Automatic Generation of Security Requirements for Cyber-Physical Systems

Jinghua Yu[1(✉)], Stefan Wagner[2], and Feng Luo[3]

[1] Tongji University, Caoan Highway 4800, Shanghai 201804, China
yujinghua@tongji.edu.cn
[2] University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
stefan.wagner@iste.uni-stuttgart.de
[3] Tongji University, Caoan Highway 4800, Shanghai 201804, China
luo_feng@tongji.edu.cn

Abstract. Security is one of the essential properties in Cyber-Physical Systems (CPS). Attacking systems like autonomous vehicles and healthcare systems may lead to financial or privacy losses of stakeholders or even life threats. Security analysis, as an early activity in the system design, addresses security issues and identifies system vulnerabilities in advance to guide further security design. However, the security analysis is mostly performed manually requiring a high workload with human oversight. Besides, the manual analysis is not flexible for modification in later design stages and largely depends on expert knowledge and experience. Therefore, a new security analysis approach has been proposed in this paper to generate security requirements automatically, which is based on the System-Theoretic Process Analysis (STPA) framework and is applicable for data-flow-based CPSs. We have also developed a software prototype to support the implementation of this automatic approach and used it to obtain the security requirements of two CPSs in the automotive domain. Finally, we compared the automatically generated outcomes with the manually obtained ones and evaluated the proposed approach. Based on the experiment results, we found that the automatic way is efficient, effective and flexible. Furthermore, the proposed approach is also extensible. Analysts in a team can establish their own empirical repository to achieve accurate security requirements for their specific systems.

Keywords: Security analysis · STPA framework · Pattern matching · Empirical repository

1 Introduction

Cyber-Physical Systems (CPS) are built from, and depend on, the seamless integration of computation and physical components, which consist of computing

Supported by the China Scholarship Council and funds of the German Federal Ministry of Education and Research under grant number 16KIS0995.

S. Paiva et al. (Eds.): SmartCity360° 2020, LNICST 372, pp. 372–385, 2021.
https://doi.org/10.1007/978-3-030-76063-2_26

devices, actuation, sensing and network infrastructure as well as possible human interactions [7]. CPSs, like vehicular systems, medical and health-care systems, industrial control systems and smart grid [11], are indispensable nowadays and transforming the way people interact with engineered systems [7]. However, due to the high complexity and connectivity of CPSs and increasing applications in our daily life, securing CPSs is becoming much more significant, especially for safety- or security-critical systems. Adversaries may attack an insecure system to control it maliciously or eavesdrop on sensitive information, which leads to financial or even life losses. Therefore, security should be kept in mind during the whole life cycle of systems, particularly in the early design stage. Early security considerations can lower costs downstream in systems' life cycle, identify vulnerabilities before being exploited and adopt a proactive, rather than reactive, approach to security [4].

Security analysis, as an early design activity, addresses potential security issues and achieves security requirements, which are defined as conditions over the phenomenon of the environment that should be ensured to mitigate risks [13]. Standards or frameworks, like the SEA J3061 guideline [5] and the EVITA [14] framework in the automotive domain, have been published to achieve systematic security design of a target. Techniques proposed in these frameworks are normally threat-oriented, which start with system decomposition and threats identification. To strengthen the consideration of interactions among system components, System-Theoretic Process Analysis (STPA) has been proposed as a hazard analysis approach originally, which views losses as components interactions [20]. An extension of STPA for security (STPA-Sec) was then proposed [19] for security design.

However, the security analysis is normally performed by human analysts, which has the following limitations. First, the analysis process requires a high manual workload, especially for complex systems with a large number of components, which makes it easy for a human to make mistakes. Second, it is not friendly for modification. Once an error is detected or modification is required in later analysis phases, it's boring and time-consuming to correct every signal related detail in all previous documents manually. Furthermore, the analysis outcomes largely depend on the knowledge and expert experience, which means that to achieve trusted and useful results, it's necessary for analysts to be familiar with both the target system and knowledge in the security fields, which is sometimes not possible in a team.

To overcome the limitations of the manual analysis, we propose an approach based on an extension of STPA-Sec for data-flow-based systems (STPA-DFSec) to achieve security requirements automatically. Then, we develop a prototype by using Excel Visual Basic for Application (VBA) to implement our concept. Finally, we conduct experimental analyses of two CPSs in the automotive domain to evaluate the automated approach.

