Sequential Pattern Mining for ICT Risk Assessment and Prevention

Michele D'Andreagiovanni, Fabrizio Baiardi, Jacopo Lipilini, Salvatore Ruggieri(✉), and Federico Tonelli

Dipartimento di Informatica, Università di Pisa,
Largo B. Pontecorvo 3, 56127 Pisa, Italy
ruggieri@di.unipi.it

Abstract. Security risk assessment and prevention in ICT systems rely on the analysis of data on the joint behavior of the system and its (malicious) users. The Haruspex tool models intelligent, goal-oriented agents that reach their goals through attack sequences. Data is synthetically generated through a Monte Carlo method that runs multiple simulations of the attacks against the system. In this paper, we present a sequential pattern mining analysis of the database of attack sequences. The intended objective is twofold: (1) to exploit the extracted patterns for the design of attack counter-measures, and (2) for gaining a better understanding of the "degree of freedom" available for the attackers of a system. We formally motivate the need for using maximal sequential patterns, instead of frequent or closed sequential patterns, and report on the results on a specific case study.

Keywords: Security risk assessment · Attack sequences
Sequential pattern mining · Maximum coverage problem

1 Introduction

Approaches for security risk assessment and prevention in ICT systems rely on the analysis of data on the joint behavior of the system and its (malicious) users. Such data is typically collected over time after the deployment of a system. This results in two main drawbacks. First, the approach cannot assess and manage risk at design time, but only with *a-posteriori* sub-optimal remedies. This prevents a principled *by-design* approach in risk analysis and management [8]. Second, the data available for the analysis is both scarce and biased because one has to wait for *real* attacks to take place and extreme events such as low frequency - high impact attacks are not included.

The Haruspex tool [2] aims at predicting the behavior of a system under attacks even before its deployment. It models intelligent, goal-oriented agents that reach their goals through attack sequences. Data is synthetically generated through a Monte Carlo method that runs multiple simulations of the attacks against the system. Simulations rely on formal models of the target system and

A. Cerone and M. Roveri (Eds.): SEFM 2017 Workshops, LNCS 10729, pp. 25–39, 2018.
https://doi.org/10.1007/978-3-319-74781-1_2

of the threat agents and they return a large sample of attack sequences. The analysis of such data can reveal paths of successful attacks in an ICT network, and suggest possible patches that block them at design-time.

This paper intends to improve simple statistical summaries of attack sequences. We exploit knowledge discovery approaches to extract informative patterns from the data in the form of sequential patterns. The intended objective is twofold: (1) to exploit the extracted patterns for the design of attack counter-measures, and (2) for gaining a better understanding of the "degree of freedom" available for the attackers of a system. Objective (1) can be achieved by using data mining models for making predictions on the next attack of a threat agent that is following a matched pattern [4,9,11], hence enforcing dynamic counter-measures. We instead take a different approach that selects from the set of extracted patterns a set that covers as much as possible the successful attack sequences of the threat agent. Such a set represents a global description of the strategies of the agent. By reasoning on such characterization, we claim that an analyst can derive useful information for the design of static counter-measures, which are especially useful at system design time. Such a description, possibly at alternative abstraction levels, is also useful to understand the "degree of freedom" of an attacker that targets the ICT system, namely objective (2).

The paper is organized as follows. Section 2 introduces the Haruspex tool. Section 3 describes a case study including a sample ICT system and modeling of threat agents. Classes of sequential patterns and measures of interest for the field of ICT risk assessment are then introduced in Sect. 4. Next, in Sect. 5, we introduce a notion of covering for sets of sequential patterns and formally prove that maximal sequential patterns is the most appropriate class for computing covers. Section 6 discusses the covers found for attack sequences of our case study. Finally, we discuss related work and summarize the contribution of the paper.

2 ICT Risk Assessment: The Haruspex Approach

This section describes the tools of the Haruspex suite [1] that simulate the attacker behaviors to produce synthetic attack sequences. In the following, we denote by *user* the team that interacts with such tools to assess and manage the security risks of an ICT system. Such a *target system* can be already existing or only in its design phase. The suite builds the models of the target system and of the threat agents, or *attackers*, to simulate the agent attacks in the scenario of interest. The accuracy of the simulation depends on the accuracy of the tools that build the model of the system and those of the agents. Table 1 defines the abbreviations we use in the following.

The *builder* and the *descriptor* are the Haruspex tools that model, respectively, the system infrastructure and the agents. Then, the *engine* uses these models to apply the Monte Carlo method and to run a number of simulations of the agent attacks.

Table 1. List of abbreviations.

S	The target system
n	A node of S
ag	A threat agent (attacker)
at	An elementary attack
v	A vulnerability
$v(at)$	The vulnerabilities enabling at
$pre(at)$	The rights to execute at
$res(at)$	The resources to execute at
$post(at)$	The rights granted if at succeeds
$succ(at)$	The success probability of at
$time(at)$	The execution time of at
$\lambda(ag)$	The look-ahead of ag

2.1 Modeling an Infrastructure

The *builder* is the first tool the user applies to build a model of the target system S that decomposes S into a network of nodes and the resulting nodes into components, i.e. hardware/software modules. Each component is affected by zero or more *vulnerabilities* that enable some *attacks* [13,15,16]. Haruspex covers social engineering attacks by modeling users of S as further components with the proper vulnerabilities. If any vulnerability in $v(at)$ is effective, at is *enabled* and succeeds with a probability $succ(at)$, otherwise it fails. For each vulnerability v that affects a node n, the *builder* computes $pre(at)$ and $post(at)$ for each attack v enables. These properties drive the sequence that an agent will compose. The *builder* computes $pre(at)$ and $post(at)$ by matching predefined patterns against the description of v in some *de facto* standard databases, e.g. the Common Vulnerability Enumeration [1,15]. The *builder* considers any attack sequence of ag, even those that involve distinct nodes of S. To let the *builder* model sequences that involve distinct nodes, the user has to describe to the *builder* the logical topology of nodes in S and the components of S that controls it, e.g. firewalls.

