustrating the idea of clever software integration, unfortunately from malicious programmers. Needless to say, most modern CPUs are expected not to execute code known to be located inside their stack area.[124])



Such a disruption of the normal execution path of the program can be more or less useful for malicious users depending on the precise case but illustrates the way low level details of the computer operation (including knowledge about both the CPU and the C compiler) can be abused to mount an illegitimate execution.

Note that, conversely, ways of systematically disrupting these attack venues are also numerous provided the computer designers accept to integrate potentially simple security concerns in either the CPU or compiler design. Apparently it also takes some incredibly difficult change in their inner mind because it took several decades to generalize the concepts, at least at the software levels, while the hardware issues are apparently just starting to get settled[125].

However, in the absence of the generic protections (and even in their presence) the most common software programming guidelines given to prevent the occurrence of buffer overflows in C code are:

- to be careful when writing into buffers, and most precisely to always check the length of the input and output memory areas when copying ;
- to never do any tricks in C that are not totally mastered ;
- to forbid the usage of functions that do not fully check the length of their arguments, even in the standard library (most precisely strcpy and strcat are to be avoided) ;
- to never do any trick in C (we already said it?).

4.4.1.2 Format strings

Many standard C display functions use a format for printing: printf(), sprintf(), fprintf(), etc. Most of the time two variants of such functions exist: one with and one without such format string.

When user input is passed to such functions, it can generate output describing the programs internal. For example, passing “%x” to a straight printf() will guide it to print its next argument from the stack. This kind of situation may allow to access areas of memory for reading. In case such areas of memory hold an interesting secret, such a format string will allow anyone yet not knowing it to see it.

It is important to never pass a string with user-supplied data as a format without using ‘%s’. An attacker can put format specifiers in the string to mangle the stack, leading to a possible security hole. See https://man.openbsd.org/printf.3 for example. Getting information on the internals of a running operation can also be a precursor to another attack necessitating runtime knowledge, such as the precise position of the program in memory.

Note system logging functions usage may also be affected by variants of this issue.

4.4.1.3 Arithmetic overflow

Arithmetic overflow occurs in finite precision computer arithmetic when the result of a calculation, most frequently a multiplication, is bigger than the maximum value storable in the (finite precision memory) variables involved. Such a case is well known among programmers dealing with control and command software (because in these cases, the commanded devices usually does some variant of the universally known bad thing) but most programmers do not frequently see these corner cases as problematic for most software. They are wrong, because most of the time such

124 Provided the operating system does not do something stupid to allow it again. Oh well...

125 Admittedly, stack execution prevention or write-execute memory access rights separation were integrated into CPU architectures in early 2000s ; but several of the most disruptive attacks techniques that make the headlines in these late 2010s are still based on advanced hardware *features* (memory organization, cache memory, speculative execution, etc.) that downplayed “a little bit” the security issues.

overflows are exploitable by attackers in order to trigger abnormal software execution conditions and further exploit them.

The most common situation is dynamic (heap) memory allocation using a variant of the common idiom:

```
data = (struct item *) malloc(n * sizeof(struct item));
if (data == NULL) {
   return ENOMEM;
}
```

where `n` is an integer value computed from user input. In this program, if `n` is big enough overflow may occur in the call to `malloc()` and a small memory allocation may be done because `n * sizeof(struct item)` will be a small value. This erroneous buffer allocation could open the path to a memory overflow.

The right idiom to use in this case should systematically rely on the convenient and extremely less known `calloc()`:

```
data = (struct item *) calloc(n, sizeof(struct item));
```

In this case, the allocated space is also initialized to zero, an operation the author considers beneficial too. Performance addicts can further consider `reallocarray()` to avoid this overhead[126].

4.4.1.4 SQL Injection

SQL injection is another classical software vulnerability that especially appeared in the headline at the beginning of the 2000s with the rise of the number of Internet facing company and merchant websites. Most of these sites were relying on some kind of database in the background and using them via dynamically built queries. Frequently, these queries were vulnerable to carefully crafted input submitted in order to trigger unwanted execution of SQL commands.

Imagine for example the site program code builds one query statement using a variant of the following command :

```
statement = "SELECT * FROM users WHERE name = '"+ userName+"'AND pwd = '"
            + userPassword + "' ;"
```

What happens if the given user name is a carefully crafted string like `johndoe' OR 1=1; --'` ? In this case, the final part of the statement query involving the check of the user password will be commented out and replaced by an always valid condition. Of course, the actual password given to the site login procedure will not be a problem any more. Maybe even a fake user name will be accepted (which could raise a few additional difficulties for later recovery).

Then, simply elaborating on the idea, after giving input like `' OR 1=1; DROP TALBES; --'` to the site, the entire application data may be discarded[127].

Mitigation techniques involve using database access APIs more completely rather than just trying to submit crafted strings to the RDBMS engine. Most specifically, prepared statements should be favoured to access databases via prepared queries. Such mechanisms, available in all RDBMS access libraries, separate the setup and parsing of the query itself, using placeholders for parametrization, and the execution where these placeholders are associated to actual input parameters. This is both beneficial to performance (query setup time, including all possible query path optimizations, is amortized over all calls) and to security in the sense that playing tricks based on the interactions between the query string (statically built into the program) and the parameters strings (usually supplied at runtime by the user) is not normally possible. In such a case, the above query statement would look like this in the program code :

```
query1 = prepare("SELECT * FROM users WHERE name = ? AND pwd = ?;")
```

while actual execution would look like this later on :

```
query1→run(userName, userPassword);
```

Alternate mitigation approaches may involve relying on fully external libraries for mapping memory data types to persistent storage transparently. This has usually other motivations than simply security and is linked to the chosen software architecture.

In the example given, the software architect may also want to eliminate the problem altogether by delegating authentication to an external library (which will itself use a RDBMS or an LDAP directory or whatever component it wants) while possibly preserving RDMBS access for non security-critical aspects[128].

126 Attentive readers have certainly noticed performance addicts have also silently and simultaneously been switched to a different operating system where the function is available (https://man.openbsd.org/reallocarray.3). This switch was done for obvious safety and security reasons (to protect themselves and others from potential kinetic harm). Performance addicts certainly don't have the time to read the present comments so they should not notice.

127 I know I resort to stupid obfuscation. This is unfortunately pretty revealing.

128 This could be a good example of risk avoidance. At least, no ghost user can place orders to try to steal some goods with falsified orders. Resistance of the website to denial of service attacks has not improved but that's a start.

Many other common guides recommend parsing or escaping input parameters but we do *not* favour this approach at all, unless you are yourself implementing such a RDBMS interface library (in which case, we heavily recommend full blown parsing of parameters, not mere abnormal comment or special characters removal attempts which are often incomplete).

A last approach usable fully independently of seditious development teams is to resort to intrusion detection mechanisms for monitoring the input flow to the application (usually available on the network in these modern always-connected days). However, we equally disfavour such an approach which is often incomplete, especially due to the next point.

4.4.1.5 Code or input obfuscation

SQL injection attacks are also pretty interesting from the teaching point of view because they offer several easy obfuscation opportunities for attackers to avoid detection from intrusion detection systems by cleverly using all the opportunities offered by the SQL language.

- Simple obfuscation techniques, like mixing comments and keywords as in SEL/**/ECT, are available in this language and already shows that intrusion detection engines monitoring the network could have some hard time analysing all this.
- But attackers imagination of course went further with ideas like:
  - abuse of white space or comments ;
  - fragmentation of the injected query at the network level ;
  - interaction with apparently independent HTTP parameters ;
  - additional abuse of comments (such as RDBMS implementation-specific handling of special or ill-formed comments) ;
  - use of unprobed areas in packets for attack-specific items storage (depending on specific implementations of detection systems).
- Possible lessons of such detection avoidance techniques seen in the field are that:
  - A full blown parser for parameter validation may not be overkill nor so complex to build[129].
  - Intrusion detection is not so easy in fact when they want to avoid detection.
  - Some people do know how to use all the intrusion detection tools efficiently: the attackers (in order to test their own malware discretion or to take advantage of detection weaknesses too).

4.4.1.6 Race conditions

Race conditions sometimes allow to exploit program execution to induce forbidden behaviour. Here is for example a classical and *insecure* way of creating a temporary file (in /tmp) while trying not to overwrite it :

```
/* Generate random file name */
name = mktemp("/tmp/tmp.XXXXXXXXXX"); – XXX is replaced by process-specific things at runtime
/* verify file does not exist */
if (stat(name,&statbuf) == 0) {
   return EEXISTS;
}
/* ok, open it */
fd = open(name, O_RDWR);
```

This code opens a possible race condition with another concurrent process using the *randomly* (but frequently predictable) generated file name. Simply imagine that another process creates a link named against the predicted /tmp/tmp.XXXXXXXXXX pattern but pointing to a critical system file between the stat() check and the actual open() call? Remember that processes execute concurrently on systems. The former program will therefore open the critical system file instead of what he thinks to be a private temporary storage. Obviously, later on, your mileage may vary.

Frequently, system developers or administrators downplay the feasibility of such abuse. (Very generally, most system designers downplay actual exploitability of their work. They simply demonstrate their lack of maturity with respect to security. Simple source code modifications that get rid of the issue altogether usually take much less time and effort than endless debates or demonstration software implementation[130].) They are simply plain wrong.

129 With adequate tooling of course. Time for totally free advertising: www.antlr.org.

130 The only positive aspect is seeing old men or women disputing like young children. Unfortunately without the adaptability.

mktemp() was deprecated in the POSIX.1 standard associated to operating system interfaces in 2011 due to the difficulty using it correctly[131]. mkstemp() is to be used to replace both system calls in an atomic operation as in:

```
fd = mkstemp("/tmp/tmp.XXXXXXXXXX");
```
– Simultaneously check name availability and create a temporary file

The older approach is to use the following open() flags to trigger an error if the file already exists :

```
fd = open(name, O_CREAT | O_EXCL);
```

This is also the opportunity for us here to point at the need for software developers to take the time to have a look at the differences between the fopen(3) library function and the open(2) system call, or between the FILE* streams and the (non negative) integer file descriptors. Not only for showing off at developers meetings but also for actual coding. By the way, did I tell you not to do any tricks in C or any other low level programming language[132]?

