Two Architectural Threat Analysis Techniques Compared

Katja Tuma and Riccardo Scandariato

Chalmers | University of Gothenburg, Sweden
katja.tuma@cse.gu.se, riccardo.scandariato@cse.gu.se

Abstract. In an initial attempt to systematize the research field of architectural threat analysis, this paper presents a comparative study of two threat analysis techniques. In particular, the controlled experiment presented here compares two variants of Microsoft's STRIDE. The two variants differ in the way the analysis is performed. In one case, each component of the software system is considered in isolation and scrutinized for potential security threats. In the other case, the analysis has a wider scope and considers the security threats that might occur in a pair of interacting software components. The study compares the techniques with respect to their effectiveness in finding security threats (benefits) as well as the time that it takes to perform the analysis (cost). We also look into other human aspects which are important for industrial adoption, like, for instance, the perceived difficulty in learning and applying the techniques as well as the overall preference of our experimental participants.

Keywords: empirical study, secure software, threat analysis, STRIDE

1 Introduction

After decades of research and knowledge transfer in the field of "security by design", the software-intensive industries have absorbed the idea that security needs to be addressed throughout the software development lifecycle. Building Security In Maturity Model (BSIMM) [12] collects statistics from 95 companies and gauges their level of adoption with respect to several secure software development techniques. According to the report, security-specific code analysis techniques have successfully found their way into the industrial practice, as two thirds of the surveyed companies adopt them routinely. In this respect, the availability of well-known automated tools has helped significantly. *Architectural threat analysis* is another important pillar of building more secure software and the above-mentioned BSIMM report mentions that about one third of the surveyed companies use architectural threat analysis techniques, like Microsoft's STRIDE [20], attack trees [19], Trike [16], CORAS [11], PASTA [24], threat patterns [2], to cite a few.

Working in collaboration with our industrial partners from the automotive industry, we noticed that Microsoft's STRIDE is well-known and often used. In

particular, our partners use the so-called STRIDE-per-element version. In this version, each component of the software system is considered in isolation and scrutinized for potential security threats. However, practitioners advocate for a threat analysis technique that allows them to analyze end-to-end scenarios where several components interact (e.g., to provide a given functionality). In this respect, the STRIDE-per-interaction variant could be more appropriate, as in this variant the analysis has a slightly wider scope and considers the security threats that might occur in a pair of interacting software components. On the other hand, there are also truly end-to-end analysis techniques, like for instance the one proposed by Tuma et al. [23]. From our perspective, it is interesting to study how these alternative techniques differ across the spectrum (analysis of isolated components vs analysis of pair-wise interactions vs analysis of end-to-end scenarios) in terms of performance. In essence, which approach to threat analysis produces more results in a faster way? Consequently, this study focuses on the differences between the analysis of isolated components and the analysis of pair-wise interactions. In current work, we are also comparing the analysis of isolated components with the analysis of end-to-end scenarios.

In the latest publication by Shostack [20] describing Microsoft's STRIDE, the author describes two variants that are dubbed 'STRIDE per element' (analysis of isolated components) and 'STRIDE per interaction' (analysis of pair-wise interactions). A more detailed description of the two is provided in Section 2. In our study, we divide our participants (110 master students) into two treatment groups (Element vs Interaction), each using one of the two variants of STRIDE to analyze the architectural design of an Internet-of-Things system. For replication purposes of this study, we have created a *companion web-site [1]*, where all the material used during the experiment is available. The study analyzes and compares the effectiveness of the two variants in unearthing security treats (benefits) as well as the time that it takes to perform the analysis (cost). We also look into other human aspects which are important to adoption, like the perceived difficulty in learning and applying the techniques as well as the overall preference of our participants.

The rest of the paper is organized as follows. Section 2 provides a primer on the STRIDE variants. Section 3 describes the experiment and states the research hypotheses. Section 4 presents the results, while Section 5 discusses them. The threats to validity are listed in Section 6. Section 7 discusses the related work and Section 8 presents the concluding remarks.

2 Treatments

STRIDE is a threat analysis approach developed to help people identify the types of attacks their software systems are exposed to, especially because of design-level flaws. The name itself is an acronym that stands for the threat categories of Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. For the definition of threat categories, we refer the reader to the documentation of STRIDE [20].

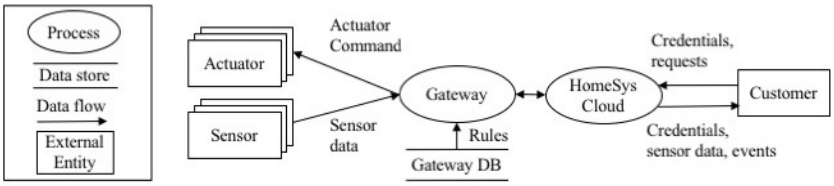

Fig. 1. A high-level DFD of the experimental object.

