A Quest for Indicators of Security Debt

Simo Huopio

ABSTRACT

Security Debt (SD) manifests itself every day: In the media, we can witness stories about debt defaults: significant data leaks, disruptions of service, and businesses of global companies affected by security incidents. Critical infrastructure customers need more practical tools to ensure that the SD is properly identified in order to make informed risk decisions. Existing tools like security audits and cross-referencing system configuration with available vulnerability information do reveal a lot of data regarding the present state of the system. Many tools that can bring up the hidden security issues like ones included in Secure Development Lifecycles (SDLC) are tuned for the product creation. Some of them can be taken into good use by the customer, but for that, the system should be already procured and in place. In practice, the customer would benefit from a comprehensive and realistic view of the security stance of the system when it is being procured to minimize the nasty surprises the underlying security issues are prone to bring.

This paper reviews the existing approaches used for managing Technical Debt (TD) and continues the discussion about Security Debt. Using this information, a proposal for the Security Debt Indicator Framework (SDIF) was constructed.

Keywords: security debt, technical debt, security tools, SDLC

INTRODUCTION

1.1 Security Debt

Security Debt (SD) is a security domain specific adaptation of well-established Technical Debt (TD), which represents all the work that is left undone in order to gain short term speed and flexibility in product creation. Technical debt concept adoption has been prominent in areas of industry where investments are significant, the systems have an extended lifetime, and modifications and new feature development is common in the middle of the system lifecycle: e.g. aviation, maritime, power and utility, and military[1-3]. In the context of software, TD has been researched mainly regarding what effects it has on the maintainability and the cost of the project. Its effects are apparent: All software projects take some TD but taking an uncontrolled amount of it makes the projects struggle in the end with hard to maintain code and inadequate architecture. There are also numerous examples of catastrophic failures where vast amounts of development work had to be scrapped, and the work had to be started all over again[4-10].

Security Debt was coined by Chris Wysopal in 2011 and has been further discussed by Dan Geer, and David Conway in the specific context of software[11,12]. The exemplary white paper by Whitehouse and Vaughn[13] kept the discussion going[14]. From these references Security Debt can be defined as all security related work which is not done at an optimal time in order to mitigate all the security issues in the system, known or unknown. The security issues, bugs, and flaws in the system are symptoms of underlying security debt.

Latent, unidentified security issues in the code are transferred to identified security debt with security measures within the development project[13]. These measures are usually formalized as a Secure Development Life Cycle (SDLC). When the security debt is made visible, risk decisions can be made with the information: Some debt is considered worth fixing, some debt is accepted. So even with SDLC in place and executed properly, there will still be security debt, it will just be visible and in actionable form; not latent and unknown as before.

Whitehouse and Vaughn state that it is common that only the most high-profile security issues are getting fixed, while the low priority issues are left unhandled[13]. This approach can leave a cumulatively increasing number of lower level issues in the system, which can have a compound effect equal to or greater than a single high severity security bug. Managing this type of security debt is a challenge worth addressing in the new versions of SDLCs.

In the software domain, the security debt has its distinct differences from technical debt: The TD concentrates on the effects, risks, and cost to the manufacturer doing maintenance and further development. The amount of cumulated SD by itself does not add friction or otherwise hinder the system maintainability, but it incorporates significant risk to the business and data of the end user. According to Ernst et al.,[6,15] the amount of TD is usually quantified as developer work, the amount of work needed to pay back the debt in the code. At the same time, the effects of TD to the actual end product are harder to estimate. On the other hand, the quantity of SD can be hard to estimate as it can hide in a compound of rather small issues. While its effects are easily quantifiable, especially in hindsight: even the smallest security breaches in the news are quoted to cost millions.

While SD is not as easy to quantify as TD, the fact that a large amount of SD is a potentially massive liability to customers makes it an important research subject. Large amounts of TD in the system is also a risk to the customer, but in most cases, TD is more the system vendors liability. The broader adoption of the Security Debt metaphor could help the security-conscious customers of large and complex software to gather and communicate the security posture of the system in a more comprehensive way.

1.2 Critical Infrastructure customer needs

The viewpoint of this research is of a customer procuring and maintaining a Critical Infrastructure (CI) system. Typical characteristics for such systems are, for example, planning for an extremely long lifecycle (especially by software industry standards), security-conscious and sensitive operating environment, and the unique role of the government entity as a procurer and operator for the finalized system.