4.4.1.7 Awkward things

Most frequently, security problems appear in awkward contexts[133]. The author likes to repeat the example of a security management checking script that was once abused using a questionable behaviour of the vi editor that the above script was using in order to send a mail report to the administrator. The abuse involved creating a file for the script to detect and report. But the filename of this file was using these questionable vi escape sequences so that the mail generation, instead of a simple listing report, became the actual moment of malicious commands execution. While believing bringing a minor improvement to the system security state, the security script henceforth became the instrument of one of the attacks it was designed to prevent[134].

4.4.2 Practical recommendations

4.4.2.1 Design first

Many modern programs and systems are simply broken and insecure by design. Self-proclaimed computer security experts do little more than apply recipes and mimic their neighbours. Given the security state of the neighbourhood, insecurity contamination prospers. Authentication is probably an iconic example of this lack of security design : storing user names and passwords still gathers unanimous consideration while incredibly more interesting authentication mechanisms have been proposed in the literature for decades. When you design a computer system to obey blindly at anyone spelling the letters of the “root password”, well it does. If you are not satisfied with the security of this mode of authentication, first you have to design a different mode of operation. (If you are content with it, then you will never be able to use the system in a context where identity theft is a concern[135].)

Authorization management similarly shows its limitations: it is still designed around Unix rwx or POSIX ACLs functionalities. So when security needs are listed by users, there is also a need to design security functionalities corresponding to them (admittedly, you are left with ample time to do what you want before the requirements reach you and even more before the funds are allocated). And you may need serious imagination or bibliographical work before getting your design right because the field has seen much less investments than one would like.

But anyway, design security first, imagine security functionalities, evaluate their applicability, test them in the field, rinse, repeat and have users realistically pay for that.

a - Know common faults

In order to design first, of course you should have an idea of the most commonly exploited defects. Knowing how some vulnerabilities are exploited is interesting.

It will also fuel possible intrusion detection systems intended to somehow reduce the impact of vulnerabilities possibly remaining in operation (e.g. which may be due to unavoidable users interactions).

---

131 Possibly also as a tribute to the lassitude of those explaining the security issue again and again. At least it makes me feel better to think about it like this on rainy days.

132 We didn’t even have a footnote saying not to do any tricks in C !

133 Note that awkward things for experienced C software developers are probably near from *unbelievably mischievous* for regularly trained human beings.

134 Imagine the oracle of Delphi predicting it to the system administrator: “*You will be r00ted by a file in /tmp*”… Then the system administrator creates a script to check, remove and report all the temporary files and schedules it for daily execution. Et voilà ! Trojan horses are not the only tribute we have to send to ΕΓΓΛΣ in our field. I will call this the Pythia attack.

135 Well, wait. This is nearly always a concern with networked computers. Do we really intend to unplug then?

You may even want to evaluate systems against these known security problems and imagine some attack simulation in order to stress them, though this is much more of a marketing issue than you may think.

b - Do not stop there

Because, in fact, demonstrating a successful attack on a system should only lead one to stop using it, no? Neither the lack of any successful attack is a real reason to fully trust a system, especially if they were only attempted by your 4 years old little brother[136] or if, even more cleverly, nobody ever tried to attack it seriously because this is prohibited by law[137].

So it is not possible to stop system security design at the basic enumeration of known vulnerabilities and possible workarounds, especially using a list of security patches as a shortcut conveniently omitting software development fault details. Neither can you simply delegate to later security testing the demonstration of security properties outside of simplistic systems accessible to exhaustive testing (where exhaustive reachable states enumeration is feasible).

c - Architectural principles

As soon as computer systems become complex enough to raise interest (both from legitimate users and attackers) they also start to become too complex to think about their security naively from a mere a posteriori outsider point of view. There is a need to organize the system hardware or software architecture around several principles allowing to address security needs with the adequate level of concern. The introduction of a network leading to a distributed system further complicate the design effort needed to build an adequate architecture. Such architectural or functionality principles (which you usually find again in the underlying principles of the highest level of security certification standards) incorporate :

- the least privilege principle, which should lead designers to give minimal rights to computer processes in order to limit security failure impact ;
- the defence in depth idea, which justifies the existence of multiple and apparently duplicate protection mechanisms with the objective to prevent that a given security error propagates to a full visible failure ;
- the explicit management of the notion of delegation, which both serves the least privilege principle and the accurate representation of security rights, possibly further complemented by multilevel security mechanisms (security labels as well as mandatory rules) to implement security confinement ;
- the secure by default configuration setup, popularized in some popular open source system and which frequently lacks in commercial systems ;
- full, extensive documentation of the system and its security operational, with as many footnotes[138] and details as needed to train the final user or the administrator for adequate security management or for the system management in general ;
- careful design of secure protocols operation in a networked context ;
- clear specification of the security properties expected from the system, of the frontier of the trusted computing base (TCB) and explicit (in our opinion open and public) assurance of the reached security level
    - with the adequate level of *formal verification* for the highest trust level.

These principles can and should guide a design process that produces decent levels of security, even possibly a high level of security if major means are put to the task to cover everything in detail.

Many of the current popular security recipes unfortunately do not fall very well in these guidelines : adding a firewall, multiplying passwords[139], adding awkward password selection filters, logging useless data extensively, deploying patching infrastructures, signing complex code and data blindly, etc.

---

136 I do not mean in any way that 4-years-old-proof testing is meaningless. Quite the contrary, confrontation with a well motivated and imaginative children can be quite a challenge for a supposedly well engineered computer system. The author could easily tell a couple more fun stories if we were not already inside a footnote. But maybe one could also expect a little *more* from security professionals than a computer system which can resist children or amateur experiments, no?

137 Oh, by the way, this is the case most the time. Nowadays, even if you are (more or less) the owner of the system. Should have said it earlier instead of simply warning against trying too much to attack systems: you are legally obliged to work on protection ! (Why are you hiring all those *pentesters* then ?)

138 Not only in the *legalese*!

139 Password selection rules with weird characters or spelling are often presented as a security feature : is it because once the user has forgotten his or her password the security has improved?

This is primarily a symptom of the lack of design effort put into proposing adequate and useful security mechanisms that really improve the trust of end users. This is certainly emphasized by the usual lack of explicit evaluation of their security needs by the latter ; but we stay convinced that good design work can lead to adequate and even commercially successful trusted systems. Admittedly, the track of actual success stories in the field is still pretty short ; but it means there is still a lot of opportunity for growth as they say in the big-business schools[140].

d - Especially APIs and protocols

The building blocks for computer architectures and distributed systems architectures are the programming interfaces of the software components and the communication protocols defined in these architectures. Incorporation of security requirements into those elements is not easy : both share the characteristic that they cannot predict all their usage environments in advance and that they frequently bring into play multiple computing processes.

For communication protocols, authentication and encryption mechanisms have been proposed in several standardized protocols (most prominently Kerberos and TLS), but outside of these industry standards other works are also available to study the security of protocols and provide good security properties. It is primarily a lack of industrial investment into this area that slows down the incorporation of the advanced protocols proposed in the literature or the academic domain. Investors should simply update their strategies accordingly unless they simply want to wait for governments to do it (whatever the moral[141] issues underlying the debate, the author note that it has been pending for too long and all the stakeholders now should simply make the needed contribution[142] – forget about censoring the Internet by the way, this is not what is needed).

Widely available security-oriented libraries of high quality are equivalently rare. Outside of a few cryptographic libraries and core authentication functionality, there are not many software components for addressing complex issues like (operational usable) security policy management, users rights distribution and revocation or application software authorization checking. A few specific projects offer such software and the renewed investment they need is also the reason why we insist in this section.

As a final note, we underline that hardware components too deserve a hand. Processor architectures incorporating dedicated, notable and useful security features are certainly still to be popularized[143].

4.4.2.2 Obscurity does not help

Hiding internal information with respect to the design of the system or trying to mask its hardware setup or obfuscating the machine code or the bytecode or even the source code, and any likewise obscurantist idea is *never beneficial* to security.

Obscurity is not confidentiality. Exploits against closed source may be just as easy to realize as against open source software, the difficulty relies mostly in the type of software fault and the software development environment, not in the source code availability. On the contrary, obfuscation will primarily work against people writing code, especially those trying to fix it or workaround it and not those trying to perturb the operation. Sometimes, obfuscation mechanisms disguise as security mechanisms: encrypting code with a key not specifically protected and stored nearby is an example of sophisticated obscurity that does not build upon serious design. The attacker will simply get the key first if he needs it (which may not even be the case, classic example of CBC versus ECB cipher modes).

In the field of cryptography, many of the secret industrial ciphers hidden by some industry players have shown their weaknesses as soon as the actual cipher design was known open. Obviously, serious attackers gaining access to this industrial implementation documentation (simply available for a fee and a NDA[144]) might have acquired such knowledge much earlier. In any case, hiding a cryptographic algorithm is contrary to all cryptographers guidelines of

140 We all knew they were followers of Emile Coué. They even think students will easily find the tuition money… But then they try to apply suggestion to others too!

141 LOL!

142 Note the contribution of end users is probably limited to sincere clarification of their trust expectations. Furthermore probably only a few good selfless souls have the real skills and knowledge for the task. So yes, it means all the others have left to bring to the table is money and probably a big chunk of it. We know they are reluctant to do that, but they always are and they always find a way to recoup their losses anyway so we are not so worried.

143 We only know of http://www.cl.cam.ac.uk/research/security/ctsrd/cheri/ and http://www.draper.com/solution/inherently-secure-processor as exceptions.

144 *Oh my god, do you mean attackers sign NDA documents in bad faith?!?* I do mean a few less obvious other things too...

algorithm design which recommend to progress simultaneously on the cryptographic and cryptanalysis aspects. Only major countries governmental agencies can claim to do that without being laughed at[145].

a - This paragraph should not need to be written

We should not have to write these lines, but many still have reasons to promote obscurity, whether to favour their private interests, to hide their misbehaviours or by mere laziness[146]. Obviously they do not know what trust is built upon, they do not help.

We will let the reader evaluate himself or herself whether obscurity can still be beneficial to other aspects of one's activity, but with respect to obscurity benefits for computer security or for cryptography, we think the final word has been spoken. There is none.

4.4.2.3 Quality is security

Most security problems are simple bugs. In this case, fixing them is the straightforward way of dealing with the issue and no security-specific point of view is needed. (In our point of view, even urgency is not impacted as some architectural measures, like defence in depth, should be introduced in the first place in order to limit urgent cases to those where full mission abandon is really considered.)