The analysis is based on a graphical representation of the system architecture as a Data Flow Diagram (DFD). As shown in Figure 1, a DFD represents how information moves around in a software-based system. The diagram consists of processes (active entities), data flows (exchanged info), external entities (e.g., users or 3rd parties), data stores (e.g., file system) and trust boundaries.

The first step in applying the STRIDE methodology is to create a DFD using the available system documentation. The second step is a systematic exploration of the DFD graph in order to identify the threats. The two STRIDE variants differ in how this exploration is carried out.

STRIDE per element. Using this approach, the analyst visits every element in the diagram (e.g., starting in the top-left corner). For each element type, STRIDE advises looking into a subset of threat categories. To this aim, STRIDE provides a table mapping element types to threat categories. For instance, the Sensor in Figure 1 is an external entity, therefore according to STRIDE, the analyst should look into Spoofing and Repudiation threats. For each pair of element type and threat category STRIDE also provides a catalog of example threats that can be used for inspiration by the analyst. With reference to the Sensor, one provided example threat for spoofing a hardware device is IP spoofing.

STRIDE per interaction. This technique adopts an approach of systematically visiting the interactions in a DFD. Interactions are patterns of DFD elements connected via data flows. The analyst has to first identify the interactions. For instance, "Sensor sends sensor data to Gateway" is a match for the above-mentioned interaction. For each type of interaction STRIDE again advises looking into a subset of threat categories and provides a catalog of example threats. For example, when an external entity is passing input to a process, the analyst is advised to look into Spoofing and Repudiation threats. If there is no logging in place, the Gateway is able to deny having received sensitive information from the Sensor.

3 The experiment

This section presents the design of a controlled experiment, conducted with participants in an academic setting.

3.1 Experimental object

As depicted in Figure 1, the Home Monitoring System (HomeSys) is a system for remote monitoring of residential homes. The purpose of this system is to provide necessary tools for customers to automatically receive and manage notifications about critical events in their homes. The system consists of a smart home gateway which communicates with sensors and actuators and a cloud system which collects data from the gateways and offers a dashboard to the customers. Sensors are analog or digital hardware devices that produce measurements and send them to the gateway. This system includes sensors that detect temperature, humidity, smoke, etc. Actuators are hardware devices that can receive commands from the gateway, like for instance, taking a picture, activating a buzzer or flicking a switch. The gateway is a hardware device which relays measurements to the cloud (via a 3G or WiFi network) and manages the actuators in the residency. The HomeSys cloud is a software system that communicates with the gateways and provides services for the customers, as well the operators of the system.

The system documentation (about 30 pages) includes (1) the description of the problem domain with scenarios, (2) the requirements of the system and (3) a hierarchical architectural description documented in UML. For instance, the documentation includes a UML deployment diagram. The complete description of the system is available with the experimental material [1]. The participants worked on the HomeSys system throughout the entire course before entering the experiment. Therefore, they were very familiar with the object of the experiment and had enough knowledge about the system to understand the problem and complete their task.

3.2 Participants

The participants of this study are 110 first-year master students of Software Engineering, attending a course on "Advanced Software Architecture", taught by the experimenters. In order to gather sufficient data, we have repeated the experiment for two consecutive academic years (2016 and 2017). The participants performed the assigned tasks in teams. Each year, the participants were randomly grouped into teams of about 4 students and assigned one analysis techniques (i.e., treatment groups). In total, we have assigned 14 teams to the Element and 13 teams to the Interaction treatment.

We have collected information with a short questionnaire before the study took place to investigate the background of the participants. As shown in Table 1, it included questions about participants' work experience and perceived familiarity with task-related concepts. Most participants have had previous experience in software development outside the university and consider to have adequate knowledge about software design and programming. A large majority of the participants are able to use UML, which is relevant as the study object is documented in such language. The course does not require background knowledge of information security, hence the participants consider to have limited expertise in this area, as expected.

Table 1. Answers to the entry questionnaire.

Questions and answers [%]			
1. Do you have any working experience in software development outside the university?			
Yes (63)	No (37)		
2. How would you describe your working experience outside the university?			
Profit (39)	Non-profit (8)	Both (27)	NA (26)
3. How would you rate your level of expertise as a programmer?			
Very insufficient (1)	Limited (17)	**Adequate (58)**	Advanced (24)
4. How would you rate your level of familiarity with software design, including the use of UML?			
Very insufficient (2)	Limited (33)	**Adequate (63)**	Advanced (2)
5. How would you rate your level of expertise in security?			
Very insufficient (16)	**Limited (65)**	Adequate (16)	Advanced (3)

3.3 Task

The teams were presented with the same task on the same experimental object. The task was divided into two sub-tasks: participants were asked to (1) build a DFD based on the provided architectural documentation and (2) analyze the DFD according to the assigned technique.

During the training, participants were provided with guidelines for creating the DFD. First, they had to create a DFD by mapping the nodes from a given deployment diagram into DFD elements. Second, the participants had to use the rest of the documentation (e.g., component and sequence diagrams) to refine the DFD and identify the data flows. The details of training are available online [1].

