Evolving Secure Information Systems through Attack Simulation

Elmar Kiesling
Vienna University of Technology
elmar.kiesling@tuwien.ac.at

Andreas Ekelhart
Secure Business Austria
aekelhart@sba-research.org

Bernhard Grill
Secure Business Austria
bgrill@sba-research.org

Christian Stummer
Bielefeld University
christian.stummer@uni-bielefeld.de

Christine Strauss
University of Vienna
christine.strauss@univie.ac.at

Abstract

In this paper, we introduce *a simulation-based, evolutionary approach for analyzing and improving the security of complex information systems. Rather than following a purely technical approach, we bring in a social and behavioral perspective through a combination of conceptual security knowledge modeling, behavioral modeling of threat agents, simulation of attacks, and evolutionary optimization.*

Based on results from numerous attack simulations for various internal and external attackers, metrics such as impact on confidentiality, availability, and integrity of the simulated attacks are monitored and efficient sets of security controls with respect to multiple risk, cost and benefit objectives are determined. We describe the developed approach as well as a prototypical implementation and demonstrate its applicability by means of an illustrative example.

1. Introduction

In the face of a rapidly evolving threat landscape, assuring the security of complex information systems is a major challenge. In recent years, the frequency and sophistication of attacks by motivated adversaries have increased dramatically. Unlike malware designed to automatically and opportunistically exploit particular technical weaknesses, these goal-driven attacks are aimed at particular targets. They combine technical and social attack techniques to exploit complex interactions and leverage architectural weaknesses in order to achieve particular objectives. Hence, understanding adversaries' motivations, capabilities, resources, available attack vectors, and objectives is paramount, because these factors determine their attack campaign and, ultimately, the risk attackers pose to an information system.

Isolated technical measures, added on top of an existing system, are typically insufficient to establish an adequate defense and are hence inefficient from a risk management perspective. Rather, an integrated approach is needed in order to recognize that adversaries may combine various technical and non-technical means to achieve their goals (e.g., network, software, physical, and social attacks). Decomposing a security analysis problem and evaluating the security of individual components or the effectiveness of particular security controls is a myopic strategy in this context. Our approach reflects this idea in that we evaluate the security of complete systems by dynamically simulating attacks. Hence, we conceive security as an emergent property of complex information systems.

The main contributions of this paper are as follow: We formally model abstract attack patterns and develop mechanisms to link them to determine possible routes of attacks. Whereas existing approaches typically aim to build attack graphs by enumerating the entire set of possible paths, we conceive the discovery of these paths as a dynamic, adversary-driven process. We model adversaries as agents that make deliberate decisions in attacking a system while actively exploiting dynamic interactions of vulnerabilities. Finally, we develop a discrete event attack simulation engine, and apply evolutionary algorithms to optimize sets of security controls while trading off multiple objectives.

2. Framework overview

Figure 1 provides a high-level architectural overview of the simulation-optimization framework. It consists of a knowledge base (Section 3) that formally specifies attack mechanisms and security controls as well as the system to be protected. A threat scenario (Section 4) describes adversaries and their objectives. Based on these inputs, our model can link attack

978-1-4799-2504-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HICSS.2014.597

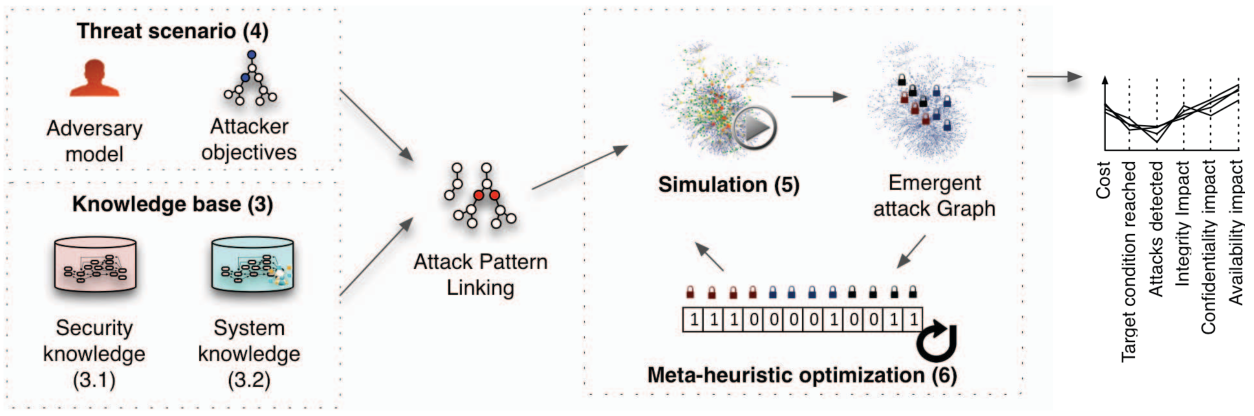

Figure 1. Framework overview

patterns dynamically in order to simulate attacks (Section 5). This simulation core facilitates evaluation of system configurations with various sets of security controls in place. Results can be analyzed with respect to multiple risk, cost, and benefit criteria.

Automatically identifying efficient sets of security controls is computationally difficult. To tackle this challenge, we therefore resort to meta-heuristic optimization techniques (Section 6). Optimization results finally allow decision-makers to explore efficient solutions, trade off conflicting objectives, and choose a satisfying solution.

