Software Security

This NESSI[1] paper reviews today's challenges of securing software throughout the entire software development lifecycle and recommends research directions to address those issues. NESSI puts particular focus on those software security approaches and techniques that will play a role in the implementation of upcoming regulations such as the European Cyber Resilience Act[2].

Motivation

Cyber attacks increasingly exploit vulnerabilities in software supply chains. Compromised software spreads along the distribution channels of the supply chains and leads to large-scale incidents with devastating effects for those who deploy and use the software. Such attacks raise serious concerns about the security of software products and digital services, leading regulatory authorities to launch initiatives with the aim of improving the security of software and software supply chains. In Europe the Cyber Resilience Act (CRA)[2] and in the US the Executive Order on Improving the Nation's Cybersecurity[3] have been initiated to address these issues.

Software vulnerabilities are not a new phenomenon. In 1988 some of the first cyber attacks took advantage of a buffer overflow vulnerability[4]. Today the number of software vulnerabilities is still growing as shown in Figure 1 – and it is surprising that overflow vulnerabilities are still among the most frequent ones[5]. What has changed over time is the complexity of software and the software lifecycle. Now rarely built from scratch, software depends increasingly on components supplied by open source projects or third parties. Those components may rely on other components, resulting in a network of components with many transient dependencies. Components such as source code, binaries, plugins, configuration files, container manifest files etc., and tools such as compilers, repositories, and code analysers, as well as people and processes involved in the software lifecycle, are all part of a software supply chain. Such a complex and interrelated system of technologies, people and processes provides a large attack surface to bad actors. This attack surface extends across all phases of the software lifecycle, as pointed out by the IEEE SWEBOK: "To design security into software, one must take into consideration every stage of the software development lifecycle. In particular, secure software development involves software requirements security, software design security, software construction security, and software testing security. In addition, security must also be taken into consideration when performing software maintenance as security faults and loopholes can be and often are introduced during maintenance"[6].

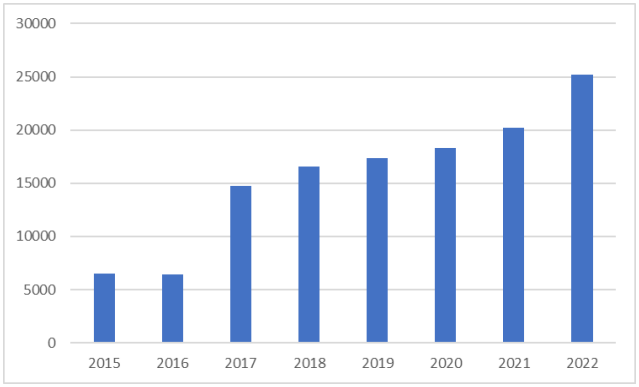

Figure 1 Number of software vulnerabilities (*https://www.cvedetails.com/vulnerabilities-by-types.php*)

[1] NESSI (Networked European Software and Services Initiative), the European association promoting research, development and innovation in the field of software, data and digital services; http://www.nessi.eu/

[2] Cyber Resilience Act | Shaping Europe's digital future (europa.eu)

[3] FACT SHEET: President Signs Executive Order Charting New Course to Improve the Nation's Cybersecurity and Protect Federal Government Networks - The White House

[4] https://en.wikipedia.org/wiki/Morris_worm

[5] https://www.cvedetails.com/vulnerabilities-by-types.php

[6] SWEBOK V3.0, Guide to the software Engineering Body of Knowledge, IEEE; chapter 13 section 17

The onus is on software security and software security engineering to mitigate the risks of vulnerabilities along the supply chain and throughout the entire software lifecycle.

Scope

The target domain of this paper is the lifecycle of engineered software products and services (hereafter collectively "software"). We focus on identifying major security challenges along this lifecycle that need to be met for a successful implementation of the CRA.

The objectives of the CRA are to improve the security of software products throughout the entire product lifecycle, to introduce certification requirements and conformity marking for product security, to enhance the transparency of security properties of software products, and to enable business and consumers to use those products securely. Consequently, we focus on software security approaches and techniques that are essential for meeting these objectives of the CRA:

- software development practices that improve the security of software and lead to fewer vulnerabilities;
- quality assurance through certification and conformity assessment;
- transparency of security properties through software composition analysis and threat intelligence sharing; and
- secure usage supported by automated vulnerability detection and reporting.