The extended lifecycles of such systems will, by itself, mean that system configuration will go through many phases throughout its lifetime: There will be new types of subsystems introduced in a mid-life update after years of the systems first version was taken into use. Some subsystems will face obsolescence and will be replaced with a more modern alternative with some considerable friction. Overall, there will be a technology-security paradigm, and testing technology changes throughout the software industry, which will challenge the fundamental system requirements many times.

Many of the mentioned characteristics can pose extra challenges in defining, onboarding, operating, and maintaining the software integrated into the system. In effect, the maintenance of such a system is more like a research and development project. If the system that has no security issues at the time of procurement is not actively maintained and adapted to the changing environment, the end user is taking a considerable risk that the system will face obsolescence in couple years.

In this context, the concept of Security Debt is very well-suited to communicating the security-related risks of the various aspects of the system in one universal way. As a further development of that notion, it was identified that developing indicators for Security Debt should result in a concrete tool that could directly be taken into use in the help of Critical Infrastructure procurement[16].

Presently the tools of analyzing and managing the technical debt—let alone the new concept of security debt—are designed to be used in the product creation. The CI customer needs concrete ways to assess the security stance of the system while at the same time having much more limited access to the source code, the underlying things behind the architecture and design choices. The customer communication about such implementation details with the system vendor during the procurement process is limited. Making sure the system requirements are met and that the system onboarding process and the maintenance contract will fulfill the customer needs has enough work without going through the underlying details. Nevertheless, the customer carries the full risk of security issues, while the manufacturer’s responsibility is usually very strictly managed.

1.3 Research Question

Breaking down the SD in the CI system end user point of view, we can identify the following distinct areas of SD.

a) The SD that cumulates in the system during the product creation

b) The SD that arises from newly discovered bugs in the system during the system lifecycle

c) The SD that cumulates from system configuration changes during the system lifecycle

d) The SD that cumulates from the environment and the usage changes during the system lifecycle

Considering the phase of system procurement, the end user needs way to estimate the amount of SD in the system (a), and the ability of the supply chain and the customer to identify and handle the new SD that is created or identified during the system use (b-d). The scope of this research is to create indicators pointing out issues that would prevent end user meeting these needs. The end-user usage of tools and methodologies for analyzing the software binaries or source code, which have been identified as good practice for CI systems in[16] is left for later research.

From these needs a research question was crafted:

RQ: What kind of indicators could be used to identify the potential for Security Debt in the procurement and maintenance of a Critical Infrastructure system?

2. METHOD

The chosen approach was to use the usage of the SDLC tools and identify best practices for managing SD and security-related TD by the vendor and possible integrators as starting point for constructing the indicators. The researcher used the SDLC standards also as a source for the best practices for the Critical Infrastructure system software maintenance.

This research used a literature study of Technology Debt done for the previous publication, “Thou shalt not fail - Targeting Lifecycle-Long Robustness while being vigilant for the Black Swans” [16] as a baseline. In addition, a new search was done to widen and update the selection of articles related to the Security Debt, SDLC and security requirements of software project. IEEE Xplore service was used as a primary source for the searches and the papers.

There is a considerable amount of research being done regarding TD in software. Most of it is concentrates on analyzing debt in the light of the work needed on the implementation of future new features, and the software maintenance. Technical Debt is seen as a slowly increasing friction in the software development activities, which must be managed or further development will face serious obstacles. TD is usually measured in units of work time or the equivalent cost, and there is research that strive to validate this measure [2,4-10,17-21].

In the specific field of technical debt and security, or separately, security debt, there were not many new research papers found. The technical debt-related terms were found to be commonly used by the software engineers when discussing background for security issues[4]. There was also research regarding mapping different aspects of TD to security issues[22-26]. Security debt was not usually used as a distinct term, but there were some exceptions. There were numerous news articles stating that security incidents were primarily caused by lax security culture, or badly maintained systems [4,7,11,13,14].

A separate search was done to identify the research regarding software security management tools, SDLCs and security requirements. Related standards, models, and requirements tools were identified and familiarized[27-31].

