Cybersecurity must come to IT systems now

Olivier Zendra, Bart Coppens

Many IT systems still lack (cyber)security. We have a lot to lose from having poorly secured IT systems, and a lot to gain from secure ones. The good news is: we can do it, in Europe. Here is how.

Cybersecurity must come to IT systems now

By OLIVIER ZENDRA and BART COPPENS

After decades of apparently low-intensity cyber attacks, during which security was not really thought of on most IT systems, recent years have brought a flurry of well-organized, larger-scale attacks that have caused billions of Euros of damage. This was made possible by the plethora of IT systems that have been produced with no or low security, a trend that has further increased with the rise of ubiquitous computing, with smartphones, IoT and smart-* being everywhere with extremely low control.

However, although the current situation in IT systems can still be considered as critical and very much working in favour of cyber attackers, there are definitely paths to massive but doable technical improvements that can lead us to a much more secure and sovereign IT ecosystem, along with strong business opportunities in Europe.

Key insights

- Cyber attacks are ever increasing, and the cost of damage caused by lack of cybersecurity is soaring.
- Security must now become a first-class citizen of new programs from day one, in both specifications and source code.
- However, legacy does not go away: tools that are able to analyze the legacy code base, and find vulnerabilities and unwanted behaviour are needed, as are those that automatically circumvent or mitigate them.
- Automated diversity is a means of resilience against security attacks and of safety.
- Liability in IT systems is crucial to the advance of security aspects. Security certification must become mandatory. Certification and liability reinforce one another. Regulation must play its role.
- Cybersecurity, hence IT sovereignty, which drastically impacts political sovereignty, depends on having control over hardware and software.

Key recommendations

- Promote that research and industry have security as a first-class citizen of new IT systems, both in specifications and in source code.
- Promote research on methods and tools to find vulnerabilities in existing code, as well as tools to automatically prevent or mitigate them.
- Mandate security certification for IT systems whose malfunctioning would impact a large number of citizens, in the same ways as certification for critical systems.
- Regulate to make IT systems providers and resellers liable.
- To reclaim IT sovereignty, base the critical parts for cybersecurity of EU IT systems either on open-source software and hardware, or on EU-made, trustable because audited, proprietary hardware or software.

CYBERSECURITY MUST COME TO IT SYSTEMS NOW

Barely a few years ago, cybersecurity was, if not unheard of, at least not on the minds of many people and leaders. IT systems seemed to be working, attacks seemed to target only "others"; in short, cybersecurity was a non-pressing matter, hence often non-existent… Since then, cyber attacks have made headlines: influence on the 2016 US Presidential elections; Petya ransomware attacks in 2016-2017 with losses estimated to US$10 billion [8], and Wannacry ransomware attacks in 2017 leading to losses estimated to up to US$4 billion [9].

However, if awareness has indeed risen, it is still true that most of the world, including Europe, has not yet fully awakened to the cybersecurity aspect of IT systems.

Threats are numerous: malware in all its forms (spyware, ransomware, trojan…), sniffing, spoofing, Man In the Middle attacks, backdoors in hardware or software, etc. They are also spread across most if not all IT domains, ranging from the simplest, cheapest IoT devices to the more expensive smartphones, cars, planes, banking systems, air traffic control systems, etc. They know no frontiers, as IT information can easily cross (most) frontiers in the world, and know no delays thanks to the quasi-instantaneousness of information transmission. Cyber threats used to be considered as being limited to hacking, which itself was seen as an uncommon activity of geeky-nerdy underground teenagers (see e.g. the movie "Wargames" [10] of 1983). Now, there is plenty of evidence that this has morphed into "industrial-scale" activities, with states using cyber warfare units even in relatively peaceful times, and organized crime also having their professional hacker teams. Even a period of global pandemic such as that of COVID-19 is not creating any truce on the cybersecurity front, since "Cybercriminals are developing and boosting their attacks at an alarming pace, exploiting the fear and uncertainty caused by the unstable social and economic situation created by COVID-19." [11]

Yet, it is clear that as a society it is important that all our systems, both industrial-scale and personal ones, be as secure as possible. The stability and continuity of our daily activities, be they private or professional, and even our lives, depend heavily on the secure and continuous operation of IT systems. Until very recently, security was not a top priority for those who design and implement IT systems. This has led to the current situation, with scores of vulnerable IT systems being used, and many still being developed with poor security.