The rest of this paper is organized as follows. In Sect. 2, we introduce existing approaches for the automatic generation of security requirements and the STPA framework with its automated extensions. In Sect. 3, the automatic generation

methodology is introduced, including the original STPA-DFSec steps, the automated generation process and a generator prototype. In Sect. 4, we present experimental analyses of two CPSs in the automotive domain, compare the experiment results and evaluate the proposed approach. Finally, we conclude the paper in Sect. 5.

2 Related Work

2.1 Existing Approaches for Automated Generation

We investigated the existing approaches for obtaining system security requirements automatically and classified them into the following classes.

The first class is permutations and combinations, which identifies variants in the requirement expressions, enumerates possible values and lists all combinations by the computer automatically. Thomas [16] established the system process model by determining context variants and their values to identify hazardous control actions for a system. Various context combinations are listed in a table but whether a concrete situation is hazardous for the system or not should be judged by human analysts. Gao and his colleagues [9] decomposed their spacecraft into subsystems and components with required variants and value ranges. By combining all possible pre-defined scenarios and objects with various parameters, a set of test cases is generated. This kind of approach can cover all possible cases efficiently and is also a common idea to generate test cases. However, the generated results may contain a number of meaningless combinations since the permutation and combination algorithm normally does not include a judgment process. Besides, if there are too many initial variants and values, the amount of the generated results may be too large to be processed manually.

The second class is recommendations, which uses machine learning to obtain recommended security requirements through training. Xu and his colleagues [18] proposed a co-occurrence recommendation model to get software security requirements. Relationships between security threats and requirements were established through training with the data extracted from security target documents. However, this approach is only for software products and may not be applicable to complex CPSs. Threats have to be extracted manually from security target documents but such documents may not even exist at the early design stages. Besides, it's also hard to ensure that the training data is proper and enough to make the recommendation system effective and trusted.

The third class is formal approaches, which use formal models to express the system and analyze it formally. Graa and his colleagues [10] modeled a system using a goal-oriented approach named Knowledge Acquisition in autOmated Specification (KAOS) and generated security policies with externally input security information. But the inputs including risk analysis results and security requirement specifications have to be obtained manually by analysts. Emeka and Liu [6] proposed a new framework based on the Structured Object-Oriented Formal Language (SOFL) to define security requirements. Yet, the authors didn't discuss the possibility of processing it automatically. Besides, the state machine

model is commonly used to model a system and performs model check based on branching-time logic (e.g. Linear Temporal Logic (LTL) and Computation Tree Logic (CTL)) [2,3,17]. However, that research is about generating test cases and verifying systems according to predefined requirements. No study of deriving requirements by the state machine model has been found.

2.2 STPA Framework and Automated Extensions

System-Theoretic Process Analysis (STPA) is a hazard analysis approach based on the System-Theoretic Accident Model and Processes (STAMP) model, which treats system safety as a dynamic control problem rather than a failure prevention problem. Many evaluations and comparisons of STPA to traditional analysis methods have been done and show that STPA identified more, often software-related and non-failure, scenarios and it costs much less time and resources than the traditional processes [12]. Extensions have been proposed to conduct better analysis with the basic STPA idea, including the STPA for security (STPA-Sec) [19], the STPA for privacy (STPA-Priv) [15] and the STPA for co-analysis of both safety and security (STPA-SafeSec) [8]. The steps of the basic STPA is shown in Fig. 1.

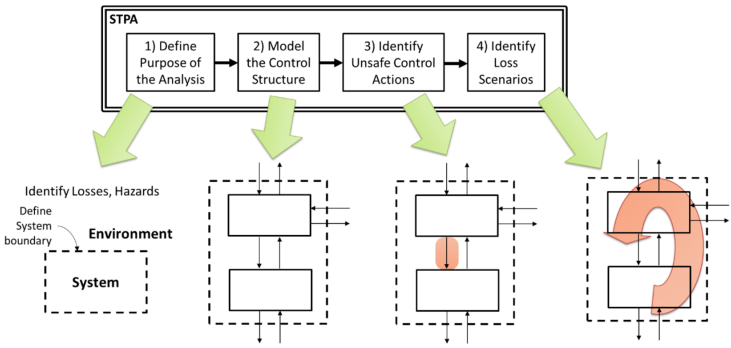

Fig. 1. Overview of the basic STPA steps [12]

To extend and automate the STPA for the requirement generation, Thomas [16] defined a formal structure underlying STPA and presented a method for automating both the STPA analysis and the requirements generation with a conflict detection feature. Abdulkhaleq [1] proposed an STPA-based safety engineering approach for software-intensive systems called STPA SwISs to conduct seamless safety analysis and software verification activities. A platform called XSTAMPP was developed to support identifying requirements, verifying practical implementation against SPTA-generated requirements by model checking tools and generating safety-based test cases automatically. These researches show the possibility of automating the STPA-based approaches to increase the efficiency and reliability of the analysis process.