The second sub-task required a systematic analysis of the DFD according to the techniques described in Section 2. The analysis results had to be documented in a report and submitted electronically. The report had to contain a list of identified threats and corresponding descriptions. Threat descriptions were made according to a provided template (available online [1]). The purpose of the template is to simplify and standardize the analysis of the reports. Note that the task has been performed during a supervised lab session. In the lab, the teams were instructed to keep an informal log of the identified threats (lab notes). The preparation of the official report had to be done after the supervised lab. However, we have monitored that the reports did not contain more threats with respect to the work done in the lab (e.g., by taking snapshots in the lab). The snapshots taken during the lab were compared with the final report to capture any threats identified outside the supervised lab. We have not recorded any discrepancies between the snapshots and the reported threats.

Finally, we asked the teams to keep track of the time they spent. To this aim, the teams were instructed to use an online time-tracking tool (`www.toggl.com`) and submitted their time-sheets electronically at the end of the supervised lab.

3.4 Execution of the study

The experiment is positioned at the end of a course on software architecture. The topic covered in the experiment aligns with the course content. Participation in

the study contributes to the teaching objectives of the course, hence participants were highly motivated. For more details about the experimental material please refer to the companion web site [1].

Entry questionnaire. A few weeks before the beginning of the study, the participants have been asked to fill in a brief questionnaire about their knowledge and background (see Section 3.2).

Training. As part of training for the experiment, the participants attended 2 lectures (mandatory 4 hours of training). In the first lecture (2 hours) the participants got an introduction to secure design and the use of the DFD notation. The lecture also included a practical exercise on how to build a DFD for a system of similar size as HomeSys. For the second lecture (2 hours) participants were split according to their assigned treatment group. Each group received a dedicated lecture explaining the philosophy of STRIDE specific to their treatment group. In addition, participants received only documentation specific to their treatment group. This was done in order to limit the problem of treatment diffusion. An overview of the HomeSys documentation was also given in the second lecture. The students were more than familiar with the system, but the experimenters wanted to be sure that they would be able to navigate the documentation without hiccups. Finally, in a lab session preceding the experiment, the participants were familiarized with the time-keeping tool.

Supervised lab. The experiment took place in one lab session of 4 hours. The session was supervised by the authors and two teaching assistants. At the beginning of the lab, the authors explained the experimental protocol to the participants, e.g., by summarizing the task, mentioning all the provided material, and reminding the participants about the time-tracking tool. Each team was provided with a printed copy of task description, the relevant book chapters, and the documentation of HomeSys. The teams performed the assigned task and kept track of their work by writing lab notes.

Report. The participants were given about a week to write a report documenting the threats they had found during the lab. To this aim, they used their lab notes. Each report contained a figure of the DFD and a list of identified threats, where each threat was documented according to a provided template [1]. In particular, each threat is described with a title, a position in the DFD where the threat is located, a threat category (STRIDE), required attacker capabilities, and a detailed description of the threat itself. The participants were also asked to document their assumptions about the system.

Exit questionnaire. At the end of the lab session, the participants were asked to fill in an exit questionnaire. Access to the questionnaire was open for a week after the lab session had finished, during which time a few reminders were sent by email. As discussed later, this questionnaire is meant to validate some experimental assumptions (e.g., the participants understood the task and were adequately prepared to carry it out) and to collect additional information about the treatments (e.g., the perceived difficulty of the tasks).

3.5 Measures

We have collected the measure of effort (in minutes) spent by each team on both sub-tasks (DFD creation and threat analysis).

We have also collected the measure of true positives (TP), false positives (FP) and false negatives (FN). True positives are reported threats that are assessed as correct by the experimenters in light of the analyzed DFD and the security assumptions made by the team. False positives are wrong or unrealistic threats reported by the team. Finally, false negatives are threats that are present in the analyzed system but had gone unnoticed by the team.

In order to record the correct threats a "ground truth" has been created by the first author. Incidentally, we decided to let the teams produce their DFD as this activity is an integral part of threat analysis in practice. Ergo, the teams have analyzed slightly different DFDs. A ground truth was built *for each team* to ensure a correct evaluation. For each report, the ground truth was used to identify the correct, incorrect and overlooked threats. In particular, a threat is considered correct if (1) it is identified at the correct location, (2) it is correctly categorized, (3) it has some impact on system assets, (4) the description of the threat agent is correct and (5) the description provided by the team is realistic from a security perspective and does not contradict their assumptions. Oftentimes the teams reported the same threat more than once, using a different title. A threat (either correct or incorrect) that is identified more than once is marked as a duplicate. Note that duplicated threats are intentionally not considered as TP or FP.

3.6 Hypothesis

We have adopted a standard design for a comparative study of one independent variable with two values, i.e., the two treatments of Element and Interaction. Our study investigates three dependent variables: productivity, precision, and recall. In this study we define the *productivity* ($Prod$) of a team as the number of correct threats (TP) per time unit. For each team, *precision* (P) is the percentage of correctly identified threats out of the total number of reported threats ($TP/(TP+FP)$). *Recall* (R) is the percentage of correctly identified threats out of the total number of existing threats ($TP/(TP+FN)$).

