Design, Implementation, and Automation of a Risk Management Approach for Man-at-the-End Software Protection

Cataldo Basile[a,1], Bjorn De Sutter[b,1], Daniele Canavese[a], Leonardo Regano[a], Bart Coppens[b]

[a]*Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129, Torino, Italy*
[b]*Computer Systems Lab, Ghent University, Technologiepark-Zwijnaarde 126, 9052, Gent, Belgium*

Abstract

The last years have seen an increase in Man-at-the-End (MATE) attacks against software applications, both in number and severity. However, software protection, which aims at mitigating MATE attacks, is dominated by fuzzy concepts and security-through-obscurity. This paper presents a rationale for adopting and standardizing the protection of software as a risk management process according to the NIST SP800-39 approach. We examine the relevant constructs, models, and methods needed for formalizing and automating the activities in this process in the context of MATE software protection. We highlight the open issues that the research community still has to address. We discuss the benefits that such an approach can bring to all stakeholders. In addition, we present a Proof of Concept (PoC) decision support system that instantiates many of the discussed construct, models, and methods and automates many activities in the risk analysis methodology for the protection of software. Despite being a prototype, the PoC's validation with industry experts indicated that several aspects of the proposed risk management process can already be formalized and automated with our existing toolbox and that it can actually assist decision making in industrially relevant settings.

Keywords: Software protection, standardization, risk framing, risk assessment, risk mitigation

1. Introduction

In the Man-At-The-End (MATE) attack model, attackers have white-box access to the software. This means they have full control over the systems on which they identify successful attack vectors in their lab, for which they use all kinds of attacker tools such as simulators, debuggers, disassemblers, decompilers, etc. Their goal is to reverse engineer the software (e.g., to steal valuable secret algorithms or embedded cryptographic keys or to find vulnerabilities in the code), to tamper with the software (e.g., to bypass license checks or to cheat in games), or to execute it in unauthorized ways (e.g., run multiple copies in parallel). In general, MATE attacks target software to violate the security requirements of assets present in that software.

Email addresses: cataldo.basile@polito.it (Cataldo Basile), bjorn.desutter@ugent.be (Bjorn De Sutter), daniele.cavanese@polito.it (Daniele Canavese), leonardo.regano@polito.it (Leonardo Regano), bart.coppens@ugent.be (Bart Coppens)

[1]Cataldo Basile and Bjorn De Sutter share dual first authorship.

MATE Software Protection (SP) then refers to *protections deployed within that software* to mitigate MATE attacks.[2] SP is hence much narrower than the broad umbrella of software security. The latter also includes scenarios in which software is exploited to violate *security requirements of other system components*, e.g., infiltrating networks or escalating privileges. In such scenarios, attackers start with limited capabilities, such as having only unprivileged, remote access to a computer via a web server interface.

Because MATE attackers have full control over the systems in their labs, SP needs to defend assets in the software *without relying on external services and capability restrictions* that are provided by the platform on which the software normally runs. For example, whereas iOS and Android restrict the end user's debugging and app monitoring capabilities, MATE attackers have root privileges on their lab's workstations that can run customized Linux versions, custom debuggers, and other reverse engineering tools. MATE defenders can hence only rely on protections deployed within the protected software itself and possibly on remote servers under control of the defenders. Advances in cryptography have yielded techniques that can provide strong security guarantees from within an application itself, but they also introduce orders of magnitude performance overhead [57]. They are hence rarely practical today. Then again, practical SP is still dominated by fuzzy concepts and techniques [83]. SPs such as remote attestation, obfuscation, and anti-debugging do not aim to mitigate MATE attacks completely. Instead, they only aim to delay attacks and to put off potential attackers by increasing the expected cost of attacks and by decreasing the attackers' expected Return In Investment (ROI).

As observed during a recent Dagstuhl seminar on SP Decision Support and Evaluation Methodologies [44], the SP field is facing severe challenges: Security-through-Obscurity (StO) is omnipresent in the industry, SP tools and consultancy are expensive and opaque, there is no generally accepted method for evaluating SPs and SP tools. Moreover, SP tools are not deployed sufficiently [21, 26, 48, 70], and expertise is largely missing in software vendors to deploy (third-party) SP tools [52, 58, 79]. Moreover, we lack standardization. The National Institute of Standards and Technology (NIST) SP800-39 IT systems risk management standard [63] or the ISO27k framework for information risk management [59], which are deployed consistently in practice to secure corporate computer networks, have no counterpart or instance in the field of SP. Neither do we have concrete technical guidelines to implement General Data Protection Regulation (GDPR) compliance in applications.We can summarize the status of the SP domain as an industry with information system business needs involving so-called wicked problems [54]. The foundations and methodologies currently available in the SP knowledge base have not met those needs.

To plug gaps in this knowledge base, most existing SP research focuses on piece-wise, bottom-up extensions to its foundations and methodologies by presenting ever more novel SP artifacts and attack artifacts in an SP arms race. In that regard, existing offensive and defensive research fits into the information systems Design-Science Research (DSR) paradigm. Hevner et al. define this paradigm as research seeking to "extend the boundaries of human and organizational capabilities by creating new and innovative artifacts," and "create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished" [54].

However, to overcome the aforementioned shortcomings and to pave the road towards a standardized risk management approach and automated decision support for SP, we are of the

[2]For the sake of brevity, we will omit the MATE and simply use SP to mean MATE SP in the remainder of this paper.

opinion that such bottom-up DSR does not suffice. Instead it needs to be complemented with holistic, top-down DSR in which we study what an end-to-end SP risk management approach has to cover and what parts can and should ideally be automated. Our own research hence includes both the bottom-up and the top-down approach in the search for answers to the following research questions (RQs):

- **RQ1**: To what extent can automated decision support tools be useful for experts and/or non-experts by assisting them with the deployment of SPs and the use of SP tools?
- **RQ2**: To adopt a standardized risk management approach in the domain of SP, which constructs, models, and methods does the adopted approach need to entail, and which ones thereof should ideally be automated?
- **RQ3**: Which parts of such an approach can already be automated using decision support tools that instantiate the identified constructs, models, and methods?

RQ1 is formulated rather broadly, as being useful covers many different aspects such as usability, efficiency, correctness, comprehensibility, and acceptability by the users. Later in the paper this RQ will be refined according to those aspects. For answering RQ1, we developed a Proof of Concept (PoC) decision support tool for SP bottom-up, based on concrete requirements and needs from industrial partners of a European research project. Towards RQ2, we explored top-down how a standardized risk management approach can benefit the domain of SP. With the birds-eye view of such an approach, we identified existing work to build on and aspects that need more research and/or collaboration in the community. To ensure the relevance of our proposed design, we build on our experience in our academic SP research and past collaborations with the industry. That experience allows us to formulate the domain-specific requirements and to consider the relevant industrial Software Development Life Cycle (SDLC) requirements and practices. It also enables us to position existing domain-specific knowledge in the design. Finally, for formulating a partial, lower bound answer to RQ3 we identified which artifacts from our answer to RQ2 are already instantiated and automated in our PoC tool.

This paper reports our research findings and presents our answers to the RQs with the following contributions. First, we provide a rationale for adopting and standardizing risk management processes for SP. We discuss several observations on the failing SP market and we analyse why existing standards are not applicable as is for SP. Where useful, we also highlight differences between SP and other security fields such as cryptography, network security, and software security.

Secondly, we discuss in depth how to adopt the NIST risk management approach. We identify which artifacts in the forms of constructs, models, methods, and instantiations (i.e., (semi-)automated tools) we consider necessary and feasible to introduce and deploy the NIST risk management approach for SP. For all the required processes, we highlight (i) the current status; (ii) SP-specific concepts/artifacts to be covered; (iii) what existing parts can be borrowed from other fields; (iv) open questions and challenges that require further research; (v) needs for the research community and industry to come together to define standards; and (vi) relevant aspects towards formalizing and automating the processes.

Finally, we demonstrate that several aspects can already be formalized and automated by presenting a PoC decision support system that automates some of the major risk management activities. Even if not completely automated, this demonstrates that the more abstract constructs, models, and methods we discuss can indeed be instantiated concretely. This PoC provides a starting point for protecting applications and for building a more advanced system that follows all the methodological aspects of a NIST 800-compliant standard with industrial-grade maturity.

The first results obtained with the tool have been validated mostly positively by industry experts on Android mobile app case studies of real-world complexity and are presented according to the Framework for Evaluation of Design Science [104].

The remainder is structured as follows. Section 2 presents our research approach. Section 3 provides background information on standardization and the state of the field of SP. Section 4 provides a motivation for standardization, formalization, and automation and discusses challenges towards them. Section 5 discusses how to adopt the four phases of the NIST IT systems risk management standard for MATE SP. Section 6 presents the PoC decision support system we designed, and Section 7 presents its evaluation. Section 8 draws conclusions and discusses future work directions.

2. Research Approach

Beyond their quotes in the previous section, Hevner et al. describe DSR as "achieving knowledge and understanding of a problem domain and its solution by the building and application of designed artifacts" [54]. For doing so in the SP risk management problem domain, in particular for answering the three RQs, we followed up on the DSR guideline of *design as an artefact* [54] by collecting, structuring, designing, building, and applying a large set of related artifacts. We did so in the three research steps shown in Figure 1.

First, in the collaborative European ASPIRE research project, we designed, developed, and evaluated a largely automated PoC called Expert system for Software Protection (ESP). This bottom-up research was driven by the industrial partners' business needs and SDLC requirements. Section 6 presents the PoC, of which Section 7 presents the evaluation.

Second, we studied how to adopt a standardized IT risk management approach, the NIST SP800-39 standard, in the domain of SP. This was driven by our observations of the state of the domain of SP as discussed in Section 3 and the motivation presented in Section 4. The result of this study is the approach presented in Section 5.

Third, we analyzed which of the constructs, models, and methods required in the adopted approach are actually covered by the automated tool support in the ESP. The result is a mapping between the artifacts introduced in Section 5 and those discussed in Section 6.

With the design, implementation, and evaluation of all the instantiation artifacts (which are available as open-source), as well as with our study and the development of the standard-based approach for MATE risk management and the discussion of more abstracts artifacts that constitute that approach, we added *design artifacts*, as well as *foundations*, and *methodologies* to the scientific knowledge base, in accordance with the DSR guideline on *research contributions* [54].

We now discuss our approach in the aforementioned three steps in more detail.

2.1. Step 1: Bottom-up Development and Evaluation of Proof-of-Concept Decision Support Tools

Our research into SP decision support intensified in the 2013–2016 European ASPIRE FP7 research project[3] in which we collaborated with three SP companies: Nagravision with a focus on DRM, Gemalto (now Thales) with a focus on software trusted execution environments and SafeNet (now Thales) with a focus on software license management. The project researched a layered SP toolchain for mobile apps and corresponding (semi-)automated decision support methods and tools. The companies identified the lack of such automated support tools as a critical,

[3] https://aspire-fp7.eu/

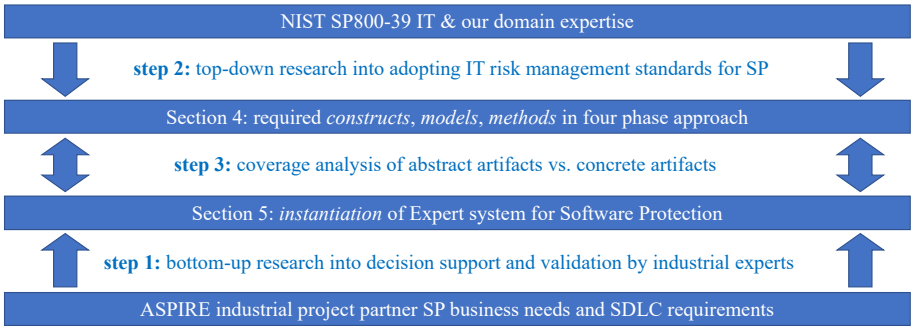


Figure 1: Three research steps leading to the results presented in this paper.

foundational gap in the SP knowledge base that hampered the effective and efficient deployment of SP in practice. In the traditions of DSR, we endeavoured to close this gap by researching the design and development of novel artifacts, including proof-of-concept tools. The companies and their technical and commercial SP business needs drove the project's requirements analysis and scope determination, as well as the considered attack model. In technical meetings, we engaged with their stakeholders and experts in SP, including software developers, SP tool developers and users, security architects, and penetration testers. We engaged with higher management in advisory board meetings. The insights obtained there drove the development of decision support techniques in a bottom-up fashion during the ASPIRE project, i.e., starting from concrete SP problems and business requirements and solutions, as well as existing, mostly informally described best practices[4]. By having our research driven by the companies needs, we acted according to the DSR guideline of *problem relevance* [54]. As we were, to the best of our knowledge, the first project to research largely automated end-to-end decision support tools, conforming to existing standards was at that time not at all a requirement or concern.

Through the SP tool flow and decision support developed in the project, we provide an answer to RQ1, demonstrating that automated decision support that effectively assists experts, and possibly non-experts, may be within reach. That evidence in the form of artifacts and their evaluation is presented later in this paper.

We performed and present that evaluation using the Framework For Evaluation of Design Science (FEDS) [104], the taxonomy of evaluation methods for IS artifacts by Prat *et al.* [89], and the evaluation criteria and terminology by Cleven *et al.* [36].

Our *ex-post* evaluation from an *engineering perspective* focused mainly on *human risk and effectiveness*, as the aim was to determine whether the artifacts consisting of our PoC decision support tool and all the data it generates are accepted by the involved users and whether it benefits their work. We focused on properties compliant with the ISO/IEC 9126-1:2001 criteria[5], namely usability, efficiency, correctness, and comprehensibility and acceptability by the users. As Section 7.1 will describe in much more detail, we organized the evaluation in multiple iterative steps to gather timely feedback, and we gradually involved more external experts. Initially, a qualitative assessment of the automatic decision support prototype (and of the artifacts it used) was performed with three industrial experts working on the ASPIRE project. The objective was to

[4]Unfortunately, many documents that formalized and structured those insights in the ASPIRE project are confidential.
[5]https://www.iso.org/standard/22749.html

improve the early versions and components iteratively, before the release of the final prototype. This back and forth between design and evaluation clearly implements the *design as a search process* guideline of DSR [54]. For this evaluation, three mobile applications provided by the industrial partners have been used as reference scenarios: a media streaming app, a licence checker, and a one-time password generator. These were developed by the partners to be representative of their actual business applications. A second qualitative assessment was performed on the final PoC, for which we involved two additional industrial experts that had not participated in the development. Moreover, they evaluated the performance of the algorithms and techniques used in the PoC on both the reference use cases and artificial applications, with measurements and with a complexity analysis.

By relying on industrial experts as subject groups, by having them deploy the artifacts on use cases representative for real-world cases, and by focusing the evaluation on aforementioned standardized criteria, we aimed to meet the requirements of the DSR *research rigor* guideline [54].

In our evaluation, we also questioned the potential use of the developed tools by non-experts. This allowed us to identify required knowledge that experts have to deploy the tools effectively and that non-experts might be lacking, as will be reported in Section 7. This reporting is part of the *communication of research* DSR guideline [54], in particular the part on communication to management-oriented audiences.

In addition, we performed a *purely technical artifact* assessment to verify that the tool provides solutions in a useful time with reasonable use of resources.

Combined, we deployed *observational* (*case study*), *analytical* (*dynamic analysis*), *experimental* (*simulation*), and *descriptive* (*informed argument*) evaluation methods to implement the *design evaluation* guideline of DSR [54].

2.2. Step 2: Top-Down Adoption of a Standardized IT Risk Management Approach

After the ASPIRE project had formally finished, we continued our collaboration and gradually developed our vision that the best way to approach decision support is from the perspective of information risk management approaches. This vision first manifested itself in the July 2019 Ph.D. thesis of Leonardo Regano [90] that presents the components of the ESP and that is structured according to the phases of information risk management standards.

We reached out to other researchers and practitioners in the SP domain to gather their opinions and insights, as well as doubts on decision support for SP. This happened in informal discussions but also in structured ones, including the August 2019 Dagstuhl seminar on Software Protection Decision Support and Evaluation Methodologies [44], of which B. De Sutter was the main organizer. In this one-week seminar, the three senior authors of this paper engaged again with a range of experts, including, amongst others, SP researchers, security economists, reverse engineering practitioners, software analysis experts, and commercial SP developers. During the seminar, the need for standardization came to the forefront, if not formalized, then at least in terms of best practices and guidelines for conducting research into SP and evaluating the strength of proposed SPs and attacks thereon.

Following that seminar, we invested in a top-down approach, studying the adoption of existing information and system security standards in the domain of SP. We investigated how the generic concepts that make up these standards are specialized and adopted in specific security domains such as network security. We then extensively brainstormed about how they can also be specialized and adopted in the domain of SP. For example, the NIST SP800-39 IT systems risk management standard [63] prescribes a top-level method consisting of four generic risk management phases,

each corresponding to their own, conceptually formulated, abstract method. We studied the domain-specific organizational problems that need to be solved in the different phases, and which more concrete domain-specific concepts those phases need to encompass for SP.

We started this research by collecting our combined insights, then structuring them, and then iteratively coming to the text of Section 5 that, in essence, presents a top-level SP risk assessment method and the necessary artifacts for using that method, thus providing our answer to RQ2.

We want to thank the reviewers of earlier versions of this text for their valuable insights that helped us produce the final result.

Our iterative process for coming to our description of the approach is again illustrative of how we implemented the *design as a search process* DSR guideline [54]. The presented approach is itself an artifact, in line with the *design as an artifact* guideline. Moreover, by rooting our work in interactions with the wide variety of stakeholders mentioned above, by building on existing standards, and by extensively discussing existing work that can be built upon, we further implemented the *problem relevance* and *research rigor* guidelines.

During our research, and in the resulting description of the approach in Section 5, we also discussed open research questions with potential interesting future research directions, as well as open standardization issues into which collaborative community effort should be invested. These discussions are particularly relevant for a management-oriented audience. With those discussions, as well as by constructing and expressing the approach in line with existing risk management standards and in terms of abstract, conceptual artifacts, we hope that this paper not only serves a technology-oriented audience, but also management-oriented audience, in line with the DSR guideline on *communication of research* [54].

Importantly, while this research step was rooted in our previous experience with SP, we tried to perform this study as independently as possible from the PoC results of the ASPIRE project. This shows, amongst others, in the fact that in Section 5, we put forward about 40% more concepts to be included in the proposed standard approach adoption than are covered in the PoC results we present in Section 6. As a concrete example, we discuss the organizational problem of SP tool vendors and their customers not giving each other white-box access because they do not trust each other in Section 5.1.5. That problem was out of scope in the ASPIRE project and is hence not tackled by the presented PoC tools.

2.3. Step 3: Coverage Analysis of the Adopted Approach in the DSR Framework

An important consideration in our study in step 2 was the need for automation, as reflected in the last part of RQ2 and in RQ3. In later sections, we argue in more detail why we consider automation of many of the adopted and specialized methods beneficial, if not crucial.

Our answer to RQ3 is not based on theoretical analysis and abstract reasoning but on tangible evidence, i.e., the existence of the concrete artifacts that form the PoC developed in step 1. To provide this answer to RQ3, we organized numerous internal discussions in which we analyzed which of the concepts from the different phases of the proposed approach are instantiated by the automated components of our PoC.

To do this more methodologically, we adopted the DSR framework by Hevner et al. [54], in particularly focusing on their *design as an artifact* guideline. First, we rephrased the adopted approach such that all essential concepts of the approach's four phases are clearly identified as either constructs (vocabulary to define and communicate concrete SP cases), models (abstractions and representations to aid case understanding and to link case features and solution components to enable exploration), and methods (algorithms, practices, and processes, as well as guidance

on how to tackle concrete cases). These are the three abstract types of artifacts that Hevner et al. identify as foundational elements of an information systems knowledge base, in this case the SP knowledge base.

Next, we identified which of these abstract artifacts are instantiated by means of components of the PoC ESP. Such implementations are called instantiation artifacts by Hevner et al. They form the fourth type of foundational element in a knowledge base.

The mapping from more abstracts DSR artifacts onto concrete PoC instantiation artifacts is documented in this paper by means of recurring tags. The tags are introduced in Section 5 when the constructs, models, and method artifacts are first introduced, and they recur in Section 6 where the corresponding instantiations are discussed.