3. APPLICATION

3.1 Starting points from the literature

According to the material, the following tool types were identified as a suitable basis for constructing the identifiers for security debt:

- Original system requirements: Analysis of the requirements used in system creation related to security: non-functional requirements regarding robustness, integrity and maintenance.[7,25,26]
- Secure Software Development Process. SDLC related technical tools, including software composition analysis, threat emulation, robustness testing, and security issue management.[13,14,32-38]
- Technical debt management: Tools used in TD estimation that are not included in SDLC tools: Architectural and other source code level complexity analysis, software uality modelling and assessment tools and related standards.[2,5-7,20]
- Customer requirements: Analysis of the requirements used in system procurement. In addition to the actual security feature and non-functional requirements, the requirements related to SDLC, security standards, and vulnerability information exchange.[16]
- Operations: Analysis of the actions related to the secure operation of the system: configuration, training, usage environment and the usage in conjunction with other systems.[39]

Using this list as the starting point, a prototype of the Security Debt Indicator Framework (SDIF) was constructed.

3.2 Indicator design choices

The SDIF areas were chosen using the tool types discovered from the source material. The use case for the indicators was a very security-conscious customer who was procuring critical infrastructure system. The data collection was assumed to be done via a dialog with contacts of the system vendor, system integrator and the owner of the to-be-procured system or the customer. The dialog could be implemented as a combination of interview, written questionnaire and document exchange. Should the customer decide so, some of the needed information could be requested through requirements in the procurement process.

The prototype was constructed so that, at the first stage, a rough maturity level was mapped onto each of the SDIF areas with a series of questions. These questions indicate the adoption of best practices as identified by the literature search. At the same time, the availability of further information and data sources was acquired. The result was scored to give a rough top-level indication of how the security debt is handled, which can be useful within the system procurement process.

Table 1: SDIF areas

	SDIF Areas
A	Vendor / Integrator internal system requirements
B	Vendor / Integrator internal software security process
C	Vendor / Integrator internal technical debt management process
D	Customer system requirements and customization
E	Customer capability on the security analysis
F	Customer capability on the secure operation of the system

The questions related to SDIF areas A to C are directed to the system vendor. The same set of questions can be used for an integrator or another third party in a similar role to the system. The questions related to areas D to F are directed for the customer itself. All areas are listed in Table 1.

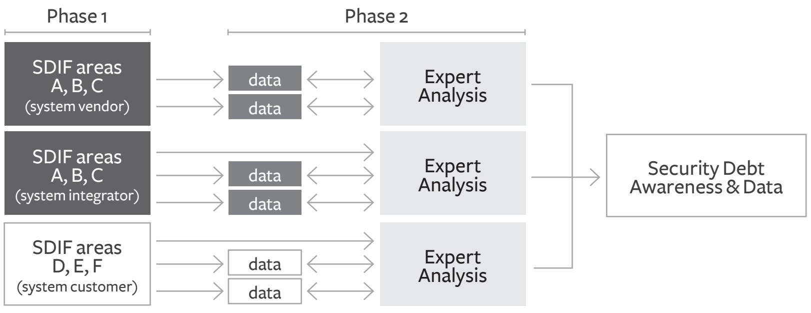
Phase 1
Phase 2
SDIF areas A, B, C (system vendor)
SDIF areas A, B, C (system integrator)
SDIF areas D, E, F (system customer)
data
Expert Analysis
Security Debt Awareness & Data

Figure 1: SDIF process

Going through all the areas will give the customer data from multiple dimensions of product creation, maintenance and the planned operation of the system. The process will also give several additional sources of information which can be used to focus on the implementation of a specific area. The overall process is illustrated in Figure 1.

3.3 High-level questionnaire

The prototype SDIF questionnaire is available in Appendix A.

With the first phase of general questions, the customer can get a rough idea of the maturity level of the security debt related choices made in the system. The availability of further data in each SDIF area and the interviewee confidence in the process was also mapped.

3.4 Additional data streams and actions to analyze them

Going through the questions related to each of the high-level SDIF areas will result in additional data streams that will need to be quantified and further analyzed.

It is to be expected that at least some of the data that is requested from the system vendor will be business sensitive, and the vendor may have some reluctance to opening the process. For example, there might be a conscious decision by the vendor to gather TD in the system in order to meet the time to market demands. If the SDIF information is requested in the form of product requirements, this might impact the pricing of the product in order to offset the business risk the vendor is taking in revealing the data.