In our previous research [21,22], we identified some limitations of the STPA-Sec, which is not effective for information-critical systems, and proposed a data-flow based extension of STPA-Sec (STPA-DFSec). Two case studies have been conducted manually to verify the effectiveness of the proposed approach comparing with other approaches. Nevertheless, we found that the manual processes required a high workload. Later modifications in the analysis documents were always boring and time-consuming. These manual analysis drawbacks motivate our research on the automatic way to achieve security requirements.

3 Methodology

3.1 Brief Introduction of STPA-DFSec

The data-flow-based STPA-Sec (STPA-DFSec) [21] follows the basic steps of the STPA framework and introduces some data-flow-related adjustments into it. The overview of STPA-DFSec steps is shown in Fig. 2.

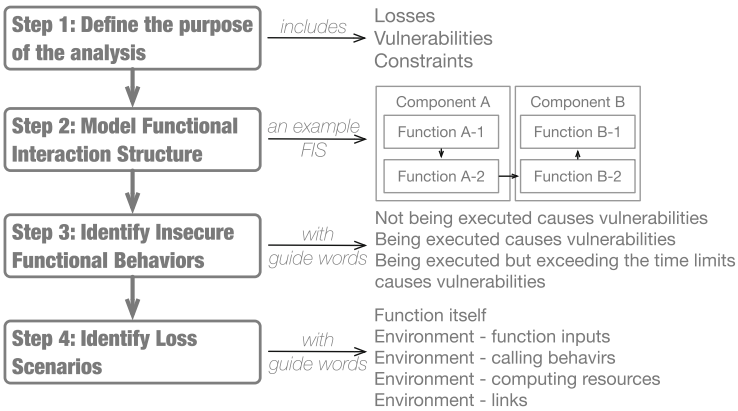

Fig. 2. Overview of the STPA-DFSec steps

First, the purpose of the analysis should be defined at a high level, including identifying system-level losses, vulnerabilities and constraints. Second, the Functional Interaction Structure (FIS) of the target system is modeled to interpret how the system works from the perspective of data flows. Functions, as the basic element of the FIS, are identified and linked with each other by data flows. Third, Insecure Function Behaviors (IFB), which lead to system vulnerabilities in a particular context, are identified with the help of guide words. Finally, Loss Scenarios (LS), describing the causal factors that lead to insecure function behaviors, are identified. Each loss scenario can be translated into system security requirements by simply inversing the conditions or defining what the system must do in case the incident occurs [12].

3.2 Automated Generation Process

The automated generation process complies with the STPA-DFSec and aims to achieve IFBs and LSs effectively and efficiently. Figure 3 is the overall workflow of the proposed generator. The automated process starts after the second step of STPA-DFSec, in which the system FIS has been created and functions have been defined. The human analysts need to fill in a questionnaire to provide information for the generator. Table 1 is the questionnaire template. Nine questions in the questionnaire are abstracted from the analysis results that we identified manually in the previous research and are closely related to the automated judgment.

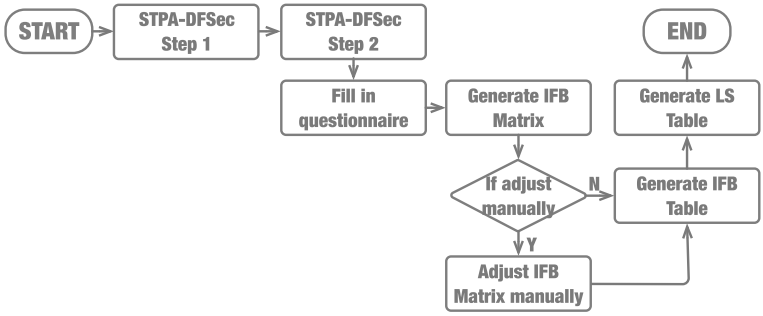

Fig. 3. Flowchart of the automatic generator

Table 1. Questionnaire template

Questions	Function1	
1. Is it related to confidentiality?	Y/N	...
2. Is it related to integrity?	Y/N	...
3. Is it related to availability?	Y/N	...
4. Is it possible to bypass the function?	Y/N	...
5. Does it return an execution result flag?	Y/N	...
6. Is its input from another components via physical links?	Y/N	...
7. Is its output to another components via physical links?	Y/N	...
8. Is it controlled by software algorithms?	Y/N	...
9. Is the supporting data (e.g. cryptography keys, system states, feedbacks on UIs) required for the execution?	Y/N	