3. Background on Standardization and the State of Software Protection

We first discuss some risk management standards and how they have been adopted in other security domains, such as network security, and the healthy market for products and services that exists there as a result. We then contrast this with the lack of such a market and standards for SP.

3.1. Standardized Risk Management Approaches

Protecting software can be seen as a risk management process, a customary activity in various industries such as finance, pharmaceutics, infrastructure, and Information Technology (IT). The NIST has proposed an IT systems risk management standard that identifies four main phases [63]:

1. *risk framing*: to establish the scenario in which the risk must be managed;
2. *risk assessment*: to identify threats against the system assets, vulnerabilities of the system, the harm that may occur if those are exploited, and the likelihood thereof;
3. *risk mitigation*: to determine and implement appropriate actions to mitigate the risks;
4. *risk monitoring*: to verify that the implemented actions effectively mitigate the risks.

The ISO27k framework also focuses on information risk management in three phases [59]:

1. *identify risk* to identify the main threats and vulnerabilities that loom over assets;
2. *evaluate risk* to estimate the impact of the consequences of the risks;
3. *treat risk* to mitigate the risks that can be neither accepted nor avoided.

ISO27k adds an explicit operational phase for handling changes that happen in the framed scenario.

Those approaches have been consistently applied in practice for securing corporate networks. Regulations stimulate companies to analyse the risks against their IT systems. For instance, the GDPR explicitly requires a risk analysis of all private data handling. Companies invest in compliance with the ISO27k family to obtain market access. Consequently, risk analysis of networks has developed a common vocabulary, and a company's tasks have been properly identified and often standardized, so offerings from consultancy firms can be compared easily. There is a business market related to this task, best practices, and big consultant firms have risk analysis of corporate networks in their catalogs [51].

In the domain of software security, several frameworks for risk analysis and decision support exist that mainly focus on Software Vulnerability Management [45] and Enterprise Patch Management [96]. Other frameworks focus on quality assurance best practices and benchmarking,

including the OWASP Software Assurance Maturity Model (SAMM) [86], the OWASP Application Security Verification Standard (ASVS) [85], and the Building Security in Maturity Model (BSIMM) [27]. These address problems of software security and are not applicable to SP.

NIST SP800-53 [62] extends beyond software security and provides a comprehensive and flexible catalog of privacy and security controls for systems and organizations as part of their organizational risk mitigation strategy, for which they build on NIST SP800-39 [63]. It targets whole IT infrastructures, including hardware and software. Regarding software, it advises to "Employ anti-tamper technologies, tools, and techniques throughout the system development life cycle" in its SR-9 Supply Chain Risk Management family of controls. Obfuscation is mentioned only as an option to strengthen the tamper protection, not to protect the original software. The document does not discuss how to deploy these protections, or how to select the ones to deploy. NIST SP800-53 is hence not applicable to SP. For much of the remainder of this paper, we will actually discuss what an SP counterpart of NIST SP800-53 needs to entail.

3.2. The State of MATE Software Protection

Compared to network security and software security, SP has years of delay. For setting the scope, Table 1 lists a number of well-known SPs. Out-of-scope are mitigations to prevent the exploitation of vulnerabilities, such as Address Space Layout Randomization (ASLR), compartmentalization techniques, or safe programming language features in, e.g., Rust. In the MATE attack model, attackers have full control over the devices on which they attack the software. They can disable security features of the operating system and the run-time environment, such as ASLR, which therefore cannot be trusted. For that reason, SP centers around protections embedded in the software itself, rather than relying on the security provided by the run-time environment.

The market of such SP is neither open nor accessible to companies with a small budget. In 2017 Gartner projected that 30% of enterprises would have used SP to protect at least one of their mobile, IoT, and JavaScript critical applications in 2020 [117]. However, two years later Arxan reported that 97% (and 100% of financial institutions) of the top 100 mobile apps are easy to decompile as they lack binary code protection or implement weak protection [70]. A study confirms the absence of both anti-debugging and anti-tampering protections for 59% of about 38k Play Store apps. The study highlights that weak Java-based methods are employed in 99% of the SP uses [21]. Repackaging benign apps to obtain malicious apps [68, 116] is easy because of the intrinsically weak app packaging process but also because used anti-repackaging protections are currently weak [82]. Furthermore, it is estimated that 37% of installed software is not licensed, for a total amount of losses estimated at $46.3B in 2015–2017 [26]. Consequently, the SP market, which accounted for $365.4M dollars in 2018, is expected to grow fast [4].

Cybersecurity competences are lacking [52]. SP is no exception. Few companies have internal SP teams: only 7% of respondents stated their organization has all it needs to tackle cybersecurity challenges; 46% stated they need additional expertise/skills to address all aspects of cybersecurity [58]. Meanwhile, many organizations lack competent staff, budget, or resources [79].

When the value of assets justifies it, developers resort to paying third parties to protect their software. The price is typically high, involving licenses to tools and often access to expert consultants. Moreover, the services and the strength of the obtained SP are covered by a cloak of opaqueness, with StO omnipresent[6]. For example, whereas early white-box cryptography

[6]Abandoning StO implies that transparency is given about the SP process, the design and implementation of all SP tools being used, including the supported SPs and decision support tools. It does not at all imply that SP users need to be

protection type	explanation
anti-debugging	Techniques to detect or prevent the attachment of an attacker's debugger [8].
branch functions	Indirect, computed jumps replace direct control transfers to prevent reconstruction of control flow graphs [73].
call stack checks	Checks if functions are called from allowed callers to block out-of-context calls.
code mobility	Code is lifted from the binary to prevent static analysis. At run time, the code is downloaded into the running app from a server [29].
code virtualization	Code in the native instruction set is replaced by bytecode and an injected interpreter interprets that bytecode, of which the format is diversified [12].
control flow flattening	A structured control flow graph graph is replaced by a dispatcher that transfers control to any of the original nodes based on data. This makes it harder to comprehend the original flow of control and the code [106].
data obfuscation	Transformations that alter data values and structures to hide the original ones.
opaque predicates	Logic that evaluates to true/false based on invariants known at protection time but that are hard to discover by an attacker [39]. This enables inserting bogus control flow to hinder code comprehension and precise analysis [23, 102].
remote attestation	Techniques in which a remote server sends attestation requests to a running program. If the program fails to deliver valid proof of integrity, it is considered to be tampered with, and an appropriate reaction can be triggered [105].
white-box crypto	Implementations of cryptographic primitives such that even white-box access to the run-time program state does not reveal the used keys [111].

Table 1: A number of software protections.

schemes were peer reviewed [22, 35] and then broken [43, 114], we could not find peer-reviewed analyses of schemes currently marketed by big vendors. Moreover, most vendors' licenses forbid the publication of reverse engineering and pen testing reports on their products. They do not share their internal procedures, tools, or reports with academics.

We deduce that many companies do not understand the risk and therefore do not feel the need for deploying SP, or they do not have the internal competences and knowledge to do so properly, or they lack the money to pay third-party providers. In short, there exists no widely accessible, functional, transparent, open SP market. At some of the big SP vendors that are also active in other security fields, risk analysis and mitigation is most certainly the principle that drives their experts and that is encoded in policies. Yet no methodology is publicly available for applying a risk analysis process when deciding how to protect software. Needless to say, no standard process guarantees the proper selection and application of available SPs given a case at hand.

4. Motivation and Challenges for Standardization, Formalization, and Automation

This section first motivates why we strive for standardization. Next, it argues why formalization and automation are (equally) important. The section concludes with a discussion of some challenges towards these objectives, thus complementing the background provided above.

transparent about the applications they protect. Indeed, the very objective of using SP to hamper MATE attacks on assets with confidentiality requirements is to keep those assets obscured. This is to be achieved by keeping the unprotected code secret, and by keeping the used tool configuration secret, not by hiding the used tools or evaluations of their effectiveness.

4.1. Motivation for Standardization

Standardization efforts aim at "striking a balance between users' requirements, the technological possibilities and associated costs of producers, and constraints imposed by the government for the benefit of society in general" [98]. The benefits come from the positive impact of standards on quality/reliability, information standards, compatibility/interoperability, and variety reduction [98]. In line with those benefits, a standardized, methodological approach to MATE risk analysis could have a plethora of benefits. This section speculates on this potential.

First, it could force stakeholders to follow a more rigorous approach to SP. Risk framing forces analysts to define workflows, processes, methods, and formulas to evaluate risks and the impact of mitigations. In network security, a structured risk analysis has limited the impact of subjective judgments by suggesting the use of collegial decisions involving more roles [63]. A more rigorous approach for SP could similarly increase the transparency of all phases, guaranteeing a more reliable estimation of the reached SP level and of the quality delivered by third parties. In turn, we expect less reliance on StO. Simply adopting the OWASP Security Design Principles forces security specialists to avoid StO, which is also considered a weakness in MITRE CWE 656 [1].

A standard could induce the community to use well-defined terminology and to agree on the meaning of each term, as happened after NIST SP 800 [63]. Building common ground and well-defined playing rules would also benefit the SP market by creating a more open and transparent ecosystem where services can be compared as normal products, thus bridging the gap with the network security market in which products are evaluated by third parties using standardized methods such as the Gartner Magic Quadrant for Network Firewalls [2]. Hence, we expect the rise of consultancy firms that can independently evaluate SP effectiveness. We also expect a price reduction, as highlighted in a study [4]. With a lower entry price and the definition of entry-level protection services, more companies can then afford professional SP services, with benefits for all the stakeholders.

When SP becomes standardized and more clearly defined, it could also create a market for decision support products that automate risk management. This could in turn lead to cost savings and to more accessible and more effective SP.

The availability of standards increases awareness, as reported by an EU agency one year after adopting the GDPR [50]. The mere existence of a standard would initially inform people about the need for SP. Compliance would then force all parties to obtain in-depth knowledge, and the standards and related best practices would eventually be incorporated into educational programs.

The work towards standards could also impact research. It could initially stimulate the community to focus on identifying and plugging existing gaps and, later on, create new or more effective, validated SPs to be integrated into a standard framework. The interest in the field and the impact of research results would then likely attract more researchers to the SP field, which is now marginal in the software engineering community. We have found analogies with the impact of the ISO/SAE 21434 standard for cybersecurity engineering of road vehicles [78]. Years before its adoption, car manufacturers anticipated effort and funded research to cope with the demanding standard. The investments in automotive cybersecurity will grow from $4.9B in 2020 to $9.7B in 2030, with a market size expected to grow from $238B in 2020 to $469B in 2030 [28]. Parts of this increase and of the focal shift towards cybersecurity might not be caused directly by the ISO/SAE 21434 standardization. However, we are convinced that the planned standardization was a major contributing factor in the past years, given that compliance with the standard as part of UN R155 has already become mandatory in Europe, Japan, and Korea since July 2022. The anticipation of the standard can also be observed in guidelines published long

before its finalization, such as in the "ENISA good practices for security of Smart Cars" published in November 2019 with contributions of major carmakers [5].

Increased attention by research institutions and academia usually translates into better education opportunities, possibly with dedicated curricula, which usually pair well with the career opportunities created by a more open market. Ultimately this could help companies employ skilled people and support a freer job market to compensate at least partially for the lack of SP experts.

In the end, the benefit would extend to the whole society, as having better-protected software reduces the global exposure of citizens to risks and, we hope, would make MATE attacks a less lucrative field, or at least reduce its growth.

4.2. Motivation for Formalization and Automation

A standardized, methodological risk management approach is not necessarily formalized or automated. We argue, however, that formalization and automation are by and large required. The main reason is the need for precision, i.e., the repeatability or reproducibility of obtained results.

In the security field, including SP, we want to avoid a scenario in which different experts that deploy the same risk management approach on the same software under the same conditions would come up with different sets of identified threats and different sets of supposedly good combinations of protections. One of the more important reasons to stay clear of such a scenario is that it would complicate the validation and enforcement of compliance.

Cognitive psychology research has shown, however, that humans are incorrigibly inconsistent in making summary judgments based on complex information [64, 84]. Hence they provide different answers when asked to evaluate the same information multiple times. Experts also suffer from this. Their judgments hence lack precision in environments that are not sufficiently regular to be predictable [65, 64]. Those environments are also known as low-validity environments. Determining the major MATE attack threats on a given piece of software given the source code, the formulated security requirements, the domain knowledge, etc., as well as selecting appropriate combinations of SPs come down to making predictions in such an environment. One of the reasons is that there are many parameters one cannot think of in advance, such as the configurations with which the final software will be deployed on-site. Psychology research has also shown that the precision of expert judgment improves when there exists backup in the form of formulas and algorithms to complement, guide, or replace otherwise imprecise human cognitive processes [42]. We hence put forward formalization and automation as important objectives for MATE risk management.

We are not the first ones to do so. For example, in their survey on architectural threat analysis, Tuma et al. analyse whether the surveyed methods are supported by formal frameworks and by (semi-)automated tools because of their impact on precision [101]. They also differentiate between template-based approaches and example-based ones, as the former yield higher precision. Similarly, we put forward that using an unambiguous vocabulary with clear definitions will benefit the precision of MATE risk management.

Economic arguments further support our claim that automation cannot be separated from the aim of adopting a risk analysis process for SP. Manual SP decision making requires expertise, effort, and hence time. As we discussed, there are not enough experts to protect all software that can benefit from rigorous SP. Even if enough experts were available to put in the necessary manual effort, they would remain costly, keeping good SP out of reach for SMEs.

Scaling up the number of experts to meet all demands without automating parts of the processes is not realistic. Every time a new version of an application is issued (e.g., because of regular

updates or a bug), it needs to be protected. Part of the work on previous versions can probably be reused, but typically the SPs at least need to be diversified.

Additionally, SP firms may have to protect many versions, such as ports of the same software to different platforms, including laptops or mobiles with limited computational power. If maintaining the application's usability is at risk on some platforms because of the SP overhead, developers may decide to limit the features on those platforms. As an example, media players with DRM will only access low-quality versions of media if the platform does not allow full protection.

Moreover, even if human experts were available, their latency would still be problematic. Software vendors face time-to-market pressure. For that reason alone, automated tool support that can cut the time and effort required to protect applications is beneficial.

4.3. Challenges towards Standardization, Formalization, and Automation

Despite the many benefits a standardized, formalized, and automated approach would bring, such an approach is a long way off, and adopting a NIST-style risk management faces several challenges.

A first challenge relates to the definition of asset categories and their relation with security properties. These are lacking today, which is problematic for the framing of risks. SP relies on the MATE attacker model that has never been defined clearly. The abilities of MATE attackers are unclear, not in the least because of the complexity of modelling human code comprehension and software tampering capabilities.

A second challenge is the definition of threat and risk assessment models that allow enough precision and objectivity. Estimating the feasibility of MATE attacks requires a white-box analysis of the assets and of the entire application. The complexity of mounting static, dynamic, symbolic, and concolic attacks heavily depends on the structure and artifacts of the software, such as the occurrence of all kinds of patterns or observable invariants.

Thirdly, moving towards a more precise categorization of protections and risk in the MATE scenario is another challenge that needs to be overcome for the risk mitigation phase. In practice, SP provides only fuzzy forms of protection. SPs have only been categorized coarsely (e.g., obfuscation vs. anti-tampering). In general, it is not clear what security level they offer where, and there yet exists no well-defined set of categories of security controls to mitigate MATE risks. This contrasts with, e.g., the field of cryptography, in which algorithms are characterized in terms of well-defined properties such as ciphertext indistinguishability or second pre-image resistance [66]. Also in network security, it is clear what firewalls and VPNs do and how to use them to mitigate network security risks. There are accepted measures and guidelines to estimate the effectiveness of categories of network security mitigations and in some cases categorization of tools and vendors that help in estimating their efficacy [60]. The MATE domain lacks such well-definedness.

Fourthly, today it remains a huge challenge to simply measure or estimate the efficacy of SPs. This is obviously necessary to assess the residual risks of deployed SPs. However, no metrics are currently available to quantify SP efficacy. Potency, resilience, and stealth are commonly accepted criteria [38], but no standardized metrics are available for measuring them. Complexity metrics originating from the field of software engineering have been proposed [31], and ad-hoc metrics are used in academic papers [23, 73]. However, none have been empirically validated for use in SP, and practitioners most often do not see the metrics used in academic papers as reliable proxies of real-world potency, resilience, or stealth. Using those metrics is hence not yet considered a viable replacement for human expertise and manual pen testing. In many cases, there are no hard proofs that SPs are effective in delaying attackers. Rather than encouraging checks by external

parties, SP vendors often contractually prevent the analysis of protected code, instead relying on StO. As a result, there is neither an objective nor a measurable assurance of protection, nor an objective evaluation of the companies' work. In academic research, the situation is not much better. For example, the seminal obfuscation versus analysis survey from Schrittwieser et al. never refers to a risk analysis framework [94]. Their results, although widely acknowledged, are hence not readily usable in a decision process.

The aforementioned challenges are particularly hard because in SP, determining the boundaries between assets and protections is no easy task. SPs are often processes that transform assets to hinder analysis and comprehension of their logic [94]. For instance, most forms of obfuscation transform code fragments. Since SPs need to be layered for stronger and mutual protection and to exploit synergies, obfuscation can transform code that results from previous transformations, such as code guards injected for anti-tampering purposes. Some obfuscations even aim for eliminating recognizable boundaries between different components [23], and others aim for re-using application code for obfuscations [102]. As a result, the code of multiple SPs and of the assets they protect becomes highly interwoven. We hence need to talk of protected assets, certainly not of separated protection and asset entities.

Furthermore, software internals must be known to the tools. This includes the types of instructions, structure and semantics of the code, and the presence of any artifacts that might benefit attackers. This information is needed to decide whether some (layered) SP can be effective or not and to tune its parameters. In addition, it is generally accepted that in order to deploy SPs effectively, an application's architecture needs to be designed with the protection of the sensitive assets in mind. If it is not designed well, SPs will only provide superficial mitigation. For theoretical definitions of SP, such as virtual black-box obfuscation, Barak already proved the impossibility of achieving obfuscation on contrived programs [16]. But also in practice, architectural weaknesses can often not be overcome with SP. Examples of design problems that are hard, if not impossible, to fix with SPs are bad external or internal APIs, missing authorization, and improper or missing crypto key ladders to protect various assets. Such ladders require complex key management, key storage, and crypto functionality, which are easy to get wrong for non-experts. Risk assessment methods must hence recognize software whose design prevents proper protection and report that risks cannot be reduced to the desired level solely with SPs. This again stresses that MATE risk analysis requires insights into software internals to identify weaknesses that may turn into vulnerabilities that cannot be protected with SP.

SP thus poses challenges that impact the standardization and automation of risk management, and, in particular, the definition of objective criteria for assessing the mitigations.

In conclusion, despite their obvious appeal, risk management standardization and a functioning open market as they exist in other areas of ICT security are in our opinion missing in SP not only because the community is late in developing them, but also because managing the risks in SP is really challenging.

5. Adopting a Standard Towards Proper Risk Management

This section provides an answer to RQ2 by discussing what the four phases of the NIST IT systems risk management standard would entail as applied to SP, i.e., what tasks need to be done in its four phases. Figure 2 presents an overview. Note how the tasks flow quite naturally, each task building on the previous ones. The discussion of these tasks will cover various recurring aspects, which are highlighted by means of numbered text markings. We introduce

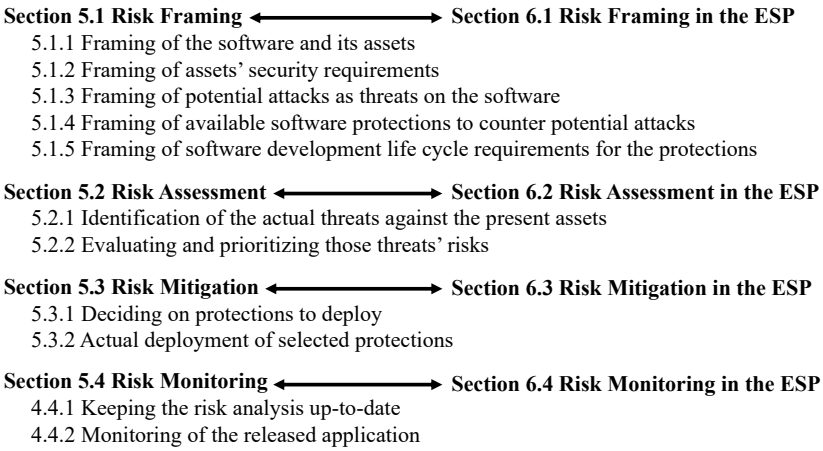


Figure 2: Four phases of the proposed risk management approach with reference to the corresponding sections in the presentation of the approach in Section 5) and in the presentation of the PoC implementation in Section 6.

the necessary c.X constructs[7], M.X models, and m.X methods/practices, introducing some useful new terminology along the way.