We use the Wilcoxon statistical test to determine whether there is a statistical difference in the three dependent variables across the two treatment groups. Accordingly, the null hypotheses are as follows:

H_0^{Prod} : *There is no statistically significant location shift between the average productivity of the two treatment groups.*

H_0^{P} : *There is no statistically significant location shift between the average precision of the two treatment groups.*

H_0^{R} : *There is no statistically significant location shift between the average recall of the two treatment groups.*

4 Results

In this section, we present the results of the study and answer to research questions. All statistical tests have been performed using the two-sample Wilcoxon test of independence with a level of significance equal to 0.05.

4.1 True positives, false positives, and false negatives

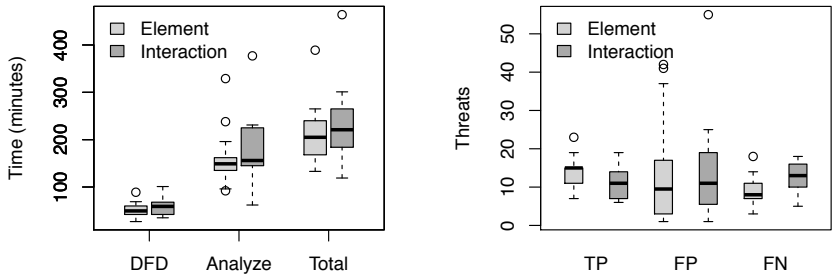

Fig. 2. Average time per sub-task and total time of treatment groups (left) and true positives, false positives, and false negatives (right).

Figure 2 reports the number of TP, FP and FN per treatment group. We have observed slightly better averages for the ELEMENT group, compared to the INTERACTION. Namely, the TP is higher (ELEMENT: 14.3, INTERACTION: 11.6), the average number of FP is lower (ELEMENT: 14.2, INTERACTION: 15) and the average number of FN is also lower (ELEMENT: 8.8, INTERACTION: 11.8). However, the analysis shows that there are no significant differences between the amount of TP, FP and FN across treatments groups.

The average number of TP, FP and FN per threat category is depicted in Figure 3. Overall, both treatment groups visibly focused less on Repudiation and Elevation of Privilege threats compared to other threat categories. A statistical analysis shows that there are significant differences between the TP of the Denial of Service (p-value = 00.02) and Tampering (p-value = 00.007) threat categories across treatments. For the Denial of Service threat category, the ELEMENT treatment group identified on average more TP (statistically significant, p-value = 00.02) and less FN (not significant). For the Tampering threat category, the ELEMENT treatment group identified on average less FN, more TP (statistically significant, p-value = 00.007), and less FP. This might be due to the two methods providing different mapping tables (from DFD to threat categories [20]). The INTERACTION has a lower rate of mappings to the Denial of Service threat category (8/72 = 11% vs 3/20 = 15%). We have also computed the recall when the Denial of Service threats are removed. The results stay similar to what is reported in Figure 4, i.e., the median recall is not "driven" by the Denial of Service category. Incidentally, there is a similar situation in the mappings for the Tampering category (3/72 = 4% vs 3/20 = 15%). Identified threats from other categories do not differ across treatment groups.

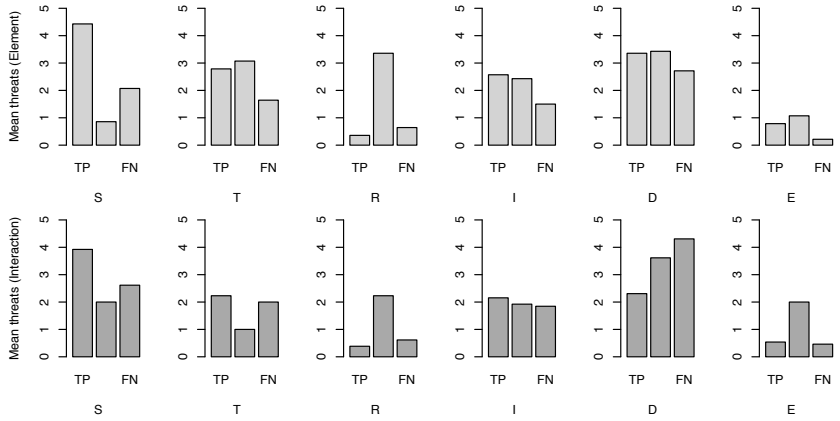

Fig. 3. The mean number of identified threats for per element (top) and per interaction treatment (bottom).

4.2 RQ1: Productivity

As shown in Figure 2, the average time spent by the teams performing STRIDE per element is 3.5 hours, whereas the average time spent per teams performing STRIDE per interaction is 3.95 hours (not statistically significant). There is a noticeable difference between time spent on the sub-tasks, with the analysis time being dominant. When looking at differences across the treatments, the ELEMENT group was on average faster in performing both sub-tasks (not statistically significant). It is interesting to notice, that even though both treatments followed the same guidelines for DFD creation, the INTERACTION group created on average DFDs with more elements (discussed in Section 5).