Then, the IFB matrix is generated automatically. The flowchart of generating the IFB Matrix is shown in Fig. 4 (a). The generator checks the type conditions of each function and marks a 'Y' (Yes) or 'N' (No) in the matrix to indicate if a

function contains a certain type of IFBs. The IFB types and their conditions are listed in Table 2. Whether a condition is matched or not is determined by the answer of a corresponding question in the questionnaire. The final IFBs and LSs will be generated on the basis of this IFB matrix, which can also be manually adjusted if necessary.

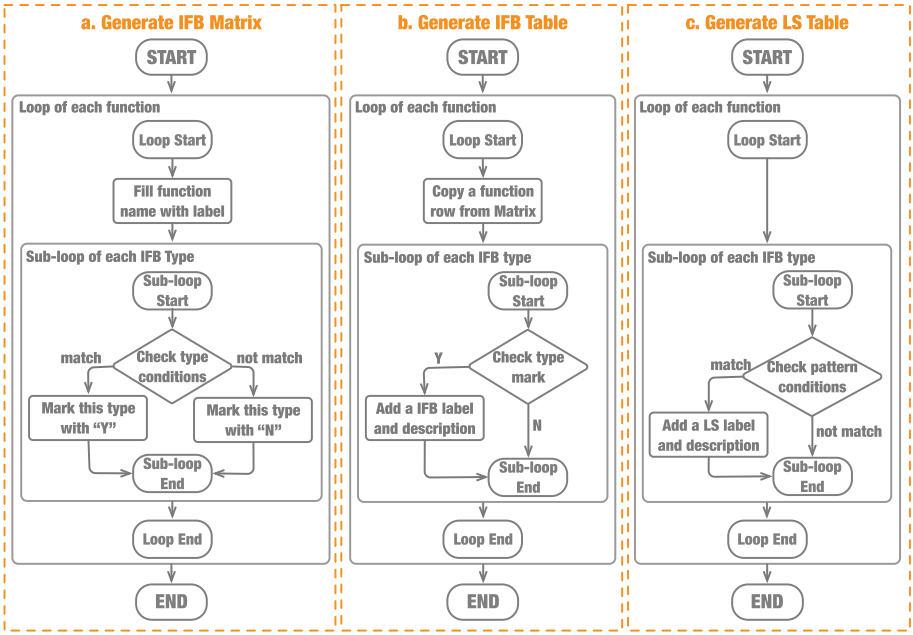

Fig. 4. Flowcharts of automatic generation

Table 2. IFB types and conditions

IFB types	Conditions (related questions)
Type1: Not being executed	It can be bypassed intendedly (4)
Type2: Being executed with information leakage risks	It is related to confidentiality (1)
Type3: Being executed with tempered data or algorithms	It is related to integrity (2)
Type4: Being executed but violating system specifications (e.g. timing limits)	It is related to availability (3)

Finally, IFBs and LSs are generated according to the workflow in Fig. 4 (b) and (c) respectively. To obtain an IFB table, the generator checks each type mark in the IFB matrix and add a concrete IFB label and description if the

mark is 'Y'. To obtain LSs, the generator checks pattern conditions of identified IFBs and add labels with the corresponding descriptions if the conditions are matched. The LS patterns and their conditions are shown in Table 3.