The importance of dealing with these cases has always been regularly put forward by the founder of the Linux kernel project, usually pretty harshly, as in:

As a security person, you need to repeat this mantra: "security problems are just bugs" and you need to _internalize_ it, instead of scoff at it. […]

I'm deadly serious about this.

Some security people have scoffed at me when I say that security problems are primarily "just bugs". [...]

Because honestly, the kind of security person who doesn't accept that security problems are primarily just bugs, I don't want to work with. If you don't see your job as "debugging first", I'm simply not interested. [Linus Torvalds on LKML, 2017-11-16, *edited*[147]]

We forgive the nervous reaction to the security circus even if it is undiscriminating targets (seriously, nobody gets used to some security experts surrealistic recommendations, even their colleagues) but we nuance slightly our position with respect to this one in the sense that we do not see all security issues as bugs. Lack of some security features or incomplete documentation or suboptimal behaviour may not directly fall in the "bug" category. But for all the cases where the security problem in question is directly associated to a coding fault, they clearly fall in the bug category and we think that they should be treated accordingly. This is the whole point of this section to state that you can factorize the quality effort aiming at better code to say that they include security bugs and contribute to the overall security of the system by providing better code. In this case, when bugs are fixed, security improves and, indeed, debugging and hardening are the same activity.

Outlining the parallel between software quality improvement activity and security improvement also helps outlining the fact that, like for software quality, there is no security "plug-in". You do not add security to an existing piece of software by adding a software library as much as you do not add quality to some existing software by incorporating some compiler switches. Not even a cryptic elliptic curve cryptographic library or a magical password complexity checking tool[148]. The software improvement process goes through the understanding and documentation of common programming rules, the detection of bad habits and their systematic elimination, audit and screening, tooling ; and global recurrence of this activity over the software production iterations for regular improvement. And at some point, this improvement reaches a plateau and that's usually the time to look into useful new security features (unless the rest of the team feeds enough bad new software functions in the meantime to fuel another iteration).

Like for quality, there is not specific return on investment to expect for regular software security bugs elimination. These are not new functionalities *per se* (yet) so they do not bring a specific ROI (they slowly bring you the inestimable: end users trust). But, empirically, it seems this security improvement activity also bring some advantages to the software development process :

145 Nowadays, the author suspects that academic cryptographers may still be laughing, just in private as a precaution.

146 Without an attacker trying, idleness may very well go unnoticed after all...

147 Original at: http://lkml.iu.edu/hypermail/linux/kernel/1711.2/01701.html

148 On the contrary of quality, some users apparently still believe in the magical securing properties of either : *a*) arcane mathematics functions, or *b*) heavily pain-inflicting but simple to understand scrambling add-ons.

- shorter test cycles ;
- less bugs, so less time spent fixing them ;
- and usually a better efficiency overall (because speed hacks frequently fall short in fact and usually lead to underperforming insecure software instead).

4.4.2.4 Multiple lines of protection are useful

Possibly differently of quality or safety management, homogeneous optimization of software security functionality is not the endeavour of secure development. If attackers find a vulnerability in your line of defence – and they certainly will find one because your software is as imperfect as any human creation can be – they will exploit it. On the contrary of physical phenomenon (like lightning, storms, cosmic rays, etc.) hitting uniformly[149] safety defences guided only by fundamental forces of nature, attacks will typically concentrate fast on the weakest point and may occur only once your vulnerability is known. Therefore, multiple lines of protection are really useful, if only because they could allow system owners to take at least some action before full system compromise prevents them from trusting it entirely – i.e. before long and costly recovery[150]. Investing in such juxtaposed security mechanisms targetting the same aspects of protection obviously makes extremely uncomfortable most managers in software development teams. Most of them frequently see as evident truth that redundant expenses mean an opportunity to save costs and not an opportunity of added protection[151].

However, this is simply false. When faced with malicious opposition, the core optimization aspect is to focus on the valuable assets or functions that you want to protect. For this perimeter, multiple lines of defence are usually perfectly realistic and enhance the trust end users can faithfully give to the system.

On the contrary, a single line of defence, though evidently more interesting than an unprotected system will fall prey to the current cat and mouse situation where each vulnerability in the software will lead to a full security failure which will necessitate urgent update (in the absence of intrusion) or lead to total distrust (if an attacker exploits the vulnerability before update).

Full specification of the security needs should denote which parts of the system may actually have the most value and justify such architectural decisions. (Historically[152], such core security kernel was only incorporating security elements and was designated as the TCB – *Trusted Computing Base* – because only the core security functions were seen as the building blocks upon which all other protections would rely.)

4.4.2.5 Quality guidelines

In a software project, all quality guidelines will not specifically target security. Software reliability engineers promoting them may have specific separate reasons. From the point of view of security, we put emphasis on some specific aspects which we want to outline here. (Though, frankly, we would really be astonished if they were not already part of many quality engineers own programming recommendations.)

a - Simple code

Many security vulnerabilities come from complex code with unpredicted corner cases. Furthermore, complex code is more difficult to understand and audit, so these vulnerabilities are less easily found. Finally, complex code is also pretty difficult to maintain and modify, so it will be more difficult to correct these vulnerabilities when they are found. So, an obvious recommendation is to ensure that simple code is favoured: small functions, limited use of macros, etc. Roughly speaking, most code should be as boring to write as to audit.

149 At least with a statistical distribution accessible to modelling.

150 Well, of course, this is the theory. In practice, some people do care less from intrusions than from small rain, as in : “Are the intruders using the system to hurt *our* business ?” Obviously, such executives will soon focus their attention on the security officers payroll too, so you had probably better run like Rincewind before they think about you in more detail.

151 Many managers blindly apply the same reasoning to redundant people, as if two guards were not evidently better than one. If you changed the environment, the same careerists would certainly conversely think that not enough redundant mates are accompanying them on a battlefield ; though one day or another, as officers, they would probably consider them expendable resources again.

152 Note we still think this architectural principle is perfectly reasonable *today*.

b - Check errors

The lack of error return code checks is also a common source of software bugs that can lead to exploitable situation especially when these code are those of system calls revealing abnormal changes in the environment that the software developer does not specifically want to manage (such as memory shortage[153]).

c - Fix bug classes

Software programmers frequently make the same kind of mistakes with respect to security impacts. So when a bug is found and corrected, this class of bug should be searched and corrected in the whole software code base, not only in the original location. Obviously, automation should be used as much as possible to facilitate the identification of these problems and possibly prevent reintroduction of similar issues.

Note the Coccinelle[154] software tool is especially known for this kind of bug hunting. A bug hunting bugs is so humanly smart after all...

d - Take care to semantics

Finally, a special recommendation is to take appropriate step to ensure that developers understand the semantics of the libraries they use, especially complex ones. Contrarily to what is frequently assumed, this is certainly not always the case when using corner sides of any operating. As an example, let us question ourselves, certainly old-time Unix users, about the exact semantics of file descriptors inheritance over fork() especially in relation with the precise moment of access rights checking when open()ing a file or over the allowed operations in a C signal handler[155]… Are we really sure of being able to answer these questions exactly ?
The problem is that these difficult to use APIs are precisely the places where attackers will exercise creative thinking in order to send your software into situations where undefined behaviour is frequent and where your software is expected to take of care itself alone, without the help of the operating system. So understanding the limitations and semantics of these APIs is important. Of course, checking erroneous return codes and early error management is frequently a straightforward way to avoid being faced with such subtleties in the first place.

4.4.2.6 Check user input

As is shown in most examples, attacks usually involve supplying specifically crafted input data to vulnerable programs. These programs not only exhibit exploitable bugs, but they also accept such input while it (probably) does not really conform to reasonable input data. Whether this is due to underspecification of the software or to expeditious implementation of input data analysis is not relevant, the truth is that after some time examining examples of security failures, one recommends to take an extremely stringent approach to user input checking. There are only benefits to such implementation strategy (even in resource constrained situations). In many cases, even full blown parsers could be useful when programs accept complex text streams as input. Contrarily to what novice programmers think, such parsers are not so difficult to build – unless you refuse to use adequate tooling of course (but in this case, other security rules should be appealed to).

Another less enticing aspect is that many things should be treated as user input, not only usual run time small user input like entering a number or a string in a form. Reading a file means parsing user input even if those files are configuration files (supposedly) produced by (supposedly) well meaning system administrators. Relying on an environment variable is also similar to user input checking. Accepting messages on a network connexion also means analysing user input (possibly submitted far far away). In all these cases, external data is to be analysed thoroughly before being trusted – or more precisely trusting the variables associated to it to represent the user supplied parameters. Briefly, everything not static in the program text should be treated like user input.

Note some programming environments provide generic mechanisms to help the developer, if not to analyse the data, at least to track user input dissemination in order to avoid relying on internal data possibly computed from external data. This approach is commonly called a *tainting* mechanism. It is not only pretty convenient (especially in those cases where external data sneaks into internal variables via many of the less known program environment parameters[156]) but also easily understood thanks to human being natural trend to beware of “foreign” items and to avoid touching potentially “contaminated” things.

153 In the days of multi-gigabytes RAM capacity, an out of memory error code certainly means that prompt memory release and cheap orderly program exit is the best (and easiest) thing a software can do to help the computer system as a whole.

154 http://coccinelle.lip6.fr/

155 Hint: *only* setting volatile atomic flags is really safe.

156 Think of LD_PRELOAD or PATH environment variables for example.

4.4.2.7 Optimization and language

Even if, as we just mentioned with tainting, the chosen programming language obviously impacts the security development guidelines to adopt when engineering software. However, there is no secure programming language *per se*. It depends on the general programming language properties and its more or less advanced integration of security concerns. We will not enter this debate but obviously, when adressing secure software engineering, such security properties should be examined carefully.

The impact of the development environment is also very important. In this case, we note that even assumed *advantages* of some development components can raise security concerns. A typical example is optimization features. Many modern compilers have the ability to remove non-executable code or do some evaluations at compile-time in order to enable optimizations of program time execution. Such features can be detrimental to security and lead to the removal of some security checks (which, of course, should never be triggered in normal conditions). Like for design, it is pretty nice to see some security mechanisms being optimized out of a program. It really makes the security officer feel so useful[157].

4.4.2.8 Remove code

Long term evolution of secure software should also emphasize a specific aspect frequently underestimated: disposal.