The overall productivity of a technique depends on the amount of correctly identified threats (TP). The average productivity of the ELEMENT group is **4.35** TP/h[1]. The INTERACTION group turned out to be less productive (**3.27** TP/h). However, the difference is not statistically significant and, hence, the null hypothesis H_0^{Prod} cannot be discarded.

As a possible explanation for lower productivity of the INTERACTION treatment, we highlight that the documentation of the STRIDE per interaction variant is more complex with regard to mapping threats to interactions. As mentioned by Shostack, "STRIDE-per-interaction is too complex to use without a reference chart handy" [20]. Such a reference chart was available to the participants, yet the complexity might still have been too high.

4.3 RQ2: Precision

Figure 4 presents the precision (i.e., the correctness of analysis) of the two treatment groups as an aggregate (left-hand side) and across each individual threat

[1] Scandariato et al. [18] reported an average productivity of 1.8 TP/h.

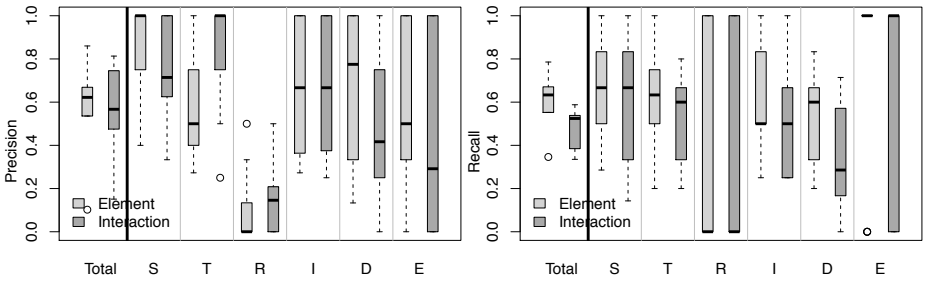

Fig. 4. Precision (left) and recall (right) aggregated across threat categories.

category (right). Overall, the mean precision is **0.60** for both treatment groups. Therefore, null hypothesis H_0^P cannot be rejected. We have also analyzed the differences in precision within threat categories. There is a statistically significant difference in the precision of Tampering threats (ELEMENT: 0.58, INTERACTION: 0.81, p-value = 00.027). A difference can also be observed for the Denial of Service category (not statistically significant).

4.4 RQ3: Recall

Shostack states that the "STRIDE-per-interaction leads to the same number of threats as STRIDE-per-element" [20]. Yet in our study, the number of reported threats was higher for the ELEMENT treatment, especially the number of FP (see Figure 2). Figure 4 presents the recall (i.e., completeness of analysis) of the two treatment groups as aggregate (left-hand side) and across each individual threat category (right). The average recall for the ELEMENT treatment is **0.62**, whereas the average recall of the INTERACTION is **0.49**. The difference is statistically significant (p-value = 0.028). Therefore, null hypothesis H_0^R can be rejected. We have also analyzed the differences within each threat category and found a statistically significant difference in the recall of the Denial of Service category (ELEMENT: 0.55, INTERACTION: 0.34, p-value = 00.014). One possible explanation relates to the fact that the ELEMENT group tends to create smaller DFDs, as discussed in Section 5. Alternatively, the difference could be linked to the fact that the threat examples in the documentation are more extensive in case of the ELEMENT treatment and the documentation of the INTERACTION treatment is more complex to navigate (as mentioned in Section 4.2). These are interesting hypotheses for future studies.

4.5 Exit questionnaire

In summary, the two treatments displayed a statistically significant difference only with respect to recall, with the ELEMENT group reporting more complete results (13% better). It is also important to appreciate how the two variants are perceived by the participants. This could have an impact on the successful

adoption of the technique and, hence, become a deciding factor beyond the performance indicators investigated in the three research hypotheses.

To this aim, we asked the participants to fill in a questionnaire at the end of the experiment. Due to space limitations, the questions and the answers are not shown here. They are available on the companion web-site [1] under the "Questionnaires" tab. To investigate differences across the treatment groups, we have performed a Cochran-Mantel-Haenszel test (similar to the Chi-square test) with a level of significance equal to 0.05.

In general, participants from both treatment groups agreed about having a clear understanding of the task, though they were sufficiently prepared and were familiar with the experimental object. Overall the task was not too difficult, while the first sub-task of creating the DFD was perceived easier than the second sub-task of analyzing the DFD (for both treatments). According to the documentation, "STRIDE per element is a simplified approach designed to be easily understood by the beginner" [20]. This implies that STRIDE per interaction is perceived by Microsoft as the technique to be used in production. However, according to our results, the techniques are the same (i.e. per element is not simpler than per interaction) in terms of productivity and precision. Concerning the learnability, the teams from both treatment groups agreed that the techniques they used were easy to understand and learn (no significant differences). Although the teams were given sufficient time to carry out the task, both treatment groups perceived the techniques as lengthy and tedious.