2.2 Modeling Agents

The *descriptor* builds an agent model starting from four properties the user defines for each agent:

1. the initial rights;
2. the goal(s);
3. the selection strategy;
4. the value of $\lambda(ag)$ of the look-ahead.

A goal is the set of rights that ag get after reaching the target node n_f, and it results in an *impact*, i.e. a loss for the owner of S [3]. An agent ag can attempt an elementary attack at. The attack is successful if the agent owns the rights defined in $pre(at)$, either as initial rights or as rights granted after previous successful attacks. ag sequentially executes elementary attacks. The attack sequence is *successful* if it reaches n_f.

Each ag is paired with a selection strategy that ranks the next attack sequences ag may execute according to its goals, its current set of rights and its preferences. $\lambda(ag)$ defines the look-ahead of ag, i.e., the largest number of attacks in the sequences ag ranks to select the one to execute. The strategy always selects one of the sequences leading to a goal, if it exists. Otherwise, the strategy ranks sequences according to attack attributes only. In this case, ag may select a sequence with useless attacks, i.e. their granted rights are useless to reach a goal. In the current version of the suite, the user can pair ag with one of the following strategies:

1. *maxProb:* returns the sequence with the best success probability,
2. *maxIncr:* returns the sequence granting the largest set of rights,
3. *maxEff:* returns the sequence with the best ratio between success probability and execution time of attacks,
4. *maxAtt:* returns the sequence granting the rights that enable the execution of the largest number of attacks,
5. *SmartSubnetFirst:* returns any attack that ag may execute and that increases these rights. It prefers attacks that enable ag to enter another subnetwork.

If $\lambda(ag) = 0$, then ag can only adopt the *SmartSubnetFirst* strategy, This strategy prefers the attacks that enable ag to enter a distinct subnetwork and it is *Smart* because it prefers the attacks that ag can execute.

Haruspex models both the time to implement the attack and the one to collect information to select the attack. Larger values of $\lambda(ag)$ result in a more accurate selection that may avoid useless attacks, on the other hand, the time ag spends to acquire information on the target system increases with $\lambda(ag)$.

2.3 Simulation Engine

The *engine* uses the infrastructure model and those of the agents to apply the Monte Carlo method. It executes an experiment with several independent runs that simulate, for a maximum time period, the attacks of a set of agents and the discovery of potential vulnerabilities. To guarantee run independence, the *engine* re-initializes the state of S and of any agent before starting a new run. An experiment ends either after executing the specified number of runs or when a predefined statistic reaches the required confidence level. Each experiment returns a database of attack sequences collected in the runs.

In each time step, the *engine* considers any idle agent that still has to reach a goal. For each agent ag, the *engine* considers the current set of rights of ag and computes the sequences with, at most, $\lambda(ag)$ attacks that ag can select.

Then the *engine* applies the selection strategy of ag and it simulates at, the first attack of the sequence the strategy returns. If at succeeds and ag has reached a goal, the *engine* updates the corresponding impact.

3 Case Study

In this section, we describe a case study that will be the subject of sequential pattern mining analysis later on. Figure 1 shows the topology of the target system network. It consists of 36 nodes (hosts) plus routers and switches (they are not part of the model). The node $n_e = 0$ is the entry node, i.e., all attack sequences start from it, and the node 34 is the goal node, i.e., $n_f = 34$. An attack sequence is then considered successful if it reaches n_f.

The system vulnerabilities result in 2,859 elementary attacks, belonging to 192 attack types. An elementary attack at is specific of a vulnerability at a node, i.e. the same attack that targets different nodes give raises to distinct elementary attacks. Attack types group elementary attacks according to the CVE [1,15] vulnerability they exploit.

Example 1. An example of elementary attack is one that grants administrative privileges on the software HP Data Protector at node 1. The attack has type:

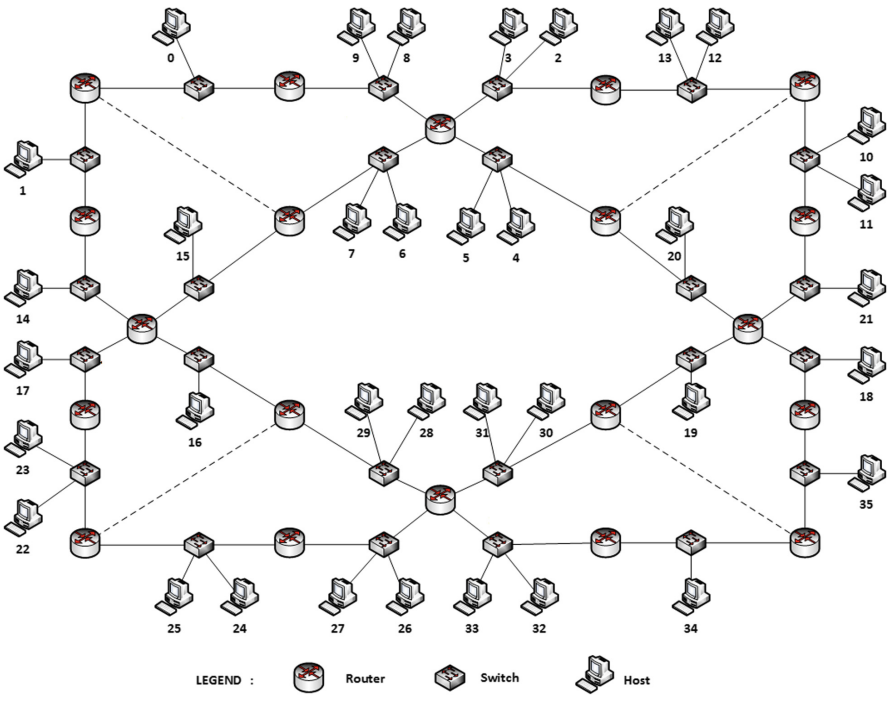

Fig. 1. Topology of the case study system.

Table 2. Threat agents in the case study.

ag	Strategy	$\lambda(ag)$
A0	*SmartSubnetFirst*	0
A1	*maxIncr*	1
A2	*maxIncr*	2
A3	*maxProb*	1
A4	*maxProb*	2
A5	*maxAtt*	2

HP Data Protector Remote Command Execution. An attack of the same type but at another node is a different elementary attack.

As another example, if an attacker wants to remotely execute arbitrary code may exploit an attack of type *MS12020: Vulnerabilities in Remote Desktop Could Allow Remote Code Execution (2671387) (uncredentialed check)*.

As a third example, the *Unencrypted Telnet Server* type models an attack to steal sensitive information through Telnet protocol.

The risk assessment in the case study considers 6 threat agents A0-A5. Table 2 lists their selection strategies and look-aheads.

The simulation engine has executed experiments up to the collection of 100K successful attack sequences for each agent. Data collected for each run includes the nodes the agent has targeted, the attacks attempted and its success, the time of each attack, plus other detailed information which this paper neglects. Attack sequences have a length between 5 and 144, for a total of 23.7M elementary attacks in the sequences the threat agents select in the simulations. The agents have actually selected only 435 of the 2,859 possible elementary attacks.

Example 2. A successful attack sequences involves the following nodes: *1, 9, 6, 5, 2, 20, 21, 18, 35, 34* of the network in Fig. 1. In such a sequence, the agent exploits an attack of type *HP Data Protector < A.06.20 Multiple Vulnerabilities* against node 1. Then, the sequence exploits the attack type *HP Data Protector Remote Command Execution* to gain administrative privileges on nodes 9, 6, 5, and 2. Starting from these nodes, the attacker exploits further attack types, e.g. *MS12020: Vulnerabilities in Remote Desktop Could Allow Remote Code Execution (2671387) (uncredentialed check)* and *HP System Management Homepage < 7.0 Multiple Vulnerabilities*. In the end, the agent launches an attack with type *Unencrypted Telnet Server* against the node 34.

4 Sequential Pattern Mining over Attack Sequences

This section recalls the framework of sequential pattern mining and instantiates it on the basis of the data the Haruspex suite returns. We refer the reader to

[6,12,14] for surveys on sequential pattern mining theory and applications. A large suite of algorithms for extracting classes of sequential patterns is implemented in Java [5].

4.1 Sequence Databases

Let $I = \{i_1, \ldots, i_m\}$ be a fixed set of *items*, where $m > 0$. Items can be symbols or, simply, natural numbers. An *itemset* A is a set of items, i.e., $A \subseteq I$. A *sequence* $S = \langle A_1, \ldots, A_l \rangle$ is an ordered list of non-empty itemsets. We write $l(S) = l$ to denote the length $l \geq 0$ of S. A sequence $S_1 = \langle A_1, \ldots, A_l \rangle$ is a *sub-sequence* of $S_2 = \langle B_1, \ldots, B_k \rangle$ if there exists $1 \leq \pi_1 < \ldots < \pi_l \leq k$ such that $A_1 \subseteq B_{\pi_1}, \ldots, A_l \subseteq B_{\pi_l}$. In such a case, we write $S_1 \sqsubseteq S_2$. Also, we say that S_2 *contains* S_1. We write $S_1 \sqsubset S_2$ when $S_1 \sqsubseteq S_2$ and $S_1 \neq S_2$, and say that S_2 *strictly contains* S_2. A *sequence database* $\mathcal{D} = \{S_1, S_2, \ldots, S_n\}$ is a multiset of sequences. $|\mathcal{D}|$ is the size of $\mathcal{D}$.

With reference to the case study, attack sequences can be modeled by considering items at different granularities. We will consider two scenarios (several others are possible).

Definition 1. *In scenario S*1*, items are elementary attacks at. Since each agent executes one attack at a time, itemsets $A \subseteq I$ are singletons of elementary attacks, i.e., $|A| = 1$. Moreover, in this scenario, we include in a sequence $S = \langle at_1, \ldots, at_l \rangle$ only successful elementary attacks at_i's.*

Example 3. Reconsider Example 2. The attack sequence in scenario S1 would be modeled as $\langle at_1, at_9, at_6, at_5, at_2, at_{20}, at_{21}, at_{18}, at_{35}, at_{34} \rangle$, where at_j is some elementary attack attempted at node n_j. E.g., at_9 is the HP Data Protector Remote Command Execution at node n_9.