In some selected areas, the additional data is both straightforward to share and have immediate value. The most evident data streams are listed below with their primary analysis approach. When directly applicable, the related SDIF question (available in Appendix A) is supplied in parenthesis. In other areas, the analysis of the data has to be done via expert opinion analysis.

A: Vendor internal system requirements

- List of non-functional requirements (A1) to be analyzed to know the case thoroughly.
- Statistics and trends regarding the test results against the non-functional requirements (A2). This data should show constant testing activity and the stability of the software.
- Timeline on handling third-party updates (A5) to be analyzed against the customer's own expectations.
- How long the system will be under active development, and how long the maintenance period will last until it is possibly replaced by succeeding product (A8 & A9)? This data should be analyzed against the customer's own plans on the system life cycle.

B: Vendor internal software security process

- Analysis of the data that will be sourced from the vendors SDLC implementation (B1 & B7): Threat analysis results and follow-up, risk assessment documentation, security testing results and trends. All critical threats should be mitigated, and the list of accepted security issues (threats, bugs, and misuse) should be acceptable to the customer. This data can also be reused in customer-side threat analysis work.

- Analysis of BSIMM, or other SDLC maturity metrics data (B3 & B6) should conform intuitively to the customer expectation of the vendor and the product maturity.
- The used vulnerability feeds followed by the vendor (B8) should include all used 3rd party software components.
- Data from possible bug bounty programs (B9) should match the data from non-functional requirements testing. The found issues should be properly mitigated.

C: Vendor internal technical debt management process

- The metrics from the TD management process (C1 & C3) will be cross-correlated with customers own testing and assumptions.
- The vendor-internal software QA metrics (C2, C4, C5) should show constant operation and overall good software quality.
- The data available about the vendor's approach to TD can be used in customer-side risk assessment work.

D: Customer system requirements and customization

- Analysis of the requirements document with all of its detail (D1-D4).
- Red teaming against the customer customization and configuration of the system (D5-D9).

E: Customer capability on the security analysis

- On a security sensitive system all of the listed capabilities are considered good practice. Any omissions should be carefully analyzed and written down as accepted risk.

F: Customer capability on the secure operation of the system

- All mentioned data should be available for further analysis. Any omissions should be carefully analyzed and written down as accepted risk.

3.5 Usage, balancing, and further development

The described process at its present form requires manual work to conduct. It is suggested that in the first phase, the customer go through questions with the vendor in an interview format. This approach gives the opportunity to make quick clarifications regarding the questions and the subsequent data. Gaining access, searching, and packaging the requested additional data in phase two takes time, as does forming expert opinions.

It is evident that, should any of the requested data be set available, the result will be beneficial to the overall security of the system. Applying these indicators before the procurement phase has begun could have the most significant impact on this approach. Then, in the best circumstances, scores from these sections could be used as one parameter in the procurement, even before the system requirements are finalized.

System updates that occur in the middle of its lifecycle can bring significant changes to the system's functionality, usage, and connectivity. These are points where security debt can quickly pile up. Applying SDIF on this phase, even when it has not been applied before, can help the customer avoid potential new security debt. SDIF can give excellent inputs to threat emulation, should the customer also want to apply it at this phase.

The result of this work is a comprehensive starting point for analyzing the critical infrastructure software system in a way that the presence of Security Debt can be indicated and the acts of mitigating it can be started before the related risks are realized.

4. DISCUSSION

4.1 Map and compass on hand

SDIF has been constructed in order to help the customer verify that the best practices in minimizing and handling security debt are in use. The absence or improper use of the tools and approaches mentioned by SDIF is an indicator that the SD is not properly controlled.

The present scope of SDIF cannot be used as a comprehensive indicator of actual SD in the CI software, even though enough data from vendor SDLC process could give some insights. The only way to know the amount of SD is to acquire even deeper knowledge of the software architecture and test results and additionally, to use one's own analysis and testing tools to verify the findings. However, before that can happen, most of the tools and methods mentioned by SDIF must be in use.

Applying the SDIF process will, in any case, increase the customer's knowledge of how the co-operation with the system vendor and possible involved third parties is planned to work should any security incident happen, be it news about new vulnerabilities on the platform the system is built on, or penetration testing results that point a finger at lousy configuration.