The participants from both treatment groups mostly believed that 50-75% of their identified threats were correct. This is a fairly accurate estimation according to the observed precision in this study (0.6). Interestingly, participants were slightly less confident about the completeness of their analysis. The aggregate recall over all teams (regardless of the treatment group) is 0.5 and only less than half of the participants (47.7%) have this perception. Finally, participants generally liked the technique they used but were not especially fond of it either. No significant differences were observed across treatments.

5 Discussion

DFD. The participants followed precise guidelines for DFD creation [1]. As such, the created DFDs have limited variability as well as consistent quality, e.g., we did not find many mistakes in the DFDs. In this study, we have made a deliberate choice to minimize the influence of the DFD creation (common to both treatments) and focus on the alternative techniques of analyzing the DFD (the key difference between the treatments). Figure 5 depicts the number of DFD elements in the models created by the teams. On average, the teams created DFDs with about 26 data flows, 6 processes, 4 data stores, 3 external entities, and 3 trust boundaries. A few differences can be noted. On average, we observed a smaller number of DFD elements in the Element group (37.4) compared to the Interaction group (41.9). This difference is consistent across the different element types, yet not statistically significant.

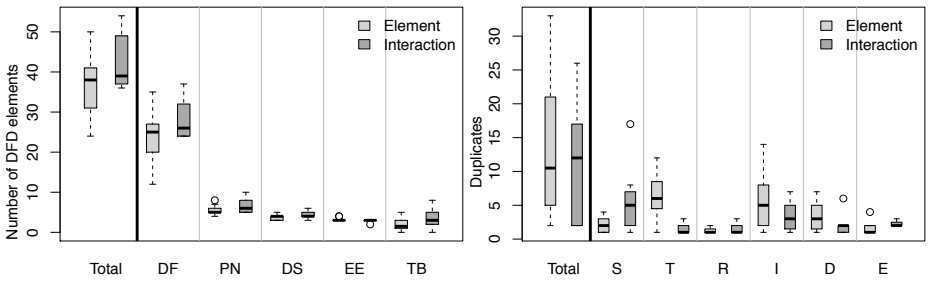

Fig. 5. The average number of DFD elements (DF=data flows, PN=process nodes, DS=data stores, EE=external entity, TB=trust boundaries) (left) and the average number of duplicated threats (right).

Mistakes. As reported in Figure 5, the teams sometimes reported several threats more than once. Duplicated threats are considered to slow down the analysis process, especially during threat prioritization. Note that the results about productivity (see Section 4.2) are not affected by duplicates, as they were discarded.

Most commonly, threats are duplicated due to (i) threat 'fabrication' or (ii) a misuse of the reduction technique. We consider threat fabrication as mistakenly identifying threats in order to achieve complete coverage of the STRIDE category mapping table. Proposed by STRIDE, threat reduction is a technique that aims towards minimizing the number of DFD elements that have to be analyzed. In particular, the reduction enables coupling the elements of the same type in order to analyze them at once. In other words, the threats identified for one DFD element may apply to other elements of the same type.

About 30% of all reported threats were duplicated. Of those, the majority belonged to the Element group (65%). The mean number of duplicated threats identified by the Element teams is bigger (6) than the Interaction teams (5) (not statistically significant). However, there is a significant difference in the amount of Tampering duplicates across treatments (Element: 6.27, Interaction: 1.67, p-value = 0.042). Incidentally, we also observed more Spoofing duplicates in the Interaction treatment. A possible explanation for fewer duplicates in the Interaction group is that the notion of interaction patterns might lend itself useful to a correct use of the reduction technique.

Interestingly, we observed that most teams correctly identified more outsider threats than insider threats. Very often, the reported insider threats were just unrealistic and assessed as false positives.

Analysis focus. Overall (including duplicated threats and FP), both treatment groups have focused their analysis on 'border' elements of the system, as well as the data flows that pierce through trust boundaries. The reports of the Element treatment group contain more threats to processes, external entities, and data stores compared to threats to data flows. In contrast, the reports of the Interaction treatment group contain more threats to data flows compared to threats to other types of DFD elements. In general, all teams were more likely to

identify correct threats to the data flows crossing a trust boundary. This confirms the usefulness of using trust boundaries to focus the attention of the analyst. However, there is a lack of precise guidelines for how and where trust boundaries should be placed, as our teams showed uncertainties in this regard. The teams were more likely to falsely identify threats (FP) to processes and data stores. The participants found the most commonly known threats (e.g., phishing, SQL injection, stealing for credentials or account). Coincidentally, these are more commonly identified on data flows. Correctly identifying a Tampering threat to a data store within system boundaries would mean finding a way to by-pass the system access control or even overcome obstacles like file-locking or other system defenses. This kind of threat requires correct assumptions and more security background. Unfortunately, most teams did not document many (if any) assumptions.