Definition 2. *In scenario S*2*, items are eithers nodes n or attack types t. Again, since agents execute the attacks sequentially, itemsets either consists of node and type ($\{n, t\}$), or of node only ($\{n\}$), or of type only ($\{t\}$). Also in this scenario, we include in a sequence $S = \langle \{n_1, t_1\}, \ldots, \{n_l, t_l\} \rangle$ only information from successful attacks.*

Example 4. Reconsider Example 2 again. The attack sequence in scenario S2 would be modeled as $\langle \{n_1, t_1\}, \{n_1, t_9\}, \{n_1, t_6\}, \{n_1, t_5\}, \{n_1, t_2\}, \{n_1, t_{20}\}, \{n_1, t_{21}\}, \{n_1, t_{18}\}, \{n_1, t_{35}\}, \{n_1, t_{34}\} \rangle$, where t_j is the attack type attempted at node n_j. E.g., t_9 is the *HP Data Protector Remote Command Execution* attack type.

4.2 Sequential Patterns and Measures of Interest

Sequences can be adopted as abstractions (patterns) of subsets of $\mathcal{D}$. We denote such abstractions as sequential patterns to differentiate them from sequences in $\mathcal{D}$. Formally, a *sequential pattern* SP is a sequence. The *cover* of a sequential pattern SP is the set of sequences in $\mathcal{D}$ that contain SP. In symbols, $cover_{\mathcal{D}}(SP) = \{S \in \mathcal{D} \mid SP \sqsubseteq S\}$. The goal of *sequential pattern mining* is

to extract from a sequence database a set of interesting sequential patterns. An objective measure of interest is support. The *absolute support* (or, simply, support) of SP is the size of its cover: $supp_{\mathcal{D}}(SP) = |cover_{\mathcal{D}}(SP)|$. The *relative support* of SP is the fraction of its cover over $\mathcal{D}$: $relsupp_{\mathcal{D}}(SP) = |cover_{\mathcal{D}}(SP)|/|\mathcal{D}|$. We omit the subscripts $\mathcal{D}$ when clear from the context.

Example 5. An example sequential pattern in scenario S1 is the following: $SP = \langle at_9, at_2, at_{21} \rangle$. It covers all sequences that perform at some point attack at_9 (which is, say, at node n_9), and after zero or more steps attack at_2 (at node n_2), and after zero or more steps attack at_{21} (at node n_{21}). The sample sequence of Example 3 is in the cover of SP.

Besides support, other measures of interest can be devised to rank sequential patterns. The following may be specifically defined to the problem of risk assessment:

- *Length* $l(SP)$, the longer a pattern is the more information it provides on a (successful) strategy of a threat agent.
- *Score* $score(SP) = supp(SP) \cdot 2^{l(SP)-1}$, defined as a combination of support and length, where length is exponentially weighed.
- *Success distance* $success(SP)$. For a sequence S, the success distance is defined as the number of itemsets in S between the last one matching SP and the last in S. The success distance $success(SP)$ is the mean success distance over the sequences in the cover of SP. Intuitively, the smaller the success distance, the higher is the risk that an ongoing sequence matching the pattern will become successful (the agent reach its goal).
- *Right distance* $right(SP)$. For a sequence S, the right distance is defined as the number of new rights in S acquired between the last itemset matching SP and the last itemset in S. The right distance $right(SP)$ is the mean right distance over the sequences in the cover of SP. This is a refinement of success distance that considers the number of missing rights before the success instead of the number of steps before the agent is successful.

Example 6. The sequence of Example 3 contributes to the success distance of SP in Example 5 with a value of 3, because it attempts 3 more attacks after the last match with SP, namely after at_{21}.

4.3 Frequent, Closed, and Maximal Sequential Patterns

While a measure of interest can prompt useful patterns from an initial collection, the number of all possible sequential patterns is exponentially large (in the number of items and pattern length). It is then important to concentrate on specific collections of sequential patterns. The frequent pattern mining literature has proposed several collections and efficient algorithms to extract them from sequence databases.

Consider a fixed minimum support threshold $minsup > 0$. A sequential pattern SP is *frequent* if $supp(SP) \geq minsup$. We denote by $\mathcal{FP}$ the set of frequent sequential patterns. Since two frequent sequential patterns may actually

denote the same set of sequences, i.e., they have a same cover, restricting to patterns with distinct covers is a lossless strategy to reduce the number of extracted patterns and to avoid duplicate analyses. SP is *closed* if it is frequent and not strictly contained in a sequential pattern SP' with the same support, i.e., $supp(SP) \geq minsup \wedge \nexists SP'.\ (SP \sqsubset SP' \wedge supp(SP') = supp(SP))$. We denote by $\mathcal{CP}$ the set of closed sequential patterns. Closed sequential patterns are the longest sequential patterns in the class of equivalence of patterns with a same cover. A further condensed representation consists of restricting to the longest frequent sequential patterns only. SP is *maximal* if it is frequent and not strictly contained in another frequent sequential pattern SP' i.e., $supp(SP) \geq minsup \wedge \nexists SP'.\ (SP \sqsubset SP' \wedge supp(SP') \geq minsup)$. We denote by $\mathcal{MP}$ the set of closed sequential patterns. Maximal sequential patterns are closed, but the converse does not necessarily holds. In summary, $\mathcal{FP} \supseteq \mathcal{CP} \supseteq \mathcal{MP}$. For non-trivial sequence databases, the inclusion is strict.