By this designation or the title of this section, we do not necessarily mean removing old code. We mean that too, but let us start with the initial intent. Security needs necessitate, frequently, that some data gets correctly deleted and not merely abandoned in an unused corner of the computer storage. Data deletion is, from the security point of view, pretty different from a simple free(). When the data is stored on magnetic storage things get more complex again, especially with modern smart storage media, until you feel obliged to fall back to those devices we already outlined earlier (e.g. figure 5). But even earlier, removing the cryptographic key of an intermediate session for example usually necessitates specific attention and possibly some good preparation (no mixing with random areas of memory in the first place for example). Secure deletion is another way of looking at secure storage, possibly from the most distinctive point of view. Similarly, secure software has to offer good destruction procedures.

At the design level, we also think that adequate consideration should be given to disposal of old, broken or poorly designed software. We think gets() should have been removed much earlier from the C standard and that some other functions should already be deprecated. Of course, the impact of code removal should be evaluated and adequate advance warning should be given if there are many users of the code, but a decade waiting for security is not acceptable.

157 If only he was allowed to apply similar optimizations techniques to some developers or managers career progression...

5 Cases studies

Case studies of embedded systems security issues are thriving. The organized part of this section (the cleanly numbered subsections) is ageing but still illustrate adequately the issues. However, this introduction is getting fancier and fancier with time. Things the author only dreamed of as fictional examples a few years ago are now turning into real life cases, with pictures, demos and exploits (and unfortunately many marketing wannabes and cyber managers running in). Only proven and classic science fiction writers or directors still have to be outrun by reality.

What do I mean here? Computer systems pretty similar (at least for any computer science graduate) to those found in classic desktop systems are being introduced inside most of the common devices surrounding us in our daily life. They get embedded in our transportation, communication and home devices up to a point that the conventional personal computer of the end of the 20th century now looks like fading away in oblivion obsoleted by these new *smart* devices.

Well, those who call them smart are not necessarily those less fool[158]. When you look at the flourishing wildlife of these embedded systems from the point of view of an attacker (or a cyberwarrior as they call it now) what you see is just a sea of innocent preys waiting to be slaughtered. Some of them do not even have an authentication step but they got connected and will happily obey any well recorded replayed order ; sometime even randomly generated garbage does nice things[159] !

Hunting pictures will be kept for the course presentation because, of course, all the manufacturers of these innocent herbivorous had rather them be presented as innovative devices than easy targets, but classes of such devices belong to the following list.

- Electricity or water home meters.
- Avionics networking switches.
- Automotive networks.
- Insulin pumps (medicine delivery devices).
- Home automation things (from fridges to toasters).
- Toys.
- Drones:
  - flying,
  - driving,
  - or even armed ones for the army.

None of these classes of devices at the moment offer much public verifiable information about their security level, which we choose to interpret as the fact that they are not secure. The author would happily review any document sent to him to demonstrate computer security requirements and functions embedded in one of these domain in order to remove the class from the list (and most certainly to provide a new section in the previous protection-oriented one). But for most cases, he is still waiting.

5.1 Wireless networks

IEEE 801.11a/b/g (most commonly called WiFi) was secured initially by an authentication and encryption protocol called WEP. Astonishingly, this protocol used RC4, a cipher with a known cryptographic weakness. The reason why RC4 was chosen in the first place is still to be given. The author suspects that it is for the same reason the Trojans brought the horse inside their walls. This design fault evidently gave an opportunity for real attack and tools to be proposed in practice as soon as the protocol was deployed in the field. WEP was soon deprecated and other authentication protocols were deployed successively[160]: WPA(TKIP), WPA2(CCMP) and EAP.

However, WEP remains an option in many available WiFi devices: retracting a widely deployed unsecure communication protocol *is* a challenge. So attacks against WEP are still an option if you can convince an unsuspecting user or an unskilled administrator to configure it. Finally, the author finds that the attack against WEP is interesting

158 No AI involved. Anyway, [Will1] would apply.

159 Which is absolutely not a reason at all to confuse serious security evaluation and childish blind random testing...

160 Yep. The first versions of the following protocols also had weaknesses. It seems it took some time to hire the right guys or do the right thing or both or something else.

from a teaching perspective. So let's look in more detail at the various steps involved into trying to get the key of a WiFi association secured by WEP.

5.2 (Not so) New generation avionics systems

Modern avionics system rely on common networking standards, adapted to the guaranteed transit time safety requirements of critical embedded systems. Using common networking standards allows to benefit from the speed and throughput of recent digital networking and (in theory) to use off the shelf equipment for testing or development of avionics equipment ; while specialized hardware components and protocols implementation adaption offer the added properties needed in the avionics domain.

This situation is summarized in the name given to this networking technology, AFDX, which stands for *Avionics Full DupleX switched ethernet*.

The transit time properties linked to real time and safety constraints are associated to all the network components existing on one communication paths: network interfaces (usually redundant) and network switches. These guarantees are not offered to all network traffic. For ensuring timing constraints, a *virtual link* (VL) must be reserved between two or more endpoints equipments, with one source equipment and one or more destination ones. Over such a link, the available communication slots and the maximum transit time of network packets are guaranteed for the endpoints involved.

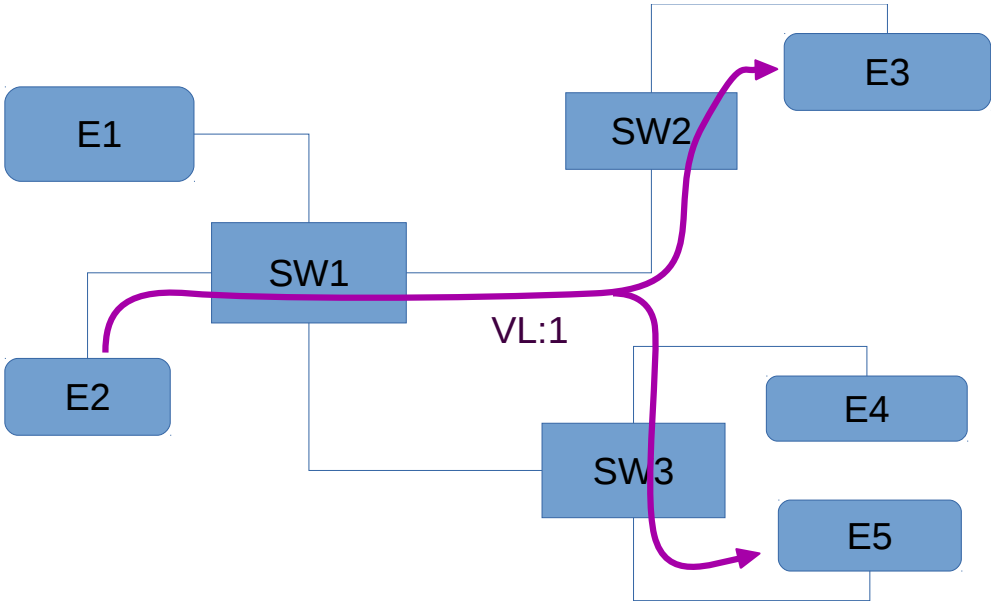


Figure 7: Virtual link definition in an AFDX switched network

Some details are available concerning the definition of the virtual links in an AFDX network, mostly summarized by everything statically configured. It is understandable how these guarantees are implemented inside the switches (for example using switching tables slots static configuration) and given the redundancy characteristics of endpoints AFDX interfaces. However, detailed information concerning the internal operation of those networking equipments is not readily available (publicly[161]).

In our view, most of the protection associated to security in an AFDX system is concentrated in the switching componenent (the AFDX switch). Up to our knowledge, little information is available on the security guarantees (possibly even less than that). Network filtering using Ethernet addresses seem to be commonly mentionned and it

161 Of course we do not know the proprietary information which could be available (esp. to certification authorities) and its proprietor of course does not want to share it publicly, certainly for fear of it then not being its property any more (though law is the real source of property, proprietors frequently exhibit low confidence in law for intellectual property, a sure sign they are thinking too much).

could be simply identical to the static preconfiguration of the virtual link circuits (though for fixed predetermined and non-reconfigurable[162] endpoints).

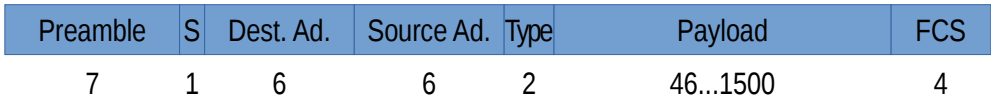


Figure 8: AFDX frame structure

The AFDX frame structure closely follows the one of Ethernet frames as in IEEE 802.3, with :

- a 7 octets physical preamble pattern of alternating 0 and 1 bits allowing devices on the network to synchronize clocks ;
- a 1 octet start of frame delimiter (SFD) marking the end of this synchronization and the beginning of the incoming frame by a double high (the 0xD5 constant, i.e. 10101011 in binary) ;
- a 6 octets destination MAC address ;
- a 6 octets source MAC address ;
- a 2 octets type or length field ;
- a variable length payload of data ;
- and 4 octets frame check sequence (FCS) which is a cyclic redundancy check (CRC32) computed over all the field (except the FCS)[163].

An end of frame indication is usually added on the physical layer (unless denoted by loss of carrier signal). Note the frame should be followed by a minimum of 12 octets of idle line state as an interpacket gap between packets.

AFDX addresses incorporate constant prefixes. The source address allows identification of the equipement.

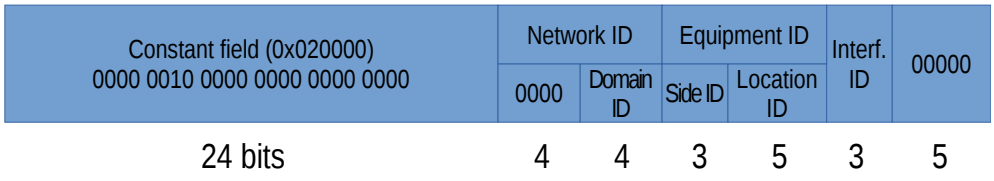


Figure 9: AFDX source address

The target address identifies the virtual link destination of the communication. A receiving equipment can potential handle several MAC destination and therefore be present on multiple virtual links

It seems to us AFDX communications are therefore always multicast communications.