Tables 2, 3, and 4 present an overview of the covered abstract artifacts. For those artifacts that have already been implemented in an actual instantiation, the ESP column lists the subsections of Section 6 in which that instantiation will be discussed in more detail. Those instantiations will demonstrate that these artifacts can in fact be implemented in a working system. They will hence demonstrate the feasibility of the covered artifacts, thus also enabling a concrete assessment of their suitability for their intended purpose, as will be discussed in Section 7.

We also highlight ?.X open issues that are research challenges and discuss where we think s.X existing state of the art can serve as a foundation, in some cases by pointing out !.X potentially useful research directions to find solutions. We present r.X recommendations and requirements, in particular a.X automation requirements, and we highlight aspects on which different stakeholders need to perform f.X future standardization and engineering work (as opposed to research).

5.1. Risk Framing

In this phase of the approach, one defines the context in which a risk analysis will be performed. For the case at hand, one defines the relevant software targets, their assets and security requirements, potential attacks, available SPs, and SDLC requirements. To enable standardization, a common vocabulary needs to be established that covers all possible constructs and models to describe all relevant scenarios. This needs to be unambiguous and formalized such that automated support tools can be engineered. f.1 Provisioning the complete vocabulary to describe the risk frame is, of course, out of reach here. That will instead need to be done in a larger document that results from a community effort. s.1 The meta-model of Basile et al. can serve as a starting point for modelling all the relevant constructs and their relations [19].

[7] In most cases, we identify only the abstract top-level constructs, under which more concrete constructs have to be included as well. For example, we will mention the "software protection" construct, without enumerating concrete protections such as opaque predicates, control flow flattening, virtualization, etc.

No.	Construct Name	ESP	No.	Construct Name	ESP
c.1	primary asset	6.1	c.26	protection applicability	6.1
c.2	secondary asset	6.1	c.27	protection composability	6.1,6.3
c.3	attack path	6.2	c.28	layered protection deployment	6.1,6.3
c.4	attack step	6.1,6.3	c.29	protection synergies	6.1,6.2
c.5	attack pivot		c.30	potency	6.1,6.3
c.6	attack time frame		c.31	resilience	6.1
c.7	asset renewability		c.32	stealth	6.1
c.8	primary security req.	6.1	c.33	overhead/cost constraints	6.1,6.3
c.9	non-functional security req.	6.1	c.34	software development life cycle req.	6.1
c.10	attack identification phase		c.35	profile information	
c.11	attack exploitation phase		c.36	software connectivity	6.1
c.12	secondary security req.	6.1	c.37	software update ability	
c.13	functional security req.		c.38	environment limitations	6.1
c.14	assurance security req.	6.1	c.39	actual threats	6.2
c.15	protection policy req.		c.40	actual risks	6.2
c.16	weaknesses		c.41	attack surface	6.2
c.17	attack resources	6.1	c.42	attack vectors	6.2
c.18	attack capabilities	6.1	c.43	attack paths of least resistance	
c.19	worst-case scenario assumptions		c.44	analysis tools / toolbox	6.1,6.2
c.20	attack enabling features	6.1	c.45	software features	6.2
c.21	attack preventing features	6.1	c.46	third-party-provided incomplete analysis	
c.22	attack effort determination features	6.1,6.2	c.47	residual risks	6.3
c.23	attack likelihood of success features	6.1	c.48	most protective protection solution	6.3.1
c.24	software protections	6.1	c.49	alternative protection targets	6.3.1
c.25	protection strength metrics	6.1,6.3	c.50	mitigation round	

Table 2: Constructs of the proposed approach, with references to the discussions of their instantiation, if any.

No.	Model Name	ESP	No.	Model Name	ESP
M.1	application and asset model	6.1	M.4	attack model	6.1
M.2	secondary asset attributes model	6.1	M.5	software protection model	6.1
M.3	asset value evolution model		M.6	actual threat model	6.1

Table 3: Models required in the proposed approach, with references to the discussions of their instantiation, if any.

5.1.1. Assets

A first task for a case at hand is to determine which assets are *potentially* relevant. This is needed for all the potential assets known a priori, i.e., in the original application, in already deployed SPs, if any, or in any of the SPs that might later be deployed in the mitigation phase.

The c.1 primary assets are static and dynamic software elements of which a MATE attacker might violate security requirements because they have value for the attacker or the vendor: monetary value, public image, customer satisfaction, bragging rights, etc. Examples are secret keys or confidential data embedded in applications, algorithms that constitute valuable intellectual property or trade secrets, multiplayer game logic that needs to remain intact to prevent cheating (e.g., see-through walls, use aim-bots, or show full world maps), and authentication checks that need to remain in place. These assets are the primary targets of MATE attackers. They cover a range of abstraction levels and granularities corresponding to a range of code and data elements (functions, variables, global data, constants, etc.). For example, an algorithm can be large and expressed in abstract terms, while a secret encryption key to steal is merely a string of bits. Primary assets are already present in the vanilla, unprotected software.

Phase & No.	Method Name	ESP	Phase & No.	Method Name	ESP
1 m.1	primary asset description	6.1	3 m.17	mitigation deployment	6.3.3
1 m.2	software analysis tools	6.1	3 m.18	mitigation validation	
1 m.3	secondary asset description	6.1	3 m.19	SP impact estimation	6.3
1 m.4	secondary asset identification algorithms	6.1	3 m.20	single-pass mitigation decision making	6.3.2
1 m.5	requirement description	6.1	3 m.21	iterative mitigation decision making	
1 m.6	export models of supported protections	6.1	3 m.22	asset hiding	6.3.2
2 m.7	threat analysis	6.2	3 m.23	SP selection optimization	6.3.1
2 m.8	threat impact estimation	6.2	3 m.24	SP select search space pruning	6.3.1
2 m.9	risk prioritization	6.2	3 m.25	cookbooks with SP recipes	
2 m.10	defender's analysis toolbox execution	6.2	3 m.26	driving the SP tool	6.3.3
2 m.11	incremental attack path enumeration		4 m.27	risk analysis updating	
2 m.12	incremental threat analysis		4 m.28	application exposure monitoring	
2 m.13	transparent threat analysis reporting	6.2	4 m.29	monitoring risk framing input evolution	
2 m.14	risk monetisation		4 m.30	monitoring running applications	6.4
2 m.15	OWASP risk rating methodology		4 m.31	monitoring communication of running apps	6.4
3 m.16	mitigation decision making	6.3	4 m.32	user experience evaluation	

Table 4: Methods in the proposed approach's phases, with references to the discussions of their instantiation, if any.

The c.2 secondary assets are software elements that attackers might target on their c.3 attack path (i.e., the sequence of executed c.4 attack steps) towards the primary assets. Attackers consider these elements as mileposts on their way to their primary targets. Secondary assets can be c.5 attack pivots (a.k.a. hooks) in the vanilla software, but they can also be artifacts or fingerprints of injected SPs that attackers need to overcome. An example pivot is a ciphertext buffer containing high-entropy data, which an attacker might first try to identify with statistical dynamic analysis. Once the buffers have been identified, the attacker might pivot to the program slices that produce the buffers' data, and in those slices they can obtain the secret keys. An example of an injected SP is an integrity check. A gamer that wants to alter the speed with which he can move around in the virtual game world might first have to undo or bypass the integrity check.

r.1 The distinction between primary and secondary assets should not be strict. For example, a cryptographic key that protects one movie might be a secondary asset if the attacker tries to steal one movie. A similar key that serves as a master key for all movie encryptions is clearly a primary asset. Moreover, SP vendors consider the SPs supported with their tools as primary assets that they do not want to be reverse-engineered easily. While those SPs protect the primary assets of their customers' software, they are the primary assets of the SP vendors. Should attackers learn how to attack or circumvent them automatically, their value goes down the drain.

The deployment of some SPs requires one to describe the relationship between assets and non-asset program elements. This is the case when SP transformations applied to the code of assets require other non-asset code to be transformed with it to conserve the program semantics. When deploying an SP on only the assets, this should not make those assets stand out to the attacker, e.g., because the entropy of encrypted data or obfuscated code is much higher than that of plain data or because the protection introduces recognizable fingerprints. To increase the attacker's effort needed to localize them, one can deploy the same SPs on non-asset code, as proposed by Regano et al. [91]. Furthermore, to decide which SPs can be deployed conservatively, it might be necessary to analyse the whole application and model it. In short, an M.1 application and asset model is needed to describe the wide range of software elements that form the target application, including the elements of assets and non-assets, and the relevant relations between

them. The s.2 application meta-model of Basile et al. can provide a useful starting point [19], but it definitely needs to be refined, as it currently only captures coarse-grained relations such as call graphs.

Multiple methods need to be considered for instantiating a concrete application model. Obviously, a m.1 primary asset description method is required to let a user identify and describe their primary assets, preferably at a high level of abstraction, such as with s.3 source code annotations [20]. Next, a.1 m.2 software analysis tools need to map those descriptions onto the corresponding lower-level software elements (e.g., onto corresponding assembly operations) and extract the structure and relevant properties of the software. Such tools are already used in all SP tools we know of, both commercially and in research. If the SP decision support tools cannot identify secondary assets themselves, a m.3 secondary asset description method is required to let a user describe the secondary assets and how they relate to primary assets. Alternatively, we foresee that a.2 m.4 secondary asset identification algorithms can be developed to automate their identification. Such algorithms would be executed in the later risk assessment phase, but in the framing phase, the necessary knowledge needs to be modelled in the form of M.2 secondary asset models that describe what technical attributes of software elements allow attackers to exploit them as mileposts. An example is the already mentioned buffers that contain high-entropy data. Precisely the fact that some buffer holds such data makes it a potential milepost. ?.1 The design of such secondary asset models is an open issue. Note that those models would not need to be recreated from scratch for every application. Instead, they would be reusable and grow over time as new types of secondary assets are considered.

As SP aims to delay attacks rather than prevent them, we need M.3 asset value evolution models to describe the evolution of their value over time, including the c.6 attack time frame in which assets have value as well as the impact a successful attack can have on a business model. This includes the c.7 renewability of assets, i.e., how easy it is to replace software and assets to reduce the impact of successful attacks. For modelling this evolving relationship between business value and assets, we expect that companies can use s.4 their existing asset valuation models.

f.2 To enable asset risk framing in a standardized manner, stakeholders first need to join forces to draft a taxonomy of possible assets and their features. A starting point can be s.5 Wyseur's list of assets in the form of private data, public data, unique data, global data, traceable code/data, code, and application execution [112]. Another starting point can be Ceccato et al.'s s.6 taxonomy of code and data elements that MATE attackers considered in their experiments [34].

5.1.2. Security Requirements

The c.8 primary security requirements of assets are often the c.9 non-functional requirements of confidentiality and integrity. These come in different forms, levels of abstraction, and granularity. Their scope differs from that in other domains, so their classifications can not be trivially reused. For example, MATE integrity requirements can include constraints on where or how code is executed, that at any point in time at most one copy of a program is running, and that certain program fragments are not lifted and executed ex-situ. In addition, there might be non-repudiation requirements. For example, unauthorized copies must be detected upon execution.

r.2 For different phases in the software SDLC, different requirements may hold, and different types of attack activities may need to be mitigated, such as in the c.10 attack identification phase versus the c.11 attack exploitation phase. Some requirements may be absolute, such as a master key that should never leak; others may be time-limited, such as a key to a live event that should remain secret for 5 minutes; still, others may be relative and economical, such as that running many copies in parallel undetected should cost more than licensing them.

Assessing whether non-functional requirements can be guaranteed is hard in practice because of the MATE attackers' white-box access. c.12 Secondary security requirements can help frame possible risks. These can be (i) non-functional requirements for secondary assets; (ii) c.13 functional requirements that are easier to check but of which the mere presence in itself provides few guarantees, such as the presence of a copy-protection mechanism; (iii) c.14 assurance security requirements that minimize the risk that relevant aspects are overlooked; and (iv) what we will call t.1 c.15 protection policy requirements. The latter relates to worst-case assumptions about attacker capabilities, such as assuming that the mere presence of some features suffices to enable certain attacks. Such assumptions can compensate for the lack of proper evaluation of primary requirements. For example, a lack of stealth resulting from easily identifiable invariants in injected SPs hints for potential c.16 weaknesses vis-à-vis certain attacks [115]. Protection policy requirements then require that elements with certain features are not present at all or meet certain requirements, such as statistical properties. This is similar to security policies in the domain of remote exploitation, where, e.g., code pointer integrity is a policy about handling code pointers that can ensure that indirect control flow cannot be hijacked by exploits [71].

In the risk framing phase, the task for a case at hand is to determine and describe the security requirements for all assets and *potential* weaknesses identified as relevant. A m.5 requirement description method is needed for the user to describe their primary and part of their secondary requirements, using a requirement taxonomy. One option is s.7 to annotate the source code [20, 41]. f.3 Standardizing a taxonomy requires a community effort. ?.2 How to model protection policy requirements is an open issue. It is closely related to the secondary asset model discussed in the previous section; the necessary models will hence best be co-designed.

5.1.3. Attack Models

MATE risk management needs to consider a range of potential attacks described in an M.4 attack model. This needs to cover attackers with different levels of c.17 attack resources and c.18 attack capabilities: money, expertise, available tools, etc. The latter involves a range of methods and evolves over time, so a f.4 living catalog is needed. ?.3 We currently do not know what level of detail will produce the best results, so both more generic attack methods and tool usage scenarios (e.g., disassembling code) and very concrete ones (e.g., using the IDA Pro 8.0 disassembler) need to be supported. As the goal of SP is to delay attacks, r.3 not only the feasibility of successful attacks is to be covered, but also the potential effort involved, possibly including what attackers would probabilistically waste in unsuccessful attack strategies.

While research has shown that attackers commonly waste time on unsuccessful attack steps in real attacks [34], ?.4 it is unclear whether useful attack models can build on c.19 worst-case scenario assumptions. Examples are attackers being served by an oracle always to choose the right attack path, and analysis tools producing results with ground-truth precision. For example, locating the code of interest is an important, time-consuming attack step that cannot simply be assumed to be performed effortlessly using an oracle [80]. Doing so would imply that increasing the stealth of SPs is not useful, which experts certainly reject.

For each potential attack step, the attack model needs to encode which features of software elements are c.20 attack-enabling features, c.21 attack-preventing features, and c.22 attack effort determination features, i.e., that enable or prevent an attack step, or that significantly affect the required time and effort of an attack step, as well as the c.23 attack likelihood of success features. An example is the presence of certain secondary assets. These features might include features of the software under attack, the environment in which attacks can be performed, but also knowledge obtained by the attacker. ?.5 The best abstraction levels to consider are an open question.

The same holds for the c.24 software protections and a set of (quantitative) c.25 protection strength metrics that can be used in later phases to estimate the effort/time/resources that attackers will need to invest in the attack steps in scope. Depending on the maturity of a decision support tool, that set may have to be selected manually during the risk framing. As discussed in Section 4.3, there currently is no widely accepted set of metrics. Many proposals [13, 31] have been made on features that should be measured (e.g., control flow complexity) and on concrete metrics for doing those measurements (e.g., cyclomatic complexity [81] or code comprehension [97]). s.8 Those proposals on metrics can serve as starting points, but ?.6 more empirical research is needed on top of existing work [80] to determine which metrics are valid under which circumstances and for which purposes. s.9 RevEngE by Taylor and Collberg seems to be a good approach for enabling more productive research of human attack activities [99]. In the context of the Grand Reverse Engineering challenge[8], their data collection software is not only used to analyze attacks on randomly generated programs but also on purposely designed MATE challenges, which allows studying the relations between human attack effort and metrics. s.10 For automated attack tools, such as symbolic execution or black-box deobfuscation, the framework proposed by Banescu et al. can be a starting point [14]. Because that framework relies on Machine Learning (ML), thus requiring evaluations on many samples, it is not suited for manual attack activities.

In the risk framing phase, the task for a case at hand is to determine the attack model, i.e., the combinations of the mentioned attributes that potential attackers in scope might *potentially* have. Existing models from network security risk analysis cannot be reused. r.4 MATE attack modelling needs to include manual tasks and human comprehension of code, which are not considered in network security. For example, in network security, the development of zero-day exploits (using tools also found in the MATE toolbox) is handled as an unpredictable event, which side-steps the complexity of analysing and predicting human activities. s.11 This entirely prevents the use of existing assessment models developed for the network security scenario.

Some s.12 studies document how MATE attackers operate in practice [33, 34, 80, 113]. Together with s.13 numerous blogs and case studies by reverse engineers, such as those by Rolles [93], they can help to determine an appropriate attack model. s.14 Existing MATE attack taxonomies can also be built upon to enable users to formulate attack models for their cases [11, 10, 15].

5.1.4. Software Protections

A M.5 software protection model is needed to describe in a unified manner the wide range of SPs that a user's tools might support. This model needs to include at least possible limitations on c.26 applicability and c.27 composability, be it for c.28 layered SP deployment to protect each other or to exploit c.29 synergies between multiple SPs; the security requirements that they help to enforce; (measurable) features or limitations they have that can enable, slow-down, ease, block, or otherwise impact potential attacks, on the SPs themselves but also on the assets they are supposed to protect; how big those impacts are on the potential attacks; and potential implementation weaknesses including how they can fail to meet protection policy requirements and become (easily) attackable assets themselves; etc. The link to validated (but as of yet still missing) metrics mentioned above is clear, and the impact that deployed SPs have on metrics used to asses attack effort, i.e., the c.30 potency, c.31 resilience, and when relevant the c.32 stealth of potentially deployed SPs obviously also needs to be modelled.

[8] https://grand-re-challenge.org/

The SP model needs to capture the costs of using an SP. This can include the direct monetary costs of SP tool licenses, but also indirect costs such as having to budget for more security servers or having a longer time to market, or any other cost that might follow from changes to the SDLC.

The potential overhead of all available SPs needs to be known w.r.t. run time, latency, throughput, size, ... This is critical because many applications have a little overhead budget when it comes to responsiveness, computation times, etc. In part, the performance impact depends solely on an SP itself, such as the (constant) time or memory required to initialize it. The impact can also depend on how an SP is deployed. For example, whenever an SP requires the injection of a few instructions into code fragments, the resulting overhead will depend heavily on how frequently executed those fragments are. r.5 Multiple ways for expressing the potential cost of SPs are hence needed.

In the risk framing phase, the user needs to determine which combinations of SPs can *potentially* be deployed to mitigate risks, given the available SP tools. For automating the later phase of risk mitigation, a.3 the used SP tool should be able m.6 to export a model of all discussed features of all SPs it supports, such that a decision support tool can import that model and such that the tool user does not have to provide the information manually. Therefore, the SP tool vendor is responsible for instantiating the SP model of their tool. f.5 This obviously requires tool vendors and other SP stakeholders to agree on a standardized taxonomy of SPs and their relevant features. To model the available composability, the s.15 finite state automata proposed by Heffner and Collberg to model pre/post-requirements, pre/post prohibitions, and pre/post suggestions for combinations of SPs are an interesting idea [53].

5.1.5. Software Development Life Cycle Requirements

SPs come with side-effects, such as slowing down software, making it bigger, making debugging harder, requiring changes to distribution models, requiring certain scalability on the side of secure servers, etc. Taking the time to decide on SPs, possibly iteratively with the involvement of experts and time-consuming human analysis, also affects the time to market.

Hard and soft c.33 constraints need to be collected in terms of quantifiable overheads/costs in all possible relevant forms, and with respect to compatibility with c.34 SDLC requirements. Different constraints might apply to different parts of a program. For example, in an online game or a movie player, the launching of the game or player might have a large overhead budget, while during the game or movie real-time behavior is critical.