We illustrate the application of the methodology with an example in Section 7.

3. Knowledge base

The knowledge base (KB) captures knowledge required to simulate attacks in a well-structured and reusable format. Our implementation uses the declarative logic programming language Prolog to model security and system knowledge and reason on possible routes of attacks.

3.1. Security knowledge

The attack and control model formally specifies how systems may be attacked and secured by means of *attack patterns*. Each pattern is applicable in a particular context under specific pre-conditions. We take advantage of the existing, publicly available CAPEC (Common attack pattern enumeration and classification) repository [1], which currently contains 400 community-developed attack patterns for exploiting software systems. These patterns derive from the concept of design patterns, but are specified in a destructive context with the intention to support the community to understand the attacker's perspective [2],[3].

The information is mostly semi-structured and hence needs to be translated into a more formal representation to be useful for attack simulations. To this end, we add formal descriptions and define properties that are used to link attack patterns based on the CAPEC sections *Summary*, *Experiments* (attack steps), *Outcomes* (success and failure results), and *Attack Prerequisites*. Another attribute obtained from CAPEC is the base success probabilities of each action, which we derived from *Typical Likelihood of Exploit*. Based on this information, we formulate actions as clauses with a set of pre-conditions in the body:

```
action_sqlInjection(Attacker, DbServer) :-
    technicalSkillLevel(Attacker, TechnicalSkillLevel),
    TechnicalSkillLevel >= 1,
    owned(Attacker, AttackHost),
    connected(AttackHost, WebServer, httpProtocol,
    httpPort),
    connected(WebServer, DbServer, dbProtocol, dbPort),
    usesDbSoftware(WebServer, DbServer),
    not(owned(Attacker, DbServer)).
```

This example predicate defines an SQL injection attack and specifies its pre-conditions. It matches all variable bindings for *Attacker* and *DbServer* that satisfy the pattern, based on the current facts in the KB.

To define possible outcomes of an attack action, we define post-conditions, i.e., state transitions that occur when the simulation is executed. Basic outcome types are *successful* and *failed* attack actions. Successful execution of an attack action can, for example, allow the attacker to gain root access to the target computer, while a failed attempt could take the target computer offline. Formally, we define both types as rules that assert a change in the KB:

```
exec_success_action_sqlInjection(Attacker, DbServer) :-
        assert(owned(Attacker, DBServer)).
```

Invoking this rule with valid bindings from the previous query asserts the fact that the attacker now has access to the DB server, which in turn affects the results of subsequent queries.

CAPEC also includes a section *CIA Impact*, which specifies the impact of an attack action in terms of loss or degradation of confidentiality, integrity, and availability on a scale of high, medium, and low. This is valuable general information, but it does not consider the context in which the impact occurs. Because we explicitly simulate attacks on a particular system, we determine impact not on an attack pattern level, but rather based on the affected asset at simulation runtime (cf. Section 5.3). The attack KB only specifies the security attributes affected by attack actions and the rule that determines how the impact level will be queried:

```
action_impact(action_sqlInjection, confidentiality).
impact_success_sqlInjection(Attacker, TargetHost,
Impact):-
    importance(TargetHost, confidentiality, Impact).
```

This example specifies that *SQL injection* affects the security attribute *confidentiality*. When an attacker successfully carries out this action, the confidentiality criticality rating of the attacked *TargetHost* will determine the impact rating.

We further enrich the security knowledge with control definitions derived from CAPEC. The sections *Solutions*, *Mitigations* and *Relevant Security Requirements* provide the necessary information. We distinguish preventive and detective controls that can be applied to particular system elements to increase the difficulty of attacks or facilitate the detection of ongoing attacks, respectively. Formally, we add controls to the KB as predefined facts. The associated properties determine (i) if the control type is *detective* or *preventive*, (ii) whether the control is visible to an attacker, (iii) the outcome of the control (*stop* or *null*), (iv) the aggregation type that determines the effect of multiple controls on an asset (*min*, *max*, *cumulate*), (v) the control response type (*immediate* or *delayed*), (vi) the detection delay, (vii) the type of assets that implement this control, and (viii) the type of assets that this control can be applied to. For example, an intrusion detection system's basic properties are defined as follows:

```
control_properties(control_ids, detective, false, null, max,
delayed, 1000, ids, hostGroup).
```

Finally, particular control instances and their cost and effectiveness in the context of an action can be specified as follows:

```
control_effectiveness(ids1, action_sqlInjection, 0.8).
ids(ids1, 7500).
```

3.2. System knowledge

To reason about the security of a particular information system, it is necessary to model its constituent components and establish relations between them. To this end, we model specific knowledge about the system to be protected, but separate it from the abstract attack knowledge. This approach facilitates sharing and reuse of domain-specific attack knowledge by multiple organizations facing similar threats. The system knowledge is organization-specific, and hence, kept in a separate file.