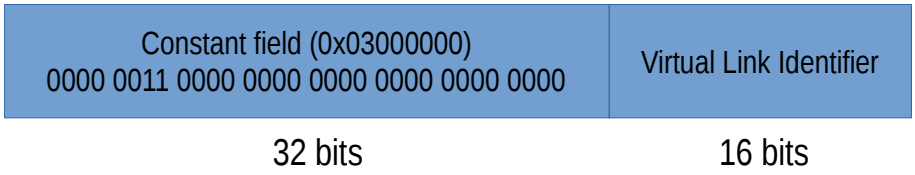


Figure 10: AFDX destination address

Note thus that, contrarily to 802.1Q extension (defining classical ground Ethernet VLANs), virtual link identifiers are encoded in a specific subclass of destination address and no additional frame field is inserted to identify them.

162 Hopefully. Dynamic MAC addresses are such a plague… They could even be used to improve privacy!

163 As the sender complements the FCS before sending it, the verification algorithm should fall on the magic number 0xC704DD7B as a CRC32 residue when encountering the FCS (for an error-free frame). When the type field is not the data length but a real type, this check importance may increase.

A similar logic of static adressing is used further inside the communication frames which imply that the IP source address is always a class A private IP unicast address (of the form 10.x.x.x) and the IP destination if either a multicast address (of the form 224.224.x.x) or unicast address (10.x.x.x). The last two options allow to make the distinction between a single or multiple client application in the destination end system.

These applications send and receive message through AFDX Ports which are basically remapped UDP ports. Two types of AFDX Comm Ports are defined (ARINC653): queuing ports (messages up to 8kiB with integrity guarantees) or sampling ports (inheriting the limitations of a single unreliable frame).

5.3 Network appliances

Network appliances are a common type of embedded systems among which you can find conventional LAN or WAN routers and switches, but also home-oriented ADSL boxes or radio-capable WiFi stations. All these devices share the common characteristic, inaugurated in the early 80s, that they provide network functionality, that they correspond to a specialized computers but a computer anyway and that most users absolutely do not want to hear about them, especially if seen as capable computers, because “they only want to use the network and they already have their own computing device”.

Such devices raises several observations with respect to their security. Most of these observation are based on a presentation from a hacker who dedicated a lot of time at one moment into analysis the security of one of theses devices class: Michael Lynn. Note first this presentation is not available any more for reference. Unfortunately, this specific example is also interesting to illustrate the manufacturer reaction to potential improvement of their devices security advertised through the prism of vulnerability discovery. (Such a reaction may or may not still be the case ; but we heavily suspect it is still pretty standard industry reaction to the discovery of software vulnerabilities they produce[164].) But back on technical observations we note that:

- Such appliances, like routers and switches, use off-the-shelf CPU to run their network-oriented software. Even though they are frequently presented like hardware only systems, they are not only made of networking-specific hardware (such hardware is even probably less and less specialized) and they incorporate a lot of standard electronics, even though sometimes less known to the public than the hardware of desktop computers.
- Consequently, these devices are programmed with common programming languages targetting general processors and they can exhibit classical software vulnerabilities, especially those affective low level programming languages like buffer overflows.
- Such vulnerabilities are exploitable. (Because an unexploitable vulnerability is not a vulnerability.) The impact of an successful exploitation may not be as widely reaching as we are used to on our too frequently fully compromised vulnerable destkop system, but its impact may be wide enough to be considered dangerous on the target device.
- And these exploits may be portable from one device to another. The devices hardware may be different which make this portability pretty difficult to achieve. This difficulty may sometime be compensated by the fact that successful appliances may also offer a very important number of exactly identical devices, due to mass production. If the numerous variant is vulnerable, the devices mass production may compensate for the variants hardware difference.
- Code quality on such platforms may be much better than on other platforms (esp. the below average desktop ones). For example, you may denote aspects like:
  - Internal integrity checks (memory heap, core data structures, etc.)
  - Overflow runtime checks
  - Seldom usage of the general processor stack (frequently originating in software development guidelines)
  - Forward recovery procedures intended to stop long term error propagation (hence the device deactivates itself or reboots to a clean state as soon as he detects an issue internally)
  - Very old code, very tested.

164 “*We know that the public-relations department handles their security vulnerabilities, and not the engineering department*.”, see: https://www.schneier.com/blog/archives/2005/07/cisco_harasses.html .

General characteristics of this kind of devices operating system also play a role in the analysis of their potential security properties.

In most case, they are operated by a monolithic operating system which means, for example, that there are no dynamic modules loaded in the kernel like on Linux for some device drivers classes. The kernel image is much more stable and all memory addresses are statically defined and they usually change from one build to another. The entire OS could be seen like a big fixed application program[165].

This software also frequently exhibits the same kind of characteristics as a realtime operating system, in the sense that as soon as you execute a program, you can hold control of the CPU (outside of interrupts or pretty minimal concurrent execution opportunities). For a malicious program, this implies specifically implies that as soon as code execution is achieved, you can take control of the whole device, but on the other hand, you had better exit and generally manage the device cleanly or you risk failing miserably and only achieve unscheduled device restart.

Because on this kind of common networking device, stability and long term unattended operation are valued over everything else, the OS would rather reboot and restart from clean state rather than try to correct errors. Especially on the Internet, short temporary failure could be tolerated by many of the networking protocols.

In the case described, the device software was also incorporating a heap integrity checking process that was constantly walking the memory allocator heap data structures in order to spot bad links that would reveal software memory corruptions. This process would execute every 30 or 60 seconds depending on the load of the platform and would make heap overflow vulnerability exploits very hard to design.

Defeating such a protection involved a lot of low level code disassembly in order to understand the software operation. This necessitates a lot of (human brain) time and energy[166]. The tricks achievable with these efforts offer little benefit but may allow to trigger some simple heap overflows (usually leading to short term reboot) or some pointers exchange. Building a more interesting attack over such small advances is challenging and first of all involved defeating some of the protections of the device software and most notably defeating the heap checking process. Achieving this was the primary improvement of the attack described and involved actually simulating a reboot in order to trigger the checking process deactivation and leave more time to complete a more interesting exploit (which, to our knowledge, was never detailed by the author).

The impact of such observations were certainly a lot of hard time for the people involved, mostly due to the manufacturer executives excessive and fully legal reaction. (Note that this reaction, in our opinion, primarily demonstrates the lack of maturity of these executives with respect to computer security which is certainly the most worrying concern after all and most probably has persisted after the whole affair in this company and in many other circles.)

Some manufacturers, especially the most well known oldest America one associated to a famous town of the west coast[167] every reader is probably already thinking about provides a lot of information on the web (after all they claim to have invented the Internet, too[168]). There are also interesting things about the security of their own code, providing alternative details about this section[169] in a company-specific view of course.

5.4 Mobile telephony

Nowadays, as smartphone more and more reveal themselves as full featured computers with innovate input/output devices (as opposed to traditional screens and keyboards), mobile telephony is becoming a complex subject with many connections to general purpose desktop computer systems management and many relationships with distributed computing systems security.

165 This can be an advantage or a drawback to mount some attacks or implement some protection, so nothing specifically qualitative with respect to security, but these characteristics must be taken into account.

166 While thinking about it, possibly the reason why only young or old men spend their time and energy on such a senseless activity and few of the intermediate but most active age?

167 *Hint*: with a bridge...

168 I frequently use that bio shortcut too with children.

169 See for example: https://www.cisco.com/c/en/us/about/security-center/integrity-assurance.html

But before turning you back to the full domain of (critical) distributed computing systems, we can glance back at the situation of mobile telephony per se and consider simply the no-so-smart communications devices that the first generations of phones were. They were already exhibiting interesting security issues that many of their much more technologically advanced successors have inherited (without showing much improvement on the security properties by the way).

At the end of the pure GSM era at the moment where mobile phone prototypes started to exhibit more advanced computing capabilities – but just before Apple debuted the smartphone phenomenon the technological landscape was pretty simple from the software point of view.

There was a major player, Nokia, with a single operating system ecosystem named Symbian which was covering most of the popular devices (from the clamshells of 2000 to the keyboard and wheels of 2005). Microsoft was claiming that it would invade this area with a variant of Windows (called Windows CE). And a bunch of open-source (or at least as open source as possible given the protection of some pieces of hardware) variants were appearing and disappearing, with Linux as a central piece: Qtopia (TrollTech), Android (Google and Motorola), OpenMoko/OpenEmbedded (FIC & several individuals).

And the everything changed[170]. Apple minds introduced the iPhone which quickly evolved all these primitive phones into real computers (which identity was initially disguised as smartphones). Google put all its weight on the Android platform which became the dominant OS in terms of numbers (whether this is due to Linux openness, Google weight, later smartphones hardware manufacturers neatness or some other magic is left open for debate to the readers).

We are going a little fast. The transition took roughly a decade (of frantic devices race) only saddened once by the passing of the major inventor.

Through this transition, from the point of view of computer security, several aspects need further details.

The first one (roughly in chronological order, but certainly not in order of importance) is associated to the telecommunication network. With this technology, we are considering digital cellular networks, second generation (2G) also generally known as GSM. The security of such networks is really worth considering given the importance of the telecommunication infrastructures. Considering later generations (2.5G, 3G, 4G, etc.) is difficult due to the lack of documentation of these later versions – even more opaque than GSM was. But we can suppose that the security design faults introduced in GSM could be more or less similar to those of later designs.

The second one is linked to the security use cases considered in the design of these smartphone devices and/or the device from the telecommunication point of view. Several interesting use cases are available (secure software download, secure channels, platform integrity, etc.) and we see at least one use case probably missing which is interesting to analyse too (an end user available privacy management system).

Finally, with several years of backlog information, it is also possible to study security state of smartphone software, as an example of embedded systems security. Especially Android is more documented on its security features, potential vulnerabilities, knows (and exploited) weaknesses and latest innovations (even with respect to general purpose computers or security-oriented architectures). Apple iOS is nearly unknown to the author so he will not comment on its security mechanisms, though it is known from the outside that this security has already raised several pretty interesting concerns (pretty favourable to trust on the technical mechanisms, even if in practice, governmental players were pretty annoyed considering them).

5.4.1 GSM security

GSM security sounds like an old issue to young wannabes. They frequently fail to realize that the SIM card they use on their brand new smartphone and the core protocols implemented by the “broadband CPU” hidden in it are precisely from that era. Of course, they were complemented by additional mechanisms when newer generations of telecommunication systems were deployed but :

- First we cannot say much of the latter because they are even hidden under even more secrecy than the previous one : GSM was initially pretty well protected behind intellectual property heavy curtains but then, metaphorically, its successors have been hidden in underground bunkers… The next generation of professors and students will still probably be looking back at supposedly obsolete technology...
- Some things were eventually found on GSM. We can try to extrapolate on the later generations ; though I can already warn you about the only possible extrapolation: obscurantists are not good at security.