For all available SPs, later phases of the risk analysis will need to estimate the impact on the relevant costs and SDLC. It is, therefore, necessary to obtain all relevant c.35 profile information on the software, including execution frequencies of all relevant code fragments.

An important complication occurs when the vendors of SP tools (hereafter named SP vendors) and users of such tools (hereafter called application vendors) do not trust each other. Both parties often put severe constraints on how the SP tools are deployed and on the amount of information they exchange. An SP vendor will typically not be very forthcoming about the weaknesses or internal artifacts of the supported SPs and disallow reverse engineering of them, while the application vendors do not want to share too many details or code with the SP vendor. Consequentially, only illegitimate attackers will get white-box access to the protected applications in which SPs and original assets are interwoven as discussed in Section 4.3. If the experts performing the risk management lack white-box access to all available SPs and to the protected application, this will have a tremendous impact on the methods and data that can be used during the risk assessment and risk mitigation phases that target attackers with white-box access. r.6 This

lack of white-box access by the defenders obviously needs to be documented, and the potential impact thereof needs to be assessed during the risk framing.

In addition, aspects of the SDLC relevant to the monitoring phase (that will be discussed later) need to be framed, such as c.36 connectivity and c.37 updatability. Whether an application will always be online, occasionally connected, or mostly offline impacts which online SPs and which monitoring techniques can be deployed. So does the ability to let application servers such as video streaming servers or online game servers interact with online security services such as a remote attestation server. Likewise, it is important r.7 to document whether updates can be forced upon users and to what extent the vendors can synchronize users' updates.

Finally, c.38 limitations to the environment in which software will be distributed and executed need to be documented. For example, Android supports fewer OS interfaces for debugging, and some device vendors limit what applications can do after installation, such as iOS's limitation on downloading binary code blobs post-installment. Such limitations clearly affect the types of SPs that can be deployed, so they need to be included in the risk framing.

a.4 To avoid the need for costly human expertise and manual intervention in the next process phase, as much as possible information discussed above needs to be formalized, such that tools can reason about them in the subsequent phases. As already noted at the beginning of Section 5.1, this obviously requires a standardization effort by the community to create a standard vocabulary and taxonomies that cover all constructs and models to be documented in the risk framing phase.

5.2. Risk Assessment

In the discussion of risk framing, the term "potential" occurs frequently, because in that phase all forms of knowledge are still considered in isolation, including potential SP weaknesses, application features, SP tool capabilities, and attacker capabilities. In the risk assessment phase, one assesses how they interact for the case at hand by determining which of all potential risks actually manifest themselves in the software at hand. First, a m.7 threat analysis needs to identify the c.39 actual threats starting from an analysis of the assets and their intrinsic weaknesses, as well as from attack strategies and their technical attributes that impact their feasibility. Then a qualitative, semi-qualitative, or preferably quantitative m.8 threat impact estimation needs to be performed to identify the c.40 actual risks, and a m.9 risk prioritization needs to be done.

5.2.1. Identification of the Actual Threats

This phase aims to determine a list of attacks that could succeed on one or more of the application's assets by violating their security requirements. This phase therefore consists of a detailed threat analysis that outputs a M.6 actual threats model that describes those analyzed attacks deemed feasible within the assets' relevant attack time frames, i.e., the actual c.41 attack surface and the c.42 attack vectors on it (e.g., exploited pivots and weaknesses), the c.43 attack paths of least resistance among them, the levels and amounts of expertise, effort, and resources attackers need to mount those attacks, the damage caused by exploitation, etc. For each attack path contributing to the major threats, r.8 the weaknesses and secondary assets used by attackers as pivots need to be included, as well as the used assumptions, such as worst-case-scenario considerations or parameters that are unknown in practice. Reporting this information in an actual threat report is necessary to enable confidence in the outcome of the assessment.

Critically, r.9 the enumeration and assessment of feasible attack steps must be performed on both the attack identification phase and the attack exploitation phase. The former takes place in the attacker's lab on their infrastructure, the latter more often takes place on other users' devices.

Several open issues need to be addressed to perform this task correctly. First, f.6 standardization should produce a more precise approach and methodology for defining the MATE threat model, the attack surface, and attack vectors. The latter includes the information attackers can extract from the target software. Assets can be attacked with different strategies, in which attackers rely on automated tools and analyses to collect and exploit information about the software and to represent the software in structured representations. A range of c.44 analysis tools and techniques are applicable, all with their own strengths and limitations, including static, dynamic, symbolic, and concolic analyses. Knowing the attacker's goals and tools is the starting point for identifying and enumerating the possible attack paths. This knowledge includes the kinds of analysis results that the different tools can produce, i.e., c.45 software features such as taint information, profiles, data, and control flow dependencies. It also includes the software features those analyses depend on to produce their results, their weaknesses, limitations, and precision.

In this phase m.10 the defender hence needs to deploy their own analysis toolbox to determine the features of the primary assets and related application elements that can have an impact on the feasibility of attacks because they enable, prevent, slow down, or otherwise impact attacks. This includes r.10 checking whether the protection policy requirements formulated in the risk framing phase are violated. It also r.11 needs to be done for all potential weaknesses that were identified in the framing phase, such as invariants or fingerprints in the code that might facilitate certain attack vectors. Moreover, r.12 the set of actually present secondary assets needs to be determined to identify the presence of features that make them pivots for attackers towards the primary assets. a.5 Obviously, most if not all of the analyses in the toolbox should be applied automatically.

While we are convinced that such defender toolboxes can produce most of the necessary information for enumerating feasible attacks, a number of research questions are open. For example, ?.7 how can the formal pieces of information extracted by the tools be used to precisely identify the viable attack paths? In particular, when attackers need to resort to manual efforts, that is not easy to formalize. ?.8 How do we then assess the required effort and likelihood of success? ?.9 To what extent can automated analysis with a defender toolbox suffice to avoid the need for actual penetration testing involving human experts?

It is also an open question ?.10 how fine-grained or concrete the enumeration of considered attacks paths and their attack steps needs to be and how their attributes are to be aggregated. Since the assessment must drive the mitigation, the generated information must be rich enough for the mitigation decision makers. Therefore, to some extent, the answer to the above question will depend on the goal of the assessment. This can be a semi-automated or fully automated mitigation phase. In the latter case, assessment information must be extensive and accurate, as an automated decision support system cannot rely on human intuition and experts' past experience.

The identification of attacks with an analysis toolbox requires r.13 white-box access to the application code. In case this is not possible, e.g, because of SDLC requirements discussed in Section 5.1.5, r.14 alternative sources of information about the different integrated components need to be considered, such as c.46 third-party-provided incomplete analysis reports, i.e., partial analysis reports provided by the involved parties. Alternatively, and as long as the discussed enumeration approach cannot completely replace human expertise, the inclusion of results of penetration tests performed by red teams could be considered. In short, ?.11 the threat analysis needs to be able to take into consideration a wide range of information sources and forms.

For the scalability and practical use of a software threat analysis process, another open issue is ?.12 m.11 incremental attack path enumeration, i.e., how to update and maintain the attack path enumeration without repeating a full analysis from scratch when any of the involved aspects evolve while the application is still being developed, be it the application itself, the SP tool flow,

the attackers' tool boxes, etc. Especially if the attack enumeration involves human expertise, a solution in the form of a.6 m.12 incremental analysis is critical.

The current state of the art still requires such human expert involvement. Past research aimed to s.16 automate the attack discovery with abductive logic and Prolog [17, 92]. That suffers from computational issues, since generating attack paths as sequences of attack steps causes a combinatorial explosion and requires massive pruning. With the pruning by Regano et al. [92] only high-level attack strategies can be generated, which often do not contain enough information to make fine-tuned selections among similar SPs. For example, they allow determining the need for using obfuscation but do not provide hints for selecting among different types of obfuscation.

!.2 ML might be useful to synthesize attack paths from attack steps more effectively [6]. Moreover, !.3 s.17 methods for exploit generation [24, 95] that automatically construct remote exploits for vulnerable applications could be investigated to determine MATE attack paths automatically. They will certainly need modifications, as finding exploitable vulnerabilities is rather different from finding MATE attack paths. For example, in the MATE threat analysis, for each identified attack path r.15 defenders need to estimate the likelihood of succeeding as a function of the invested effort, attacker expertise, time, money, and luck in trying the right strategy first or not, etc. All of that is absent in the mentioned automated exploit generation.

Regarding automation, we think the identification and description of primary assets cannot be automated, as those depend on the business model around the software. They can hence not be determined by only analysing the software. By contrast, a.7 the identification of secondary assets, as mentioned in Section 5.1.1, as well as the discovery of attack paths and the assessment of their likelihood, complexity, and other risk factors, should be prime targets for automation.

Even if full automation is out of reach because parts cannot be automated or do not produce satisfactory results, automating large parts of the threat identification phase will already have benefits. It will reduce human effort, thus making proper risk assessment cheaper and hence more accessible, and it can raise awareness about identified attack strategies, thus making the assessment more effective. !.4 A gradual evolution from a mostly manual process, over a semi-automated one, to potentially a fully automated one, is hence a valuable R&D goal. We stress that in order to succeed, automated tools should then not only provide the necessary inputs for later (automated) phases of the risk management, a.8 they should also enable experts to validate the produced results to grow confident in the tools, by m.13 providing a transparent report on the performed threat analysis. Section 6 will present a tool that, although rather basic, achieves just that. s.18 For presenting the threats to human experts, different formats have been proposed in the literature, including attack graphs [88] and Petri Nets [107].

5.2.2. *Evaluating and Prioritizing Risks*

The r.16 risk assessment report must indicate the consequences that exploitation of an identified actual threat may have. It must produce an easily intelligible value or score associated with all the risks to all assets. Since the objective of the report is prioritizing the risks to drive the mitigation phase, r.17 it must not only consider the direct value of the violated primary assets, but also the side effects, like impact on the business reputation or market share losses.

Furthermore, r.18 it may consider the likelihood that attackers are interested in executing the identified threats because of different expected ROIs. For example, an attack path that offers a lot of potential gains for the attacker might be less attractive when it comes with a high probability of being detected and having to face legal consequences.

When outcomes from the impact analysis are available in proper form, our feeling is that this phase has no peculiarities compared to risk analysis in other fields. Models and methods can

therefore be adopted from existing literature to build a system that allows the consistent evaluation of the impact. As a promising option, we consider !.5 s.19 m.14 risk monetisation [47], the process of estimating the economic loss related to risk and the ROI of mitigation activity. This eases reporting to higher management and is general enough to work for every asset type, including software assets. !.6 s.20 Investigating the aspects of the m.15 OWASP risk rating methodology could also yield interesting results that might work in the MATE context [109]. Automation support for the available options can then obviously also be reused, possibly after some adaptations.

5.3. Risk Mitigation

This phase comprises two parts: first m.16 mitigation decision making, and next there is m.17 implementing and m.18 validating the decisions.

5.3.1. Mitigation Decision Making

r.19 Risk mitigation requires the defenders to evaluate how the deployment of combinations and configurations of SPs will affect the high(est) risk attack paths.

Ideally, this evaluation can be done through m.19 SP impact estimation without having to actually deploy the considered SPs and having to measure their effect. This is a major difference from the risk assessment phase, which relied heavily on measurements. ?.13 How precise the estimations need to be to enable a sufficiently precise comparison of c.47 residual risks is an open question. We consider two possible approaches.

First, we consider m.20 single-pass mitigation. !.7 This builds on an assumption that estimations are accurate enough to determine the best possible combination of SPs without additional measurement. A human or tool then first determines the c.48 most protective selection, i.e., the combination and configuration of SPs that achieves the minimal residual risk while not violating hard constraints. Next, one selects c.49 alternative protection targets that trade off some of the residual risks for other aspects, such as lower performance penalty. For each alternative target, one then again selects the best target-specific SPs and estimates the delta in residual risk and in other relevant aspects over the selection that yielded the minimal residual risk. Finally, one then chooses between the most protective selection and the alternatives. This human decision will typically involve SP experts, application architects, and managers familiar with the business strategy. Given the complexity of SP as discussed before, we consider such a decision making process not automatable at this point in time, nor in the near future.

The alternative is m.21 iterative mitigation. !.8 This approach, which is familiar to practitioners in the industry, adds additional SPs iteratively in a layered fashion. The assessment and mitigation phases are not executed once, but alternated over multiple rounds. In each c.50 mitigation round, an assessment is followed by mitigation. In the first round, the risk assessment is done on the vanilla application. In later rounds, the assessment is performed on the application protected with all SPs selected in previous rounds. During such later assessments, measurements are performed on already selected and deployed SPs. This works around the lack of precise enough estimation methods as needed for the first approach. It also eases the handling of novel risks introduced by deployed SPs, such as when the location of non-stealthy SPs might leak the location of assets.

In each round, the mitigation adds an SP layer consisting of a few additional SPs to the ones already selected in previous rounds. In each round, different combinations of SPs can be proposed that offer different risk reduction and cost trade-offs. Humans will then again select one combination and continue to the next round, or stop once the whole cost budget is consumed or no more significant risk reduction is achieved. In each round, different constraints can be imposed

that limit the SPs considered in that round, and the set of SP is chosen that offers the best potential to reduce the residual risk. Estimating the reduction potential rather than the immediate reduction in each round allows for taking into account a priori knowledge about the fact that some SPs have the potential to become much stronger after additional rounds corresponding to additional layers are deployed, while other SPs cannot become stronger because of a lack of synergies.

An example of constraints evolving between rounds is that in the first rounds SPs might only be deployed on assets, while in later m.22 asset hiding rounds, non-stealthy SPs can be added for non-assets to avoid that protected assets stand out because of SP fingerprints. Our PoC contains such an asset hiding step, albeit in the same round as the asset protection step, i.e., without performing measurements in between, as will be detailed in Section 6.3.2.

The iterative approach is more realistic for several reasons. The humans making decisions in each round can make up for deficiencies in the existing tool support and formalized knowledge, and they can build more confidence in the outcomes of the mitigation process. Secondly, measurements are performed in each round, which again allows for more confidence in the outcomes.

Automation poses the most severe constraints on the mitigation task. m.23 Optimizing the selection of of SPs must comply with computational constraints. a.9 In most usage scenarios, optimization models must return results within minutes or hours. Given the large search space to explore, this requires ad hoc m.24 search space pruning methods that prune less relevant combinations efficiently. In some usage scenarios, optimization models returning far-from-optimal results quickly are acceptable, such that the time-to-market requirements of a software launch can be met while spending more time to find better SP combinations for later updates.[9]

r.20 Within one round of decision making, the optimization process should be driven by at least the potency of the selected SP combination and by estimating the protected software's performance. Ideally, resilience is also considered. Current methods for estimating the potency, resilience, and (to some extent) overheads are not usable for automatic decision support, as they require the deployment of the SPs to perform a measurement. Given the time and resources needed to apply SPs on non-toy programs to measure objective metrics and run-time overheads, an optimization process that requires measurements instead of estimations would only consider a very limited solution space, which would make the optimization process useless.

?.14 Estimating the strength (and overhead) of layered SPs is really hard, as their code is highly interwoven.Our work on s.21 estimating the potency of obfuscations [30] and on s.22 a game-theoretic approach to optimize the selection of SPs [90] will be discussed in Section 6.3.1. !.9 ML can likely help to solve this difficult problem, as already demonstrated for specific aspects of strength such as resilience against symbolic execution [14], but clearly need further research.

?.15 Another open issue is that SPs have varying effects on attack success probability, in particular when the security requirements are time-limited or relative. In some cases, the effects can be quantified in absolute terms, such as increased brute-force effort required to leak an encryption key from well-studied white-box crypto protection. In other cases, such as the delay in human comprehension of code that has undergone design obfuscations [83], the effect is harder to quantify. When software contains different assets with different forms of security requirements, the relative value of different SPs hence becomes difficult to determine, and hence the overall risk mitigation optimization becomes increasingly difficult.

[9] Anonymously, SP suppliers confirm to us that for many of their customers the norm is weak implementation at first because security/protection is not on the feature list from product management, and then complaining when things get broken, after which the supplier needs to help out. Obviously, they prohibit us to document concrete cases.

s.23 There is a limited body of existing work available on the described decision support, and it does not cover all necessary constructs, models, and methods. In industrial practice, companies provide so-called m.25 cookbooks with SP recipes. For each asset, users of their tools are advised to manually select and deploy the prescribed SPs in an iterative, layered fashion as long as the overhead budget allows for additional SPs. Automated approaches are either overly simplistic or limited to specific types of SPs, and hence only support specific security requirements. Collberg et al. [38], and Heffner and Collberg [53] studied how to decide which obfuscations to deploy in which order and on which fragments given an overhead budget. So did Liu et al. [74, 75]. They differ in their decision logic and in the metrics they use to measure SP effectiveness. Importantly, however, their used metrics are fixed and limited to specific program complexity and program obscurity metrics, without adapting them to the identified attack paths. Coppens et al. proposed an iterative software diversification approach to counter a concrete form of attack, namely diffing attacks on security patches [40]. Their work measured the performance of concrete attack tools to steer diversification and reduce residual risks. All of the mentioned works are limited to obfuscations. In all works, measurements are performed after each round of transformations, much like in the second approach we discussed above.

To improve the user-friendliness of manually deployed SP tools, Brunet et al. proposed composable compiler passes and reporting of deployed transformations [25]. Holder et al. evaluated which combinations and orderings of obfuscating transformations yield the most effective overall obfuscation [56]. However, they did not discuss the automation of the selection and ordering according to a concrete program and security requirements.

5.3.2. Actual Deployment

r.21 In each mitigation round, the chosen SP combination needs to be deployed, so SP tools need to be configured and run to inject the SPs selected so far. Ideally, this is completely automated. a.10 This requires tool interfaces that allow m.26 the decision support tools to drive the SP tools. Providing such interfaces and enabling this automation would have significant benefits. Besides saving effort on manual user interventions, it would also skip the learning curve of configuring the used tool flows properly. Moreover, having such an integrated framework could pave the road for an open standard for an API for SP.

Following the deployment, r.22 it is critical to validate that the SP tools actually delivered as expected. Were the selected SPs injected in the intended way? Do the injected SPs have weaknesses that were not expected? ?.16 How to obtain the necessary validation is an open question in some usage scenarios, in particular when the deployment of the mitigation is executed by multiple parties that do not want to share sensitive information and do not provide white-box access to their software components.

5.4. Risk Monitoring

According to the NIST [61] risk monitoring includes "assessing control effectiveness, documenting changes to the system or its environment of operation, conducting risk assessments and impact analysis, and reporting the security and privacy posture of the system." For SP, r.23 monitoring involves the continuous tasks to be performed once the software has been released to track how the risk exposition evolves over time. This consists of two related activities: m.27 updating the risk analysis, and m.28 monitoring the risk exposure of the released application.

5.4.1. *Keeping the Risk Analysis Up-to-date*

Keeping the analysis up-to-date requires m.29 monitoring how the inputs used in the three earlier risk analysis phases evolve over time and how that evolution affects the decisions made in those phases. We can abstract these into monitoring the evolution of three different pillars of information: the information related to the assessments (e.g.,, new attacks, attack techniques, tool updates), the information related to SPs (e.g.,, updates, vulnerabilities, breaches), and the information related to the protected application. Of course, monitoring can then lead to the decision that a differently-protected version of the application should be released whenever any tracked changes lead to a re-evaluation of earlier decisions.

The r.24 monitoring of information related to SPs concerns both attacks against existing SPs and newly developed SPs. For example, when a complex attack technique (e.g., generic deobfuscation [115]) is first presented in the academic literature, it might not be considered relevant during an original risk assessment because the attack is hard to replicate and its effectiveness has not been demonstrated on more complex pieces of software. However, when attackers later release a toolbox that automates the replication and publish a blog discussing how they used it to attack a complex application successfully, this should lead to a re-evaluation. Similarly, r.25 when new SPs become available with higher effectiveness against old or new attacks, or with lower overhead, this may lead to a re-evaluation.

Similarly, r.26 monitoring must consider that information related to the application can itself evolve. One example is where a company might decide that there are, in fact, additional assets in the program that need to be protected. This can happen both as a late realisation after deployment, but also in the case where the application itself evolves over time, by virtue of new versions being released with changes in functionality or structure. Another example is that the priorities in the company's value estimation can change over time. This would mean that the associated formulas for the risk analysis produce different values.

