Security Attack Analysis Using Attack Patterns

Tong Li, Elda Paja, John Mylopoulos
University of Trento, Italy
{tong.li, paja, jm}@disi.unitn.it

Jennifer Horkoff
City University London, UK
horkoff@city.ac.uk

Kristian Beckers
Technische Universität München, Germany
beckersk@in.tum.de

***Abstract*—Discovering potential attacks on a system is an essential step in engineering secure systems, as the identified attacks will determine essential security requirements. The prevalence of Socio-Technical Systems (STSs) makes attack analysis particularly challenging. These systems are composed of people and organizations, their software systems, as well as physical infrastructures. As such, a thorough attack analysis needs to consider strategic (social and organizational) aspects of the involved people and organizations, as well as technical aspects affecting software systems and the physical infrastructure, requiring a large amount of security knowledge which is difficult to acquire. In this paper, we propose a systematic approach to efficiently leverage a comprehensive attack knowledge repository (CAPEC) in order to identify realistic and detailed attack behaviors, avoiding severe repercussions of security breaches. In particular, we propose a systematic method to model CAPEC attack patterns, which has been applied to 102 patterns, in order to semi-automatically select and apply such patterns. Using the CAPEC patterns as part of a systematic and tool-supported process, we can efficiently operationalize attack strategies and identify realistic alternative attacks on an STS. We validate our proposal by performing a case study on a smart grid scenario.**

***Index Terms*—Attack Analysis, Attack Pattern, Contextual Goal Model, Prototype, Validation**

I. Introduction

Security breaches in Socio-Technical Systems (STSs) have repeatedly resulted in multi-million dollar losses per year to large organizations, and this cost is on the rise [1]. A primary reason for these breaches is the complexity and heterogeneity of STSs, consisting of people, processes, technology and infrastructures. All of these heterogeneous components raise a plethora of security concerns and present a larger attack surface compared to more homogeneous software systems. As reported by Ponemon Institute [1], in 2015, the average total cost of a data breach for the 350 companies participating in the study increased from 3.52 to 3.79 million dollars; and these breaches are caused by a broad range of attacks, including trusted insiders (inadvertent or malicious), malware, SQL injections, hijacked devices, etc. In addition, given the increased number of attacks, STSs have become the ideal target of multistage attacks, as attackers can compose atomic attack actions associated with different components to perform more dangerous attacks [2]. Failing to consider such diverse attacks while designing STSs can result in vulnerable systems.

Holistically discovering potential attacks is an essential step for engineering secure STSs, as the identified attacks will determine essential security requirements and shed light on what and why security mechanisms are required. Therefore, we have outlined a holistic attack analysis framework in our previous work [3], [4], which proposed to holistically explore the alternative attacks from an attacker's viewpoint, as shown in Fig. 1. The framework takes as input a three-layer requirements model, developed as part of our previous security requirements analysis work [5], which contains both system functional requirements and security requirements (i.e, the right part of Fig. 1). The three-layer model consists of a social layer, a software layer and a physical layer, capturing holistic system context, and is used as the domain model during the attack analysis. The output of the framework will be a set of alternative (multistage) attacks, which are transferred back to the three-layer requirements model in order to identify critical security requirements [5]. For the holistic attack analysis framework, in our previous work [6], we have first investigated how to identify an attacker's strategies by systematically elaborating an attacker's malicious intentions (i.e., the top-left part of Fig. 1). Each attack strategy, consists of one or several anti-goals that describe attackers' malicious intentions, sheds light on what and when attackers may intend to attack, but does not answer how attackers can achieve their malicious intentions.

For this paper, we aim at analyzing how attackers implement an identified attack strategy in terms of realistic attack behaviors (i.e., the bottom-left part of Fig. 1), completing our holistic attack analysis. Without this step, the analysis would stop at abstract attack objectives, which may or may not be realistically obtainable through attack mechanisms. Determining which attack strategies are likely to be operationalized is critical for targeted security analysis. However, a lack of knowledge about impending attacks introduces a primary challenge to such operationalization analysis, as analysts cannot *realistically* identify how attackers might attack a system and thus have either false positives or false negatives during security analysis. As the security community has summarized practical attack knowledge in terms of 504 attack patterns[1], i.e., CAPEC (Common Attack Pattern Enumeration and Classification), we intend to solve the knowledge problem with the CAPEC attack patterns. However, as reported by Shostack "the impressive size and scope of CAPEC may make it intimidating for people to jump in" [7], and thus analysts are reluctant to use such patterns in practice. Although a number of studies have been performed to deal with the scalability problem of CAPEC [8], [9], [10], they all require manual

[1] https://capec.mitre.org

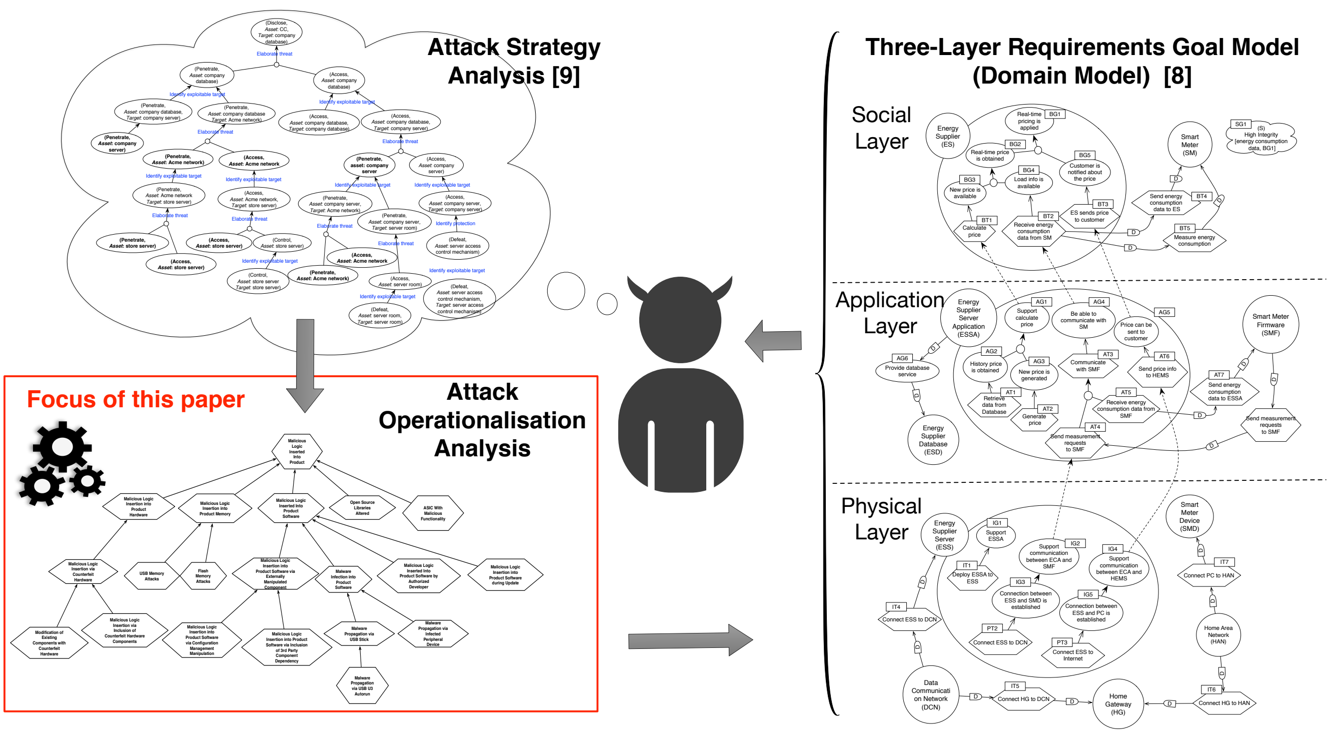

Attack Strategy Analysis [9]
Three-Layer Requirements Goal Model (Domain Model) [8]
Social Layer
Application Layer
Physical Layer
Focus of this paper
Attack Operationalisation Analysis

Fig. 1: An overview of the holistic attack analysis framework

assessment of the applicability of attack patterns, and cannot efficiently incorporate CAPEC patterns into attack analysis.