170 Possibly with the exception of Microsoft mobile computers world domination plan which stays as is.

- Anyway, most of the end user devices or communication towers are still compatible with GSM, so [please fill the blank]...

GSM security was thought to be pretty good at some point in the past. So good that public authorities and legal enforcement administrations were worried about the impossibility to eavesdrop on users and even the author of these lines initially said nice things about it. But the fog finally settled and evil details emerged.

First of all, it became clear that industrial grade cryptography cannot really compete with academic crypto. Not only pretentious engineers and marketing experts are probably overpaid but their ciphers frequently fall short of mathematical analysis by bearded experienced professors ironed by years of fightings with innumerable students. The A5 family of ciphers protecting GSM started to exhibit weaknesses months after initially leaked from the "secret specifications", most infamously A5/1 (or even A5/2 which was a deliberate weakening for certain export regions). Now, A5/3 is also reported to exhibit weaknesses. Even if they are probably not applicable in the field, they were used to motivate evolution. However, the A5 family is still the way of the standard. In short, GSM crypto is showing its age, has not evolved much (industrials apparently do not care any more) and anyway, other attacks have been found independently of the cipher. Next point.

Secondly, most of the design ideas of GSM security focus on protecting the (nice generous big telco owned) network infrastructure from the (evil mean teenager customer) mobile device potential disruptions. Hence the security protocols really do concentrate on authenticating the device, checking its integrity and technical conformance from the point of view of the telecommunication and making sure the billing target is identified (hence the separate "broadband" CPU dedicated to that and the associated smart card issued by the networking operator). The authentication of the device to the network is pretty good. The segregation between networks too ; even if commercial cooperation is, of course possible[171]. On the other hand, the network does not really authenticate to the phone and phones are preconfigured to accept many network offerings (whether good or not so good commercial cooperation exist in the surroundings). The network infrastructure itself too can sometimes tolerate lack of confidentiality (such as direct radio transmission between antennas) if the networking operate sees it fit (its own interest).

The most straightforward way for attacking a communication is therefore to impersonate the network and use one's own radio equipment to monitor the phones surrounding a specific area. Many legal enforcement authorities have understood the issue and also asked the telco to give them a few base stations (as an addition to their normal mandatory cooperation for legal prosecution). They probably do not like the idea of a "rogue station" so much now however as many police officers do use similar devices too for their own communications during operations. (And, obviously, the bad guys too may have the idea.)

Finally, some open source projects finally made progress on open implementation of this protocol. We hope that this initiative may, in the end, be fruitful as a whole. Because security improvements would be so much easier on an open implementation[172]. But the current observation is that with a software radio device and this software, a DIY SMS jamming antenna bill of material budget has gone below the $7000 value several years ago – for a whole 30km radius cell. And we say jamming because we do not want to alert the monitoring probes[123]. We could probably use a microwave oven too for jamming only.

5.4.2 Mobile phone security use cases

Another interesting aspect of mobile phone security is associated to the various security uses cases that have been identified in the literature and implemented via the usual mechanisms known in common cellular networks. For example, the trusted computing group behind the TPM standard provided some examples associated with this environment to demonstrate how the TPM chipset could be used in this context.

These uses cases demonstrate interesting examples of how security properties should be identified to be later specified in detail and then implemented. They also demonstrate how such properties can be difficult to advertise (especially from a marketing point of view) because, frequently, one's security objective do not fit well with those of others.

171 Between the telecommunication companies (of course).

172 Some people may even have ideas on the overall improvement of the GSM technology. Better ideas than incrementing the "*N*G" number...

Especially in the case of private companies, the basic focus on ((board and executives) revenue or) business protection is frequently omitted from the publicized security policy. Whether we have to resort to non profit organizations for implementing computer security policies is left as a reflection topic to the reader – in the meantime we will analyse the situation of a cellular telecommunication network and illustrate some of the security properties that the various stakeholders will desire.

A classical objective in security is to ensure platform integrity. The use case is clear in the case of a mobile device like a smartphone: both the manufacturer, the telecommunication operator and the owner would like to be sure that the smartphone has not been hacked by a bearded hacker, a foreign spy or an evil stepmother. Less obvious is the fact that neither the manufacturer nor the telecommunication operator trust the owner of the smartphone and would like to hold technical integrity guarantees on the device even if, legally speaking, they do not own it any more.

Fortunately, modern software and hardware offer the ability to keep technical control on the smart device and carefully limit the capabilities of the average smartphone user with respect to device personalization. From the point of view of an engineer that loves tinkering with the latest electronics, this is very bad ; but one has to admit that, especially after having seen a few disturbances generated by misconfigured equipment, one understands the basic concern of preventing a radio emitter to operate fully randomly. However, these safety (and to some extent security) guarantees have been rather narrowly interpreted by the mobile phone industry.

In practice, the platform integrity use case consists in ensuring that devices possess and run only authorized operating systems and hardware, as seen from the manufacturer, the network operator and the government of the country where the device is located. Initially focussed on the “broadband CPU” controlling the radio part of the device (and possibly an explanation of its separation form the rest of the hardware) and the software associated to network registration, call establishment and management, this integrity objective has been extended to the entire OS and most of the hardware (if only to check the integrity of the OS and its initialization firmware and loader). This control on the smartphone OS allows the software/hardware manufacturer to maintain a pretty strict authority on the applications that can be run on any given OS and usually maintain a monopoly on the application market available to users for purchasing said applications.

This control is seen as necessary in order to guarantee several things: the protection of the content eventually sent on the device (especially audio or video) against uncontrolled copying, the protection of the owner (via controlled application download) and the protection of the networking infrastructure against abnormal device operation.

Another more specific use case is associated to device authentication. Such an authentication is usually coupled with user authentication: the device acting on behalf of the user for authentication[173]. The device usually hold the authentication keys that allow it to enter the network. To prevent tampering as well as to separate the key generation process from the device manufacturing, mobile phone technology took the smart card road pretty early, with SIM cards associated to phones in order to perform an authentication protocol with the network relying on asymmetric cryptography.

The entities doing this authentication are the SIM card representing the subscriber on the one hand and some stations on the network on the other hand. Only the user (the SIM) is challenged for authentication so the actual use case is really authentication of the device to the network (and not mutual authentication between the two).

A now mostly obsolete security requirement which used to be associated with the SIM-phone security association was device personalisation. The objective in this use case was that the device remained locked to a particular network (under the logic that the phone terminal was subsidized by the operator and rent to the end user). Later legal evolutions obliged the operators to limit in time this renting constraint and unlock the phone for use with alternative operator. Economical reasons then quickly lead to market structure change where bought phone were the norm and SIMlocking of phones a thing of past.

173 For the public mobile network access control, this is deemed sufficient, as well as for user billing so the device is directly responsible for most of the trust given to the technical authentication process. The authentication of the user to the device remains pretty basic. Of course, this raises problems for application where strong user authentication would be desirable too (like mobile payment, ID cards, ²etc.) but note this was not part of the initial use case...

In practice, the use case was easy to implement via the hierarchical structure of the public-key certificates installed in the SIMs: for example as specific network only needed to check the certification authority associated to some certificate in order to verify that the device was on his home network (or on a network with the appropriate business agreement[174]).

SIMunlocking was also pretty interesting technically, though much more difficult to teach due to the legal restrictions on reverse engineering this kind of heavily business-encumbered technology. Let's say simply that the secure channel between some pieces of the device and some pieces of the network hardware could be used to update the rest of the device software to prevent it from remaining locked. Fortunately, both things design were *old* enough to rely on classic security architecture principles with an immutable and small security kernel allowing to do just that, securely but also simply reliably.

Complementing authentication another use-case that could be associated to mobile smartphone is the one of a strong DRM implementation (Digital Rights Management). The objective here is usually to protect the rights of one owner of a digital media (the producer of an audio or video) distributed through the digital platform while still allowing the end user to use the media in a strictly controlled manner (especially while preventing any form of digital copy) – the latter usage still being necessary to collect a fee from users in the first place[175]. This use case is pretty interesting technically due to it involving control of the manipulation of data, like in the case of multilevel mandatory policies. Outside of the technical interest, we find the use case a little idealistic, neglecting many of the users participating in the process (authors, actors, interpreters, parents, etc.). Of course, ideally, creation would be rewarded fairly. Maybe there is still hope. But on the technical point of view, we are still looking at DRM things.

Device authentication and some of principles underlying digital rights management[176] allow to implement secure software download functions. Such secure software deployment capabilities were immediately relied upon by the clever software industry in order to create application markets allowing them simultaneously to grow and to maintain control[177] on the business associated to the smartphone ecosystem. Security of software download most probably operated primarily to protect these markets business, quality of the application software and security of user data being varying possible side products of this protection (extremely varying concerning the second case).

The downside of these secure software download systems is the inability for most software developers to tinker with their mobile phones easily in order to deploy home-made applications. Motivated hackers may need to register to foreign electronic signature infrastructure system or circumvent software locking preventing to use one's phone in a normal fashion, or both, in order to program their device. They frequently think afterwards that they do not own the device so much after all.

Old fashioned and still nearly never deployed alternative use cases were also studied in association with mobile phones and security modules (like SIM or TPM). Mobile ticketing and mobile payment were such ideas. Admittedly, such ideas have been associated with all the proposed hardware security modules for four decades, so they are not so original to grey haired players. But still, they always sound like good ideas : use your mobile phone like a crypto credit card to debit your account or even use it to store some protected cryptographic voucher that could be used to actually activate a service (like boarding a train or a plane). The author humbly recognizes that it has never understood why such usages have not yet been widespread. He knowns now however that business issues prevent such successes. That's the advantage of 30 years of technical experience and the ability to list several alternative already existing technical solutions for such an idea: you start to guess when a problem is not a technical problem after all. Only young engineers think that a better technical solution is required[178].

174 Or any alternative way of proving that the user had agreed to pay expensive phone bills for the pleasure to show off at remote locations.

175 Note alternative models have been proposed : e.g. everyone paying some of the money to use a hard disk (whatever the thing stored). Astonishing, some people objected too.