We now have a prototype of this framework and we are working to adapt it to internal use. Applying the framework will help the customers be much more knowledgeable in the procurement process, and at the same time, give a positive signal to the companies who have taken secure development seriously. Proper implementation and further development of SDLCs is imperative in order to add the ruggedness needed for the critical infrastructure to survive in the hostile threat landscape.

4.2 Rocky road ahead

Procuring an extensive, critical system already promises a daunting fight through bureaucracy, massive requirements and service level description documents, and a multitude of supplements. Taking on the additional steps to perform an obscure technical risk assessment might be a too much to take on.

Emphasizing the security and technical debt aspects during the procurement process can end up with a hefty price tag. This is a risk especially when the customer implies that the vendor should give out sensitive, internal data that is used in the internal risk assessments.

Still, the assessment of security debt handling cannot be ignored. It would mean too much of blind risk taking. The way forward will be a further development of related standards, reliable maturity models, and other ways to convey the data about the product creation and maintenance process to the customer. Then, it will be easier to trust that the system has been created with due diligence, and the inevitable incidents can be handled in a way that do not lead to catastrophes.

4.3 Quest continues

The next step with SDIF will be testing it with real systems, ones that are already in use and ones that will be procured. It will also be tried in the context of open source software (OSS) onboarding; in this case, the framework might need a variant where the relevant factors can be extracted from the OSS project, the code, documentation and the community itself. An exciting experiment will also be to analyze the data gathered in SDIF using the different security-auditing criteria that are in use on different nations for governmental use.

5. ACKNOWLEDGEMENTS

I want to thank my employer for the support for my research and for openness to give my new ideas a chance in the procurement pipeline. The colleagues on my research group at FDRA and Professor Juha Röning at the University of Oulu also deserve a big thanks for their support and comments.

Author Bio

Simo Huopio

Mr. Simo Huopio is currently working as Research Manager, Cyber Defence, at Finnish Defence Research Agency (FDRA). He received his Master's Degree (CS) from Helsinki University of Technology in 1999 and has since worked in multiple mobile and information security roles in F-Secure and Nokia before joining the Finnish Defence Forces. In addition to his work at FDRA, he is a doctoral student at the University of Oulu. His professional interests include software robustness testing, threat analysis, and practical cyber defence capability development.

NOTES

1. R.G. Muñoz, E. Shehab, M. Weinitzke, R. Bence, C. Fowler, S. Tothill, P. Baguley, Key Challenges in Software Application Complexity and Obsolescence Management within Aerospace Industry, *Procedia CIRP* 2015, 37, 24-29.
2. Z. Li, P. Avgeriou, P. Liang, A systematic mapping study on technical debt and its management. *J Syst Software* 2015, 101, 193-220.
3. B. Bartels, U. Ermel, P. Sandborn, M.G. Pecht, *Strategies to the prediction, mitigation and management of product obsolescence*, John Wiley & Sons, 2012.
4. R.L. Nord, Software Vulnerabilities, Defects, and Design Flaws: A Technical Debt Perspective, Fourteenth Annual Acquisition Research Symposium, Naval Postgraduate School, 2017, 67-75.
5. T. Besker, A. Martini, J. Bosch, Impact of Architectural Technical Debt on Daily Software Development Work - A Survey of Software Practitioners, 43rd Euromicro Conference on Software Engineering and Advanced Applications, 2017, 278-287.
6. G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou, A. Ampatzoglou, How do developers fix issues and pay back technical debt in the Apache ecosystem? 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2018, 153-163.
7. N.A. Ernst, S. Bellomo, I. Ozkaya, R.L. Nord, I. Gorton, Measure it? manage it? ignore it? software practitioners and technical debt, Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, 2015, 50-60.
8. B. Curtis, J. Sappidi, A. Szynkarski, Estimating the size, cost, and types of technical debt, Proceedings of the Third International Workshop on Managing Technical Debt, IEEE Press: 2012, 49-53.
9. Z.S. Hossein Abad, R. Karimpour, R.; Ho, J.; Didar-Al-Alam, S.M.; Ruhe, G.; Tse, E.; Barabash, K.; Hargreaves, I. Understanding the impact of technical debt in coding and testing: an exploratory case study, Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice, ACM: 2016; 25-31.
10. R. Marinescu, Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development* 2012, *56*, 9: 13.
11. D. Geer, C. Wysopal, For Good Measure - Security Debt. *;login*: 2013, *38 no. 4*, 62-64.
12. D. Geer, D. Conway, Foor Good Measure - The Price of Anything Is the Foregone Alternative, *login*: 2013, 3*8 no. 3,* 58-60.
13. Ollie Whitehouse, James Vaughan, Software Security Austerity - Software security debt in modern software development, *Rexc Whitepaper* 2012, *1*.
14. Nccgroup blog post: Revisiting security debt: Are we ready to have a discussion yet? Available online: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/march/revisiting-security-debt-are-we-ready-to-have-a-discussion-yet/, accessed on May 1, 2019.
15. A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology* 2015, 64, 52-73.
16. S. Huopio, Thou shalt not fail - Targeting Lifecycle-Long Robustness while being vigilant for the Black Swans, 23rd IC-CRTS (International Command and Control Research and Technology Symposium), November 6-9, 2018.
17. H. Ghanbari, T. Besker, A. Martini, J. Bosch, Looking for Peace of Mind? Manage your (Technical) Debt: An Exploratory Field Study, ESEM 2017: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ISBN 978-1-5090-4039-1, IEEE Computer Society Press: 2017.
18. R.L. Nord, I. Ozkaya, P. Kruchten, M. Gonzalez-Rojas, In search of a metric for managing architectural technical debt, 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, IEEE: 2012, 91-100.
19. N. Zazworka, C. Izurieta, S. Wong, Y. Cai, C. Seaman, F. Shull, Comparing four approaches for technical debt identification. *Software Quality Journal* 2014, 22, 403-426.
20. A. Martini, T. Besker, J. Bosch, Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. *Science of Computer Programming* 2018, 163, 42-61.
21. P.N. Bideh, M. Höst, M. Hell, HAVOSS: A Maturity Model for Handling Vulnerabilities in Third Party OSS Components, International Conference on Product-Focused Software Process Improvement, Springer: 2018, 81-97.
22. C. Izurieta, Kimball, D. Rice, T. Valentien, A position study to investigate technical debt associated with security weaknesses, 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), IEEE: 2018, 138-142.