In this paper, we propose a systematic and efficient approach to leverage CAPEC attack patterns to operationalize attack strategies in terms of realistic and detailed attack behaviors, completing our holistic attack analysis framework (as illustrated in Fig. 1). The specific contributions of this paper include:

1) Proposing a systematic method to model CAPEC attack patterns as contextual goal models in order to semi-automatically select and apply attack patterns based on system context. The method is line with the one we have applied to security patterns [11], but has been adjusted to accommodate specific concepts in attack patterns. We have pragmatically applied this method to model 102 CAPEC attack patterns, which can be reused for operationalizing attack strategies.
2) Defining a systematic process and a collection of formal inference rules to efficiently select attack patterns for operationalizing attack strategies in terms of practical attacks.
3) Implementing a prototype tool in order to semi-automate the operationalization analysis and to facilitate the practical adoption of our approach.
4) Validating the entire holistic attack analysis approach, both the attack strategy identification [6] and the attack strategy operationalization, using a smart grid case study.

In the remaining part of this paper, we first describe the baseline approaches we use for operationalizing attack strategies in Section II. After that we detail the attack strategy operationalization in Section III and Section IV, where we present our method for modeling CAPEC attack patterns and a systematic operationalization process. The supporting tool for this approach is introduced in Section V, with the help of the tool we evaluate our proposal using a smart grid case study in Section VI. We compare our approach with existing work, and discuss the advantages of our proposal in Section VII. Finally, we conclude the paper and discuss future work in Section VIII.

II. Baseline

A. *A Three-Layer Security Requirements Analysis Framework*

We have proposed a three-layer security requirements analysis framework, in which we use an extended goal modeling language to separately model and analyze the requirements of STSs in three layers, namely, a business layer, an application layer, and a physical layer, while capturing the connections among layers [5]. In that work, we iteratively analyze security requirements for each layer in order to eventually produce holistic security solutions for the entire system. During the security requirements analysis in each layer, related threat information is required to in order to identify critical security requirements. Such threat analysis was not accommodated by the framework, instead we proposed to incorporate external threat analysis approaches for this purpose (e.g., [12]) or to import corresponding threat information from related studies that have been performed in the same domain (e.g., [18]). However, existing threat analysis approaches cannot holistically analyze multistage attacks on STSs. As a result, when analyzing a particular domain that has not been investigated by other studies, the three-layer security requirements framework might not be able to precisely identify all critical security goals due to the missing of holistic multistage attacks.

The holistic attack analysis framework, which we have preliminarily outlined in [3], [4], fills the vacancy of holistic multistage attack analysis for the three-layer security requirements framework. As shown in Fig. 1, a three-layer requirements goal model, which we assume has already been built and used in the holistic security requirements analysis based on the methods in [5], is taken as an input of the holistic attack analysis framework. This model can provide information about the system under attack and help to identify realistic attacks on the system across all three layers. Eventually, the identified attacks will be transferred back to the three-layer security requirements framework, helping us to analyze critical security requirements and generate security solutions.

B. Attack Strategy Identification

As the first analysis step of our holistic attack framework, we identify the alternative attack strategies that attackers can adopt to damage systems. This has been investigated in our recent work [6], as shown in the top left of Fig. 1. In particular, we model an attacker's high-level malicious intentions as structured *anti-goals*, which can be refined and operationalized like typical *goals* but from an attacker's viewpoint. We argue that the attacker has specific ways of elaborating their high-level anti-goals until reaching clear and specific anti-goals, and thus the elaboration of anti-goals amounts to the generation of their attack strategies. In order to systematically investigate the elaboration of malicious intention, we propose to characterize an anti-goal as a quadruple:

- *Asset:* anything of value to stakeholders.
- *Threat:* an undesired condition imposed on an asset, which is specified in terms of STRIDE threat categories [7]. Note that STRIDE is derived from an acronym for the following six threat categories: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.
- *Target:* a system component which involves assets and is vulnerable to attacks.
- *Interval:* a time period during which attackers carry out attacks.

Using the structured anti-goal, we systematically examined three real attacks to STSs [2, Chap.11] and investigated how attackers elaborate their malicious intentions and produce attack strategies. Specifically, we identified five anti-goal refinement patterns, as well as a systematic process for using such patterns to generate attack strategies. In other words, our previous study yielded the meta-strategy for generating attack strategies from an attacker's viewpoint, assisting us in discovering potential attack strategies. Details of the attack strategy analysis approach can be found in [6]. In this paper, we take the identified attack strategies as input and operationalize them in terms of realistic attack behaviors.

C. CAPEC Attack Patterns

Attack patterns document reusable attack knowledge to bridge the knowledge gap and assist with attack analysis. CAPEC, as a comprehensive and well-documented attack knowledge repository, has been incrementally built, starting in from 2007, and includes 504 attack patterns thus far. In particular, CAPEC involves a wide range of attack categories, from social engineering, to software attacks, to physical attacks, perfectly meeting our need for holistic attack analysis. However, such patterns are specified in textual form, which is difficult to automatically analyze.

In this paper, we leverage CAPEC attack patterns to realistically operationalize potential attacks from an attacker's viewpoint. In particular, given an attacker's malicious intentions (i.e., anti-goals), our approach will tell whether and how attackers can achieve such malicious intentions with respect to the comprehensive attack knowledge provided by CAPEC attack patterns. To efficiently incorporate such a large amount of knowledge into our analysis, we propose a systematic method of modeling and analyzing CAPEC attack patterns, enabling us to semi-automatically select and apply these patterns.

D. Contextual Goal Models

Goal modeling languages have been extended with the concept of *context* in order to accommodate context-sensitive analysis, such as the proposal by Ali et al. [13]. In particular, such approaches associate context labels with specific goal model elements. Thus, a goal model element is taken into account during corresponding goal model analysis if and only if the associated context holds. In this paper, we propose to model attack patterns in terms of such contextual goal models in order to efficiently select applicable attack patterns according to specific anti-goals and analyze the alternative attack behaviors. We follow the approach in [13] to construct contextual goal models, and propose a semi-automatic process to check context and identify applicable attack patterns.

III. Modeling Attack Patterns

A pattern consists of three primary pattern concepts: *Context*, *Problem*, and *Solution* [14], which specify a particular solution can be applied to solve a problem in certain context. In our previous work, we have mapped such pattern concepts to contextual goal model elements (Table I) to semi-automate selection and application of security patterns [11]. Attack patterns, as a different type of pattern, are specified in the same spirit, but from an attacker's viewpoint, i.e., what an attacker wants to attack (problem), how does the attacker perform the attack (solution). Based on the semantics of attack pattern attributes[2], we have identified relevant attack pattern attributes that specify *problem, context, solution*, and indirectly map them to the contextual goal model elements, as shown in Table I.

[2]A full CAPEC schema, https://capec.mitre.org/data/xsd/ap_schema_v2.7.xsd

TABLE I: Pattern-related conceptual mappings

Primary Pattern Concept	**Attack Pattern Attribute**	**Goal Model Element**
Context	Attack Prerequisites; Technical Context	Context; Domain Assumption
Problem	Attack Motivation-Consequences; Domains of Attack	Goals
Solution	Attack Execution Flow	Tasks

TABLE II: Mappings between attack pattern impact and STRIDE threats categories

STRIDE	**Attack Impact**
Information Disclosure	Read application data Read memory Read files or directories
Tampering	Modify application data Modify application data memory Modify application data files or directories Unexpected states Alter execution logic
Denial of Service	DoS: instability DoS: resource consumption (CPU) DoS: resource consumption (memory) DoS: crash / exit / restart DoS: amplification DoS: resource consumption (other)
Elevation of Privilege	Gain privileges / assume identity Execute unauthorized code or commands Bypass protection mechanism
Spoofing	Gain privileges / assume identity
Repudiation	Hide Activities

On the basis of these mappings, we propose a systematic method to model attack patterns as contextual goal models. In the remainder of this section, we take the attack pattern *CAPEC-66: SQL Injection* as an example to illustrate each modeling step in detail (shown in Fig. 2). The complete specification of this pattern can be found online[3].

A. Modeling Attack Pattern Problems

A *Problem* solved by an attack pattern is actually the malicious intention that an attacker wants to achieve, which will be modeled as a *Goal* in the goal model. Specifically, we identify such malicious intentions from the attack pattern attributes *Attack Motivation-Consequences* and *Domains of Attack*.