5.4.2. *Risk Monitoring of the Released Application*

Next, one needs to m.30 monitor how copies of the released software are running on their users' premises. This can be achieved by m.31 monitoring the information that the protected application communicates to the vendors. Such information may originate from a s.24 monitoring-by-design SP such as reactive remote attestation [105], but also from communication with other online components that were not originally designed for online monitoring. This is particularly the case when anomaly detection can link irregular communication patterns to unauthorized activities, such as running multiple copies in parallel or executing program fragments in a debugger in execution orders or frequencies not consistent with authorized uses. Such patterns can occur from communications present in the original applications, or from online SPs such as code s.25 renewability [7] and s.26 client-server code splitting [32]. Importantly, the use of non-monitoring communication does not require the implementation of reaction mechanisms in the protected application to be effective.

In many cases, r.27 it is advisable to analyse the data obtained with the monitoring. The insights extracted can be helpful to respond to detected anomalies, for example, by letting the application server take action in case of discovered attacks, as well as to keep the risk analysis process up-to-date, for example, to re-valuate the threats and their likelihood.

Finally, the vendor of the released application needs m.32 user experience evaluation methods to monitor whether the impacts of the deployed SPs on the user experience and cost are in line with expectations or promises by the SP vendor. For example, if users start reporting usability

issues or if online SPs lead to scalability issues, e.g., because more copies are sold than originally anticipated, those evolutions might also warrant a revision of the risk mitigation strategy.

6. Proof-of-Concept Expert System for Software Protection

Expert systems exist in cybersecurity since 1986 when Hoffman proposed one for the risk analysis of computer networks [55]. From the initial intrusion detection systems [46, 100] to modern ones using AI [87], expert systems have been used to automatically configure security controls [49], for post-incident network forensics [69, 72] and decision making.

The Expert System for Software Protection (ESP) is our PoC tool that implements a semi-automated SP risk analysis[10]. Its complete code is available[11], as well as a technical report on its inner workings [20], a user manual [41], and a demonstration video[12]. The ESP is primarily implemented in Java as a set of Eclipse plug-ins with customized UI. It protects software written in C and needs source code access. The target users are software developers or SP consultants. After the user manually annotates the assets in the source code, the ESP can generate what it considers optimally protected binaries and the corresponding security-server-side logic without human intervention. As requested by the SP experts involved in the evaluation, a step-wise execution is also available where users can check and possibly override any information generated by the tool before executing the next step.

Figure 3 depicts the high-level workflow, split into the four phases discussed in Section 5.

All the data needed for the risk analysis process, starting from the *risk framing* information and including all the data obtained during the other three phases, are modeled and stored in a Knowledge Base (KB). The used model and corresponding KB structure are designed specifically to support the reasoning methods needed for a software risk analysis [19]. For example, it allows representing context information, like the attacker model and the SPs available to mitigate the risks, and information about the application to protect, including the assets and abstract representations of the application code collected through code analyses. More details about these constructs and the model are presented in Section 6.1.

Next, the ESP performs the *risk assessment* phase, whose details are provided in Section 6.2. This phase enriches the data in the model. It infers the possible attacks against the assets and assesses the risks against each asset by estimating the complexity of executing those attacks. The risk is evaluated by considering the software's structure and the attacker model, i.e., the skills an attacker is likely to have, and asset values as defined by the user during the risk framing. The ESP's *risk mitigation* phase, detailed in Section 6.3, is also based on innovative methods. It uses ML and optimization techniques to select the best *solution*, i.e., the best sequence of SPs to be deployed and their configurations. It then automatically deploys it on the software to generate the protected application binaries. If remote SPs are included in the selected solution, the deployment phase also generates the server-side logic to be executed on a trusted remote entity.

Finally, the *risk monitoring* is performed. However, the ESP does not dynamically update the risk analysis process parameters. It only performs real-time integrity checking (as discussed in Section 5.4.2) depending on the methods implemented by the SPs used.

[10] In the ASPIRE project and in some cited papers, the ESP was called the ASPIRE Decision Support System (ADSS).
[11] https://github.com/daniele-canavese/esp/
[12] https://www.youtube.com/watch?v=pl9p5Nqsx_o

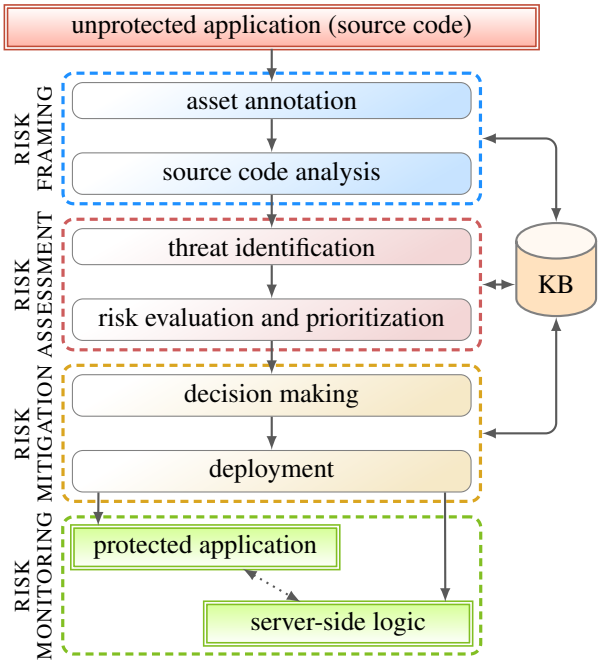


Figure 3: The ESP workflow.

The ESP can also be used in two additional modes. It can be configured to propose a set of solutions that experts can manually edit to control the SP deployment fully. Moreover, it can be used to evaluate the effectiveness of solutions manually proposed by experts.

6.1. Risk Framing in the ESP

This tasks' purpose is to initialize all the constructs and their relations as needed for risk analysis, and to store them into a model formally defined in [19] and named the KB. It covers half of the models (M.1, M.4, M.5) highlighted in Section 5. Figure 4 presents the core classes, which will be discussed in the next sections. The KB is instantiated as an OWL 2 ontology [3].

The *risk framing* starts with the preparation of the KB with *generic a-priori information*. This includes the core concepts and data not related to the specific application to be protected but relevant to framing the risk analysis process. A priori information includes the assets types (c.1, c.2); the supported security requirements (c.8, c.9, c.12, c.13, c.14); all the known attack steps and their characterization (c.4, c.17, c.18, c.20, c.21, c.22); the available SPs and their composability (c.24, c.26, c.27, c.28, c.29); and the necessary constructs to evaluate risks and mitigations (c.23, c.25, c.30, c.31, c.32, c.33) that were discussed in Section 5.1. The user can also set preferences and analysis parameters (c.38, c.44), including hard and soft constraints and SDLC requirements (c.34, c.36), as well as the SPs to consider and the kinds of attacks to counter.

The ESP then performs a *source code analysis* (m.2) that populates the KB with *a-priori analysis-specific information* using the Eclipse C Development Toolkit[13]. The analysis collects

[13] https://projects.eclipse.org/projects/tools.cdt

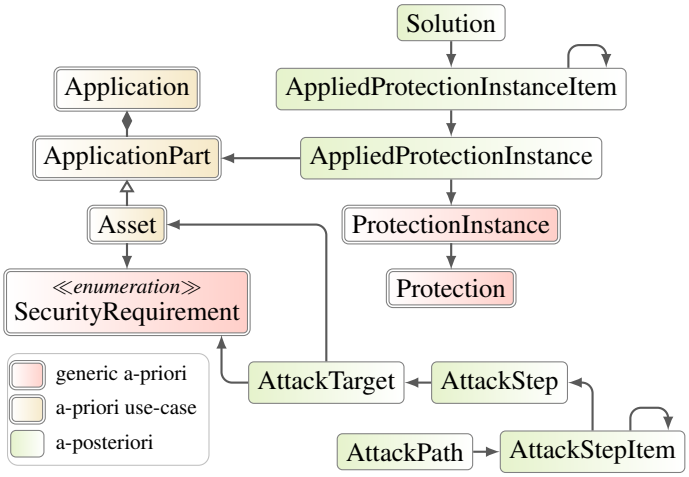


Figure 4: The top level of ESP meta-model to support modeling the relations between all relevant constructs.

all the *application parts*, i.e., the variables, functions, and code regions. It determines additional information such as variables' data types and function signatures. It produces additional representations such as the call graph, which are useful for making decisions about the SPs to apply.

The PoC of the ESP supports confidentiality and integrity requirements. The user needs to annotate the source code with custom pragma and attribute annotations [20, 41] to formally identify the code's assets and to specify their security requirements (m.1, m.5). The ESP then uses the call graph to identify potential secondary assets. These are listed in the GUI where the user can manually select which ones are to be considered assets in the later phases, and with which security requirements (m.3, m.4). Using only a call graph as a model (M.2) to find potential secondary assets that need to be manually confirmed is overly simple, and hence definitely a topic of future research.

Together with a-priori information, the KB model represents *a-posteriori information*, i.e., data inferred and stored during later workflow phases such as the inferred attacks and the solutions.

In addition, the ESP offers a GUI to edit the framing information, e.g., to mark additional assets, characterize the attacker, and choose SPs. The GUI also allows importing and exporting risk framing data as XML or OWL files (m.6). This feature was appreciated during the validation as it allows augmenting the analysis with information that may be missed by the automatic but as of yet incomplete process, like the secondary assets that might be linked into a protected program as part of certain SPs.

6.2. Risk Assessment in the ESP

The risk assessment implements several methods to estimate the actual threats and risks. In the *threat analysis*, the ESP uses backward reasoning methods (m.7) to identify the attacks (c.39) that can breach the primary assets' security requirements, and stores them in the KB [92]. This stage is roughly equivalent to the ISO27k "identify risk" step as discussed in sections 3.1 and 5.2.

The identified attacks are represented as a set (M.6) of *attack paths* (c.3). These are ordered sequences of atomic attacker tasks called *attack steps* (c.4). Attack paths are equivalent to attack

graphs [88] and can serve to simulate attacks with Petri Nets [107]. The attack steps that populate our PoC KB originate from a study and taxonomy by Ceccato *et al.* [33, 34] and from data from industrial SP experts who participated in the ASPIRE project.

The attack paths are built via backward chaining (m.7) as proposed in earlier work [17, 92] and implemented with SWI-Prolog [108]. An attack step can be executed if its premises are satisfied. It produces the results of its successful execution as conclusions. The chaining starts with steps that allow reaching an attacker's final goal (the breach of a primary requirement) and stops at steps without any premise. The search algorithm builds a proof tree with increasing depth and width, with exponential complexity. The ESP hence implements basic yet aggressive search space pruning to build an attack catalogue, e.g., by considering a maximum length for the inferred attack paths [90]. The proof tree models the actual threats (M.6). Its nodes can be seen as the exploited attack vectors (c.42), of which the leaves form the identified attack surface (c.41).

The ESP performs the *threat impact evaluation* (m.8) and *risk prioritization* (m.9) by assigning a *risk index* (c.40) to each identified attack path. Every attack step in the KB is associated with multiple attributes, including the *complexity* to mount it, the minimum *skills* required, the availability of support *tools* and their *usability*. Additional attributes can be associated with entities trivially. Each attribute assumes a numeric value in a five-valued range. For assessing the actual risks, the values of complexity metrics and software features (c.45) computed on the involved assets (m.10) with the available analysis tools (c.44) are used as modifiers on the attributes (c.22). For instance, an attack step labelled as medium complexity can be downgraded to lower complexity if the asset to compromise has a cyclomatic complexity below some threshold.

The risk index of an attack path is obtained by aggregating the modified attributes of its steps into a single value (m.8). Our PoC is rather simple. Per attack step, it first aggregates all the step's modified attributes into a single attack step risk index. The attack path risk index is then computed by multiplying its steps' indices. Other aggregation functions are supported, such as summing the steps' indices, selecting maxima, and more complex features can easily be incorporated, like making the attack path risk index depend on how many different expert tools are required.

The report (m.13) presenting the attack paths and the computed risk indices was welcomed by security experts (as will be discussed in more detail in Section 7), amongst others because they serve as a starting point for evaluating the weaknesses of an application before more manual risk mitigation. Experts were interested in refining the identified, most risky paths into more concrete sequences of attack operations, and in some cases, they would have manually updated the risk indices. In our PoC, the attack steps are coarse-grained, such as "locate the variable using dynamic analysis" and "modify the variable statically". This is an important limitation. As Section 5.2.1 discusses, understanding how much refinement is needed is an open research question.

6.3. Risk Mitigation in the ESP

Before presenting the ESP's risk mitigation process m.16, we introduce more precise constructs. In the ESP, an *SP* is a specific implementation of an SP technique by a specific SP tool. For instance, control flow flattening [106] as applied by Diablo in the ASPIRE Compiler Tool Chain (ACTC) and by Tigress are considered distinct SPs [37, 103].[14] A *protection instance* (PI) is a concrete configuration of an SP technique. The ESP can use the PI to drive the SP tool to apply an SP technique on a chosen application part. Depending on the available parameters, multiple PIs can be defined for the same SP. An *applied PI* is the association of a PI with an application

[14] https://github.com/aspire-fp7/actc and https://tigress.wtf/

part, which states that the PI has been selected to be applied to the part. A *candidate solution* is a sequence of applied PIs. It is ordered because of composability and layering requirements and benefits (c.27).

The ESP first searches for *suitable SPs*. These are SPs that impact attributes of the listed attack steps (m.19). For example, they are able to defer an attack step. Each PI is associated with a formula that alters these attributes for each attack step. After the application of an SP, the risk index of the attack steps and paths are re-assessed.

The formulas also consider complexity metrics (c.25) computed on the protected assets' code. This way, the ESP incorporates Collberg's prescription of *potency* [38] (c.30) as a measure of the additional effort that attackers have to invest on protected code. The parameters to be used in the formulas for evaluating the impact of SPs on attack steps are stored in the KB. They are based on a survey among the developers of all SPs integrated into the ASPIRE SP tool flow [20], whom we asked to score the impact of their SPs on a range of attack activities in terms of concrete impacts. These include the impact on human comprehension difficulty by increasing code complexity, the impact of moving relevant code fragments from the client-side software to a secure server not under the control of an attacker [7, 29, 105], the impact on the difficulty of tampering through anti-tampering techniques with different reaction mechanisms and monitoring capabilities [105], and the impact of preventive SPs such as anti-debugging [8, 9]. The survey results were complemented with expert feedback and validated in pen test experiments [33, 34].

Additional modifiers are activated when specific combinations of PIs are applied on the same application part. They model the impact of layered SPs (c.28) when recomputing the risk indices and synergies between SPs. The existence of synergies (c.29) was part of the mentioned survey.

Candidate solutions must also meet cost and overhead constraints (c.33). Our PoC filters candidate SPs using five overhead criteria: client and server execution time overheads, client and server memory overheads, and network traffic overhead.

Finally, the *SP index* associated with a candidate solution is computed based on the recomputed risk indices of all discovered attack paths against all assets, weighted by the importance associated with each asset. The SP index is the ESP's instantiation of residual risk (c.47).

6.3.1. Asset Protection Optimization

The ESP finds the mitigations by building an optimization model that it solves with a game-theoretic approach (m.23). The ESP tries to combine the suitable SPs to build the optimal layered solutions, i.e., the candidate solution that maximizes the SP index and satisfies the constraints.

Computing the SP index by re-computing the risk index requires knowledge of the metrics on the protected application. As applying all candidate solutions would consume an infeasible amount of resources, we have built an ML model to estimate the metrics delta after applying specific solutions without building the protected application [30]. The ESP's ML model has been demonstrated to be accurate for predicting variations of up to three PIs applied on a single application part. With more SPs, however, the accuracy starts decreasing significantly. This issue seems to be solvable with larger data sets and more advanced ML techniques.

The ESP uses the same predictors to estimate the overheads associated with candidate solutions. Per PI and kind of overhead, the KB stores a formula for estimating the overhead based on complexity metrics computed on the vanilla application. These formulas were determined by the developers of the different SPs integrated into the tool flow of the ASPIRE project.

Combinations greatly increase the solution space. To explore it efficiently and to find (close to) optimal solutions in an acceptable time, the ESP uses a game-theoretic approach, simulating a non-interactive SP game (m.24). In the game, the defender makes one first move, i.e., proposes

a candidate solution for the protection of all the assets. Each proposed solution yields a base SP index, with a positive delta over the risk index of the vanilla application. Then the attacker makes a series of moves that correspond to investments of an imaginary unit of effort in one attack path, which the attacker selects from the paths found in the attack discovery phase. Similarly to how potency-related formulas of the applied SPs yield a positive delta in the SP index, we use resilience-related formulas that estimate the extent to which invested attack efforts eat away parts of the SP potency, thus decreasing the SP index. These formulas are also based on expert feedback. We refer to Regano's thesis for more details on this game-theoretic approach that uses mini-max trees and a number of heuristics to yield acceptable outcomes in acceptable times [90].

After solving the game, the ESP shows the best SP solutions (c.48, c.49) it found, i.e., the best first moves by the defender, from which the user can choose one. The ESP then invokes the automated SP tools to apply the solution, as will be explained in Section 6.3.3.

6.3.2. Asset Hiding

As discussed in Section 5.1, SPs are not completely stealthy because they leave fingerprints. In a previous paper [91], we proposed a solution to this problem based on the refinement of existing SP solutions with additional SPs also deployed on non-asset code regions (m.22). Those lure the attacker into analyzing such regions in lieu of the assets' code, thus hiding the assets from plain sight. We have devised three asset-hiding strategies. In *fingerprint replication*, SPs already deployed on assets are also applied to other code parts to replicate the fingerprints such that attackers analyse more parts. With *fingerprint enlargement*, we enlarge the assets' code regions to which the SPs are deployed to include adjacent regions such that attackers need to process more code per region. With *fingerprint shadowing*, additional SPs are applied on assets to conceal fingerprints of the chosen SPs to prevent leaking information on the security requirements.

The PoC ESP hides the protected assets in an additional decision making step. In this step, it adds *confusion indices* to the SP indices, which are computed by an ad hoc formula built to estimate the additional time the attacker needs to find the assets in the application binary after the application of hiding strategies. The computation of the confusion indices requires estimating the code complexity metrics after the application of the SPs. To build this model, we have studied the effects of the hiding strategies for the SPs devised during the ASPIRE project. The results of this study, stored in the ESP KB, are used to compute the confusion index.

Starting from the solutions generated via the game-theoretic approach, the ESP proposes additional application parts to protect by solving a Mixed Integer-Linear Programming (MILP) problem, expressed as a heavily customized instance of the well-known 0-1 knapsack problem [67] that maximizes the confusion index and uses overhead as weight in constraints. The MILP problem is solved using an external solver; the PoC ESP supports lp_solve and IBM CPLEX Optimizer[15].

In between the asset protection optimization and the asset hiding, no measurement is done on code on which SPs are deployed. The ESP's decision making is a single-pass process (m.20).

6.3.3. Deployment

The final step in the ESP workflow deploys the solution on the target application (m.17, m.26). The solution is chosen by the user amongst the ones presented by the ESP. The result of this step (and of the whole workflow) is the protected binary plus source code for the server-side components for selected online SPs. The ESP deploys a solution by driving automatic SP tools. At

[15] See http://lpsolve.sourceforge.net/5.5/ and https://www.ibm.com/analytics/cplex-optimizer.

PROTECTION TYPE	REQUIREMENTS		TOOL	
	CONFIDENTIALITY	INTEGRITY	ACTC	TIGRESS
anti-debugging	●	●	●	○
branch functions	●	○	●	○
call stack checks	○	●	●	○
code mobility	●	●	●	○
code virtualization	●	●	○[16]	●
control flow flattening	●	○	●	●
data obfuscation	●	○	●	●
opaque predicates	●	○	●	●
remote attestation	○	●	●	○
white-box crypto	●	●	●	○

Table 5: SPs supported by the ESP, with enforced security requirements and tools used to deploy the SPs. For each tool, we only mark techniques supported on our target platforms, i.e., Android and Linux on ARMv7 processors.

the time of writing, the ESP supports Tigress, a source code obfuscator developed at the University of Arizona, and the ACTC, which automates the deployment of SP techniques developed in the ASPIRE FP-7 project [20, 41]. Table 5 summarizes the SP techniques supported by the ESP.