NOTES

23. L. Lavazza, S. Morasca, D. Tosi, Technical debt as an external software attribute, Proceedings of the 2018 International Conference on Technical Debt, ACM: 2018, 21-30.

24. R. Alfayez, C. Chen, P. Behnamghader, K. Srisopha, B. Boehm, An empirical study of technical debt in open-source software systems. In *Disciplinary Convergence in Systems Engineering Research* Springer: 2018, 113-125.

25. M. Benaroch, Managing information technology investment risk: A real options perspective. *J Manage Inf Syst* 2002, 19, 43-84.

26. P. Avgeriou, P. Kruchten, I. Ozkaya, C. Seaman, Managing technical debt in software engineering (dagstuhl seminar 16162), Dagstuhl Reports, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik: 2016.

27. C. Banerjee, S.K. Pandey, Software security rules, SDLC perspective. *arXiv preprint arXiv*:0911.0494, 2009.

28. G. Disterer, ISO/IEC 27000, 27001 and 27002 for information security management, 2013.

29. C. Gikas, A general comparison of fisma, hipaa, iso 27000 and pci-dss standards. *Information Security Journal: A Global Perspective* 2010, *19*, 132-141.

30. N.F. Khan, N. Ikram, Security requirements engineering: A systematic mapping (2010-2015), 2016 International Conference on Software Security and Assurance (ICSSA), IEEE: 2016, 31-36.

31. N.M. Mohammed, M. Niazi, M. Alshayeb, S. Mahmood, Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces* 2017, *50*, 107-115.

32. M. Gustafsson, O. Holm, Fuzz testing for design assurance levels, Linköping University, 2017.

33. B. West, M. Wengelin, Effectiveness of fuzz testing high-security applications -&nbsp;A case study of the effectiveness of fuzz-testing applications with high security requirements, KTH Royal Institute of Technology, 2017.

34. S. Ognawala, A. Petrovska, K. Beckers, An Exploratory Survey of Hybrid Testing Techniques Involving Symbolic Execution and Fuzzing, *arXiv preprint arXiv:1712.06843* 2017.

35. Anonymous Application Threat Modeling, *https://www.owasp.org* 2017.

36. J.A. Ingalsbe, L. Kunimatsu, T. Baeten, N.R. Mead, Threat modeling: diving into the deep end. *IEEE Software* 2008, *25*.