Cybersecurity can be seen as a defence system, and has often been compared to medieval castles. Although it is true that some cyber defences can be nested like medieval castle defences were, the metaphor is inappropriate for whole IT systems. Unlike medieval castles, in which attackers must pass all the defences *successively* (climb the slope, and cross the outer moat, and climb or breach the outer wall, and the same for the inner wall, *and* for the dungeon), IT systems are indeed much less nested. They are more layered, or stacked, meaning that any breach present in *either* the hardware, *or* the operating system, *or* the execution environment, *or* the application, could be exploited by an attacker to gain information or control from the system. In that sense, *one* hole in IT systems is enough. This is a bit similar to the proverbial "forgotten, concealed postern" in a castle, or to a secret tunnel. Of course, there still may be some compartmentalization, in which case not the whole system falls but only part of it (a bit like one castle among several). But overall, IT systems and their defence appear rather more vulnerable in principle that the iconic medieval castle.

However, solutions exist to this grim situation, some of them being specific to one layer of the IT system, some others being applicable to several layers. Many defence techniques exist, and can add their "stone" to the security walls of IT systems.

Address Space Layout Randomization (ASLR) is a defense technique which consists in making the structure of processes more random with regards to the places in which code, data and libraries are found in memory, so as to make the task of an attacker (e.g. malware) more difficult. Indeed, by offering it a less predictable target, the attacker can less easily move/jump the instruction pointer at execution to a target of its choice, which makes the executable less vulnerable to buffer overflows and code injection. ASLR is a cross-layer technique, since it can be applied to the memory of application code/data, execution environment (e.g. VM) code/data, and even OS code/data.

Another way to create a "stronger stronghold" for IT systems is by having a smaller attack surface, by basing everything on a smaller yet highly secure base, like the **TCB** (Trusted Computing Base). Research work has clearly shown that "From the security point of view, the monolithic OS design is flawed and a root cause of the

majority of compromises." [22], and that **microkernels** make it easier to have more secure OSes, even more so when they are verified. Indeed, OSes must have some parts that execute with the highest level of privileges, in kernel mode. But being software, kernels are also flawed. The bigger the kernel, the more flaws can be exploited in kernel mode to gain access to all information on the system. By reducing this attack surface to a bare minimum, the risk is mitigated. By reducing this kernel code to a bare minimum, it becomes easier to check it, secure it, or even verify it formally.

At the application and OS levels, a simple solution would be to use end-to-end cryptography in order to provide better protection for user data with regard to attacks, at storage level (encrypted filesystems), memory level and communication level. Recent OSes make it easy to encrypt files or the whole filesystem, and dedicated hardware support has made the cost of this largely unnoticeable in practice. At application level, several messaging or conferencing applications have introduced end-to-end crypto, some of them even putting a strong emphasis on this aspect (e.g. Olvid [23], Signal [24]). This need has only grown with COVID-19-induced teleworking, and the flurry of cyber attacks it has allowed [11].

Many more techniques exist. We focus in the remainder of this article on a few techniques that must be supported and promoted since they provide the means for the EU to secure its IT systems and IT ecosystem. We show very promising *technical solutions* that make it possible to **find and fix vulnerabilities** in existing IT systems, to **strengthen** the security and **resilience** of IT systems, and to **express security properties** in order to be able to *verify* the security of existing systems and to *produce* new ones that are much more *secure* if not devoid of any vulnerability. But if we do want to increase the security of our systems, we also need to *motivate* all actors to act in their best shared interest. This may imply **regulations**, based on **liability** in the face of the law and on **certification** processes. Finally, cybersecurity and IT sovereignty, hence simply **sovereignty** for the EU, depend on **trustable and auditable hardware and software**.