To seamlessly integrate the attack pattern analysis with previous attack strategy analysis, we specify the malicious intention of each attack pattern using structured anti-goals (as introduced in Section II). In particular, we focus on the *Threat* that is imposed by an attack pattern and the *Target* that is exploited by an attack pattern, which are important for automatic pattern matching (Section IV). The threat information is elicited based on the attack impact specified in *Attack Motivation-Consequences*. As the CAPEC schema enumerates a total of 18 types of attack impacts, we map these impacts to STRIDE threat categories (Table II), using these categories as the *Threat* attribute in the resulting anti-goal. We elicit the

[3]https://capec.mitre.org/data/definitions/66.html

TABLE III: Typical targets of attack domains

Attack Domain	**Target**
Social Engineering	People
Supply Chain	Software Hardware
Communication	People Software Hardware
Software	Software
Physical Security	Hardware
Hardware	Hardware

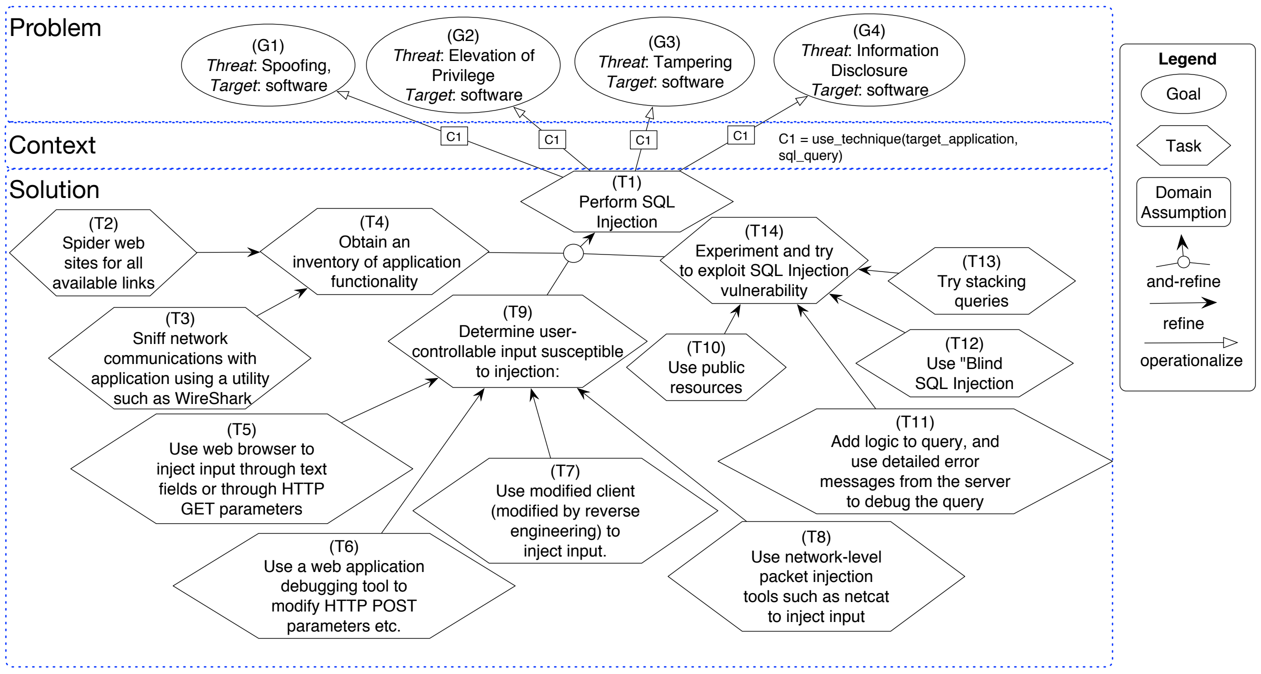
target information with respect to *Domains of Attack* of an attack pattern. In particular, the CAPEC schema includes six specific domains, and we identify typical attack targets for each of these domains, as shown in Table III. In the attack pattern example (Fig. 2), the pattern has an impact "*Read application data*", which is mapped into the threat *Information Disclosure* as modeled in the anti-goal *G4*. In addition, as this pattern is under the software domain, we specify the *Target* attribute of *G4* as *software*.

B. Modeling Attack Pattern Context

The context of an attack pattern specifies under which situation the attack can be applied to achieve an attacker's malicious intention. Thus, we model such context and associate it with the operationalization link between goals and tasks in the goal model, where the goals can only be operationalized into the tasks if the context holds (e.g., context *C1* shown in Fig. 2).

We obtain the context information of an attack pattern by looking at the attack pattern attributes *Attack Prerequisites* and *Technical Context*. As the context information is specified in natural language, analysts have to manually check such context during the application of attack patterns. After reviewing the context information of 102 attack patterns that we have modeled (Section III-D), we propose a set of formal predicates for specifying domain-independent context, which can be automatically checked based on the formalized concepts defined in three-layer requirements goal model [5]. Such predicates include $protected\_by$, $communicate$, $use\_technique$, use_data_from, and $accept\_user\_input$, detailed reasoning with these predicates will be presented in Section IV. It is worth noting that the proposed predicates do not express contexts that cannot be captured and checked in the three-layer requirements goal model. Such contexts are normally too detailed to capture in the domain model, for example, "*The targeted application runs with elevated OS privileges*". These cases will require manual analyst intervention via an interactive pop-up in the tool.

In our example, the *SQL Injection* pattern has an attack prerequisite "*SQL queries used by the application to store, retrieve or modify data*", which is then formalized using the predicate *use_technique* as shown in Fig. 2. Note that if there are several pieces of context information, they are specified in a conjunctive way, i.e., all of them have to hold in order to apply the corresponding attack pattern.



Problem
(G1) Threat: Spoofing, Target: software
(G2) Threat: Elevation of Privilege Target: software
(G3) Threat: Tampering Target: software
(G4) Threat: Information Disclosure Target: software
Context
C1
C1 = use_technique(target_application, sql_query)
Solution
(T1) Perform SQL Injection
(T2) Spider web sites for all available links
(T3) Sniff network communications with application using a utility such as WireShark
(T4) Obtain an inventory of application functionality
(T5) Use web browser to inject input through text fields or through HTTP GET parameters
(T6) Use a web application debugging tool to modify HTTP POST parameters etc.
(T7) Use modified client (modified by reverse engineering) to inject input.
(T8) Use network-level packet injection tools such as netcat to inject input
(T9) Determine user-controllable input susceptible to injection:
(T10) Use public resources
(T11) Add logic to query, and use detailed error messages from the server to debug the query
(T12) Use "Blind SQL Injection
(T13) Try stacking queries
(T14) Experiment and try to exploit SQL Injection vulnerability
Legend
Goal
Task
Domain Assumption
and-refine
refine
operationalize

Fig. 2: An example attack pattern model

C. Modeling Attack Pattern Solutions.

Solutions of attack patterns are specific attack actions that are performed by attackers using concrete attack techniques. We elicit such information from the attack pattern attribute *Attack Execution Flow*, and model each attack action as a task in the goal model. In particular, we focus on capturing the alternative attack actions for implementing the attack.

When modeling the solution section of the pattern, we first create a general task to summarize the overall attack that achieves the previously modeled anti-goals, such as *T1: Perform SQL Injection* shown in Fig. 2. The *Attack Execution Flow* is specified in terms of a sequence of attack steps that are required to fulfill the attack, thus, we capture this information as sub-tasks, and-refining the general task. In our example, the tasks *T4*, *T9*, and *T14* are individual attack steps specified in the *Attack Execution Flow*. Moreover, within each attack step, the attack pattern also describes alternative attack techniques that can be used for performing the attack step. Thus, we model each of these techniques as a refinement to the corresponding attack step. For example, in Fig. 2, the tasks *T2* and *T3* present two alternative attack techniques that can be applied to perform task *T4*. Note that when specifying the tasks, we reuse the original description provided by attack patterns, maintaining their security expertise in the model.

D. Applying the Modeling Method

In order to promote the adoption of our attack pattern analysis approach, we have pragmatically modeled 102 CAPEC patterns. In particular, we decide which the patterns to be modeled according to the following criteria: first, the select patterns should cover all attack pattern domains (as shown in Table III) in order to assist our attack analysis with comprehensive attack knowledge. Secondly, the pattern specifications need to be complete, i.e., all the attack pattern attributes that are required to build the contextual goal model should be well documented. Specifically, each CAPEC pattern has been specified with an attribute *Completeness*, valuing from *Hook*, *Stub*, to *Complete*, and thus we focus on patterns that have complete specification. It is worth noting that most attack patterns under the *Social Engineering* and *Physical* domains have only incomplete specifications. In order to preserve the comprehensive coverage of the selected patterns, instead of dropping all such incomplete patterns, we manually identify the required pattern attributes based on the *References* of those patterns, which have been specified for all attack patterns.