Finally, we point out that the ESP has been engineered to be extensible. All the modules can be replaced with alternative components. For example, the risk assessment based on backward reasoning could be replaced with a more advanced attack discovery tool, the only constraint being that it needs to produce output compliant with the SP meta-model. It is also possible to support new SPs. It is enough to add the required information into the KB, such as the evaluation of strengths and impacts on attack steps, conflicts, and synergies with other SPs plus all parameters of the discussed formula. The only demanding activities are training the ML algorithms to predict how new SPs alter the metrics, and the automation of the deployment of the SPs.

6.4. Risk Monitoring in the ESP

If the selected SPs include online SPs such as code mobility [29] and reactive remote attestation [105], the ESP generates all the server-side logic, including the backends that perform the risk monitoring of the released application. This includes the untampered execution as checked with remote attestation, but also the communication with the code mobility server (m.30, m.31).

Our PoC does not automatically include the feedback and other monitoring data such as the number and frequency of detected attacks and compromised applications, and server-side performance issues. The knowledge base needs to be manually updated using GUIs to change risk framing data related to attack exposure and SP effectiveness. Issues related to insufficient server resources also need to be addressed independently; the ESP only provides the logic, not the server configurations.

6.5. Coverage

As could already be seen in Tables 2, 3, and 4, the ESP covers many of the constructs, models, and methods we positioned in the overall risk management approach in Section 5. Albeit to some

[16]The ACTC provides limited support for code virtualization, meaning that it is not reliably applicable to all code fragments. Hence the ESP does not consider it a potential protection instance.

extent in a rudimentary form, as can be expected from a proof-of-concept tool, the ESP instantiates 36 of the 50 identified constructs, 5 of the 6 discussed models, and 21 of the 32 methods.

All of the instantiated artifacts were required to meet the objectives and requirements of the ASPIRE project. The reasons why the other 14+1+11 artifacts are not instantiated in the ESP are that ASPIRE research project plan was drafted and executed before the development of our vision on standardization, and that the project had a limited time frame and resources, and hence a limited scope and set of requirements to meet.

The non-instantiated artifacts relate to five major limitations that in our opinion do not impact the possibility to build an entire SP workflow that includes mostly automated tasks. Indeed, all the activities that we have not (yet) automated can be performed manually, so if a fully automated approach would be proven impossible at some point in the future, a semi-automated approach is certainly possible.

First, ASPIRE focused solely on the technical threats, neglecting the relationship with business risks. Constructs c.6 and c.11, model M.3, and methods m.14, m.15, and m.28 that relate to business models were hence out of scope. With the attack exploitation phase (c.11) being out of scope, the decision support tool did not have to differentiate between threats in that phase and in the attack identification phase, and hence no tool support to treat (c.10) as an explicit construct was needed. Existing methods to link technical threats and constraints to business risks are available as discussed in Section 5.2.2, if not automated then certainly relying on human judgments. Furthermore, providing support for considering multiple attack phases requires no fundamental changes to the used models and methods.

Second, another scope limitation was that ASPIRE only considered the protections of application instances in isolation, not as they evolve over time, and with the SP tool having white-box access to all relevant application code. Hence constructs c.7, c.37 and c.46, and methods m.11, m.12, m.27, and m.29 were out of scope. Experts can manually deal with those SDLC issues in case future research would fail to provide automated solutions.

Third, whenever functional requirements were stated, such as the need to deploy a copy-protection scheme, only one implementation of that functionality was developed. Hence no decisions needed to be made on how to meet those requirements, making decision support for functional requirements (c.13) irrelevant within the project. In general, decision support for functional requirements is simpler than for non-functional requirements: functional requirements are typically expressed as "some form of protection functionality X needs to be included." If anything, such requirements limit the search space that the SP optimization algorithms need to explore, rather than complicating it.

Fourth, whereas Section 4.2 argued for maximal formalization and automation to minimize the potential reduction in precision that can stem from the subjective expert judgments, within the ASPIRE project complete automation was not considered viable yet. The involvement of experts in making judgments was still accepted, so some aspects were not formalized and automated but instead left to human experts. This is the case for methods m.18 and m.132 for validating that the SP deployment is in line with made choices and requirements; for constructs c.5, c.15, and c.16 that serve to identify which application parts require protection despite not being primary assets; for identifying the path of least resistance (c.43) among the enumerated attack paths; and for iterative mitigation decision making (c.50, m.21). An expert can use the ESP manually in an iterative manner, but the ESP does not automate this. Finally, while the SP tool developed in ASPIRE considers profile information for minimizing the performance impact of injected control flow obfuscations, the ESP does not consider profile information (c.35) for selecting SPs. It is

hence left up to the human expert to manually exclude expensive SPs for assets on which they cannot be afforded.

Fifth, the one remaining unsupported artifact is that of cookbooks with SP recipes (m.25). Those cookbooks are only intended as backups for when automated SP selection is not supported. They are hence superfluous in the ESP.

7. Evaluation of the Instantiated Artifact

The ESP has been designed and implemented to answer RQ1 (whether automated decision support tools can assist experts with the deployment of SPs and the use of SP tools) within the scope and requirements of the ASPIRE project. However, this evaluation also provides evidence to answer RQ3 on which parts of a standardized risk management approach to SP can already be automated, as it provides a lower bound on the set of those parts. Moreover, since the ESP implements a NIST-based four-phase risk analysis approach, a positive evaluation of the tool provides evidence that modelling SP as a risk management task is feasible. The ESP hence partially backs up the list of constructs, models, and methods discussed in Section 5 to answer RQ2 on which artifacts a standardized risk management approach in the domain of SP needs to entail.

For this evaluation, the research question RQ1 is further split into more precise technical questions according to ISO/IEC 9126-1:2001 evaluation criteria. Table 6 lists these questions and the main results/answers. We answer RQ1.a–RQ1.c with the qualitative evaluation in Section 7.2 of which the design is first discussed in Section 7.1. RQ1.d is answered in Section 7.3 and Section 7.4.

7.1. Design of the Qualitative Evaluation

The ESP has been validated with expert SP users drawn from the ASPIRE project consortium and advisory boards. This qualitative evaluation is a snapshot of experts' opinions about the final ASPIRE PoC at the end of the project in Q4 2016, when they were last available to us.

In ASPIRE, each industrial partner provided an Android app use case: a One-Time Password generator for home banking apps, an app licensing scheme, and a video player with Digital Rights Management (DRM) for protected content. Each app included security-sensitive code and data elements in dynamically linked, native libraries written in C. Those libraries served as reference use cases for all research. They were designed and implemented to represent the industrial partners' commercial software. Being sensitive, the industrial partners only gave access rights to their use cases to academic partners, not to each other. Less sensitive information on the use cases, their assets, their security requirements, the software features obtained with the analysis tools, and their experts' assessments are available in a public report that presents a joint validation of all project results [18]. Table 7 presents the Source Lines Of Code (SLOC) metrics and the number of assets. Clearly, the ESP was not evaluated merely on toy examples.

The evaluation of the ESP in ASPIRE was planned to be done by two experts per industrial partner: one *internal expert* familiar with the project and involved in it, and one *external expert* that was not involved in the project. However, one of the three partners only made the internal expert available, who hence participated in both roles. According to FEDS, this limitation in the number of experts and the limitation in the scope (each company's experts evaluated the artifact using only their own use case) needs to be considered a significant constraint.

We then organized the qualitative evaluation of usability, correctness, comprehensibility, and acceptability in three steps, as visualised in Figure 5.

Usability	
RQ1.a.1	Is the ESP usable by software protection experts? → positive answer: experts did not have problems in using the tool and understanding the meaning of all the artifacts
RQ1.a.2	Can the ESP become part of the experts' daily workflow? → positive answer: experts reported that the ESP was suitable for their daily job but further effort is needed for a better business alignment
RQ1.a.3	Could the ESP be used also by software developers with limited or no background in software protection? → partially positive answer: proper tuning of the ESP requires a solid background in SP, protection will be less effective when using the ESP as a push-one-button tool
Correctness	
RQ1.b	Does the ESP propose appropriate combinations of protections to protect the assets and to hide them properly? → positive answer: experts analysed all artifacts produced by the ESP, compared them to data from tiger teams, and judged them as correct or appropriate
Comprehensibility and Acceptability	
RQ1.c.1	Is the ESP's output useful to comprehend and assess the tasks (automatically) performed by the ESP? → positive answer
RQ1.c.2	Is the output produced by the ESP useful to help software protection experts manually perform their job? → answer is limited: the inferred attacks has been judged too coarse-grained, this affects the precision in automatically deciding the mitigating protections
Efficiency	
RQ1.d.1	Is the ESP fast enough to produce valid solutions in a useful time? → positive answer
RQ1.d.2	Is the complexity of the algorithms it uses acceptable for the size of the tasks it has to perform? → positive answer: experts and developers considered the execution times acceptable, scalability may be an issue due to worst-case complexity, but heuristics and optimization allowed producing solution in useful time

Table 6: Refinement of RQ1 into concrete ESP research questions and this paper's main answers to them.

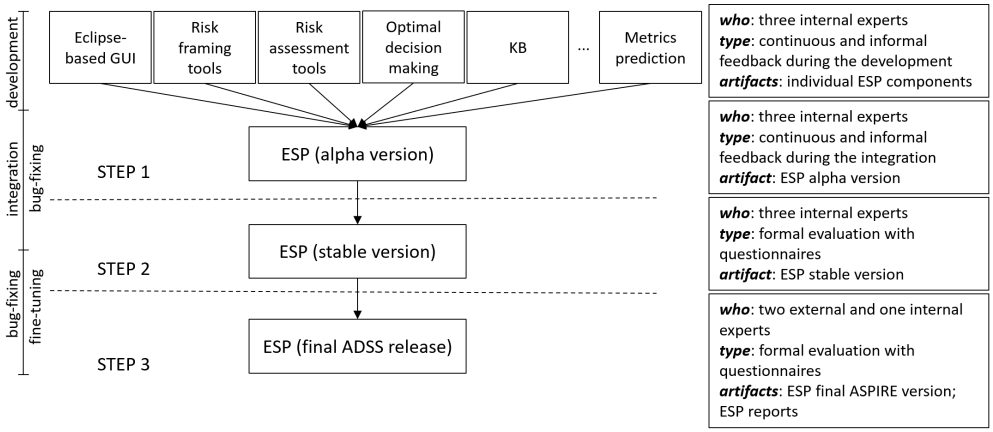


Figure 5: Graphical representation of the qualitative evaluation process.

7.1.1. Step 1: Early Internal Expert Assessment

During the ESP's development the internal experts performed a qualitative analysis of the prototype artifact to improve early versions. They followed the design and development of the ESP throughout the three-year project and continuously monitored the results of the SP on their use cases, using the ESP to provide constructive feedback. They analysed methods, models, constructs, and instantiations of the individual ESP components to check the correctness of their results. They were also involved in designing the workflows to comply with their corporate needs. The result was an alpha version of the tool components integrated into the workflow.

This alpha version was then evaluated as a whole by the internal experts. Each internal expert was asked to protect their software only using the artifact with the support of the ESP developers. They manually annotated the assets in the source code using the ESP GUI and then used the ESP to identify the best SPs and their configuration parameters. They then discussed and analysed the identified threats and the selected SPs, as well as the entire decision making process, on which they then commented in detail. We collected their inputs through interviews during face-to-face meetings, ad-hoc calls, and emails. Since these inputs were collected during the normal project development, the collection was managed informally. The experts also gave a qualitative assessment of the correctness of the artifact's used models.

Moreover, the effectiveness of the ESP's selection of SPs was tested against the judgment of other experts. Indeed, the use cases' developers and security architects proposed the best combination of SPs for each of the assets, according to their expertise and experience. Each of the three use cases protected with the best combination were then pen tested by two external pen testers per use case for several weeks to establish the attack paths. The pen testers reported on the attacks that were prevented entirely for the DemoPlayer use case within the pen test time frame, and that the attacks were delayed effectively for the other two use cases[17].

[17]This material is available in Section 5 of the public ASPIRE Validation Report [18] and in sections 8–11 of the public ASPIRE Security Evaluation Methodology Report [31]

7.1.2. Step 2: Final Internal Expert Assessment

Towards the end of the ASPIRE project, a first stable version of the whole ESP was available. On the basis of this version, experts were asked to assess the ESP by answering a set of open-ended questions, which are provided in Appendix A. The experts' answers were used to develop the final release of the tool during the ASPIRE project. Their answers are not reported in this paper as they were considered confidential material. Several calls took place with those experts to clarify questions and to ensure that we interpreted their answers correctly.

This assessment of the design and implementation of the ESP constitutes a qualitative evaluation in a naturalistic scenario, which, using FEDS terminology, means that a real system (artifacts) is used by real users to solve real problems [89]. In our case, the artifact consists of the first complete version of the ESP; the real users are the industrial experts; and the real problem is the selection of protections to mitigate real MATE attacks (as evaluated in pen tests by pen testing experts [34]) on applications developed by the industrial project partners to be equivalent to their commercial applications, i.e., feature the same types of assets, similar functionality, and similar complexity.

7.1.3. Step 3: Assessment with External Experts

Finally, external experts from the industrial partners, which had no prior insights or bias regarding the ESP, were involved in a qualitative evaluation of the final ESP version in the ASPIRE project. We used the same questionnaire for this evaluation.

We prepared a virtual machine (VM) with a running copy of the ESP pre-configured with all the manual operations already performed by their colleagues. The assets were already annotated, and the other ESP running parameters were set to default values, which they were allowed to modify. However, they experienced configuration issues when integrating the ESP VM with the SP tools. To make good use of their extremely limited time, we therefore also executed the tool with the pre-configured information and all automation enabled and provided them with the output generated by the tool in the form of a report[18]. The expert was then asked to assess the identified threats, the selected SPs, and the selected properties of the evaluation.

7.2. Qualitative Evaluation Results and Discussion

Overall, based on the analysis of their questionnaires, the experts have judged the ESP as promising and potentially effective because of the high level of automation and configurability (including the possibility to override default configurations) and the detailed output. Nonetheless, they were skeptical about extending the tool's use to software developers with a more limited background in SP as this background is needed for understanding the artifacts, making decisions, and evaluating results. This is not preventing software developers from using the ESP as a push-of-the-button tool and having their applications protected. However, they feared that in the push-of-the-button mode, the applications risked being less protected than under the supervision of experts. Furthermore, the acceptability showed limitations at the level of integration with their daily work and tool chains, which means that further effort is needed to ensure so-called alignment with business [89]. The usability was hence assessed positively for experts in SP (RQ1.a.1),

[18]The reports are available at https://github.com/daniele-canavese/esp/tree/master/reports. The ESP user manual [41] describes how to interpret the different parts of those reports. In two of the three reports, we renamed identifiers of code and data elements (such as function names) in consideration of the two companies' confidentiality requirements. Apart from that, the linked reports are identical to the ones assessed by the experts. A summary of the most significant data, their interpretation, and the major findings is presented on the aforementioned GitHub site.

positively with limitations for the integration into the experts' tool flow (RQ1.a.2), and only partial for developers (RQ1.a.3).

Among the data extracted by the tool, experts highlighted the importance of making decisions by considering the application structure and metrics because results are to be tailored to the target application. They also appreciated that all the data extracted and represented in the KB are structured using a formal meta-model, as this reassured them of the correctness of the inferences.

Experts analysed the attack paths inferred by the tool as well as the SPs solutions that were proposed by the optimization process to mitigate the inferred attacks. The experts compared those solutions to the ones they (or their colleagues) had assembled manually earlier on during the project as part of the requirements formulation. Solutions have been validated in terms of achieved security for the assets, preservation of the application business logic, and containment of the inevitable slow-down of the protected application w.r.t. the original one. Furthermore, the attack paths have been compared with the real attacks discovered by the professional pen testers previously involved, as discussed in Section 7.1.1. In particular, the inferred solutions have been judged as appropriate to protect the use case code and effective in blocking the inferred attack paths and the real attacks reported by the pen testers. In addition, the protected binaries were evaluated as semantically unaltered and usable: they still delivered the original observable IO-relation without excessive overhead introduced by the SPs. It is worth mentioning that, as is the case for all the risk analysis processes, there is not a correct set of answers forming a ground truth. The experts hence provided their qualitative estimations of solution effectiveness.

The main flaw of ESP reported by the experts is that inferred attack steps were too coarse-grained because of too generic attack rules. This limitation has a technical impact on the possibility of making fine-grained decisions on the SPs to use. For instance, consider the listed attack step `staticallyLocate('ProvisioningManager_LaunchProcess.r16'(attacker))`. This denotes an attacker disassembling the binaries with static tools such as Radare2 or IDA Pro to locate basic block `r16` in function `LaunchProcess`. From this, it is possible to infer that obfuscation is needed. However, on the basis of this information alone, it is impossible to determine which specific obfuscation technique should be preferred without looking at the actual code. So expert judgment and interaction to refine the SP selection by the ESP is currently still required. It is definitely possible to populate the KB with more fine-grained attacks steps, but as already mentioned in Section 5.2.1, further research is needed to determine the best level of granularity to model and enumerate attacks steps.

From all of the above, we conclude a positive evaluation of the correctness of the artifact (RQ1.b) and the comprehensibility of the artifact-generated data (RQ1.c.1). The usability of specific artifacts has limitations (RQ1.c.2).

The efficiency has been measured with a quantitative assessment, as will be discussed in Section 7.3. In addition, no experts reported issues with the performance of the artifact.

Overall, the evaluation result is hence positive. Quoting from the related project deliverable, *"after the analysis of the validation data, the experts concluded that the tool has a very high potential"* to be used in their everyday tasks and to enter their current workflow in the near future, even if some had doubts on the maturity of the tool and its readiness to be used to protect commercial applications with all of their SDLC intricacies and complexity. For an artifact developed as a research proof-of-concept, this should of course not come as a surprise.

We conclude that a large part of the proposed risk management approach can indeed be automated through decision support tools, as identified by many of the checkmarks in Table 4 that provide an answer to RQ3. While not yet capable of completely replacing human experts, those proof-of-concept automated tools have shown promise to aid users of SP tools.

APPLICATION	C		JAVA	C++	ASSETS
	SOURCES	HEADERS			
DemoPlayer	2,595	644	1,859	1,389	25
LicenseManager	53,065	6,748	819	0	43
OTP	284,319	44,152	7,892	2,694	25

Table 7: Lines of source code counts of the ASPIRE validation use cases.

APPLICATION	TOTAL	FRAMING	ASSESSMENT	MITIGATION
DemoPlayer	145.6	0.1	76.3	69.1
LicenseManager	296.1	0.3	187.6	108.1
OTP	170.0	0.9	69.2	99.9

Table 8: ESP times in seconds.

7.3. Technical Assessment

Our evaluation also includes a purely technical analysis of the performance of the algorithms and techniques used in the ESP on both the reference use cases and artificial applications.

First, we have measured the execution time of the final version of the ESP on the three use cases, with the assets annotated by the experts, as reported in Table 7. In all three cases, 17 PIs have been considered using the SPs listed in Table 5. Opaque predicates, branch functions, and control flow flattening were applied at three configuration levels (low, medium, and high frequencies with corresponding overhead levels). Data obfuscation included three techniques: XOR-masking, residue number encoding, and data-to-procedural conversions [38].

Table 8 shows the ESP computation times. The framing phase is almost instantaneous and driven by the lines of annotated code. Regarding the assessment and the mitigation phases, these measurements do not provide sufficient data for a full assessment of the scalability and complexity of the used algorithms, as the three applications do not provide enough data points to identify correlations between the computation times and the number of assets/PIs.

We hence complemented the measurements with a formal evaluation of the algorithms' complexity and a performance measurement on artificial scenarios (in FEDS terminology). The formal validation investigated the most influential factors for the attack discovery tool and the game-theoretic optimization of the mitigation phase. The complexity of the attack discovery algorithms is exponential in the number of attack steps in the KB, hence the need to consider pruning strategies. The complexity of the algorithms in the mitigation phase is linear in the number of assets. It exponentially depends on the number of PIs and the number of attacks discovered in the assessment phase. In this case, having a limited number of PIs and pruning the sequences of SPs helped with reasonable performance.