In addition, NESSI feels that the integration of software into complex systems deserves dedicated consideration, because growing system complexity creates specific difficulties and challenges in ensuring the security of those software-intensive systems.

Figure 2 provides an overview of the scope of this paper. It shows the stages of the software development life cycle and the corresponding phases of a DevOps pipeline including the feedback loops. Figure 2 also maps the key requirements of the CRA to the corresponding phases of the lifecycle.

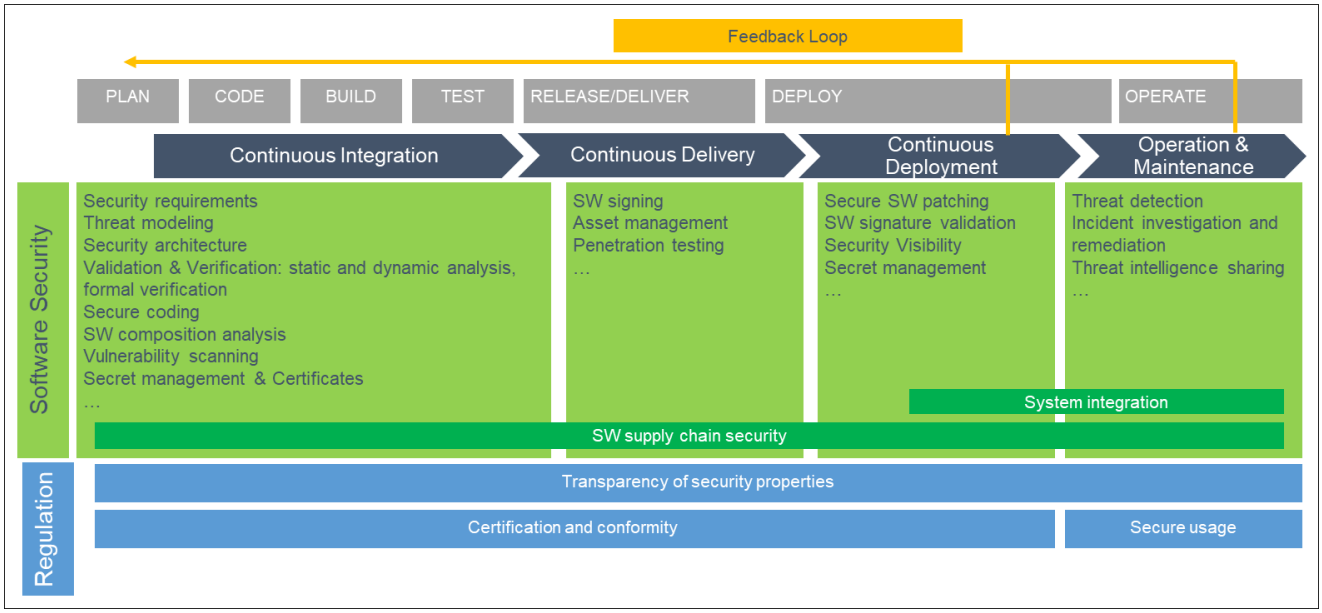

Figure 2 Software security scope

Software security in the context of software supply chain regulation

Automated verification and validation of software

Verifying and validating that software design and code meet specified security requirements are fundamental principles of developing secure software[7]. There are different approaches applied at different stages of the development process[8].

- Static analysis examines a program without executing it by using an abstract representation of the code and its runtime behavior. The abstract models can be control flow and call graphs which allow the application of control flow analysis or data flow analysis techniques.
- Dynamic analysis modifies a program or its runtime environment so that the program can be monitored while it is executed. Techniques include dynamic taint analysis (tracking the propagation of untrusted data throughout the software), dynamic symbolic execution (trying to execute all possible program paths), or fuzzing (feeding the interfaces of a program with random, semi-random, and malformed inputs – often used for penetration tests).
- Formal verification uses an abstract mathematical model of a software system to prove the correctness of an algorithm or certain program properties, including security properties. Methods include model checking, theorem proving, and bounded verification.