In the CAPEC repository, each attack pattern has been documented with related patterns that are more abstract or more detailed to it, using $ChildOf$ relations. We also capture such relations among the modeled 102 patterns, forming pattern hierarchies[4], which can help us to reduce the complexity of the attack operationalization analysis.

Overall, one author has spent three person-days pragmatically applying the proposed method to model 102 (out of 504) attack patterns. In particular, during the modeling practice, we noticed that this modeling task requires modelers to first thoroughly understand the rationale of the pattern to be modeled. On average, each pattern took the author 10-20 minutes to model, depending on the complexity of the pattern. The obtained models can be (re-)used to operationalize attack strategies in a semi-automatic way, using our prototype tool

[4]http://disi.unitn.it/~li/ap/pattern_hierarchy.pdf

(Section IV-V). A full list of modeled attack patterns can be found online[5]. Based on the above inclusion criteria, we argue the 102 patterns, as the initial repository of modeled attack patterns, are good enough to accommodate our attack analysis in different layers. We can incrementally add other patterns to further improve our attack analysis later. Also, other researchers can follow our method to model attack patterns and contribute to the repository.

IV. Operationalizing Attack Strategies Using Attack Patterns

In this section, we present a tool-supported systematic process for operationalizing attack strategies and eventually generating a collection of realistic attack alternatives that can satisfy an attacker's root anti-goal. The overall attack operationalization process is shown in Fig. 3, which can be semi-automated with the support of our prototype tool (Section V). In particular, after manually selecting a leaf anti-goal to analyze, we can automatically find attack patterns that are relevant to an anti-goal. Then, we semi-automatically determine the applicability of the relevant patterns. Finally, we automatically generate alternative attacks based on the applicable attack patterns. In the remainder of this section, we will describe each analysis step in detail.

A. Select A Leaf Anti-Goal

The operationalization analysis takes the attack strategy model as an input, which is obtained from our attack strategy analysis (discussed in Section II). An example of the attack strategy model is shown at the top of Fig. 4, which specifies what an attacker intends to attack and why. To operationalize different attack strategies, we need to iteratively perform operationalization analysis for each leaf anti-goal in the attack strategy model (i.e., *G8,G9,G10,G11*). As highlighted in Fig. 4, we select anti-goal *G10* for illustration.

B. Find Relevant Attack Patterns

To select appropriate attack patterns that operationalize the given anti-goal, we first identify all relevant attack patterns according to the ***problem*** we have modeled for each attack pattern. Since the problem of an attack pattern has also been modeled as structured anti-goals, we can identify relevant patterns by matching the threat and target specified in the structured anti-goals. Such a match is automated using the inference rule REV (Table IV), implemented in our supporting tool: given an anti-goal *G1* which imposes a threat *TH* to a target *TA1*, if an attack pattern *AP* has an anti-goal *G2* (as its problem), where *G2* imposes the same threat *TH* to a category of target *TA2* that *TA1* belongs to, then the attack pattern *AP* is relevant to the anti-goal. For example, as shown in Fig. 4, we identify CAPEC pattern *SQL Injection* is relevant to *G10*, as *G10* can be matched with the problem *G2* of *SQL Injection* according to rule REV. Similarly, the other two patterns *CAPEC-186 Malicious Software Update* and *CAPEC-100 Overflow Buffers* are also identified as relevant to the anti-goal *G10*. Note that, in Fig. 4, we use pentagons to represent collapsed attack pattern models in order to provide a better overview of the operationalization analysis (an excerpt of an uncollapsed pattern model is indicated in the bottom left corner).

When performing the attack pattern selection analysis, we take into account the pattern hierarchies in order to select the most appropriate patterns. For example, as the *SQL Injection* pattern has four child patterns which are also relevant to *G10*, we model them as refinements of *SQL Injection*, as shown in Fig. 4. According to such a hierarchy, a pattern and its ascendants represent only one operationalization alternative rather than multiple alternatives.

C. Identify Applicable Attack Patterns

After finding attack patterns that are relevant to an anti-goal, we further check their context to determine whether these patterns are applicable in current system context. We import the three-layer requirements goal model as the domain model that captures system context, against which we can automatically check the attack pattern context. To this end, we have defined a number of inference rules that specify the implication relation between the formal context predicates and the goal model predicates, which are shown as $CR1$-6 in Table IV. Such rules are defined based on the semantics of concepts defined in the three-layer requirements goal model. The application of these rules is domain-independent, i.e., any domain scenario that has been modeled as a three-layer requirements goal model can apply such rules, and thus further check the context of attack patterns. In particular, the formal context predicates are put in the left hand side, while the right hand side presents the corresponding facts in the three-layer requirements goal model. For instance, rule $CR1$ and $CR2$ express that if there is a dependency relation between two actors, it implies the context that the two actors are communicating with each other. Detailed information about the formal predicates of the three-layer goal model can be found in [5].

On top of these context check rules ($CR1$-6), we have defined specific applicability rules for each attack pattern, as different patterns require different contexts. For example, the rule APP shown in Table IV is specifically defined for the pattern *SQL Injection*. This rule says if the $sql\_injection$ pattern has been identified as relevant to an anti-goal G, and the target of G uses the technique sql_query, then the $sql\_injection$ pattern is applicable to operationalize G. The entire set of pattern-specific applicability rules, which we have defined for all the 102 attack patterns, can be found online[6].

When identifying the applicable pattern, we also consider the hierarchy among patterns. As "parent" patterns focus on a more abstract problem, concerning a more general context, we first check the parent patterns. If a parent pattern is identified as inapplicable, then all its child patterns will be identified as inapplicable without additional checking; if a parent pattern is

[5]http://disi.unitn.it/~li/ap/all_attack_pattern_models.pdf

[6]http://disi.unitn.it/~li/ap/attack_pattern_contexts.dl

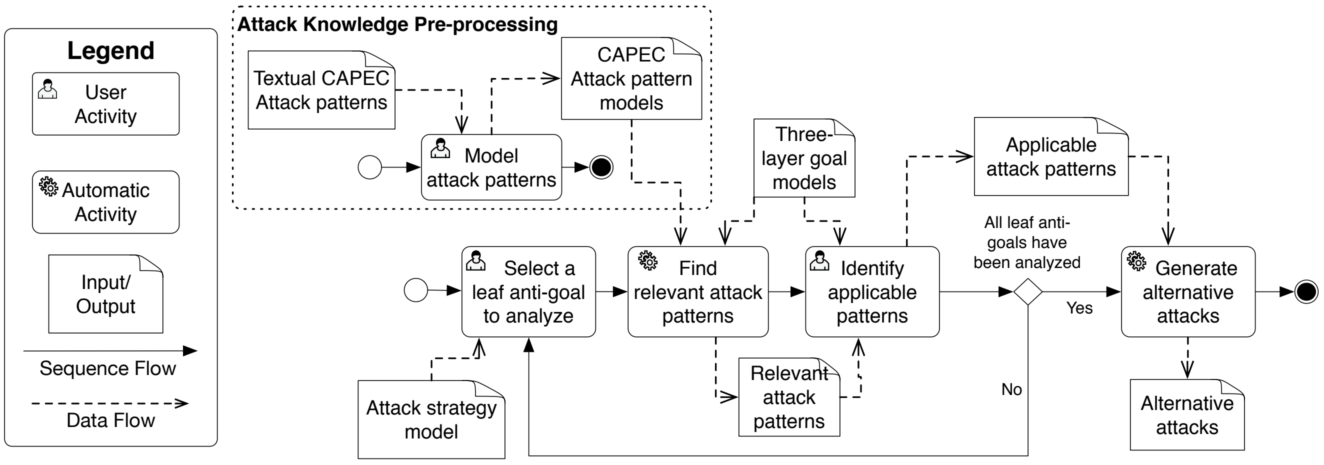

Legend
User Activity
Automatic Activity
Input/ Output
Sequence Flow
Data Flow
Attack Knowledge Pre-processing
Textual CAPEC Attack patterns
CAPEC Attack pattern models
Model attack patterns
Three-layer goal models
Applicable attack patterns
Select a leaf anti-goal to analyze
Find relevant attack patterns
Identify applicable patterns
All leaf anti-goals have been analyzed
Yes
No
Generate alternative attacks
Attack strategy model
Relevant attack patterns
Alternative attacks