This model stores a set of tangible and intangible assets, including hardware components, networks, data, employees, policies etc. To illustrate how an asset is modeled and controls are associated, consider the example of an intrusion detection control. First, an instance of *host* is added to the system model. Second, the *installed* relation is used to link a candidate intrusion detection software asset to the host. Third, we can state that *dbServers_host_1* stores the project database *projectDb1*, and that the server is part of the host group *dbServerHosts*. *workstationHosts* can access the *dbServerHosts* via HTTP.

```
host(dbServers_host_1).
installed(dbServers_host_1, ids1).
stores(dbServers_host_1, projectDb1).
inHostGroup(dbServers_host_1, dbServerHosts).
hacl(workstationHosts, dbServerHosts, httpProtocol,
httpPort).
```

To rate the impact of threats, modeled assets can be assigned criticality values for multiple security attributes. The values can be obtained from existing impact analyses or asset criticality assessment reports. If this documentation does not exist or such assessments have not been performed yet, the system owners are responsible for determining the ratings for their own systems and information [4].

Criticality values can, for example, represent monetary values as well as categories of a qualitative or continuous scale (e.g., a three-point Likert scale, 1 = low, 2 = medium, 3 = high).

To complete the previous example, we define criticality values for the *projectDb1*.

```
    criticality(projectDb1, confidentiality,2).
    criticality(projectDb1, integrity, 3).
    criticality(projectDb1, availability, 2).
```

4. Threat scenario

An appropriate adversary model is required to understand risks and identify effective controls. Whereas adversaries are typically classified based on a natural language description in the existing literature [5], we take advantage of our formal model to define more specific attacker profiles that include capabilities, resources (e.g., time), risk preferences, and other behavioral attributes (*risk aversion*, *propensity to alternate between different attack strategies* etc.). These attributes determine whether particular advanced attack patterns are available, affect the choice of attack paths, and determine success probabilities in the simulation. We also specify initial access where appropriate (an *employee* may, for example, have access privileges on *workstationHosts* and be a member of the *workstationUserGroup*).

A complete threat scenario consists of an adversary model and an objective specification, stated as a target condition that describes a desired system state (e.g., *attacker* has access to *db2*).

5. Attack simulation

The attack simulation leverages and combines the knowledge embodied in the security and system KB by executing attacks for given threat scenarios.

Adversaries' choices regarding their course of action, the outcome of individual attack actions, and the detection of attacks are determined probabilistically in the simulation. It is hence necessary to perform multiple replications with varying sets of random seeds for each attack scenario to capture variability and uncertainty. For each replication, the simulation executes a schedule of discrete events and records outcome variables that can be analyzed and aggregated. This flexible framework can capture complex causal relations and timing interactions. The types of events used in the simulation are illustrated in Figure 2. *Action Selection* events schedule *Action Start e*vents based on available actions determined through pattern linking as wells as the adversary's behavioral model. Upon execution, these events determine the effective duration of the action (which is influenced by factors such as difficulty, adversary skills, and preventive controls etc.) and schedule an *Action End* event.

Detection events may be triggered when assets associated with a detective control are being attacked; an *Attacker Stopped* event may be scheduled by terminating detective controls. Finally, a *Target Condition Reached* event is scheduled in case the attacker has reached the target condition.

The mechanisms invoked through these events, i.e., pattern linking, action selection, action execution, and detective control response, are explained in the following sections.

5.1. Attack pattern linking

Attack patterns are linked automatically based on shared pre- and post-condition properties. This approach is related to the attack graph concept introduced in [6], particularly to advanced formalisms and methods to construct large trees [7],[8],[9], as well as extensions that incorporate security controls to obtain defense and protection trees [10],[11]. However, these existing approaches typically enumerate the entire set of possible paths to build a complete attack graph, which is problematic due to the exponential size of the search space. Advanced formalisms for dynamic security simulations, such as Generalized Stochastic Petri Nets [12] and Boolean Logic Driven Markov Processes (BDMP) [13] aim to alleviate this problem. Our approach conceives the discovery of attack paths as a dynamic process and is therefore more general and flexible in several respects (e.g., it is not necessary to assume "monotonicity", i.e., that the attacker can never loose previously gained privileges or that the negation operator cannot be used to express pre-conditions). In this respect, we follow some ideas for state-space modeling and simulation from the literature [14].

5.2. Attack action selection

We associate adversaries with a behavioral model that iteratively selects attack actions based on (i) individual adversary characteristics, (ii) the attacker's general knowledge about possible routes of attack, and (iii) the outcomes of prior attack actions. In our illustrative application example discussed in Section 7, we model five types of internal and external adversaries that differ in their characteristics.

Our behavioral model is based on the idea that adversaries possess general security knowledge (to varying degree), but typically do not have complete a-priori information about the system they attack. The general knowledge about possible routes of attack is represented via an abstract attack graph that can be queried from the KB for a particular attacker and target condition. This graph may be interpreted as an adversary's mental map of how actions can be combined to achieve a particular outcome.

The simulation applies this abstract knowledge by querying the KB for valid asset assignments for all pre-conditions of an abstract action in the context of the modeled system. This yields a set of action instances (i.e., abstract actions with assigned variables) that can be executed against particular assets.