Image: ID 133487501 | © FabioConcetta | Dreamstime.com

Finding and fixing vulnerabilities in existing systems

Since it is now infeasible to recreate all systems from scratch and redesign and rewrite them with perfect security, we have to live with the legacy of our existing systems for a long time to come. As a result, these systems will need to be made secure to ensure that users and companies can access and store their private data for years to come.

In some cases, pragmatism can help us move forward on this path. As an example, consider the now commonplace technique of ASLR, which defends against memory-related vulnerabilities in software. Its core idea is that software will contain many bugs that can lead to exploitable vulnerabilities, but that attacks against these vulnerabilities were made exceedingly easy by the fact that program code and shared libraries were located on fixed addresses in memory. ASLR randomizes the locations of program code and libraries in memory, which makes such easy attacks fail. Thus, applying ASLR to all systems increases the cost and difficulty of mounting attacks, even though it cannot completely prevent all possible attacks. Still, users are safer thanks to it.

This goes to show that in the cases where we cannot rigorously ensure the required security properties, we can still increase the overall security of systems through other means. In particular, **we need to have techniques to find, and fix or mitigate, security vulnerabilities in legacy code bases**; or techniques to at least isolate, thanks to containerization, these potentially insecure legacy code bases in hypervisor-like infrastructure, controlling at the same time their exchanges with the rest of the system. While none of these techniques will be able to guarantee that programs are *completely* secure, each of them can lower the impact of inadvertent mistakes, and can raise the bar against specific attacks. The more of these techniques that are combined, the greater the security of the resulting system will be.

Even in large-scale systems, which are hard to deal with globally, vulnerability finding techniques can perform a dual function. While an automated tool being unable to find security vulnerabilities in a certain amount of time does not constitute proof of the security of a system, if such an automated tool *does* find a security vulnerability, that vulnerability serves as a constructive proof of the system's insecurity. Such security vulnerabilities (found automatically) can be passed as actionable items to the appropriate engineers to solve. This is a much more tractable task than trying to prove the security and correctness of an entire, large-scale system. For example, operating system kernels are complex, security-critical systems that are hard to analyze. Special-purpose techniques can be developed that target different kinds of bugs that can have a security impact in operating systems [12,13]. These analyses can still be improved in terms of how many such bugs they find in certain hard-

to-analyze contexts. Furthermore, as such tools only target specific classes of bugs, more such tools need to be developed in order to find so-far under-reported classes of bugs.

For the cases in which the systems still contain vulnerabilities which neither automated tools nor manual code analyses find, we need to have ways to mitigate the impact of these remaining vulnerabilities. This is typically done by targeting the ways in which an attacker typically exploits such vulnerabilities. An example of this was already touched upon in the context of ASLR: if typical attacks use hard-coded memory locations, making memory locations vary between executions will make attacks harder. Similarly, attackers can try to redirect the execution of the program in a way which the original developer did not write in the source code of the application. Then a type of defence, called control-flow integrity, inserts checks that verify that functions are executed only from the calling context of functions of which the developer intended this [14]. More such defences need to be researched and developed.

Diversity is a mean of resilience against attacks

One way of mitigating and protecting against security vulnerabilities is introducing *diversity*. This is akin to monoculture in crops: having a more diverse gene pool in crops can make fields more resilient to diseases and infections. This was already alluded to in the context of ASLR: if every execution of a program has a totally unpredictable memory layout, it is harder for an attacker to create a single exploit that works against all these different memory layouts. An attacker would need to carefully craft an exploit that works around these limitations, or there would need to be a way for the attacker to gain knowledge of the exact memory layout.