Fig. 3: A systematic process for attack strategy operationalization

TABLE IV: Disjunctive Datalog rules for attack operationalization

REV	$relevant\_to(AP, G1)$:- $has\_threat(G1, TH), has\_target(G1, TA1), has(AP, G2), has\_threat(G2, TH),$ $has\_target(G2, TA2), isa(TA1, TA2)$
$CR1$	$communicate(A, B)$:- $depend(A, \_, B)$
$CR2$	$communicate(A, B)$:- $depend(B, \_, A)$
$CR3$	$use\_technique(A, R)$:- $has(A, R), resource(R)$
$CR4$	$use\_data\_from(A, B)$:- $depend(A, R, B), data(R)$
$CR5$	$accept\_user\_input(A)$:- $depend(A, R, B), data(R), human(B)$
$CR6$	$protected\_by(A, SM)$:- $sec\_goal(SG), has\_asset(SG, A), operationalize(SM, SG)$
APP	$applicable\_to(sql\_injection, G)$:- $relevant\_to(sql\_injection, G), has\_target(G, TA), use\_technique(TA, sql\_query)$
$ALT1$	$achieved(G1) \vee ... \vee achieved(Gn)$:- $refine(\{G1...Gn\}, G0), achieved(G0)$
$ALT2$	$achieved(G1)$:- $and\_refine(G1, G0), achieved(G0)$

applicable, then we will further check each of its child patterns. For example, as shown in Fig. 4, we first check the context of *SQL Injection, Malicious Software Update, Overflow Buffers*, (i.e., *C1, C2, C3*), which turns out that only *SQL Injection* is applicable (i.e., *C1* holds). Then, we further check the context of child patterns of *SQL Injection*, and identify that only pattern *Blind SQL Injection* is applicable.

D. Generate Alternative Attacks

Once identifying all applicable attack patterns to an anti-goal, we unfold the collapsed applicable attack patterns (i.e., the pentagon notations in Fig. 4) and show detailed attack behaviors (i.e., the solution part of an attack pattern model in Fig. 2). As such, we complete the entire attack model, including both the attack strategies and attack behaviors.

Once the entire attack model is obtained, we want to answer the question "*Is the root anti-goal achievable?*", "*If so, how many different combinations of attack behaviors can be performed to achieve the goal?*". To this end, we define disjunctive Datalog rules to exhaustively explore the space of alternatives, which has been implemented in the prototype tool using the DLV inference engine[7]. As shown in Table IV, the rule $ALT1$ means if a goal $G0$ is alternatively refined by sub-goals $G1...Gn$, then the achievement of **each** sub-goal serves as an alternative to achieve $G0$. On the other hand, if a goal $G0$ is and-refined by sub-goals $G1...Gn$, then the achievement of **all** the sub-goals is required to achieve $G0$, i.e., no more alternatives are introduced. This rationale is implemented as the rule $ALT2$. By applying these two rules to the root anti-goal of the attack model, we are able to obtain all the alternative attacks.

The identified alternative attacks will be prioritized and used to assess the criticality of security requirements, as part of our three-layer security requirements analysis [5], which is beyond the scope of this paper.

[7]www.dlvsystem.com

V. Supporting Tool

We have developed a prototype tool to support the application of the proposed attack analysis approach. This prototype is implemented on top of our existing modeling and analysis tool *MUSER* [15], which was designed to support our three-layer security requirements analysis framework. In this sense, the attack analysis approach can be well integrated with the three-layer security requirements analysis. In particular, the three-layer requirements goal model, which was built and used in security requirements analysis, can be easily imported and used by the attack analysis.

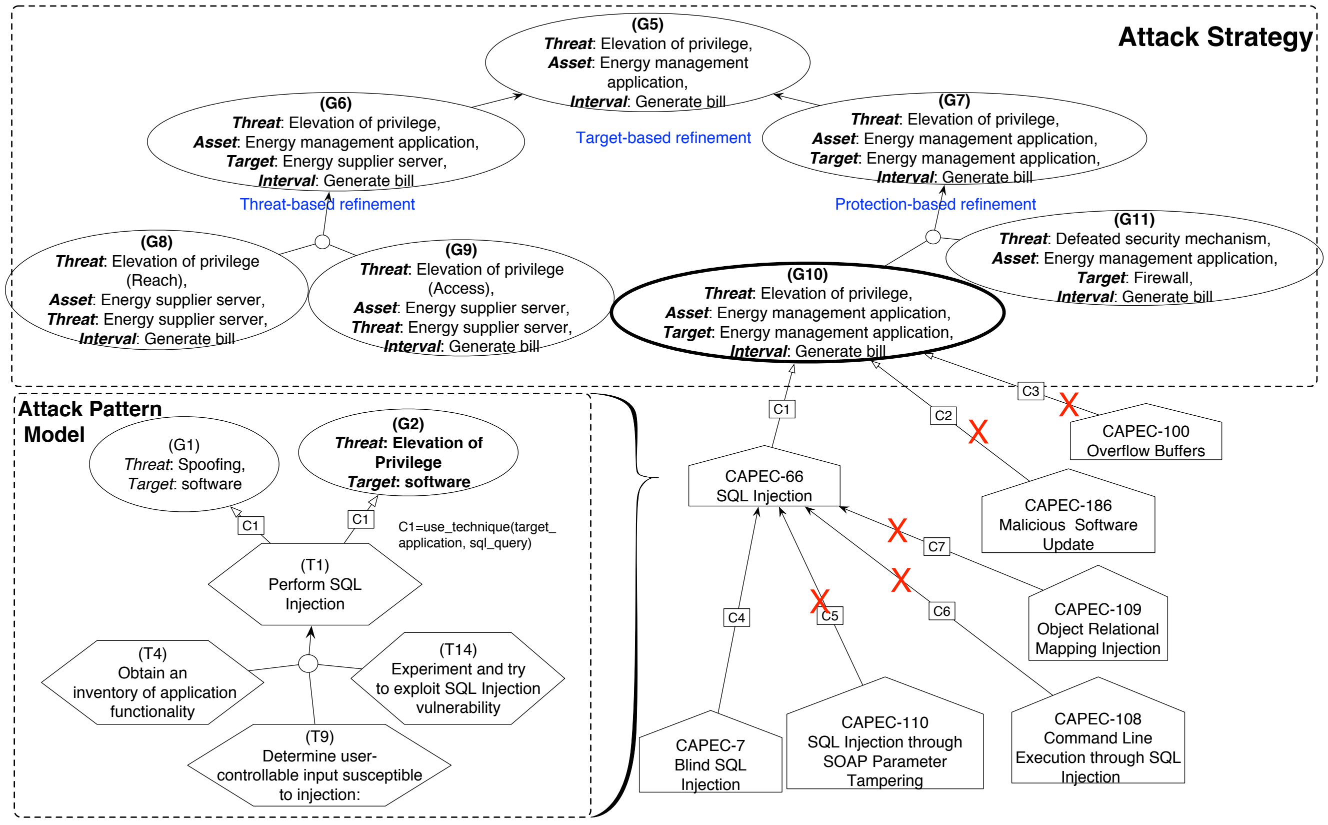

Attack Strategy
(G5) Threat: Elevation of privilege, Asset: Energy management application, Interval: Generate bill
(G6) Threat: Elevation of privilege, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
(G7) Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Target-based refinement
Threat-based refinement
Protection-based refinement
(G8) Threat: Elevation of privilege (Reach), Asset: Energy supplier server, Threat: Energy supplier server, Interval: Generate bill
(G9) Threat: Elevation of privilege (Access), Asset: Energy supplier server, Threat: Energy supplier server, Interval: Generate bill
(G10) Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
(G11) Threat: Defeated security mechanism, Asset: Energy management application, Target: Firewall, Interval: Generate bill
Attack Pattern Model
(G1) Threat: Spoofing, Target: software
(G2) Threat: Elevation of Privilege Target: software
C1
C1=use_technique(target_application, sql_query)
(T1) Perform SQL Injection
(T4) Obtain an inventory of application functionality
(T14) Experiment and try to exploit SQL Injection vulnerability
(T9) Determine user-controllable input susceptible to injection:
C1
C2
C3
C4
C5
C6
C7
CAPEC-66 SQL Injection
CAPEC-100 Overflow Buffers
CAPEC-186 Malicious Software Update
CAPEC-109 Object Relational Mapping Injection
CAPEC-7 Blind SQL Injection
CAPEC-110 SQL Injection through SOAP Parameter Tampering
CAPEC-108 Command Line Execution through SQL Injection

Fig. 4: Operationalize a leaf anti-goal using attack patterns

This prototype tool, as an "add-on" to *MUSER*, is a Java-based program. It translates graphical models in the canvas into Datalog facts, which are inferred according to the Datalog rules we have proposed in this paper (Table IV) using the inference engine DLV. All the inference results will be transferred back and visualized in the canvas. In particular, this prototype provides us with the following features that support the attack analysis.