176 i.e. cryptographic signature algorithms and public-key infrastructure protocols.

177 And to tax I suppose.

178 An interesting parallel (and direct testimony) can be made with digital music. As a teenager in the 80s, the author used analogue tapes and old vinyl records because digital music was not available. As a young engineer one decade later, he believed that working on better sound or video compression algorithms to break through the needed improvements was key to bringing digital music to the masses. Only in the 2010s did he realize that he had nearly been taught at school the audio compression

Finally, a few use cases arise easily symmetrically when looking back at the ones we have just browsed.

We could imaging the platform and application integrity could also be attested to the end user when he or she wants to know that a device or an application can be trusted (or at least trusted to still be unspoiled). Such attestation capabilities could be interestingly extended in the direction of theft prevention or remediation.

We could also envisage security mechanisms to be available to the end user in order to improve user data protection and privacy: not only in order to conceal and protect personally identifiable information, but also to regulate the distribution of such information[179].

Advanced developers and security researchers would also be pretty happy to use these security kernels in order to play and develop more advanced security models (like those presented in other chapters for example).

Amusingly, little of these use cases oriented towards privacy and system security have seen much advances (with the possible exception of SEAndroid to confirm the fact that there are always exceptions).

5.4.3 Android-oriented issues

Among the security characteristics of Android, one of the most popular smartphone operating system, we can distinguish a few general orientations. The last one is now a little deprecated as it got several upgrades in the direction of more granular control over security authorizations by users in the latest versions of Android.

- The system control lines are implemented via a Linux kernel-enforced sandboxing.
  - This means that a single applications will have a lot of "permissions" to request in order to operate normally (and conversely that the user will be able to refuse a lot of authorizations if he does not like them – obviously, said users are fond of browsing individuals permission lists[180]).
- Software security is achieved via application signing. You may get default permissions depending on the signature level (in practice, Google-originated apps may get a little more permissions – especially if they are part of the OS – for alternate distributors, we guess your mileage may vary).
- Good old Unix *user identifiers* and *file access rights* are used to implement access control.
  - It means that 2 installed applications are given 2 different user id UID[181].
  - And then, fortunately or unfortunately depending on your mood, there is "shareUserID" to share… more.
- The declaration and enforcement of permissions requests or declaration is done per application via a single file distributed with application packages
  - The androidManifest.xml file, which is starting to get pretty infamous because it is sometimes seen as growing without control.
  - The detractors of the approach should admit that it exists and actually corresponds to the rights granted. They had better criticize its actual readability, or the fact that many applications were taking advantage of its operation to mandate users towards full grant of the whole set of requested permissions.
  - Latest evolutions could allow for better granularity control by granting permissions progressively (on demand) which will prevent developers from extorting all permissions from users at once at install time. But we still would like to see more real world examples of actual added privacy thanks to this logic change of the underlying OS.

algorithm initially standardized in the 80s that are currently being used in most of available music stores. Maybe the obstacle to him using digital music as a child was not very technical after all… (Anyway, the 80s rocked !)

179 Because a phone is a communication device after all, so we suppose that if the user owns a mobile one, it's probably in order to communicate, not only to use it as a data safe.

180 As a matter of fact, neither developers: usually refusing a single permission make the entire application bail out.

181 In some sense, it means Android is not really multi-user any more?

5.5 Gaming devices

Many gaming devices exhibit more interesting security properties than most industrial control systems. Generally speaking, this is pretty worrying by the way. Fortunately for this course, much more information is available on gaming devices weaknesses and strengths than on industrial critical systems. Manufacturers apparently always took some pride into explaining (possibly to the video game industry) how they can prevent piracy and secure the revenue stream. Some hackers and players alike have always taken as part of the game some tentative cheating activity[182]. Game consoles manufacturers may even be suspected of having encouraged some of these capabilities in order to gain more audience in the first periods of a new device life-cycle.

5.5.1 Xbox and Linux

The Xbox used a hardware modification and a digital signature system to prevent the public from running unsigned code. The interest in the hardware and the project to port the Linux operating system to the console fuelled some analysis of these security mechanisms as, of course, porting a full in-development operating system necessitates the ability to run arbitrary code.

Initially, one had to use a modchip to launch homebrew programs unsigned by the manufacturer. A modchip is a hardware modification, an independent chipset that operate by replacing or overriding some of the original hardware in order to circumvent some protection measures.

Later, it was demonstrated that it was possible to reflash the original Xbox BIOS storage memory chipset in order to install a custom one. The “Cromwell” BIOS of is unusual in the sense that it was entirely developed through reverse-engineering of the original BIOS and the boot process of the console. Its main function is to load the Linux operating system (and it is not able to load Xbox games, whether originals or copies). If flashed to the original flashrom chipset, it would then entirely repurpose the hardware to be used as a Linux based system (and not the original gaming console).

The effort needed for such a reprogramming of the original BIOS is obviously important. In the case of the Xbox it was motivated by passion of the hackers involved as well as the rather wide availability and generality of the hardware ; however, it would obviously have been faster to perform with internal information from the manufacturer[183]. But the key aspect from the technical point of view is that this is perfectly feasible. The protection of the original Xbox was only relying on firmware-based signature.

If we look in more detail at the boot operation mechanisms of the original Xbox, we find that the initial start code at CPU initialization involves a 512 bytes area which is stored in a hidden ROM area in the SouthBridge of the platform, instead of the initial area of the regular flash ram of the motherboard. This small area was specifically added in order to protect the firmware of the console from tampering via alteration of the (conventional) flash memory, while still allowing such a memory type to be used (instead of more expensive ROM for example) and programmed at the factory via standard procedures (involving simple bus override).

The firmware itself store in the conventional 1MB flash memory was not raw assembler code directly executable, in order to limit its analysis. In the “secret ROM” area was implemented a small interpreter as well as a RC4 algorithm. The interpreter was executing a pretty simple virtual machine allowing to use a few read and write instructions, access the PCI configuration space, do a few logical operations and conditional jumps and has one internal register (accumulator). The virtual machine offers a one byte instruction set with two 32 bits operands and can use immediate values (nicknamed the ‘xcodes’). This simple virtual machine executed the code stored in the regular flash memory, which was then doing complex RAM initialization, and loading of the second stage bootloader (actually the final operating kernel).

182 Some even argued that said cheaters may find satisfaction too in the punishment associated to discovery of these abuses. At least, they have given the opportunity of a pretty naughty footnote. This course really is for the older students even if it finishes with video games.

183 In this case, the resulting firmware would obviously have been produced illegally from the point of view of the manufacturer’s lawyers. But who does not remember at the end of this text that some attackers do not follow the law?

Furthermore, this firmware was encrypted in the flash memory. The code of the secret ROM was also verifying the integrity of the kernel stored in this firmware and decrypting it (using its RC4 algorithm).

Additional technical measures were also taken to try to improve the confidentiality of the code existing in this hidden ROM and to prevent disruption of the execution of its virtual machine until the kernel stored in the firmware was activated or in case of loading failure (either to limit firmware tampering or to slow down reverse engineering attempts).

These protection measures failed on several aspects. First, for example, a copy of an old version of the small secret ROM was actually deployed in the field in the first 512 bytes of the flash memory firmware. Its presence allowed the hackers studying the flash memory content to understand that the rest of the firmware was actually virtual codes and that this part of the software was not at all the one executing for real at the console power up[184]. Finding the actual content of the hidden ROM proved more difficult (sniffing on a high speed HyperTransport bus connecting the CPU and the SouthBridge) but was done pretty using specific hardware skills. And then, the analysis of the secret ROM revealed at least a design fault: the usage of the RC4 encryption algorithm for hashing is not valid and could be exploited to provide alternative firmwares that would still pass the signature check. So modchips appeared. Some of them had complete replacement flash memory chips on them, others only patched a few bytes while passing most reads down to the original flash.

Another popular way of installing Linux on the Xbox was through the use of a few software bugs, buffer overflows found in a few games. Theses games were relying on some form of executable save files (a pretty astonishing idea in the first place by the way) and the method for taking control of the console CPU was based on loading a hacked save file transferred to the hard drive with these games. The save file was actually containing a Linux kernel and, as the console hardware was globally similar to most desktop PC of the era, that kernel was able to reinitialize the system in order to switch to a different purpose.

This is typical of centralized signature based software control schemes. The central authority signing the authorization for running software should obviously too validate the actual operation of the software to prevent further exploitation or delegation. When said validation is too minimal (or only based on business agreements), the signing authority obviously imports all the vulnerabilities of its subsidiaries. In the long term, such a strategy is probably inevitably leading to a rather serious security degradation of the whole platform. In the case of the gaming industry, such a limited time frame of reasonable protection may be acceptable ; but the whole strategy may not be so easily extendible to critical systems[185].

In the case of the Xbox, the games allowing to circumvent the protection also simply allowed to overwrite the original firmware without having to resort to hardware modifications. This would of course repurpose the console entirely (while later on, less ethical pirates found ways to retarget these firmware modification for piracy).

The console manufacturer reacted several times during the lifecycle of its device. It changed the encryption algorithm to another variant (TEA). It finally used a real ROM to prevent flash memory tampering. It fixed a few of the shortages that had been found. However, overall, from our point of view and those of most of the hackers involved in the device analysis, it did not react very wisely. It kept on trying to use some of the least efficient methods for computer protection (and consequently failed several times repeating similar mistakes, like relying on obscurity, setting up mere obstacles or delegating without verifying) and consistently refused to cooperate with its users – even those that were demonstrating such a thorough understanding of its gaming hardware.

Some of the recommendations made by those who broke the Xbox security system are still very interesting to study [MIST2005] and could be very beneficial to many of those currently making similar recommendations. Overall, it seems the console engineers primarily tried to out-hack hackers. Obviously, they forgot there is always someone (much) smarter than yourself – and that playing tricks has nothing to do with serious security systems. Fortunately, it was just for gaming and provided interesting challenges. Hopefully these engineers stayed in the gaming industry.

184 Simply because changing it did not prevent the normal start-up.

185 Ahem. Guess what they did then ?

5.5.2 Wii

"Homebrew" is often considered piracy's cousin, which is why every major console is full of security measures to lock users out of running unauthorized code on them. [186]

Most games provided for the Nintendo Wii were exhibiting encryption with AES in CBC mode using a disc-specific key, itself stored on the disc encrypted with a master key. The disc content is also signed with hierarchical SHA-1 hashes finally encrypted with RSA-2048. Another place to look for protection, especially given the preceding section, are game saves: they were signed using SHA-1 signed by an ECC algorithm. (This signature key is specific to one console, but the public key is distributed with the saved file. This console specific public key is signed by a manufacturer key which public part is available to all consoles.)