We denote abstract actions by a and concrete action instances by $\bar{a}$. The behavioral model specified in Algorithm 1 stipulates that adversaries alternate between following a chosen attack path (i.e., a sequence of abstract attack actions) and trying new approaches (i.e., choosing new entry points and attack paths). We assume that adversaries minimize expected effort, i.e., they tend to launch attacks "close" to the target if possible. This idea is implemented in the $choose(\bar{A})$ procedure, which chooses among pre-selected sets of action instances. It calculates the shortest path distances $d(a,t)$ in terms of cumulative effort required between each abstract attack action a and the target condition t using Dijkstra's algorithm [15]. Using the longest abstract distance found as a reference, it calculates the relative distance for all candidate actions based on their position in the abstract graph.

The procedure then calculates individual weights $w_{\bar{a}}$ for each action instance $\bar{a}$ based on $\bar{a}$'s success and detection probabilities $p_{suc}(\bar{a})$ and $p_{det}(\bar{a})$, the relative distance $d_{rel}\,(\bar{a})$ to the target condition, and the attacker's preferences w_{suc}, w_{det}, and w_{dist}.

Procedure $weightedChoice(\bar{A}, W)$ then determines the final choice from the candidate action instances with probabilities proportional to the calculated weights W.

After the initial action has been selected, subsequent choices are dependent upon results of the previously executed action p. If the previous action was successful and new action instances $\bar{N}$ have become available as a consequence, the adversary will choose among them with probability $p_{continueNew}$ and choose among all available actions otherwise. If the previous action was successful, but did not yield any new action instances, then there are two options. With probability $p_{alternativesNoNew}$, the adversary will choose among action instances that have become available as a consequence of the same action as $\bar{p}$, i.e., those returned by $getAlternatives(\bar{a})$. Otherwise, the adversary will search for a new entry point by selecting from all available actions

In case the previous action $\bar{p}$ was not successful, the adversary will retry with probability p_{retry}. Otherwise, it will either choose a "neighboring" action instance (with probability $p_{alternativesFailed}$) or search for a new entry point among all available actions.

At each step during an attack, the adversary will chose available action instances directly if their outcome fulfills the target condition.

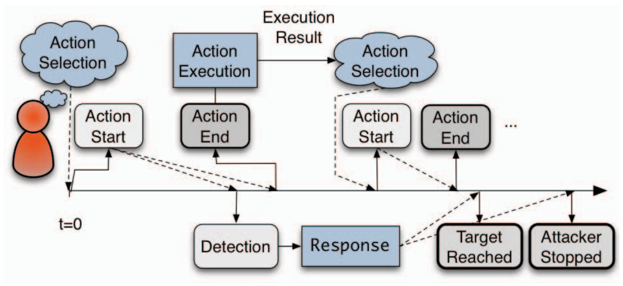

Figure 2. Event types and scheduling

```
**Input:**    previous action $\bar{p}$,
              available actions $\bar{A}$,
              new actions $\bar{N}$
**Output:**   selected action

$selectAction(\bar{p}, \bar{A}, \bar{N})$
    **for** $a \in \bar{A}$
        **if** $fullfillsTargetCond\ (\bar{a})$ **then**
            **return** $\bar{a}$
        **endif**
    **endfor**

    **if** $p\ =\ null$ **then**
        **return** $choice(\bar{A})$
    **else if** $wasSuccessful(\bar{p})$
        **if** $|N|\ >\ 0$ **then**
            **if** $p_{continueNew}\ >\ random()$ **then**
                $choose(\bar{N})$
            **else**
                **return** $choose(\bar{A})$
            **endif**
        **else if** $p_{altNoNew}\ >\ random()$ **then**
            $\boldsymbol{\bar{S}\ \leftarrow\ getAlternatives(p)}$
            **return** $choose(\bar{S})$
        **else**
            **return** $choose(\bar{A})$
        **endif**
    **else if** $p_{retry} > random()$
        **return** $p$
    **else if** $p_{\{altFailed\}} > random()$ **then**
        $\bar{S} \leftarrow getAlternatives(p)$
        **return** $choose(\bar{S})$
    **else**
        **return** $choose(\bar{A})$
    **endif**


$choose(\bar{A})$
**for** $\bar{a}$ **do**
    $d_{\bar{a}}^{rel} \leftarrow \frac{d(a,t)}{\max(d(a,t))+1}$
    $W_{\bar{a}} \leftarrow p_{suc}(\bar{a})^{w_{suc}}\left(1 -\ p_{det}(\bar{a})\right)^{w_{det}}\left(1 - d_{\bar{a}}^{rel}\right)^{w_{dist}}$
**end for**

**return** $weightedChoice(\bar{A}, W)$
```

Algorithm 1. Attacker behavior

5.3. Attack action execution

Upon execution, we determine (i) whether the attack action was successful, (ii) whether the attack target has been reached, (iii) what actions become unavailable as a consequence, (iv) what new actions are available, and (v) the impact on security attributes (e.g., confidentiality, integrity, availability). The impact severity of an executed action is dependent upon the execution context. The attack patterns stored in the security KB specify the attributes affected by an action (e.g., *action_sqlInjection* affects *confidentiality*), but not the severity of an impact. The latter is obtained by querying asset valuation with respect to the security attribute from the system KB. The impacts on security attributes are registered and can be analyzed and used for optimization purposes. The result of an action then drives the selection of subsequent actions.