Challenges

Challenges and limitations of the different approaches are manifold[8]. Static analysis tends to produce false positives (i.e. flagging vulnerabilities which are not exploitable) whereas dynamic analysis techniques run the risk of producing false negatives (i.e. exploitable vulnerabilities are not detected). Thus nominally automated processes based on static analysis frequently require manual intervention as reported vulnerabilities must be verified by inspection. In dynamic analysis the instrumentation required for monitoring may lead to performance problems, and so-called probe effects may cause unintended changes of the behaviour of the software under test. Another challenge is to capture the impact of different runtime environments. How do the dependencies of the deployment environment affect the component? Will different cloud environments affect the component under test differently?

Performance and scalability can be issues for formal verification. Computationally intensive algorithms make it challenging to verify business-relevant complex software systems. Developing formal specifications and ensuring their correctness is a non-trivial task that requires expert know-how: “if a verified proof is based on the wrong assumptions, it can validate invalid code and produce useless, and even dangerous, results”[9].

Recommendations

Recent research has achieved remarkable advances in formal verification[10] and has resulted in tools such as Z3[11], a state-of-the art theorem prover from Microsoft Research, and Coq[12], a theorem prover from Inria. Research is needed to address performance, scalability and usability issues, the automated generation of formally precise specifications, and how to decompose a formal specification into parts that can be analysed independently from each other so that performance can be improved by analysing these parts in parallel[13].

[7] Fundamental practices for secure software development, SAFECode, March 2018
[8] Sungdeok Cha, Richard N. Taylor, Kyochul Kang, Handbook of Software Engineering, Springer, 2019
[9] Formal Software Verification measures up, Communications of the ACM, July 2021
[10] Formal Software Verification measures up, Communications of the ACM, July 2021
[11] https://microsoft.github.io/z3guide/docs/logic/intro/
[12] Welcome! | The Coq Proof Assistant (inria.fr)
[13] Sungdeok Cha, Richard N. Taylor, Kyochul Kang, Handbook of Software Engineering, Springer, 2019

Use should be made of information from multiple vulnerability databases, and feedback information gained from the software execution. Approaches could include sandbox testing using digital twin technology for the simulation of the deployment and operational environment and using self-learning and self-evolving software test systems that are informed by recently discovered vulnerabilities.

Software composition analysis

As previously noted, software is now rarely built from scratch and has many third-party dependencies. Vulnerabilities are continually being discovered in both open source and proprietary components, and it is in the interest of developers and users that it is possible to quickly determine whether a particular vulnerability affects their software. Both the CRA[2] and the US Executive Order[3] recommend a Software Bill of Materials (SBOM)[14] to provide transparency and traceability of software dependencies and provenance. There are several commercial offerings software developers can use to create and analyse SBOMs[15].

Challenges

There are several competing SBOM formats, notably SPDX[16], CycloneDX[17], and SWID[18]. They cover roughly the same information[19], which should make it feasible to translate between them. Today's SBOM tools have difficulty analysing dependencies of binary code and software containers, resulting in inaccurate and incomplete SBOMs. The correctness and completeness of SBOMs also depends on the extent to which suppliers provide a comprehensive SBOM for all the commercial and open source software included in their components. Moreover, SBOMs need to be kept up to date with each new software release.

Recommendations

The different SBOM standards need to converge into a common, internationally accepted solution that allows uniform and effective usage of SBOMs. Further research is required to improve software composition analysis, with the goal of providing accurate and complete SBOMs along the entire software supply chain.[20]

SBOM handling needs to be automated and embedded into DevOps pipelines. The integrity of SBOM files needs to be protected from malicious tampering. Additional tools are needed in order to get all the benefits from the information provided by a SBOM. For example, it is essential to know whether a newly reported vulnerability is exploitable. A SBOM does not provide this information, and additional security advice is needed e.g. from a Vulnerability Exploitability eXchange (VEX) document[21]. Research and open source projects could help to evolve such a tool landscape.

Certification and conformity assessment

For the purposes of this paper, certification is defined as conformance testing of software[22], confirmation that it meets certain standards and the assertion of the certifying body that the tests have been correctly carried out and that the unit under test has passed them.

Challenges

Conformance verification leading to certification is often a costly process involving significant human effort and expert knowledge.