Table 3. LS patterns and conditions

IFB types	LS patterns	Conditions (related questions)
Type1	Function is bypassed intendedly, which can be detected by user	Matched by default
	Function is bypassed intendedly and outputs fake results	Function should return a flag value (5)
Type2	No or inadequate mechanism is used to protect data confidentiality	Matched by default
	Physical input links are not protected, adversary eavesdrops data on links	There should be an input channel via physical links to function (6)
	Physical output links are not protected, adversary eavesdrops data on links	There should be an output channel via physical links from function (7)
	Algorithm is tampered maliciously to eavesdrop data during function execution	Function should by controlled by software algorithms (8)
	Supporting data is tampered, leading to execution with risks of information leakage	Some kinds of supporting data are required for function execution (9)
Type3	No or inadequate mechanism is used to protect data integrity	Matched by default
	Physical input links are not protected, adversary tampers data on links	There should be an input channel via physical links to function (6)
	Physical output links are not protected, adversary tampers data on links	There should be an output channel via physical links from function (7)
	Algorithm is tampered, which leads to unexpected behaviors of function	Function should by controlled by software algorithms (8)
Type4	Data is blocked on links or routed to an unexpected destination	There should be an output channel via physical links from function (7)
	Transmission is slowed down by additional mechanisms on links	There should be an output channel via physical links from function (7)
	Algorithm is tampered, which leads to unexpected behaviors (e.g. reject legal request)	Function should by controlled by software algorithms (8)
	Algorithm is tampered, which needs more computing resources	Function should by controlled by software algorithms (8)
	Computing resource is occupied to cause violation of execution timing limitations	Function should by controlled by software algorithms (8)
	Supporting data is tampered, which leads function being executed wrongly	Some kinds of supporting data are required for function execution (9)

3.3 Generator Prototype

To implement and verify the proposed approach in practice, a quick prototype has been developed using Excel VBA. After filling the questionnaire in the first sheet, an IFB matrix (in Fig. 5) is generated. Then, by clicking the 'Generate IFBs' and 'Generate LSs' buttons in the IFB matrix sheet (Fig. 5), the final IFB and LS tables are obtained in a second (shown in Fig. 6).

	A	B	C	D	E
1		STPA-DFSec Generator			
2		IFB Matrix Generate IFBs Generate LSs			Clear IFB Matrix
3	Function	Not being executed	Being executed with information leakage risks	Being executed with tempered data or algorithms	Being executed but violating system specifications
4	V/E_F1: data check	Y	Y	Y	Y
5	V_F2: data transforming	N	Y	Y	Y
6	E_F3: req/resp en/decaps	N	Y	Y	Y
7	L_F4: data transmission	N	Y	Y	Y
8	E_F5: service process	Y	Y	Y	Y

Fig. 5. Generated IFB matrix

	A	B	C	D
1	STPA-DFSec Generator			
2	LS Table		Clear LS Table	
3	IFB Label	LS Label	LS Description	
4	V/E_F1_IFB1	V/E_F1_IFB1_LS1	The function 'data check' is bypassed by the adversary, which can be detected by the user.	
5	V/E_F1_IFB1	V/E_F1_IFB1_LS2	The function 'data check' is bypassed by the adversary and fake an output.	
6	V/E_F1_IFB2	V/E_F1_IFB2_LS1	No or inadequate mechanism related to the function 'data check'is used to protect data confidential	
7	V/E_F1_IFB2	V/E_F1_IFB2_LS2	The	
8	V/E_F1_IFB2	V/E_F1_IFB2_LS3	The	
9	V/E_F1_IFB3	V/E_F1_IFB3_LS1	No	
10	V/E_F1_IFB3	V/E_F1_IFB3_LS2	The	
11	V/E_F1_IFB3	V/E_F1_IFB3_LS3	The	
12	V/E_F1_IFB4	V/E_F1_IFB4_LS1	The	
13	V/E_F1_IFB4	V/E_F1_IFB4_LS2	The	
14	V/E_F1_IFB4	V/E_F1_IFB4_LS3	Com	
15	V_F2_IFB1	V_F2_IFB1_LS1	No	
16	V_F2_IFB1	V_F2_IFB1_LS2	The	

	A	B	C	D	E
1		STPA-DFSec Generator			
2		IFB Table			Clear IFB Table
3	Function	Not being executed	Being executed with information leakage	Being executed with tempered data or algorithms	Being executed but violating system specifications
4	V/E_F1: data check	V/E_F1_IFB1	V/E_F1_IFB2	V/E_F1_IFB3	V/E_F1_IFB4
5	V_F2: data transforming	NA	V_F2_IFB1	V_F2_IFB2	V_F2_IFB3
6	E_F3: req/resp en/decaps	NA	E_F3_IFB1	E_F3_IFB2	E_F3_IFB3
7	L_F4: data transmission	NA	L_F4_IFB1	L_F4_IFB2	L_F4_IFB3
8	E_F5: service process	E_F5_IFB1	E_F5_IFB2	E_F5_IFB3	E_F5_IFB4
9					
12	IFB Description				
13	V/E_F1_IFB1	The function 'data check' is not executed.			
14	V/E_F1_IFB2	The function 'data check' is executed with information leakage risks.			
15	V/E_F1_IFB3	The function 'data check' is executed with tempered data or algorithms.			