5.4. Detective control response

Detective controls define two possible outcomes: (i) *Stop*, i.e., the simulation run terminates, or (ii) a defined path to a post-condition (modeled in the abstract attack graph) to execute. These post-conditions can change property values in the system model. For example, when an intrusion detection system blocks the adversary's source IP address it inhibits certain attacks but potentially also facilitates new attack patterns (e.g., IPS reaction leads to service disruption and causes an availability impact).

6. Meta-heuristic optimization

Decision-makers can obtain valuable insights by iteratively modeling a system with a particular set of security controls in place and running the simulation to assess their joint effectiveness. Discovering interactions between individual security controls and optimizing the overall configuration by altering the system manually until satisfactory results can be achieved is, however, a tedious and difficult process and infeasible for larger problem sizes.

We therefore introduce biologically inspired optimization techniques and the concept of a *control portfolio* characterized by a *genotype* string of binary indicator variables. Each of these variables represents a control-asset pairing (e.g., *logPolicy1* implemented on *dbServerHosts*) that takes the value 1 if the respective control is applied to the asset and 0 otherwise.

To evaluate a control portfolio, we initialize the system by applying controls to assets according to its genotype. We then simulate a number of attacks with varying random seeds and record outcome metrics. These metrics can be aggregated across simulation runs (e.g., *sum*, *min*, *max*, *average*, *median*) and selected as optimization objectives.

Because we consider several objectives simultaneously, the optimization does not generally result in a single "best" control portfolio, but typically in a set of *(Pareto-) efficient* portfolios. Each proposed efficient portfolio is non-dominated, i.e., there is no other portfolio with equal or better values for all objectives and a strictly better value for one of the objectives.

Due the huge combinatorial decision space (with 2^n potential portfolios, where n is the number of control-asset combinations), the expensive simulation-based evaluation procedure, and the need to account for multiple optimization objectives, this simulation-optimization problem is highly challenging computation-wise. Exact solutions (i.e., complete sets of all efficient portfolios for a given number of simulation replications) can only be determined through complete enumeration for rather small problem instances. In order to tackle larger problem instances we resort to meta-heuristic solution procedures.

In particular, we experimented with genetic algorithms, i.e., population-based approaches that are inspired by nature and optimize problems implicitly. These algorithms evolve a population of individuals (control portfolios) iteratively by evaluating their fitness (assessing objectives), selecting fit individuals (control portfolios), and performing crossover and mutation operations on the selected individuals to generate offspring. The binary genotype strings generated in this process are evaluated by means of a number of simulation replications, which yield aggregate objective values used for assessing the "fitness" of the respective control portfolio.

We conducted experiments with the NSGA-II [16] and SPEA-II [17] genetic algorithms and evaluated solution quality and convergence characteristics for small problem instances (search space 2^{12}) with respect to results obtained by complete enumeration. We found that both algorithms performed reasonably well. Using default parameters, NSGA-II performed slightly better and on average identified approximately 65% of all efficient solutions within covering the first 20% of the search space. By adding the all-zero and all-one genotype strings to the otherwise randomly generated initial population, applying a two-point rather than the single-point crossover operator, and using an adaptive mixed mutation strategy that randomly selects between insert/revert/swap operations, we could improve NSGA-II performance for the given problem significantly. For our illustrative application discussed in the following section, we observed a convergence toward the full set of efficient solutions typically within the first 250 generations.

7. Application example

7.1. Implementation

The attack simulation and optimization components are implemented in Java. We use the scheduling mechanisms provided by MASON [18], a fast discrete-event simulation core which also provides the pseudo-random numbers used in the simulation. The underlying knowledge base is implemented in SWI-Prolog [19] and accessed in Java via JPL [20]. The genetic algorithm-based optimization is implemented using the meta-heuristics framework Opt4J [21].

7.2. Knowledge modeling

For our sample application, we mapped 10 CAPEC attack patterns and added additional required actions, such as accessing data after the required permissions have been obtained. To account for attacks that exploit previously unknown vulnerabilities, we added a *zero-day* attack action. Successful execution of this action will give the adversary access to the attacked host. Existing controls such as antivirus software are largely ineffective against these kinds of attacks. The attack patterns used, including their CAPEC reference numbers and modeled security controls, are listed in Table 1.

Next, we enriched the model with security control definitions from the corresponding CAPEC patterns. Controls include, for example, antivirus software, patches for specific software, intrusion detection systems, log policies, and security training.

Finally, the organization and its IT infrastructure is represented by creating instances of concepts such as *Host*, *Subnet*, *Data*, and *User*, as well as relations between these concepts (e.g., *Host* stores *Data*, *Host* uses *Software*, *User* in *UserGroup*). The synthetic example system model used in this illustrative scenario was generated automatically. It contains 30 hosts, 5 web servers, 5 database servers, 30 employees, and 3 administrators (cf. Figure 3 for an overview; details such as connections between systems, and installed software have been omitted for sake of clarity of exposition).