[14] https://ntia.gov/sites/default/files/publications/ntia_sbom_framing_2nd_edition_20211021_0.pdf
[15] https://snyk.io/blog/building-sbom-open-source-supply-chain-security/
[16] https://spdx.dev/
[17] https://cyclonedx.org/
[18] https://csrc.nist.gov/Projects/Software-Identification-SWID
[19] https://ntia.gov/sites/default/files/publications/ntia_sbom_framing_2nd_edition_20211021_0.pdf
[20] https://thenewstack.io/sboms-are-great-for-supply-chain-security-but-buyers-beware/
[21] VEX one-page summary (ntia.gov)
[22] https://www.guru99.com/conformance-testing.html

Certification can be brittle. Examples include requirements going out of date, usage contexts changing, software functionality evolving through patching, and AI components evolving through learning (hence current concerns about regulatory approval of locked and adaptive self-learning AI, e.g. in the context of AI as a medical device[23]).

For classified systems the Common Criteria (CC) ISO/IEC 15408[24] has been used for security evaluation and certification for some years. Some attempts have been made to create CC Protection Profiles for civil applications such as smart meters, but in general the complexity and scope of a CC evaluation makes it too costly other than for military use. Furthermore, CC suffers from brittleness, making it incompatible with modern software engineering paradigms that involve continuous changes.

Recommendations

Research into more efficient and automated certification processes and tooling for compliance tests is needed in order to reduce costs, especially when repeated testing is required. Certification is closely tied to domains, so investigation of different domains will be required, and which processes and tools can be used across domains identified.

There is a clear need for agile ongoing compliance testing against potentially moving targets for certification. Research is needed into strategies for and consequences of adaptive certification, including the following.

- Determination of which circumstances render an existing certification void. If new threats emerge that a component is not resilient to, its certification should be void.
- Specification of practical methods to trigger any voiding of certification. This could include automated monitoring or periodic field testing.
- A new evaluation and certification scheme that is suitable for civilian applications and for modern software engineering paradigms such as DevOps.

Integration of software into complex systems

The complexity of software systems is a key source of security challenges, which are typically considered in terms of risks. The risk level is determined by the impact of the risk combined with its likelihood. There are several processes and tools for cybersecurity risk management, notably the ISO27000 series of standards, but a common theme is that vulnerabilities in components allow threats to affect assets (the components themselves or other things of value such as data or people).

Challenges

The integration of components into systems allows a threat due to a vulnerability in one component to cause risks affecting other components. The system is affected by not only the vulnerabilities of the components but also by the complexities arising from the interconnection of the components. It may be the case that a certain combination of components interconnected in a particular way or given specific stimuli triggers vulnerabilities previously undiscovered.

Integrated systems typically have multiple stakeholders with their own priorities and concerns. They may be differently affected by security risks, so their different perspectives need to be considered. This includes those who are not direct actors in the system (e.g. users or operators) but may be still affected by risks in the system. A key example of this is a data subject whose data is processed in the system who will be affected if there is a data breach.

[23] https://www.core-md.eu/artificial-intelligence-and-medical-devices-regulation-discussing-the-legal-framework-and-the-ethical-challenges-within-the-core-md-project/

[24] https://www.iso.org/standard/72891.html

Recommendations

Further research is needed to examine threat propagation in systems of components, each of which may be a system in its own right.

There should be investigation into the composition of system-level BOMs from the SBOMs of the components within the system, and into how the system-level BOMs can be utilised to identify vulnerabilities in the system components, e.g. developing on existing methods of using SBOMs[25]. There will be challenges of integrating SBOMs into system level BOMs, of interoperation between the formats of the individual SBOMs, the vulnerability schemes they use, and how they are mapped.

Research is needed regarding vulnerability assessment at system level and the creation of dependency analysis tools to determine the most effective patching strategy for systems of components.

Automated vulnerability detection & maintenance

There has been considerable research into vulnerability detection and multiple mature databases of vulnerabilities exist, along with communities of practice who frequently contribute to them. When a vulnerability is detected, automated update and patch management can remediate it quickly to shorten the time a system is exposed to potential attacks. However, several challenges remain.

Challenges

There is little consistency between the different databases and schemes that record software vulnerabilities. This is a potential issue during the operational phase, if it involves testing against all known vulnerabilities coming from different sources and databases as there may be overlaps of vulnerabilities and different formats to deal with. While NIST, MITRE and others have established ways to report and uniquely identify vulnerabilities, there are no formal, (self-)consistent models to describe them. CVSS v2 provides metrics covering access, complexity and impact on confidentiality, integrity and availability, but misses important aspects such as loss of authenticity or loss of control. CVSS v3 requires users to assign impact metrics taking account of consequences beyond the vulnerable asset(s), but this depends on the system context, which changes often and may be open to different interpretations.