Next lemma is referred to as the *anti-monotonicity* property of *cover* (or, equivalently, of *supp*). It is a trivial consequence of transitivity of relation $\sqsubseteq$.

Lemma 1. *If* $SP_1 \sqsubseteq SP_2$ *then* $cover(SP_1) \supseteq cover(SP_2)$.

5 The Covering Problem

5.1 Motivation

Sequential patterns provide a useful abstraction for understanding the strategies an agent adopts to reach its goal. Ranking patterns using one of the measures of interest from Sect. 4.2 prompts sub-sequences that occur with high frequency among the successful ones (using support as measure) that are characterized in higher detail (using length), or that are close to success (using success/right distance). In particular, the last ranking appears extremely useful in the design of *dynamic counter-measures*, such as blocking an agent that is following a pattern of successful attack before it reaches the goal. In this paper, however, we concentrate on a different approach, which tries and overcome the main limitation of (sequential) patterns, namely the *local view* each of them provides. A *global understanding* of the behavior of agents is missing. It is not obvious how to grasp a set of strategies that characterize the behavior of a threat agent[1]. Such a set provides a better understanding of how exposed is the target system to the attack strategies of the agent. Furthermore, it also supports what-if analyses and the selection of *static counter-measures*, such as network redesign and software patches. For instance, the integration of the Haruspex suite and sequential pattern mining in iteration generate sequence databases, characterize agent strategies, design counter-measures, and then repeat the process on the revised system.

[1] A further problem is to extract a set of strategies that, in addition, are also specific to a threat agent because they are not (often) used by the other agents. In this sense, they define a *signature* of the threat agent.

5.2 The Covering Problem

Let us extend the *cover* notation to sets of sequential patterns.

Definition 3. *For a set $\mathcal{C}$, we define:* $cover_{\mathcal{D}}(\mathcal{C}) = \cup_{SP \in \mathcal{C}} cover_{\mathcal{D}}(SP)$.

We say that $\mathcal{C}$ *covers* the sequence database $\mathcal{D}$ if every sequence in $\mathcal{D}$ is covered by some pattern in $\mathcal{C}$. The problem of finding a set $\mathcal{C}$ such that $cover_{\mathcal{D}}(\mathcal{C}) = \mathcal{D}$ is an instance of the well-known set cover problem [7]. Typically, however, one aims at finding a set $\mathcal{C}$ of a maximum size k (called *budget*). In our case study, for instance, domain experts may have to examine the set $\mathcal{C}$. Hence, it needs to be sufficiently small for human inspection. A cover of size k may not exist, hence one aims at finding a set $\mathcal{C}$ of size at most k which maximizes $|cover_{\mathcal{D}}(\mathcal{C})|$ – the number of covered sequences from $\mathcal{D}$. This is an instance of the maximum coverage problem [7]. As a final generalization, we assume that sequential patterns in $\mathcal{C}$ are constrained to belong to a pre-determined set $\mathcal{P}$, e.g., frequent, closed or maximal sequential patterns.

Definition 4. *Let $\mathcal{P}$ be a set of sequential patterns, and $k \geq 1$. We define the k-cover of $\mathcal{P}$:* $\mathbf{Cov}(k, \mathcal{P}) = argmax_{\mathcal{C} \subseteq \mathcal{P} \wedge |\mathcal{C}| \leq k} |cover(\mathcal{C})|$.

More than one subset $\mathcal{C} \subseteq \mathcal{P}$ can maximize the objective function. Thus, we write $\mathcal{C} \in \mathbf{Cov}(k, \mathcal{P})$ to denote any such subset.

Computing the k-cover is an NP-hard problem in general. A greedy algorithm chooses sequential patterns from $\mathcal{P}$ according to one rule: at each stage, choose the one whose cover contains the largest number of sequences uncovered by the already chosen elements. Such a greedy algorithm achieves an approximation ratio of ≈ 0.632 w.r.t. the optimal cover [7].

We will now formally show that frequent $\mathcal{FP}$ and closed $\mathcal{CP}$ sequential patterns are not suitable choices for $\mathcal{P}$ in the computation of a cover, while maximal $\mathcal{MP}$ sequential patterns are. First, we introduce a notation for the minimal elements of a set of patterns w.r.t. the $\sqsubseteq$ relation.

Definition 5. $Min(\mathcal{P}) = \{SP \in \mathcal{P} \mid \nexists SP' \in \mathcal{P}.\ SP' \sqsubset SP\}$.

The next key result shows that when looking for the k-cover of $\mathcal{P}$, one can restrict to select only minimal elements from $\mathcal{P}$.