Diversity can be introduced at many different, if not all, levels of an IT system: hardware, operating system, execution environment, application level. This can be done by having different implementations, by generating different versions at compile time[15], by randomizing the programs at install time[16], by randomizing them at load time such as with ASLR, or even by randomizing them during the execution of the program [15,16]. Diversity has even more security-related applications. For example, multi-variant techniques take their inspiration from reliability in critical systems such as airplanes in which multiple different implementations are compared against one another [18]. By executing (specially chosen) diversified instances of the same application, feeding these the same inputs, and then comparing their outputs, some classes of attack can be prevented from the fact that these attacks will impact these instances in a different way, which can be detected[19]. An application of diversity in a wider sense can be found in the context of security updates. When a developer discovers and fixes a security vulnerability and releases an update, users don't typically apply this update immediately. However, attackers can still compare the original application which contains the vulnerability with the just-released updated version of the application in which the vulnerability has been patched. Based on this difference, attackers can easily create attacks against the users who do not yet have the update installed [20]. This is sometimes called patch *Tuesday / exploit Wednesday*, as Microsoft typically releases their (security) updates on a Tuesday, after which attackers can try to exploit the unpatched users the day afterwards. A mitigation to this can consist of making the original version and the updated version differ more, that is, making both versions more diverse from each other. This will slow down the analysis and attack-generation by the attacker, allowing more users to apply the security update [21].

Some of these diversity techniques are already widely adopted, but definitely not all. One reason that some proposed techniques are not yet used in practice, is that they have too high an overhead to be applicable in most practical situations, or still have other limitations. Furthermore, as no technique can prevent every possible vulnerability, it is important that more such techniques are investigated and can be applied in practice. **This means researching new diversity techniques that are both efficient and effective, as well as making existing technologies more widely applicable**, by raising their maturity levels such that they can be adopted at large, rather than existing as academic prototypes.

Non-functional security properties must be included as first class citizens in new IT systems

In addition to taking care of vulnerabilities in existing legacy systems, it is of the utmost importance to include security as a first-class citizen when building new IT systems. Way too often, security is not considered upfront in programs at design and implementation stages, but only as an afterthought. This situation absolutely must change. The non-functional property that security is, or more precisely the set of non-functional properties that pertain to security in its various aspects, must be present in programs, in the minds of designers and implementers, from the very beginning of the creation of an IT system.

An example of a security property would be the level of threat by interception and decryption of communications in a given environment (e.g. known by its location in a military conflict). Another example would be, say, the strength of a cryptographic algorithm. An IT system could adapt its operation depending on the threat level, sending unencrypted or lightly (hence cheaply in term of time and energy)

Image: ID 128042127 | © Blackboard373 | Dreamstime.com

encrypted information in safe cases, choosing a stronger encryption algorithm in higher threat situations, or even avoiding transmitting if the level of threat is above the strength of the available algorithms.

To get to this point, security properties must be treated as first-class citizens, which means that they have to be expressed as clearly and as explicitly as the functional properties (i.e. what the program does, its algorithms), and that designers and developers should be able to reason about them, query them, manipulate them, same as for the functional aspects of the program. It is only by having security interleaved in all the fibres of the IT system that it can be solid.

To this end, first, security properties must be present in the **specifications** of the IT system from the beginning. Designers must be able to express what levels of security they want, for the various facets of security, or reasons about them. **Security contracts** must be present in the system that express and guarantee security at module, or component boundaries. Developers must be able to pick up off-the-shelf **modules** or components knowing the security levels they provide for specific facets or security, and plug them in as part of the security continuum of their system.

Programming languages are also not all equals in terms of security. Many programming languages tolerate sloppy programming, where code that looks reasonable at first sight may in fact contain major vulnerabilities. Some, e.g. C/C++, tend to be harder to master, more error-prone, making it difficult to find bugs like security issues. For a compendium of programming language vulnerabilities, see the work by ISO/IEC TR 24772 Programming languages — Guidance to avoiding vulnerabilities in programming languages [2]. Other languages should be promoted: those that have stricter rules, more safeguards, and make it easier to develop more secure code. The Rust programming language, originally developed by Mozilla, is one example that makes many aspects of memory management and memory safety explicit in its language constructs, making it harder to leave room for security glitches that could be exploited malevolently.