To assess scalability, we evaluated the performance of the ESP on three synthetic standalone Linux applications with an increasing number of assets. Table 9 summarizes their metrics. These artificial applications have been randomly generated with a process that selects a call graph (from a set of call graphs extracted from real applications), and then generates randomized function bodies to meet specific code metrics. Then it randomly selects fragments in the generated code as data or code assets. In this experiment, we used all the previously listed PIs except white-box crypto (which was a proprietary algorithm of one industrial ASPIRE partner). We also added four instances of obfuscation using Tigress, i.e., the ones marked in Table 5.

APPLICATION	SLOC	FUNCTIONS	ASSETS		
			CODE	DATA	TOTAL
demo-s	443	18	2	2	4
demo-m	1029	47	12	3	15
demo-l	3749	178	26	13	39

Table 9: Statistics of ESP experimental assessment applications.

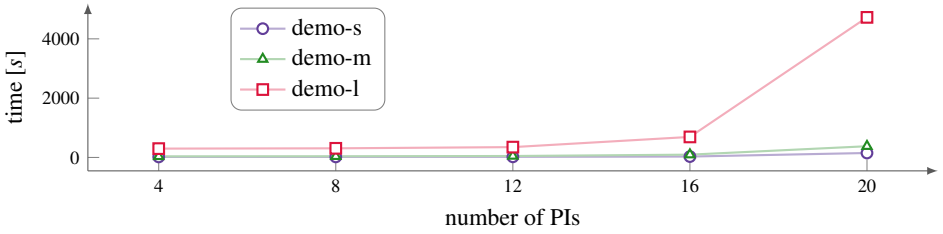


Figure 6: ESP execution times on applications reported in Table 9.

On the three artificial applications, we deployed the asset protection optimization approach described in Section 6.3 multiple times for different configurations that feature varying numbers of available PIs. This deployment was done on an Intel i7-8750H workstation with 32 GB RAM, using Java 1.8.0_212 under GNU/Linux Debian 4.18.0. Figure 6 depicts the measured total ESP computation time, along with the time needed for the risk assessment, asset protection, and asset hiding phases. The time needed to complete the workflow increases with the number of PIs considered during the mitigation; such an increase strongly depends on the application code complexity, and in particular on its SLOC and number of assets and functions.

The time needed to analyze the applications' source code and to generate the application meta-model instance was negligible at less than 1s. The time required to deploy the solution is irrelevant for assessing the ESP' computational feasibility, as it only measures the negligible time needed to execute the external SP tools for the single selected solution.

As expected, the time needed to execute the risk assessment phase does not depend on the number of PIs available to protect the application, as attacks are determined on the vanilla application. Nonetheless, we report that it has limited impact because of the aggressive pruning we have implemented that avoids exponential growth. The asset protection phase is by far the most computationally intensive, especially when the number of available PIs increases. Since the mitigation considers sequences of SPs, the execution time is exponential as it depends on the combination of PIs. The same holds for the asset hiding phase, although less time is needed to execute the latter compared to the asset protection phase.

These experiments allowed the positive evaluation of RQ1.d.1, as the computation times were considered acceptable by both the tool developers and the involved experts. They also enable the positive evaluation of RQ1.d.2, as the heuristics implemented in the game-theoretic approach scaled sufficiently well in our experiments to allow producing solutions in useful time, even though its theoretic worst-case behavior might be intractable.

7.4. Framing Effort in the ESP

As mentioned in Section 7.1.3, all framing tasks, including the annotation of the assets in the source code, were performed before the external industrial experts got involved in the assessment of the ESP. This enabled them to focus on the qualitative assessment without wasting their precious time on the more mechanistic framing tasks. This section analyzes the effort that is needed to perform those framing tasks with the ESP.

First, the user needs to annotate the assets in the source code. The ESP supports two options: manually annotating the source code or manually tagging code and data elements in the ESP. Our evaluation with experts only used source code annotations, which consist of (mostly single-line) pragmas and attributes [20, 41] that identify the assets and specify security requirements. For users proficient with their syntax, typing out the annotations requires at most tens of seconds per asset.

More time is required to determine precisely which elements in the source code need to be annotated because they correspond to the application assets. Security architects and software designers describe assets abstractly. When those are known upfront, developers can annotate their code as they write it, thus only requiring the aforementioned tens of seconds per asset. For developers that are familiar with the code base but need to annotate the code afterwards, we estimate that locating the assets in the code takes less than a minute for assets with high locality (e.g., single variables or single functions) to potentially tens of minutes for assets that are spread out more throughout the code base (e.g., the invocations of a specific reaction mechanism spread throughout the code for remote attestation).

The annotations were added to the ASPIRE use cases by their original developers after the development was finished. It was the first time they were adding our style of annotations. They hence faced a learning curve. Moreover, while adding the annotations, they had to validate the syntax and expressiveness of the annotation language on the fly. Had they already been proficient with the annotations beforehand and had they just needed to inject them without having to validate their design, we estimate that the time needed to annotate their use cases would have been less than one hour for the use cases with 25 assets, and less than two hours for the one with 43 assets.

In any case, locating the assets in the code base given abstract descriptions is something that any user of any non-trivial SP tool needs to do, both to configure the tool to protect the relevant code and to validate that that code has actually been protected by the tool. So compared to other SP approaches, the mentioned times are not considered overhead required to use the ESP. The same holds for selecting the attacks the user wants to mitigate and for determining which of the available SPs to consider. In the ESP, selecting attacks and SPs from the ones modelled in the KB happens with a click-of-a-button GUI interface. The time required for clicking is negligible compared to the time for deciding which ones to include or exclude. That decision making needs to happen with any decision support tool, so the ESP is not less efficient in this regard than any other decision making process. This discussion completes the answer to RQ1.d.1.

7.5. Threats to Validity

We have checked the procedure we used for evaluating the ESP against a checklist of the possible threats to validity: construct, internal, conclusion, and external validity threats [110], as well as instantiation validity threats [76].

Threats to *construct validity* concern the metrics defined for the evaluation. We have used a set of standard metrics from the ISO. Nonetheless, the risk remains that the selected metrics are not the best ones for our assessment. The evaluation scores were positive, negative, and partial,

which appeared expressive enough for our purposes. However, we could not objectively assess these criteria' satisfaction as the questionnaires included open answers. To limit subjectivity, we have evaluated the answers within the ASPIRE project. Moreover, the ASPIRE project reviewers hired by the European Commission did not contradict our conclusions.

Threats to *internal validity* concern the inferences between independent variables and evaluation outcomes. One possible noise factor that may confound the inference relates to the task comprehension. We assess this factor as negligible in this case. The task, resembling their day-to-day job and their typical applications, was described by the experts as clear; the use of the tool was documented; moreover, the experts were assisted in case of doubts (by their colleagues or by us). Another potential noise factor is the experts' commitment to perform their tasks diligently before answering the questionnaire. We gave the experts the tool and the reports to be assessed offline. We hence cannot establish the effort they invested in using and reading them. We checked that all relevant artifacts were analysed in their comments (main attack paths and all the combinations of protections); however, we cannot assess their commitment accurately. Moreover, another confounding factor is the actual objective evaluation. The experts were selected from the industrial partners of the project. Even if they were asked to evaluate the artifact objectively, their judgment may have been biased by the will not to hinder the project and the positive evaluation by the European funding agency.

Threats to *external validity* affect the generalisation of the research results to the real world, i.e., experts who want to protect real applications using an automated decision support system. In our case, the subjects of the evaluation are real experts that are protecting apps. The evaluation could hence be generalized to experts with a similar background protecting programs analogous to the ones presented in the evaluation and not too dissimilar from the ones they protect during normal job tasks in the same companies. However, it does not necessarily extend to other experts protecting different applications in other companies. Significant effort was invested in the use case applications to ensure that they are representative (in terms of size and complexity) of the code bases such experts have to protect in their daily jobs. However, since every expert only evaluated the ESP on one application, we cannot be sure that applications with different structures or from a different domain would yield similar results.

Another potential threat to external validity concerns the possibility to generalize the evaluation made on a specific tool, the ESP, to general SP tasks, which may affect answers to RQ3. In other words, this threat is not limited to the ESP, but extends to the whole approach we presented in Section 5. In this case, we assessed this threat as negligible. Having automated a specific SP task, we have proved that automation is feasible, even if the same task could be done in different, possibly better ways. Moreover, the external experts performing the evaluation did not assess the ESP within the ASPIRE scope and requirements, as they did not know about the ASPIRE-defined scope. Instead, they evaluated it vis-à-vis their day-to-day job requirements. The limitations identified in Section 6.5 on the ESP, which does not automate all tasks of the full risk management approach we presented, do not apply, as RQ3 is related to individual parts of a risk management approach for SP.

Threats to *conclusion validity* affect the validity of the methods to draw reasonable conclusions from the assessment. In the evaluation of the ESP, we have asked experts to answer open questions from a standard questionnaire structured according to the main protection workflow. Given the experts' limited availability, a state-of-the-art controlled experiment was impossible. The number of experts involved was limited as well, which does not allow us to use statistical methods that are standard in quantitative evaluations. However, through interactions with the experts, and by splitting the questions and formulating them clearly, we have been able to interpret their qualitative

inputs reliably, thus minimizing the noise and errors that might otherwise have obscured the data from which we have drawn conclusions.

Finally, threats to *instantiation validity* affect the possibility of considering the artifact we have implemented, i.e., the ESP, an instance of the theoretical object we had in mind, i.e., a semi-automated decision support system for SP [77]. The instantiation space is very large, as one can imagine many ways to implement all the tasks the ESP performs. In this space, we opted for a NIST-based four-phases approach. We have not considered alternative approaches, first and foremost because we were convinced upfront that the standard approach that works for other fields is also valid for SP. Secondly, we have implemented the ESP in the ASPIRE project, where resources for PoC and completion times were constrained. This relates to another threat, the artifact cost, that prevented the implementation of more alternatives.

Two more threats to instantiation validity apply to the ESP: auxiliary features and emergent properties, which stress the complexity of IT tools and oblige considering additional aspects that are not the main focus of the instantiation. Indeed, the integration of the components and the definition of the workflow has revealed several auxiliary features related to the UI comprehension, the user experience, and the effectiveness of the designed workflow to cope with daily SP experts' tasks. Furthermore, several emergent properties appeared related to the complexity of the data, their relationships, and the correct data presentation. We tried to mitigate these threats by continuously interacting with the internal experts during the design, development, integration, and validation of the ESP and the data artifact it produces. Moreover, every time we received suggestions in the answers to the questionnaires, we have incorporated them before the next evaluation phase. Nonetheless, we cannot exclude that these threats to the instantiation validity may have an impact.

8. Conclusion and future work

8.1. Conclusions

We discussed the necessity and potential benefits of a standardized, formalized, and automated approach for risk management in the context of software protection against man-at-the-end attacks. To that end, we discussed just such a risk management approach for software protections, which we based on the NIST SP800-39 standard for risk management for information security.

To provide an answer to RQ1 on the feasibility of automated decision support tools to improve the useability of SP tools, we developed and presented the ESP design and an evaluation of its PoC implementation. We found that many human expert judgment tasks can already benefit from automated tools and the data they produce, which experts found sufficiently usable, acceptable, and efficient; and of which they assessed the results as sufficiently correct and comprehensible.

As an answer to RQ2 on how standardized risk management approaches can be adopted for SP, we discussed in detail how the different aspects of software protection deployment decision making could and should be mapped onto risk framing, risk assessment, risk mitigation, and risk monitoring phases. For all phases combined, we identified 50 required constructs, 6 models, and 32 methods that the adopted approach should entail.

We answered RQ3 on the feasibility of formalizing and automate parts of the adopted approach by providing a mapping of the abstract construct, model, and method artifacts identified for the adopted approach onto the concrete instantiation artifacts that make up the ESP.

With these answers, we have provided convincing evidence that the proposed approach is feasible and can be automated to a large degree, and deserves the launch of a structured community effort that leads to future standardization and automation.

No.	Open Issue / Research Question Subjects
?.1	secondary asset (a.k.a. pivots, hooks, and mileposts) model design
?.2	selection and models of protection policy requirements
?.3	level of detail needed in models of attacker capabilities
?.4	to what extent worst-case assumptions are useful
?.5	best abstractions to model software features that impact the execution of attack steps
?.6	empirical validation of such models and metrics, in particular for manual human activities
?.7	identification of viable attack paths on the basis of software analysis results
?.8	estimation of attack step's required effort and likelihood of success
?.9	extent to which automated techniques can replace human pen testing
?.10	required granularity of attack steps forming attack paths
?.11	incorporation of informal information obtained from experts (e.g., pen testers) in automated threat analysis
?.12	incremental attack path enumeration
?.13	required precision of pre-deployment SP impact estimation
?.14	pre-deployment potency, resilience, and stealth estimation for layered SPs
?.15	pre-deployment estimation of SP impact on attack success probability
?.16	validation of deployed SP against assumptions made pre-deployment

Table 10: Open issues and topics of open research questions identified in the paper

No.	Research Direction
!.1	the concept and use of protection policy requirements
!.2	machine learning techniques to identify and quantify feasible attack paths
!.3	adoption of exploit generation techniques to identify feasible attack paths
!.4	gradual path from a mostly manual process to automated feasible attack path identification
!.5	adoption of risk monetisation to evaluate and prioritize actual risks
!.6	adoption of the OWASP risk rating methodology to evaluate and prioritize actual risks
!.7	single-pass selection of layered SPs with accurate assessment of impact on threats and risks
!.8	multi-pass selection of layered SPs with accurate assessment of impact on threats and risks
!.9	machine learning techniques to select the most effective layered combinations of SPs

Table 11: Potentially interesting research directions identified in the paper

8.2. Future Work

It is clear that quite some future work is needed, however, for the standardization itself, as well as for improving, refining, extending, replacing, and complementing the rather embryonic instantiations of the necessary constructs, models and methods currently available in the presented ESP in support of the automation of tasks in the approach. The artifacts in Tables 2, 3, and 4 which have no counterpart in the ESP yet are clear examples of where more research is needed.

Section 5 also highlighted a number of topics for future work in the form of open issues and open research questions, research directions that we consider interesting, and development steps for which a community effort is needed. Tables 10, 11, and 12 provide an overview of that future work.

9. Funding

This research was partly funded by the Cybersecurity Initiative Flanders (CIF) from the Flemish Government and by the Fund for Scientific Research - Flanders (FWO) [Project No. 3G0E2318]. Part of the presented results were obtained in the context of the ASPIRE FP7 research project. This project ran until October 2016 and has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 609734.

No.	Required community efforts
f.1	provisioning a complete vocabulary and methodology to describe the risk frame
f.2	provisioning a standard taxonomy of assets and their relevant features
f.3	provisioning a standard taxonomy of SP security requirements
f.4	provisioning and maintaining a living catalog of potential attack steps and their relevant features
f.5	provisioning a standard taxonomy of SPs and their relevant features in support of decision support
f.6	standardizing methodology for defining the actual threat model, attack surface and attack vectors

Table 12: Topics requiring a collaborative development effort by various stakeholders in the SP community

References

[1] CWE-656: Reliance on security through obscurity. https://cwe.mitre.org/data/definitions/656.html

[2] Gartner Magic Quadrant for Network Firewalls. https://www.gartner.com/en/documents/4007809

[3] Owl 2 web ontology language new features and rationale (second edition). W3c recommendation, World Wide Web Consortium (W3C), Cambridge, MA, US (2012). URL https://www.w3.org/TR/owl2-new-features/

[4] Global software licensing and monetization market, forecast to 2021. Tech. Rep. 3715874, Frost & Sullivan (2016)

[5] ENISA good practices for security of Smart Cars. Tech. rep., European Union Agency for Cyber Security (2019)

[6] van der Aalst, W.: Process mining: Overview and opportunities. ACM Trans. Manage. Inf. Syst. **3**(2) (2012). DOI 10.1145/2229156.2229157

[7] Abrath, B., Coppens, B., Van den Broeck, J., Wyseur, B., Cabutto, A., Falcarin, P., De Sutter, B.: Code renewability for native software protection. ACM Transactions on Privacy and Security **23**(4) (2020)

[8] Abrath, B., Coppens, B., Nevolin, I., De Sutter, B.: Resilient self-debugging software protection. In: 2020 IEEE European Symposium on Security and Privacy Workshops, pp. 606–615. IEEE Computer Society (2020)

[9] Abrath, B., Coppens, B., Volckaert, S., Wijnant, J., De Sutter, B.: Tightly-coupled self-debugging software protection. In: Proc. of the 6th Workshop on Software Security, Protection, and Reverse Engineering (SSPREW), pp. 7:1–7:10. ACM (2016)

[10] Ahmadvand, M., Pretschner, A., Kelbert, F.: A taxonomy of software integrity protection techniques. Advances in Computers (2019). DOI https://doi.org/10.1016/bs.adcom.2017.12.007

[11] Akhunzada, A., Sookhak, M., Anuar, N.B., Gani, A., Ahmed, E., Shiraz, M., Furnell, S., Hayat, A., Khurram Khan, M.: Man-at-the-end attacks: Analysis, taxonomy, human aspects, motivation and future directions. Journal of Network and Computer Applications **48**, 44–57 (2015). DOI https://doi.org/10.1016/j.jnca.2014.10.009

[12] Anckaert, B., Jakubowski, M., Venkatesan, R.: Proteus: Virtualization for diversified tamper-resistance. In: Proceedings of the ACM Workshop on Digital Rights Management, p. 47–58. ACM (2006)

[13] Anckaert, B., Madou, M., De Sutter, B., De Bus, B., De Bosschere, K., Preneel, B.: Program obfuscation: a quantitative approach. In: Proc. 2007 ACM workshop on Quality of Protection, pp. 15–20 (2007)

[14] Banescu, S., Collberg, C., Pretschner, A.: Predicting the resilience of obfuscated code against symbolic execution attacks via machine learning. In: Proc. 26th USENIX Security Symposium, pp. 661–678 (2017)

[15] Banescu, S., Pretschner, A.: A tutorial on software obfuscation. In: Advances in Computers, vol. 108, pp. 283–353. Elsevier (2017). DOI 10.1016/bs.adcom.2017.09.004

[16] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: J. Kilian (ed.) Advances in Cryptology, pp. 1–18. Springer Berlin Heidelberg (2001)

[17] Basile, C., Canavese, D., d'Annoville, J., De Sutter, B., Valenza, F.: Automatic discovery of software attacks via backward reasoning. In: Proc. 1st Int'l Workshop on Software Protection, SPRO '15, pp. 52–58. IEEE Press (2015)

[18] Basile, C., Canavese, D., Regano, L.: ASPIRE Validation. In: ASPIRE Project Deliverable 1.06 (2016). URL https://aspire-fp7.eu/sites/default/files/D1.06-ASPIRE-Validation-v1.01.pdf

[19] Basile, C., Canavese, D., Regano, L., Falcarin, P., De Sutter, B.: A meta-model for software protections and reverse engineering attacks. Journal of Systems and Software **150**, 3–21 (2019)

[20] Basile, C., et al.: ASPIRE Framework Report. Deliverable D5.11, ASPIRE EU FP7 Project (2016). URL https://aspire-fp7.eu/sites/default/files/D5.11-ASPIRE-Framework-Report.pdf

[21] Berlato, S., Ceccato, M.: A large-scale study on the adoption of anti-debugging and anti-tampering protections in android apps. Journal of Information Security and Applications **52**, 102,463 (2020)

[22] Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006/468 (2006)

[23] Van den Broeck, J., Coppens, B., De Sutter, B.: Obfuscated integration of software protections. Int'l Journal of Information Security **20**, 73–101 (2021)