Theorem 1. *Let $\mathcal{C} \in \mathbf{Cov}(k, \mathcal{P})$. There exists $\mathcal{C}' \in \mathbf{Cov}(k, Min(\mathcal{P}))$ such that $cover(\mathcal{C}) = cover(\mathcal{C}')$ and $|\mathcal{C}| \geq |\mathcal{C}'|$.*

Proof. We define the set $\mathcal{C}'$ as follows. For every $SP \in \mathcal{C}$ choose any $SP' \in Min(\mathcal{P})$ such that $SP' \sqsubseteq SP$. For a given SP, at least one such a SP' exists – it could be SP itself, if it is minimal in $\mathcal{P}$. Any of such SP''s can be chosen. By construction $|\mathcal{C}'| \leq |\mathcal{C}| \leq k$ (in fact, a same minimal element can be chosen for two SP's in $\mathcal{C}$). By Lemma 1, $cover(SP) \subseteq cover(SP')$. As a consequence:

$$cover(\mathcal{C}) = \cup_{SP \in \mathcal{C}} cover(SP) \subseteq \cup_{SP' \in \mathcal{C}'} cover(SP') = cover(\mathcal{C}').$$

Since $\mathcal{C}$ is a k-cover and $\mathcal{P} \supseteq Min(\mathcal{P})$, then $|cover(\mathcal{C})|$ is maximal, which implies that the inclusion above is an equality. Finally, observe that a k-cover from

$\mathbf{Cov}(k, Min(\mathcal{P}))$ cannot cover more sequences than a k-cover of $\mathcal{P}$, by definition of maximization. Since $\mathcal{C}' \subseteq Min(\mathcal{P})$ and $|\mathcal{C}'| \leq k$ and $|cover(\mathcal{C}')|$ is maximal, we conclude $\mathcal{C}' \in \mathbf{Cov}(k, Min(\mathcal{P}))$. □

The consequence of this result is considerable. It makes no sense to try and cover a database of sequences using frequent patterns or closed patterns.

Example 7. It is readily checked that the minimal element of frequent sequential patterns $\mathcal{FP}$ is the empty sequence, namely $Min(\mathcal{FP}) = \{\langle\rangle\}$. Trivially, $\{\langle\rangle\}$ is a cover of any sequence database. By Theorem 1, $\{\langle\rangle\} \in \mathbf{Cov}(k, \mathcal{FP})$. Unfortunately, the empty sequential pattern is of no practical help in any application.

When not considering the empty sequential pattern, the result is not interesting, because sequential patterns are not needed at all when looking for a cover. As an example, a standard approach for entity selection can apply the greedy algorithm of the maximum coverage problem [7] to select items (and not sequential patterns) that have the largest residual frequency in the yet uncovered sequences.

Example 8. Consider non-empty frequent sequential patterns: $\mathcal{FP}' = \mathcal{FP} \setminus \{\langle\rangle\}$. We have that $Min(\mathcal{FP}') = \{FP \in \mathcal{FP} \mid \exists i \in I.\ FP = \langle\{i\}\rangle\}$. Then, to find a cover, we can consider only sequential patterns with one frequent item. In other words, there would be no need of extracting sequential patterns at all.

For closed sequential patterns $\mathcal{CP}$, one can reason as in the above example, and reach similar conclusions. To find a way out, one may consider $\mathbf{Cov}(k, \mathcal{P})$ for $\mathcal{P}$ being a set of frequent or closed sequential patterns with support in a range $[minsup, maxsup]$. However, which maximal threshold $maxsup$ to choose remains unclear. We propose, instead, to use maximal sequential patterns $\mathcal{MP}$, for two reasons. First, minimal elements of maximal patterns are themselves:

$$Min(\mathcal{MP}) = \mathcal{MP}.$$

The drawbacks highlighted for frequent sequential patterns are then overcome. Second, albeit a k-cover $\mathcal{C} \subseteq \mathcal{MP}$ is not a partition of $\mathcal{D}$, for any two patterns in $\mathcal{C}$ the number of sequences shared by them can be upper-bounded. This is a useful property when a business action has to be done for each sequential pattern in a cover, but sequences should not be subject to several business actions. In our case study, in particular, we aim at finding a cover consisting of sequential patterns which abstract alternative attack strategies.

Lemma 2. *Let $SP_1, SP_2 \in \mathcal{MP}$ with $SP_1 \neq SP_2$. We have: $|cover(SP_1) \cap cover(SP_2)| < 3 \cdot minsup - 2$.*

Proof. Let SP be the maximal initial sub-sequence shared by SP_1 and SP_2. Since $SP_1 \neq SP_2$, at least one between SP_1 and SP_2 is longer than SP. Since SP_1 and SP_2 are maximal, both SP_1 and SP_2 are longer than SP (otherwise, one of them would be contained in the other). Thus, let $SP_1 = SP \cdot \langle A\rangle \cdot \hat{SP_1}$ and $SP_1 = SP \cdot \langle B\rangle \cdot \hat{SP_2}$, where $\cdot$ is the sequence appending operator, and $A \neq B$.

Any $S \in cover(SP_1) \cap cover(SP_2)$ either satisfies: (1) $SP \cdot \langle A, B \rangle \cdot \hat{SP}_1 \sqsubseteq S$; or (2) $SP \cdot \langle B, A \rangle \cdot \hat{SP}_1 \sqsubseteq S$; or (3) $SP \cdot \langle A \cup B \rangle \cdot \hat{SP}_1 \sqsubseteq S$. Assume now, by absurd, that $|cover(SP_1) \cap cover(SP_2)| \geq 3 \cdot minsup - 2$. Then, the number of sequences that satisfy (1) (resp., (2) or (3)) are at least one third of $3 \cdot minsup - 2$, i.e., at least $minsup$. Therefore, SP_1 cannot be maximal. □

A better bound can be stated when the sequences are made of singleton itemsets, as in the case of scenario S1 (see Definition 1).

Lemma 3. *Assume a sequence database where itemsets are singletons. Let* SP_1, $SP_2 \in \mathcal{MP}$ *with* $SP_1 \neq SP_2$. *We have* $|cover(SP_1) \cap cover(SP_2)| < 2 \cdot minsup - 1$.