Attack action	CAPEC (ID)
sql injection	SQL Injection (66)
social attack	Information Elicitation via Social Engineering (410)
brute force	Password Brute Forcing (49)
spearfish attack	Social Information Gathering via Pretexting - (407)
buffer overflow	Buffer overflow in an API call (8)
email keylogger	Email Injection (134)
email backdoor	Email Injection (134)
zero day	Privilege Escalation (233)
directory traversal	Directory Traversal (213)
shoulder surfing	Social Information Gathering (404)
access data	Legitimate Action
access host	Legitimate Action

Table1. Example attack patterns used

7.3. Experimental Setup

We define five external and internal adversaries, all of them with the objective of gaining access to data set *db2*. Internal adversaries already have certain levels of access to the system (e.g., *shoulder surfing* requires *physical access* to the environment). For external attackers, the demilitarized zone (DMZ) is the main entry point to the company's internal network. We assign varying preference weights, time budgets and other parameters that shape adversaries' simulated behavior as summarized in Table 2. An advanced persistent threat (APT), for example, has a large time budget, the highest technical skill level (and therefore a wide range of sophisticated attack actions), and is highly risk-averse (i.e., has a strong preference for avoiding detection).

An unskilled external attacker, on the other hand, is less patient and hence has a low time budget, a low technical skill level, and primarily aims at successfully executing attack actions without caring much about detection.

For other behavioral parameters, we set uniform values for all adversary types ($p_{continueNew} = 0.9$, $p_{retry} = 0.5$, $p_{alternativesNoNew} = 0.7$, and $p_{alternativesFailed} = 0.3$).

We used six optimization objectives in our experiments: *minimize costs*, *minimize successful attacks*, *maximize detection of attacks, minimize total confidentiality impact, minimize total integrity impact, and minimize total availability impact*.

To map the impact category valuations *low*, *medium*, and *high* to a scalar objective value, we used a lexicographic mapping, i.e., only the highest impact category is relevant for the optimization. We performed 30 simulation replications for each portfolio and evolved the system model for 500 generations using recommended NSGA-II standard parameters (cf. [16]).

7.4. Results

All optimization experiments were executed on a 2x3Ghz Xeon machine and required a runtime between

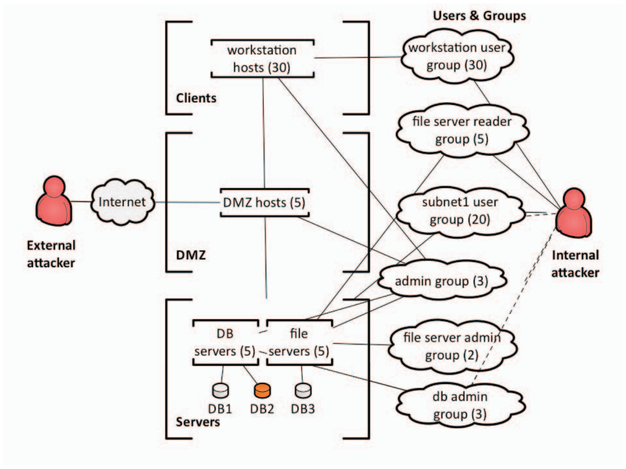

Figure 3. Example system model

approximately 90 minutes (Admin scenario) and 50 hours (APT scenario). However, the optimization typically converged well before 500 generations. The genetic algorithm identified 251 efficient portfolios for the advanced persistent threat scenario, 306 for the skilled external attacker and 104 for the unskilled external attacker. Compared to these external threat sources, the number of efficient solutions was much lower for internal attackers, namely just 2 for the administrator scenario and 58 for the employee scenario. Figure 4 illustrates the objective values of efficient portfolios in a parallel coordinate plot. The impact objective axes (CIA) are visually separated into high (above the red mark), medium (above the orange mark), and low (above the green mark) impact categories. Only the impact from the highest category is visualized for each portfolio.

Figure 5 illustrates the set of solutions obtained for the different attacker types in a heatmap representation. For the *admin* attacker, there are only two efficient portfolios not depicted in this representation. For all other types, the labels on the right hand side indicate the proposed efficient portfolios identified in the optimization. Each individual line represents a portfolio and contains its genotype string on the left hand side (blue) and the optimization objective values obtained for the portfolios on the right hand side (red). Control columns are arranged by control type and each individual field represents an assignment of a control

adversary	time limit (sec.)	w_{det}	w_{suc}	w_{dist}	access
Employee	150,000	0.45	0.25	0.30	workstation hosts
Admin	300,000	0.50	0.20	0.30	all hosts
Skilled	200,000	0.30	0.40	0.30	external
Unskilled	100,000	0.30	0.40	0.30	external
APT	1,000,000	0.50	0.20	0.30	external

Table 2. Attacker types

to a particular asset (i.e., blue if the control is included in the portfolio, and white otherwise). Result columns for impact metrics consist of high, medium, and low impact categories grouped by attributes.

This rather dense representation is not suitable for in-depth analysis, but it provides an overview of which control types tend to be more or less efficient for particular attacker types. It also facilitates a rough comparison of the relative impact these different adversaries may achieve.

For the *administrator*, we find that this adversary always reaches the specified target, which is expected given that he/she already has all required access privileges. There is a single highly critical confidentiality impact caused by the confidentiality breach on the target *db2*. For this type of attacker, there are no effective preventive controls, but applying a log policy on *dbServerHosts* raises the detection rate substantially.