Automated or semi-automated **tools** must be able to rely on these specification and security contracts **to verify security**, to prove security properties, to provide strong guarantees about the quality of IT systems with regard to security, at both specification and code level. Notable examples that can help in this endeavour in the line of program verification include Frama-C [3], Coq [5] and Ada SPARK's Discovery toolset [4]: those tools operate on the premise that the source code must conform to some formal specification.

Although **security by design** of whole IT systems seems a perfect answer to security issues, the current approach is often more limited and pragmatic, with only a limited part of the IT system being trusted. This Trusted Computing Base (or TCB) [1] comprises the system hardware, firmware and software components whose combination is intended to provide the system with mechanisms for a secure environment. The idea here is that **verifying**, either automatically with verification tools or manually by human examination, this on a small set of hardware and software is a more tractable and less costly task than doing it on the whole system. However, verification of large pieces of real-life software is already doable, as proven by recent research-level successes like the CompCert compiler [6] and the verified parts of the seL4 microkernel [7].

Liability for IT systems

Liability and **certification** are crucial building blocks needed to mandate taking into account non-functional security properties in IT systems. Indeed, **adding the legal building block of liability**, hanging the threat of a potential cost on non-secure systems, makes it more appealing for system builders and providers to commit effort, and hence money, on having secure systems. With liability, the extra time of specification and the extra step of verifica-

Image: ID 119668736 | © Alexandersikov | Dreamstime.com

tion would become worth taking. **Certification is the technical building block** that makes the liability legal one viable. With certification, system builders can have their efforts for security quantified, priced and legally acknowledged; purchasers can mandate security based on an independent assessment; regulatory bodies can outlaw low-security systems. Certification can be based on test suites that must be passed, on verification tools, on development practices that must be adhered to, etc.

With liability and certification, companies have the incentive to create secure software, or to keep finding, and mitigating or fixing vulnerabilities.

Sovereignty depends on trustable and auditable hardware and software

A castle can have the deepest moat and the highest and strongest walls, but they are of no use if an adversary has the key to the secret tunnel or hidden postern. Similarly, if we don't control the parts of our IT systems that are crucial for cybersecurity, we can hardly guarantee it. If backdoors exist in the operating system or even in the hardware we buy, unknown to us, they can be exploited by attackers and it is very difficult to add extra elements of security to counter them. The same is true for development tools, like the compilers that generate the actual executables, or the software libraries used as building blocks to compose programs: being closed source and distributed as binaries, they could embed backdoors in the programs that are produced with them.

Currently, by basing most, if not all, of its activities on IT systems running on proprietary hardware and operating systems not made in the EU, the EU effectively entrusts the providers with the keys to its whole economy and all aspects of its (cyber)security. The situation is barely better for development tools, compilers, and libraries, in which EU production is quantitatively very limited. This is why it is crucial to retain the keys of the castle, which means to retain sovereignty and control over the important hardware and software components for at least the TCB, and hopefully the whole software stack.

The way for the EU to reclaim its IT **sovereignty** is thus to **base the TCB of EU IT systems either on open-source software and hardware, or on EU-made, trustable-because-audited, proprietary hardware or software.**

Conclusion

(Cyber)security is only as strong as its weakest link, which means that the level of security of an IT system is at the minimum of the level of security of its components. For way too long has security been forgotten in the interests of better time-to-market and lower prices. However, increased awareness of the **cost of poor security**, coupled with the flurry of attacks in recent years, in a much more organised way than before, is now making it both necessary and possible to reverse this trend, and to take a path of better security in IT systems, providing better security for economies and for people. The technical building blocks to this end are within sight in the research community and must be nurtured and pushed forward, so as to quickly mature and then irrigate the whole IT industry. Europe has a key role to play in terms of **IT systems that are more valuable because of their higher quality thanks to higher security**, by promoting **targeted research**, taking the appropriate **regulatory steps**, and taking the necessary steps to reclaim **sovereignty of the critical building blocks** of its own IT systems.