- Support modeling attack strategies and attack patterns.
- Automatically identifying relevant attack patterns for each leaf anti-goal in the attack strategy model, using the proposed operationalization rules.
- Semi-automatically check the context of the identified relevant attack patterns to determine their applicability.
- Once all leaf anti-goals are operationalized, automatically generating all the multistage attack alternatives with realistic and concrete attack behaviors.

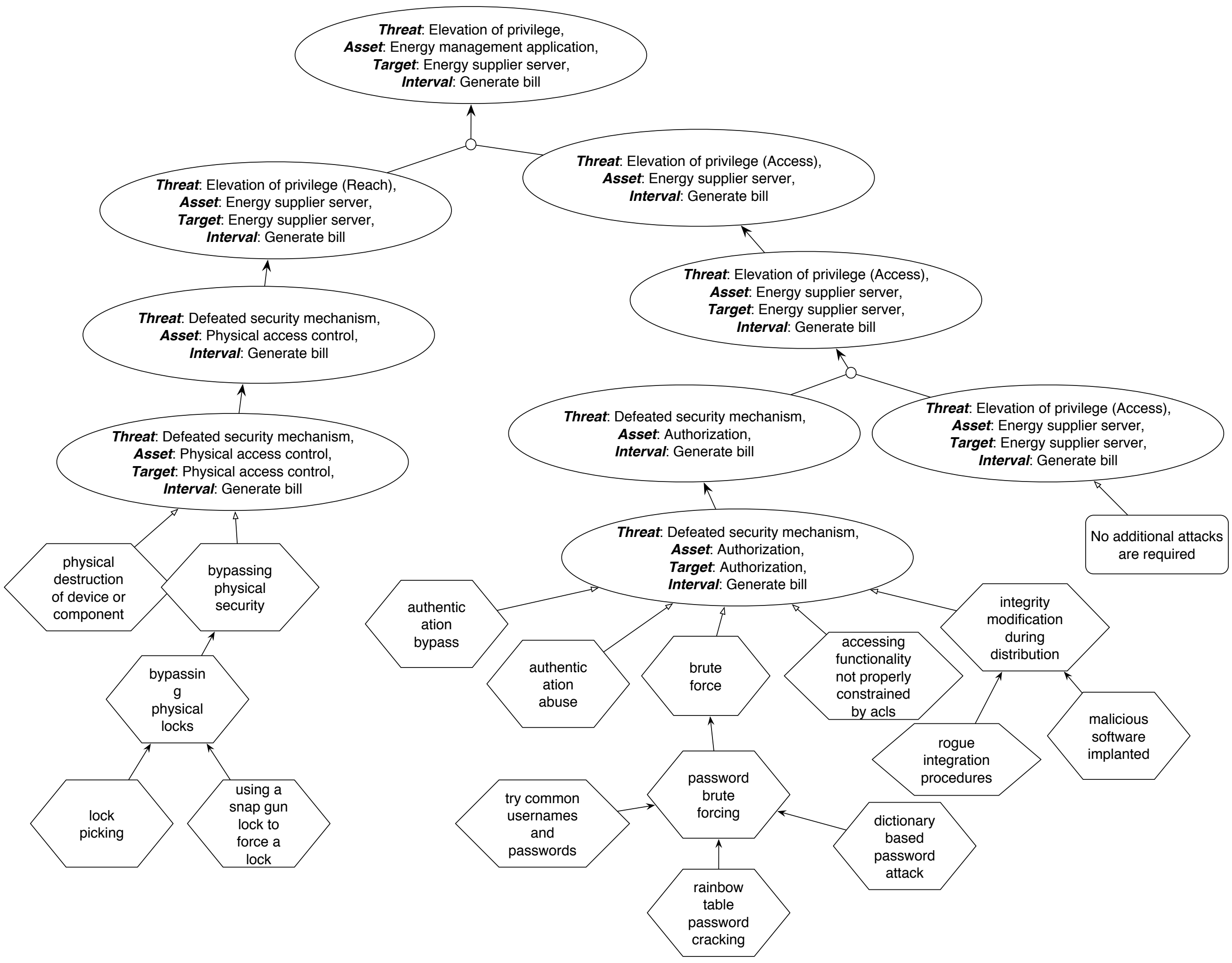
VI. Validation

In order to validate the proposed holistic attack analysis approach, we apply it to a smart grid case study. We focus on identifying applicable attacks which can tamper with a customer's energy consumption data. The advanced metering infrastructure of a smart grid presents a typical STS, which involves social actors (e.g., energy supplier and consumer) in a real-time metering and pricing process. In addition, a number of applications (e.g., energy management system) and physical hardware (e.g., smart meter) are used to support the interactive process. A three-layer requirements goal model, which was constructed and used to analyze holistic security requirements for smart grid [5], is taken as the domain model for the holistic attack analysis. The model is built based on the smart grid specification [16], [17], including 16 actors, 60 goals, 64 tasks, 6 resources, 54 (and-)refinement links, 49 operationalization link, and 15 dependency links. Given the domain model, we spent three person-hours in applying the holistic attack analysis approach with the tool support. Note that the domain model and all other models that have been built and analyzed during our validation can be found online[8].

A. Generate Attack Strategies

As the first part of our holistic attack framework, we identify alternative attack strategies using our systematic approach [6]. In particular, we start from determining an anti-goal to analyze, which attacks the integrity of energy consumption data, i.e., {*Threat: Tampering, Asset: Energy consumption data, Interval: Real-time pricing is applied*}. Having this anti-goal as a root goal, we manually applied anti-goal refinement patterns based on the domain model to systematically refine the anti-goal from an attacker's viewpoint and generate alternative attack strategies [6]. The resulting attack strategy model includes 53 anti-goals, 39 refinement links, and 20 and-refinement links, implying 12 alternative attack strategies.

[8]http://disi.unitn.it/~li/ap/validation.zip



Threat: Elevation of privilege, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Reach), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Physical access control, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Physical access control, Target: Physical access control, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Authorization, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
physical destruction of device or component
bypassing physical security
Threat: Defeated security mechanism, Asset: Authorization, Target: Authorization, Interval: Generate bill
No additional attacks are required
bypassin g physical locks
authentic ation bypass
authentic ation abuse
brute force
accessing functionality not properly constrained by acls
integrity modification during distribution
lock picking
using a snap gun lock to force a lock
try common usernames and passwords
password brute forcing
dictionary based password attack
rogue integration procedures
malicious software implanted
rainbow table password cracking

Fig. 5: An excerpt of the final attack model

B. Operationalize Attack Strategies

Once alternative attack strategies are obtained, we operationalize them in terms of realistic attacks, using the attack pattern approach proposed in this paper. Using the 102 attack patterns we have modeled, we first automatically identify relevant attack patterns for all the 14 leaf anti-goals in the attack strategy model, resulting in 368 attack patterns in total. Each of the leaf anti-goals, on average, has been matched with 26 relevant patterns. All the identified relevant patterns are automatically organized in a hierarchical manner to facilitate the applicability analysis. After semi-automatically checking the context of all the relevant attack patterns, we derive 28 applicable attack patterns for all of the 14 anti-goals, covering social, software, and physical attacks.

Take Fig. 5 as an example, which shows an excerpt of the final attack model. A wide spectrum of attack patterns have been identified concerning the root anti-goal, varying from social attacks (e.g., rogue integration procedures), to software attacks (e.g., rainbow table password cracking), to physical attack (e.g., lock picking). In particular, according to the attack model, we can identify alternative (multistage) attacks that achieve the root anti-goal, i.e., imposing the *elevation of privilege* threat to the *energy management application*. For instance, an attacker can first perform the *lock picking* attack to reach the *energy supplier server* and then perform the *rainbow table password cracking* attack to gain access into the *energy supplier server*.