Keeping complex systems updated is becoming more demanding due to increasing scale and increasing interoperation between components. Patching at scale in distributed systems is especially challenging due to the complexity of managing the patching operation over multiple domains of control. If multiple vulnerabilities are detected it is a challenge to understand which are the most important. Vulnerability databases include a priority score, which is certainly helpful, but this does not take account of the system-wide impact. Obsolete software is challenging because patches are not available so any vulnerabilities found cannot be addressed via updates.

Recommendations

Research is needed into how the various vulnerability databases and formats may be seamlessly integrated into automated vulnerability detection and prioritised patching systems.

Research is needed into helping people understand the priorities of vulnerabilities detected at system level, so that they may be appropriately handled. This is related to the risks to the relevant actors, a component or the whole system due to threats propagated from other system components at runtime

Investigation is needed into automated tools for vulnerability detection and automated patching for software at scale and over distributed networks involving multiple domains of control. These tools should begin from SBOMs (or system-level BOMs), consult vulnerability databases, and determine and execute a patching strategy for the component or system under evaluation.

[25] https://cloudsmith.com/blog/how-to-analyze-an-sbom/

Investigation into automated tooling to detect the presence of unmaintained software26 and give warnings to operators is needed. Investigation of mitigation strategies for unmaintained software is also needed, for example suggestions of alternatives to the unmaintained software component or isolating the unmaintained software from the rest of the system, along with analysis of the negative impact of the mitigation strategies (e.g. cost of replacing unmaintained software or reduced system functionality from isolation).

Vulnerability reporting and cyber threat intelligence sharing

At the very beginning of software security as a discipline, McGraw27 identified the importance of "feedback from the field". This has become even more relevant with the advent of the DevOps paradigm, where e.g. web or cloud services are under continuous development, and discovered vulnerabilities can be fixed virtually immediately, without waiting for a prescribed patch cycle. When a successful attack is detected, details are fed back to developers to determine where in the code the vulnerability lies, and upon successful identification to search through the entire codebase to locate similar instances. This can then be further fed back to architectural patterns to avoid similar flaws in the future, and to custom security testing and penetration testing tools to better catch flaws that evade the "good practice net".

Cyber Threat Intelligence (CTI) which details vulnerabilities that are being exploited in the wild can be used both for learning about potentially undetected vulnerabilities and also for prioritization of remedial efforts for known (but unhandled) vulnerabilities.

Challenges

CSIRTs/CERTs and Information Sharing and Analysis Centres (ISACs) have an established tradition of sharing CTI such as indicators of compromise, based on one-to-one trust relationships. Platforms such as MISP28 provide the mechanics of CTI sharing, but equally important is the shared understanding of code of conduct according to the Traffic Light Protocol (TLP)29 which governs how shared information may be used and distributed. However, many who could share CTI choose not to do so, through either perceived risk to themselves, or insufficient benefit to themselves to warrant the effort involved in sharing.

Recommendations

The work of CSIRTs and ISACs should involve software developers, ensuring that CTI which can help to identify vulnerabilities makes its way back to those responsible for the code. This includes determining motivations and barriers to those who could share CTI but do not do so. Research is needed into multidisciplinary aspects such as education of the benefits of sharing CTI, tools to simplify CTI sharing, and cultural and prosocial behaviour in CTI sharing. This could be linked to the European strategy for data, potentially resulting in a Common European data space for CTI.

Research is needed into feedback from deployment and operation to identify key factors (e.g. environments, parameters, dependencies, vulnerabilities etc) and how they are incorporated into component-level testing. Feedback from real systems can improve the design of components within them. Collection of data from operational systems may yield improved synthesised test stimuli.

Editors

Josef Urban, Nokia;
Steve Taylor, IT Innovation Centre; and
Martin Gilje Jaatun, SINTEF.

[26] https://cwe.mitre.org/data/definitions/1104.html
[27] https://www.informit.com/store/software-security-building-security-in-9780321356703
[28] https://www.misp-project.org/
[29] https://www.first.org/tlp/