References

[1] TCB: Trusted Computing Base: https://en.wikipedia.org/wiki/Trusted_computing_base

[2] ISO/IEC TR 24772 Programming languages — Guidance to avoiding vulnerabilities in programming languages: https://committee.iso.org/sites/isoorg/contents/data/committee/04/52/45202/x/catalogue/p/1/u/1/w/0/d/0

[3] Frama-C: https://frama-c.com/features.html

[4] Ada SPARK's Discovery toolset: https://www.adacore.com/sparkpro

[5] Coq proof assistant: https://en.wikipedia.org/wiki/Coq

[6] CompCert compiler: http://compcert.inria.fr

[7] seL4 microkernel: https://sel4.systems

[8] Petya ransomware attacks: https://en.wikipedia.org/wiki/Petya_(malware)

[9] Wannacry ransomware attacks: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack

[10] "Wargames" movie: https://www.imdb.com/title/tt0086567/

[11] INTERPOL report shows alarming rate of cyber attacks during COVID-19: https://www.interpol.int/News-and-Events/News/2020/INTERPOL-report-shows-alarming-rate-of-cyberattacks-during-COVID-19

[12] Precise and Scalable Detection of Double-Fetch Bugs in OS Kernels. Meng Xu, Chenxiong Qian, Kangjie Lu, Michael Backes, Taesoo Kim. IEEE Symposium on Security and Privacy, 2018

[13] Check It Again: Detecting Lacking-Recheck Bugs in OS Kernels. Wenwen Wang, Kangjie Lu, Pen-Chung Yew. ACM Conference on Computer and Communications Security, 2018.

[14] Control-flow integrity principles, implementations, and applications. Martín Abadi, Mihai Budiu, Úlfar Erlingsson, Jay Ligatti. ACM Trans. Inf. Syst. Secur. 2009

[15] Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization. Cristiano Giuffrida, Anton Kuijsten, Andrew S. Tanenbaum. In USENIX Security Symposium, 2012

[16] SoK: Automated Software Diversity. Per Larsen, Andrei Homescu, Stefan Brunthaler, Michael Franz. IEEE Symposium on Security and Privacy, 2014

[17] Librando: transparent code randomization for just-in-time compilers. Andrei Homescu, Stefan Brunthaler, Per Larsen, Michael Franz In ACM Conference on Computer and Communications Security, 2013

[18] N-Variant Systems: A Secretless Framework for Security through Diversity. Benjamin Cox, David Evans. USENIX Security Symposium, 2006

[19] Cloning Your Gadgets: Complete ROP Attack Immunity with Multi-Variant Execution. Stijn Volckaert, Bart Coppens, Bjorn De Sutter. IEEE Trans. Dependable Secur. Comput. 2016

[20] Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications. David Brumley, Pongsin Poosankam, Dawn Xiaodong Song, Jiang Zheng. IEEE Symposium on Security and Privacy. 2008

[21] Feedback-driven binary code diversification. Bart Coppens, Bjorn De Sutter, Jonas Maebe. ACM Trans. Archit. Code Optim. 2013

[22] The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-based Designs Improve Security. Simon Biggs, Damon Lee, Gernot Heiser: APSys 2018: 16:1-16:7

[23] Olvid. https://olvid.io/technology/en/

[24] Signal. https://signal.org/docs/

Olivier Zendra is a Tenured Computer Science Researcher at Inria, Rennes, France.

Bart Coppens is Postdoctoral Researcher in the Electronics department of Ghent University, Ghent, Belgium.

This document is part of the HiPEAC Vision available at hipeac.net/vision.

This is release v.1, January 2021.

Cite as: O. Zendra and B. Coppens. Cybersecurity must come to IT systems now. In M. Duranton et al., editors, HiPEAC Vision 2021, pages 74-79, Jan 2021.

DOI: 10.5281/zenodo.4719394

The HiPEAC project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 871174.