Fig. 6. Screenshots of the generated IFB table and LS table

3.4 Approach Conclusion

The main idea of this automated generation approach is to enumerate all possible IFBs and LS patterns at a general level and decide whether an enumerated item is matched to a particular function. The judgment is based on the input

information obtained from the questionnaire. Other than the three classes introduced in the '2.1 Existing Approaches for Automated Generation' section, the proposed approach can be classified into a fourth 'pattern matching' class.

The complete sets of IFB types, LS patterns and corresponding conditions are established based on the original STPA theories and empirical conclusions in practices. These sets are regarded as empirical study repository, in which previously extracted IFB types and LS patterns are stored. This empirical repository is an open framework, into which newly recognized patterns can be added. Companies or specific teams can establish their own type and pattern repository to better support their specific target systems.

4 Experiments and Evaluation

4.1 Example Cases

To verify and evaluate the proposed approach, experiments have been conducted based on two example cases in the automotive field. In this sub-section, both cases are introduced briefly. More details about the example systems and the manual analysis processes are elaborated in the previous publications [21,22].

The first system (notated as Sys1) is an in-vehicle diagnostic and software update system, which transmits data via in-vehicle networks, like Controller Area Network (CAN) and Automotive Ethernet (AE), to achieve the diagnostics and software or configuration updates of electronic devices in vehicles. This system consists of a vehicle interface, in-vehicle networks and an end device (shown in Fig. 7 (a)). Due to the increasing interactions between the in-vehicle systems and the outside entities (e.g. handheld devices, manufacturer's cloud), such a system is security-critical and needs to be protected against hazards including eavesdropping on the device firmware or customer data and injecting malicious software. By using STPA-DFSec, 17 IFBs and 23 LSs were identified manually.

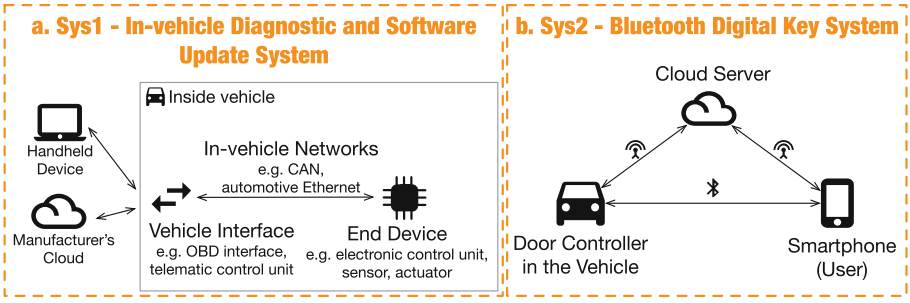

Fig. 7. System sketch of two example systems

The second system (notated as Sys2) is a Bluetooth digital key system of the vehicle consisting of three main physical components (shown in Fig. 7 (b)).

The user uses a smartphone to lock or unlock the vehicle doors via Bluetooth. A cloud server is required to support cryptographic mechanisms for secure communications between components. Such a digital key system should be protected against malicious actions including unlocking vehicle doors illegally and Denial-of-Service (DoS) attacks. 22 IFBs and 41 LSs were identified manually by STPA-DFSec.

4.2 Experiments and Comparison

We followed the workflow of the automated generator (Fig. 3) to obtain the IFBs and LSs of both Sys1 and Sys2 automatically. The time duration and the outcomes of the automated process are recorded and compared with the manual ones which we recorded in previous studies (see 'Sect. 4.1 Example Cases').

Figure 8 shows the time consumption of the analysis processes. For the automated process, the analysts took approximately 1.7 min per function to fill in the questionnaire, review and adjust the IFB Matrix when necessary. Then, IFBs and LSs were generated instantly by two clicks. By contrast, it took at least 50 mins to identify and write down all labels and descriptions of IFBs and LSs manually.