Based on the complete attack model, including both high-level attack strategies and realistic attacks behaviors, we automatically generate 108 realistic attack alternatives, each of which consists of one or multiple attacks. To validate the analysis results, we compared them with a comprehensive security analysis performed on the same smart grid case [18], this study took a total of 16 person-months to identify threats, vulnerabilities and security requirements. We find out that our identified attack alternatives can cover all the threats to the integrity of energy consumption data that have been identified manually in the comparison case. Beyond that, our results can discover additional detailed attacks that can be performed by attackers, particularly how such attacks can be composed to form multistage attacks. For example, the comparison study discovered a high-level threat "*Tampering with SM's firmware*" (Table 4, T.5 in [18]), while our results yield a corresponding multistage attack which first performs *CAPEC-16: Dictionary-based Password Attack* (consisting of four detailed attack steps) to defeat the password-based authorization and gains access to the smart meter firmware, and then performs *CAPEC-186: Malicious Software Update* to tamper with the smart meter firmware.

By checking the vulnerabilities exploited by the identified attack alternatives, analysts can identify holistic and effective security requirements. We leave the specific description of how to feed these attacks back into the three-layer model (the back-edge in Fig. 1) to future work.

C. Threats to Validity

We here adopt a specific classification used by Runeson and Höst [19] to classify validity, which has an focus on case study research in software engineering. In particular, we describe a number of threats to internal validity, external validity, and reliability, respectively.

Internal validity considers the causal relations between factors investigated in the case study. We evaluated the effectiveness of our attack analysis approach by considering how many threats can be identified by the approach. As our approach relied on attack knowledge from existing attack knowledge sources (i.e., the 102 attack patterns we have modeled in Section III), the coverage and the quality of the imported attack knowledge can affect the analysis results, introducing a threat to the internal validity of our study.

Regarding this threat, firstly, we have modeled a significant number of attack patterns (102 patterns in total), which cover a broad scope of attacks (e.g., social attacks, software attacks, and physical attacks), ensuring the coverage of the reused attack knowledge as much as we can. In addition, we have provided detailed guidelines for modeling attack patterns, allowing other researchers to incrementally model new patterns and enrich the reused knowledge set. Secondly, as we have noticed that a number of CAPEC attack patterns have incomplete specifications when we pragmatically modeled the 102 attack patterns (reported in Section III-D), we manually complemented such patterns with corresponding information in order to ensure the quality of the reused attack patterns. In the future, we should keep updating the attack knowledge used in our approach with the recent advances of attack patterns in order to ensure the effectiveness of our approach.

External validity is concerned with to what extent it is possible to generalize the findings of our case study. Thus far we have only applied our approach to one case study, imposing a threat to the external validity of our study. However, as a confirmative case study, we purposely selected a typical STS rather than random cases to study, which might gain more insight into common situations [20]. In order to further mitigate this threat, in the future, we plan to perform more case studies.

Reliability is concerned with to what extent the data and the analysis are dependent on the specific researchers. This study was performed by only one person (i.e., the first author), which introduced a threat to reliability. Although such a threat can be mitigated by the fact that the performer is one of the method designers and has related security expertise in attack analysis, we acknowledge the need of having more people evaluate the analysis results. In the future, when performing additional case studies, we intend to have multiple participants work together.

VII. Discussion and Related Work

A. Attacker-oriented Analysis.

"*Know your enemies and know yourself, you will not be imperiled in a hundred battles.*" [21]

Inspired by the "Art of War" philosophy, several approaches have been proposed to model and analyze security attacks from an attacker's viewpoint. Lin et al. capture the requirements of a malicious user that subverts an existing requirement as anti-requirements, which are incorporated into abuse frames to represent threats and analyze security requirements [22]. Van Lamsweerde proposed to use anti-goals to model attacker's malicious intention, and then exploit alternative attacks by systematically refining such anti-goals [23]. Sindre and Opdahl extend traditional use cases to cover misuse cases, which describe harmful behaviors to a system performed by adversaries [24]. Building on the misuse cases, they propose a systematic process for eliciting security requirements. Elahi et al. extend goal models to model attacker templates which consist of malicious goals and tasks, based on which they assess system risks and identify countermeasures [25]. All of theses approaches require attacker knowledge as input, based on which they can analyze the influences of attacks on systems and identify corresponding countermeasures. In particular, these approaches make a strong assumption about the availability of relevant knowledge, e.g., "*The proposed framework assumes that analysts have knowledge about vulnerabilities, potential attacks, and proper countermeasure or can obtain such information*" [25].

We argue that performing the attack analysis from an attacker's viewpoint is a knowledge-intensive task, where the body of attack knowledge plays an important role. However, as pointed out by Souag et al., security knowledge is hard to acquire for software designers in reality [26]. Without bridging the knowledge gap, the assumptions made in the above approaches become unrealistic, preventing the real adaption of those attack analysis approaches. Our approach tackles this challenge by building on realistic and reusable knowledge from existing attack knowledge repository, and can complement the above attacker-oriented analysis approaches. In particular, our approach identifies alternative attacks based on realistic attack knowledge, which can be used by those approaches to perform particular analysis, e.g., analyzing the impact of the attacks.

B. Attack Pattern-based Knowledge Reuse.

Moore et al. first emphasized the importance of reusing known attack knowledge, which significantly affects the practicality of attack analysis methods [27]. Therefore, they define attack patterns, which encapsulate attack knowledge, in order to facilitate knowledge-intensive attack analysis. In particular, each pattern consists of four sections: goal, precondition, attack steps, and post-condition.

Other researchers have been inspired by Moore's work, and have defined various types of attack patterns. Gegick and Williams define software attack patterns in term of a sequence of events, using regular expressions [28]. Specifically, each event is expressed by its associated component, such as user, server, hard disk, etc. By automatically matching such patterns with system design, the approach can assist analysts in identifying system vulnerabilities. In [29], Fernandez et al. specify attack patterns (i.e., misuse pattern) based on POSA template [30], which they have also used to specify security patterns. The POSA template includes much more sections than the initial one defined by Moore et al., such as context, known uses, countermeasures, etc., which contribute to the practicality of attack pattern-based analysis. Although the above approaches contribute to the theoretical foundation of attack patterns, they have not been pragmatically applied to develop attack patterns. For example, Moore et al. illustrate their approach with four patterns [27], and we are unaware of subsequent work to develop further patterns; Fernandez has only developed three misuse patterns, as presented in his recent book [31].

Compared to the above theoretical approaches which focus on defining attack patterns, CAPEC emphasizes the pragmatic development of security patterns, which is initiated as a baseline catalog of attack patterns along with a comprehensive schema and classification taxonomy and has accumulated 504 attack patterns thus far. Since CAPEC provides a significant amount of practical security knowledge, it is receiving an increase in attention from both academia and industry. Thus, we choose CAPEC as the realistic attack knowledge source used in our approach.

One of the challenges of using the CAPEC repository is dealing with it's considerable size. Kaiya et al. define term-maps, which link terms in requirements specifications to specific security terms used in CAPEC, so as to automatically associate attack patterns to requirements specifications and further derive security requirements [10]. Engebretson and Pauli enrich the CAPEC attack patterns with the concepts parent threat and parent mitigation in order to facilitate the navigation among the large number of attack patterns [8]. Yuan et al. map CAPEC patterns to the STRIDE threat categories, based on which they develop a tool to facilitate the retrieval of CAPEC patterns [9]. However, all the above approaches do not check the applicability of attack patterns based on context. Thus, the retrieved patterns in these approaches may still include many non-applicable patterns, which need to be further analyzed by analysts in order to determine their applicability.

Our approach contributes to the context-based pattern selection by clearly modeling the *context*, *problems*, and *solutions* of each attack pattern in terms of contextual goal models, which can be semi-automatically analyzed based on domain models. As such, our proposal can help analysts to identify applicable attack patterns in a more effective manner.

Apart from the retrieval and selection issues of CAPEC patterns, existing approaches only focus on reusing the knowledge of attack behaviors from the CAPEC patterns, without mining the intention of attacks. For example, Kim and Kim extract possible attack behaviors from CAPEC, and specify such behaviors using formal language in order to simulate attacks within specific system settings [32]. However, their approach cannot analyze combined behaviors from different attack patterns, and thus can only detect limited number of attacks. Our proposal is a continuation of our previous work concerning attacker intention analysis [6]. By associating the CAPEC patterns with an attacker's high-level intention, we are able to capture multistage attacks which consists of several attack patterns, revealing a larger space of attacks.