37. R. Khan, K. McLaughlin, D. Laverty, S. Sezer, STRIDE-based threat modeling for cyber-physical systems, Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2017 IEEE PES, IEEE: 2017, 1-6.

38. S. Hussain, A. Kamal, S. Ahmad, G. Rasool, S. Iqbal, Threat modelling methodologies: a survey. *Sci.Int. (Lahore)* 2014, *26*, 1607-1609.

39. J. Stark, Product Lifecycle Management: 21st Century Paradigm for Product Realisation, In *Product Lifecycle Management (Volume 1)* Springer: 2015, 1-29.

APPENDIX A

A: Vendor internal system requirements

Nr.	Query text	Answer type(s)
A1	Do you have non-functional security requirements? Can you share these requirements and the related process?	Mandatory: Yes/No Optional: Free text & documents
A2	Do you test the product against the non-functional security requirements? Can you share the results?	Mandatory: Yes/No Optional: Free text & documents
A3	Do you have a common security architecture for the security features? What can you share regarding that architecture?	Mandatory: Yes/No Optional: Free text & documents
A4	Do you design and implement security features or functionality in a different manner than other features?	Mandatory: Yes/No Optional: Free text & documents
A5	Describe your software sourcing process: How do you manage requirements, quality, and software releases with third parties?	Mandatory: Free text & documents
A6	What platforms or codebases of the product is shared with your other products? How do you handle possible conflicts between the domains?	Mandatory: Free text & documents
A7	Is the product still in active development?	Mandatory: Yes/No Optional: Free text & documents
A8	How long are you going to support the product actively?	Mandatory: Free text & documents
A9	How long will the security patches be produced? Will their availability be tied to a support contract?	Mandatory: Time, Yes/No Optional: Free text & documents
A10	How confident you are on the completeness of the internal security requirements on scale 1 to 5?	Mandatory: Number Optional: Free text & documents

B: Vendor internal software security process

Nr.	Query text	Answer type(s)
B1	Is there a Secure Development Lifecycle (SDLC) or equivalent in use on R&D? Can you share the process description?	Mandatory: Yes/No Optional: Free text & documents
B2	Are you following a standard like ISO27k? Can you share the details?	Mandatory: Yes/No Optional: Free text & documents
B3	Has the maturity of the company's software security approach been evaluated with BSIMM or equivalent? Can you share the results?	Mandatory: Yes/No Optional: Free text & documents
B4	Do you use robustness testing in product creation?	Mandatory: Yes/No Optional: Free text & documents
B5	How do you plan and schedule the fixing of security issues?	Mandatory: Free text & documents
B6	What security-related metrics do you collect and follow? Are there any metrics data you could share?	Mandatory: Free text & documents
B7	Do you use threat emulation? Can you share the results?	Mandatory: Yes/No Optional: Free text & documents
B8	Are you routinely following the vulnerability feeds of the third-party components included in the product? What feeds do you follow? What is the process to act on findings?	Mandatory: Free text & documents
B9	Do you do additional security testing on the third-party components you use? Have you found issues?	Mandatory: Time, Yes/No Optional: Free text & documents
B10	How confident you are about your software security process (range 1 to 5)?	Mandatory: Number Optional: Free text & documents

APPENDIX A

C: Vendor internal technical debt management process

Nr.	Query text	Answer type(s)
C1	Do you track the technical debt (TD) that is incorporated with the codebase?	Mandatory: Yes/No Optional: Free text & documents
C2	Do you differentiate the TD work with the rest of quality assurance?	Mandatory: Yes/No Optional: Free text & documents
C3	What mechanisms do you use for tracking the TD (for example error ticketing, backlog)?	Mandatory: Free text & documents
C4	Do you routinely use static source code analysis tools for assessing code quality?	Mandatory: Yes/No Optional: Free text & documents
C5	Do you routinely use architectural health analysis tools for assessing potential issues?	Mandatory: Yes/No Optional: Free text & documents
C6	How fast do you apply the new versions of the third-party component code? How do you manage if tooling, architectural or design changes are needed to incorporate the new versions?	Mandatory: Free text & documents
C7	How do you handle software obsolescence, or "code rot"—a situation when the underlying library or other dependency reaches end-of-life, or the needed platform support ends.	Mandatory: Free text & documents
C8	Do you assign TD with financial value (for example "the amount of work needed to fix the issue")	Mandatory: Yes/No Optional: Free text & documents
C9	Does your risk management process take TD into account? What other software related risks are incorporated?	Mandatory: Time, Yes/No Optional: Free text & documents
C10	How confident you are on your internal technical debt management process (range 1 to 5)?	Mandatory: Number Optional: Free text & documents