The *employee* succeeds in obtaining access to the target data set in approximately half of the simulation runs when no security controls are in place. Even though the employee has access to internal systems, he/she cannot access the hosting servers directly and lacks the skills for technical attacks. He/she must therefore rely on social attacks. The relatively high success rate can be attributed to the availability of more effective social attacks (shoulder surfing) due to physical access. Accordingly, the control portfolios that include intensive security trainings (e.g., *train2* and *train3*) for the database (*dbAdminGroup*) and server administrators (*adminGroup*) turn out highly effective against this type of attacker. With these controls implemented, the share of simulation runs in which the employee reaches the target can be reduced to 6%. Technical controls, on the other hand, are not included in any efficient control portfolios for this adversary type. Again, a log policy on the hosting *dbServerHosts* raises the detection probability.

As expected, the *advanced persistent threat* has the highest success rate of all external attackers (approx. 2/3). Due to its proficiency and budget, a wide range of

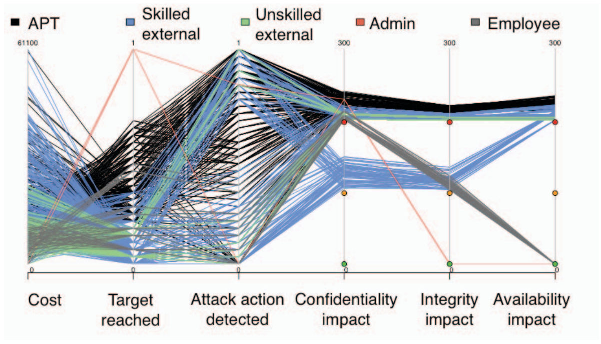

Figure 4. Efficient portfolio objective values

attacks and attack paths are available, and, hence, a wide range of security controls have to be implemented. Patching vulnerable systems, deploying antivirus software, code reviews, and performing security trainings are the most effective preventive controls against this adversary type. For the *dmzHosts* in particular, antivirus software *av2* is frequently included in efficient portfolios. Intrusion detection systems and log policies result in a high detection rate. In contrast to internal attack scenarios, controls are applied to a much wider range of assets (e.g., log policies on various servers and not only on the target database server). This can be attributed to the fact that this attacker type causes significant "collateral" impacts while trying to find ways to achieve the target (including high impacts on availability and integrity).

Even with high investments in security controls, however, this attacker can rarely be stopped from reaching the target. The most efficient sets of controls reduced the share of successful attacks to 20% and raised detection rates to 63%.

Optimization results for the *skilled external* attacker are similar, even though this type of adversary succeeds less frequently. In general, the security impact is lower compared to advanced persistent threats. The impacts on confidentiality and integrity are significantly reduced by the choice of efficient security controls. Furthermore, technical controls, such as antivirus and web server hardening, are prevalent in control portfolios against skilled external attackers.

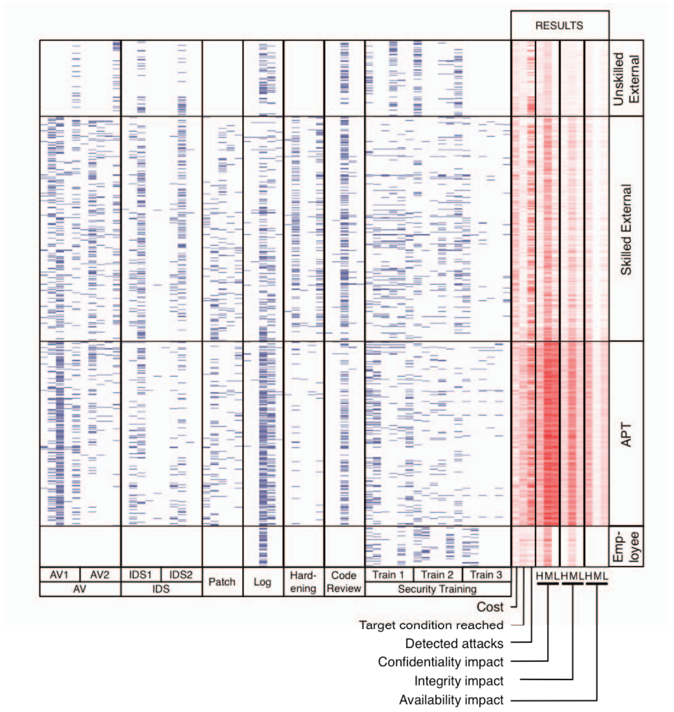

Figure 5. Control sets and simulation results

Against an unskilled external attacker, intrusion detection systems prove effective and show a high detection rate. To prevent email attacks (including backdoors and keyloggers), antivirus software and security training, especially for workstation users (*workstationUserGroup*) are included in a large fraction of portfolios. The impacts of this adversary type are less severe, and the share of successful attacks can be reduced to 3%.

In a real-world setting, both systems and threats are evolving. The method hence needs to be applied on a continuous basis. To this end, it is necessary to update the system model to reflect configuration changes, update the threat scenarios as the attack landscape is changing, and update the security KB as new attacks become known.

8. Conclusions

The evolutionary optimization approach towards security introduced in this paper aims at improving the resilience of complex information systems. It is built upon a knowledge base that models multi-step attacks for distinct adversary types and leverages this knowledge to identify promising combinations of information security controls through simulation-optimization.