[24] Brumley, D., Poosankam, P., Song, D., Zheng, J.: Automatic patch-based exploit generation is possible: Techniques and implications. In: IEEE Symposium on Security and Privacy, pp. 143 – 157. IEEE Computer Society (2008)
[25] Brunet, P., Creusillet, B., Guinet, A., Martinez, J.M.: Epona and the obfuscation paradox: Transparent for users and developers, a pain for reversers. In: Proceedings of the 3rd ACM Workshop on Software Protection, pp. 41–52. Association for Computing Machinery (2019)
[26] BSA | The Software Alliance: BSA Global Software Piracy Survey. Online at https://gss.bsa.org/ (2018)
[27] Building Security in Maturity Model. URL https://www.bsimm.com/
[28] Burkacky, O., Deichmann, J., Klein, B., Pototzky, K., Scherf, G.: Cybersecurity in automotive. Tech. rep., McKinsey & Company (2020)
[29] Cabutto, A., Falcarin, P., Abrath, B., Coppens, B., De Sutter, B.: Software protection with code mobility. In: Proc. of the 2nd ACM Workshop on Moving Target Defense, MTD '15, pp. 95–103. ACM (2015)
[30] Canavese, D., Regano, L., Basile, C., Viticchié, A.: Estimating software obfuscation potency with artificial neural networks. In: Security and Trust Management, pp. 193–202. Springer International Publishing (2017)
[31] Ceccato, M.: ASPIRE Security Evaluation Methodology. Deliverable D4.06, ASPIRE EU FP7 Project (2016)
[32] Ceccato, M., Dalla Preda, M., Nagra, J., Collberg, C., Tonella, P.: Barrier slicing for remote software trusting. In: 7th IEEE Int'l Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 27–36. IEEE Computer Society (2007)
[33] Ceccato, M., Tonella, P., Basile, C., Coppens, B., De Sutter, B., Falcarin, P., Torchiano, M.: How professional hackers understand protected code while performing attack tasks. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), pp. 154–164. IEEE Computer Society (2017)
[34] Ceccato, M., Tonella, P., Basile, C., Falcarin, P., Torchiano, M., Coppens, B., De Sutter, B.: Understanding the behaviour of hackers while performing attack tasks in a professional setting and in a public challenge. Empirical Software Engineering **24**(1), 240–286 (2019)
[35] Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: J. Feigenbaum (ed.) Digital Rights Management, pp. 1–15. Springer Berlin Heidelberg (2003)
[36] Cleven, A., Gubler, P., Hüner, K.M.: Design alternatives for the evaluation of design science research artifacts. In: Proc. 4th Int'l Conf. on Design Science Research in Information Systems and Technology, DESRIST '09. Association for Computing Machinery, New York, NY, USA (2009). DOI 10.1145/1555619.1555645
[37] Collberg, C., Martin, S., Myers, J., Nagra, J.: Distributed application tamper detection via continuous software updates. In: Proc. of the 28th Annual Computer Security Applications Conference, p. 319–328. ACM (2012)
[38] Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Computer Science Technical Reports 148, Dep. of Computer Science, University of Auckland, New Zealand (1997)
[39] Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: Proc. 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 184–196 (1998)
[40] Coppens, B., De Sutter, B., Maebe, J.: Feedback-driven binary code diversification. ACM Transactions on Architecture and Code Optimization (TACO) **9**(4), 1–26 (2013)
[41] Coppens, B., et al.: ASPIRE Open Source Manual. Deliverable D5.13, ASPIRE EU FP7 Project (2016). URL https://aspire-fp7.eu/sites/default/files/D5.13-ASPIRE-Open-Source-Manual.pdf
[42] Dawes, R.M.: The robust beauty of improper linear models in decision making. American psychologist **34**(7), 571 (1979)
[43] De Mulder, Y., Wyseur, B., Preneel, B.: Cryptanalysis of a perturbated white-box AES implementation. In: G. Gong, K.C. Gupta (eds.) Progress in Cryptology-INDOCRYPT 2010, pp. 292–310. Springer Berlin Heidelberg (2010)
[44] De Sutter, B., Collberg, C., Dalla Preda, M., Wyseur, B.: Software Protection Decision Support and Evaluation Methodologies (Dagstuhl Seminar 19331). Dagstuhl Reports **9**(8), 1–25 (2019)
[45] Dempsey, Kelley and Takamura, Eduardo and Eavy, Paul and Moore, George: NISTIR 8011 Vol. 4. Automation Support for Security Control Assessments: Software Vulnerability Management. Tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States (2020)
[46] Denning, D., Neumann, P.G.: Requirements and model for ides – a real-time intrusion-detection expert system. Tech. rep., SRI International, Menlo Park, CA, USA (1985)
[47] Doerry, N., Sibley, M.: Monetizing risk and risk mitigation. Naval Engineers Journal **127**, 35–46 (2015)
[48] van der Ende, M., Hageraats, M., Poort, J., Quintais, J.P., Yagafarova, A.: Global online piracy study 2018. Online at https://www.ivir.nl/projects/global-online-piracy-study/ (2018)
[49] Eronen, P., Zitting, J.: An expert system for analyzing firewall rules. In: Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001), pp. 100–107 (2001)
[50] European Union Agency for Fundamental Rights (FRA): The general data protection regulation – one year on. Tech. rep., European Union Agency for Fundamental Rights (FRA) (2019)
[51] Gartner, Inc.: Risk assessment process and methodologies primer for 2019. Online at https://www.gartner.com/en/documents/3938592 (2019)

[52] Gartner, Inc.: Cybersecurity labor shortage and COVID-19. Online at https://www.gartner.com/en/human-resources/research/talentneuron/cybersecurity-labor-shortage-and-covid-19 (2020)
[53] Heffner, K., Collberg, C.: The obfuscation executive. In: K. Zhang, Y. Zheng (eds.) Information Security, pp. 428–440. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
[54] Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS quarterly pp. 75–105 (2004)
[55] Hoffman, L.J.: Risk analysis and computer security: bridging the cultural gaps. In: Proceedings of the 9th National Computer Security Conference, pp. 156–161. National Institute of Standards and Technology (1986)
[56] Holder, W., McDonald, J.T., Andel, T.R.: Evaluating optimal phase ordering in obfuscation executives. In: Proc. 7th Software Security, Protection, and Reverse Engineering Workshop, SSPREW-7. ACM (2017)
[57] Horváth, M., Buttyán, L.: Cryptographic Obfuscation: A Survey. Springer (2020)
[58] Irdeto: Irdeto global connected industries cybersecurity survey. Online at https://go.irdeto.com/connected-industries-cybersecurity-survey-report/ (2020)
[59] Information technology - Security techniques - Information security management systems - Overview and vocabulary. Standard, International Organization for Standardization, International Electrotechnical Commission, Geneva, CH (2018)
[60] Information technology - Security techniques - Information security management - Monitoring, measurement, analysis and evaluation. Standard, International Organization for Standardization, International Electrotechnical Commission (2016)
[61] Joint Task Force: SP 800-37. risk management framework for information systems and organizations (revision 2). Tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States (2018)
[62] Joint Task Force: SP 800-53. security and privacy controls for information systems and organizations (revision 5). Tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States (2020)
[63] Joint Task Force Transformation Initiative: SP 800-39. managing information security risk: Organization, mission, and information system view. Tech. rep., National Institute of Standards & Technology (2011)
[64] Kahneman, D.: Thinking, fast and slow. Macmillan (2011)
[65] Kahneman, D., Klein, G.: Conditions for intuitive expertise: a failure to disagree. American psychologist **64**(6), 515 (2009)
[66] Kaltz, J., Lindell, Y.: Introduction to modern cryptography: principles and protocols. Chapman and Hall (2008)
[67] Kellerer, H., Pferschy, U., Pisinger, U.: Knapsack Problems. Springer-Verlag (2004)
[68] Khanmohammadi, K., Hamou-Lhadj, A., Ebrahimi, N., Khoury, R.: Empirical study of android repackaged applications. Empirical Software Engineering **24**(6), 3587–3629 (2019)
[69] Kim, J.S., Kim, M., Noh, B.N.: A fuzzy expert system for network forensics. In: Computational Science and Its Applications (ICCSA), pp. 175–182. Springer Berlin Heidelberg (2004)
[70] Knight, A.: In plain sight: The vulnerability epidemic in financial mobile apps (April 2019)
[71] Kuznetsov, V., Szekeres, L., Payer, M., Candea, G., Sekar, R., Song, D.: Code-pointer integrity. In: 11th USENIX Symposium on Operating Systems Design and Implementation, pp. 147–163. USENIX Association (2014)
[72] Liao, N., Tian, S., Wang, T.: Network forensics based on fuzzy logic and expert system. Comput. Commun. **32**(17), 1881–1892 (2009)
[73] Linn, C., Debray, S.: Obfuscation of executable code to improve resistance to static disassembly. In: Proceedings 10th ACM conference on Computer and communications security, pp. 290–299. ACM, New York, NY, USA (2003)
[74] Liu, H.: Towards better program obfuscation: Optimization via language models. In: Proc. 38th Int'l Conference on Software Engineering Companion, ICSE '16, pp. 680–682. Association for Computing Machinery (2016)
[75] Liu, H., Sun, C., Su, Z., Jiang, Y., Gu, M., Sun, J.: Stochastic optimization of program obfuscation. In: Proceedings of the 39th International Conference on Software Engineering, ICSE '17, pp. 221–231. IEEE Press (2017)
[76] Lukyanenko, R., Evermann, J., Parsons, J.: Instantiation validity in is design research. In: Advancing the Impact of Design Science: Moving from Theory to Practice: 9th International Conference, DESRIST 2014, Miami, FL, USA, May 22-24, 2014. Proceedings 9, pp. 321–328. Springer (2014)
[77] Lukyanenko, R., Evermann, J., Parsons, J.: Instantiation validity in is design research. In: M.C. Tremblay, D. VanderMeer, M. Rothenberger, A. Gupta, V. Yoon (eds.) Advancing the Impact of Design Science: Moving from Theory to Practice, pp. 321–328. Springer International Publishing, Cham (2014)
[78] Macher, G., Schmittner, C., Veledar, O., Brenner, E.: ISO/SAE DIS 21434 automotive cybersecurity standard - in a nutshell. In: Proceedings of Computer Safety, Reliability, and Security (SAFECOMP) Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, pp. 123—-135. Springer-Verlag (2020). DOI 10.1007/978-3-030-55583-2_9
[79] Mandiant: 2020 security effectiveness: Deep dive into cyber reality. Online at https://content.fireeye.com/security-effectiveness/rpt-security-effectiveness-2020-deep-dive-into-cyber-reality (2020)
[80] Mantovani, A., Aonzo, S., Fratantonio, Y., Balzarotti, D.: RE-Mind: a first look inside the mind of a reverse engineer. In: Proc. 32st Usenix Security Symposium (2022). To appear

[81] McCabe, T.J.: A complexity measure. IEEE Transactions on Software Engineering **SE-2**(4), 308–320 (1976)
[82] Merlo, A., Ruggia, A., Sciolla, L., Verderame, L.: You shall not repackage! demystifying anti-repackaging on android. Computers and Security **103**, 102,181 (2021)
[83] Nagra, J., Collberg, C.: Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Pearson Education, London, UK (2009)
[84] Osbeck, L.M., Held, B.S.: Rational intuition: Philosophical roots, scientific investigations. Cambridge University Press (2014)
[85] OWASP Application Security Verification Standard v4.03 (2021). URL https://owasp.org/www-project-application-security-verification-standard/
[86] OWASP Software Assurance Maturity Model v2.0 (2020). URL https://owaspsamm.org/
[87] Owens, S.F., Levary, R.R.: An adaptive expert system approach for intrusion detection. Int. J. Secur. Netw. **1**(3/4), 206–217 (2006)
[88] Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 Workshop on New Security Paradigms, NSPW '98, pp. 71–79. ACM (1998)
[89] Prat, N., Comyn-Wattiau, I., Akoka, J.: A taxonomy of evaluation methods for information systems artifacts. Journal of Management Information Systems **32**(3), 229–267 (2015). DOI 10.1080/07421222.2015.1099390
[90] Regano, L.: An expert system for automatic software protection. Ph.D. thesis, Politecnico di Torino (2019)
[91] Regano, L., Canavese, D., Basile, C., Lioy, A.: Towards optimally hiding protected assets in software applications. In: Proc. Int'l Conf. on Software Quality, Reliability and Security, pp. 374–385. IEEE Computer Society (2017)
[92] Regano, L., Canavese, D., Basile, C., Viticchié, A., Lioy, A.: Towards automatic risk analysis and mitigation of software applications. In: Information Security Theory and Practice, pp. 120–135. Springer International Publishing (2016)
[93] Rolles, R.: Unpacking virtualization obfuscators. In: Proceedings of the 3rd USENIX Conference on Offensive Technologies, WOOT'09, pp. 1–1. USENIX Association (2009)
[94] Schrittwieser, S., Katzenbeisser, S., Kinder, J., Merzdovnik, G., Weippl, E.: Protecting software through obfuscation: Can it keep pace with progress in code analysis? ACM Comput. Surv. **49**(1), 4:1–4:37 (2016)
[95] Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C., Kruegel, C., Vigna, G.: Sok: (state of) the art of war: Offensive techniques in binary analysis. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 138–157. IEEE Computer Society (2016)
[96] Souppaya, Murugiah and Scarfone, Karen: SP 800-40 Rev. 4. Guide to Enterprise Patch Management Planning: Preventive Maintenance for Technology. Tech. rep., National Institute of Standards & Technology (2017)
[97] Tamada, H., Fukuda, K., Yoshioka, T.: Program incomprehensibility evaluation for obfuscation methods with queue-based mental simulation model. In: 13th ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing, pp. 498–503 (2012)
[98] Tassey, G.: Standardization in technology-based markets. Research policy **29**(4-5), 587–602 (2000)
[99] Taylor, C., Collberg, C.: Getting RevEngE: a system for analyzing reverse engineering behavior. In: 14th International Conference on Malicious and Unwanted Software (MALCON) (2019)
[100] Tsudik, G., Summers, R.C.: Audes - an expert system for security auditing. In: Proceedings of the The Second Conference on Innovative Applications of Artificial Intelligence, pp. 221—-232. AAAI Press (1990)
[101] Tuma, K., Calikli, G., Scandariato, R.: Threat analysis of software systems: A systematic literature review. Journal of Systems and Software **144**, 275–294 (2018)
[102] Van den Broeck, J., Coppens, B., De Sutter, B.: Flexible software protection. Computers & Security **116**, 102,636 (2022)
[103] Van Put, L., Chanet, D., De Bus, B., De Sutter, B., De Bosschere, K.: DIABLO: a reliable, retargetable and extensible link-time rewriting framework. In: Proc. Fifth IEEE Int'l Symposium on Signal Processing and Information Technology, pp. 7–12. IEEE Computer Society (2005)
[104] Venable, J., Pries-Heje, J., Baskerville, R.: FEDS: a framework for evaluation in design science research. European Journal of Information Systems **25**(1), 77–89 (2016). DOI 10.1057/ejis.2014.36
[105] Viticchié, A., Basile, C., Avancini, A., Ceccato, M., Abrath, B., Coppens, B.: Reactive attestation: Automatic detection and reaction to software tampering attacks. In: Proceedings of the 2016 ACM Workshop on Software PROtection, SPRO '16, p. 73–84. ACM (2016)
[106] Wang, C., Hill, J., Knight, J., Davidson, J.: Software tamper resistance: Obstructing static analysis of programs. Tech. rep., University of Virginia, Charlottesville, VA, USA (2000)
[107] Wang, H., Fang, D., Wang, N., Tang, Z., Chen, F., Gu, Y.: Method to evaluate software protection based on attack modeling. In: Int'l Conf. on High Performance Computing and Communications (HPCC) & Int'l Conf. on Embedded and Ubiquitous Computing (EUC), pp. 837–844. IEEE Computer Society (2013)
[108] Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming **12**(1-2), 67–96 (2012)

[109] Williams, J.: OWASP Risk Rating Methodology. Online at https://owasp.org/www-community/OWASP_Risk_Rating_Methodology (1930)
[110] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering - An Introduction. Kluwer Academic Publishers (2000)
[111] Wyseur, B.: White-box cryptography. Ph.D. thesis, KU Leuven (2011)
[112] Wyseur, B.: Assets. In: ASPIRE Project Deliverable 1.02: Attack Model, pp. 10–12 (2014)
[113] Wyseur, B.: Classification of attacks. In: ASPIRE Project Deliverable 1.02: Attack Model, pp. 13–52 (2014)
[114] Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of white-box DES implementations with arbitrary external encodings. In: C. Adams, A. Miri, M. Wiener (eds.) Selected Areas in Cryptography, pp. 264–277. Springer Berlin Heidelberg (2007)
[115] Yadegari, B., Johannesmeyer, B., Whitely, B., Debray, S.: A generic approach to automatic deobfuscation of executable code. In: Proc. Symposium on Security and Privacy, pp. 674—-691. IEEE Computer Society (2015)
[116] Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy, pp. 95–109 (2012). DOI 10.1109/SP.2012.16
[117] Zumerle, D., Bhat, M.: Gartner: Market guide for application shielding (2017)

Appendix A. The questionnaire for the expert assessment of the ESP.

This questionnaire is semantically equivalent to the questionnaire provided to the experts during the ASPIRE project. We removed ASPIRE-specific terms, which have been substituted with a wording coherent with this paper, and slightly rephrased some sentences for clarity.

Preparation
The primary Assets are the ones selected and protected for preparing the application for the tiger team experiments. The assets' relevance has been assigned by *NAME* during the *PLACE* meeting.
NOTE: In general, any feedback on the visualisation, presentation, data, and options in both the tool and the report is welcome.

Application Parts

1. Is the list of the APPLICATION PARTS useful for your software protection purposes/tasks?
2. Report if you want to see them differently or if some information is missing.

Attack Steps

1. VALIDATE if the attack steps identified by the automatic analysis are meaningful. Check if they are exhaustive.
2. For all the attack steps, VALIDATE if the *Suggested Protections* are correct, sound, proper, and effective. Report if the estimated attack EFFECTIVENESS is correct.
3. REPORT if the attack data shown are useful. Report if you want to see them differently or if you miss some important information.

Golden Configurations

1. VALIDATE if the golden combinations are meaningful and you consider them effective/optimal.
2. For all the 10 golden combinations presented in the tool and report, you should NOTIFY us if you noticed something strange in the use of the protection techniques, like some association/combination of techniques which is strange according to your expert judgment, the use of some techniques to protect specific assets that you may consider anomalous
3. REPORT us if the data shown are useful. Report if you want to see them differently or if you miss some important information.

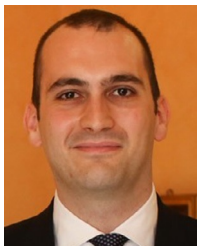
Asset hiding

1. VALIDATE if the layer-two protections (asset hiding) and check if they are effective/optimal.
2. NOTIFY if you noticed something anomalous in the use of techniques, e.g., some association of techniques which is strange for experts, the use of techniques for assets in anomalous/inappropriate ways, or some cases where you wouldn't extend/randomly pick other areas.
3. REPORT us if the data shown are useful. Report if you want to see them differently or if you miss some important information.
4. REPORT if the estimated maximum degradation thresholds (overheads sections of the Golden Configurations and Asset Hiding) are reasonable.

Cataldo Basile is an assistant professor at the Politecnico di Torino, from which he received an M.Sc.in 2001 and a Ph.D. in Computer Engineering in 2005. His research concerns software protection, software attestation, policy-based security management, and general models for detecting, resolving, and reconciling security policy conflicts.

Bjorn De Sutter is associate professor at Ghent University in the Computer Systems Lab. He obtained his MSc and PhD degrees in Computer Science from the university's Faculty of Engineering in 1997 and 2002. His research focuses on techniques to aid programmers with non-functional aspects such as performance and software protection to mitigate reverse engineering, software tampering, code reuse attacks, fault injection, and side channel attacks. He co-authored over 100 papers.

Daniele Canavese received an M.Sc. degree in 2010 and a Ph.D. in Computer Engineering in 2016 from Politecnico di Torino, where he is currently a research assistant. His research interests are concerned with security management via machine learning and inferential frameworks, software protection systems, public-key cryptography, and models for network analysis.



Leonardo Regano received an M.Sc.degree in 2015 and a Ph.D. in Computer Engineering in 2019 from Politecnico di Torino, where he is currently a research assistant. His current research interests focus on software security, artificial intelligence and machine learning applications to cybersecurity, security policies analysis, and software protection techniques assessment.

Bart Coppens is a assistant professor at Ghent University in the Computer Systems Lab. He received his PhD in Computer Science Engineering from the Faculty of Engineering and Architecture at Ghent University in 2013. His research focuses on protecting software against different forms of attacks using compiler-based techniques and run-time techniques.