6 Threats to validity

The time spent for performing the task was measured by the participants themselves. To mitigate this threat, we have continuously reminded the participants to pause and continue measuring time during breaks. The amount of work that was reported was consistent with the reported time, which indicates that this is a minor concern. The use of student participants instead of professionals is a potential issue threatening the generalization of results. This kind of population sampling is sometimes referred to as *convenience sampling* [25]. It is considered controversial due to certain drawbacks [3]. However, studies have shown [6,15,17] that the differences between the performance of professionals and graduate students are often limited. The experiments were conducted by using teams of 3-5 students, which threatens the generalization of results to a single analyst. Nonetheless, the state-of-the-art [7, 20, 22] advises sound-boarding the analysis with a team of experts in the industrial setting as well. This is what happens in the medium-to-large companies we collaborate with. Finally, the results of this study may be influenced by the experimental object and in turn, may not be applicable to a system of different complexity or from a different domain.

Possible mistakes might have been made during the assessment of the reports and during the creation of the ground truth. In order to avoid over-loading the participants and make them tired, the supervised experiment was performed in a span of 4 hours. Additional time was given for documenting the identified threats outside the supervised lab. The teams were not monitored after the experiment has ended, however, we have made sure that the final report included only the threats in the lab notes (e.g., by taking snapshots in the lab).

7 Related work

McGraw conducted a study including 95 well-known companies [12]. The study analyzes the security practices that are in place in the companies. The BSIMM model does not mention STRIDE per se, rather it highlights the importance of

threat analysis. Microsoft has not published evidence of the effectiveness of the STRIDE variants analyzed in this paper. Guidelines, best practices, and shortcomings are discussed, yet there is no evidence about how the two approaches differ in terms of performance [20].

Scandariato et al. [18] have analyzed a previous version of STRIDE-per-element and evaluated the productivity, precision, and recall of the technique in an academic setting. The purpose of their descriptive study was to provide an evidence-based evaluation of the effectiveness of STRIDE. Our study, on the other hand, provides a comparative evaluation (by means of a controlled experiment) of the two latest approaches for STRIDE. Also, our study has a larger number of participants and uses a larger object. We remark that our study has some discrepancies with respect to the observed productivity (4.35 in our study vs. 1.8 threats per hour), precision (0.6 vs. 0.81), and recall (0.62 vs. 0.36). However, a direct comparison is not entirely possible, as the two studies use different versions of STRIDE-per-element (our being the most up-to-date).

A privacy oriented analysis methodology (LINDDUN [4]) has been evaluated with 3 descriptive studies [26]. LINDDUN is inspired by STRIDE and is complementary to it. Both techniques start from a representation of a system, which is described as a DFD. Similarly, the authors assess the productivity, precision (correctness) and recall (completeness) of the technique, as well as its usability.

Labunets et al [10] have performed an empirical comparison of two risk-oriented threat analysis techniques by means of a controlled experiment with students. The aim of the study was to compare the effectiveness and perception of a visual technique with a textual technique. The main findings were that the visual method is more effective for identifying threats than the textual one, while the textual method is slightly more effective for eliciting security requirements.

The work of Karpati, Sindre, Opdahl, and others provide experimental comparisons of several techniques. Opdahl et al. [14] measure the effectiveness, coverage and the perception of the techniques. Karpati et al. [8] present an experimental evaluation of MUC Map diagrams focusing on identification of not only vulnerabilities but also mitigations. Finally, Karpati et al. [9] have experimentally compared MUCs with mal-activity diagrams in terms of efficiency.

Diallo et al. [5] conducted a descriptive comparison of MUCs, attack trees, and Common Criteria [21]. The authors have applied these approaches to the same problem and discuss their observations about the individual technique's strengths and weaknesses. An interesting evaluation of the reusability of threat models (MUC stubs and MUC Maps diagrams, both coupled with attack trees) is presented by Meland et al. [13]. The authors conducted an experiment including seven professional software developers. The study suggests that overall, the productivity is improved by reusing threat models for both techniques.

8 Conclusion

This study has presented an empirical comparison of two variants of a popular threat analysis technique. The comparison has been performed in-vitro by

means of a controlled experiment with master students. As presented in Section 4, this work provides reproducible analysis and observations about the effectiveness of applying both techniques, in terms of productivity, precision and recall. In summary, with the type of population used in this study (non-experts), our study observed better results with the STRIDE-per-element variant. For instance, STRIDE-per-element yielded 1 additional threat per hour in terms of productivity, with no consequences on the average correctness of the results (i.e., same precision). The proponents of STRIDE have claimed that "STRIDE-per-interaction leads to the same number of threats as STRIDE-per-element" [20]. However, in this study, we have observed a statistically significant higher level of completeness in the results returned by the teams using STRIDE-per-element. This is possibly influenced by the tendency of the STRIDE-per-interaction group to create larger DFD models, which might not be necessarily needed. Another explanation is related to the more complex documentation in the case of the INTERACTION treatment. As security budgets are quite tight in companies, knowing that one variant might be more productive is a useful piece of information.