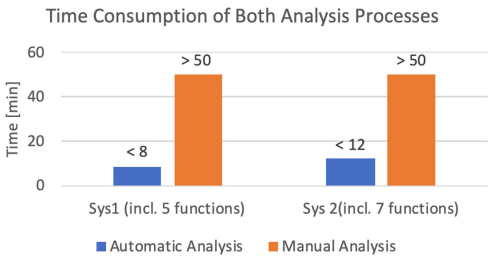

Fig. 8. Time consumption of both analysis processes

Outcomes have also been compared and shown in Table 4. We mapped each IFB achieved by both processes and found that one automatically obtained IFB may be related to several manually obtained IFBs. Besides, the understanding error also impacts the number of outcomes. The understanding of a system by a person is subjective. For example, expert A thinks function1 is related to confidentiality while expert B doesn't think so. Even the same person may have different ideas at different times. Therefore, to make the numbers comparable, the outcome differences caused by the understanding errors are removed. The comparison conclusion is listed in the final column of Table 4, which shows that the automated generation can achieve almost the same numbers of IFBs and obviously more LSs than those in the manual analysis. The reason may be that the number of IFBs is relatively small and can be handled well by the human mind, while it would be easy for human analysts to miss something when the number of items increases.

Table 4. Outcome comparison of both analysis processes

Item type	System	Number in automatic	Number in manual	Comparison conclusion (considering understanding errors)
IFB	Sys1	17	17	The number is the same
	Sys2	24	22	1 new IFB in the automatic one
LS	Sys1	42	23	5 new LSs in the automatic one
	Sys2	60	41	20 new LSs in the automatic one

Furthermore, the expressions generated automatically are more abstract and at higher levels, while the manually identified ones are sometimes more concrete with the detailed context in a specific case.

4.3 Evaluation and Discussion

Finally, the automated generation approach was evaluated based on three metrics, which are efficiency, effectiveness and flexibility.

As for the efficiency, the automated analysis requires much less time than the manual one, especially for complex systems with a large number of functions. The workload of human analysts is reduced significantly by the automated process.

As for the effectiveness, the generated IFBs and LSs cover all manually identified outcomes. Since the generation conditions are designed based on the empirical repository, which stores previous experience and knowledge collected in practice, the generated results are reliable. Note that the outcomes of security analysis, no matter they are identified by the proposed approach automatically or manually, are not able to be proven complete. In practice, analysis outcomes largely depend on the analysis emphasis, available system information and the knowledge and experience of the analysts who build the repository or perform the manual analysis. In our experiments, the same authors did the manual analyses and designed the generator with its repository, which makes the obtained IFBs and LSs comparable in our cases. Besides, the low workload also reduces the probability of making mistakes by a human. Complex systems and long working duration may cause mistakes and omitted points, while the automated process does not have such problems.

As for the flexibility, it's much easier to perform an iterated analysis or modify something at any design stage. The analysts can update the questionnaire and IFB matrix within at most 2 mins per function and then regenerate everything instead of modifying every related label and description by hand. Furthermore, the automated process is also a good way to deal with understanding errors, which are inevitable due to the knowledge and experience of the analysts, the description in system documents and personal understanding ability. Once an understanding error is identified even in the later design stage, the analysts only need to revise the questionnaire and get updated outcomes immediately.

Two limitations of the proposed generator have been identified. First, since the generation is based on the empirical repository, the quality of such a repository and the abstracted patterns is important for obtaining reliable outcomes.

It requires experienced experts and many practical case studies to build a high-quality repository, which will spend much time and effort for a new type of system at the beginning. However, once the corresponding repository is established, the work efficiency can be increased significantly. Second, the generated IFBs and LSs are described at a general level, which can not explain specific situations related to a particular system. Therefore, such general descriptions required further refinement by human analysts to support concrete design work later. However, the generated outcomes can be used as the outputs of a high-level analysis at early design stages, preliminary requirements of the system or the elicitation hints for more specific requirements with concrete details in the later design stage.

5 Conclusion

In this paper, we proposed an approach based on the STPA-DFSec to achieve system security requirements automatically and provided a prototype tool to support the implementation. We conducted the experiments of two CPSs in the automotive domain and compared the outcomes of both manual and automatic analyses. Finally, we evaluated the proposed approach from the perspective of efficiency, effectiveness and flexibility.

Comparing with the manual process, the automatic analysis requires less time and workload, obtains reliable outcomes based on previous empirical cases and is friendly for human analysts to perform iterations or modify analysis details even in the later design stages. Besides, the proposed approach is more reliable because the automated way reduces both the probability of human mistakes and the dependency on the knowledge and experience of analysts. Furthermore, this approach is an open framework for automatic generation. New patterns and conditions can be added to the repository to support specific and accurate requirement generation.