D: Customer system requirements and customization

Nr.	Query text	Answer type(s)
D1	Do you have non-functional security requirements on the system?	Mandatory: Yes/No Optional: Free text & documents
D2	Do you validate the effects of non-functional security requirements?	Mandatory: Yes/No Optional: Free text & documents
D3	Are there software security clauses that state SLA on fixing the issues on the contract?	Mandatory: Yes/No Optional: Free text & documents
D4	Are there requirements to notify, and fix the errors found on third party code included in the system?	Mandatory: Yes/No Optional: Free text & documents
D5	Will there be custom, additional R&D work on the product?	Mandatory: Yes/No Optional: Free text & documents
D6	Will there be any of your own in-house developed code running on the system? Can you share the process of managing this codebase?	Mandatory: Yes/No Optional: Free text & documents
D7	Will the system configuration settings default safely?	Mandatory: Yes/No Optional: Free text & documents
D8	Are there security-related instructions for the configuration?	Mandatory: Free text & documents
D9	Are there a security-related manual or user training available on the system?	Mandatory: Time, Yes/No Optional: Free text & documents
D10	How confident are you that the relevant security issues have been captured in the system requirements (range 1 to 5)?	Mandatory: Number Optional: Free text & documents

APPENDIX A

E: Customer capability on the security analysis

Nr.	Query text	Answer type(s)
E1	Are you able to do your own robustness testing on the product? Can you describe the process?	Mandatory: Yes/No Optional: Free text & documents
E2	Are you able to analyze the results of robustness testing findings? Can you elaborate on the process and resources?	Mandatory: Yes/No Optional: Free text & documents
E3	Do you have access to the product source code? Are you able to analyze and work on it?	Mandatory: Yes/No Optional: Free text & documents
E4	Are you able to send the system vendor technical error reports on any issues found?	Mandatory: Yes/No Optional: Free text & documents
E5	Are you planning to do technical security audits on the system?	Mandatory: Yes/No Optional: Free text & documents
E6	Have you done, or are you planning to do threat emulation, or threat analysis sessions regarding the system? How are you planning to use the results?	Mandatory: Yes/No Optional: Free text & documents
E7	How do you treat new versions of the system software?	Mandatory: Free text & documents
E8	Are you planning to do Software Composition Analysis (SCA) or Bill of Materials (BoM) analysis on the system software?	Mandatory: Yes/No Optional: Free text & documents
E9	Are you following vulnerability databases and able to cross-correlate with the system software composition?	Mandatory: Time, Yes/No Optional: Free text & documents
E10	How confident you are about the capability of doing your own analysis on the product security posture (range 1 to 5)?	Mandatory: Number Optional: Free text & documents

F: Customer capability on the secure operation of the system

Nr.	Query text	Answer type(s)
F1	Are you planning to train the staff who are going to operate the system?	Mandatory: Yes/No Optional: Free text & documents
F2	Does the training include the handling security-related configuration and incident management?	Mandatory: Yes/No Optional: Free text & documents
F3	How do you analyze the effects of the changes in the operating environment of the system (other systems, networks, way of using the system)?	Mandatory: Free text & documents
F4	What kind of usage metrics you are collecting of the use of the system? Is any metric related to security?	Mandatory: Free text & documents
F5	Do you have a Security Operations Centre (SOC) or similar, which will monitor the security status of the system operation?	Mandatory: Yes/No Optional: Free text & documents
F6	How are you planning to decide on, track and manage changes on the configuration of the system?	Mandatory: Free text & documents
F7	How are you going to secure the system physically?	Mandatory: Free text & documents
F8	Are you planning to engage in penetration testing activities to the system?	Mandatory: Yes/No Optional: Free text & documents
F9	How do you plan to decide on the end of life of the system?	Mandatory: Free text & documents
F10	How confident are you that the product is documented, trained, and operated in the way the operation is secure (range 1 to 5)?	Mandatory: Number Optional: Free text & documents