Nintendo clearly was not going to make it easy to brew home made programs for its console.

However, some progresses were made on keys identification. First, the master AES key could be extracted from one console using a temporary hardware fault (a short on memory chips) as well as a dedicated program running in legacy compatibility mode that was able to read the system private encryption key.

Similarly, extracting one private ECC key from one console allowed to create save files usable on all consoles.

However, faster progress was made when someone looked at the core function doing the RSA and SHA1 verification tests via a disassembler. This checking function looked like the following pseudo-code:

```
bool is_valid_signature(byte signature[256], byte public_key[256], byte
content_sha1[256]) {
    byte decrypted_signature[256];
    decrypt_rsa(signature, public_key, decrypted_signature);
    if(strncmp(content_sha1, decrypted_signature + 236, 20) == 0)
        return 1;
    else
        return 0;
}
```

This verification check was used in all security-critical points in the OS. Apparently, their verification signature was not fully checked. Furthermore, they were using the strncmp() function, which shows a direct problem here best seen for example as opposed to the memcmp() one. strcmp() treats memory areas as strings, hence terminates its comparison when reaching end of strings (which corresponds to byte 0x00). All it took to bypass the check was then to analyze a few valid existing signatures (e.g. on games discs) to find one with early zero byte, and then brute force the preceding bytes by exhaustive search in order to build programs that would have such a “zero beginning hash” which would be seen by the earlier check like zero SHA signature.

Furthermore, given the operation of RSA, zero valued signatures are encrypted as zero SHA hash too.

This allowed to create custom code that would be allowed to run on the console system.

It allowed enthusiasts to created a dedicated Wii “homebrew channel” to distribute custom apps (which included many old fashioned emulations). From the security point of view, it also shows that all the security checks were done at boot or install time. Once deployed to the system, an application was considered trusted (enough, to run as an application only).

The manufacturer reacted to these modifications by forcing firmwares updates on the consoles, though pretty lazily (as opposed to some general purpose OS vendors) and clearly prioritizing system stability. Over several of the known weaknesses of the console, only a few were deliberately closed in order to disrupt hacked configurations, especially piracy (but also the fan-powered channel).

186 https://thedailywtf.com/articles/Anatomii-of-a-Hack

But some alternate exploits were still working: those involving modified save games. The most famous involved one bug of the most famous title on the platform, *The Legend of Zelda: Twilight Princess*. The bug was a classic stack overflow on some of the strings in the game save file, including the player's name or his horse's name. This is the classic "*Twilight Hack*" of the Wii. Specifically crafted saved files were made available for using (with the original disc game title) that would launch another executable (possibly setting up a permanent channel).

Another similar problem is known on a game involving *Lego* characters.

Another one is known on *Super Smash Bros. Brawl*, working a little differently as it exploits the way the game program loads custom stages (from an external SD card) instead of save files. For obvious reasons, that exploit is nicknamed *Brawl Smash Stack*[187].

The manufacturer tried to patch the first problem by including a small modification that was verifying if the save file length was a multiple of 32. This was the case for the legitimate save files of LoZ. In case of detection, the update was trying to delete the target files and also prevent their further transfer from SD cards. Detailed analysis of the added code revealed several possible further workarounds outside of the obvious one, distributed a few days after the update.

Similarly to the Xbox case, this technical configuration raised a challenge to many users who took it as an opportunity for analysis, experimentation and some out of the roads programming. Still, even given their known limitations, such a system exhibit interesting security mechanisms targeted at their narrow economical protection objective ; especially when contrasted with some other systems.

5.5.3 Concluding notes and perspectives

More recent examples are left for future studies ; when they be old enough to respect the original business interest of their respective manufacturers. These are just gaming platform after all.

Furthermore, nowadays, the online gaming business has somewhat weakened our technical interest in the topic. Because, through enforcement of regular checks of software integrity and account validity by online operations, modern game platforms have more or less solved their security issues with respect to piracy – so nowadays the technical topics on this part are much less interesting[188].

So we will turn to mid-2000s era gaming devices, which have ended their cycle and present us most of the interesting information fully (including weaknesses and exploits, and even without the associated hype problems).

Prospectively, it remains to us the rhetoric about why the *gaming* software industry reached its security target so fast while some other big industrial players seem still stuck in their risk analysis endless debates. Such debates can be pretty entertaining sometimes, especially when the discussion approaches the engineering team technical skills and knowledge and you look at it from the outsied ; but it usually falls short to authoritarian pressures to hide everything under the carpet as soon as you approach very serious people circles. So the interesting technical discussion is limited to activists circles which frequently fall short on raw money fuel…

Fortunately, the online gaming industry and its players have *also* developed alternative abuse scenarios which could lead to our renewed interest for technical topics in computer security. Behavioural intrusion detection of software abuses, real money trading via in game currency and things like this will once again appeal to us as soon as the author will have enough time to dig the topic and structure the next part of this course.

187 Though the author still thinks *Smash Hack* rocks.

188 Note that full privacy protection may not really have ever been in the security target of the computer online gaming business. This is something that even some legal decisions seem to confirm as only minimal security measures seem to be expected from such businesses by legal enforcement authorities. And, for sure, they do more than minimal (for their own sake). And even if they were convicted, the entertainment industry could certainly appeal to unfair treatment in comparison of much more critical and certainly as wealthy *other* industries, services or administrations that potentially do much less than them to protect people (not only their privacy). Privacy issues are probably only the tip of the iceberg.

In the meantime, the overall conclusion for the prospective student is, in itself, awfully stunning: from many point of view, outside of the academic domain, some of the best technical job opportunities in computer security really, really shows around WoW, GW2, EVE Online, and co. Unfortunately, if those considering computers from the entertainment point of view are more serious at computer security than those trying to manage them in the rest of the engineering community, the implications are not entertaining at all.

Références bibliographiques

[Anderson2008] Ross Anderson, *Security Engineering*, 2008.

[avizienis2004] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, 2004.

[Biba75] K. J. Biba, Integrity Considerations for Secure Computer Systems, 1975.

[Bieber92] Pierre Bieber and Frédéric Cuppens, *A Logical View of Secure Dependencies*, 1992.

[Bishop2003] Matt Bishop, *Computer Security: Art and Science*, .

[BLP75] Bell, LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, 1975.

[Clark&Wilson87] D. Clark, D. Wilson, *A Comparison of Commercial and Military Computer Security Policies*, 1987.

[Dacier93] Marc Dacier, *A Petri Net Representation of the Take-Grant Model*, 1993.

[DPA99] Paul Kocher, Joshua Jaffe, Benjamin Jun, *Dierential Power Analysis*, 1999.

[Gar2011] Simson Garfinkel, Alan Schwartz, Gene Spafford, *Practical UNIX and Internet Security*, 2011.

[Goguen82] J. Goguen, J. Meseguer, *Security Policies and Security Models*, 1982.

[GPPTY2016] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, Yuval Yarom, *ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels*, 2016.

[GST2014] Daniel Genkin, Adi Shamir, Eran Tromer, *RSA key extraction via low-bandwidth acoustic cryptanalysis*, 2014.

[Gutmann96] Peter Gutmann, *Secure Deletion of Data from Magnetic and Solid-State Memory*, 1996.

[Haigney2017] Sophie Haigney, *Author's Hard Drive is Steamrolled*, 2017.

[Handbook1996] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone , *Handbook of Applied Cryptography*, 1996.

[Hergé56] Hergé, *L'Affaire Tournesol*, 1956.

[HRU76] M. A. Harrison, W. L. Ruzzo, J. D. Ullman, *Protection in Operating Systems*, 1976.

[ITSEC91] CEE, *Critères d'évaluation de la sécurité des systèmes informatiques*, 1991.

[ITSEM93] CEE, *Manuel d'évaluation de la sécurité des systèmes informatiques*, 1993.

[Jones76] A. K. Jones, R. J. Lipton, L. Snyder, *A Linear Time Algorithm for deciding Security*, 1976.

[Kocher96] Paul Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, 1996.

[KSWH98] John Kelsey, Bruce Schneier, David Wagner, Chris Hall, *Side Channel Cryptanalysis of Product Ciphers*, .

[Lampson71] B. Lampson, *Protection*, 1971.

[Levy96] Aleph One, *Smashing The Stack For Fun And Profit*, 1996.

[McCullough87] 1987, *Specifications for Multi-Level Security and a Hook-Up Property*, .

[McCullough90] D. McCullough, *A Hookup Theorem for Multilevel Security*, 1990.

[McLean94] John McLean, *Security Models*, 1994.

[MFCLWS2009] Bertrand Meyer, Arno Fiva, Ilinca Ciupa, Andreas Leitner, Yi Wei, Emmanuel Stapf, *Programs That Test Themselves*, 2009.

[MMEBA2008] Nancy R. Mead, Gary McGraw, Robert J. Ellison, Sean Barnum, Julia H. Allen, *Software Security Engineering: A Guide for Project Managers*, 2008.

[NIST80057] Elaine Baker, *Recommendation for Key Management*, 2016.

[Oppliger2011] Rolf Oppliger, *Contemporary Cryptography*, 2011.

[Pfleeger2015] Charles P. Pfleeger, Shari L. Pfleeger and Jonathan Margulies, *Security in Computing*, 2015.

[Sandhu88] Ravi S. Sandhu, *The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuation Schemes*, 1988.

[Schneier93] Bruce Schneier, *Applied Cryptography*, 1996.

[Snyder81] L. Snyder, *Theft and Conspiracy in the Take-Grant Model*, 1981.

[Spaff03] Eugene H. Spafford, A failure to learn from the past, 2003.

[Sutherland86] D. Sutherland, *A Model of Information*, 1986.

[TG77] R. J. Lipton and L. Snyder, *A Linear Time Algorithm for Deciding Subject Security*, 1977.

[Thomson84] Ken Thomson, *Reflections on Trusting Trust*, 1984.

[TPM2007] 2007, *TPM Main Part 1 Design Principles, Specification*, .

[Will1] William Shakespeare, *As You Like It*, 1623.

Index

TBD