We illustrate the approach with a sample application. Using an abstract graph to map the attacker's mental model, our framework simulates a large number of attacks for multiple adversaries and evolves system configurations that are resistant against these adversaries. We simultaneously consider multiple, partly conflicting, objectives, i.e., maximizing prevention and detection of (simulated) attacks while minimizing their confidentiality, integrity, and availability impacts at minimal cost. The results of our experiments suggest that the proposed approach can be usefully applied to study and optimize the security of complex information systems.

Further research is possible in several directions. First, to cover CAPEC more extensively, the security knowledge base can be extended with additional attack patterns. This is not a trivial task that requires the development of an appropriate vocabulary for the specification of cause and effect across multiple levels of abstraction. Moreover, additional sources of attack knowledge beyond software security can be integrated. The resulting knowledge base needs to be provided to the information security community and other interested stakeholders in a shared repository. This would enable organizations to independently model their own systems as well as the threats they face and use the domain knowledge formalized in the shared knowledge base to obtain recommendations on effective improvements. Because information security is a “moving target”, knowledge needs to be maintained and extended continuously. We therefore intend to design mechanisms that allow users to automatically update the knowledge base as new attacks become known, which in turn would allow them to determine their exposure to these new attacks through simulation and obtain recommendations on efficient mitigations. This could reduce the knowledge gap between attackers and defenders.

Because the list of proposed efficient sets of measures is typically large, we also develop interactive visualizations for the exploration of the solution space and mechanisms that support decision-makers in the selection of a set of security controls to implement.

Finally, it would be interesting to evaluate our approach in the context of other domains with strict security requirements, such as critical infrastructures.

9. References

[1] “CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC)” Available: http://capec.mitre.org/ [Accessed June 14, 2013].

[2] S. Barnum and A. Sethi, “Attack Patterns as a Knowledge Resource for Building Secure Software,” in: *OMG Software Assurance Workshop: Cigital*, 2007.

[3] S. Barnum and G. McGraw, “Knowledge for Software Security,” *IEEE Security Privacy*, vol. 3, no. 2, 2005, pp. 74–78.

[4] NIST Computer Security Division, *Special Publication 800-39: Managing Information Security Risk - Organization, Mission, and Information System View*, 2010.

[5] C. Muehrcke, E. Ruitenbeek, K. Keefe, and W. Sanders, “Characterizing the Behavior of Cyber Adversaries: The Means, Motive, and Opportunity of Cyberattacks,” 2010.

[6] B. Schneier, “Attack trees,” *Dr. Dobb’s Journal*, vol. 24, no. 12, 1999, pp. 21–29.

[7] P. Ammann, D. Wijesekera, and S. Kaushik, “Scalable, graph-based network vulnerability analysis,” in: *Proceedings of the 9th ACM conference on Computer and communications security*, Washington, DC, USA, 2002, pp. 217–224.

[8] R.E. Sawilla and X. Ou, “Identifying critical attack assets in dependency attack graphs,” in: *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, Springer, 2008, pp. 18–34.

[9] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing, “Automated generation and analysis of attack graphs,” in: *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002, pp. 273– 284.

[10] S. Bistarelli, F. Fioravanti, and P. Peretti, “Defense trees for economic evaluation of security investments,” in: *Proceedings of the First International Conference on Availability, Reliability and Security (ARES 2006)*, 2006, pp. 416–423.

[11] K.S. Edge, “A Framework for Analyzing and Mitigating The Vulnerabilities of Complex Systems Via Attack And Protection Trees,” Air Force Institute of Technology, Faculty Graduate School of Engineering and Management, 2007.

[12] G.C. Dalton, R.F. Mills, J.M. Colombi, and R.A. Raines, “Analyzing attack trees using generalized stochastic petri nets,” in: *Proceedings of the 2006 IEEE Workshop on Information Assurance*, 2006, pp. 116–123.

[13] L. Pietre-Cambacédès and M. Bouissou, “Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP),” in: *Proceedings of the 2010 European Dependable Computing Conference*, Apr. 2010, pp. 199–208.

[14] S. Noel, S. Jajodia, L. Wang, and A. Singhal, “Measuring security risk of networks using attack graphs,” *International Journal of Next-Generation Computing*, vol. 1, no. 1, 2010, pp. 135–147.

[15] E.W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, 1959, pp. 269–271.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast elitist multi-objective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, 2000, pp. 182–197.

[17] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2: Improving the Strength Pareto Evolutionary Algorithm,” in: *Evolutionary Methods for Design, Optimisation and Control*, K. Giannakoglou, D. Tsahalis, K. Papailiou, and T. Fogarty, Eds., International Center for Numerical Methods in Engineering, 2002.

[18] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan, “MASON: A new multi-agent simulation toolkit,” in: *2004 SwarmFest Workshop*, 2004.

[19] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, “SWI-Prolog,” *Theory and Practice of Logic Programming*, vol. 12, no. Special Issue 1-2, 2012, pp. 67–96.

[20] “JPL: A bidirectional Prolog/Java interface” Available: http://www.swi-prolog.org/packages/jpl/ [Accessed June 14, 2013].

[21] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, “Opt4J: a modular framework for meta-heuristic optimization,” in: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 1723–1730.