Proof. By assumption, the case (3) in the proof of Lemma 2 cannot occur. □

6 Covering Attack Sequences of the Case Study

Recall that the input dataset in our case study consists of 600 K successful attack sequences, 100K for each of the six threat agents (see Table 2). The empirical distribution of sequence lengths is reported in Fig. 2 (left). It is a mixture of two distributions. Sequences of threat agents A1 and A2 have length 12 or 13. Sequence lengths of the other agents, instead, are well approximated by Gaussian distribution with mean 32.3. Figure 2 (right) shows the empirical cumulative distribution of the rank of attacks. Let $rk(at)$ be the rank of attack at, with 1 being the rank of the most frequent attack, 2 of the second most frequent, etc. The cumulative distribution CDF($rk(at)$) is the probability that an attack occurrence in the database belongs to one of the top $rk(at)$ most frequent attacks. It is well-fitted by a cumulative exponential distribution $1 - e^{-\lambda \cdot rk(at)}$ with $\lambda = 0.0213$. A naïve approach that tries and patches the top most frequent attacks, has to deploy 33 patches to cover half of the occurrences of attacks in the sequence database, and 108 to cover 90% of the occurrences. Such an approach is naïve because sequences include multiple attacks, hence after patching the vulnerability that enables an attack, all the sequences containing such an attack does not need further patches. Therefore, a covering approach should be considered, e.g., by applying a greedy algorithm that selects an attack on the basis of residual support (number of covered sequences among those not covered by previously selected attacks). Covering sequences using elementary attacks, however, does not result in an effective approach. In fact, the last attack of any sequence is a specific one on the target node n_f. Thus, such an attack alone will cover *all* sequences. But, since it occurs as the last attack: (1) first, it cannot enable dynamic counter-measures, because after the attack has been attempted the goal of the agent is already reached; (2) second, it gives no hint on the possible strategies of the attacker for reaching the final node.

The approach followed in this paper is instead more refined. We search for a covering using sequential patterns rather than elementary attacks. Consider the scenario S1. For each threat agent, we extract the maximal sequential patterns from the 100K attack sequence of the agent, assuming $minsup = 5K$, namely 5%

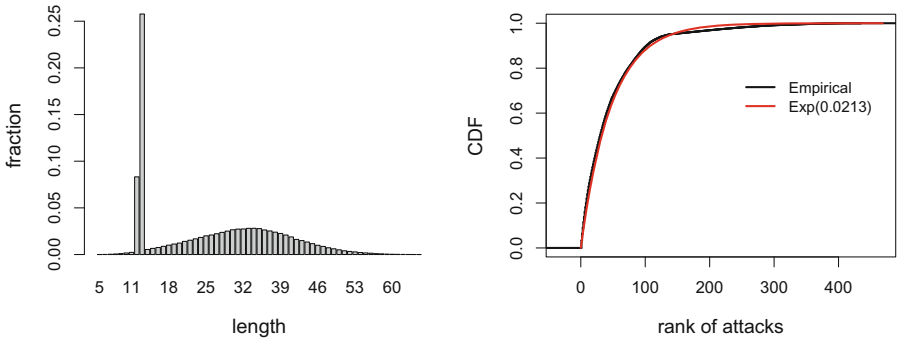

Fig. 2. Left: distribution of sequence lengths. Right: CDF of attack ranks.

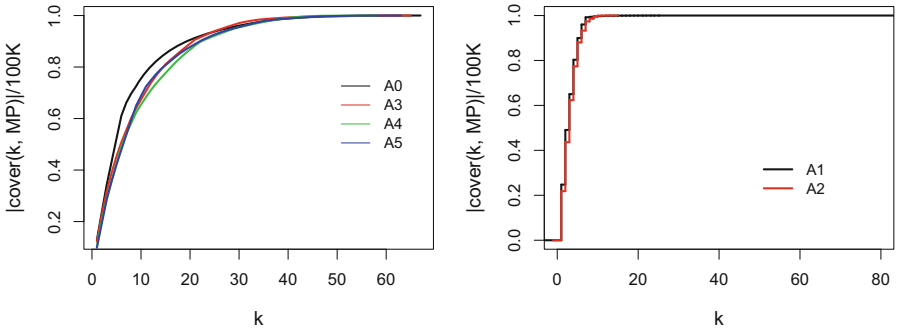

Fig. 3. Covering sequences. Left: Scenario S1. Right: Scenario S2.

relative minimum support threshold. Starting from the set of maximal sequential pattern, we compute a k-cover using the greedy algorithm. Figure 3 (left) shows the fraction of the 100K sequences that are covered by $cover(k, \mathcal{MP})$ at the variation of k for threat agents A0, A3–A5. In all cases, the top 20 sequential patterns do provide a covering of more than 85% of the attack sequences. The mean length of the top 20 sequential patterns is 3.75 for A0, 4.67 for A3, 3.6 for A4, and 3.7 for A5 (all patterns include the final node n_f). This means the k-cover provides some useful information on the strategies adopted by the threat agent. Let us consider as an example, M0. The top 7 sequential patterns in the result of the greedy algorithm cover 66% of the sequences. One specific elementary attack occurs in 4 of them, and another in the other 3 of them. Patching such two attacks will then result in an effective means for blocking 66% of the successful attack sequences. This shows that the discovery of patterns strongly reduces the number of patches to deploy without decreasing the effectiveness of the patching strategy.