In the future, we will analyze more CPSs in practice to improve the pattern repository for a better generation of security requirements in various fields. Besides, a software tool will be designed based on the proposed prototype to support practical implementation in industries with better user experience features.

References

1. Abdulkhaleq, A.: A system-theoretic safety engineering approach for software-intensive systems. Ph.D. thesis (2017)
2. Abdulkhaleq, A., Wagner, S.: A systematic and semi-automatic safety-based test case generation approach based on systems-theoretic process analysis. arXiv preprint arXiv:1612.03103 (2016)
3. Aouadi, M.H.E., Toumi, K., Cavalli, A.: A formal approach to automatic testing of security policies specified in XACML. In: Cuppens, F., Garcia-Alfaro, J., Zincir Heywood, N., Fong, P.W.L. (eds.) FPS 2014. LNCS, vol. 8930, pp. 367–374. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17040-4_25

4. Carter, B.T., Bakirtzis, G., Elks, C.R., Fleming, C.H.: Systems-theoretic security requirements modeling for cyber-physical systems. Syst. Eng. **22**(5), 411–421 (2019)
5. SAE International: SAE J3061 - Cybersecurity Guidebook for Cyber-Physical Automotive Systems (2016)
6. Emeka, B.O., Liu, S.: Security requirement engineering using structured object-oriented formal language for m-banking applications. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 176–183. IEEE (2017)
7. US National Science Foundation: Cyber-physical systems program solicitation (nsf 20–563) (2020). https://www.nsf.gov/pubs/2020/nsf20563/nsf20563.htm
8. Friedberg, I., McLaughlin, K., Smith, P., Laverty, D., Sezer, S.: STPA-SafeSec: safety and security analysis for cyber-physical systems. J. Inf. Secur. Appl. **34**, 183–196 (2017)
9. Gao, S., Lyu, J., Wuniri, Q., Meng, X., Ma, S.: Spacecraft test requirement description and automatic generation method. J. Beijing Univ. Aeronaut. Astronaut. **41**(7), 1275–1286 (2015)
10. Graa, M., et al.: Using requirements engineering in an automatic security policy derivation process. In: Garcia-Alfaro, J., Navarro-Arribas, G., Cuppens-Boulahia, N., de Capitani di Vimercati, S. (eds.) DPM/SETOP -2011. LNCS, vol. 7122, pp. 155–172. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28879-1_11
11. Khaitan, S.K., McCalley, J.D.: Design techniques and applications of cyberphysical systems: a survey. IEEE Syst. J. **9**(2), 350–365 (2014)
12. Leveson, N.G., Thomas, J.P.: STPA Handbook (2018). https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf
13. Matulevičius, R.: Fundamentals of Secure System Modelling. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61717-6
14. Ruddle, A., et al.: Deliverable D2.3: security requirements for automotive on-board networks based on dark-side scenarios. Technical report, EVITA (2009)
15. Shapiro, S.S.: Privacy risk analysis based on system control structures: adapting system-theoretic process analysis for privacy engineering. In: 2016 IEEE Security and Privacy Workshops (SPW), pp. 17–24. IEEE (2016)
16. Thomas, J.P.: Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis. Ph.D. thesis, Massachusetts Institute of Technology (2013)
17. Wardell, D.C., Mills, R.F., Peterson, G.L., Oxley, M.N.: A method for revealing and addressing security vulnerabilities in cyber-physical systems by modeling malicious agent interactions with formal verification. Procedia Comput. Sci. **95**, 24–31 (2016)
18. Xu, Y., Ge, W., Li, X., Feng, Z., Xie, X., Bai, Y.: A co-occurrence recommendation model of software security requirement. In: 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE), pp. 41–48. IEEE (2019)
19. Young, W., Leveson, N.G.: Systems thinking for safety and security. In: Proceedings of the 29th Annual Computer Security Applications Conference, pp. 1–8 (2013)
20. Young, W., Leveson, N.G.: Inside risks-an integrated approach to safety and security based on system theory: Applying a more powerful new safety methodology to security risks. Commun. ACM **57**(2), 232–242 (2014)
21. Yu, J., Wagner, S., Luo, F.: Data-flow-based adaption of the System-Theoretic Process Analysis for Security (STPA-Sec). PeerJ Comput. Sci. **7**, e362 (2021)
22. Yu, J., Wagner, S., Luo, F.: An STPA-based approach for systematic security analysis of in-vehicle diagnostic and software update systems. In: FISITA Web Congress 2020, F2020-VES-020 (2020)