C. Practical Challenges in Reusing Attack Patterns.

Encapsulating knowledge as structural patterns is an effective way of reusing knowledge. Various patterns have been proposed to relieve knowledge-intensive analysis in different domains, such as requirements patterns, design patterns, security patterns, attack patterns, etc. Souag et al. survey reusable knowledge-based security requirements engineering approaches over the last 20 years, which shows that 9 out of 95 surveyed papers represent reusable knowledge in the form of patterns (other forms include catalogs, taxonomies, etc.) [26]. Although patterns can be reused in a comparatively easy manner, Araujo et al. have pointed out that analysts need first to have a thorough understanding of available patterns in order correctly select and apply them [33]. This issue has been confirmed by us when applying our approach to the CAPEC patterns. When dealing with a small number of patterns, this issue will not be a challenge, as the analysts can afford the learning costs. However, we argue that such issue can impose practical challenges when analyzing a large number of patterns, e.g., the 504 CAPEC attack patterns. Facing this challenge, our approach formalizes the context of attack patterns and semi-automates the context-based attack pattern selection analysis, relieving analysts from scrutinizing the detailed context of all patterns. We have applied the approach to pragmatically process 102 (out of 504) attack patterns. Such processing must be performed only once, and the resulting models can be directly used by our prototype tool.

VIII. Conclusions and Future Work

In this paper, we first propose a systematic method to construct contextual goal model from CAPEC attack patterns, and have practically applied this method to model 102 patterns. Based on these models, we propose a systematic analysis process and a collection of formal inference rules in order to help analysts to effectively select attack patterns for operationalizing an attacker's anti-goals. A prototype tool has been developed to semi-automate such analysis. In particular, this analysis approach is integrated with our previous work which holistic analyzes attack strategies from an attacker's viewpoint, and thus completes an entire attack analysis framework. We use a smart grid case study to validate the entire attack analysis framework, the output of which is compared with an expert process that was performed manually. The comparison shows that our approach can effectively identify realistic attacks, especially those involving multistage attacks.

In the future, beyond the 102 CAPEC patterns that have been modeled, we intend to model more domain-specific attack patterns, especially physical attack patterns, in order to improve the usefulness of our framework in conducting multi-layered analysis. Moreover, we will further validate the utility of this attack analysis approach in the context of our three-layer security requirements framework. Finally, based on the attack model we constructed, we want to investigate how to timely identify and tackle multistage attacks at runtime.

Acknowledgements

Trento authors are supported by the ERC advanced grant 267856, titled "Lucretius: Foundations for Software Evolution". Jennifer Horkoff is supported by an ERC Marie Skodowska-Curie Intra European Fellowship (PIEF-GA-2013-627489), and by a Natural Sciences and Engineering Research Council of Canada Postdoctoral Fellowship (Sept. 2014 - Aug. 2016).

References

[1] L. Ponemon, "Cost of data breach study: Global analysis," Poneomon Institute sponsored by IBM, Tech. Rep., 2015.

[2] K. D. Mitnick and W. L. Simon, *The art of deception: Controlling the human element of security*. John Wiley & Sons, 2011.

[3] T. Li, E. Paja, J. Mylopoulos, J. Horkoff, and K. Beckers, "Holistic security requirements analysis: An attacker's perspective," in *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*. IEEE, 2015, pp. 282–283.

[4] T. Li, J. Horkoff, K. Beckers, E. Paja, and J. Mylopoulos, "A holistic approach to security attack modeling and analysis," in *Proceedings of the Eighth International i* Workshop*, 2015, pp. 49–54.

[5] T. Li and J. Horkoff, "Dealing with security requirements for socio-technical systems: A holistic approach," in *Advanced Information Systems Engineering (CAiSE 2014)*. Springer International Publishing, 2014, pp. 185–200.

[6] T. Li, J. Horkoff, E. Paja, K. Beckers, and J. Mylopoulos, "Analyzing attack strategies through anti-goal refinement," in *The Practice of Enterprise Modeling (PoEM 2015)*. Springer International Publishing, 2015, pp. 75–90.

[7] A. Shostack, *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.

[8] P. H. Engebretson and J. J. Pauli, "Leveraging parent mitigations and threats for capec-driven hierarchies," in *Sixth International Conference on Information Technology: New Generations, 2009. ITNG'09.*, 2009, pp. 344–349.

[9] X. Yuan, E. B. Nuakoh, J. S. Beal, and H. Yu, "Retrieving relevant capec attack patterns for secure software development," in *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. ACM, 2014, pp. 33–36.

[10] H. Kaiya, S. Kono, S. Ogata, T. Okubo, N. Yoshioka, H. Washizaki, and K. Kaijiri, "Security requirements analysis using knowledge in capec," in *Advanced Information Systems Engineering Workshops*. Springer, 2014, pp. 343–348.

[11] T. Li, J. Horkoff, and J. Mylopoulos, "Integrating security patterns with security requirements analysis using contextual goal models," in *The Practice of Enterprise Modeling (PoEM 2014)*. Springer Berlin Heidelberg, 2014, pp. 208–223.

[12] Y. Asnar, T. Li, F. Massacci, and F. Paci, "Computer aided threat identification," in *Commerce and Enterprise Computing (CEC), 2011 IEEE 13th Conference on*. IEEE, 2011, pp. 145–152.

[13] R. Ali, F. Dalpiaz, and P. Giorgini, "A goal-based framework for contextual requirements modeling and analysis," *Requirements Engineering*, vol. 15, no. 4, pp. 439–458, 2010.

[14] C. Alexander, S. Ishikawa, and M. Silverstein, *A pattern language: towns, buildings, construction*. Oxford University Press, 1977, vol. 2.

[15] T. Li, J. Horkoff, and J. Mylopoulos, "A prototype tool for modeling and analyzing security requirements from a holistic viewpoint," in *The CAiSE'14 Forum at the 26th International Conference on Advanced Information Systems Engineering*, 2014, pp. 185–192.

[16] J. Cuellar and S. Suppan, "A smart metering scenario," Network of Excellence on Engineering Secure Future Internet Software Services and Systems, eRISE 2013, 2013.

[17] N. Framework, "Roadmap for smart grid interoperability standards, release 2.0," *NIST special publication*, vol. 1108R2, 2012.

[18] H. Suleiman and D. Svetinovic, "Evaluating the effectiveness of the security quality requirements engineering (square) method: a case study using smart grid advanced metering infrastructure," *Requirements Engineering*, vol. 18, no. 3, pp. 251–279, 2013.

[19] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[20] H. Estrada, A. M. Rebollar, O. Pastor, and J. Mylopoulos, "An empirical evaluation of the i* framework in a model-based software generation environment," in *Advanced Information Systems Engineering*. Springer, 2006, pp. 513–527.

[21] S. Tzu, *The art of war*. Shambhala Publications, 2011.

[22] L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett, "Introducing abuse frames for analysing security requirements," in *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*. IEEE, 2003, pp. 371–372.

[23] A. V. Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," in *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004, pp. 148–157.

[24] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements Engineering*, vol. 10, no. 1, pp. 34–44, 2005.

[25] G. Elahi, E. Yu, and N. Zannone, "A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities," *Requirements Engineering*, vol. 15, no. 1, pp. 41–62, 2010.

[26] A. Souag, R. Mazo, C. Salinesi, and I. Comyn-Wattiau, "Reusable knowledge in security requirements engineering: a systematic mapping study," *Requirements Engineering*, pp. 1–33, 2015.

[27] A. P. Moore, R. J. Ellison, and R. C. Linger, "Attack modeling for information security and survivability," CMU-SEI-2001-TN-001., Tech. Rep., 2001.

[28] M. Gegick and L. Williams, "Matching attack patterns to security vulnerabilities in software-intensive system designs," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[29] E. B. Fernandez, N. Yoshioka, and H. Washizaki, "Modeling misuse patterns," in *2009 International Conference on Availability, Reliability and Security*. IEEE, 2009, pp. 566–571.

[30] F. Buschmann, K. Henney, and D. Schimdt, *Pattern-oriented Software Architecture: On Patterns and Pattern Language*. John Wiley & Sons, 2007, vol. 5.

[31] E. Fernandez-Buglioni, *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.

[32] J.-Y. Kim and H.-J. Kim, "Defining security primitives for eliciting flexible attack scenarios through capec analysis," in *Information Security Applications*. Springer, 2014, pp. 370–382.

[33] I. Araujo and M. Weiss, "Linking patterns and non-functional requirements," in *Proceedings of the Ninth Conference on Pattern Language of Programs (PLOP 2002), September 8-12, 2002*, 2002.