Let us consider now the scenario S2. Figure 3 (right) shows the fraction of the 100K sequences that are covered by $cover(k, \mathcal{MP})$ at the variation of k for threat agents A1 and A2 and with $minsup = 15K$. The covered fraction

grows rapidly with k. However, such sequential patterns cannot be used for designing countermeasures, since they include attack types or attacked nodes, without specifying the specific elementary attacks that are used (and possibly covering different elementary attacks at the same node). They are instead useful in evaluating the "degree of freedom" of a threat agent on the target system. In particular, the sequential patterns highlight the (maximal) paths that an agent may follow to reach the goal. The more sequential patterns are needed to reach a certain fraction of covered sequences, the more complex is the set of strategies/paths the agent can implement/follow – and, ultimately, the more complex is to stop the attacker. At one extreme, a topology with a single path would produce a single maximal sequential pattern.

7 Related Work

Sequential patterns, measures of interest, and their generalization have been studied for more than 20 years in the data mining literature [6,12,14], with applications to the most diverse areas. Several sequential, parallel, and distributed algorithms for extracting different classes of sequential patterns have been proposed. We adopted the SPMF Java library [5], which includes open-source implementations of a large number of them. Most of the attention on interest measures has considered ranking individual sequential patterns, e.g., on the basis of unexpectedness. Notable exceptions consider sets of sequential patterns and adopt the Minimum Description Length principle [10,18].

The maximum coverage problem and its approximation algorithms have been considered in many application contexts and variants [7]. The approach most related to ours is the computation of Coverage Patterns, a class of (non-sequential) itemsets X, such that the items in X cover a specified percentage of transactions and they overlap at most for a specified threshold [17].

Finally, the usage of data mining models in IDS has been widely adopted, e.g., using classifiers [11], association rules [9], and closed sequential patterns [4]. All of them struggle with the lack of enough data for building accurate models.

8 Conclusions

We have proposed a novel data-driven approach, based on sequential pattern mining, in ICT risk assessment and prevention that offers potentials at design-time. Starting from a sequence database generated through Monte Carlo simulations, we formally showed that maximal sequential patterns are the most appropriate classes for such a context. On a sample case study, we have shown that a maximal k-cover based on sequential patterns can provide an abstraction which is useful both for designing countermeasures and for providing to the security analyst an abstraction of the strategies of attackers on the target system.

References

1. Baiardi, F., Corò, F., Tonelli, F., Sgandurra, D.: A scenario method to automatically assess ICT risk. In: Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2014), pp. 544–551. IEEE (2014)
2. Baiardi, F., Telmon, C., Sgandurra, D.: Haruspex: simulation-driven risk analysis for complex systems. ISACA J. **3**, 46–51 (2012)
3. Baiardi, F., Tonelli, F., Bertolini, A.: CyVar: extending Var-At-Risk to ICT. In: Seehusen, F., Felderer, M., Großmann, J., Wendland, M.-F. (eds.) RISK 2015. LNCS, vol. 9488, pp. 49–62. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26416-5_4
4. Brahmi, H., Yahia, S.B.: Discovering multi-stage attacks using closed multi-dimensional sequential pattern mining. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013. LNCS, vol. 8056, pp. 450–457. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40173-2_38
5. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C., Tseng, V.S.: SPMF: a Java open-source pattern mining library. J. Mach. Learn. Res. **15**, 3389–3393 (2014)
6. Fournier-Viger, P., Lin, J.C.-W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Sci. Pattern Recogn. **1**, 54–77 (2017)
7. Hochbaum, D.S.: Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard Problems, pp. 94–143. PWS Publishing Co. (1997)
8. Joint Task Force Transformation Initiative Interagency Working Group. SP 800–30 revision 1: Guide for conducting risk assessments. National Institute of Standards & Technology (2012)
9. Katipally, R., Gasior, W., Cui, X., Yang, L.: Multistage attack detection system for network administrators using data mining. In: Proceedings of the Cyber Security and Information Intelligence Research Workshop (CSIIRW 2010), pp. 51. ACM (2010)
10. Lam, H.T., Mörchen, F., Fradkin, D., Calders, T.: Mining compressing sequential patterns. Stat. Anal. Data Min. **7**(1), 34–52 (2014)
11. Lee, W., Stolfo, S.J., Mok, K.W.: Adaptive intrusion detection: a data mining approach. Artif. Intell. Rev. **14**(6), 533–567 (2000)
12. Mabroukeh, N.R., Ezeife, C.I.: A taxonomy of sequential pattern mining algorithms. ACM Comput. Surv. **43**(1), 3:1–3:41 (2010)
13. MITRE: Common Weakness Enumeration. https://cwe.mitre.org/
14. Mooney, C., Roddick, J.F.: Sequential pattern mining - approaches and algorithms. ACM Comput. Surv. **45**(2), 19:1–19:39 (2013)
15. NIST: National Vulnerability Database. https://nvd.nist.gov/
16. Schiffman, M.: Common Vulnerability Scoring System. https://www.first.org/cvss
17. Srinivas, P.G., Reddy, P.K., Trinath, A.V., Sripada, B., Kiran, R.U.: Mining coverage patterns from transactional databases. J. Intell. Inf. Syst. **45**(3), 423–439 (2015)
18. Tatti, N., Vreeken, J.: The long and the short of it: summarising event sequences with serial episodes. In: Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD 2012), pp. 462–470. ACM (2012)