This work calls for future studies about the effectiveness of the threat analysis variants, especially with more expert analysts. In particular, we are planning a case study in two companies where the local, per-element analysis is compared to a global, end-to-end analysis. Furthermore, it would be beneficial to study the effect on the importance (in terms of risk) of the discovered threats, as well as the quality of the of security requirements that are derived from them.

References

1. Empirical study: Threat modeling. `https://sites.google.com/site/empiricalstudythreatanalysis/`, accessed: 2017-08-25
2. Abe, T., Hayashi, S., Saeki, M.: Modeling security threat patterns to derive negative scenarios. In: Software Engineering Conference (APSEC), 2013 20th Asia-Pacific. vol. 1, pp. 58–66. IEEE (2013)
3. Carver, J., Jaccheri, L., Morasca, S., Shull, F.: Issues in using students in empirical studies in software engineering education. In: Software Metrics Symposium, 2003. Proceedings. Ninth International. pp. 239–249. IEEE (2003)
4. Deng, M., Wuyts, K., Scandariato, R., Preneel, B., Joosen, W.: A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. Requirements Engineering 16(1), 3–32 (2011)
5. Diallo, M.H., Romero-Mariona, J., Sim, S.E., Alspaugh, T.A., Richardson, D.J.: A comparative evaluation of three approaches to specifying security requirements. In: 12th Working Conference on Requirements Engineering: Foundation for Software Quality, Luxembourg (2006)
6. Höst, M., Regnell, B., Wohlin, C.: Using students as subjectsa comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering 5(3), 201–214 (2000)
7. Howard, M., Lipner, S.: The security development lifecycle, vol. 8. Microsoft Press Redmond (2006)
8. Karpati, P., Opdahl, A.L., Sindre, G.: Experimental comparison of misuse case maps with misuse cases and system architecture diagrams for eliciting security

vulnerabilities and mitigations. In: Availability, Reliability and Security (ARES), 2011 Sixth International Conference on. pp. 507–514. IEEE (2011)
9. Karpati, P., Sindre, G., Matulevicius, R.: Comparing misuse case and mal-activity diagrams for modelling social engineering attacks. International Journal of Secure Software Engineering (IJSSE) 3(2), 54–73 (2012)
10. Labunets, K., Massacci, F., Paci, F., et al.: An experimental comparison of two risk-based security methods. In: Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on. pp. 163–172. IEEE (2013)
11. Lund, M.S., Solhaug, B., Stølen, K.: A guided tour of the coras method. In: Model-Driven Risk Analysis, pp. 23–43. Springer (2011)
12. McGraw, G., Migues, S., West, J.: Building security in maturity model (BSIMM). `https://www.bsimm.com`, accessed: 2017-08-25
13. Meland, P.H., Tøndel, I.A., Jensen, J.: Idea: Reusability of threat models-two approaches with an experimental evaluation. In: ESSoS. pp. 114–122. Springer (2010)
14. Opdahl, A.L., Sindre, G.: Experimental comparison of attack trees and misuse cases for security threat identification. Information and Software Technology 51(5), 916–932 (2009)
15. Runeson, P.: Using students as experiment subjects–an analysis on graduate and freshmen student data. In: Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering. pp. 95–102 (2003)
16. Saitta, P., Larcom, B., Eddington, M.: Trike v. 1 methodology document [draft] `http://dymaxion.org/trike/Trike\_v1\_Methodology\_Documentdraft.pdf`
17. Salman, I., Misirli, A.T., Juristo, N.: Are students representatives of professionals in software engineering experiments? In: Proceedings of the 37th International Conference on Software Engineering-Volume 1. pp. 666–676. IEEE Press (2015)
18. Scandariato, R., Wuyts, K., Joosen, W.: A descriptive study of microsofts threat modeling technique. Requirements Engineering 20(2), 163–180 (2015)
19. Schneier, B.: Attack trees. Dr Dobb's Journal, v.24, n.12 (1999)
20. Shostack, A.: Threat modeling: Designing for security. John Wiley & Sons (2014)
21. Stoneburner, G., Hayden, C., Feringa, A.: Engineering principles for information technology security (a baseline for achieving security). Tech. rep., BOOZ-ALLEN AND HAMILTON INC MCLEAN VA (2001)
22. Torr, P.: Demystifying the threat modeling process. IEEE Security & Privacy 3(5), 66–70 (2005)
23. Tuma, K., Scandariato, R., Widman, M., Sandberg, C.: Towards security threats that matter. In: Computer Security, pp. 47–62. Springer (2017)
24. UcedaVelez, T., Morana, M.M.: Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis. John Wiley & Sons (2015)
25. Wohlin, C., Höst, M., Henningsson, K.: Empirical research methods in software engineering. In: Empirical methods and studies in software engineering, pp. 7–23. Springer (2003)
26. Wuyts, K., Scandariato, R., Joosen, W.: Empirical evaluation of a privacy-focused threat modeling methodology. Journal of Systems and Software 96, 122–138 (2014)